Table of Contents

3rd Year Project eDiary: *Project 72 - Classifying Cosmological Data with Machine Learning*

Week Starting January 29th:

Thursday Febuary 1st 10am-11am: SUPERVISOR MEETING

- Met with Adam Moss and discussed project, we will be classifying galaxies using deep learning similar to the galaxy zoo project. https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/
- Discussed previous knowledge of machine learning and got introduced to the fundamentals of machine learning.
- Discussed problems faced in classifcation problems with convoloutional neural networks and discussed techniques to fix these problems, i.e. domain adaptation.
- Created timeline for project and drafted a project plan
- Read up to section 1.3 of the Machine Learning Notes:

- 1.1 introduction to deep learning:
  - Notation
  - Loss functions
  - Neurons
  - Activation functions
- 1.2 Basic Types of Networks:
  - Linear model
  - Perceptron
  - Logistic Regression
  - Multi-Layer perceptron
- 1.3 Back propagation:

11am-1pm PROJECT PARTNER MEETING:

- Drafted project plan
- Divided tasks
- Look at galaxy zoom and familarised ourselves with the classification classes.

Malachy & Aroushi -

PROJECT PLAN:

PROJECT 72

Classifying Cosmological Data with Machine Learning

The aim of this project is to build a deep learning algorithm which classifies galaxies into specified sub-classes by identifying coarse and fine features. Deep learning eliminates the need for manual feature extraction, and instead relies on convolutional neural networks to classify images based on simpler representations learned from a training dataset. We will use domain adaptation techniques to improve results obtained from unseen data, thereby achieving our objective of galaxy classification.

Week	Task Description	Task Division
1 (05/02)	1. Introduction to Neural Networks 2. Learn about the basic building blocks such as neurons, loss functions, and activation functions 3. Gain understanding of optimising cost parameters during training 4. Learn how to use GitHub	All tasks - Aroushi and Malachy
2 (12/02)	1. Set up Google CoLab (connect to T4 GPU) 2. Gain proficiency in TensorFlow/ PyTorch 3. Learn how to split data into training and test sets 4. Study strategies to prevent overfitting	All tasks - Aroushi and Malachy
3 (19/02)	1. Focus on convolutional neural networks (CNNs) 2. Transfer learning 3. Fine-tuning methods 4. Techniques to generalise to unseen data	All tasks - Aroushi and Malachy
4 (26/02)	1. Galaxy Zoo Datasets: Get acquainted with the Galaxy Zoo datasets 2. Understand the data structure and how it has been previously packaged in Python	All tasks - Aroushi and Malachy
5 (04/03)	1. Convert classification variables from the galaxy zoo to one hot encoding (majority vote = 1, else = 0). 2. Use raw classification variables from the galaxy zoo data set (consider all voted classifications and their values).	1 – Aroushi 2 – Malachy
6 (11/03)	1. Initial Model Training: Train a neural network on the Sloan Digital Sky Survey data to identify coarse features (i.e. galaxy shape – spiral/elliptical, galaxy rotation, etc.). 2. Test the model on the Dark Energy Survey data to evaluate performance with higher resolution images, show that model classifies well the test data but fails with unseen data.	1 – Aroushi and Malachy 2 – Aroushi and Malachy
7 (18/03)	1. Fine detail feature classification – (i.e. number of spiral arms, gravitational lensing, etc.). 2. Domain Adaptation Techniques: Begin exploring domain adaptation techniques, with an emphasis on adversarial domain adaptation.	1 – Malachy 2 – Aroushi
8 (25/03)	1. Re-test neural network on the images from the galaxy zoo's data set (the Hawaii Two-0 (H20) survey) and compare our prediction classification and the voted classification from the websites users to evaluate accuracy.	1 – Malachy and Aroushi

1. Training Datasets from Galaxy Zoo (https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/)
2. Sloan Digital Sky Survey (https://www.sdss.org/)
3. Deep Learning Resources (Machine Learning in Science, Part 2, Adam Moss)

I can confirm I am happy with this plan.

Dr Adam Moss

Aroushi -

1 Introduction to deep learning:

Deep learning algorithms have the advantage of automatically discovering the representations needed for feature detection or classification from raw data, eliminating the need for manual feature extraction. This ability to learn complex, hierarchical features at multiple levels of abstraction allows deep learning models to handle a wide range of tasks with higher accuracy, especially as the volume and variety of data increase.

1.1 Neural Network Basics:

1.1.1 Notation

We were introduced to the mathematical symbols for data representation, including input X and target Y, and how they form a dataset. This dataset can be split into training, test and validation datasets.

1.1.2 Loss functions

Describes the function $J(\theta)$ to evalaute predicition errors, using per loss dependent on the task, like MSE or BCE. It is defined as

$J(\theta) = \frac{1}{N}\sum_i^N l(\hat{y}^{(i)}(\theta), y^{(i)})$

where $\hat{y}$ and y are the prediciton and the target respectively.

Loss	Applications	$\ell(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$
Mean Square Error (MSE)	Regression	$\left(\hat{y}^{(i)} - y^{(i)}\right)^2$
Mean Absolute Error (MAE)	Regression	$\lvert\hat{y}^{(i)} - y^{(i)}\rvert$
Binary Cross Entropy (BCE)	Binary classification	$-\left(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right)$
Categorical Cross Entropy (CCE)	Multi-class classification	$-\sum_{m=0}^{M-1} y_m^{(i)} \log \hat{y}_m^{(i)}$

Common loss functions and their typical applications

For the purpose of this project, Galaxy Zoo target data contains probabilities of galaxy classifications rather than a single correct answer, due to the ambiguitity of some galaxy images. We will be trying two different approaches - one wherein we use the probability matrix as the target output, and one where we create a 'one-hot' target vector for the algorithm to aim towards.

- **1.1.3 Neurons**

Explains artificial neurons that output a weighted sum of inputs passed through an activation function. This is represented by the following equation,

$z = \mathbf{x}^\mathbf{T}\mathbf{w} + b$ and $a = \Phi(z)$

where Φ is an activation function.

A multi-layer perceptron consists of many such neurons stacked into layers, with the output of one layer serving as the input for the next

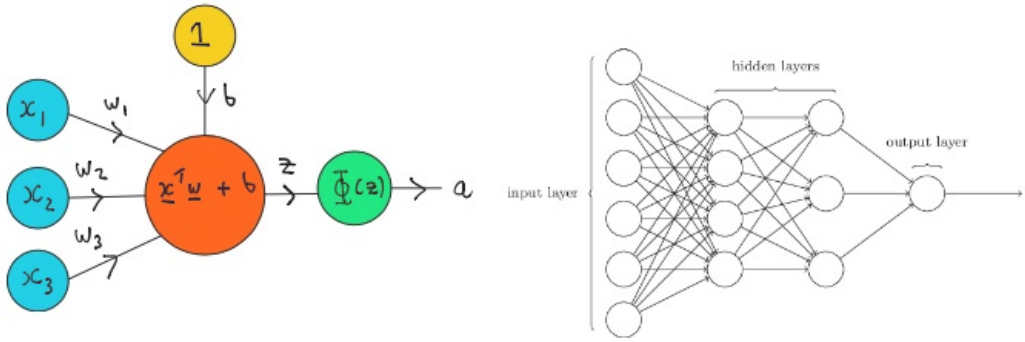
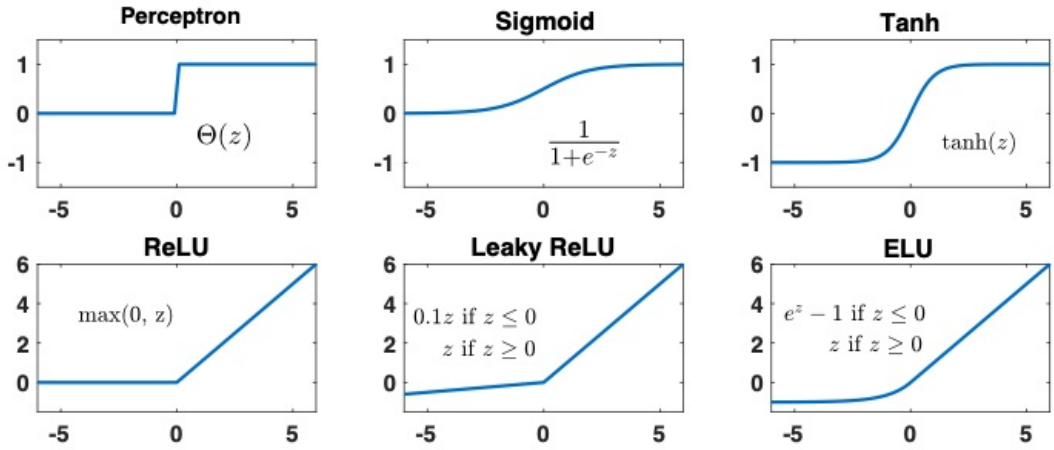


Figure 1.1.1: **Basic architecture of a multi-layer perceptron.** (Left) The basic component is a neuron, consisting of a linear transformation that weights the importance of various inputs, followed by an activation function. (Right) Neurons are arranged into layers with the output of one layer serving as the input to the next layer.

- **1.1.4 Activation functions**

Discusses non-linear functions like sigmoid or ReLU essential for learning complex patterns.



- ## 1.2 Basic Types of Networks:
  - ### 1.2.1 Linear model

    Covers the simplest neural model using a linear activation function, showing its limitation with non-linear data like XOR
  - ### 1.2.2 Perceptron

    Introduces the binary classifier perceptron with a step function and weight update rules based on the perceptron learning algorithm.
  - ### 1.2.3 Logistic Regression

    Describes logistic regression using the sigmoid function for binary classification and its training via gradient descent.
  - ### 1.2.4 Multi-Layer perceptron

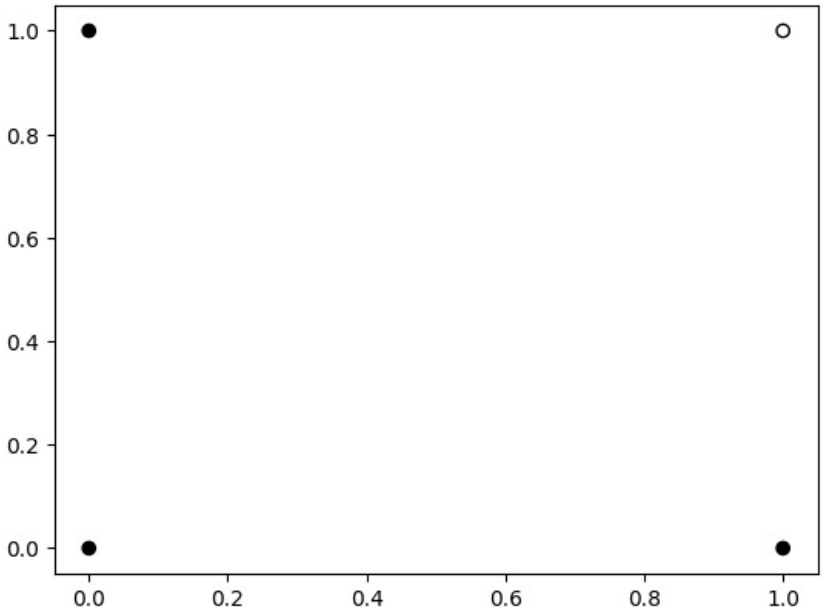
    Describes MLPs capable of learning non-linear functions, detailing their structure and function, including the ReLU activation and training considerations.
- ## 1.3 Back propagation:
  - Core algorithm for training neural networks, especially MLPs.
  - Involves two phases: forward pass (computing activations) and backward pass (propagating errors backward).
  - Errors calculated at the output layer are propagated back to update weights.
  - Mathematically, this involves the chain rule to compute gradients for weight updates.
  - Process is repeated iteratively for a number of epochs or until convergence.

Malachy -

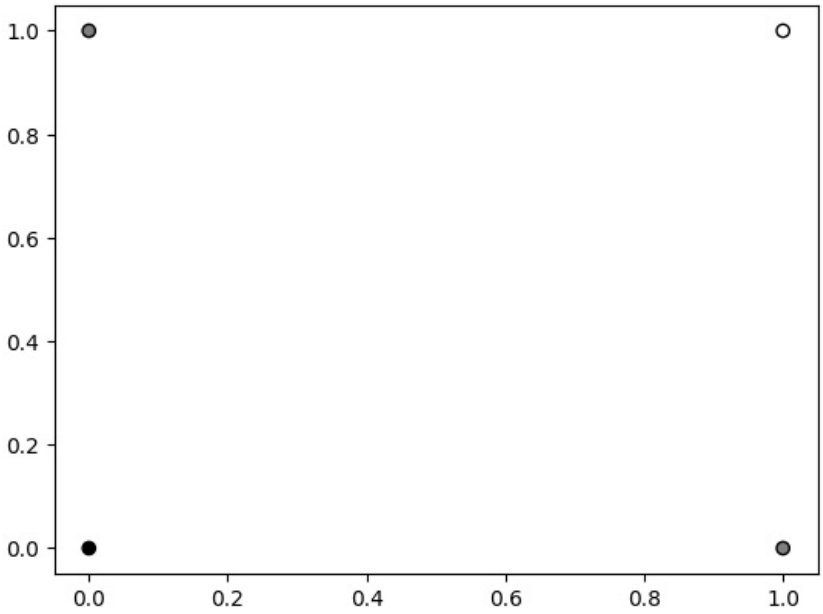
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

run = False

if run:
    #AND function:
    X = np.array([[1,0,0], [1,0,1], [1,1,0], [1,1,1]]) #x1, x2
    print(X)

    inv = np.linalg.inv(np.matmul(X.T, X))
    Y = np.array([[0], [0], [0], [1]]) #AND gate output
    print(Y)

    plt.scatter(X[:,1], X[:,2],c=np.reshape(Y,4), edgecolors='black', cmap='gray')
    plt.show() #AND gate outputs

    w = np.matmul(inv,  np.matmul(X.T, Y)) #weights, found by choosing to model as a linear regression problem
    print(w)

    Yhat = np.matmul(X, w)
    print(Yhat)

    plt.scatter(X[:,1], X[:,2],c=np.reshape(Yhat,4), edgecolors='black', cmap='gray') #plot the predictions
    plt.show() #AND gate predictions

    #if we have new points, we cab use the trained model to predict the Y values

    n = 100
    Xtest = np.transpose(np.reshape(np.append(np.ones(n),np.random.uniform(0,1,n*2)),(3,n)))
```

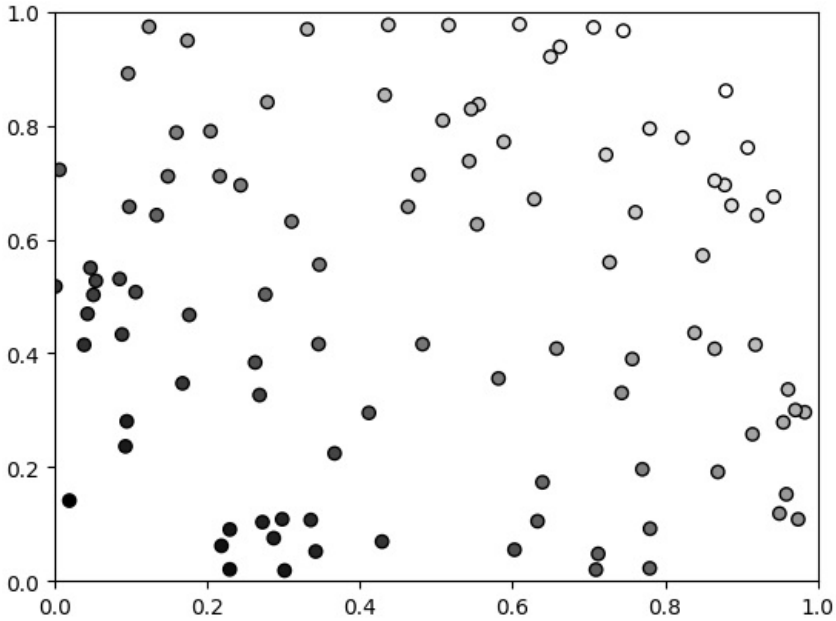
```
    Yhattest = np.matmul(Xtest, w)

    plt.scatter(Xtest[:,1], Xtest[:,2],c=np.reshape(Yhattest,n), edgecolors='black', cmap='gray')
    plt.xlim([0,1])
    plt.ylim([0,1])
    plt.show()

    #XOR function:
    Y = np.array([[0], [1], [1], [0]])
    print(Y)

    w = np.matmul(inv,  np.matmul(X.T, Y))
    print(w)

    Yhat = np.matmul(X, w)
    print(Yhat)

    plt.scatter(X[:,1], X[:,2],c=np.reshape(Yhat,4), edgecolors='black', cmap='gray')
    plt.show() #XOR gate outputs

    #The model is not able to predict the XOR gate outputs, outputting 1/2 for all examples.

    n=100
    Xtest = np.transpose(np.reshape(np.append(np.ones(n),np.random.uniform(0,1,n*2)),(3,n)))
    Yhattest = np.matmul(Xtest, w)

    plt.scatter(Xtest[:,1], Xtest[:,2],c=np.reshape(Yhattest,n), edgecolors='black', cmap='gray')
    plt.xlim([0,1])
    plt.ylim([0,1])
    plt.show()

    #This is true for the test samples as well.
```

```
[[1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]
[[0]
 [0]
 [0]
 [1]]
```

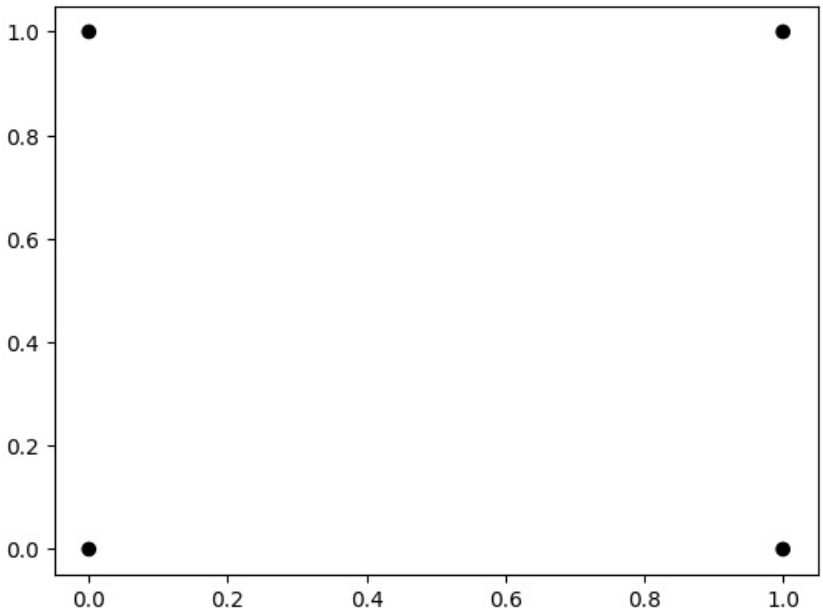


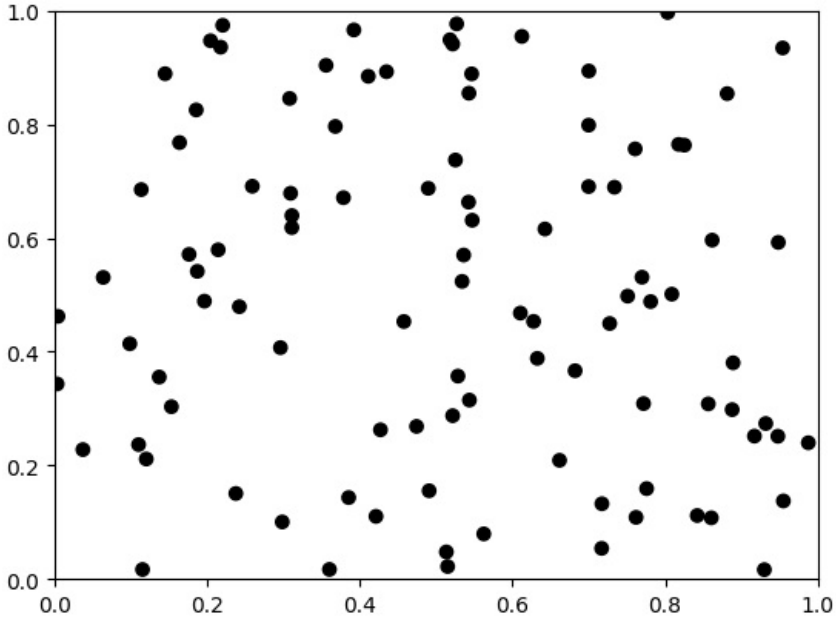

```
[[-0.25]
 [ 0.5 ]
 [ 0.5 ]]
[[-0.25]
 [ 0.25]
 [ 0.25]
 [ 0.75]]
```






```
[[0]
 [1]
 [1]
 [0]]
[[0.5]
 [0. ]
 [0. ]]
[[0.5]
 [0.5]
 [0.5]
 [0.5]]
```





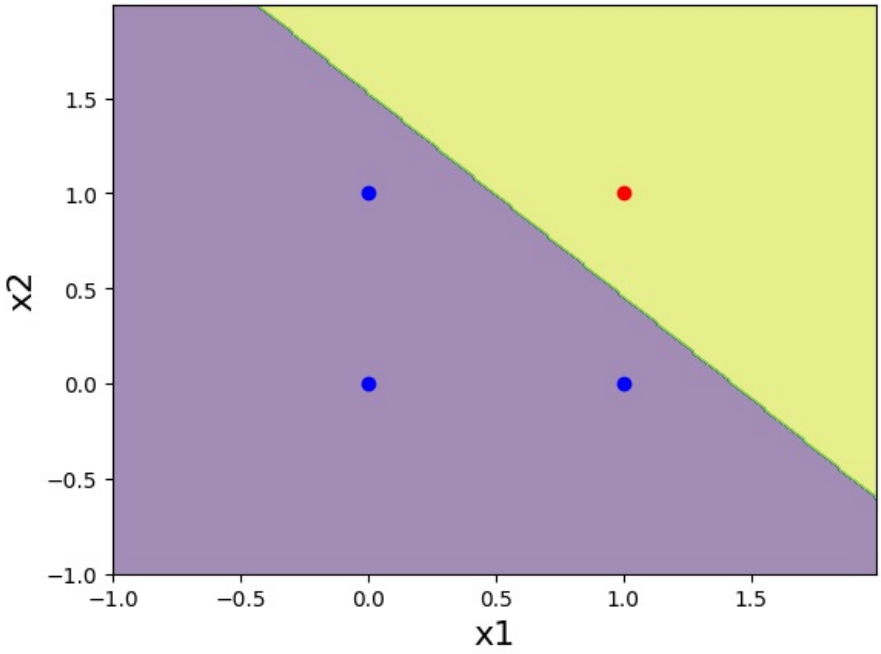
Malachy -

In [ ]:

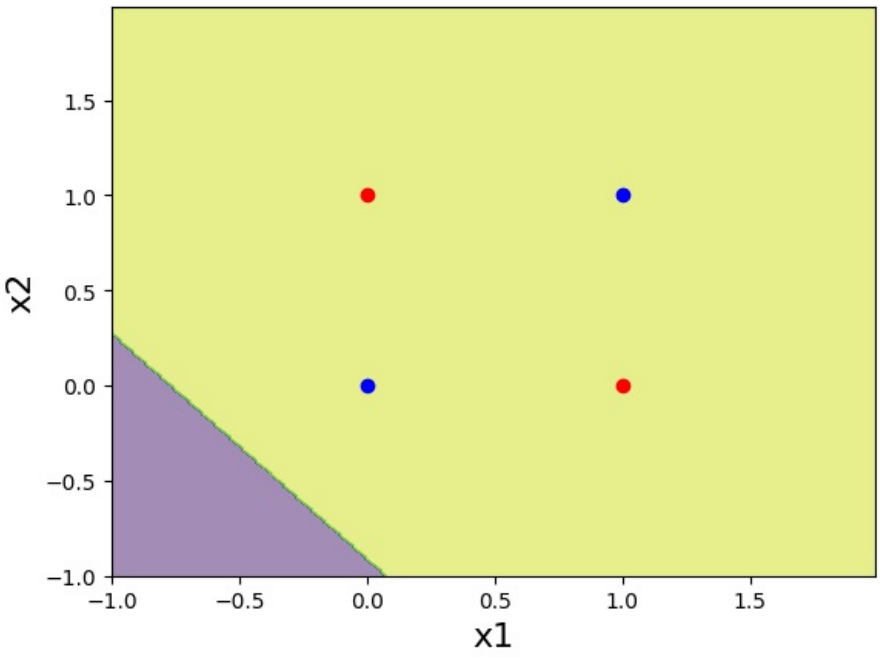
```python
import numpy as np
import matplotlib.pyplot as plt

run = False

#perceptron model is given by:
#yhat = θ (x^T w)
#where θ is the Heaviside step function. This is an example of a binary classifier.
#due to the non-linearty of the Heaviside step function we cannot solve for the weights using the normal equati
#we can use the perceptron learning algorithm to iteratively find the weights depending on the size of the pred
#1/ Initialise the weights to 0 or small random values
#2/ Iteravitely update the weights by w -> w + αX^T (Yhat - Y), where α is the learning rate.
#3/ Repeat step 2 until a stopping codition is met, this could be a maximum number of iterations or that the
# loss has not improved significantly from the previous iteration.

if run:
    #AND function:

    X = np.array([[1,0,0], [1,0,1], [1,1,0], [1,1,1]]) #x1, x2
```

```
    print(X)

    Y = np.array([[0], [0], [0], [1]]) #AND gate output
    print(Y)

    np.random.seed(0)
    w = 0.1 * np.random.random(size=(3, 1)) #initialise the weights to small random values
    print(w)

    num_epochs = 10
    learning_rate = 0.1 #α

    for i in range(num_epochs):
        Yhat = np.heaviside(np.matmul(X, w), 0)
        w -= learning_rate * np.matmul(X.T, Yhat - Y)
    print(w)

    #check the classification of the training samples
    Yhat = np.heaviside(np.matmul(X, w), 0)
    print(Yhat)

    #plot the descision boundary for all values of x
    id0 = np.where(Y[:, 0] == 0)
    id1 = np.where(Y[:, 0] == 1)

    xx, yy = np.mgrid[-1:2:.01, -1:2:.01]
    Yhat = np.heaviside(w[0] + w[1] * xx + w[2] * yy, 0)

    plt.figure()
    plt.contourf(xx, yy, Yhat, alpha=0.5)
    plt.scatter(X[id0, 1], X[id0, 2], color='blue')
    plt.scatter(X[id1, 1], X[id1, 2], color='red')
    plt.xlabel('x1', fontsize=16)
    plt.ylabel('x2', fontsize=16)
    plt.show()

    #XOR function:
    Y = np.array([[0], [1], [1], [0]]) #XOR gate output
    print(Y)

    np.random.seed(0)
    w = 0.1 * np.random.random(size=(3, 1)) #initialise the weights to small random values
    print(w)

    for i in range(num_epochs):
        Yhat = np.heaviside(np.matmul(X, w), 0)
        w -= learning_rate * np.matmul(X.T, Yhat - Y)
    print(w)

    #check the classification of the training samples
    Yhat = np.heaviside(np.matmul(X, w), 0)
    print(Yhat)

    #plot the descision boundary for all values of x
    id0 = np.where(Y[:, 0] == 0)
    id1 = np.where(Y[:, 0] == 1)

    xx, yy = np.mgrid[-1:2:.01, -1:2:.01]
    Yhat = np.heaviside(w[0] + w[1] * xx + w[2] * yy, 0)

    plt.figure()
    plt.contourf(xx, yy, Yhat, alpha=0.5)
    plt.scatter(X[id0, 1], X[id0, 2], color='blue')
    plt.scatter(X[id1, 1], X[id1, 2], color='red')
    plt.xlabel('x1', fontsize=16)
    plt.ylabel('x2', fontsize=16)
    plt.show()

#The perceptron manages to classify the AND gate outputs correctly, but not the XOR gate outputs, giving all exa
#or class. This is because the perceptron is a linear classifier and will never be able to classify all examples
#data is not linearly separable, that is all examples of one class cannot be seperated from the other by a hype
#This is the case for the XOR gate outputs.

#The perceptron algorith is guaranteed to converge on some solution if the training data is linearly separable,
#solution, depending on the inital weights and the training procedure. In the case of the AND example, there ar
#seperating the two classes.
```

```
[[1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]
[[0]
 [0]
 [0]
 [1]]
[[0.05488135]
 [0.07151894]
 [0.06027634]]
[[-0.24511865]
 [ 0.17151894]
 [ 0.16027634]]
[[0.]
 [0.]
 [0.]
 [1.]]
```




```
[[0]
 [1]
 [1]
 [0]]
[[0.05488135]
 [0.07151894]
 [0.06027634]]
[[0.05488135]
 [0.07151894]
 [0.06027634]]
[[1.]
 [1.]
 [1.]
 [1.]]
```



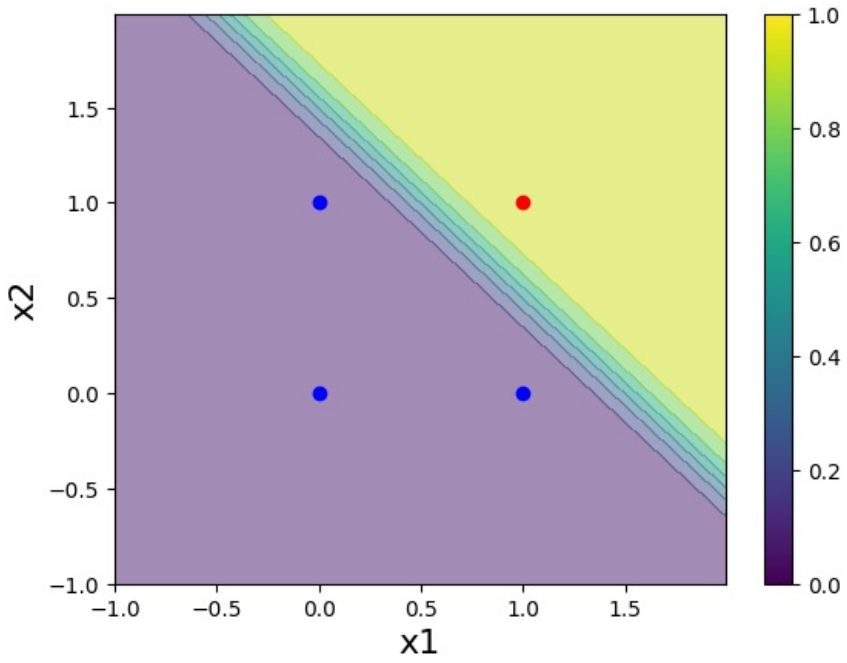
Malachy -

In [ ]:

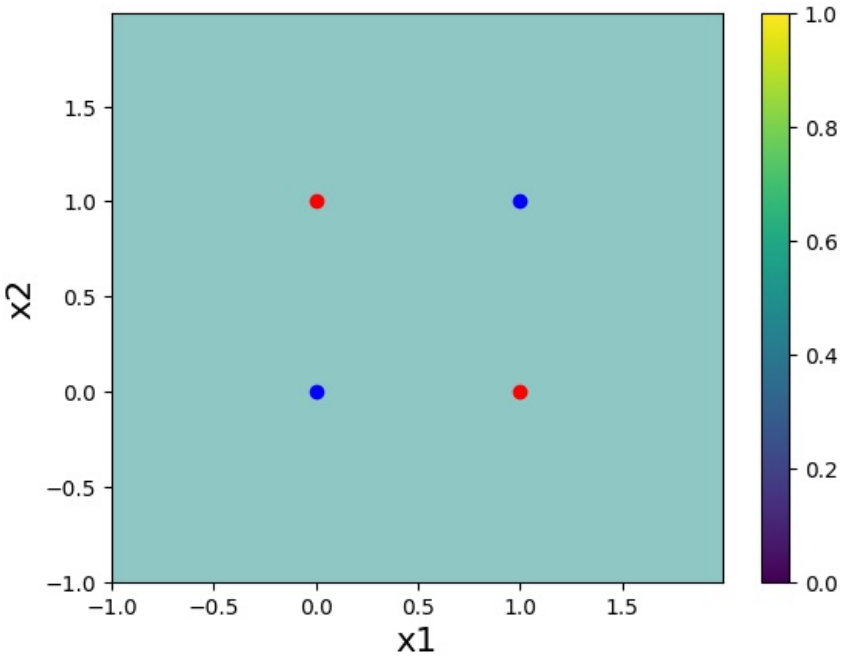
```
#The logistic model is given by:
#yhat = σ (x^T w)
#where σ is the sigmoid function. This is an example of a binary classifier.
#logistic regression is used to estimate the probability that an example belongs to a specific binary class.
#yhat = P(Y=1|x)
#There is no closed form solution for the weights, so we use a iterative procedure to find the weights.
#since the activation function is differentiable, we can use gradient descent, and if the loss function is
#convex it is guaranteed to converge to the global minimum.
#gradient descent proposes a new set of weights:
#w -> w - α∇L(w)
#where α is the learning rate, a positive scalar that determines the size of the update. There are several
#ways to choose α. For now we will set it to a small constant value.
#we will attempt to classify the AND and XOR functions but this time use the BCE loss function.
#In the logistic model, the gradient of the BCE loss with respect to the weights can be shown to be:
#∇L(w) = 1/N * X^T (Yhat - Y)
#gradient descent for logistic regression strongly resembles that of the perceptron.

import numpy as np
import matplotlib.pyplot as plt

run = False

if run:
    def sigmoid(x):
        return 1/(1 + np.exp(-x))

    X = np.array([[1,0,0], [1,0,1], [1,1,0], [1,1,1]]) #x1, x2
    print(X)

    Y = np.array([[0], [0], [0], [1]]) #AND gate output
    print(Y)

    np.random.seed(0)
    w = 0.1 * np.random.random(size=(3, 1)) #initialise the weights to small random values
    print(w)

    num_epochs = 10000
    learning_rate = 0.1

    for i in range(num_epochs): #gradient descent
        Yhat = sigmoid(np.matmul(X, w))
        w -= learning_rate * np.matmul(X.T, Yhat - Y)
    print(w)

    Yhat = sigmoid(np.matmul(X, w))
    print(Yhat)

    id0 = np.where(Y[:, 0] == 0)
    id1 = np.where(Y[:, 0] == 1)

    xx, yy = np.mgrid[-1:2:.01, -1:2:.01]
    Yhat = sigmoid(w[0] + w[1] * xx + w[2] * yy)

    plt.figure()
```

```
    plt.contourf(xx, yy, Yhat, alpha=0.5)
    plt.scatter(X[id0, 1], X[id0, 2], color='blue')
    plt.scatter(X[id1, 1], X[id1, 2], color='red')
    plt.xlabel('x1', fontsize=16)
    plt.ylabel('x2', fontsize=16)
    plt.colorbar()
    plt.show()

    #XOR function:
    Y = np.array([[0], [1], [1], [0]]) #XOR gate output
    print(Y)

    np.random.seed(0)
    w = 0.1 * np.random.random(size=(3, 1)) #initialise the weights to small random values
    print(w)

    for i in range(num_epochs): #gradient descent
        Yhat = sigmoid(np.matmul(X, w))
        w -= learning_rate * np.matmul(X.T, Yhat - Y)
    print(w)

    Yhat = sigmoid(np.matmul(X, w))
    print(Yhat)

    id0 = np.where(Y[:, 0] == 0)
    id1 = np.where(Y[:, 0] == 1)
    xx, yy = np.mgrid[-1:2:.01, -1:2:.01]
    Yhat = sigmoid(w[0] + w[1] * xx + w[2] * yy)

    plt.figure()
    plt.contourf(xx, yy, Yhat, alpha=0.5)
    plt.scatter(X[id0, 1], X[id0, 2], color='blue')
    plt.scatter(X[id1, 1], X[id1, 2], color='red')
    plt.xlabel('x1', fontsize=16)
    plt.ylabel('x2', fontsize=16)
    plt.colorbar()
    plt.show()

#logistic regression correctly predicts the AND function, but fails to predict the XOR function, giving all exa
#Similar to the peceptron, it is not able to classify all examples correctly if the training data is not linear
```

```
[[1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]
[[0]
 [0]
 [0]
 [1]]
[[0.05488135]
 [0.07151894]
 [0.06027634]]
[[-15.5074077 ]
 [ 10.22541225]
 [ 10.22541225]]
[[1.84169761e-07]
 [5.05658027e-03]
 [5.05658027e-03]
 [9.92920285e-01]]
```




```
[[0]
 [1]
 [1]
 [0]]
[[0.05488135]
 [0.07151894]
 [0.06027634]]
[[-3.28399609e-16]
 [ 2.41896476e-16]
 [ 2.47572506e-16]]
[[0.5]
 [0.5]
 [0.5]
 [0.5]]
```



https://www.youtube.com/watch?v=2hj9MdWExDQ good visualisation of perceptron trying to classify XOR

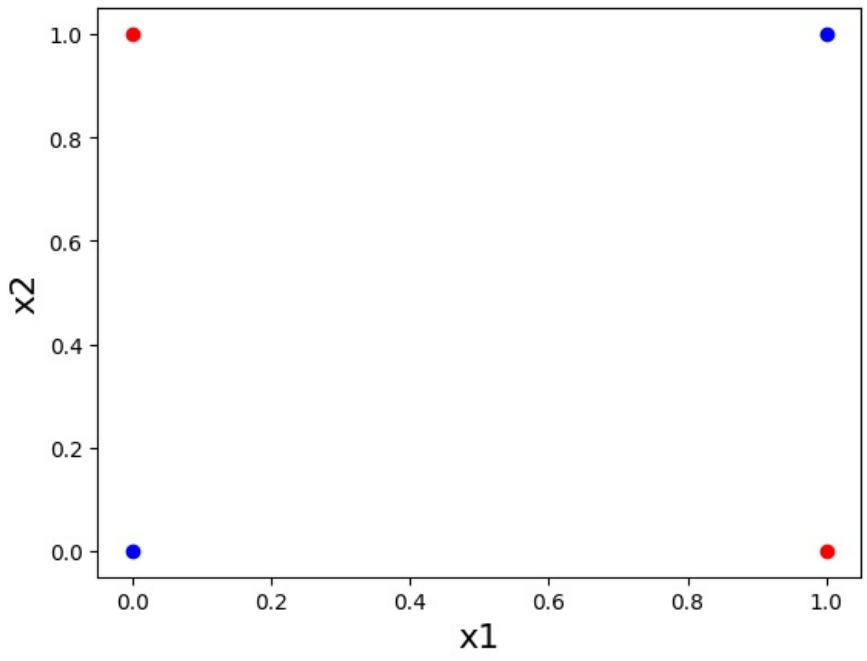
Malachy -

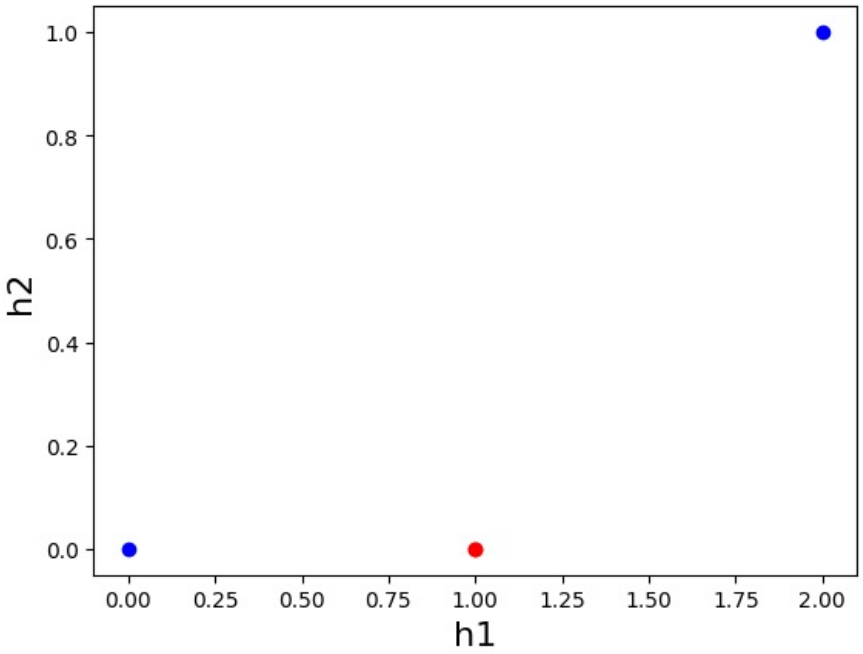
In [ ]:

```
#multi-layer perceptron model used to reproduce the XOR function is given by:
#Yhat = (ReLU(XW))w,
#with one hidden layer containing two hidden units.
#the optimal weights are W =[[0, -1], [1, 1], [1, 1]] and w = [[0], [1], [2]].

import numpy as np
import matplotlib.pyplot as plt

run = False

if run:
    X = np.array([[0,0], [0,1], [1,0], [1,1]])
    X = np.hstack((np.ones(shape=(X.shape[0], 1)), X))
    print(X)

    Y = np.array([[0], [1], [1], [0]])
    print(Y)

    W = np.array([[0, -1], [1,1], [1,1]], dtype=float)
    print(W)

    w = np.array([[0], [1], [-2]], dtype=float)
    print(w)

    h = np.maximum(np.matmul(X, W), 0)
    h = np.hstack((np.ones(shape=(h.shape[0], 1)), h))
    print(h)

    Yhat = np.matmul(h, w)
    print(Yhat)

    #It is instructive to see how the MLP is able to learn the XOR function
    #by plotting the rep- resentation of data it learns in the hidden space
```

```
    id0 = np.where(Y[:, 0] == 0)
    id1 = np.where(Y[:, 0] == 1)

    plt.figure()
    plt.scatter(X[id0, 1], X[id0, 2], color='blue')
    plt.scatter(X[id1, 1], X[id1, 2], color='red')
    plt.xlabel('x1', fontsize=16)
    plt.ylabel('x2', fontsize=16)
    plt.show()

    plt.figure()
    plt.scatter(h[id0, 1], h[id0, 2], color='blue')
    plt.scatter(h[id1, 1], h[id1, 2], color='red')
    plt.xlabel('h1', fontsize=16)
    plt.ylabel('h2', fontsize=16)
    plt.show()

#The original data is not linearly separable, but in the representation space
#it has mapped the two points with output 1 into a single point, and this space is now separable
```

```
[[1. 0. 0.]
 [1. 0. 1.]
 [1. 1. 0.]
 [1. 1. 1.]]
[[0]
 [1]
 [1]
 [0]]
[[ 0. -1.]
 [ 1.  1.]
 [ 1.  1.]]
[[ 0.]
 [ 1.]
 [-2.]]
[[1. 0. 0.]
 [1. 1. 0.]
 [1. 1. 0.]
 [1. 2. 1.]]
[[0.]
 [1.]
 [1.]
 [0.]]
```





Malachy -

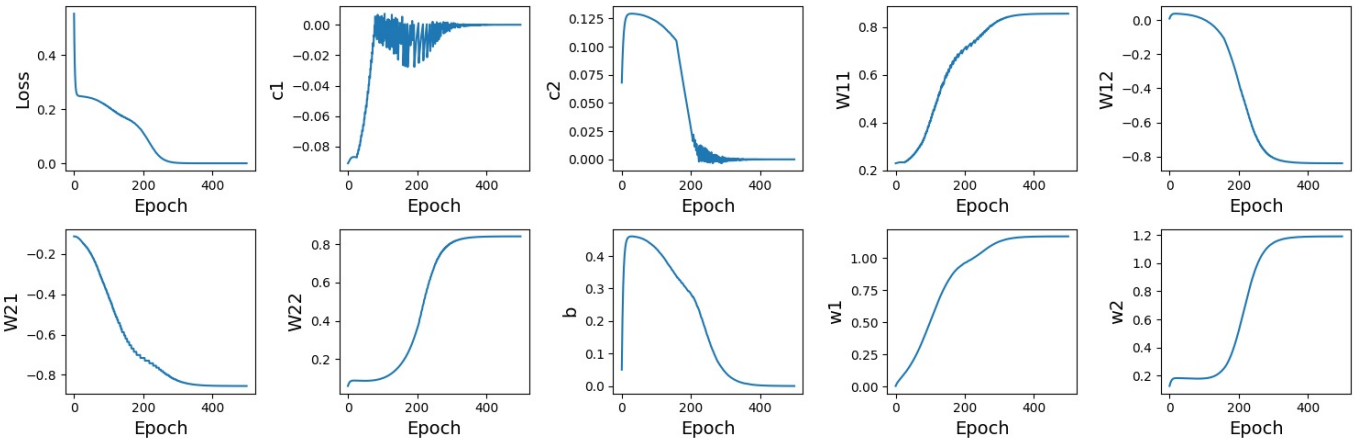
In [ ]:

```python
import numpy as np
import matplotlib.pyplot as plt

#attempt to model the XOR function using a multi-layer perceptron (MLP)
#using backpropagation with simple gradient descent

#the heaviside step function is the derivative of the ReLU function.

run = False

if run:
  np.random.seed(2)

  X = np.array([[0,0], [0,1], [1,0], [1,1]])
  print(X)

  Y = np.array([[0], [1], [1], [0]])
  print(Y)

  #define the MLP model
  class MLP(object):
    def __init__(self):
      """
      Simple MLP with 2 input nodes, 2 hidden nodes and 1 output node
      """
      # Initialise with random weights
      self.weights_1 = 0.1 * np.random.normal(size=(3,2))
      self.weights_2 = 0.1 * np.random.normal(size=(3,1))

    def forward(self, x):
      """
      Forward pass through the network
      """
      if len(x.shape) == 1:
        # Single example, so add a batch dimension of 1
        x = np.expand_dims(x, axis=0)
      # Hidden layer
      z_1 = np.matmul(np.hstack((np.ones(shape=(x.shape[0], 1)), x)), self.weights_1)
```

```
    # Apply ReLU activation function
    a_1 = np.maximum(z_1, 0)
    # Output layer
    z_2 = np.matmul(np.hstack((np.ones(shape=(a_1.shape[0], 1)), a_1)), self.weights_2)
    # Linear activation
    a_2 = z_2
    return z_1, a_1, z_2, a_2

#push the true solution through the network and check it gives zero loss

m = MLP()
m.weights_1 = np.array([[0, -1], [1,1], [1,1]], dtype=float)
m.weights_2 = np.array([[0], [1], [-2]], dtype=float)
z_1, a_1, z_2, a_2 = m.forward(X)
print(0.25 * np.sum((a_2 - Y)**2))

num_epochs = 500
learning_rate = 0.1

#update the weights and biases using backpropagation

m = MLP()
loss_history = []
weights_1_history = []
weights_2_history = []

for epoch in range(num_epochs):

  # Do forward pass
  z_1, a_1, z_2, a_2 = m.forward(X)
  loss = 0.25 * np.sum((a_2 - Y)**2)
  loss_history.append(loss)
  if epoch % 100 == 0:
    print(epoch, loss)

  # Delta_2 has shape(4, 1), the first dimension being the batch dimension
  delta_2 = 0.5 * ( a_2 - Y)
  g_prime_1 = np.heaviside(z_1, 0)

  # Delta_1 has shape (4, 2)
  delta_1 = np.matmul(delta_2, m.weights_2[1:3, :].T) * g_prime_1

  # Biases of layers connecting input and hidden layers
  m.weights_1[0, :] -= learning_rate * np.sum(delta_1[:, :], axis=0)

  # Weights of layers connecting input and hidden layers
  m.weights_1[1:3, :] -= learning_rate * np.matmul(X.T, delta_1)

  # Biases of layers connecting hidden and output layers
  m.weights_2[0, :] -= learning_rate * np.sum(delta_2[:, :], axis=0)

  # Weights of layers connecting hidden and output layers
  m.weights_2[1:3, :] -= learning_rate * np.matmul(a_1.T, delta_2)
  weights_1_history.append(np.copy(m.weights_1))
  weights_2_history.append(np.copy(m.weights_2))

loss_history = np.array(loss_history)
weights_1_history = np.array(weights_1_history)
weights_2_history = np.array(weights_2_history)

#plotting the loss and weights over the epochs
plt.figure(figsize=(15, 5))
ax = plt.subplot(2, 5, 1)
ax.plot(loss_history[:])
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('Loss', fontsize=14)
ax = plt.subplot(2, 5, 2)
ax.plot(weights_1_history[:,0,0])
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('c1', fontsize=14)
ax = plt.subplot(2, 5, 3)
ax.plot(weights_1_history[:,0,1])
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('c2', fontsize=14)
ax = plt.subplot(2, 5, 4)
ax.plot(weights_1_history[:,1,0])
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('W11', fontsize=14)
ax = plt.subplot(2, 5, 5)
ax.plot(weights_1_history[:,1,1])
ax.set_xlabel('Epoch', fontsize=14)
ax.set_ylabel('W12', fontsize=14)
ax = plt.subplot(2, 5, 6)
```

```
  ax.plot(weights_1_history[:,2,0])
  ax.set_xlabel('Epoch', fontsize=14)
  ax.set_ylabel('W21', fontsize=14)
  ax = plt.subplot(2, 5, 7)
  ax.plot(weights_1_history[:,2,1])
  ax.set_xlabel('Epoch', fontsize=14)
  ax.set_ylabel('W22', fontsize=14)
  ax = plt.subplot(2, 5, 8)
  ax.plot(weights_2_history[:,0,0])
  ax.set_xlabel('Epoch', fontsize=14)
  ax.set_ylabel('b', fontsize=14)
  ax = plt.subplot(2, 5, 9)
  ax.plot(weights_2_history[:,1,0])
  ax.set_xlabel('Epoch', fontsize=14)
  ax.set_ylabel('w1', fontsize=14)
  ax = plt.subplot(2, 5, 10)
  ax.plot(weights_2_history[:,2,0])
  ax.set_xlabel('Epoch', fontsize=14)
  ax.set_ylabel('w2', fontsize=14)
  plt.tight_layout()
  plt.show()

  print(m.weights_1)

  print(m.weights_2)

  z_1, a_1, z_2, a_2 = m.forward(X)
  print(a_2)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
[[0]
 [1]
 [1]
 [0]]
0.0
0 0.5523485297294992
100 0.20759268485068139
200 0.11289892843584107
300 0.0014812675618753757
400 3.4251827763134314e-06
```




```
[[ 2.89951162e-06 -5.72659156e-06]
 [ 8.55850924e-01 -8.40680620e-01]
 [-8.55849632e-01  8.40673755e-01]]
[[9.82877968e-05]
 [1.16823513e+00]
 [1.18931798e+00]]
[[1.01675108e-04]
 [9.99919890e-01]
 [9.99936789e-01]
 [1.03184563e-04]]
```

Malachy and Aroushi -

Week Starting January 5th:

Read sections 1.1 - 1.3:

Thursday Feburary 8th 10am-11am: SUPERVISOR MEETING

- Discussed information from sections 1.1 - 1.3 and tested understanding.
- Contexualised this information to galaxy classification.

- Adam looked over our plan draft and gave feedback.
- Introduced us to google colab and how to use it (& why it is useful for machine learning, $$$ GPU)
- Look at tensorflow documentation, quickstart guide & tutorials
- Look at google colab.
- Read up to section of the notes.

11am-1pm PROJECT PARTNER MEETING

- Finalised project plan, considering recived feeback.
- Submitted project plan.
- Looked at google colab (Emailed Matt Young as we have problems acessing repo through this).
- Set up eDiary.
- Introduced Aroushi to github pushing/pulling changes breifely.

Malachy - wrote notes on CNNs

Aroushi -

- # 1 Continued:
  - ## 1.4 Deep Learning Frameworks:

    Facilitate building and training neural networks by abstracting the complexity. Provides pre-built layers, activation functions, and optimizers. Enables automatic differentiation, simplifying the implementation of backpropagation. Offers various levels of abstraction, from high-level (easy to use, less flexible) to low-level (complex, more control). For the purpose of this project, we plan to use xyz

  - ## 1.5 Optimisation:
    - The learning rate must be optimised to prevent the situations illustrated in Fig x.
    - Since cost functions can have multiple local and saddle points, we must ensure that in flat directions (small gradients), one would like to take large step sizes to adjust learning rate, and in steep directions, one must take small step sizes. There are various popular optimisation methods.

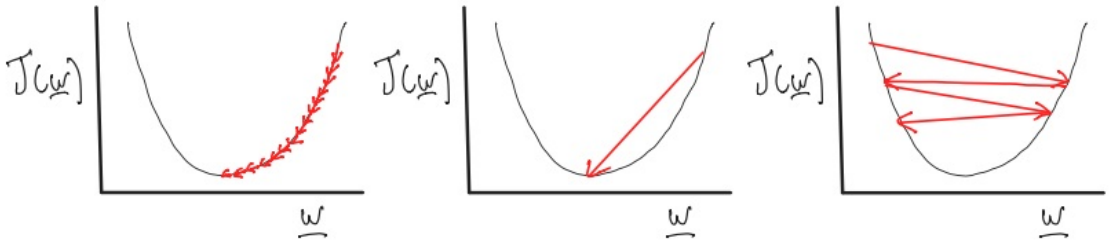


Figure 1.5.1: Illustration of a learning rate that is (left) too small; (middle) optimal; (right) too large.

    - **1.5.1 Stochastic Gradient Descent**

      Fundamental technique for minimizing the loss function, adjusting weights in the direction of the negative gradient multiplied by the learning rate α. It uses gradient values over a mini-batch of B examples such that

      $$\hat{\mathbf{g}} = \frac{1}{B}\nabla_w \sum_i^B l(\hat{y}^{(i)}(\theta), y^{(i)})$$

    - **1.5.2 Stochastic Gradient Descent with Momentum**

      Variation of gradient descent using a subset of data, reducing computation and adding noise that can prevent local minima convergence. It accelerates gradients vectors in the right direction, leading to faster converging.

    - **1.5.3 RMSprop**

      Adjusts the learning rate for each weight based on recent magnitudes of gradients for the weight, which helps in faster convergence.

    - **1.5.5 Adam**

Combines momentum and RMSprop, adjusting the learning rate based on first and second moments of the gradients.

- **Comparison**

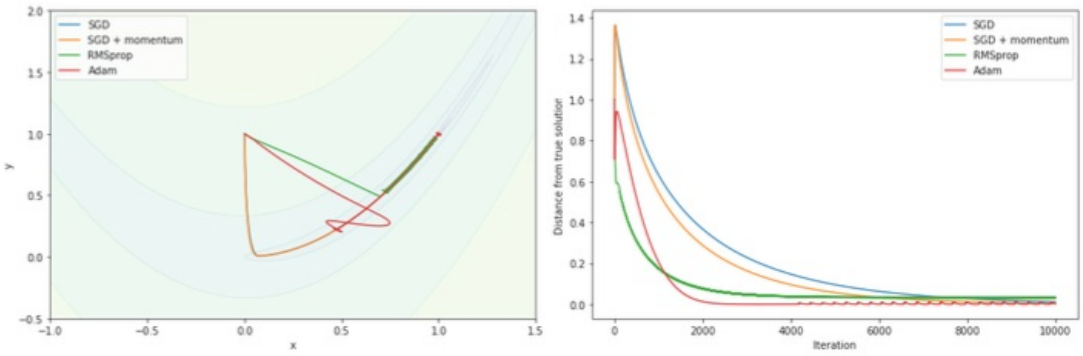


Figure 1.5.2: Benchmark of the SGD, SGD + momentum, RMSprop and Adam optimisers on the Rosenbrock function, starting from $(x, y) = (1, 0)$.

Fig. X we show the trajectories of each update, along with the distance from the global minimum as a function of iteration. SGD takes the longest to reach the minimum, but the addition of momentum helps move along the flat region more quickly. The adaptive methods work best, and it is noticeable how initially, rather than moving almost vertically, they track towards the minima, as the result of the larger learning rate in the x direction.

1.6 Initialisation:

Proper initialization of weights is vital for effective training. Poor initialization can lead to vanishing/exploding gradients or slow convergence. Strategies include random initialization, Xavier/Glorot initialization, and He initialization.

1.7 Regularisation:

Purpose: Techniques to reduce overfitting, ensuring the model generalizes better to unseen data.

- **1.7.1 Early Stopping**

  Stops training as soon as the performance on a validation set starts to deteriorate.

- **1.7.2 L_1/L_2 regularisation**

  Adds penalty terms to the loss function proportional to the magnitude of coefficients to encourage simpler models.

- **1.7.3 Dropout**

  Randomly sets a fraction of input units to 0 at each update during training time, which helps prevent overfitting.

- **1.7.4 Batch Normalisation**

  Standardises the inputs to a layer for each mini-batch, stabilizing the learning process and reducing the number of epochs required to train deep networks.

- **1.7.5 Data Augmentation**

  Increases the diversity of data available for training models, without actually collecting new data.

Malachy -

In [ ]:
```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np

run = False

if run:
    tf.random.set_seed(1)
    print(tf.__version__)

    X = np.array([[0,0], [0,1], [1,0], [1,1]]) #x1, x2
    print(X)
```

```
    Y = np.array([[0], [1], [1], [0]]) #XOR gate output
    print(Y)

    def build_model():
        """
        Builds a simple neural network with 2 input nodes, 2 hidden nodes and 1 output node.
        """
        #build the model, define the number of layers, the number of nodes in each layer, the activation functi
        model = keras.Sequential([
            layers.Dense(2, activation='relu', input_shape=[X.shape[1]]),
            layers.Dense(1)
        ])

        optimizer = keras.optimizers.RMSprop(0.01) #optimisation algorithm, Root Mean Square Propagation

        #compile the model
        model.compile(loss='mse',
                      optimizer=optimizer,
                      metrics=['mae', 'mse'])

        return model

    model = build_model()
    model.summary()

    num_epochs = 300

    model.fit(X, Y, epochs=num_epochs, verbose=1)

    plt.plot(model.history.history['loss'])
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

    model.predict(X)
```

```
2.10.0
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
[[0]
 [1]
 [1]
 [0]]
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
Epoch 1/300
1/1 [==============================] - 1s 641ms/step - loss: 0.6346 - mae: 0.6144 - mse: 0.6346
Epoch 2/300
1/1 [==============================] - 0s 10ms/step - loss: 0.5060 - mae: 0.5278 - mse: 0.5060
Epoch 3/300
1/1 [==============================] - 0s 8ms/step - loss: 0.4371 - mae: 0.4967 - mse: 0.4371
Epoch 4/300
1/1 [==============================] - 0s 5ms/step - loss: 0.3916 - mae: 0.4959 - mse: 0.3916
Epoch 5/300
1/1 [==============================] - 0s 7ms/step - loss: 0.3591 - mae: 0.4954 - mse: 0.3591
Epoch 6/300
1/1 [==============================] - 0s 4ms/step - loss: 0.3350 - mae: 0.4950 - mse: 0.3350
Epoch 7/300
1/1 [==============================] - 0s 6ms/step - loss: 0.3169 - mae: 0.4946 - mse: 0.3169
Epoch 8/300
1/1 [==============================] - 0s 13ms/step - loss: 0.3032 - mae: 0.4944 - mse: 0.3032
Epoch 9/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2927 - mae: 0.4941 - mse: 0.2927
Epoch 10/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2847 - mae: 0.4939 - mse: 0.2847
Epoch 11/300
1/1 [==============================] - 0s 9ms/step - loss: 0.2785 - mae: 0.4936 - mse: 0.2785
Epoch 12/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2739 - mae: 0.4934 - mse: 0.2739
Epoch 13/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2703 - mae: 0.4932 - mse: 0.2703
```

```
Epoch 14/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2674 - mae: 0.4929 - mse: 0.2674
Epoch 15/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2652 - mae: 0.4927 - mse: 0.2652
Epoch 16/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2633 - mae: 0.4924 - mse: 0.2633
Epoch 17/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2617 - mae: 0.4921 - mse: 0.2617
Epoch 18/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2603 - mae: 0.4918 - mse: 0.2603
Epoch 19/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2590 - mae: 0.4914 - mse: 0.2590
Epoch 20/300
1/1 [==============================] - 0s 10ms/step - loss: 0.2577 - mae: 0.4910 - mse: 0.2577
Epoch 21/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2564 - mae: 0.4906 - mse: 0.2564
Epoch 22/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2551 - mae: 0.4901 - mse: 0.2551
Epoch 23/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2538 - mae: 0.4897 - mse: 0.2538
Epoch 24/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2525 - mae: 0.4891 - mse: 0.2525
Epoch 25/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2511 - mae: 0.4885 - mse: 0.2511
Epoch 26/300
1/1 [==============================] - 0s 4ms/step - loss: 0.2497 - mae: 0.4879 - mse: 0.2497
Epoch 27/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2483 - mae: 0.4873 - mse: 0.2483
Epoch 28/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2468 - mae: 0.4865 - mse: 0.2468
Epoch 29/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2453 - mae: 0.4858 - mse: 0.2453
Epoch 30/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2438 - mae: 0.4849 - mse: 0.2438
Epoch 31/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2422 - mae: 0.4840 - mse: 0.2422
Epoch 32/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2406 - mae: 0.4831 - mse: 0.2406
Epoch 33/300
1/1 [==============================] - 0s 9ms/step - loss: 0.2390 - mae: 0.4821 - mse: 0.2390
Epoch 34/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2373 - mae: 0.4810 - mse: 0.2373
Epoch 35/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2356 - mae: 0.4798 - mse: 0.2356
Epoch 36/300
1/1 [==============================] - 0s 10ms/step - loss: 0.2339 - mae: 0.4786 - mse: 0.2339
Epoch 37/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2321 - mae: 0.4773 - mse: 0.2321
Epoch 38/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2303 - mae: 0.4760 - mse: 0.2303
Epoch 39/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2284 - mae: 0.4745 - mse: 0.2284
Epoch 40/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2265 - mae: 0.4730 - mse: 0.2265
Epoch 41/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2246 - mae: 0.4714 - mse: 0.2246
Epoch 42/300
1/1 [==============================] - 0s 9ms/step - loss: 0.2227 - mae: 0.4697 - mse: 0.2227
Epoch 43/300
1/1 [==============================] - 0s 4ms/step - loss: 0.2207 - mae: 0.4680 - mse: 0.2207
Epoch 44/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2187 - mae: 0.4662 - mse: 0.2187
Epoch 45/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2167 - mae: 0.4643 - mse: 0.2167
Epoch 46/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2146 - mae: 0.4623 - mse: 0.2146
Epoch 47/300
1/1 [==============================] - 0s 7ms/step - loss: 0.2126 - mae: 0.4603 - mse: 0.2126
Epoch 48/300
1/1 [==============================] - 0s 5ms/step - loss: 0.2104 - mae: 0.4582 - mse: 0.2104
Epoch 49/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2083 - mae: 0.4560 - mse: 0.2083
Epoch 50/300
1/1 [==============================] - 0s 8ms/step - loss: 0.2062 - mae: 0.4537 - mse: 0.2062
Epoch 51/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2040 - mae: 0.4514 - mse: 0.2040
Epoch 52/300
1/1 [==============================] - 0s 6ms/step - loss: 0.2017 - mae: 0.4490 - mse: 0.2017
Epoch 53/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1995 - mae: 0.4466 - mse: 0.1995
Epoch 54/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1972 - mae: 0.4440 - mse: 0.1972
Epoch 55/300
```

```
1/1 [==============================] - 0s 10ms/step - loss: 0.1949 - mae: 0.4414 - mse: 0.1949
Epoch 56/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1925 - mae: 0.4388 - mse: 0.1925
Epoch 57/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1901 - mae: 0.4360 - mse: 0.1901
Epoch 58/300
1/1 [==============================] - 0s 14ms/step - loss: 0.1877 - mae: 0.4332 - mse: 0.1877
Epoch 59/300
1/1 [==============================] - 0s 10ms/step - loss: 0.1853 - mae: 0.4304 - mse: 0.1853
Epoch 60/300
1/1 [==============================] - 0s 9ms/step - loss: 0.1828 - mae: 0.4275 - mse: 0.1828
Epoch 61/300
1/1 [==============================] - 0s 8ms/step - loss: 0.1802 - mae: 0.4245 - mse: 0.1802
Epoch 62/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1777 - mae: 0.4214 - mse: 0.1777
Epoch 63/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1751 - mae: 0.4183 - mse: 0.1751
Epoch 64/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1724 - mae: 0.4152 - mse: 0.1724
Epoch 65/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1698 - mae: 0.4119 - mse: 0.1698
Epoch 66/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1671 - mae: 0.4087 - mse: 0.1671
Epoch 67/300
1/1 [==============================] - 0s 8ms/step - loss: 0.1644 - mae: 0.4053 - mse: 0.1644
Epoch 68/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1616 - mae: 0.4019 - mse: 0.1616
Epoch 69/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1588 - mae: 0.3985 - mse: 0.1588
Epoch 70/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1575 - mae: 0.3968 - mse: 0.1575
Epoch 71/300
1/1 [==============================] - 0s 9ms/step - loss: 0.1549 - mae: 0.3932 - mse: 0.1549
Epoch 72/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1529 - mae: 0.3895 - mse: 0.1529
Epoch 73/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1510 - mae: 0.3860 - mse: 0.1510
Epoch 74/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1481 - mae: 0.3839 - mse: 0.1481
Epoch 75/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1450 - mae: 0.3802 - mse: 0.1450
Epoch 76/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1430 - mae: 0.3777 - mse: 0.1430
Epoch 77/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1411 - mae: 0.3744 - mse: 0.1411
Epoch 78/300
1/1 [==============================] - 0s 8ms/step - loss: 0.1385 - mae: 0.3710 - mse: 0.1385
Epoch 79/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1371 - mae: 0.3684 - mse: 0.1371
Epoch 80/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1341 - mae: 0.3647 - mse: 0.1341
Epoch 81/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1327 - mae: 0.3623 - mse: 0.1327
Epoch 82/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1295 - mae: 0.3585 - mse: 0.1295
Epoch 83/300
1/1 [==============================] - 0s 9ms/step - loss: 0.1282 - mae: 0.3562 - mse: 0.1282
Epoch 84/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1251 - mae: 0.3523 - mse: 0.1251
Epoch 85/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1238 - mae: 0.3499 - mse: 0.1238
Epoch 86/300
1/1 [==============================] - 0s 9ms/step - loss: 0.1221 - mae: 0.3482 - mse: 0.1221
Epoch 87/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1216 - mae: 0.3456 - mse: 0.1216
Epoch 88/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1176 - mae: 0.3410 - mse: 0.1176
Epoch 89/300
1/1 [==============================] - 0s 8ms/step - loss: 0.1152 - mae: 0.3383 - mse: 0.1152
Epoch 90/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1131 - mae: 0.3350 - mse: 0.1131
Epoch 91/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1117 - mae: 0.3327 - mse: 0.1117
Epoch 92/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1102 - mae: 0.3308 - mse: 0.1102
Epoch 93/300
1/1 [==============================] - 0s 7ms/step - loss: 0.1074 - mae: 0.3261 - mse: 0.1074
Epoch 94/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1078 - mae: 0.3271 - mse: 0.1078
Epoch 95/300
1/1 [==============================] - 0s 5ms/step - loss: 0.1055 - mae: 0.3215 - mse: 0.1055
Epoch 96/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1031 - mae: 0.3187 - mse: 0.1031
```

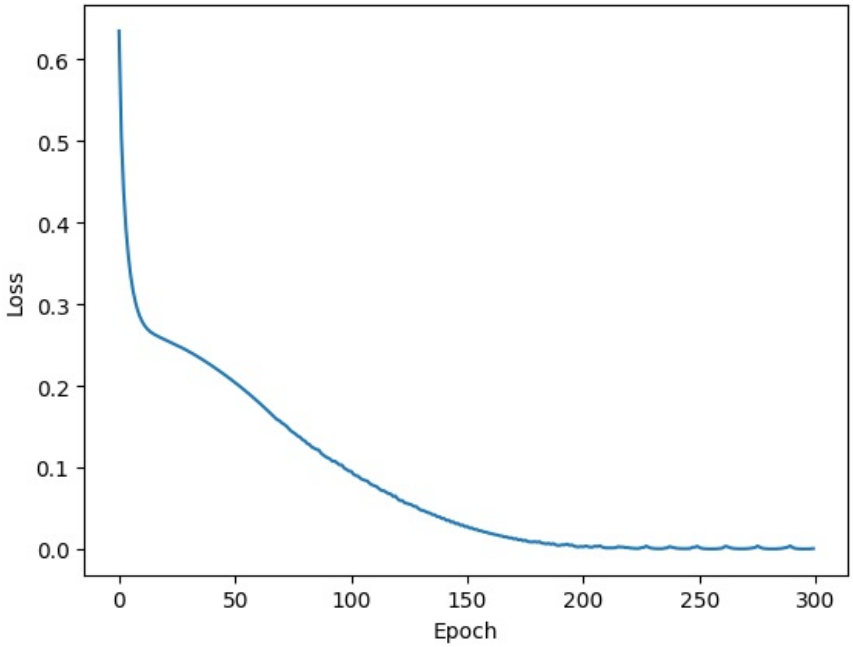
```
Epoch 97/300
1/1 [==============================] - 0s 6ms/step - loss: 0.1031 - mae: 0.3167 - mse: 0.1031
Epoch 98/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0992 - mae: 0.3125 - mse: 0.0992
Epoch 99/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0976 - mae: 0.3099 - mse: 0.0976
Epoch 100/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0956 - mae: 0.3075 - mse: 0.0956
Epoch 101/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0954 - mae: 0.3052 - mse: 0.0954
Epoch 102/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0917 - mae: 0.3004 - mse: 0.0917
Epoch 103/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0901 - mae: 0.2986 - mse: 0.0901
Epoch 104/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0895 - mae: 0.2963 - mse: 0.0895
Epoch 105/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0865 - mae: 0.2918 - mse: 0.0865
Epoch 106/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0854 - mae: 0.2894 - mse: 0.0854
Epoch 107/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0841 - mae: 0.2882 - mse: 0.0841
Epoch 108/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0834 - mae: 0.2846 - mse: 0.0834
Epoch 109/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0804 - mae: 0.2807 - mse: 0.0804
Epoch 110/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0786 - mae: 0.2774 - mse: 0.0786
Epoch 111/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0778 - mae: 0.2770 - mse: 0.0778
Epoch 112/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0765 - mae: 0.2724 - mse: 0.0765
Epoch 113/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0740 - mae: 0.2695 - mse: 0.0740
Epoch 114/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0719 - mae: 0.2652 - mse: 0.0719
Epoch 115/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0717 - mae: 0.2659 - mse: 0.0717
Epoch 116/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0699 - mae: 0.2602 - mse: 0.0699
Epoch 117/300
1/1 [==============================] - 0s 10ms/step - loss: 0.0680 - mae: 0.2582 - mse: 0.0680
Epoch 118/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0677 - mae: 0.2551 - mse: 0.0677
Epoch 119/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0646 - mae: 0.2509 - mse: 0.0646
Epoch 120/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0652 - mae: 0.2498 - mse: 0.0652
Epoch 121/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0615 - mae: 0.2446 - mse: 0.0615
Epoch 122/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0595 - mae: 0.2418 - mse: 0.0595
Epoch 123/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0596 - mae: 0.2404 - mse: 0.0596
Epoch 124/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0568 - mae: 0.2355 - mse: 0.0568
Epoch 125/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0560 - mae: 0.2331 - mse: 0.0560
Epoch 126/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0552 - mae: 0.2328 - mse: 0.0552
Epoch 127/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0544 - mae: 0.2279 - mse: 0.0544
Epoch 128/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0524 - mae: 0.2256 - mse: 0.0524
Epoch 129/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0523 - mae: 0.2224 - mse: 0.0523
Epoch 130/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0496 - mae: 0.2190 - mse: 0.0496
Epoch 131/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0478 - mae: 0.2149 - mse: 0.0478
Epoch 132/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0474 - mae: 0.2155 - mse: 0.0474
Epoch 133/300
1/1 [==============================] - 0s 4ms/step - loss: 0.0461 - mae: 0.2096 - mse: 0.0461
Epoch 134/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0448 - mae: 0.2085 - mse: 0.0448
Epoch 135/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0442 - mae: 0.2042 - mse: 0.0442
Epoch 136/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0423 - mae: 0.2019 - mse: 0.0423
Epoch 137/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0421 - mae: 0.1986 - mse: 0.0421
Epoch 138/300
```

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0398 - mae: 0.1955 - mse: 0.0398
Epoch 139/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0398 - mae: 0.1930 - mse: 0.0398
Epoch 140/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0374 - mae: 0.1893 - mse: 0.0374
Epoch 141/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0377 - mae: 0.1875 - mse: 0.0377
Epoch 142/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0352 - mae: 0.1833 - mse: 0.0352
Epoch 143/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0356 - mae: 0.1819 - mse: 0.0356
Epoch 144/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0331 - mae: 0.1776 - mse: 0.0331
Epoch 145/300
1/1 [==============================] - 0s 20ms/step - loss: 0.0336 - mae: 0.1763 - mse: 0.0336
Epoch 146/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0311 - mae: 0.1720 - mse: 0.0311
Epoch 147/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0316 - mae: 0.1707 - mse: 0.0316
Epoch 148/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0292 - mae: 0.1665 - mse: 0.0292
Epoch 149/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0297 - mae: 0.1651 - mse: 0.0297
Epoch 150/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0275 - mae: 0.1612 - mse: 0.0275
Epoch 151/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0278 - mae: 0.1594 - mse: 0.0278
Epoch 152/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0258 - mae: 0.1560 - mse: 0.0258
Epoch 153/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0260 - mae: 0.1538 - mse: 0.0260
Epoch 154/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0242 - mae: 0.1509 - mse: 0.0242
Epoch 155/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0242 - mae: 0.1481 - mse: 0.0242
Epoch 156/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0227 - mae: 0.1459 - mse: 0.0227
Epoch 157/300
1/1 [==============================] - 0s 13ms/step - loss: 0.0225 - mae: 0.1425 - mse: 0.0225
Epoch 158/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0212 - mae: 0.1410 - mse: 0.0212
Epoch 159/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0209 - mae: 0.1369 - mse: 0.0209
Epoch 160/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0198 - mae: 0.1363 - mse: 0.0198
Epoch 161/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0193 - mae: 0.1312 - mse: 0.0193
Epoch 162/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0185 - mae: 0.1316 - mse: 0.0185
Epoch 163/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0178 - mae: 0.1256 - mse: 0.0178
Epoch 164/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0173 - mae: 0.1270 - mse: 0.0173
Epoch 165/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0164 - mae: 0.1201 - mse: 0.0164
Epoch 166/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0161 - mae: 0.1225 - mse: 0.0161
Epoch 167/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0150 - mae: 0.1145 - mse: 0.0150
Epoch 168/300
1/1 [==============================] - 0s 10ms/step - loss: 0.0150 - mae: 0.1181 - mse: 0.0150
Epoch 169/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0137 - mae: 0.1090 - mse: 0.0137
Epoch 170/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0139 - mae: 0.1138 - mse: 0.0139
Epoch 171/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0125 - mae: 0.1035 - mse: 0.0125
Epoch 172/300
1/1 [==============================] - 0s 10ms/step - loss: 0.0129 - mae: 0.1095 - mse: 0.0129
Epoch 173/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0113 - mae: 0.0981 - mse: 0.0113
Epoch 174/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0120 - mae: 0.1054 - mse: 0.0120
Epoch 175/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0102 - mae: 0.0928 - mse: 0.0102
Epoch 176/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0111 - mae: 0.1013 - mse: 0.0111
Epoch 177/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0093 - mae: 0.0876 - mse: 0.0093
Epoch 178/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0090 - mae: 0.0913 - mse: 0.0090
Epoch 179/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0087 - mae: 0.0846 - mse: 0.0087
```

```
Epoch 180/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0093 - mae: 0.0902 - mse: 0.0093
Epoch 181/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0085 - mae: 0.0798 - mse: 0.0085
Epoch 182/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0093 - mae: 0.0887 - mse: 0.0093
Epoch 183/300
1/1 [==============================] - 0s 4ms/step - loss: 0.0076 - mae: 0.0759 - mse: 0.0076
Epoch 184/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0071 - mae: 0.0797 - mse: 0.0071
Epoch 185/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0065 - mae: 0.0708 - mse: 0.0065
Epoch 186/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0072 - mae: 0.0786 - mse: 0.0072
Epoch 187/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0060 - mae: 0.0685 - mse: 0.0060
Epoch 188/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0069 - mae: 0.0769 - mse: 0.0069
Epoch 189/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0054 - mae: 0.0652 - mse: 0.0054
Epoch 190/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0045 - mae: 0.0628 - mse: 0.0045
Epoch 191/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0048 - mae: 0.0604 - mse: 0.0048
Epoch 192/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0054 - mae: 0.0650 - mse: 0.0054
Epoch 193/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0053 - mae: 0.0680 - mse: 0.0053
Epoch 194/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0063 - mae: 0.0667 - mse: 0.0063
Epoch 195/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0046 - mae: 0.0644 - mse: 0.0046
Epoch 196/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0053 - mae: 0.0647 - mse: 0.0053
Epoch 197/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0037 - mae: 0.0563 - mse: 0.0037
Epoch 198/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0030 - mae: 0.0512 - mse: 0.0030
Epoch 199/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0028 - mae: 0.0458 - mse: 0.0028
Epoch 200/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0033 - mae: 0.0523 - mse: 0.0033
Epoch 201/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0029 - mae: 0.0507 - mse: 0.0029
Epoch 202/300
1/1 [==============================] - 0s 14ms/step - loss: 0.0039 - mae: 0.0537 - mse: 0.0039
Epoch 203/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0031 - mae: 0.0541 - mse: 0.0031
Epoch 204/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0025 - mae: 0.0436 - mse: 0.0025
Epoch 205/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0026 - mae: 0.0494 - mse: 0.0026
Epoch 206/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0037 - mae: 0.0477 - mse: 0.0037
Epoch 207/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0032 - mae: 0.0555 - mse: 0.0032
Epoch 208/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0042 - mae: 0.0496 - mse: 0.0042
Epoch 209/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0027 - mae: 0.0508 - mse: 0.0027
Epoch 210/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0019 - mae: 0.0381 - mse: 0.0019
Epoch 211/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0015 - mae: 0.0372 - mse: 0.0015
Epoch 212/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0016 - mae: 0.0354 - mse: 0.0016
Epoch 213/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0015 - mae: 0.0377 - mse: 0.0015
Epoch 214/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0018 - mae: 0.0345 - mse: 0.0018
Epoch 215/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0018 - mae: 0.0426 - mse: 0.0018
Epoch 216/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0032 - mae: 0.0394 - mse: 0.0032
Epoch 217/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0026 - mae: 0.0492 - mse: 0.0026
Epoch 218/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0027 - mae: 0.0369 - mse: 0.0027
Epoch 219/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0019 - mae: 0.0434 - mse: 0.0019
Epoch 220/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0020 - mae: 0.0338 - mse: 0.0020
Epoch 221/300
```

```
1/1 [==============================] - 0s 6ms/step - loss: 0.0015 - mae: 0.0378 - mse: 0.0015
Epoch 222/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0011 - mae: 0.0267 - mse: 0.0011
Epoch 223/300
1/1 [==============================] - 0s 6ms/step - loss: 9.4747e-04 - mae: 0.0308 - mse: 9.4747e-04
Epoch 224/300
1/1 [==============================] - 0s 6ms/step - loss: 8.5214e-04 - mae: 0.0229 - mse: 8.5214e-04
Epoch 225/300
1/1 [==============================] - 0s 6ms/step - loss: 9.6876e-04 - mae: 0.0311 - mse: 9.6876e-04
Epoch 226/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0019 - mae: 0.0309 - mse: 0.0019
Epoch 227/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0021 - mae: 0.0435 - mse: 0.0021
Epoch 228/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0038 - mae: 0.0442 - mse: 0.0038
Epoch 229/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0024 - mae: 0.0450 - mse: 0.0024
Epoch 230/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0014 - mae: 0.0265 - mse: 0.0014
Epoch 231/300
1/1 [==============================] - 0s 6ms/step - loss: 9.7255e-04 - mae: 0.0306 - mse: 9.7255e-04
Epoch 232/300
1/1 [==============================] - 0s 8ms/step - loss: 7.3193e-04 - mae: 0.0200 - mse: 7.3193e-04
Epoch 233/300
1/1 [==============================] - 0s 7ms/step - loss: 6.1628e-04 - mae: 0.0247 - mse: 6.1628e-04
Epoch 234/300
1/1 [==============================] - 0s 6ms/step - loss: 5.6522e-04 - mae: 0.0173 - mse: 5.6522e-04
Epoch 235/300
1/1 [==============================] - 0s 5ms/step - loss: 5.9940e-04 - mae: 0.0242 - mse: 5.9940e-04
Epoch 236/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0013 - mae: 0.0256 - mse: 0.0013
Epoch 237/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0015 - mae: 0.0356 - mse: 0.0015
Epoch 238/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0031 - mae: 0.0410 - mse: 0.0031
Epoch 239/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0025 - mae: 0.0429 - mse: 0.0025
Epoch 240/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0017 - mae: 0.0309 - mse: 0.0017
Epoch 241/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0012 - mae: 0.0312 - mse: 0.0012
Epoch 242/300
1/1 [==============================] - 0s 5ms/step - loss: 8.5956e-04 - mae: 0.0222 - mse: 8.5956e-04
Epoch 243/300
1/1 [==============================] - 0s 8ms/step - loss: 7.0694e-04 - mae: 0.0250 - mse: 7.0694e-04
Epoch 244/300
1/1 [==============================] - 0s 8ms/step - loss: 6.3778e-04 - mae: 0.0195 - mse: 6.3778e-04
Epoch 245/300
1/1 [==============================] - 0s 8ms/step - loss: 6.4017e-04 - mae: 0.0235 - mse: 6.4017e-04
Epoch 246/300
1/1 [==============================] - 0s 7ms/step - loss: 7.0397e-04 - mae: 0.0212 - mse: 7.0397e-04
Epoch 247/300
1/1 [==============================] - 0s 7ms/step - loss: 9.5164e-04 - mae: 0.0277 - mse: 9.5164e-04
Epoch 248/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0023 - mae: 0.0362 - mse: 0.0023
Epoch 249/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0024 - mae: 0.0404 - mse: 0.0024
Epoch 250/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0036 - mae: 0.0441 - mse: 0.0036
Epoch 251/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0020 - mae: 0.0374 - mse: 0.0020
Epoch 252/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0010 - mae: 0.0234 - mse: 0.0010
Epoch 253/300
1/1 [==============================] - 0s 8ms/step - loss: 6.1974e-04 - mae: 0.0228 - mse: 6.1974e-04
Epoch 254/300
1/1 [==============================] - 0s 6ms/step - loss: 4.3276e-04 - mae: 0.0152 - mse: 4.3276e-04
Epoch 255/300
1/1 [==============================] - 0s 6ms/step - loss: 3.5307e-04 - mae: 0.0178 - mse: 3.5307e-04
Epoch 256/300
1/1 [==============================] - 0s 10ms/step - loss: 3.2829e-04 - mae: 0.0138 - mse: 3.2829e-04
Epoch 257/300
1/1 [==============================] - 0s 7ms/step - loss: 3.5012e-04 - mae: 0.0174 - mse: 3.5012e-04
Epoch 258/300
1/1 [==============================] - 0s 7ms/step - loss: 4.2564e-04 - mae: 0.0166 - mse: 4.2564e-04
Epoch 259/300
1/1 [==============================] - 0s 7ms/step - loss: 5.9209e-04 - mae: 0.0210 - mse: 5.9209e-04
Epoch 260/300
1/1 [==============================] - 0s 7ms/step - loss: 9.0497e-04 - mae: 0.0248 - mse: 9.0497e-04
Epoch 261/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0015 - mae: 0.0313 - mse: 0.0015
Epoch 262/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0036 - mae: 0.0462 - mse: 0.0036
```

```
Epoch 263/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0031 - mae: 0.0464 - mse: 0.0031
Epoch 264/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0020 - mae: 0.0349 - mse: 0.0020
Epoch 265/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0012 - mae: 0.0282 - mse: 0.0012
Epoch 266/300
1/1 [==============================] - 0s 9ms/step - loss: 7.7998e-04 - mae: 0.0221 - mse: 7.7998e-04
Epoch 267/300
1/1 [==============================] - 0s 5ms/step - loss: 5.6815e-04 - mae: 0.0201 - mse: 5.6815e-04
Epoch 268/300
1/1 [==============================] - 0s 6ms/step - loss: 4.6856e-04 - mae: 0.0174 - mse: 4.6856e-04
Epoch 269/300
1/1 [==============================] - 0s 4ms/step - loss: 4.4457e-04 - mae: 0.0178 - mse: 4.4457e-04
Epoch 270/300
1/1 [==============================] - 0s 6ms/step - loss: 4.7739e-04 - mae: 0.0178 - mse: 4.7739e-04
Epoch 271/300
1/1 [==============================] - 0s 4ms/step - loss: 5.7976e-04 - mae: 0.0195 - mse: 5.7976e-04
Epoch 272/300
1/1 [==============================] - 0s 6ms/step - loss: 7.7060e-04 - mae: 0.0228 - mse: 7.7060e-04
Epoch 273/300
1/1 [==============================] - 0s 9ms/step - loss: 0.0011 - mae: 0.0261 - mse: 0.0011
Epoch 274/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0015 - mae: 0.0319 - mse: 0.0015
Epoch 275/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0021 - mae: 0.0383 - mse: 0.0021
Epoch 276/300
1/1 [==============================] - 0s 8ms/step - loss: 0.0039 - mae: 0.0479 - mse: 0.0039
Epoch 277/300
1/1 [==============================] - 0s 5ms/step - loss: 0.0024 - mae: 0.0414 - mse: 0.0024
Epoch 278/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0013 - mae: 0.0276 - mse: 0.0013
Epoch 279/300
1/1 [==============================] - 0s 5ms/step - loss: 7.3052e-04 - mae: 0.0222 - mse: 7.3052e-04
Epoch 280/300
1/1 [==============================] - 0s 5ms/step - loss: 4.7360e-04 - mae: 0.0171 - mse: 4.7360e-04
Epoch 281/300
1/1 [==============================] - 0s 5ms/step - loss: 3.6162e-04 - mae: 0.0158 - mse: 3.6162e-04
Epoch 282/300
1/1 [==============================] - 0s 5ms/step - loss: 3.2144e-04 - mae: 0.0144 - mse: 3.2144e-04
Epoch 283/300
1/1 [==============================] - 0s 6ms/step - loss: 3.3447e-04 - mae: 0.0150 - mse: 3.3447e-04
Epoch 284/300
1/1 [==============================] - 0s 9ms/step - loss: 3.9986e-04 - mae: 0.0164 - mse: 3.9986e-04
Epoch 285/300
1/1 [==============================] - 0s 5ms/step - loss: 5.4454e-04 - mae: 0.0184 - mse: 5.4454e-04
Epoch 286/300
1/1 [==============================] - 0s 6ms/step - loss: 8.1005e-04 - mae: 0.0234 - mse: 8.1005e-04
Epoch 287/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0013 - mae: 0.0284 - mse: 0.0013
Epoch 288/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0018 - mae: 0.0350 - mse: 0.0018
Epoch 289/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0025 - mae: 0.0412 - mse: 0.0025
Epoch 290/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0040 - mae: 0.0486 - mse: 0.0040
Epoch 291/300
1/1 [==============================] - 0s 7ms/step - loss: 0.0022 - mae: 0.0404 - mse: 0.0022
Epoch 292/300
1/1 [==============================] - 0s 6ms/step - loss: 0.0011 - mae: 0.0251 - mse: 0.0011
Epoch 293/300
1/1 [==============================] - 0s 5ms/step - loss: 6.0787e-04 - mae: 0.0207 - mse: 6.0787e-04
Epoch 294/300
1/1 [==============================] - 0s 9ms/step - loss: 3.9077e-04 - mae: 0.0154 - mse: 3.9077e-04
Epoch 295/300
1/1 [==============================] - 0s 5ms/step - loss: 2.9969e-04 - mae: 0.0140 - mse: 2.9969e-04
Epoch 296/300
1/1 [==============================] - 0s 5ms/step - loss: 2.7052e-04 - mae: 0.0132 - mse: 2.7052e-04
Epoch 297/300
1/1 [==============================] - 0s 5ms/step - loss: 2.8839e-04 - mae: 0.0135 - mse: 2.8839e-04
Epoch 298/300
1/1 [==============================] - 0s 14ms/step - loss: 3.5591e-04 - mae: 0.0155 - mse: 3.5591e-04
Epoch 299/300
1/1 [==============================] - 0s 8ms/step - loss: 5.0313e-04 - mae: 0.0179 - mse: 5.0313e-04
Epoch 300/300
1/1 [==============================] - 0s 8ms/step - loss: 7.8002e-04 - mae: 0.0229 - mse: 7.8002e-04
```




```
1/1 [==============================] - 0s 75ms/step
```

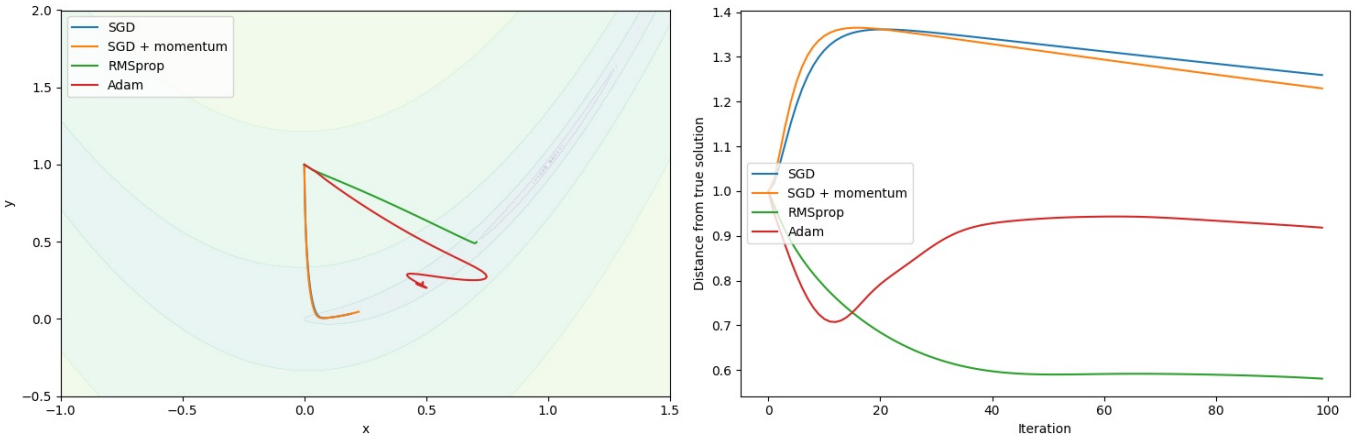
Malachy -

In [ ]:

```python
#the rosenbrock function is a non-convex function used for bench-marking optimization methods.
#it is defined as f(x,y) = (1-x)^2 + 100(y-x^2)^2
#and has a global minimum at (x,y) = (1,1)

#benchmark the optimisers included with tensorflow, SGD, SGD with momentum, RMSprop, and Adam
#starting from (x,y) = (0,1).

from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np

run = False
if run:

    def rosenbrock(x, y): #rosenbrock function
        return (1 - x)**2 + 100 * (y - x**2)**2

    #benchmark function
    def benchmark(f, opt, iterations, x_init, y_init):

        x = tf.Variable(x_init)
        y = tf.Variable11(y_init)
        history = []

        for i in range(iterations):
            with tf.GradientTape() as tape:
                z = f(x, y)

                grads = tape.gradient(z, [x, y])
                history.append([x.numpy(), y.numpy(), grads[0].numpy(), grads[1].numpy()])

                processed_grads = [g for g in grads]
                grads_and_vars = zip(processed_grads, [x, y])

                opt.apply_gradients(grads_and_vars)

        return np.array(history)

    iterations = 100
    x_init = 0.0
    y_init = 1.0

    #benchmark the optimisers
```

```
    sgd_history = benchmark(rosenbrock, keras.optimizers.SGD(learning_rate=0.001), iterations, x_init, y_init)
    momentum_history = benchmark(rosenbrock, keras.optimizers.SGD(learning_rate=0.001, nesterov=True, momentum=
    rmsprop_history = benchmark(rosenbrock, keras.optimizers.RMSprop(learning_rate=0.01), iterations, x_init, y_
    adam_history = benchmark(rosenbrock, keras.optimizers.Adam(learning_rate=0.05), iterations, x_init, y_init)

    xx, yy = np.mgrid[-1:2:.01, -0.5:2:.01]
    zz = rosenbrock(xx, yy)

    #plotting
    plt.figure(figsize=(15, 5))
    ax = plt.subplot(1, 2, 1)
    ax.contourf(xx, yy, np.log(zz), alpha=0.1)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.plot(sgd_history[:, 0], sgd_history[:, 1], label='SGD')
    ax.plot(momentum_history[:, 0], momentum_history[:, 1], label='SGD + momentum')
    ax.plot(rmsprop_history[:, 0], rmsprop_history[:, 1], label='RMSprop')
    ax.plot(adam_history[:, 0], adam_history[:, 1], label='Adam')
    ax.set_xlim([-1,1.5])
    ax.set_ylim([-0.5,2])
    ax.legend(loc='upper left')
    ax = plt.subplot(1, 2, 2)
    ax.set_xlabel('Iteration')
    ax.set_ylabel('Distance from true solution')
    ax.plot(((sgd_history[:, 0] - 1)**2 + (sgd_history[:, 1] - 1)**2)**0.5, label = 'SGD')
    ax.plot(((momentum_history[:, 0] - 1)**2 + (momentum_history[:, 1] - 1)**2)**0.5, label = 'SGD + momentum')
    ax.plot(((rmsprop_history[:, 0] - 1)**2 + (rmsprop_history[:, 1] - 1)**2)**0.5, label = 'RMSprop')
    ax.plot(((adam_history[:, 0] - 1)**2 + (adam_history[:, 1] - 1)**2)**0.5, label = 'Adam')
    ax.legend()
    plt.tight_layout()
    plt.show()
```

```
C:\Users\Ma1a_\AppData\Local\Temp\ipykernel_5328\2609381022.py:58: RuntimeWarning: divide by zero encountered in
log
  ax.contourf(xx, yy, np.log(zz), alpha=0.1)
```

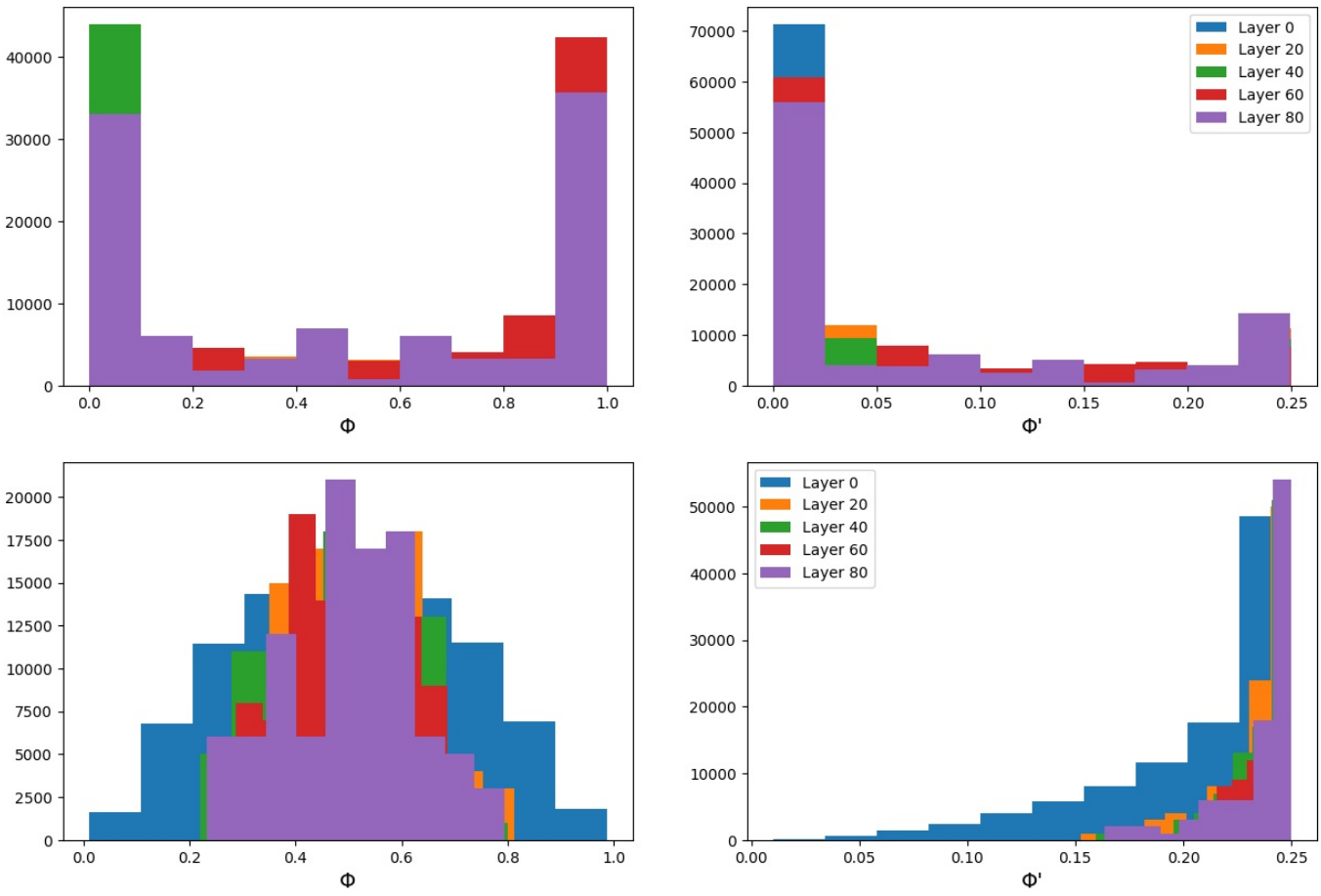


Malachy -

In [ ]:

```
# during backpropagation, it is important that the gradients do not become too large or too small, as
# the alogirthm progresses down to the lower layers of the network. This is known as the vanishing gradients
# or exploding gradients problem. This can be mitigated by using activation functions that do not saturate
# too quickly, and by using batch normalisation.

# consider a MLP with 100 hidden layers, each with 100 hidde units (with zero bias) and a sigmoid activation fu
# we assume the input data (with dimension 100) and the weights are normally distributed with mean 0 and varian

import numpy as np
import matplotlib.pyplot as plt

run = False

if run:
    np.random.seed(0)

    def sigmoid(x):
        return 1/(1 + np.exp(-x))

    hidden_units = 100

    X = np.random.normal(size=(1000, hidden_units))

    def test_initialisation(X, num_layers, activation, scale):
        """
        Test the initialisation of the weights and biases for a MLP with 100 hidden layers,
        each with 100 hidden units (with zero bias) and a sigmoid activation function.
        """
```

```
        a = np.copy(X)
        plt.figure(figsize=(15,10))

        for i in range(num_layers):

            W = np.random.normal(scale=scale, size=(hidden_units, hidden_units)) #initialise the weights to sma
            z = np.matmul(a, W)
            a = np.copy(activation(z))

            if i % 20 == 0: #plot the activation and its derivative every 20 layers

                ax = plt.subplot(2,2,1)
                _ = ax.hist(a.flatten(), label='Layer ' + str(i))
                ax.set_xlabel('Φ', fontsize=14)
                ax = plt.subplot(2,2,2)
                _ = ax.hist(a.flatten() * (1 - a.flatten() ), label='Layer ' + str(i))
                ax.set_xlabel("Φ'", fontsize=14)

        plt.legend()
        plt.show()

    test_initialisation(X, 100, sigmoid, 1)

    # ideally, we preserve the signal variance as we pass through the network, such that it neither vanishes no
    # The variance of the output of a neuron should therefore be equal to the variance of its input.
    # In order for the variance ot to be preserved, we need to set the variance of the weights to 1/number_of_i
    # During backpropagationm, we would like to preserve the variance of the gradients as well, which can be ac
    # the variance of the weights to 1/number_of_outputs, where number_of_outputs is the number outputs from a

    # a heuristic for balancing both requirements is a weight initialsation of either:
    # W = uniform(-sqrt(6/(n_in + n_out)), sqrt(6/(n_in + n_out)))
    #This is Xavier initialisation.

    #if the weights are drawn from a uniform distribution and the inital weights are drawn from a normal distri
    # W = normal(0, sqrt(2/(n_inputs + n_outputs)))
    #This is Glorot initialisation.

    test_initialisation(X, 100, sigmoid, np.sqrt(1 / hidden_units))
```



Malachy and Aroushi -

Week Starting Febuary 12th:

Read sections 1.4 - 1.7.5

Thursday Febuary 15th 12pm-1pmL SUPERVISOR MEETING

- looked at basic image classifcation in tensorflow.
- Ran TensorFlow's image classification example code line by line and formed an understanding of keras functions
- Example code used: https://www.tensorflow.org/tutorials/keras/classification - "This guide trains a neural network to classify images of clothing"
- The example used tf.keras, which we discussed was a good option for a high-level API to help train our neural network

Malachy - More notes on CNNs (Extracted from Machine Learning notes provided):

2 Convolutional Neural Networks:

Convolutional neural networks (CNNs) are a specialized type of neural network designed for processing tensor-like data, such as images. They utilize convolutional layers, where input tensors are convolved with learnable kernels instead of traditional matrix multiplication. The objective of training a CNN is to optimize the weights and biases within these kernels, typically achieved through techniques like backpropagation and gradient descent. CNNs are widely used for tasks such as image classification, object detection, and image segmentation.

2.1 History:

Before deep learning, image classification relied on feature extraction and data compression, they would look for complex non-linear relationships between the features. Deep supervised neural networks, perform well but are too difficult to train due to the huge number of parameters and non convex loss functions.

In 1988 neural networks were used on image datasets similar to the MNIST dataset, a handwritten 10-digit recognition problem, it was shown that multi-layered networks in a hierachical manner reduce the number of free parameters and provide significant improvement in the generalisation and peformance of a network.

This concept was refined and in 1998 LeNet-5, a convolutional network trained with gradient descent back propagation, caused a paradigm shift in machine learning, and deep learning.

2.2 Architecture:

The architecture of a network is the sequence of layers that it contains. Each layer has a different function, and not all layers have trainable parameters.

2.2.1 Convolution

Convolution (*) is an operation on 2 functions f(t) and g(t), that produces a new smoothed signal s(t) at time t,

$$s(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)\,d\tau$$

- s(t) = feature map
- f(t) = input
- g(t) = kernel filter

(Convolution does not have to be time specific)

For signals at discrete intervals we have discrete convolution:

$$s(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau)$$

The convolutional layer is the principle layer type in a CNN. Whilst time series data is most commonly a 1D problem, image data is a 2D problem.

In a 2D discrete convolution layer, the input (**I**) is a feature map of size

$$I_x \times I_y \times I_z$$

For an RGB image the feature map can be represented as a tensor, a 2D matrix where each element of the matrix is a vector containing the RGB values of a pixel (overall a 3D matrix).

A Kernel (**K**) of size

$$K_x \times K_y \times I_z \times N$$

is applied to the input tensor (**I**) where N is the number of filters, and produces an output feature map (**O**) of size

$$O_x \times O_y \times O_z$$

The output of a fully connected neural network node is,

$$\hat{y} = \phi \sum_i w_i x_i + b$$

where x is the input, w are the multiplicative weights, b is the additive bias, and Φ is the activation function.

The 2D discrete convolutional layer is given by

$$\hat{Y} = \phi \sum_{} I * K + \vec{b}$$

$$\vec{b}$$

is the bias, which is a vector of length:

$$O_z$$

i.e a scalar value additive bias is applied to every pixel for each output channel.

For stride one and no pading, the convolutional part is

$$O_{i,j,k} = (I * K)(i,j,k)$$

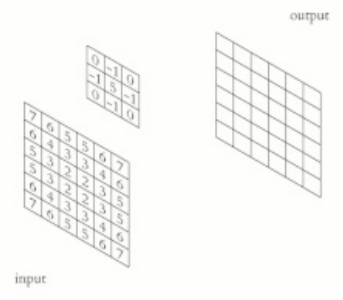
$$= \sum_{l,m,n} K_{l,m,k,n} \cdot I_{i+l-1,j+m-1,n}$$

The convolution of the kernel is applied at each position of the image. Each kernel element is multiplied bu the overlapping image values and summed.

For a N x N x 1 image matrix, it can be reshaped into a column vector of size (N x N) x 1.

and the kernel can be unrolled into a convolutional matrix of size

$$(I_x - K_x + 1)(I_y - K_y + 1) \times (I_x I_y)$$



Animation of Discrete 2D Convolution

The values in the kernel are unknown parameters and are determined during the training of the network.

The total number of parameters in a convolutional layer is

$$N_{param} = (K_x \times K_y \times O_z + 1) \times I_z$$

The + 1 is due to the biases.

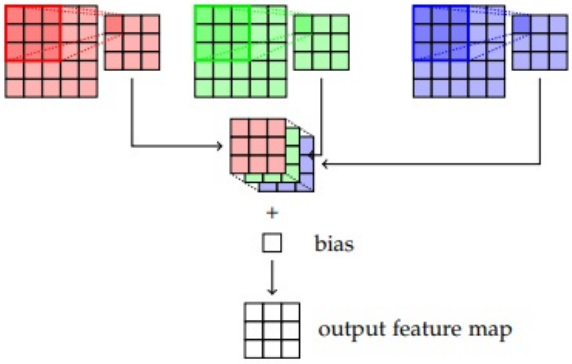
Convolutional layers have 4 hyperparameters which control the output volume of each layer that must be defined. These are the kernel size, number of fiters, padding, and stride.

- **2.2.2 Kernel size**

Kenel size is also called the receptive field.

Most images have 3 channels (Red, Green, Blue) and the number of channels increases with the depth of the network. In the case of Inception moduels (section 2.5) where parallel feature maps are concatenated together, the number of channels can increase rapidly.

For multi-channel inputs, a kernel with the same number of channels is applied and the resulting feature maps are summed together before the bias is applied to produce the output feature map.



2D Convolution for multi-channel inputs, the resulting feature maps are summed to give a single channel map and a bias is added to produce a single channel output feature map

The number of filters determines the number of channels of the output (feature map), and the depth of the filter must always be the same as the number of input channels.

Each filter is looking for different features within the image. Each of the feature maps have their own bias applied, and these are then concatenated together to output a feature map with the same number of channels are the number of filters.

Padding:

Due to the way convolution is applied, the size of the input image will very quickly shrink after a series of convoloutions -> loss of information.

The pixels at the edges of each input feature are convolved with the kernel less frequently than the inner pixels.

This can be solved with padding, a margin is added around the input image. Typically this is filled with zeroes (zero padding).

Half padding or same padding is when the padding value is

$$P = (K - 1)/2$$

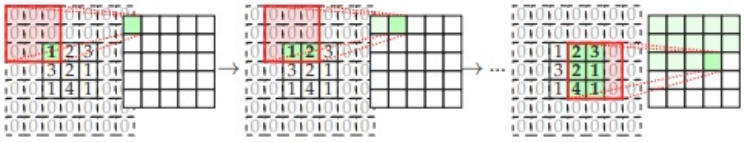
This results in an output image of the same dimensions as the input image.

Convolution of a 3x3 input with same padding applied with a 3x3 kernel to output a 3x3 output feature map.

Full padding occurs when the padding value is

$$P = K - 1$$

This ensures that every possible convolution of the kernel with the input is taken into account equally.



Convolution of a 3x3 input with sfull padding applied with a 3x3 kernel to output a 5x5 output feature map.

Any other value of passing can be used and it does not have to be symmetric to any of the 4 corners (bottom left, bottom right, top left, top right).

Valid padding refers to no padding being applied to the input image.

Stride:

The stride determines how many pixels each kernel moves across the input before it is applied.

The output of a fully connected layer used in traditional neural networks is independent of input size, this is not the case for convolutional layers. The output is dependent on the input size, but also the stride, padding, and kernel size.

The output of the convolutional layer is a matrix of size

$$W_2 \times H_2 \times D_2$$

pixels, where:

$$O_x = (I_x - K + 2P)/S + 1$$

$$O_y = (I_y - K + 2P)/S + 1$$

$$O_z = N$$

Where N is the number of filters, P is the padding, S is the stride, and K is the kernel size. These equations also apply to pooling layers, where by default P = 0 .

NOTE: For S > 1, If (I - K + 2P) % S = 0 . Then the output size will always be the same regardless of the input size. The number of parameters in the network is therefore independent of the input image size, and only dependent on the size and number of filters.

Activation:

After convolving the image with the filters, a bias value is added and an activation function is applied elementwise for the aggregation of information.

The most common activation function used in CNNs is the rectified linear unit (ReLU). It is common because its derivative is simple and therefore fast to compute.

Activation functions add non-linearity to the network and are typically applied after each convolutional layer since convolution is a linear operation.

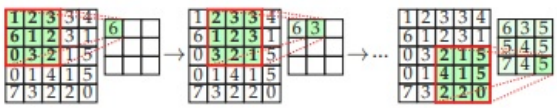
- **2.2.3 Pooling**

  Pooling is used to down sample, to reduce the dimensionality of the input and thus reducing the number of free parameters in the network.

  Pooling helps with the invariance of the network to small shifts and transformations in the data.

  The Pooling layer has no trainable parameters but the kernel size, stride, and padding must be defined.

  There are many different kinds of pooling layers, the max pooling layer has a max pool kernel, which maps the largest value of the input within the kernel to the output for each stride. The average pool layer similary returns the mean value of the input within the kernel.



Max pool operation where a 5x5 input is convolved with a 3x3 max pool kernel that returns the maximum value in regions of size 3x3 in the image.

- **2.2.4 Fully connected**

  The convolutional and pooling layers can be thought of as feature extractors. They enchance the relevant features or absence of features.

  The fully connected (or dense) layers aggregate together this information across the final feature maps. In this layer every input element has its own weight to map it to the output layer.

  They are typically the last layer that provides the results of final classification.

  The fully connected layer takes 2 hyperparameters - the number of nodes, which is typically the same as the number of classes (C), and the activation function.

  If the layer is used as a classification then usually this will be the softmax function:

$$S(x_i) = \frac{exp(x_i)}{\sum_j^C exp(x_j)}$$

  To ensure the sum of scores over all classes is normalised, since the softmax is a discrete probability distribution. The derivative of the softmax is then,

$$\frac{\delta S(x_i)}{\delta x_j} = \begin{cases} S(x_i)(1 - S(x_j)), & \text{if } i = j \\ -S(x_i)S(x_j), & \text{otherwise} \end{cases}$$

  NOTE: softmax can be numerically unstable for large values of x, since the exponential of a large number is astronomically large. Therefore we rescale the data first, often the largest value of x is subtracted from each element of x such that the largest value of the data is 1.

2.3 Other layers:

- **2.3.1 Batch normalisation**

  In CNNs, after each layer the values of the elements in the feature maps can grown very quickly. This can cause issues for the convergence of weights because the parameter search can be very large and thus slow.

  CNNs are therefore restricted to a slow learning rate values and can be sensitive to the initialisation of weights.

  Batch Normalisation improves the efficiency and performance of CNNs.

$$\hat{x}_i = \frac{x_i - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}}$$

  Where, μ_x is the mean of x and σ_x^2 is its variance. ϵ is a small arbitary constant added for numerical stability.

  The layer renormalises the input such that it has a mean of 0 and a variance of 1 when

$$\epsilon = 0$$

  This changes the representation of the layer so an additional scale (γ) and shift (β)

  parameters are introduced to restore the representation of the layer and these parameters are learned during training.

$$y = \gamma\hat{x}_i + \beta$$

  In the cases of stochastic and batch gradient descent, batch norm can be applied on mini-batches. They are applied after the activation function.

  Batch norm improves effficiency and introduces noise into the network which helps to regularise the network.

  Batch norm should be used instead of dropout (section 2.2.4).

- **2.3.2 Pointwise convolution**

  Since the number of channels quickly increases with the depth of the network, so does the number of parameters and consequently the computational complexity increases.

  So we haave to reduce the number of parameters in the layers, however pooling only reduces the dimensionality in the width and height.

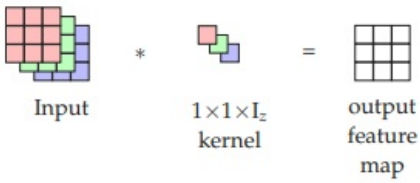
  Pointwise convolutions are $1x1xI_z$ kernel filters.

  Where I_z are the number of input channels.

  The number of pointwise filters applied will give the same number of output channels in this layer. Pointwise convolution can be used to reduce or inncrease the depth of the input, and thus reduce or increase it's dimensionality.

  Or they can be used to increase the non-linearity of the layers.

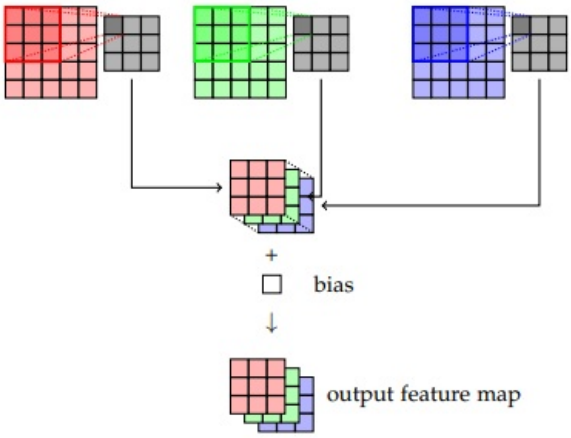
  Sometimes described as a cross channel pooling filter.



  Pointwise convolution of a 3x3x3 input with a 1x1x3 kernel. The number of kernels determines the number of output channels.

- **2.3.3 Depthwise convolution**

  In depthwise convolution, each channel of the multi-channel input is convolved with the respective channel of the filter as per the standard multi-channel convolution, however the output features are not summed together but instead are concatenated.



Depthwise convolution. A 5x5x3 input is convolved with a single 3x3 kernel.

This operation specifically uses only 1 filter. Section 2.5 covers the use of this operation for dimensionality reduction.

- **2.3.4 Dilated convolution**

  Dilated convolution is where the kernel is applied with padding between kernel values. This layer can be used to integrate information from different spatial scales, balancing between high accuracies on the pixel lvel whilst also inferring information from the global context. i.e., super resolution and denoisinng problems.

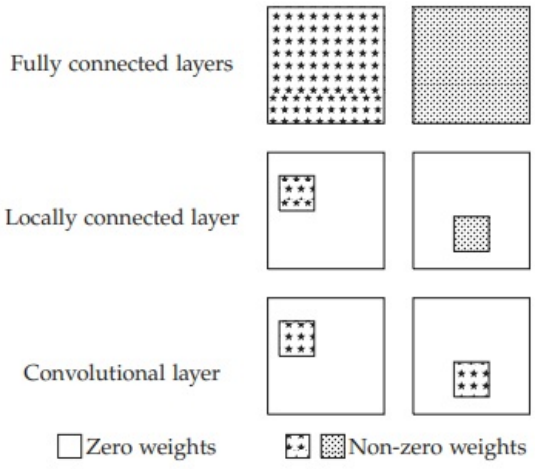
- **2.3.5 Locally connected layer**

  The locally connected layer is a cross between a standard convolutional layer and a fully connected layer. In a fully connected layer each input pixel has its ownn weights andn biases connecting it to each output pixel and therefore the number of parameters is very high.

  For a 5x5 input the number of parameters is 5x5+1 = 26 (+1 for the bias).

  In convolutional layers, local regions share the weights of the filter kernel, significantly reducing the number of parameters. The same 5x5 input using a 3x3 kernel would only have 3x3+1 parameters (+1 for the bias).

  Locally connected layers have a median number of parameters, like a convolutional layer its looking for specific features and therefore uses a small kernel with weights, but these weights are not shared over all regions of the image and instead focus on specific regions.



A visual comparison of fully connected layers, locally connected layers, and convolutional layers.

  - *In fully connected layers all elements of the input have their own weights and biases*
  - *In convolutional layers, weights and biases are shared across local regions of the size of the kernel, and the kernel is applied across the entire input*
  - *In locally connected layers the kernel is only applied over a small region of the input and there is no weight and bias sharing*

  This layer is useful for when classifying objects with known locations of features, i.e., when classifying humans based on portrait images, the nose is always at the centre of the image and thus the kernels responsible for picking out this feature will only be applie to the centre of the image.

- **2.3.6 Up sampling**

  CCNs are not limited to classification problems, you for example may want to keep the dimensions of the network output the same or larger than the input dimensions. For this up sampling of layers is required.

Transposed Convolution:

Transposed convolution is an up sampling layer. It takes an input and returns a spatially larger output.

It is sometimes known as a deconvolution layer (IT IS NOT THE DECOVOLUTION OPERATION - THE INVERSE OF A CONVOLUTION OPERATION)

The transpose of the convolutional matrix that represents the kernel filter is multiplied by the unrolled input matrix to produce an output matrix with higher dimensions than the input.

With the same padding, the output size of the convolution layer is the I x S, and with vald padding, the output size is (I- 1)S + K.

Images created using transposed convolution can leave a checkerboard like artefact caused by the uneven overlapping of the mapping from the input to the output at each kernel operation. This occurs when the kernel size is non-divisible by the stride. Trying to avoid such choices of hyperparmaters in these layers can help reduce this effect, but they generally cannot completely mitigate the problem since CNNs use multiple layers of transpose convolution futher complicating the effect.

Resize-convolution:

An alternative method that avoids the checkerboard artefacts.

This involves first upsampling the input feature map using an interpolation method such as Bi-linear interpolation or nearest neightbour interpolation and then applying a normal convolutional layer.

- ## 2.4 Reguarisation:

In deep learning there are many more parameters than a standard neural network so you are more likely to overfit to your training data if the training data set is small.

One way to improve the generalisability of a model is to increase the size of the training dataset.

- **2.4.1 Augmentation**

  Augmentation is the process of increasing the size of your training dataset by applying transformations to the already existing training data.

  This could be:

  - cropping an image
  - changing the brightness or contrast
  - rescaling and translation
  - rotation and mirroring

  It might be tempting to use all the possible augmentation to maximise the training data size, however, not all the transformatios will have impactful gains on the perfomance of the network and many significantly increase training time.

  The types of transformations chosen should be carefully considered within the context of the problem. Horizontally flipping an image for detecting car lanes would be a good idea, however the same transformation on handwriting images, i.e. distinguishing the letter 'b' from 'd', would only confuse the network.

- **2.4.2 Dropout**

  You can prevent over fitting on small datasets by training multiple different architectures on a dataset and take the average.

  This is known as ensemble learning methods.

  These methods are computationally expensive, and memory intensive to store the networks.

  The dropout method can be used instead. Here random nodes in the model are randomly ignored according to a set probability during the training. This simulates having a large number of different architectures.

  Having dropout in a layer means the activation to the layer is sparse and allows the network to learn a sparse represenntation.

  Dropout is commonly used in deep networks, however dropout in a fully connected layer is not the same as dropout in a convolutional layer. After a convolutional layer, dropout can be seen as adding noise to the network and therefore it is perferable to use batch normalisation instead.

- ## 2.5 Architectures:

Architecture refers to the layout of different neurons, layers and activation functions. Typically it is designed through trial and error, although there are some proven techniques.

Convolutional networks are powerful however they do not work well unless the data reflects the real world. Imagenet is a compilation of 14 million everyday objects categorised into over 200,000 classes with hand made bounding boxes.

Imagenet allowed for the development of multipurpose architectures. Most notably Alexnet.

Alexnet:

LeNet-5 had 5 layers; 3 convolutional and 2 fully connected layers, giving it around 60,000 parameters. Alexnet has 5 convolutional layers and 3 fully connected layers, giving it around 60 million parameters. Alexnet was the first deep convolutional neural network.

CNNs were limited depthwise due to the lack of computational power which limited their perfomance however Alexnet was one of the first networks to use GPU hardware acceleration, the Alexnet architecture is designed for use with 2 GPUs. It's record breaking performace was due to the training on the huge Imagenet dataset, along with the use of reguarisation techniques, like data augmentation and dropout. Along with the use of the RELU activation function which significantly improved training efficiency. Despite this the network still almost took a week to train.

MobileNet:

The design of architectures is a trade off between efficiency and accuracy. A big, complex network may be more accurate, howerever it will have a large number of parameters and be slow to train.

MobileNet is designed to be a small and low latency architecture, such that training can be performed on mobile devices.

The main concept behind the architecture is that the standard convolutional filter is computationally expensive. For layers where there is more than 1 filter being used, the standard convolutional layer is replaced with 2 layers; a depthwise convolution followed by a pointwise convolution with the same depth as the number of filters required when using the standard convolutional layer.

This is the depthwise separable filter.

It recuces the number of parameters in the layer but also the number of multiplication operations and hence the speed.

MobileNet uses a series of 3x3 depthwise separable filters for a factor of 8 increase over conventional convolutional layers.

VGGNet:

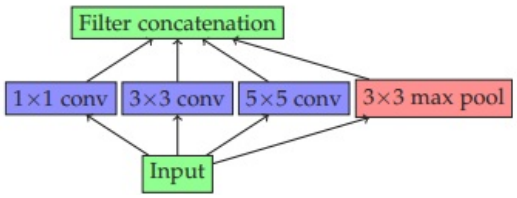
VGGNet demonstrated that two 3x3 convolutional layers is equivalent to a 5x5 receptive field and thus able to capture larger features whilst maintaining a small number of parameters to train for.

This is why 3x3 kernels are commonly used in CNNs. As they reduce the complexity of the network but allow for deep architectures that save on memory and power and are more generalisable.

Inception:

Whilst previous architectures used linearly stacked convolutional layers, Inception uses an inception module.

The main concept behind the inception module is to apply different sized filters in parallel and then concatenate them together. In Inception v1, the inception module was essentially a 1x1, a 3x3 and a 5x5 filter.



The naive Inception cell contains various sized filters in parallel. The full Inception cell also contians various pointwise convolutions to obtain consistnent dimensions

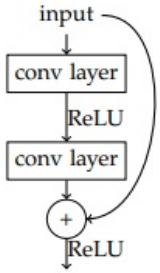
This allows for learning on multiple scales. i.e., classifying the same objects but of different sizes.

Resnet:

An ultradeep CNN with 152 layers. Given that the best human top 5 error on the Imagenet dataset is 5%, Resnet supassed humans with a 3.6% top 5 error.

Naively, it may appear that simply increasing the depth of a network can improve testing accuracies, however the authors show this is not the case.

The main feature of Renset is the use of a residual cell.



Residual cell use in Resnet consists of a convolutional-ReLU convolution before the output is summed with the input and another ReLU is applied.

The residual net cell consists of the operations of a convolutional layer, followed by ReLU activation, followed by convolution. The output of this is then summed with the input before another ReLU activation is applied.

If a traditional convolutional layer takes an input x and ouputs s(x), the residual cell instead outputs s(x) + x.

The cell is computing the residual changes to the input image, which is easier to optimise than the traditional convolutional layer.

Region based-CNNs:

Are CNNs designed specifically for object detection. The problem is no longer simply classificaton, but also localisation of objects. The input to an R-CNN would be an image, and the output would be a vector containing the class of object, classification score and coordinates for a bounding box around the object of interest.

R-CNNNs do this by splitting the problem into two networks.

The first extracts region proposals from the image using a selective search to generate thousands of regions in the image where there is a high probability of a positive detection. These propoals are then resized to a common size for input into the second part of R-CNNs.

A CNN that extracts the features in the proposed regions. Here the output vector enters a linear SVM and bounding box regressor to train for the classification and bounding box coordinates.

- ## 2.6 Applications:

CNNs typically refer to 2D convolution, such as images. However CCNs are not exclusive to image classification. They can be applied to 1D data such as audio or other time series data, they can be applied to 3D data such as videos.

For 1D data CNNs are searching from fixed-length features, irrespective of the location.

- **2.6.1 Image recognition**

  Image recognition is the term to descibe computer vision tasks for identifying objects in images. It can be divided into 4 main categories:

  1. Classification:

  Here the task is to predict the class that features in an image. i.e., an image may contain a cat, and the task could be to predictr the breed of the cat.

  2. Localisation:

  In image classification, you typically have only one object in the image, and although you may have more, the object types will be the same.

  Sometimes you may not be interested in the object itself but instead where they are located in the image.

  Image localisation using CNNs works exactly the same way as classification but instead of returning a score for a given class, we CNN returns 8 numbers to describe the location of the object, this could be the 4 coordinates of the bounding box of the object.

  $$(C \quad)$$
  $$\_ \; i,j$$

  or more typically the width, height and centre coordinate of the box.

  The loss function could be the L2 loss, equivalent to the euclidean

  distance between the true coordinates and the predicted $\hat{C}$
  $$\_ \; i,j$$

  $$l = \sum_{i=1}^{2} \sum_{j=1}^{2} || C \quad - \hat{C} \quad ||_2$$
  $$\_ \; i,j \quad \_ \; i,j$$

where $||x||_2$ is the L2 norm.

3. Detection:

For images wherein exists multiple objects of different classes, image detection can be used to both localise objects and classify them. Object detection is a combination of image classification and localisation.

The output of the network would be the bounding box parameters for the detected objects amd their classification scores.

One way to to this would to use sliding boxes of various scales to apply a classical CNN for image classification each patch of the input image, this however is extremely inefficient and the choice of box scale and aspect ratio needs to be determined.

Architectures like R-CNN were developed to obtain a more principled delection of boxes to apply the standard CNN, using selective search to propose regions of interset, a CNN to classify and then regression to optimise.

The variations on R-CNN such as as fast RCNN and faster RCNN, split the tasks of classification and localisation.

These methods do not process the entire image at once, instead they look at high resolution patches.

The selective search algorithm can be very slow - thousands of forward passes through the CNN are required for each proposal region. So R-CNN architectures are not ideal for real time processing.

Single shot detectors SSDs use a single CNN architecture to obtain both bounding box parameters and object class. In an SSd, convolutional layers are first applied to reduce the dimensionality of the input.

The input is split into a grid of cells and at each cell, anchors (predefined initial guesses of the box) are then used to make detections.

These types of models use low resolution images to make the detection, they are very fast but trade off accuracy. They are more adapted to real time processing for this reason.

The performance on low resolution images is scale dependent, affecting the detection of small scale objects.

4. Segmentation:

In segmentation, objects are classified on a pixel-by-pixel basis. This is desired if you have multiple overlapping objects or prehaps are interested in the outlines of objects.

Image segmentation can be split into two different categories:

- Semantic Segmentation: Classifies each pixel of an input to identify the object class it belongs to. Using a sliding box CNN approach to classify every pixel would be possible but very inefficient. Fully convolutional layers are commonly used for image segmentation. They use a combinatiopn of downsampling (e.g. convolution) and upsampling (transpose convolution) layers to produce an output wth the same size as the input image.
- Instance Segmentation: Classifies each pixel of an input to identify the object instance it belongs to. For example a litter of puppies segmented using semantic segmentation would produce a single segment for all the puppies, but instance segmentation provides segments for each individual puppy.

  Instance segmentation is a combination of semantic segmentation and object detection.

  Mask-RCNN is a popular architecture for instance segmentation, it is based on Faster R-CNN. In addition to the object class scoe, and bounding box, the output of the CNN will also be an image mask to represent the segmentation.

  The image mask is produced by taking into account each region of intrest separately.

Applications of image segmentation are: facial recognition, medical imaging, and autonomous driving.

- **2.6.2 Image enhancement**

  Fully convolutional networks are used in image enhancement and restoration problems including:

  - denoising
  - deblurring
  - artefact removal
  - super resolution (taking a low resolution image and transforming it to a high resolution output)

2.7 Model Evaluation:

During training, the loss if often the metric that is used to monitor how the model is performing as a function of training time.

Simultanesouly monitoring both the loss of the training dataset and the validation data, can tell you if the model is performing as it should (i.e. dropping loss), if it's underfitting (i.e. dropping validation loss) or if it's overfitting (i.e. increasing validation loss).

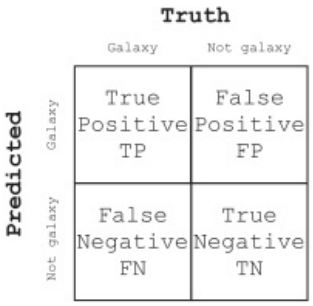
However, the loss cannot tell you how well the model is performing. The value of the loss is relative and depends on the loss

function used.

Other metrics are required to describe the performance of a network and enable comparison between other models.

- **2.7.1 Confusion Matrix**

  For CNNs classification is the most common task. For binary classification, i.e. if an image is of a galaxy or not a galaxy, we can log the classifications of a network on a test data set in a confusion matrix.



  Confusion Matrix of a binary classification model, the matix values are filled with the numbers from the model prediction.

  - True positives TP are the number of images correctly classified as being a galaxy
  - True Negatives TN are the number of images correctly classified as not being a galaxy
  - False Positives FP are the number of non-galaxy images incorrectly classified as a galaxy.
  - False Negatives FN are the number of galaxy images incorretly classified as not being a galaxy.

  The sum of the columns tell you the total number of galaxies P and non-galaxies N.

  This matrix can be extended for multi-class classification by extending the number of rows and columns to represent each class.

  The confusion matrix can be used to spot biases in the model.

- **2.7.2 Accuracy**

  The accuracy of a model is given by

$$ACC = \frac{TP + TN}{P + N}$$

  The accuracy in general is a good metric to use for classification since it tells us with a single number how well the networks performs on a scale of 0-1.

  However it is not a good metric if the data classes are unbalanced. Accuracy favours the larger class. i.e., if there were 9 non-galaxies and 1 galaxy, if the model predicts all objects as non-galaxies, it would still get a 90% accuracy, even though it is a terrible galaxy classifier.

- **2.7.3 Precision, Recall, and F1 score**

  - Precision tells us the purity of the predictions,

$$PRE = \frac{TP}{TP + FP}$$

  - Recall tells us the completeness of the predictions,

$$REC = \frac{TP}{TP + FN}$$

  Precision is more important where you need predictions free of contaminants. e.g. supernoca lightcurves, you wouldn't want a variable star to contaminate the signal.

  Recall is more important where you need to know even if there is a slight possibility, e.g. cancer detection.

  - The F1 Score is a mid point between being pure and complete,

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

- **2.7.4 Multiclass metrics**

  For multiple classes, the above metrics can be calculated as a micro-average to give equal weights to each individual classification, e.g.

$$PRE_{mic} = \frac{TP_1 + TP_2 + \dots}{TP_1 + TP_2 + \dots + FP_1 + FP_2 + \dots}$$

This metric is biased towards large classes, therefore the macro-average is more preferable and give equal weight to each class, e.g.,

$$PRE_{mac} = \frac{PRE_1 + PRE_2 + \ldots + PRE_K}{K}$$

where K is the number of classes.

- **2.7.5 ROC curve and AUC**

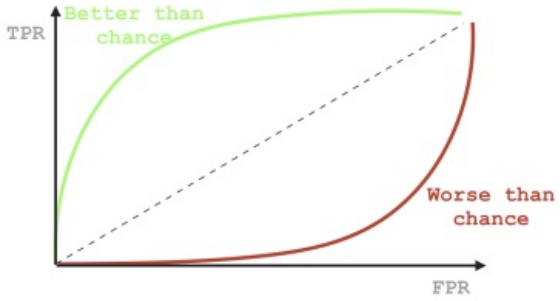
The receiver operating characteristic (ROC) curve is a common visual used to show the performance of a classification network, it plots the true positive rate (TPR) which is the recall, against the false positive rate (FPR),

$$FPR = \frac{FP}{FP + TN}$$

for different threshold values of positive prediction.

The area under the ROC curve (AUC) summaries the ROC into a single metric between 0 and 1.

$$\int_{x=0}^{1} TPR(TPR^{-1}(x))\, dx$$



ROC curve. The dashed line is if the model perfomance is equal to chance. The green line is for a network performing better than chance, and the red line is for a network performing worse than chance.

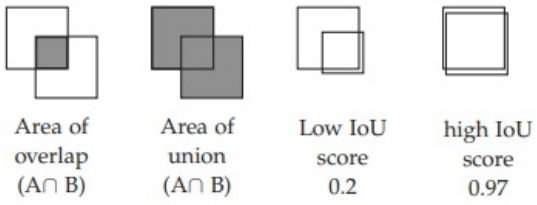
- **2.7.6 IoU and mAP**

A common metric for interpreting the perfomance of a localisation model is the intersection over union (IoU) score, which measures the area of overlap between the true bounding box A and the predicted box B.

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

The IoU score scales from 0-1 and higher values are better.

The average precision, given as the area under the precision-recall curve for various threshold value sof IoU scores is another common metric, and for multiple classes the mean average precision (mAP) can be used - the mean average precison over each class.



IoU is a metric used to quantify accuracy in image localisation tasks. Taking into account the area of overlap and the area of union.

2.8 Sparse Coding:

In deep learning, we are working in the domain of big data, the data may be very different, but the data contains structure. The purpose in machine learning is to exploit the structure in data. In general reliable models tend to be simple ones. Sparse representation is already a popular concept for priors in the signal processing community.

Sparse coding is an unsupervised learning method to learn a set of bases that can more efficiently represent data.

It reduces the complexity of neural networks with minimal decrease in performance.

In image processing each small patch in an image, (X) has a sparse representation α.

With respect to a common dictionary D such that,

$$X = D\alpha$$

D and α are redundant matrices,

since typically their combined dimensions are larger than the input.

The nature of the sparse matrix α can lead to increased efficency in CNNs.

When X and D are known, α can take on many different solutions.

We require α to take on the sparset solution - that is containing the least non-zero values.

$$min||\alpha||_0 \text{ s.t. } X = D\alpha,$$

Where $||\alpha||_2$ is the L2 norm of α and ϵ is the noise.

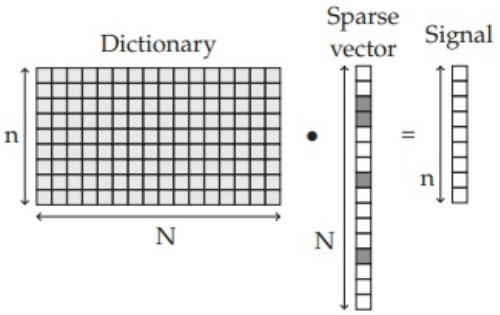
Both these equations are NP-hard - their computational cost to solve increases exponentially with the number of elements in the dictionary.

This problem of element decomposition can be solved using approximation algorithms. There are two main approaches, greedy methods - such as orthogonal matching pursuit (OMP) gradually introduces non-zeros to the sprase representation to solve:

$$min||\alpha||_0 \text{ s.t. } X = D\alpha,$$

and stops when the error falls below ϵ

and relaxation methods such as basis pursuit (BP) relaxes the L0-norm into a L1-norm which is easier to solve for.



Any image can be approximated as a linear combination of basis images, it therefore can be represented as the dot product between a dictionary of bases and a sparse matrix. There are N columns in the dictionary each representing a small image patch of size n, the matrix multiplication with the size N sparse vector produces a signal with the same size as each column.

- **2.8.1 Multi-layered convolutional sparse coding**

  Larger images (X) can be writtern as the sum of N filters (K)

  convolved with their sparse representations, where Γ_i is a feature map that

  holds the sparse representation of the ith filter.

  $$X = \sum_{i=1}^{} NK_i * \Gamma_i$$

  For the first layer in a CNN the input signal X of size I can be written in terms of the convolutional dictionary

  D_1 with dimensions $I \times IN_1$ and the locally sparse representation of

  Γ_1 of size IN_1, $X = D_1\Gamma_1$

  where locally sparse refers to the constraint:

  $$||\Gamma_1||_0 \leq N_1(2n_0 - 1),$$

  with, N_1 ,fitlers in the first layer of length, n_0

  In the 2nd convolutional layer, the input is the sparse representation of layer 1, and this too can represented in the form of

$$\underset{\_}{\Gamma}_1 = D \underset{\_}{\Gamma}_2 \underset{\_}{\Gamma}_2$$

D has dimensions, $IN_1 \times IN_2$ and $\underset{\_}{\Gamma}_2$ has dimensions IN_2.

$\underset{\_}{\Gamma}_2$ is subject to the sparsity constraint of:

$$||\underset{\_}{\Gamma}_2||_0 \leq N_2(2n_1N_1 - 1),$$

where N_2 is the number of filters in the 2nd layer and n_1 are their lengths.

This structure is cascaded through the layers of the CNN.

A M-layered sparse convolutional neural network can be written as:

$$\underset{\_}{X} = \underset{\_}{D}_1 \underset{\_}{D}_2 \ldots \underset{\_}{D}_M \underset{\_}{\Gamma}_M \quad \text{or} \quad \underset{\_}{X} = \underset{\_}{D}_{eff} \underset{\_}{\Gamma}_M$$

where the effective dictionary $\underset{\_}{D}_{eff}$ is a chain of each layer's dictionary.

- **2.8.2 Comparison to classic CNN**

The sparse vector in the first layer has solution:

$$min||\underset{\_}{\Gamma}_1||_0 \text{ s.t } \underset{\_}{\hat{X}} = \underset{\_}{D}_1 \underset{\_}{\Gamma}_1 + \underset{\_}{E}$$

where E is the noise. This can be described as solving:

$$\underset{\_}{\hat{\Gamma}} = \min_{\Gamma} \frac{1}{2} ||\underset{\_}{\hat{X}} - \underset{\_}{D}_1^T \underset{\_}{\Gamma}_1||_2^2 + \beta||\underset{\_}{\Gamma}_1||_0$$

for some scalar value of β.

A simple way to solve this is with a threshold algorithm, where the inner product of the elements in the dictionary and the signal is computed and then the lowest responses are removed via thereshold.

$$\underset{\_}{\mathcal{T}}_1(\underset{\_}{D}_1^T \underset{\_}{\hat{X}})$$

The thresholding is applied sequentially for each layer such that the 2nd layer would take the form,

$$\underset{\_}{\mathcal{T}}_2(\underset{\_}{D}_2^T \underset{\_}{\mathcal{T}}_1(\underset{\_}{D}_1^T \underset{\_}{\hat{X}}))$$

if the sparse vector is non-negative, we can use the soft non-negative threshold function

$$\mathcal{T}(\mathrm{x}) = \begin{cases} 0, & x \leq \beta \\ x, & x > \beta \end{cases}$$

therefore

$$\underset{\_}{\mathcal{T}}_2(\underset{\_}{D}_2^T \underset{\_}{\mathcal{T}}_1(\underset{\_}{D}_1^T \underset{\_}{\hat{X}}))$$

can be written as:

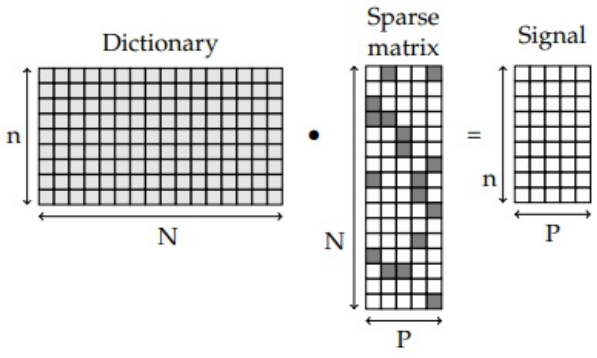
$$ReLU_2(\underset{\_}{D}_2^T ReLU_1(\underset{\_}{D}_1^T \underset{\_}{\hat{X}} + \beta_1) + \beta_2)$$

This is equivalent to a 2 layer CNN.

- **2.8.3 Learning the dictionary**

Sparse coding traditionally uses pre-defined dictionaries, or analytically derived dictionaries from known transformations (e.g. wavelet, discrete cosine transform ect.).

However another approach is to instead learn the dictionary from the data. This can be done if there are multiple signals, and hence multiple sparse representations of the data with respect to a common dictionary.



Multiple signals can be represented by a single dictionary multipled by a sparse matrix, where each column corresponds to the sparse vector representation of each column in the signal matrix

To obtain a sparse representation for all the signals we need to solve,

$$\min_{D,\Gamma} \sum_{j=1}^{P} ||X_j - D\Gamma_j||_2^2 + \beta ||\Gamma_j||_0$$

Where P is the number of signals.

The dictionary can be learned in either a supervised or unsupervised manner following these general steps:

1. Initiate the elements in the dictionary by randomly sampling them from the signal
2. Using the dictionary and signal, compute the sparse representation by cycling through each column of the sparse matrix and using a pursuit algorithm, i.e., solving:

$$\Gamma^*(X_j, D) = \min_{\Gamma} ||\Gamma_j||_0 \text{s.t.} X_j = D\Gamma_j$$

3. These sparse representations can be fed to a classifier with parameters U and the dictionary is updated by solving:

$$\min_{D,U} \sum_j l(h(X_j), U, \Gamma^*(X_j, D))$$

where l is the loss with respect to the true signal $(h(X)_j)$.

This can be solved via gradient descent in a supervised manner.

If U is of no importance, it reduces to the unsupervised form,

$$\min_{D} ||X - D\Gamma^*||_2^2$$

using K-SVD, this is solved element wise, by taking all the signals that use the given element, subtracting the contribution from other elements, and using a method such as singular value decomposition (SVD) to update the element and reduce the residuals.

4. Steps 2-3 are repeated iteratively.

2.9 Geometric deep learning:

So far we have only considered CNNs in the euclidean domain, acting on grid based, structured data.

However there are cases where the data of interest is non-eucliidean in nature. Things like trees, networks, manifolds. This kind of data is generally not structured.

2.9.1 Graph Theory

$$\text{Graphs } G = (V, E) \text{ consist of vertices } V = \{v_1, v_2, \ldots, v_n\}$$

$$\text{known as the vertex set and a set of edges } E = \{e_1 1, e_1 2, .., e_{i,j}\}$$

that link pairs of vertices together.

The number of neighbours of a node graph is known as the degree, defined as,

$$d(v_i) = \sum e_{i,j}$$

where the sum is made of neighbouring nodes.

- **2.9.2 Graph convolution**

  Graph neural networks are inspired by recurrent neural networks. They take graph data as inputs and sequentially process it through hidden layers that are equivalent to intermediate feature representations in neural networks in order to classify the vertices (nodes) or the edges (connections) according to a class label.

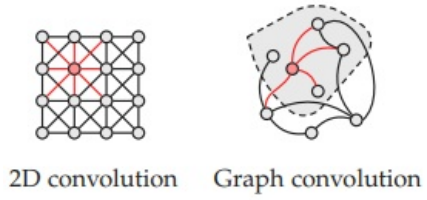
  Example of graph data would be social networks, where individuals are nodes, and interactions between people are the edges, and the classes could be their age, additionally you may havbe feature vectors for each of the nodes, such as their favourite music genre, and favourite food. If the edges represent transactions between people, the classes could be fraud and valid transcations.

  Standard neural networks would not work well on this data, since the nodes are not fixed spatially (they can move around), and the nodes that are close spatially do not necessairly connect via an edge.

  Given how large graph data can grow it may make sense to use a method analagous to convolutional neural networks - graph convolutional networks.

  In 2D convolution, the data is structured. Each pixel can be represented as a node, and the filter size determines which nodes are neighbours. The filters extract features in the local neighbourhood and aggregate neighbouring features in subsequent layers. Here the nodes are ordered and they have the same size (all pixels are equivalent).

  In graph convolution, the data is not structured. Like in 2D convolution, in order to get the representation of a node, one way is to take the average of its neighbouring nodes, however graph data is not structured. Nodes can vary in size and the filter size cannot be used to determine which nodes are neighbours.



  2D convolution represented in graph form: The circles represent nodes and the lines the connections. The data is sturctured and all neighbouring nodes, determined by the filter size are connected.

  Convolution aggregates information together of neighbouring nodes.

  In graph convolution, the nodes are not structured and can vary in size, and therefore neighbours are less well defined.

  Convolution, the method of sequentially aggregating node feature, on graph data must therefore be redefined.

  In order to generalise convolution to graph data, we need to define the parameterised filters used in the convolutional layers. There are 2 main methods to do so:

  - Spatial Methods: are problem specific architectures that aggregate features in neighbouring nodes
  - Spectral Methods: Involve eigenvalue decomposition like in Principle Component Analysis, however here, it is the smaller eignevalues that describe the structure.

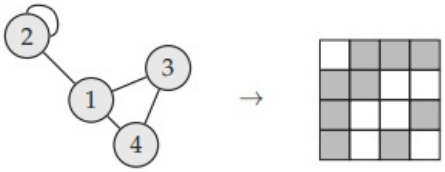
  For graph data to be used in a GCN we need to project it onto an adjacency matrix A which encodes the information about the nodes and the connections and has size NxN where N is the number of nodes in the input.

  Visualisation of some graph data with nodes and connections labelled arbitrarily. The right shows the corresponding adjacent matrix, where the grey elements are corresponding to connections and the white elements correspond to no connections. At (1,2), and (2,1) the elements of the adjacent matrix are grey showing the connections between nodes 1 and 2. At (1,1) the element is white to show there is no self connection at node 1.

  Note that the adjacency matrix has to be symmetric and so one limitation is that edges are treatedd as undirected. This restriction means that representational hierarchy can only be built by graph convolutional operation followed by pooling or expansion, whch was common for early CNN methods. This has since been replaced by strided and transpose convolutions and skip connections.

  There are ways around this by example using Bipartite graph convolutions or weighted directed graphs where each edge is represented by a 3-tuple rather than a 2-tuple.

Furthermore ajacent matrices unlike classic CNNs are not invariant to small shifts or transformations.



We have changed just ome pixel in the adjacent matrix but the structure of the graph data is now drastically different.

However regardless of how the graphs are sampled to produce the adjacent matrix, the targets will always be the same.

The input to the graph convolution layer is a feature matrix X which consists of a value for each of the F features, for each of the N nodes and therefore has size F x N. Each neural layer is a non-linear function of the adjacent and feature matrix.

$$H^{(l)} = f(H^{(l-1)}, A)$$

where l is the lth layer in the network. in the first layer $H^{(0)} = X$,

but since the features may not be known a priori, they can either be randomly initialised or X can be taken as the identity matrix.

The difference between spectral and spatial GCNs is how f(x,y) is defined and parameterised. Following in the context of CNNs, we could naively use,

$$f(H^{(l)}, A) = \phi(A H^{(l)} K^{(l)})$$

where $K^{(l)}$ is a matrix of weights of the lth layer and $\phi(x)$

is a non-linear activation function like ReLU.

The problem is that $AH^{(l)}$ sums up the feature vectors all neighbouring

nodes but not the feature vector of the node itself unless a self connection is imposed.

Therefore we replace A with $\hat{A} = A + I$ where I is the identity matrix.

Additionally for large graph data, the number of neighbours of nodes can be very large, and consequently the number of aggregated features will become increasingly large at each layer.

For this reason A is normalised by the degree matrix D, a diagonal matrix where:

$$D_{ii} = \sum_j A_{ij}$$

Since A is a symmetric matrix, to ensure the normalised matrix is also symmetirc, this is applied as:

$$D^{-1/2} A D^{-1/2}$$

thus $f(H^{(l)}, A) = \phi(A H^{(l)} K^{(l)})$ becomes:

$$f(H^{(l)}, A) = \phi(\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} H^{(l)} K^{l})$$

Just like CNNs, a simple 2 layer semi supervised GCN could look like:

$$\hat{Y} = \text{softmax}(\hat{A}\,\text{ReLU}(\hat{A} X W^{(0)}) W^{(1)})$$

In semi supervised classification, some nodes are labeled and others are not. Any loss can be used, e.g. cross entropy, however when calculating the loss only the labelled nodes are taken into account. This assumes that locally connected nodes are likely to share the same label.

Graph Downsampling:

The hierarchical structure of CNNs is largely down to the downsampling (pooling) layers.

Graph networks require first clustering the graph before the pooling layers can be applied. This is known as graph coarsening, and can be done with various algorithms.

For point cloud data there is VoxelGrid and self organising networks, for general weighted graphs there is Graculus.

The max or average pooling occurs across collapsed vertices, and this coarsening structure can be represented as a binary tree.

GCN Applications:

GCNs are not only looking for patterns in the direct connections between nodes, but also the indirect connections. The features of nodes become increasingly more abstract at each consecutive layer.

The applications are not limited to classification, but also generation of new drug candidates, detecting fraudulent bank transactions, ect.

GCNs are still an area of active reseach, graph data is difficult to store in vRAM because the adjacency matrix needs to be stored. It has been show that for large graphs, CPUs can be used without too much loss in efficiency.

- **2.9.3 Riemannian manifolds**

The graph networks mentioned so far are in a fixed domain - the connections between the nodes are fixed. There are cases where we would want to have varying domains, i.e., for 3D moving objects.

A 3D shape can be described by a topological surface called a Manifold, χ. At any point x on the surface, the local area has a Euclidean representation of χ,

$$\text{known as the tangent plane } T_x\chi.$$

The Riemannian metric is defined as the family of inner products of all tangent planes about x,

$$g_x\langle .,. \rangle_{T_x\chi} : T_x\chi \times T_x\chi \rightarrow \mathrm{R}$$

this describes the smooth local intrinsic structure at x, and allows us to measure lengths and angles. The scalar fields f describe a scalar quantitu at all points over

$$\chi, f : \chi \rightarrow \mathrm{R}$$

and the vector fields F describe a tangent vector quantity at all points over

$$\chi, F : \chi \rightarrow T\chi$$

The laplacian is important for learning on non-euclidean domains, as its eigenfunctions are used to generalise the classical fourier bases in spectral analyses. The laplacian is the divergence of the gradient of a function on euclidean space.

$$\mathrm{div}\ \vec{F}(x) = \vec{\nabla} \cdot \vec{F}(x)$$

$$\mathrm{grad}\ f(x) = \vec{\nabla} f(x)$$

$$\Delta f(x) = -\vec{\nabla} \cdot \vec{\nabla} f(x)$$

In non-Euclidean space, the laplacian uses the intrinsic gradient - the direction of greatest change in slope as represented by a tangent vector,

$$\vec{\nabla} f : L^2(\chi) \rightarrow L^2(T\chi)$$

and the intrinsic divergence - the density of outward flux of F from an infinitesimal sphere about tangent vector fields,

$$\vec{\nabla} \cdot \vec{F} : L^2(T\chi) \rightarrow L^2(\chi)$$

The laplacian is the difference between the average of a function on an infinitesimal sphere around a point and the value of the function at that point.

On a manifold the eigendecomposition of the laplacian is:

$$\Delta_\chi \phi_k = \lambda_k \phi_k$$

$$\text{where } \lambda_k \text{ are the eigenvalues and } \phi_k \text{ are the eigenvectors}$$

The non-euclidean laplacian has the propeties that it is intrinsic, so can be expressed soley in terms of the Riemannian metric, it is isometry invariant, so it doesn't change with the deformation of the manifold, and it's positive semidefinite.

A function f on χ can be written a a linear combination of the eigenvectors:

$$f(x) = \sum_{i \geq 0} \langle f, \phi_i \rangle_{L^2(\chi)} \phi_i(x)$$

$$= \sum_{i \geq 0} \hat{f}_i \phi_i(x)$$

with $\hat{f}$ serving as the coordinates of the eigenbasis.

Here $\hat{f}$ is the forward fourier transform of f

In euclidean space, the convolution of two functions in the spectral domain is the element-wise product of their fourier transforms,

$$\widehat{(f * g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$$

In non-euclidean space the operation x - x' cannot be defined on the manifold, so an equivalent transformation cannot be made, however if we use the above equation as a definition, then

$$(f * g)(x) = \sum_{k \geq 0} \langle f, \phi_k \rangle_{L^2(\chi)} \langle g, \phi_k \rangle_{L^2(\chi)} \phi_k(x)$$

spectral convolution in non-euclidean space is basis dependent.

This means that a filter applied in this domain is basis dependent and will not generalise across domains.

A basis independent, non-euclidean convolution can be obtained by construction of a local system of geodesic polar coordinates on a manifold,

$$\underset{-}{u}(x, x') = (p(x, x'), \theta(x, x'))$$

where p is the radial component that describes the geodesic distance from x to x' and θ is the angular component that describes the direction from x to x'. Additionally, their locally defined weights can be written as:

$$w_1(\underset{-}{u}(x, x')) = (v_p(x, x'), v_\theta(x, x'))$$

for a function f, the patch operator produces a local representation of f around a point x.

$$(D(x)f)\underset{-}{u} = \frac{\int_\chi v_p(x, x'), v_\theta(x, x') f(x')\, dx'}{\int_\chi v_p(x, x'), v_\theta(x, x')\, dx'}$$

The spatial convolution can then be defined as a filter g applied on local patches,

$$(f * g)(x) = \underset{p,\theta}{sum} (D(x)f)(\underset{-}{u}) g(\underset{-}{u})$$

Domain	Euclidean	Non-Euclidean
Spatial	$(f * g)(x) = \int f(x')g(x - x')dx'$	$(f * g)(x) = \int (D(x)f)(\boldsymbol{u})g(\boldsymbol{u})d\boldsymbol{u}$
Spectral	$\widehat{(f * g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$	$\widehat{(f * g)}(\omega)_k = \langle f, \phi_k \rangle_{L^2(\chi)} \langle g, \phi_k \rangle_{L^2(\chi)}$

Spatial and Spectral convolutions for Euclidean and non-Euclidean domains.

Malachy CNN example -

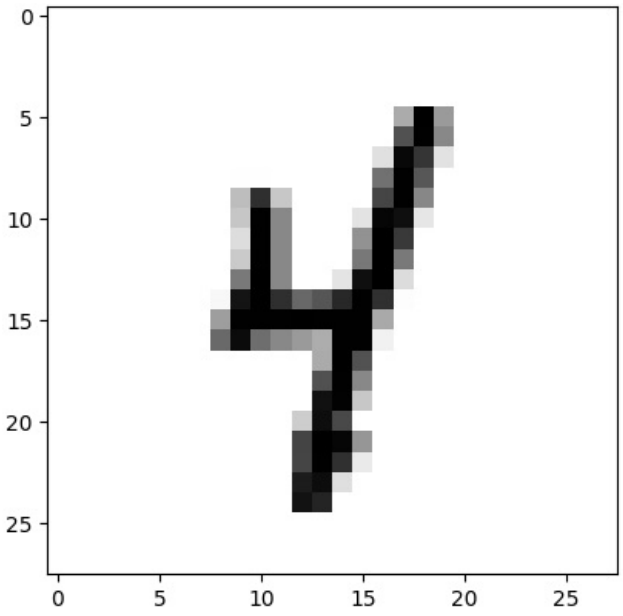
In [ ]:
```
# CNN using keras.

import tensorflow as tf
import numpy as np
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt

import subprocess

run = False # set to True to run the code below
print_confusion = False # set to True to print confusion matrix

if run:

    def print_gpu_info():

        output = subprocess.check_output(['nvidia-smi', '--query-gpu=index,name,memory.total,pci.bus_id', '--fo
        gpu_info = output.decode().strip().split('\n')

        for info in gpu_info:
            index, name, vram, pcie = info.split(',')
            print(f"Device Number: {index.strip()}, Name: {name.strip()}, VRAM: {vram.strip()}, PCIe: {pcie.str

    print_gpu_info()
```

```
# download mnist dataset
(train_ds, validation_ds, test_ds), metadata = tfds.load(

'mnist',
split=['train[:80%]', 'train[80%:100%]', 'test'],
with_info=True,
as_supervised=True
)

ntrain = len([image[0] for image in train_ds])
nvalid = len([image[0] for image in validation_ds])
ntest = len([image[0] for image in test_ds])

print('Train sample: ', ntrain)
print('Validation sample: ', nvalid)
print('Test sample: ', ntest)

# take 1 image from training dataset and convert to float64 and normalise it.

data1 = train_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64), label)
).take(1)

# get first image data and label and plot it
features, labels = iter(data1).next()
print('image 1 shape: ', np.shape(features))
plt.imshow(features[:,:,0], cmap='gray_r')
plt.show()

# ======================================== Edge Detection Example with convolution ========================

# horizontal edge detection kernal = [[1, 1, 1], [0, 0, 0], [-1, -1, -1]]
# vertical edge detection kernal = [[1, 0, -1], [1, 0, -1], [1, 0, -1]]

hor_kernal = [[1,1,1],
              [0,0,0],
              [-1,-1,-1]]

#2D convolution
output = tf.nn.conv2d(
    input=np.reshape(features, [1,28,28,1]), # batch, height, width, depth
    filters=np.reshape(hor_kernal, [3,3,1,1]), # height, width, in_channels, out_channels
    strides=[1,1,1,1], # amount to move the kernel across the input tensor
    padding="VALID"
)

print('image shape: ', np.shape(output))
plt.imshow(output[0,:,:,0], cmap='gray')
plt.show()

# NOTE: output image shape is 26x26 because of padding="VALID" and 28x28 input image size. (valid padding m
# If padding="SAME" then output image size will be same as input image size.

vert_kernal = [[1,0,-1],
               [1,0,-1],
               [1,0,-1]]

output = tf.nn.conv2d(
    input=np.reshape(features, [1,28,28,1]),
    filters=np.reshape(vert_kernal, [3,3,1,1]),
    strides=[1,1,1,1], #in batch, x, y, channel
    padding="VALID"
)

print('image shape: ', np.shape(output))
plt.imshow(output[0,:,:,0], cmap='gray')
plt.show()

# ======================================== Pooling Example with max pooling ==============================

# The pooling layer is used to reduce the spatial dimensions of the input volume. The max pooling operation
# returns the maximum value from the portion of the image covered by the kernel.
# Lets plot a 3x3 max pooling output of our image.

output = tf.nn.max_pool(
    input=np.reshape(features, [1,28,28,1]),
    ksize=3,
    strides=1,
    padding="VALID"
)

print('image shape: ', np.shape(output))
```

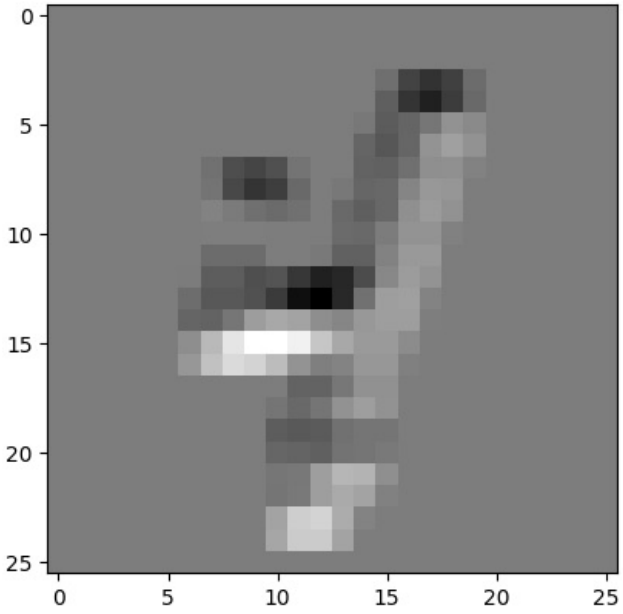
```
plt.imshow(output[0,:,:,0], cmap='gray')
plt.show()

# if we increase the kernal size to 10, we get even more shrinkage in the output image size.

output = tf.nn.max_pool(

    input=np.reshape(features, [1,28,28,1]),
    ksize=10,
    strides=1,
    padding="VALID"
)

print('image shape: ', np.shape(output))
plt.imshow(output[0,:,:,0], cmap='gray')

# ======================================== Fully connected layer example ==================================

# set a weight matrix of 1 in the 10th and 11th columns and 0 elsewhere.

weights = np.zeros([19,19])
weights[0:18,9]=1
weights[0:18,10]=1

plt.imshow(weights)
plt.show()

output2 = tf.nn.conv2d(
    input=np.reshape(output, [1,19,19,1]), #batch, height, width, depth
    filters=np.reshape(weights, [19,19,1,1]), #height, width, in_channels, out_channels
    strides=[1,1,1,1],
    padding="VALID"
)

tf.print(output2)

# ======================================== CNN model example ========================================

# First we will take make a training dataset which the model sees and uses to update model parameters,
# a validation dataset that is not used to update model parameters, but ensures that the model is not overf
# and decide when to stop training, and a test dataset that is used to evaluate the model's performance. Th
# are sampled from the original dataset.

# we apply augmentation to the training dataset to increase the number of training samples.

# NOTE : we use shuffle to shuffle buffer 100 images batch split into batches of 24 take 1 batch and repeat
# NOTE: order of operations is important. If we take a batch of 24 images and then shuffle, we will shuffle

train = train_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64), label)
).shuffle(100
).batch(24).take(100)

valid = validation_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64), label)
).shuffle(100
).batch(24).take(100)

test = test_ds.map(
    lambda image, label: (tf.image.convert_image_dtype(image, tf.float64), label)
).batch(100).take(100)

# We can now build a CNN model. Let's start with a simple model with 2 convolutional layers each with 10 3x
# 2 max pooling layers with 3x3 kernal, and 2 fully connected layers.

tf.keras.backend.clear_session() #Clear keras session

num_classes = 10
input_shape = [28, 28, 1]

# define the model
model = tf.keras.Sequential([
tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu', name='conv1',input_shape=input_shape
tf.keras.layers.MaxPool2D(pool_size=(3,3), name='pool1'),
tf.keras.layers.Conv2D(filters=10,kernel_size=(3,3),strides=1,padding='valid',activation='relu', name='conv
tf.keras.layers.MaxPool2D(pool_size=(3,3), name='pool2'),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(20, activation='relu',name='dense1'),
tf.keras.layers.Dense(num_classes, activation='softmax', name='dense2')
])

model.summary()
```

```
    LR = 0.001 # learning rate

    model.compile(
        loss=tf.keras.losses.sparse_categorical_crossentropy,
        optimizer=tf.keras.optimizers.Adam(LR),
        metrics=['accuracy']
    )

    history = model.fit(train, epochs=500, validation_data=valid)
    score = model.evaluate(test, verbose=0)

    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()


    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

    if print_confusion:
        for image, label in test:
            pred = model.predict(image)
            for idx in label:
                tf.print('predicted:', np.argmax(pred[idx]), '- truth:', label[idx])
```

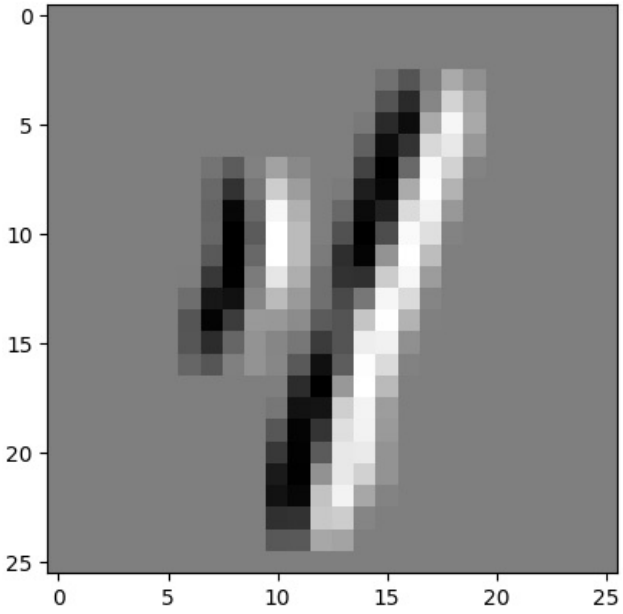
```
Device Number: 0, Name: NVIDIA GeForce RTX 2080 SUPER, VRAM: 8192 MiB, PCIe: 00000000:0A:00.0
Train sample:  48000
Validation sample:  12000
Test sample:  10000
image 1 shape:  (28, 28, 1)
```



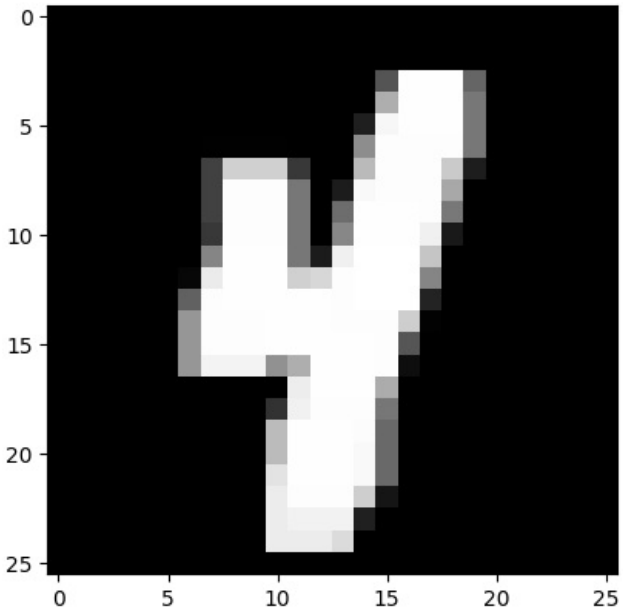

```
image shape:  (1, 26, 26, 1)
```



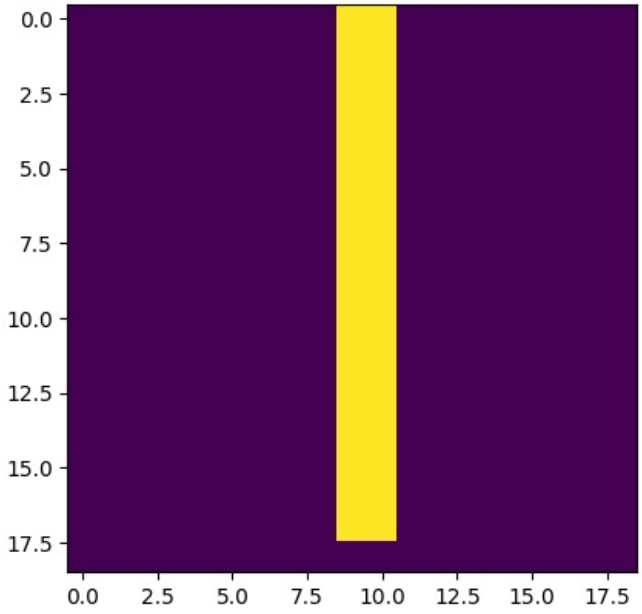

```
image shape:  (1, 26, 26, 1)
```




```
image shape:  (1, 26, 26, 1)
```




```
image shape:  (1, 19, 19, 1)
```




```
[[[[35.9372559]]]]
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1 (Conv2D)	(None, 26, 26, 32)	320
pool1 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2 (Conv2D)	(None, 6, 6, 10)	2890
pool2 (MaxPooling2D)	(None, 2, 2, 10)	0
flatten (Flatten)	(None, 40)	0
dense1 (Dense)	(None, 20)	820
dense2 (Dense)	(None, 10)	210

```
=================================================================
Total params: 4,240
Trainable params: 4,240
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
100/100 [==============================] - 1s 8ms/step - loss: 2.1863 - accuracy: 0.2000 - val_loss: 1.9191 - va
l_accuracy: 0.3779
Epoch 2/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2975 - accuracy: 0.6117 - val_loss: 0.8258 - va
l_accuracy: 0.7496
Epoch 3/500
100/100 [==============================] - 1s 6ms/step - loss: 0.6000 - accuracy: 0.8213 - val_loss: 0.5641 - va
l_accuracy: 0.8138
Epoch 4/500
100/100 [==============================] - 1s 7ms/step - loss: 0.4198 - accuracy: 0.8692 - val_loss: 0.4604 - va
l_accuracy: 0.8517
Epoch 5/500
100/100 [==============================] - 1s 6ms/step - loss: 0.3504 - accuracy: 0.8913 - val_loss: 0.3929 - va
l_accuracy: 0.8733
Epoch 6/500
100/100 [==============================] - 1s 6ms/step - loss: 0.2920 - accuracy: 0.9096 - val_loss: 0.3722 - va
l_accuracy: 0.8842
Epoch 7/500
100/100 [==============================] - 1s 6ms/step - loss: 0.2550 - accuracy: 0.9187 - val_loss: 0.3341 - va
l_accuracy: 0.8975
Epoch 8/500
100/100 [==============================] - 1s 6ms/step - loss: 0.2241 - accuracy: 0.9275 - val_loss: 0.3109 - va
l_accuracy: 0.9029
Epoch 9/500
100/100 [==============================] - 1s 7ms/step - loss: 0.1951 - accuracy: 0.9379 - val_loss: 0.3027 - va
l_accuracy: 0.9046
Epoch 10/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1815 - accuracy: 0.9458 - val_loss: 0.2824 - va
l_accuracy: 0.9129
Epoch 11/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1662 - accuracy: 0.9475 - val_loss: 0.2800 - va
l_accuracy: 0.9137
Epoch 12/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1472 - accuracy: 0.9600 - val_loss: 0.2635 - va
l_accuracy: 0.9208
Epoch 13/500
100/100 [==============================] - 1s 7ms/step - loss: 0.1450 - accuracy: 0.9529 - val_loss: 0.2580 - va
l_accuracy: 0.9246
Epoch 14/500
100/100 [==============================] - 1s 7ms/step - loss: 0.1321 - accuracy: 0.9633 - val_loss: 0.2537 - va
l_accuracy: 0.9258
Epoch 15/500
100/100 [==============================] - 1s 7ms/step - loss: 0.1127 - accuracy: 0.9700 - val_loss: 0.2534 - va
l_accuracy: 0.9262
Epoch 16/500
100/100 [==============================] - 1s 7ms/step - loss: 0.1118 - accuracy: 0.9679 - val_loss: 0.2488 - va
l_accuracy: 0.9246
Epoch 17/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1038 - accuracy: 0.9721 - val_loss: 0.2482 - va
l_accuracy: 0.9283
Epoch 18/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0925 - accuracy: 0.9729 - val_loss: 0.2520 - va
l_accuracy: 0.9262
Epoch 19/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0927 - accuracy: 0.9750 - val_loss: 0.2347 - va
l_accuracy: 0.9308
Epoch 20/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0851 - accuracy: 0.9767 - val_loss: 0.2349 - va
l_accuracy: 0.9308
Epoch 21/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0762 - accuracy: 0.9783 - val_loss: 0.2566 - va
l_accuracy: 0.9254
Epoch 22/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0785 - accuracy: 0.9758 - val_loss: 0.2500 - va
l_accuracy: 0.9258
Epoch 23/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0645 - accuracy: 0.9854 - val_loss: 0.2519 - va
l_accuracy: 0.9262
Epoch 24/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0651 - accuracy: 0.9833 - val_loss: 0.2384 - va
l_accuracy: 0.9325
Epoch 25/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0584 - accuracy: 0.9867 - val_loss: 0.2340 - va
l_accuracy: 0.9325
Epoch 26/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0524 - accuracy: 0.9862 - val_loss: 0.2476 - va
l_accuracy: 0.9279
```

```
Epoch 27/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0506 - accuracy: 0.9879 - val_loss: 0.2292 - va
l_accuracy: 0.9350
Epoch 28/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0481 - accuracy: 0.9871 - val_loss: 0.2208 - va
l_accuracy: 0.9367
Epoch 29/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0538 - accuracy: 0.9837 - val_loss: 0.2383 - va
l_accuracy: 0.9350
Epoch 30/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0422 - accuracy: 0.9887 - val_loss: 0.2208 - va
l_accuracy: 0.9413
Epoch 31/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0355 - accuracy: 0.9917 - val_loss: 0.2234 - va
l_accuracy: 0.9404
Epoch 32/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0347 - accuracy: 0.9929 - val_loss: 0.2378 - va
l_accuracy: 0.9333
Epoch 33/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0397 - accuracy: 0.9887 - val_loss: 0.2497 - va
l_accuracy: 0.9342
Epoch 34/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0342 - accuracy: 0.9921 - val_loss: 0.2260 - va
l_accuracy: 0.9417
Epoch 35/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0276 - accuracy: 0.9958 - val_loss: 0.2544 - va
l_accuracy: 0.9329
Epoch 36/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0331 - accuracy: 0.9933 - val_loss: 0.2408 - va
l_accuracy: 0.9375
Epoch 37/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0246 - accuracy: 0.9937 - val_loss: 0.2351 - va
l_accuracy: 0.9404
Epoch 38/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0251 - accuracy: 0.9950 - val_loss: 0.2301 - va
l_accuracy: 0.9413
Epoch 39/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0220 - accuracy: 0.9954 - val_loss: 0.2337 - va
l_accuracy: 0.9392
Epoch 40/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0235 - accuracy: 0.9967 - val_loss: 0.2319 - va
l_accuracy: 0.9417
Epoch 41/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0221 - accuracy: 0.9950 - val_loss: 0.2578 - va
l_accuracy: 0.9362
Epoch 42/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0204 - accuracy: 0.9962 - val_loss: 0.2369 - va
l_accuracy: 0.9417
Epoch 43/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0137 - accuracy: 0.9987 - val_loss: 0.2381 - va
l_accuracy: 0.9392
Epoch 44/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0144 - accuracy: 0.9992 - val_loss: 0.2361 - va
l_accuracy: 0.9400
Epoch 45/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0143 - accuracy: 0.9979 - val_loss: 0.2369 - va
l_accuracy: 0.9442
Epoch 46/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0124 - accuracy: 0.9996 - val_loss: 0.2501 - va
l_accuracy: 0.9413
Epoch 47/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0115 - accuracy: 0.9996 - val_loss: 0.2330 - va
l_accuracy: 0.9433
Epoch 48/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0115 - accuracy: 1.0000 - val_loss: 0.2495 - va
l_accuracy: 0.9396
Epoch 49/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0091 - accuracy: 0.9996 - val_loss: 0.2528 - va
l_accuracy: 0.9400
Epoch 50/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0100 - accuracy: 0.9992 - val_loss: 0.2477 - va
l_accuracy: 0.9413
Epoch 51/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0095 - accuracy: 0.9996 - val_loss: 0.2449 - va
l_accuracy: 0.9425
Epoch 52/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0128 - accuracy: 0.9983 - val_loss: 0.2550 - va
l_accuracy: 0.9433
Epoch 53/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0098 - accuracy: 0.9992 - val_loss: 0.2664 - va
l_accuracy: 0.9404
Epoch 54/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0112 - accuracy: 0.9979 - val_loss: 0.2462 - va
```

```
l_accuracy: 0.9450
Epoch 55/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0077 - accuracy: 1.0000 - val_loss: 0.2491 - va
l_accuracy: 0.9446
Epoch 56/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0057 - accuracy: 1.0000 - val_loss: 0.2717 - va
l_accuracy: 0.9388
Epoch 57/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0060 - accuracy: 1.0000 - val_loss: 0.2515 - va
l_accuracy: 0.9446
Epoch 58/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0049 - accuracy: 1.0000 - val_loss: 0.2636 - va
l_accuracy: 0.9413
Epoch 59/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0050 - accuracy: 1.0000 - val_loss: 0.2537 - va
l_accuracy: 0.9425
Epoch 60/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0054 - accuracy: 0.9996 - val_loss: 0.2662 - va
l_accuracy: 0.9413
Epoch 61/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0147 - accuracy: 0.9967 - val_loss: 0.3253 - va
l_accuracy: 0.9283
Epoch 62/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0284 - accuracy: 0.9879 - val_loss: 0.2715 - va
l_accuracy: 0.9367
Epoch 63/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0130 - accuracy: 0.9971 - val_loss: 0.2536 - va
l_accuracy: 0.9425
Epoch 64/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.2915 - va
l_accuracy: 0.9354
Epoch 65/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0164 - accuracy: 0.9962 - val_loss: 0.2904 - va
l_accuracy: 0.9342
Epoch 66/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0059 - accuracy: 0.9992 - val_loss: 0.2416 - va
l_accuracy: 0.9433
Epoch 67/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.2638 - va
l_accuracy: 0.9421
Epoch 68/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0032 - accuracy: 0.9996 - val_loss: 0.2557 - va
l_accuracy: 0.9429
Epoch 69/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.2614 - va
l_accuracy: 0.9446
Epoch 70/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.2638 - va
l_accuracy: 0.9438
Epoch 71/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.2494 - va
l_accuracy: 0.9454
Epoch 72/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.2656 - va
l_accuracy: 0.9433
Epoch 73/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.2531 - va
l_accuracy: 0.9446
Epoch 74/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.2617 - va
l_accuracy: 0.9433
Epoch 75/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.2583 - va
l_accuracy: 0.9442
Epoch 76/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0023 - accuracy: 0.9996 - val_loss: 0.2603 - va
l_accuracy: 0.9438
Epoch 77/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0367 - accuracy: 0.9854 - val_loss: 0.3423 - va
l_accuracy: 0.9325
Epoch 78/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0251 - accuracy: 0.9904 - val_loss: 0.2705 - va
l_accuracy: 0.9404
Epoch 79/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0043 - accuracy: 0.9992 - val_loss: 0.3110 - va
l_accuracy: 0.9350
Epoch 80/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0029 - accuracy: 0.9996 - val_loss: 0.2798 - va
l_accuracy: 0.9392
Epoch 81/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.2722 - va
l_accuracy: 0.9408
Epoch 82/500
```

```
100/100 [==============================] - 1s 6ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.2745 - va
l_accuracy: 0.9417
Epoch 83/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.2744 - va
l_accuracy: 0.9421
Epoch 84/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.2771 - va
l_accuracy: 0.9417
Epoch 85/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.2729 - va
l_accuracy: 0.9429
Epoch 86/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.2739 - va
l_accuracy: 0.9438
Epoch 87/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.2724 - va
l_accuracy: 0.9429
Epoch 88/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.2694 - va
l_accuracy: 0.9454
Epoch 89/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.2730 - va
l_accuracy: 0.9442
Epoch 90/500
100/100 [==============================] - 1s 6ms/step - loss: 9.1645e-04 - accuracy: 1.0000 - val_loss: 0.2721
- val_accuracy: 0.9433
Epoch 91/500
100/100 [==============================] - 1s 7ms/step - loss: 8.4294e-04 - accuracy: 1.0000 - val_loss: 0.2712
- val_accuracy: 0.9450
Epoch 92/500
100/100 [==============================] - 1s 6ms/step - loss: 7.9450e-04 - accuracy: 1.0000 - val_loss: 0.2639
- val_accuracy: 0.9442
Epoch 93/500
100/100 [==============================] - 1s 6ms/step - loss: 8.2032e-04 - accuracy: 1.0000 - val_loss: 0.2701
- val_accuracy: 0.9446
Epoch 94/500
100/100 [==============================] - 1s 7ms/step - loss: 7.7809e-04 - accuracy: 1.0000 - val_loss: 0.2664
- val_accuracy: 0.9450
Epoch 95/500
100/100 [==============================] - 1s 7ms/step - loss: 6.9670e-04 - accuracy: 1.0000 - val_loss: 0.2710
- val_accuracy: 0.9458
Epoch 96/500
100/100 [==============================] - 1s 7ms/step - loss: 6.7202e-04 - accuracy: 1.0000 - val_loss: 0.2708
- val_accuracy: 0.9450
Epoch 97/500
100/100 [==============================] - 1s 6ms/step - loss: 6.3888e-04 - accuracy: 1.0000 - val_loss: 0.2779
- val_accuracy: 0.9438
Epoch 98/500
100/100 [==============================] - 1s 7ms/step - loss: 5.9592e-04 - accuracy: 1.0000 - val_loss: 0.2786
- val_accuracy: 0.9454
Epoch 99/500
100/100 [==============================] - 1s 6ms/step - loss: 5.9664e-04 - accuracy: 1.0000 - val_loss: 0.2809
- val_accuracy: 0.9446
Epoch 100/500
100/100 [==============================] - 1s 6ms/step - loss: 5.7849e-04 - accuracy: 1.0000 - val_loss: 0.2761
- val_accuracy: 0.9438
Epoch 101/500
100/100 [==============================] - 1s 6ms/step - loss: 5.2219e-04 - accuracy: 1.0000 - val_loss: 0.2782
- val_accuracy: 0.9458
Epoch 102/500
100/100 [==============================] - 1s 6ms/step - loss: 5.2470e-04 - accuracy: 1.0000 - val_loss: 0.2737
- val_accuracy: 0.9458
Epoch 103/500
100/100 [==============================] - 1s 6ms/step - loss: 4.6669e-04 - accuracy: 1.0000 - val_loss: 0.2825
- val_accuracy: 0.9454
Epoch 104/500
100/100 [==============================] - 1s 6ms/step - loss: 4.6664e-04 - accuracy: 1.0000 - val_loss: 0.2761
- val_accuracy: 0.9458
Epoch 105/500
100/100 [==============================] - 1s 6ms/step - loss: 4.4954e-04 - accuracy: 1.0000 - val_loss: 0.2865
- val_accuracy: 0.9446
Epoch 106/500
100/100 [==============================] - 1s 6ms/step - loss: 4.5953e-04 - accuracy: 1.0000 - val_loss: 0.2824
- val_accuracy: 0.9458
Epoch 107/500
100/100 [==============================] - 1s 6ms/step - loss: 4.0636e-04 - accuracy: 1.0000 - val_loss: 0.2814
- val_accuracy: 0.9450
Epoch 108/500
100/100 [==============================] - 1s 6ms/step - loss: 4.7950e-04 - accuracy: 1.0000 - val_loss: 0.2801
- val_accuracy: 0.9467
Epoch 109/500
100/100 [==============================] - 1s 7ms/step - loss: 6.5160e-04 - accuracy: 1.0000 - val_loss: 0.2805
- val_accuracy: 0.9475
```

```
Epoch 110/500
100/100 [==============================] - 1s 7ms/step - loss: 3.5486e-04 - accuracy: 1.0000 - val_loss: 0.2820
- val_accuracy: 0.9471
Epoch 111/500
100/100 [==============================] - 1s 8ms/step - loss: 3.3259e-04 - accuracy: 1.0000 - val_loss: 0.2719
- val_accuracy: 0.9463
Epoch 112/500
100/100 [==============================] - 1s 6ms/step - loss: 3.2407e-04 - accuracy: 1.0000 - val_loss: 0.2723
- val_accuracy: 0.9467
Epoch 113/500
100/100 [==============================] - 1s 6ms/step - loss: 3.0373e-04 - accuracy: 1.0000 - val_loss: 0.2900
- val_accuracy: 0.9463
Epoch 114/500
100/100 [==============================] - 1s 7ms/step - loss: 2.8991e-04 - accuracy: 1.0000 - val_loss: 0.2867
- val_accuracy: 0.9475
Epoch 115/500
100/100 [==============================] - 1s 7ms/step - loss: 2.8436e-04 - accuracy: 1.0000 - val_loss: 0.2880
- val_accuracy: 0.9488
Epoch 116/500
100/100 [==============================] - 1s 7ms/step - loss: 2.7799e-04 - accuracy: 1.0000 - val_loss: 0.2900
- val_accuracy: 0.9475
Epoch 117/500
100/100 [==============================] - 1s 7ms/step - loss: 2.5827e-04 - accuracy: 1.0000 - val_loss: 0.2912
- val_accuracy: 0.9479
Epoch 118/500
100/100 [==============================] - 1s 6ms/step - loss: 2.4407e-04 - accuracy: 1.0000 - val_loss: 0.2901
- val_accuracy: 0.9450
Epoch 119/500
100/100 [==============================] - 1s 6ms/step - loss: 2.3021e-04 - accuracy: 1.0000 - val_loss: 0.2855
- val_accuracy: 0.9471
Epoch 120/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1506e-04 - accuracy: 1.0000 - val_loss: 0.2835
- val_accuracy: 0.9475
Epoch 121/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1454e-04 - accuracy: 1.0000 - val_loss: 0.2947
- val_accuracy: 0.9463
Epoch 122/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0217e-04 - accuracy: 1.0000 - val_loss: 0.2865
- val_accuracy: 0.9475
Epoch 123/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9711e-04 - accuracy: 1.0000 - val_loss: 0.2957
- val_accuracy: 0.9463
Epoch 124/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0825e-04 - accuracy: 1.0000 - val_loss: 0.3015
- val_accuracy: 0.9483
Epoch 125/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8035e-04 - accuracy: 1.0000 - val_loss: 0.3011
- val_accuracy: 0.9467
Epoch 126/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6892e-04 - accuracy: 1.0000 - val_loss: 0.2998
- val_accuracy: 0.9463
Epoch 127/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6315e-04 - accuracy: 1.0000 - val_loss: 0.3024
- val_accuracy: 0.9467
Epoch 128/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4819e-04 - accuracy: 1.0000 - val_loss: 0.3015
- val_accuracy: 0.9475
Epoch 129/500
100/100 [==============================] - 1s 7ms/step - loss: 1.4295e-04 - accuracy: 1.0000 - val_loss: 0.3067
- val_accuracy: 0.9467
Epoch 130/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4602e-04 - accuracy: 1.0000 - val_loss: 0.3042
- val_accuracy: 0.9463
Epoch 131/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2369e-04 - accuracy: 1.0000 - val_loss: 0.3104
- val_accuracy: 0.9467
Epoch 132/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2673e-04 - accuracy: 1.0000 - val_loss: 0.3063
- val_accuracy: 0.9475
Epoch 133/500
100/100 [==============================] - 1s 7ms/step - loss: 1.1692e-04 - accuracy: 1.0000 - val_loss: 0.2929
- val_accuracy: 0.9496
Epoch 134/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0897e-04 - accuracy: 1.0000 - val_loss: 0.3144
- val_accuracy: 0.9467
Epoch 135/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0004e-04 - accuracy: 1.0000 - val_loss: 0.3084
- val_accuracy: 0.9483
Epoch 136/500
100/100 [==============================] - 1s 7ms/step - loss: 9.6404e-05 - accuracy: 1.0000 - val_loss: 0.3015
- val_accuracy: 0.9496
Epoch 137/500
100/100 [==============================] - 1s 6ms/step - loss: 8.8942e-05 - accuracy: 1.0000 - val_loss: 0.3142
```

```
- val_accuracy: 0.9475
Epoch 138/500
100/100 [==============================] - 1s 6ms/step - loss: 8.7162e-05 - accuracy: 1.0000 - val_loss: 0.3204
- val_accuracy: 0.9488
Epoch 139/500
100/100 [==============================] - 1s 6ms/step - loss: 8.4901e-05 - accuracy: 1.0000 - val_loss: 0.3158
- val_accuracy: 0.9479
Epoch 140/500
100/100 [==============================] - 1s 6ms/step - loss: 8.2380e-05 - accuracy: 1.0000 - val_loss: 0.3266
- val_accuracy: 0.9479
Epoch 141/500
100/100 [==============================] - 1s 6ms/step - loss: 7.3103e-05 - accuracy: 1.0000 - val_loss: 0.3237
- val_accuracy: 0.9471
Epoch 142/500
100/100 [==============================] - 1s 6ms/step - loss: 6.9070e-05 - accuracy: 1.0000 - val_loss: 0.3226
- val_accuracy: 0.9471
Epoch 143/500
100/100 [==============================] - 1s 6ms/step - loss: 6.8905e-05 - accuracy: 1.0000 - val_loss: 0.3259
- val_accuracy: 0.9479
Epoch 144/500
100/100 [==============================] - 1s 6ms/step - loss: 7.1364e-05 - accuracy: 1.0000 - val_loss: 0.3279
- val_accuracy: 0.9475
Epoch 145/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1372 - accuracy: 0.9658 - val_loss: 0.4165 - va
l_accuracy: 0.9300
Epoch 146/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0276 - accuracy: 0.9904 - val_loss: 0.3296 - va
l_accuracy: 0.9450
Epoch 147/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.2954 - va
l_accuracy: 0.9442
Epoch 148/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.9996 - val_loss: 0.2955 - va
l_accuracy: 0.9467
Epoch 149/500
100/100 [==============================] - 1s 6ms/step - loss: 4.7709e-04 - accuracy: 1.0000 - val_loss: 0.2881
- val_accuracy: 0.9463
Epoch 150/500
100/100 [==============================] - 1s 6ms/step - loss: 4.1246e-04 - accuracy: 1.0000 - val_loss: 0.2980
- val_accuracy: 0.9467
Epoch 151/500
100/100 [==============================] - 1s 6ms/step - loss: 3.6611e-04 - accuracy: 1.0000 - val_loss: 0.3063
- val_accuracy: 0.9467
Epoch 152/500
100/100 [==============================] - 1s 6ms/step - loss: 3.1649e-04 - accuracy: 1.0000 - val_loss: 0.2987
- val_accuracy: 0.9492
Epoch 153/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9935e-04 - accuracy: 1.0000 - val_loss: 0.3025
- val_accuracy: 0.9475
Epoch 154/500
100/100 [==============================] - 1s 6ms/step - loss: 2.7315e-04 - accuracy: 1.0000 - val_loss: 0.2999
- val_accuracy: 0.9488
Epoch 155/500
100/100 [==============================] - 1s 6ms/step - loss: 2.5843e-04 - accuracy: 1.0000 - val_loss: 0.2931
- val_accuracy: 0.9488
Epoch 156/500
100/100 [==============================] - 1s 6ms/step - loss: 2.8192e-04 - accuracy: 1.0000 - val_loss: 0.2980
- val_accuracy: 0.9496
Epoch 157/500
100/100 [==============================] - 1s 6ms/step - loss: 2.4061e-04 - accuracy: 1.0000 - val_loss: 0.3083
- val_accuracy: 0.9500
Epoch 158/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1454e-04 - accuracy: 1.0000 - val_loss: 0.3036
- val_accuracy: 0.9496
Epoch 159/500
100/100 [==============================] - 1s 7ms/step - loss: 1.9435e-04 - accuracy: 1.0000 - val_loss: 0.3110
- val_accuracy: 0.9488
Epoch 160/500
100/100 [==============================] - 1s 7ms/step - loss: 1.8265e-04 - accuracy: 1.0000 - val_loss: 0.3107
- val_accuracy: 0.9471
Epoch 161/500
100/100 [==============================] - 1s 7ms/step - loss: 1.7656e-04 - accuracy: 1.0000 - val_loss: 0.3083
- val_accuracy: 0.9467
Epoch 162/500
100/100 [==============================] - 1s 7ms/step - loss: 1.7427e-04 - accuracy: 1.0000 - val_loss: 0.3066
- val_accuracy: 0.9471
Epoch 163/500
100/100 [==============================] - 1s 7ms/step - loss: 1.6442e-04 - accuracy: 1.0000 - val_loss: 0.3128
- val_accuracy: 0.9467
Epoch 164/500
100/100 [==============================] - 1s 8ms/step - loss: 1.7163e-04 - accuracy: 1.0000 - val_loss: 0.3097
- val_accuracy: 0.9488
Epoch 165/500
```

```
100/100 [==============================] - 1s 7ms/step - loss: 1.5668e-04 - accuracy: 1.0000 - val_loss: 0.3081
- val_accuracy: 0.9492
Epoch 166/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4572e-04 - accuracy: 1.0000 - val_loss: 0.3107
- val_accuracy: 0.9483
Epoch 167/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4132e-04 - accuracy: 1.0000 - val_loss: 0.3000
- val_accuracy: 0.9500
Epoch 168/500
100/100 [==============================] - 1s 7ms/step - loss: 1.4102e-04 - accuracy: 1.0000 - val_loss: 0.2942
- val_accuracy: 0.9488
Epoch 169/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3875e-04 - accuracy: 1.0000 - val_loss: 0.3087
- val_accuracy: 0.9483
Epoch 170/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2788e-04 - accuracy: 1.0000 - val_loss: 0.3168
- val_accuracy: 0.9471
Epoch 171/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2074e-04 - accuracy: 1.0000 - val_loss: 0.3137
- val_accuracy: 0.9475
Epoch 172/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1393e-04 - accuracy: 1.0000 - val_loss: 0.3122
- val_accuracy: 0.9475
Epoch 173/500
100/100 [==============================] - 1s 7ms/step - loss: 1.1582e-04 - accuracy: 1.0000 - val_loss: 0.3098
- val_accuracy: 0.9471
Epoch 174/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1047e-04 - accuracy: 1.0000 - val_loss: 0.2992
- val_accuracy: 0.9475
Epoch 175/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0262e-04 - accuracy: 1.0000 - val_loss: 0.3137
- val_accuracy: 0.9467
Epoch 176/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0079e-04 - accuracy: 1.0000 - val_loss: 0.3170
- val_accuracy: 0.9471
Epoch 177/500
100/100 [==============================] - 1s 7ms/step - loss: 9.9464e-05 - accuracy: 1.0000 - val_loss: 0.3006
- val_accuracy: 0.9479
Epoch 178/500
100/100 [==============================] - 1s 7ms/step - loss: 9.5143e-05 - accuracy: 1.0000 - val_loss: 0.3062
- val_accuracy: 0.9488
Epoch 179/500
100/100 [==============================] - 1s 6ms/step - loss: 9.1056e-05 - accuracy: 1.0000 - val_loss: 0.3128
- val_accuracy: 0.9467
Epoch 180/500
100/100 [==============================] - 1s 6ms/step - loss: 8.9088e-05 - accuracy: 1.0000 - val_loss: 0.3128
- val_accuracy: 0.9475
Epoch 181/500
100/100 [==============================] - 1s 6ms/step - loss: 8.5750e-05 - accuracy: 1.0000 - val_loss: 0.3156
- val_accuracy: 0.9467
Epoch 182/500
100/100 [==============================] - 1s 6ms/step - loss: 8.3247e-05 - accuracy: 1.0000 - val_loss: 0.3202
- val_accuracy: 0.9471
Epoch 183/500
100/100 [==============================] - 1s 6ms/step - loss: 8.0728e-05 - accuracy: 1.0000 - val_loss: 0.3175
- val_accuracy: 0.9483
Epoch 184/500
100/100 [==============================] - 1s 6ms/step - loss: 7.8178e-05 - accuracy: 1.0000 - val_loss: 0.3174
- val_accuracy: 0.9483
Epoch 185/500
100/100 [==============================] - 1s 6ms/step - loss: 7.6327e-05 - accuracy: 1.0000 - val_loss: 0.3196
- val_accuracy: 0.9483
Epoch 186/500
100/100 [==============================] - 1s 6ms/step - loss: 7.1277e-05 - accuracy: 1.0000 - val_loss: 0.3213
- val_accuracy: 0.9483
Epoch 187/500
100/100 [==============================] - 1s 7ms/step - loss: 7.0812e-05 - accuracy: 1.0000 - val_loss: 0.3136
- val_accuracy: 0.9483
Epoch 188/500
100/100 [==============================] - 1s 7ms/step - loss: 6.7125e-05 - accuracy: 1.0000 - val_loss: 0.3015
- val_accuracy: 0.9492
Epoch 189/500
100/100 [==============================] - 1s 7ms/step - loss: 6.5791e-05 - accuracy: 1.0000 - val_loss: 0.3130
- val_accuracy: 0.9479
Epoch 190/500
100/100 [==============================] - 1s 6ms/step - loss: 6.6763e-05 - accuracy: 1.0000 - val_loss: 0.3157
- val_accuracy: 0.9479
Epoch 191/500
100/100 [==============================] - 1s 6ms/step - loss: 6.4545e-05 - accuracy: 1.0000 - val_loss: 0.3183
- val_accuracy: 0.9471
Epoch 192/500
100/100 [==============================] - 1s 6ms/step - loss: 6.0741e-05 - accuracy: 1.0000 - val_loss: 0.3195
- val_accuracy: 0.9483
```

```
Epoch 193/500
100/100 [==============================] - 1s 6ms/step - loss: 5.8391e-05 - accuracy: 1.0000 - val_loss: 0.3144
- val_accuracy: 0.9488
Epoch 194/500
100/100 [==============================] - 1s 7ms/step - loss: 5.6561e-05 - accuracy: 1.0000 - val_loss: 0.3149
- val_accuracy: 0.9483
Epoch 195/500
100/100 [==============================] - 1s 6ms/step - loss: 5.3928e-05 - accuracy: 1.0000 - val_loss: 0.3212
- val_accuracy: 0.9483
Epoch 196/500
100/100 [==============================] - 1s 7ms/step - loss: 5.7423e-05 - accuracy: 1.0000 - val_loss: 0.3175
- val_accuracy: 0.9488
Epoch 197/500
100/100 [==============================] - 1s 6ms/step - loss: 5.4774e-05 - accuracy: 1.0000 - val_loss: 0.3028
- val_accuracy: 0.9500
Epoch 198/500
100/100 [==============================] - 1s 6ms/step - loss: 5.0948e-05 - accuracy: 1.0000 - val_loss: 0.3053
- val_accuracy: 0.9496
Epoch 199/500
100/100 [==============================] - 1s 7ms/step - loss: 4.8631e-05 - accuracy: 1.0000 - val_loss: 0.3160
- val_accuracy: 0.9488
Epoch 200/500
100/100 [==============================] - 1s 6ms/step - loss: 4.6751e-05 - accuracy: 1.0000 - val_loss: 0.3192
- val_accuracy: 0.9479
Epoch 201/500
100/100 [==============================] - 1s 7ms/step - loss: 4.6332e-05 - accuracy: 1.0000 - val_loss: 0.3192
- val_accuracy: 0.9483
Epoch 202/500
100/100 [==============================] - 1s 7ms/step - loss: 4.5553e-05 - accuracy: 1.0000 - val_loss: 0.3168
- val_accuracy: 0.9496
Epoch 203/500
100/100 [==============================] - 1s 7ms/step - loss: 4.4187e-05 - accuracy: 1.0000 - val_loss: 0.3226
- val_accuracy: 0.9483
Epoch 204/500
100/100 [==============================] - 1s 7ms/step - loss: 4.1645e-05 - accuracy: 1.0000 - val_loss: 0.3154
- val_accuracy: 0.9496
Epoch 205/500
100/100 [==============================] - 1s 7ms/step - loss: 4.0620e-05 - accuracy: 1.0000 - val_loss: 0.3228
- val_accuracy: 0.9483
Epoch 206/500
100/100 [==============================] - 1s 7ms/step - loss: 4.1500e-05 - accuracy: 1.0000 - val_loss: 0.3154
- val_accuracy: 0.9492
Epoch 207/500
100/100 [==============================] - 1s 7ms/step - loss: 4.0427e-05 - accuracy: 1.0000 - val_loss: 0.3198
- val_accuracy: 0.9504
Epoch 208/500
100/100 [==============================] - 1s 8ms/step - loss: 3.7282e-05 - accuracy: 1.0000 - val_loss: 0.3116
- val_accuracy: 0.9492
Epoch 209/500
100/100 [==============================] - 1s 6ms/step - loss: 3.5680e-05 - accuracy: 1.0000 - val_loss: 0.3234
- val_accuracy: 0.9492
Epoch 210/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3839e-05 - accuracy: 1.0000 - val_loss: 0.3272
- val_accuracy: 0.9483
Epoch 211/500
100/100 [==============================] - 1s 7ms/step - loss: 3.3608e-05 - accuracy: 1.0000 - val_loss: 0.3218
- val_accuracy: 0.9492
Epoch 212/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3235e-05 - accuracy: 1.0000 - val_loss: 0.3220
- val_accuracy: 0.9504
Epoch 213/500
100/100 [==============================] - 1s 7ms/step - loss: 3.0676e-05 - accuracy: 1.0000 - val_loss: 0.3303
- val_accuracy: 0.9483
Epoch 214/500
100/100 [==============================] - 1s 6ms/step - loss: 3.0091e-05 - accuracy: 1.0000 - val_loss: 0.3260
- val_accuracy: 0.9496
Epoch 215/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9168e-05 - accuracy: 1.0000 - val_loss: 0.3274
- val_accuracy: 0.9496
Epoch 216/500
100/100 [==============================] - 1s 6ms/step - loss: 2.8439e-05 - accuracy: 1.0000 - val_loss: 0.3289
- val_accuracy: 0.9492
Epoch 217/500
100/100 [==============================] - 1s 6ms/step - loss: 2.6367e-05 - accuracy: 1.0000 - val_loss: 0.3146
- val_accuracy: 0.9492
Epoch 218/500
100/100 [==============================] - 1s 6ms/step - loss: 2.5220e-05 - accuracy: 1.0000 - val_loss: 0.3232
- val_accuracy: 0.9492
Epoch 219/500
100/100 [==============================] - 1s 7ms/step - loss: 2.5213e-05 - accuracy: 1.0000 - val_loss: 0.3389
- val_accuracy: 0.9492
Epoch 220/500
100/100 [==============================] - 1s 8ms/step - loss: 2.4261e-05 - accuracy: 1.0000 - val_loss: 0.3300
```

```
- val_accuracy: 0.9483
Epoch 221/500
100/100 [==============================] - 1s 6ms/step - loss: 2.3383e-05 - accuracy: 1.0000 - val_loss: 0.3143
- val_accuracy: 0.9504
Epoch 222/500
100/100 [==============================] - 1s 7ms/step - loss: 2.2463e-05 - accuracy: 1.0000 - val_loss: 0.3191
- val_accuracy: 0.9500
Epoch 223/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1631e-05 - accuracy: 1.0000 - val_loss: 0.3325
- val_accuracy: 0.9492
Epoch 224/500
100/100 [==============================] - 1s 7ms/step - loss: 2.0782e-05 - accuracy: 1.0000 - val_loss: 0.3336
- val_accuracy: 0.9483
Epoch 225/500
100/100 [==============================] - 1s 8ms/step - loss: 1.9905e-05 - accuracy: 1.0000 - val_loss: 0.3366
- val_accuracy: 0.9492
Epoch 226/500
100/100 [==============================] - 1s 7ms/step - loss: 1.9678e-05 - accuracy: 1.0000 - val_loss: 0.3227
- val_accuracy: 0.9504
Epoch 227/500
100/100 [==============================] - 1s 7ms/step - loss: 1.8856e-05 - accuracy: 1.0000 - val_loss: 0.3274
- val_accuracy: 0.9496
Epoch 228/500
100/100 [==============================] - 1s 8ms/step - loss: 1.7937e-05 - accuracy: 1.0000 - val_loss: 0.3402
- val_accuracy: 0.9496
Epoch 229/500
100/100 [==============================] - 1s 8ms/step - loss: 1.7274e-05 - accuracy: 1.0000 - val_loss: 0.3248
- val_accuracy: 0.9504
Epoch 230/500
100/100 [==============================] - 1s 10ms/step - loss: 1.5895e-05 - accuracy: 1.0000 - val_loss: 0.3285
- val_accuracy: 0.9517
Epoch 231/500
100/100 [==============================] - 1s 8ms/step - loss: 1.5683e-05 - accuracy: 1.0000 - val_loss: 0.3414
- val_accuracy: 0.9496
Epoch 232/500
100/100 [==============================] - 1s 9ms/step - loss: 1.4528e-05 - accuracy: 1.0000 - val_loss: 0.3401
- val_accuracy: 0.9504
Epoch 233/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4278e-05 - accuracy: 1.0000 - val_loss: 0.3407
- val_accuracy: 0.9504
Epoch 234/500
100/100 [==============================] - 1s 7ms/step - loss: 1.4167e-05 - accuracy: 1.0000 - val_loss: 0.3413
- val_accuracy: 0.9500
Epoch 235/500
100/100 [==============================] - 1s 9ms/step - loss: 1.6710e-05 - accuracy: 1.0000 - val_loss: 0.3315
- val_accuracy: 0.9508
Epoch 236/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0684 - accuracy: 0.9829 - val_loss: 0.4361 - va
l_accuracy: 0.9346
Epoch 237/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0394 - accuracy: 0.9887 - val_loss: 0.4328 - va
l_accuracy: 0.9321
Epoch 238/500
100/100 [==============================] - 1s 8ms/step - loss: 0.0058 - accuracy: 0.9971 - val_loss: 0.3585 - va
l_accuracy: 0.9446
Epoch 239/500
100/100 [==============================] - 1s 7ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.3195 - va
l_accuracy: 0.9471
Epoch 240/500
100/100 [==============================] - 1s 7ms/step - loss: 1.8630e-04 - accuracy: 1.0000 - val_loss: 0.3257
- val_accuracy: 0.9483
Epoch 241/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4186e-04 - accuracy: 1.0000 - val_loss: 0.3340
- val_accuracy: 0.9479
Epoch 242/500
100/100 [==============================] - 1s 7ms/step - loss: 1.3062e-04 - accuracy: 1.0000 - val_loss: 0.3240
- val_accuracy: 0.9500
Epoch 243/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2317e-04 - accuracy: 1.0000 - val_loss: 0.3229
- val_accuracy: 0.9508
Epoch 244/500
100/100 [==============================] - 1s 7ms/step - loss: 1.1618e-04 - accuracy: 1.0000 - val_loss: 0.3251
- val_accuracy: 0.9496
Epoch 245/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0668e-04 - accuracy: 1.0000 - val_loss: 0.3237
- val_accuracy: 0.9513
Epoch 246/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0433e-04 - accuracy: 1.0000 - val_loss: 0.3178
- val_accuracy: 0.9525
Epoch 247/500
100/100 [==============================] - 1s 8ms/step - loss: 9.6704e-05 - accuracy: 1.0000 - val_loss: 0.3238
- val_accuracy: 0.9500
Epoch 248/500
```

```
100/100 [==============================] - 1s 6ms/step - loss: 9.5457e-05 - accuracy: 1.0000 - val_loss: 0.3225
- val_accuracy: 0.9508
Epoch 249/500
100/100 [==============================] - 1s 6ms/step - loss: 9.0202e-05 - accuracy: 1.0000 - val_loss: 0.3262
- val_accuracy: 0.9513
Epoch 250/500
100/100 [==============================] - 1s 7ms/step - loss: 8.7316e-05 - accuracy: 1.0000 - val_loss: 0.3319
- val_accuracy: 0.9504
Epoch 251/500
100/100 [==============================] - 1s 6ms/step - loss: 8.0828e-05 - accuracy: 1.0000 - val_loss: 0.3258
- val_accuracy: 0.9500
Epoch 252/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2228e-04 - accuracy: 1.0000 - val_loss: 0.3104
- val_accuracy: 0.9525
Epoch 253/500
100/100 [==============================] - 1s 6ms/step - loss: 3.5257e-04 - accuracy: 1.0000 - val_loss: 0.3217
- val_accuracy: 0.9492
Epoch 254/500
100/100 [==============================] - 1s 6ms/step - loss: 8.0765e-05 - accuracy: 1.0000 - val_loss: 0.3289
- val_accuracy: 0.9500
Epoch 255/500
100/100 [==============================] - 1s 6ms/step - loss: 7.9732e-05 - accuracy: 1.0000 - val_loss: 0.3316
- val_accuracy: 0.9517
Epoch 256/500
100/100 [==============================] - 1s 6ms/step - loss: 7.4542e-05 - accuracy: 1.0000 - val_loss: 0.3131
- val_accuracy: 0.9517
Epoch 257/500
100/100 [==============================] - 1s 7ms/step - loss: 6.5460e-05 - accuracy: 1.0000 - val_loss: 0.3281
- val_accuracy: 0.9513
Epoch 258/500
100/100 [==============================] - 1s 6ms/step - loss: 6.6856e-05 - accuracy: 1.0000 - val_loss: 0.3331
- val_accuracy: 0.9504
Epoch 259/500
100/100 [==============================] - 1s 6ms/step - loss: 6.1795e-05 - accuracy: 1.0000 - val_loss: 0.3151
- val_accuracy: 0.9508
Epoch 260/500
100/100 [==============================] - 1s 6ms/step - loss: 5.9160e-05 - accuracy: 1.0000 - val_loss: 0.3238
- val_accuracy: 0.9521
Epoch 261/500
100/100 [==============================] - 1s 6ms/step - loss: 5.6409e-05 - accuracy: 1.0000 - val_loss: 0.3295
- val_accuracy: 0.9492
Epoch 262/500
100/100 [==============================] - 1s 6ms/step - loss: 5.6467e-05 - accuracy: 1.0000 - val_loss: 0.3282
- val_accuracy: 0.9500
Epoch 263/500
100/100 [==============================] - 1s 6ms/step - loss: 5.3885e-05 - accuracy: 1.0000 - val_loss: 0.3264
- val_accuracy: 0.9500
Epoch 264/500
100/100 [==============================] - 1s 6ms/step - loss: 5.1853e-05 - accuracy: 1.0000 - val_loss: 0.3253
- val_accuracy: 0.9500
Epoch 265/500
100/100 [==============================] - 1s 6ms/step - loss: 4.9135e-05 - accuracy: 1.0000 - val_loss: 0.3177
- val_accuracy: 0.9496
Epoch 266/500
100/100 [==============================] - 1s 6ms/step - loss: 4.6886e-05 - accuracy: 1.0000 - val_loss: 0.3191
- val_accuracy: 0.9508
Epoch 267/500
100/100 [==============================] - 1s 6ms/step - loss: 4.7214e-05 - accuracy: 1.0000 - val_loss: 0.3206
- val_accuracy: 0.9504
Epoch 268/500
100/100 [==============================] - 1s 6ms/step - loss: 4.7544e-05 - accuracy: 1.0000 - val_loss: 0.3334
- val_accuracy: 0.9500
Epoch 269/500
100/100 [==============================] - 1s 6ms/step - loss: 4.4641e-05 - accuracy: 1.0000 - val_loss: 0.3265
- val_accuracy: 0.9504
Epoch 270/500
100/100 [==============================] - 1s 6ms/step - loss: 4.2215e-05 - accuracy: 1.0000 - val_loss: 0.3212
- val_accuracy: 0.9513
Epoch 271/500
100/100 [==============================] - 1s 6ms/step - loss: 4.2406e-05 - accuracy: 1.0000 - val_loss: 0.3296
- val_accuracy: 0.9500
Epoch 272/500
100/100 [==============================] - 1s 6ms/step - loss: 3.9915e-05 - accuracy: 1.0000 - val_loss: 0.3300
- val_accuracy: 0.9500
Epoch 273/500
100/100 [==============================] - 1s 7ms/step - loss: 3.8437e-05 - accuracy: 1.0000 - val_loss: 0.3279
- val_accuracy: 0.9500
Epoch 274/500
100/100 [==============================] - 1s 6ms/step - loss: 3.8035e-05 - accuracy: 1.0000 - val_loss: 0.3263
- val_accuracy: 0.9513
Epoch 275/500
100/100 [==============================] - 1s 6ms/step - loss: 3.6963e-05 - accuracy: 1.0000 - val_loss: 0.3155
- val_accuracy: 0.9517
```

```
Epoch 276/500
100/100 [==============================] - 1s 6ms/step - loss: 3.5353e-05 - accuracy: 1.0000 - val_loss: 0.3176
- val_accuracy: 0.9508
Epoch 277/500
100/100 [==============================] - 1s 7ms/step - loss: 3.4563e-05 - accuracy: 1.0000 - val_loss: 0.3153
- val_accuracy: 0.9517
Epoch 278/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3062e-05 - accuracy: 1.0000 - val_loss: 0.3157
- val_accuracy: 0.9513
Epoch 279/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3133e-05 - accuracy: 1.0000 - val_loss: 0.3318
- val_accuracy: 0.9508
Epoch 280/500
100/100 [==============================] - 1s 6ms/step - loss: 3.1492e-05 - accuracy: 1.0000 - val_loss: 0.3297
- val_accuracy: 0.9508
Epoch 281/500
100/100 [==============================] - 1s 7ms/step - loss: 3.1512e-05 - accuracy: 1.0000 - val_loss: 0.3397
- val_accuracy: 0.9504
Epoch 282/500
100/100 [==============================] - 1s 7ms/step - loss: 3.0974e-05 - accuracy: 1.0000 - val_loss: 0.3298
- val_accuracy: 0.9517
Epoch 283/500
100/100 [==============================] - 1s 7ms/step - loss: 2.9508e-05 - accuracy: 1.0000 - val_loss: 0.3302
- val_accuracy: 0.9521
Epoch 284/500
100/100 [==============================] - 1s 7ms/step - loss: 2.7617e-05 - accuracy: 1.0000 - val_loss: 0.3313
- val_accuracy: 0.9521
Epoch 285/500
100/100 [==============================] - 1s 6ms/step - loss: 2.8289e-05 - accuracy: 1.0000 - val_loss: 0.3333
- val_accuracy: 0.9517
Epoch 286/500
100/100 [==============================] - 1s 7ms/step - loss: 2.5688e-05 - accuracy: 1.0000 - val_loss: 0.3333
- val_accuracy: 0.9521
Epoch 287/500
100/100 [==============================] - 1s 7ms/step - loss: 2.4936e-05 - accuracy: 1.0000 - val_loss: 0.3393
- val_accuracy: 0.9517
Epoch 288/500
100/100 [==============================] - 1s 6ms/step - loss: 2.5270e-05 - accuracy: 1.0000 - val_loss: 0.3365
- val_accuracy: 0.9525
Epoch 289/500
100/100 [==============================] - 1s 7ms/step - loss: 2.3521e-05 - accuracy: 1.0000 - val_loss: 0.3344
- val_accuracy: 0.9521
Epoch 290/500
100/100 [==============================] - 1s 6ms/step - loss: 2.3776e-05 - accuracy: 1.0000 - val_loss: 0.3324
- val_accuracy: 0.9517
Epoch 291/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2314e-05 - accuracy: 1.0000 - val_loss: 0.3338
- val_accuracy: 0.9525
Epoch 292/500
100/100 [==============================] - 1s 7ms/step - loss: 2.1790e-05 - accuracy: 1.0000 - val_loss: 0.3359
- val_accuracy: 0.9513
Epoch 293/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0732e-05 - accuracy: 1.0000 - val_loss: 0.3382
- val_accuracy: 0.9521
Epoch 294/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0365e-05 - accuracy: 1.0000 - val_loss: 0.3327
- val_accuracy: 0.9521
Epoch 295/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9193e-05 - accuracy: 1.0000 - val_loss: 0.3341
- val_accuracy: 0.9521
Epoch 296/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9118e-05 - accuracy: 1.0000 - val_loss: 0.3218
- val_accuracy: 0.9525
Epoch 297/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8657e-05 - accuracy: 1.0000 - val_loss: 0.3346
- val_accuracy: 0.9517
Epoch 298/500
100/100 [==============================] - 1s 6ms/step - loss: 1.7733e-05 - accuracy: 1.0000 - val_loss: 0.3412
- val_accuracy: 0.9525
Epoch 299/500
100/100 [==============================] - 1s 7ms/step - loss: 1.6771e-05 - accuracy: 1.0000 - val_loss: 0.3358
- val_accuracy: 0.9508
Epoch 300/500
100/100 [==============================] - 1s 6ms/step - loss: 1.7178e-05 - accuracy: 1.0000 - val_loss: 0.3416
- val_accuracy: 0.9525
Epoch 301/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5912e-05 - accuracy: 1.0000 - val_loss: 0.3415
- val_accuracy: 0.9521
Epoch 302/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5460e-05 - accuracy: 1.0000 - val_loss: 0.3435
- val_accuracy: 0.9517
Epoch 303/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4554e-05 - accuracy: 1.0000 - val_loss: 0.3227
```

```
- val_accuracy: 0.9529
Epoch 304/500
100/100 [==============================] - 1s 7ms/step - loss: 1.4578e-05 - accuracy: 1.0000 - val_loss: 0.3407
- val_accuracy: 0.9529
Epoch 305/500
100/100 [==============================] - 1s 7ms/step - loss: 1.3702e-05 - accuracy: 1.0000 - val_loss: 0.3405
- val_accuracy: 0.9521
Epoch 306/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3698e-05 - accuracy: 1.0000 - val_loss: 0.3476
- val_accuracy: 0.9513
Epoch 307/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4634e-05 - accuracy: 1.0000 - val_loss: 0.3335
- val_accuracy: 0.9538
Epoch 308/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3450e-05 - accuracy: 1.0000 - val_loss: 0.3391
- val_accuracy: 0.9525
Epoch 309/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2440e-05 - accuracy: 1.0000 - val_loss: 0.3239
- val_accuracy: 0.9529
Epoch 310/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2185e-05 - accuracy: 1.0000 - val_loss: 0.3388
- val_accuracy: 0.9521
Epoch 311/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2096e-05 - accuracy: 1.0000 - val_loss: 0.3368
- val_accuracy: 0.9525
Epoch 312/500
100/100 [==============================] - 1s 7ms/step - loss: 1.1785e-05 - accuracy: 1.0000 - val_loss: 0.3362
- val_accuracy: 0.9529
Epoch 313/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0979e-05 - accuracy: 1.0000 - val_loss: 0.3412
- val_accuracy: 0.9517
Epoch 314/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0429e-05 - accuracy: 1.0000 - val_loss: 0.3401
- val_accuracy: 0.9525
Epoch 315/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0351e-05 - accuracy: 1.0000 - val_loss: 0.3425
- val_accuracy: 0.9508
Epoch 316/500
100/100 [==============================] - 1s 6ms/step - loss: 9.8778e-06 - accuracy: 1.0000 - val_loss: 0.3513
- val_accuracy: 0.9504
Epoch 317/500
100/100 [==============================] - 1s 7ms/step - loss: 9.4283e-06 - accuracy: 1.0000 - val_loss: 0.3414
- val_accuracy: 0.9517
Epoch 318/500
100/100 [==============================] - 1s 7ms/step - loss: 8.9061e-06 - accuracy: 1.0000 - val_loss: 0.3429
- val_accuracy: 0.9508
Epoch 319/500
100/100 [==============================] - 1s 7ms/step - loss: 8.7786e-06 - accuracy: 1.0000 - val_loss: 0.3419
- val_accuracy: 0.9513
Epoch 320/500
100/100 [==============================] - 1s 6ms/step - loss: 8.3288e-06 - accuracy: 1.0000 - val_loss: 0.3489
- val_accuracy: 0.9517
Epoch 321/500
100/100 [==============================] - 1s 7ms/step - loss: 8.0788e-06 - accuracy: 1.0000 - val_loss: 0.3460
- val_accuracy: 0.9513
Epoch 322/500
100/100 [==============================] - 1s 7ms/step - loss: 8.0167e-06 - accuracy: 1.0000 - val_loss: 0.3403
- val_accuracy: 0.9517
Epoch 323/500
100/100 [==============================] - 1s 7ms/step - loss: 7.5264e-06 - accuracy: 1.0000 - val_loss: 0.3455
- val_accuracy: 0.9513
Epoch 324/500
100/100 [==============================] - 1s 7ms/step - loss: 7.2609e-06 - accuracy: 1.0000 - val_loss: 0.3460
- val_accuracy: 0.9508
Epoch 325/500
100/100 [==============================] - 1s 7ms/step - loss: 7.3717e-06 - accuracy: 1.0000 - val_loss: 0.3507
- val_accuracy: 0.9492
Epoch 326/500
100/100 [==============================] - 1s 8ms/step - loss: 7.5084e-06 - accuracy: 1.0000 - val_loss: 0.3430
- val_accuracy: 0.9517
Epoch 327/500
100/100 [==============================] - 1s 6ms/step - loss: 7.6367e-06 - accuracy: 1.0000 - val_loss: 0.3548
- val_accuracy: 0.9517
Epoch 328/500
100/100 [==============================] - 1s 7ms/step - loss: 6.4763e-06 - accuracy: 1.0000 - val_loss: 0.3357
- val_accuracy: 0.9508
Epoch 329/500
100/100 [==============================] - 1s 7ms/step - loss: 6.1248e-06 - accuracy: 1.0000 - val_loss: 0.3562
- val_accuracy: 0.9508
Epoch 330/500
100/100 [==============================] - 1s 7ms/step - loss: 5.9264e-06 - accuracy: 1.0000 - val_loss: 0.3511
- val_accuracy: 0.9517
Epoch 331/500
```

```
100/100 [==============================] - 1s 6ms/step - loss: 5.6684e-06 - accuracy: 1.0000 - val_loss: 0.3551
- val_accuracy: 0.9504
Epoch 332/500
100/100 [==============================] - 1s 6ms/step - loss: 5.5261e-06 - accuracy: 1.0000 - val_loss: 0.3516
- val_accuracy: 0.9513
Epoch 333/500
100/100 [==============================] - 1s 6ms/step - loss: 5.1972e-06 - accuracy: 1.0000 - val_loss: 0.3606
- val_accuracy: 0.9508
Epoch 334/500
100/100 [==============================] - 1s 6ms/step - loss: 5.1775e-06 - accuracy: 1.0000 - val_loss: 0.3513
- val_accuracy: 0.9517
Epoch 335/500
100/100 [==============================] - 1s 7ms/step - loss: 4.7835e-06 - accuracy: 1.0000 - val_loss: 0.3612
- val_accuracy: 0.9508
Epoch 336/500
100/100 [==============================] - 1s 7ms/step - loss: 4.6484e-06 - accuracy: 1.0000 - val_loss: 0.3537
- val_accuracy: 0.9513
Epoch 337/500
100/100 [==============================] - 1s 6ms/step - loss: 4.6087e-06 - accuracy: 1.0000 - val_loss: 0.3590
- val_accuracy: 0.9500
Epoch 338/500
100/100 [==============================] - 1s 6ms/step - loss: 4.3030e-06 - accuracy: 1.0000 - val_loss: 0.3567
- val_accuracy: 0.9504
Epoch 339/500
100/100 [==============================] - 1s 6ms/step - loss: 4.3102e-06 - accuracy: 1.0000 - val_loss: 0.3668
- val_accuracy: 0.9504
Epoch 340/500
100/100 [==============================] - 1s 6ms/step - loss: 4.0186e-06 - accuracy: 1.0000 - val_loss: 0.3566
- val_accuracy: 0.9504
Epoch 341/500
100/100 [==============================] - 1s 6ms/step - loss: 3.8150e-06 - accuracy: 1.0000 - val_loss: 0.3557
- val_accuracy: 0.9517
Epoch 342/500
100/100 [==============================] - 1s 6ms/step - loss: 4.1640e-06 - accuracy: 1.0000 - val_loss: 0.3577
- val_accuracy: 0.9513
Epoch 343/500
100/100 [==============================] - 1s 6ms/step - loss: 4.0815e-06 - accuracy: 1.0000 - val_loss: 0.3573
- val_accuracy: 0.9508
Epoch 344/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3984e-06 - accuracy: 1.0000 - val_loss: 0.3732
- val_accuracy: 0.9496
Epoch 345/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3556e-06 - accuracy: 1.0000 - val_loss: 0.3594
- val_accuracy: 0.9513
Epoch 346/500
100/100 [==============================] - 1s 7ms/step - loss: 3.0264e-06 - accuracy: 1.0000 - val_loss: 0.3620
- val_accuracy: 0.9517
Epoch 347/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9299e-06 - accuracy: 1.0000 - val_loss: 0.3669
- val_accuracy: 0.9500
Epoch 348/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9005e-06 - accuracy: 1.0000 - val_loss: 0.3631
- val_accuracy: 0.9521
Epoch 349/500
100/100 [==============================] - 1s 6ms/step - loss: 2.6680e-06 - accuracy: 1.0000 - val_loss: 0.3651
- val_accuracy: 0.9504
Epoch 350/500
100/100 [==============================] - 1s 7ms/step - loss: 2.5467e-06 - accuracy: 1.0000 - val_loss: 0.3691
- val_accuracy: 0.9508
Epoch 351/500
100/100 [==============================] - 1s 6ms/step - loss: 2.6159e-06 - accuracy: 1.0000 - val_loss: 0.3700
- val_accuracy: 0.9504
Epoch 352/500
100/100 [==============================] - 1s 6ms/step - loss: 2.6357e-06 - accuracy: 1.0000 - val_loss: 0.3631
- val_accuracy: 0.9513
Epoch 353/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2982e-06 - accuracy: 1.0000 - val_loss: 0.3737
- val_accuracy: 0.9496
Epoch 354/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1608e-06 - accuracy: 1.0000 - val_loss: 0.3564
- val_accuracy: 0.9508
Epoch 355/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2637e-06 - accuracy: 1.0000 - val_loss: 0.3665
- val_accuracy: 0.9508
Epoch 356/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1718e-06 - accuracy: 1.0000 - val_loss: 0.3777
- val_accuracy: 0.9492
Epoch 357/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9946e-06 - accuracy: 1.0000 - val_loss: 0.3735
- val_accuracy: 0.9517
Epoch 358/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8858e-06 - accuracy: 1.0000 - val_loss: 0.3764
- val_accuracy: 0.9496
```

```
Epoch 359/500
100/100 [==============================] - 1s 7ms/step - loss: 1.7733e-06 - accuracy: 1.0000 - val_loss: 0.3791
- val_accuracy: 0.9504
Epoch 360/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6380e-06 - accuracy: 1.0000 - val_loss: 0.3746
- val_accuracy: 0.9517
Epoch 361/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6028e-06 - accuracy: 1.0000 - val_loss: 0.3739
- val_accuracy: 0.9504
Epoch 362/500
100/100 [==============================] - 1s 7ms/step - loss: 1.5360e-06 - accuracy: 1.0000 - val_loss: 0.3878
- val_accuracy: 0.9521
Epoch 363/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4579e-06 - accuracy: 1.0000 - val_loss: 0.3815
- val_accuracy: 0.9517
Epoch 364/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3943e-06 - accuracy: 1.0000 - val_loss: 0.3891
- val_accuracy: 0.9517
Epoch 365/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3490e-06 - accuracy: 1.0000 - val_loss: 0.3790
- val_accuracy: 0.9504
Epoch 366/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2562e-06 - accuracy: 1.0000 - val_loss: 0.3800
- val_accuracy: 0.9517
Epoch 367/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2161e-06 - accuracy: 1.0000 - val_loss: 0.3848
- val_accuracy: 0.9517
Epoch 368/500
100/100 [==============================] - 1s 7ms/step - loss: 1.1458e-06 - accuracy: 1.0000 - val_loss: 0.3825
- val_accuracy: 0.9525
Epoch 369/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1182e-06 - accuracy: 1.0000 - val_loss: 0.3901
- val_accuracy: 0.9517
Epoch 370/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0076e-06 - accuracy: 1.0000 - val_loss: 0.3845
- val_accuracy: 0.9525
Epoch 371/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0247e-06 - accuracy: 1.0000 - val_loss: 0.3940
- val_accuracy: 0.9513
Epoch 372/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0697e-06 - accuracy: 1.0000 - val_loss: 0.3903
- val_accuracy: 0.9529
Epoch 373/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4524e-06 - accuracy: 1.0000 - val_loss: 0.3893
- val_accuracy: 0.9504
Epoch 374/500
100/100 [==============================] - 1s 6ms/step - loss: 9.0251e-07 - accuracy: 1.0000 - val_loss: 0.4064
- val_accuracy: 0.9488
Epoch 375/500
100/100 [==============================] - 1s 7ms/step - loss: 8.5001e-07 - accuracy: 1.0000 - val_loss: 0.4000
- val_accuracy: 0.9529
Epoch 376/500
100/100 [==============================] - 1s 8ms/step - loss: 7.8857e-07 - accuracy: 1.0000 - val_loss: 0.3989
- val_accuracy: 0.9508
Epoch 377/500
100/100 [==============================] - 1s 7ms/step - loss: 7.5265e-07 - accuracy: 1.0000 - val_loss: 0.4001
- val_accuracy: 0.9513
Epoch 378/500
100/100 [==============================] - 1s 6ms/step - loss: 7.1823e-07 - accuracy: 1.0000 - val_loss: 0.4011
- val_accuracy: 0.9517
Epoch 379/500
100/100 [==============================] - 1s 6ms/step - loss: 6.8391e-07 - accuracy: 1.0000 - val_loss: 0.4000
- val_accuracy: 0.9508
Epoch 380/500
100/100 [==============================] - 1s 6ms/step - loss: 6.5615e-07 - accuracy: 1.0000 - val_loss: 0.3866
- val_accuracy: 0.9533
Epoch 381/500
100/100 [==============================] - 1s 6ms/step - loss: 6.3369e-07 - accuracy: 1.0000 - val_loss: 0.4100
- val_accuracy: 0.9525
Epoch 382/500
100/100 [==============================] - 1s 6ms/step - loss: 6.0682e-07 - accuracy: 1.0000 - val_loss: 0.4039
- val_accuracy: 0.9508
Epoch 383/500
100/100 [==============================] - 1s 7ms/step - loss: 5.4781e-07 - accuracy: 1.0000 - val_loss: 0.4121
- val_accuracy: 0.9521
Epoch 384/500
100/100 [==============================] - 1s 6ms/step - loss: 5.1985e-07 - accuracy: 1.0000 - val_loss: 0.4060
- val_accuracy: 0.9508
Epoch 385/500
100/100 [==============================] - 1s 7ms/step - loss: 4.9859e-07 - accuracy: 1.0000 - val_loss: 0.4065
- val_accuracy: 0.9517
Epoch 386/500
100/100 [==============================] - 1s 6ms/step - loss: 4.8205e-07 - accuracy: 1.0000 - val_loss: 0.4073
```

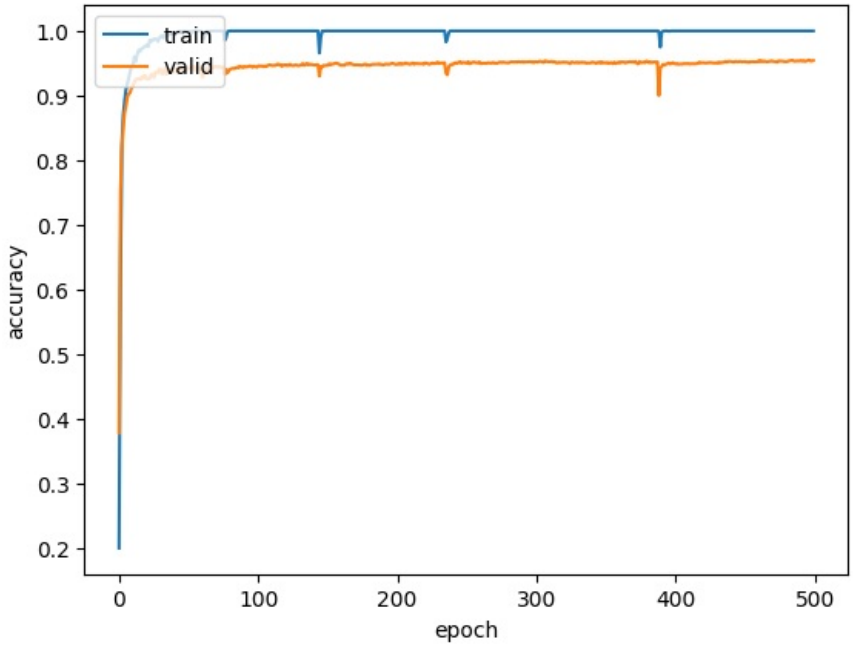
```
- val_accuracy: 0.9521
Epoch 387/500
100/100 [==============================] - 1s 7ms/step - loss: 4.5583e-07 - accuracy: 1.0000 - val_loss: 0.4204
- val_accuracy: 0.9513
Epoch 388/500
100/100 [==============================] - 1s 7ms/step - loss: 5.1031e-07 - accuracy: 1.0000 - val_loss: 0.3945
- val_accuracy: 0.9521
Epoch 389/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0033 - accuracy: 0.9992 - val_loss: 0.9731 - va
l_accuracy: 0.9000
Epoch 390/500
100/100 [==============================] - 1s 6ms/step - loss: 0.1401 - accuracy: 0.9750 - val_loss: 0.4824 - va
l_accuracy: 0.9425
Epoch 391/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0056 - accuracy: 0.9979 - val_loss: 0.4336 - va
l_accuracy: 0.9450
Epoch 392/500
100/100 [==============================] - 1s 6ms/step - loss: 7.7139e-04 - accuracy: 0.9996 - val_loss: 0.4097
- val_accuracy: 0.9483
Epoch 393/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0430e-04 - accuracy: 1.0000 - val_loss: 0.4230
- val_accuracy: 0.9471
Epoch 394/500
100/100 [==============================] - 1s 6ms/step - loss: 8.3392e-05 - accuracy: 1.0000 - val_loss: 0.3912
- val_accuracy: 0.9483
Epoch 395/500
100/100 [==============================] - 1s 6ms/step - loss: 7.2603e-05 - accuracy: 1.0000 - val_loss: 0.4065
- val_accuracy: 0.9492
Epoch 396/500
100/100 [==============================] - 1s 6ms/step - loss: 7.5086e-04 - accuracy: 0.9996 - val_loss: 0.4089
- val_accuracy: 0.9504
Epoch 397/500
100/100 [==============================] - 1s 6ms/step - loss: 0.0012 - accuracy: 0.9996 - val_loss: 0.4446 - va
l_accuracy: 0.9488
Epoch 398/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8605e-04 - accuracy: 1.0000 - val_loss: 0.4201
- val_accuracy: 0.9496
Epoch 399/500
100/100 [==============================] - 1s 6ms/step - loss: 6.7225e-05 - accuracy: 1.0000 - val_loss: 0.4127
- val_accuracy: 0.9508
Epoch 400/500
100/100 [==============================] - 1s 6ms/step - loss: 5.1742e-05 - accuracy: 1.0000 - val_loss: 0.4099
- val_accuracy: 0.9496
Epoch 401/500
100/100 [==============================] - 1s 6ms/step - loss: 4.3971e-05 - accuracy: 1.0000 - val_loss: 0.4212
- val_accuracy: 0.9488
Epoch 402/500
100/100 [==============================] - 1s 6ms/step - loss: 4.2035e-05 - accuracy: 1.0000 - val_loss: 0.4084
- val_accuracy: 0.9496
Epoch 403/500
100/100 [==============================] - 1s 6ms/step - loss: 3.7836e-05 - accuracy: 1.0000 - val_loss: 0.4202
- val_accuracy: 0.9483
Epoch 404/500
100/100 [==============================] - 1s 6ms/step - loss: 3.5201e-05 - accuracy: 1.0000 - val_loss: 0.4102
- val_accuracy: 0.9488
Epoch 405/500
100/100 [==============================] - 1s 6ms/step - loss: 3.2264e-05 - accuracy: 1.0000 - val_loss: 0.4179
- val_accuracy: 0.9488
Epoch 406/500
100/100 [==============================] - 1s 6ms/step - loss: 3.2889e-05 - accuracy: 1.0000 - val_loss: 0.4200
- val_accuracy: 0.9488
Epoch 407/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9040e-05 - accuracy: 1.0000 - val_loss: 0.4021
- val_accuracy: 0.9496
Epoch 408/500
100/100 [==============================] - 1s 6ms/step - loss: 2.8808e-05 - accuracy: 1.0000 - val_loss: 0.4077
- val_accuracy: 0.9492
Epoch 409/500
100/100 [==============================] - 1s 6ms/step - loss: 2.7743e-05 - accuracy: 1.0000 - val_loss: 0.4179
- val_accuracy: 0.9492
Epoch 410/500
100/100 [==============================] - 1s 6ms/step - loss: 2.4903e-05 - accuracy: 1.0000 - val_loss: 0.4050
- val_accuracy: 0.9496
Epoch 411/500
100/100 [==============================] - 1s 6ms/step - loss: 2.5586e-05 - accuracy: 1.0000 - val_loss: 0.4194
- val_accuracy: 0.9488
Epoch 412/500
100/100 [==============================] - 1s 6ms/step - loss: 2.3195e-05 - accuracy: 1.0000 - val_loss: 0.4195
- val_accuracy: 0.9492
Epoch 413/500
100/100 [==============================] - 1s 6ms/step - loss: 2.1992e-05 - accuracy: 1.0000 - val_loss: 0.4102
- val_accuracy: 0.9492
Epoch 414/500
```

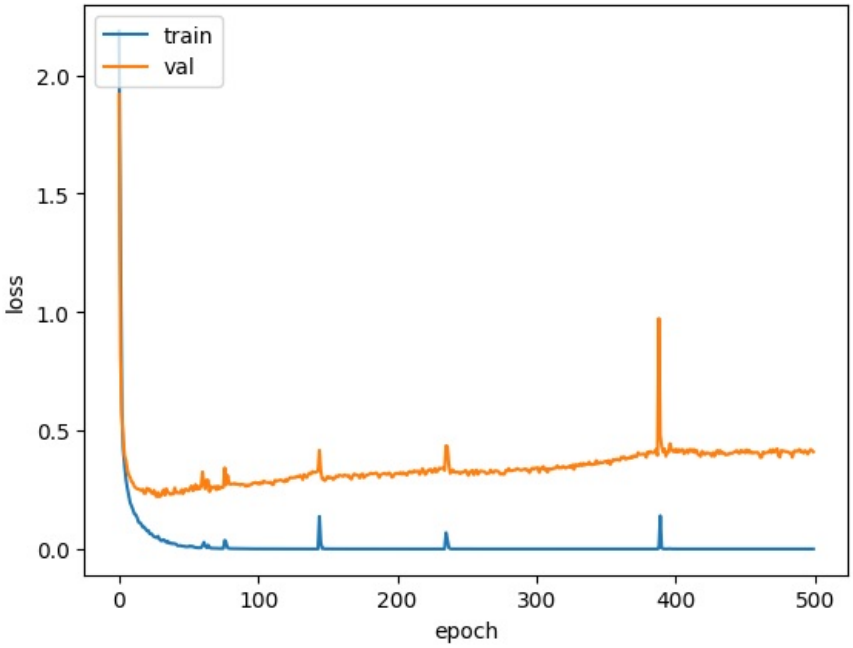
```
100/100 [==============================] - 1s 6ms/step - loss: 2.2794e-05 - accuracy: 1.0000 - val_loss: 0.4198
- val_accuracy: 0.9492
Epoch 415/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0197e-05 - accuracy: 1.0000 - val_loss: 0.4068
- val_accuracy: 0.9492
Epoch 416/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0898e-05 - accuracy: 1.0000 - val_loss: 0.4164
- val_accuracy: 0.9492
Epoch 417/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8574e-05 - accuracy: 1.0000 - val_loss: 0.4007
- val_accuracy: 0.9500
Epoch 418/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9322e-05 - accuracy: 1.0000 - val_loss: 0.3956
- val_accuracy: 0.9517
Epoch 419/500
100/100 [==============================] - 1s 6ms/step - loss: 1.7913e-05 - accuracy: 1.0000 - val_loss: 0.3908
- val_accuracy: 0.9504
Epoch 420/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6239e-05 - accuracy: 1.0000 - val_loss: 0.4128
- val_accuracy: 0.9508
Epoch 421/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6230e-05 - accuracy: 1.0000 - val_loss: 0.4042
- val_accuracy: 0.9508
Epoch 422/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5860e-05 - accuracy: 1.0000 - val_loss: 0.3981
- val_accuracy: 0.9504
Epoch 423/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6331e-05 - accuracy: 1.0000 - val_loss: 0.4060
- val_accuracy: 0.9504
Epoch 424/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5569e-05 - accuracy: 1.0000 - val_loss: 0.4056
- val_accuracy: 0.9504
Epoch 425/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4750e-05 - accuracy: 1.0000 - val_loss: 0.4037
- val_accuracy: 0.9513
Epoch 426/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4247e-05 - accuracy: 1.0000 - val_loss: 0.4008
- val_accuracy: 0.9508
Epoch 427/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2914e-05 - accuracy: 1.0000 - val_loss: 0.3918
- val_accuracy: 0.9508
Epoch 428/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2554e-05 - accuracy: 1.0000 - val_loss: 0.4062
- val_accuracy: 0.9504
Epoch 429/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2800e-05 - accuracy: 1.0000 - val_loss: 0.3920
- val_accuracy: 0.9504
Epoch 430/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2302e-05 - accuracy: 1.0000 - val_loss: 0.4081
- val_accuracy: 0.9504
Epoch 431/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1452e-05 - accuracy: 1.0000 - val_loss: 0.4155
- val_accuracy: 0.9500
Epoch 432/500
100/100 [==============================] - 1s 6ms/step - loss: 1.0937e-05 - accuracy: 1.0000 - val_loss: 0.4021
- val_accuracy: 0.9517
Epoch 433/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0809e-05 - accuracy: 1.0000 - val_loss: 0.4151
- val_accuracy: 0.9504
Epoch 434/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1008e-05 - accuracy: 1.0000 - val_loss: 0.4038
- val_accuracy: 0.9513
Epoch 435/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0447e-05 - accuracy: 1.0000 - val_loss: 0.4072
- val_accuracy: 0.9513
Epoch 436/500
100/100 [==============================] - 1s 7ms/step - loss: 1.0257e-05 - accuracy: 1.0000 - val_loss: 0.4033
- val_accuracy: 0.9517
Epoch 437/500
100/100 [==============================] - 1s 6ms/step - loss: 9.6370e-06 - accuracy: 1.0000 - val_loss: 0.3968
- val_accuracy: 0.9529
Epoch 438/500
100/100 [==============================] - 1s 6ms/step - loss: 8.8926e-06 - accuracy: 1.0000 - val_loss: 0.3998
- val_accuracy: 0.9517
Epoch 439/500
100/100 [==============================] - 1s 6ms/step - loss: 9.3446e-06 - accuracy: 1.0000 - val_loss: 0.3986
- val_accuracy: 0.9529
Epoch 440/500
100/100 [==============================] - 1s 6ms/step - loss: 8.5181e-06 - accuracy: 1.0000 - val_loss: 0.4043
- val_accuracy: 0.9529
Epoch 441/500
100/100 [==============================] - 1s 6ms/step - loss: 8.5862e-06 - accuracy: 1.0000 - val_loss: 0.4042
- val_accuracy: 0.9525
```

```
Epoch 442/500
100/100 [==============================] - 1s 6ms/step - loss: 8.2182e-06 - accuracy: 1.0000 - val_loss: 0.4039
- val_accuracy: 0.9521
Epoch 443/500
100/100 [==============================] - 1s 6ms/step - loss: 7.9002e-06 - accuracy: 1.0000 - val_loss: 0.4086
- val_accuracy: 0.9517
Epoch 444/500
100/100 [==============================] - 1s 6ms/step - loss: 7.6697e-06 - accuracy: 1.0000 - val_loss: 0.4123
- val_accuracy: 0.9529
Epoch 445/500
100/100 [==============================] - 1s 6ms/step - loss: 7.3324e-06 - accuracy: 1.0000 - val_loss: 0.4151
- val_accuracy: 0.9525
Epoch 446/500
100/100 [==============================] - 1s 6ms/step - loss: 7.1388e-06 - accuracy: 1.0000 - val_loss: 0.4062
- val_accuracy: 0.9521
Epoch 447/500
100/100 [==============================] - 1s 6ms/step - loss: 6.5950e-06 - accuracy: 1.0000 - val_loss: 0.4036
- val_accuracy: 0.9529
Epoch 448/500
100/100 [==============================] - 1s 6ms/step - loss: 6.3809e-06 - accuracy: 1.0000 - val_loss: 0.4165
- val_accuracy: 0.9525
Epoch 449/500
100/100 [==============================] - 1s 6ms/step - loss: 6.4711e-06 - accuracy: 1.0000 - val_loss: 0.4081
- val_accuracy: 0.9521
Epoch 450/500
100/100 [==============================] - 1s 6ms/step - loss: 6.3765e-06 - accuracy: 1.0000 - val_loss: 0.4135
- val_accuracy: 0.9525
Epoch 451/500
100/100 [==============================] - 1s 6ms/step - loss: 5.9861e-06 - accuracy: 1.0000 - val_loss: 0.4057
- val_accuracy: 0.9525
Epoch 452/500
100/100 [==============================] - 1s 6ms/step - loss: 5.8717e-06 - accuracy: 1.0000 - val_loss: 0.4184
- val_accuracy: 0.9517
Epoch 453/500
100/100 [==============================] - 1s 6ms/step - loss: 5.6227e-06 - accuracy: 1.0000 - val_loss: 0.4067
- val_accuracy: 0.9521
Epoch 454/500
100/100 [==============================] - 1s 6ms/step - loss: 5.4877e-06 - accuracy: 1.0000 - val_loss: 0.4162
- val_accuracy: 0.9513
Epoch 455/500
100/100 [==============================] - 1s 6ms/step - loss: 5.2905e-06 - accuracy: 1.0000 - val_loss: 0.4051
- val_accuracy: 0.9525
Epoch 456/500
100/100 [==============================] - 1s 6ms/step - loss: 4.8833e-06 - accuracy: 1.0000 - val_loss: 0.4084
- val_accuracy: 0.9521
Epoch 457/500
100/100 [==============================] - 1s 6ms/step - loss: 4.9982e-06 - accuracy: 1.0000 - val_loss: 0.4049
- val_accuracy: 0.9525
Epoch 458/500
100/100 [==============================] - 1s 6ms/step - loss: 4.7606e-06 - accuracy: 1.0000 - val_loss: 0.4076
- val_accuracy: 0.9521
Epoch 459/500
100/100 [==============================] - 1s 6ms/step - loss: 4.3597e-06 - accuracy: 1.0000 - val_loss: 0.4072
- val_accuracy: 0.9521
Epoch 460/500
100/100 [==============================] - 1s 6ms/step - loss: 4.3703e-06 - accuracy: 1.0000 - val_loss: 0.4076
- val_accuracy: 0.9525
Epoch 461/500
100/100 [==============================] - 1s 6ms/step - loss: 4.2251e-06 - accuracy: 1.0000 - val_loss: 0.4005
- val_accuracy: 0.9542
Epoch 462/500
100/100 [==============================] - 1s 6ms/step - loss: 4.1565e-06 - accuracy: 1.0000 - val_loss: 0.3857
- val_accuracy: 0.9525
Epoch 463/500
100/100 [==============================] - 1s 6ms/step - loss: 3.9706e-06 - accuracy: 1.0000 - val_loss: 0.3876
- val_accuracy: 0.9538
Epoch 464/500
100/100 [==============================] - 1s 6ms/step - loss: 3.9445e-06 - accuracy: 1.0000 - val_loss: 0.4100
- val_accuracy: 0.9525
Epoch 465/500
100/100 [==============================] - 1s 6ms/step - loss: 3.7341e-06 - accuracy: 1.0000 - val_loss: 0.4038
- val_accuracy: 0.9533
Epoch 466/500
100/100 [==============================] - 1s 7ms/step - loss: 3.5312e-06 - accuracy: 1.0000 - val_loss: 0.4062
- val_accuracy: 0.9525
Epoch 467/500
100/100 [==============================] - 1s 6ms/step - loss: 3.4568e-06 - accuracy: 1.0000 - val_loss: 0.4081
- val_accuracy: 0.9521
Epoch 468/500
100/100 [==============================] - 1s 6ms/step - loss: 3.2735e-06 - accuracy: 1.0000 - val_loss: 0.4074
- val_accuracy: 0.9529
Epoch 469/500
100/100 [==============================] - 1s 6ms/step - loss: 3.3012e-06 - accuracy: 1.0000 - val_loss: 0.4184
```

```
- val_accuracy: 0.9529
Epoch 470/500
100/100 [==============================] - 1s 6ms/step - loss: 3.0507e-06 - accuracy: 1.0000 - val_loss: 0.4060
- val_accuracy: 0.9538
Epoch 471/500
100/100 [==============================] - 1s 6ms/step - loss: 3.0678e-06 - accuracy: 1.0000 - val_loss: 0.4192
- val_accuracy: 0.9529
Epoch 472/500
100/100 [==============================] - 1s 6ms/step - loss: 2.9627e-06 - accuracy: 1.0000 - val_loss: 0.4086
- val_accuracy: 0.9525
Epoch 473/500
100/100 [==============================] - 1s 6ms/step - loss: 2.8829e-06 - accuracy: 1.0000 - val_loss: 0.4164
- val_accuracy: 0.9525
Epoch 474/500
100/100 [==============================] - 1s 7ms/step - loss: 2.6918e-06 - accuracy: 1.0000 - val_loss: 0.4228
- val_accuracy: 0.9517
Epoch 475/500
100/100 [==============================] - 1s 6ms/step - loss: 2.6082e-06 - accuracy: 1.0000 - val_loss: 0.3924
- val_accuracy: 0.9538
Epoch 476/500
100/100 [==============================] - 1s 6ms/step - loss: 2.5050e-06 - accuracy: 1.0000 - val_loss: 0.3878
- val_accuracy: 0.9533
Epoch 477/500
100/100 [==============================] - 1s 7ms/step - loss: 2.4001e-06 - accuracy: 1.0000 - val_loss: 0.4159
- val_accuracy: 0.9533
Epoch 478/500
100/100 [==============================] - 1s 6ms/step - loss: 2.3466e-06 - accuracy: 1.0000 - val_loss: 0.4136
- val_accuracy: 0.9533
Epoch 479/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2913e-06 - accuracy: 1.0000 - val_loss: 0.4133
- val_accuracy: 0.9525
Epoch 480/500
100/100 [==============================] - 1s 6ms/step - loss: 2.2213e-06 - accuracy: 1.0000 - val_loss: 0.4104
- val_accuracy: 0.9525
Epoch 481/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0726e-06 - accuracy: 1.0000 - val_loss: 0.4115
- val_accuracy: 0.9525
Epoch 482/500
100/100 [==============================] - 1s 6ms/step - loss: 2.0154e-06 - accuracy: 1.0000 - val_loss: 0.4081
- val_accuracy: 0.9533
Epoch 483/500
100/100 [==============================] - 1s 6ms/step - loss: 1.9575e-06 - accuracy: 1.0000 - val_loss: 0.4047
- val_accuracy: 0.9538
Epoch 484/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8424e-06 - accuracy: 1.0000 - val_loss: 0.4056
- val_accuracy: 0.9542
Epoch 485/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8703e-06 - accuracy: 1.0000 - val_loss: 0.3869
- val_accuracy: 0.9546
Epoch 486/500
100/100 [==============================] - 1s 6ms/step - loss: 1.8106e-06 - accuracy: 1.0000 - val_loss: 0.4117
- val_accuracy: 0.9533
Epoch 487/500
100/100 [==============================] - 1s 6ms/step - loss: 1.7136e-06 - accuracy: 1.0000 - val_loss: 0.4145
- val_accuracy: 0.9533
Epoch 488/500
100/100 [==============================] - 1s 6ms/step - loss: 1.6610e-06 - accuracy: 1.0000 - val_loss: 0.4116
- val_accuracy: 0.9533
Epoch 489/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5791e-06 - accuracy: 1.0000 - val_loss: 0.4134
- val_accuracy: 0.9538
Epoch 490/500
100/100 [==============================] - 1s 6ms/step - loss: 1.5007e-06 - accuracy: 1.0000 - val_loss: 0.4200
- val_accuracy: 0.9533
Epoch 491/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4536e-06 - accuracy: 1.0000 - val_loss: 0.4097
- val_accuracy: 0.9542
Epoch 492/500
100/100 [==============================] - 1s 6ms/step - loss: 1.4207e-06 - accuracy: 1.0000 - val_loss: 0.4127
- val_accuracy: 0.9533
Epoch 493/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3293e-06 - accuracy: 1.0000 - val_loss: 0.4050
- val_accuracy: 0.9542
Epoch 494/500
100/100 [==============================] - 1s 6ms/step - loss: 1.3060e-06 - accuracy: 1.0000 - val_loss: 0.4154
- val_accuracy: 0.9529
Epoch 495/500
100/100 [==============================] - 1s 7ms/step - loss: 1.2852e-06 - accuracy: 1.0000 - val_loss: 0.4099
- val_accuracy: 0.9533
Epoch 496/500
100/100 [==============================] - 1s 6ms/step - loss: 1.2364e-06 - accuracy: 1.0000 - val_loss: 0.4006
- val_accuracy: 0.9542
Epoch 497/500
```

```
100/100 [==============================] - 1s 6ms/step - loss: 1.1950e-06 - accuracy: 1.0000 - val_loss: 0.4097
- val_accuracy: 0.9550
Epoch 498/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1389e-06 - accuracy: 1.0000 - val_loss: 0.4199
- val_accuracy: 0.9533
Epoch 499/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1161e-06 - accuracy: 1.0000 - val_loss: 0.4159
- val_accuracy: 0.9538
Epoch 500/500
100/100 [==============================] - 1s 6ms/step - loss: 1.1099e-06 - accuracy: 1.0000 - val_loss: 0.4098
- val_accuracy: 0.9542
Test loss: 0.43929633498191833
Test accuracy: 0.9549000263214111
```





Malachy and Aroushi-

Week Starting Febuary 19th:

Read sections 2 - 2.9.3

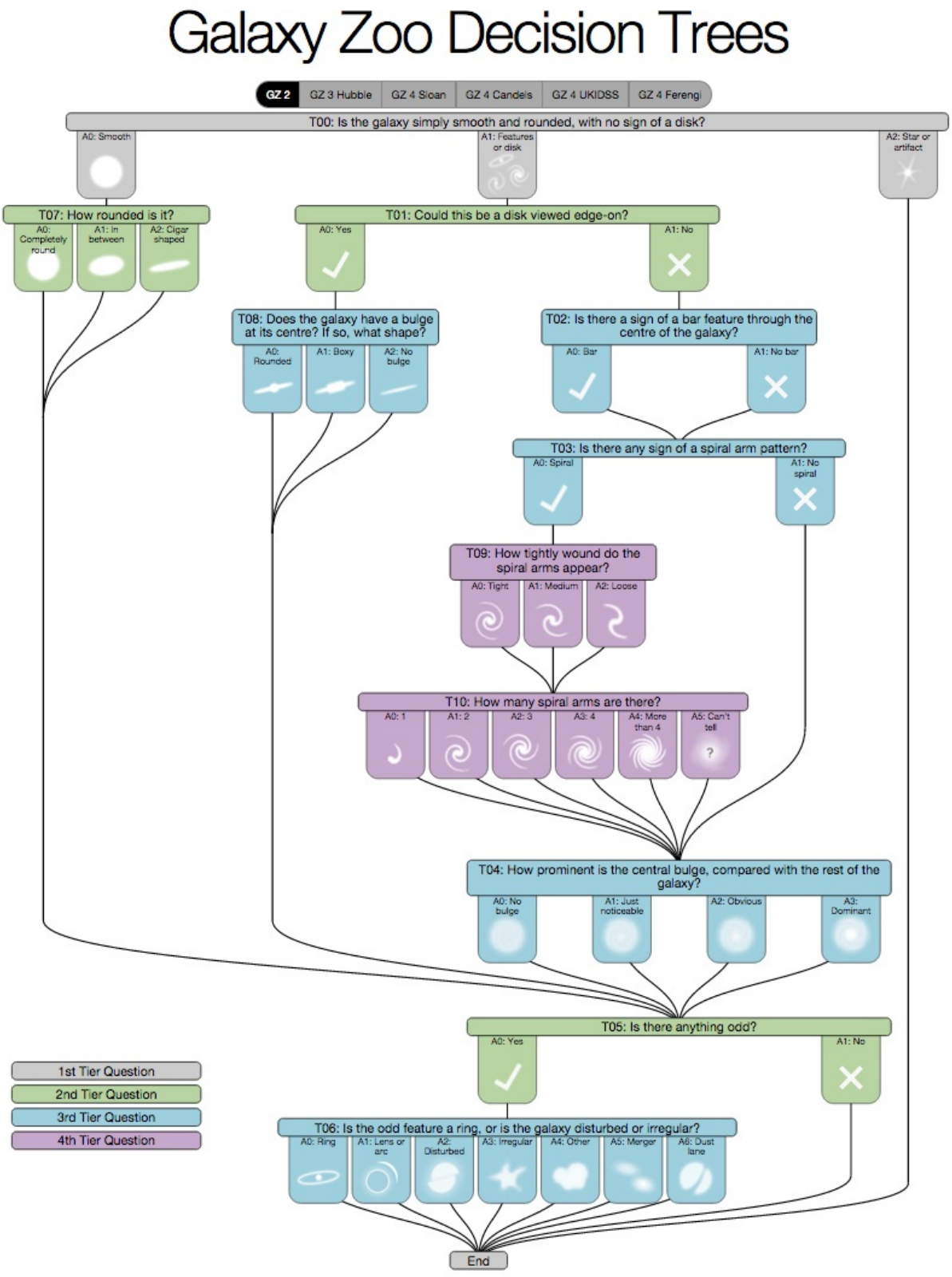
Thursday Febuary 22nd 10am-11am SUPERVIOR MEETING

- look at galaxy dataset, galaxy zoo 2 . https://pypi.org/project/galaxy-datasets/
- get comfortable with data and labels
- concentrate on top part of the classification tree
- being to classify basic galaxy classify
- Begun discussions how we would define class labels within code (choice between one-hot encoded, fractional probabilities) - if we were to choose to use probabilities, would we define a probability as a fraction of total votes, or weight the voting fraction based on which question of the tree is being answered, amongst other methods
- Discussed previously conducted research - past student groups have used layers of the tree separately rather than the 35 classes as

a whole

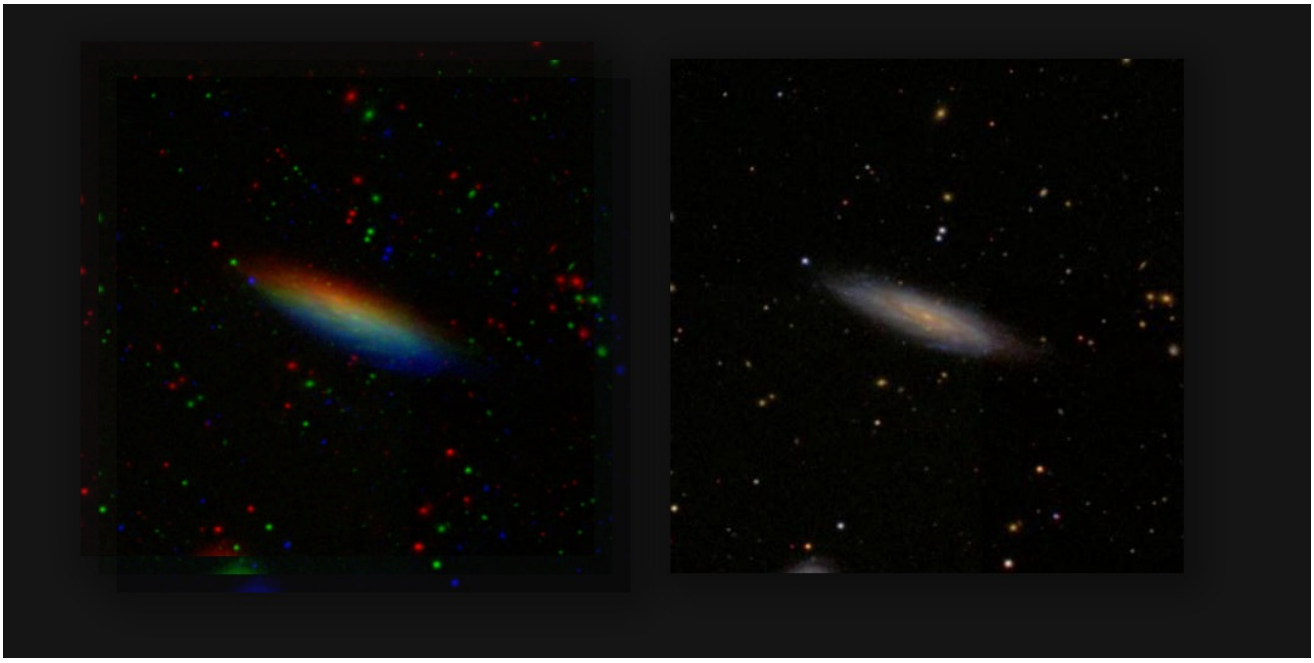
- Aroushi will work on one-hot encoded labels and Malachy will work on voting fractions to start with, we will them compare approaches

Malachy - Research into galaxy zoo dataset



Galaxy zoo classification decision tree diagrams

167434 images of distinct galaxies in the dataset Images are 424 x 424 JPG files 3 Channels - RGB 8-bit colour depth.



Colour Channels of Example Galaxy Image

class labels are a number from 0 to 6, each number represents a different class, with -1 being an unclassified class class names: 7 distinct classes. 1 unclassified class.

Class Label	Class Name	Number of Galaxies	% of Galaxies
-1	Missing Value (could be unclassified or artifact)	9461 galaxies	5.65%
0	"smooth_inbetween"	62857 galaxies	37.54%
1	"smooth_round"	43800 galaxies	26.16%
2	"smooth_cigar"	15801 galaxies	9.44%
3	"edge_on_disk"	3729 galaxies	2.23%
4	"unbarred_spiral"	20992 galaxies	12.54%
5	"barred_spiral"	7346 galaxies	4.39%
6	"featured_without_bar_or_spiral"	3448 galaxies	2.06%

---

can interpret these as the basic classes of galaxies. coarse features of galaxies that are easy to classify. the other features are more fine grained, and are not as easy to classify as they are rarer.

Galaxy Zoo 2 decision tree: 37 Options in total in the decision tree

- **Percentages of answers to the questions in the decision tree: (data from datawrangler, need to go through this manually do some more analysis and get information on some of the questions)**
  - is the galaxy simply smooth and rounded with no sign of a disk, or is it featured/disk, or a aritfact?
    - 75% of the galaxies are classified as smooth
    - 24% of the galaxies are classified as featured-or-disk
    - <1% of the galaxies are classified as deadlock
    - <1% of the galaxies are classified as artifact
  - is the galaxy disk viewed edge on?
    - 76% of the galaxies are unclassified in the disk edge on question
    - 24% of the galaxies are classified as no
    - 3% of the galaxies are classified as yes
    - <1% of the galaxies are classified as deadlock
  - is there any sign of a spiral arm pattern?
    - 76% of the galaxies are unclassified in the has spiral arms question
    - 18% of the galaxies are classified as yes
    - 4% of the galaxies are classified as no
    - 2% of the galaxies are classified as deadlock
  - is there a sign of a bar feature through the centre of the galaxy?
    - 76% of the galaxies are unclassified in the bar question

        - 6% of the galaxies are classified as yes
        - 16% of the galaxies are classified as no
        - 2% of the galaxies are classified as deadlock
    - how prominent is the central bulge, compared to the rest of the galaxy?
        - 76% of the galaxies are unclassified in the bulge size question
        - 13% of the galaxies are classified as just noticable
        - 7% of the galaxies are classified as obvious
        - 3% of the galaxies are classified as dominant or deadlock
    - is there anything odd?
        - 91% of the galaxies do not have something odd
        - 8% of the galaxies do have something odd
        - <1% of the galaxies are deadlock
    - how round is the galaxy?
        - 26% of the galaxies are unclassified in the how rounded question
        - 26% of the galaxies are classified as round
        - 38% of the galaxies are classified as in-between
        - 10% of the galaxies are classified as cigar or deadlock
    - does the galaxy have a bulge at its centre? if so, what is the shape of the bulge?
        - 97% of the galaxies are unclassified in the bulge shape question
        - 2% of the galaxies are classified as round
        - 1% have no bulge
        - <1% are classified as boxy
    - how tightly wound do the spiral arms appear?
        - 87% of the galaxies are unclassified in the spiral winding question
        - 6% of the galaxies are classified as medium
        - 4% of the galaxies are classified as tight
        - 2% of the galaxies are classified as loose
    - how many spiral arms are there?
        - 87% of the galaxies are unclassified in the spiral arm count question
        - 8% of the galaxies are classified as 2
        - 3% of the galaxies are classified as can't tell
        - 2% of the galaxies are classified as 3, 4, or more than 4
- final classifications:
    - 38% of the galaxies are smooth inbetween
    - 26% of the galaxies are smooth round
    - 13% of the galaxies are unbared spiral
    - 18% of the galaxies are "other"

NOTE: percentages obtained from datawrangler, I need to go through this manually do some more analysis

Dataset Structure: Key-Value Table .parquet

There are 90 columns in the dataset with keys and values (of vectors) corresponding to:

- 'ra' = right ascension
- 'dec' = declination
- 'smooth-or-featured-gz2_smooth' = number of votes for smooth
- 'smooth-or-featured-gz2_featured-or-disk' = number of votes for featured or disk
- 'smooth-or-featured-gz2_artifact' = number of votes for artifact
- 'disk-edge-on-gz2_yes' = number of votes for seen edge on
- 'disk-edge-on-gz2_no' = number of votes for not seen edge on
- 'bar-gz2_yes' = number of votes for bar
- 'bar-gz2_no' = number of votes for no bar
- 'has-spiral-arms-gz2_yes' = number of votes for spiral arms
- 'has-spiral-arms-gz2_no' = number of votes for no spiral arms
- 'bulge-size-gz2_no' = number of votes for no bulge
- 'bulge-size-gz2_just-noticeable' = number of votes for just noticeable bulge
- 'bulge-size-gz2_obvious' = number of votes for obvious bulge
- 'bulge-size-gz2_dominant' = number of votes for dominant bulge
- 'something-odd-gz2_yes' = number of votes for something odd in the image
- 'something-odd-gz2_no' = number of votes for nothing odd in the image

- 'how-rounded-gz2_round' = number of votes for round galaxy shape
- 'how-rounded-gz2_in-between'= number of votes for inbetween galaxy shape
- 'how-rounded-gz2_cigar' = number of votes for cigar galaxy shape
- 'bulge-shape-gz2_round' = number of votes for round bulge shape
- 'bulge-shape-gz2_boxy' = number of votes for boxy bulge shape
- 'bulge-shape-gz2_no-bulge' = number of votes for no bulge
- 'spiral-winding-gz2_tight' = number of votes for tight spiral arm winding
- 'spiral-winding-gz2_medium' = number of votes for medium spiral arm winding
- 'spiral-winding-gz2_loose' = number of votes for loose spiral arm winding
- 'spiral-arm-count-gz2_1' = number of votes for 1 spiral arm
- 'spiral-arm-count-gz2_2' = number of votes for 2 spiral arms
- 'spiral-arm-count-gz2_3' = number of votes for 3 spiral arms
- 'spiral-arm-count-gz2_4' = number of votes for 4 spiral arms
- 'spiral-arm-count-gz2_more-than-4' = number of votes for more than 4 spiral arms
- 'spiral-arm-count-gz2_cant-tell' = number of votes for can't tell the no. of spiral arms
- 'smooth-or-featured-gz2_total-votes' = total number of votes to the smooth or featured question
- 'smooth-or-featured-gz2_smooth_fraction' = probability that the galaxy in the image is smooth
- 'smooth-or-featured-gz2_featured-or-disk_fraction'= probability that the galaxy in the image is featured or a disk
- 'smooth-or-featured-gz2_artifact_fraction'= probability that the galaxy in the image is an artifact
- 'disk-edge-on-gz2_total-votes' = total number of votes to the disk edge on question
- 'disk-edge-on-gz2_yes_fraction' = probability that the galaxy in the image is seen edge on
- 'disk-edge-on-gz2_no_fraction' = probability that the galaxy in the image is not seen edge on
- 'has-spiral-arms-gz2_total-votes' = total number of votes to the has spiral arms question
- 'has-spiral-arms-gz2_yes_fraction' = probability that the galaxy in the image has spiral arms
- 'has-spiral-arms-gz2_no_fraction' = probability that the galaxy in the image has no spiral arms
- 'bar-gz2_total-votes' = total number of votes to the bar question
- 'bar-gz2_yes_fraction' = probability that the galaxy in the image has a bar
- 'bar-gz2_no_fraction' = probability that the galaxy in the image doesn't have bar
- 'bulge-size-gz2_total-votes' = total number of votes to the bulge size question
- 'bulge-size-gz2_dominant_fraction' = probability that the galaxy in the image has a dominant bulge
- 'bulge-size-gz2_obvious_fraction' = probability that the galaxy in the image has an obvious bulge
- 'bulge-size-gz2_just-noticeable_fraction' = probability that the galaxy in the image has a just noticeable bulge
- 'bulge-size-gz2_no_fraction' = probability that the galaxy in the image has no bulge
- 'something-odd-gz2_total-votes' = total number of votes for the something odd question
- 'something-odd-gz2_yes_fraction' = probability that the galaxy in the image has something odd in it
- 'something-odd-gz2_no_fraction' = probability that the galaxy in the image doesn't have something odd in it
- 'how-rounded-gz2_total-votes' = total number of votes to the how rounded question
- 'how-rounded-gz2_round_fraction' = probability that the galaxy in the image is round in shape
- 'how-rounded-gz2_in-between_fraction' = probability that the galaxy in the image is in between in shape
- 'how-rounded-gz2_cigar_fraction' = probability that the galaxy in the image is cigar like in shape
- 'bulge-shape-gz2_total-votes' = total number of votes to the bulge shape question
- 'bulge-shape-gz2_round_fraction' = probability that the galaxy in the image has a round bulge
- 'bulge-shape-gz2_boxy_fraction' = probability that the galaxy in the image has a boxy bulge
- 'bulge-shape-gz2_no-bulge_fraction' = probability that the galaxy in the image has no bulge
- 'spiral-winding-gz2_total-votes' = total number of votes to the spiral winding question
- 'spiral-winding-gz2_tight_fraction' = probability that the galaxy in the image has tight spiral winding
- 'spiral-winding-gz2_medium_fraction' = probability that the galay in the image has medium spiral winding
- 'spiral-winding-gz2_loose_fraction' = probability that the galaxy in the image has loose spiral winding
- 'spiral-arm-count-gz2_total-votes' = total number of votes to the spiral arm count question
- 'spiral-arm-count-gz2_1_fraction' = probability that the galaxy in the image has 1 spiral arm
- 'spiral-arm-count-gz2_2_fraction' = probability that the galaxy in the image has 2 spiral arms
- 'spiral-arm-count-gz2_3_fraction' = probability that the galaxy in the image has 3 spiral arms
- 'spiral-arm-count-gz2_4_fraction' = probability that the galaxy in the image has 4 spiral arms
- 'spiral-arm-count-gz2_more-than-4_fraction' = probability that the galaxy in the image has more than 4 spiral arms
- 'spiral-arm-count-gz2_cant-tell_fraction' = probability that the galaxy in the image has ??? spiral arms
- 'file_loc' = file location of the image, it's path
- 'subfolder' = subfolder of the image, it's root
- 'iauname' = IAU Name of the galaxy
- 'smooth-or-featured-gz2_semantic' = final classification of the galaxy feature, either smooth or featured/disk
- 'disk-edge-on-gz2_semantic' = final classification of the galaxy feature, either edge on or not edge on
- 'has-spiral-arms-gz2_semantic' = final classifiction of the galaxy feature, either has spiral arms or doesn't
- 'bar-gz2_semantic' = final classification of the galaxy feature, either has a bar or doesn't
- 'bulge-size-gz2_semantic' = final classification of the galaxy feature, either has a just noticeable, obvious, dominant, or no bulge
- 'something-odd-gz2_semantic' = final classification of the galaxy feature, either has something odd or doesn't

- 'how-rounded-gz2_semantic' = final classification of the galaxy feature, either is rounded, inbetween, or cigar shaped
- 'bulge-shape-gz2_semantic' = final classification of the galaxy feature, either is round, boxy, or no bulge
- 'spiral-winding-gz2_semantic' = final classification of the galaxy feature, either loose, medium, or tight spiral arm winding
- 'spiral-arm-count-gz2_semantic' = final classification of the galaxy feature, either 1, 2, 3, 4, more than 4, or ??? number of spiral arm(s)
- 'summary' = final classification of the galaxy based off the final classifications of the other galaxy features.
- 'leaf_prob' = probability of being a leaf
- 'label' = final classification of the galaxy as an integer label
- 'filename' = the filename of the image containing the galaxy
- 'id_str' = the unique id given to the galaxy within the dataset

TO DO:

- BASIC VISUALISATION OF SOME OF THE DATA, SUCH AS HISTOGRAMS OF THE VOTES AT EACH LAYER & PIE CHARTS OF CLASSES
- will have to clean-up data as there is a fair bit of missing data or incomplete data, possibly just delete these entries.
- will have to augment the classes that are rare/ low probability, i.e try to make probability dist for all classes somewhat uniform.

Malachy and Aroushi -

Thursday 29th Feb SUPERVISOR MEETING

- techniques to get around class imbalences
- undersampling, use less of claases with high % (wasteful)
- oversampling, use more of classes with low % (augment more of the smaller % classes to match larger classes)
- guassian blur - go to fourier space, low amplitudes of high frequency modes
- discrete resolution changing
- visualise data (to figure out how to crop images to remove unwanted surrounding features / reduce size of input matrix)
- see how model performs with no augmentation to start with, using the 8 classes from summary column
- could augment before training (manually)
- could agument on the fly (might be difficult to do)
- convert labels to 0 - 8 (add -1 to end so -1 -> 8) catalog["label"] + 1
- Can use catalog as a pandas dataframe and access data directly in python using the following code
- catalog['summary']
- catalog['file_loc']
- im = PIL.image.open(catalog['file_loc'].iloc[10]) #to look at images easily
- Can give model.fit file names instead of numpy arrays
- It can randomly sample file names using flow_from_dataframe

29/02/2024 Malachy - Galaxy zoo dataset research

made some quick and dirty scripts to do some basic statistical analysis on the dataset

In [ ]:
```
# Galaxy Zoo 2 dataset analysis

import pandas as pd
import os
import matplotlib.pyplot as plt

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

# Read the Parquet file
df = pd.read_parquet(parquet_file_path)

class_labels = df['label']

print(f"\nNumber of Galaxies in the dataset: {len(class_labels)}\n")
```

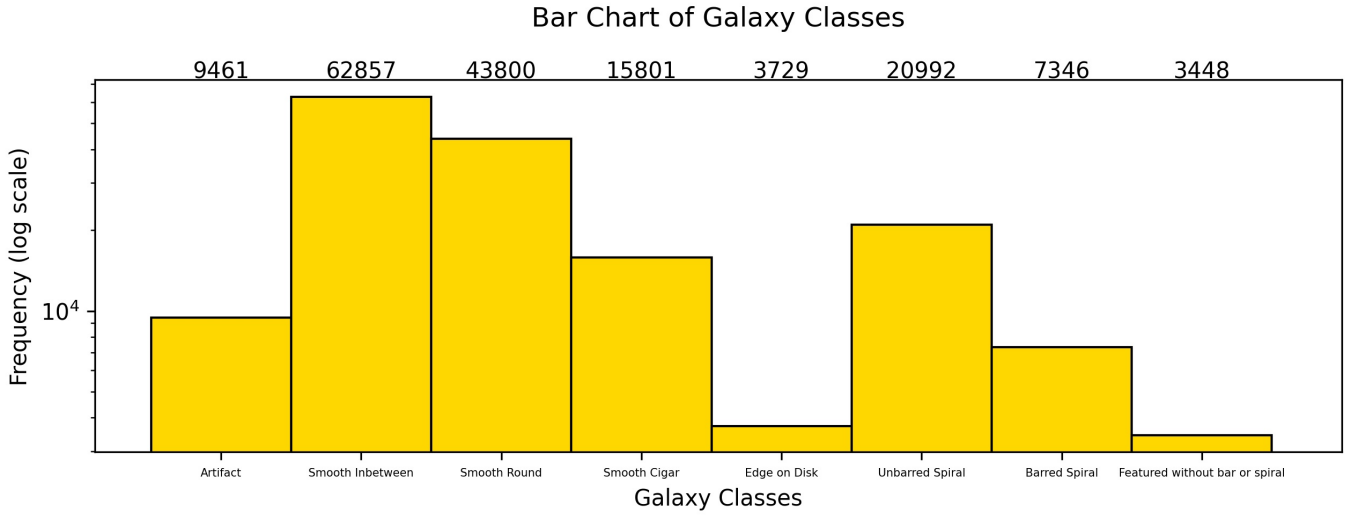
```
print(f"class labels:  {sorted(class_labels.unique())}\n")
print("Number of galaxies in each class: \n")
print(f"Artifact: {len(class_labels[class_labels == -1])} = {len(class_labels[class_labels == -1])/len(class_la
print(f"Smooth Inbetween: {len(class_labels[class_labels == 0])} = {len(class_labels[class_labels == 0])/len(cl
print(f"Smooth Round: {len(class_labels[class_labels == 1])} = {len(class_labels[class_labels == 1])/len(class_
print(f"Smooth Cigar: {len(class_labels[class_labels == 2])} = {len(class_labels[class_labels == 2])/len(class_
print(f"Edge on Disk: {len(class_labels[class_labels == 3])} = {len(class_labels[class_labels == 3])/len(class_
print(f"Unbarred Spiral: {len(class_labels[class_labels == 4])} = {len(class_labels[class_labels == 4])/len(cla
print(f"Barred Spiral: {len(class_labels[class_labels == 5])} = {len(class_labels[class_labels == 5])/len(class_
print(f"Featured without bar or spiral: {len(class_labels[class_labels == 6])} = {len(class_labels[class_labels


# ------------------------------------------------------------plot histogram of data-------------------------

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(class_labels, np.arange(9)-1.5, color='gold', ec = 'k')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('Bar Chart of Galaxy Classes', pad =24)
plt.xlabel('Galaxy Classes')
plt.ylabel('Frequency (log scale)')

class_names = ['Artifact', 'Smooth Inbetween', 'Smooth Round', 'Smooth Cigar', 'Edge on Disk', 'Unbarred Spiral

ax.tick_params(axis='x', which='major', labelsize=5)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 7])
ax.set_yscale('log')

plt.show()

#-----------------------------------------------------------------Histogram of smooth or featured---------------

smooth_or_featured = df['smooth-or-featured-gz2_semantic']

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(smooth_or_featured, np.arange(5)-0.5, color='lightgray', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('Is the galaxy smooth or featured?', pad =24)
plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')

class_names = ['Smooth', 'Featured or Disk', 'Deadlock', 'Artifact', None]

ax.set_xticklabels(class_names)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 4])

plt.show()

#-----------------------------------------------------------------Histogram of disk edge on----------------------

disk_edge_on = df['disk-edge-on-gz2_semantic']

class_names = ['-','No', 'Yes', 'Deadlock', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(disk_edge_on, np.arange(5)-0.5, color='palegreen', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')
```

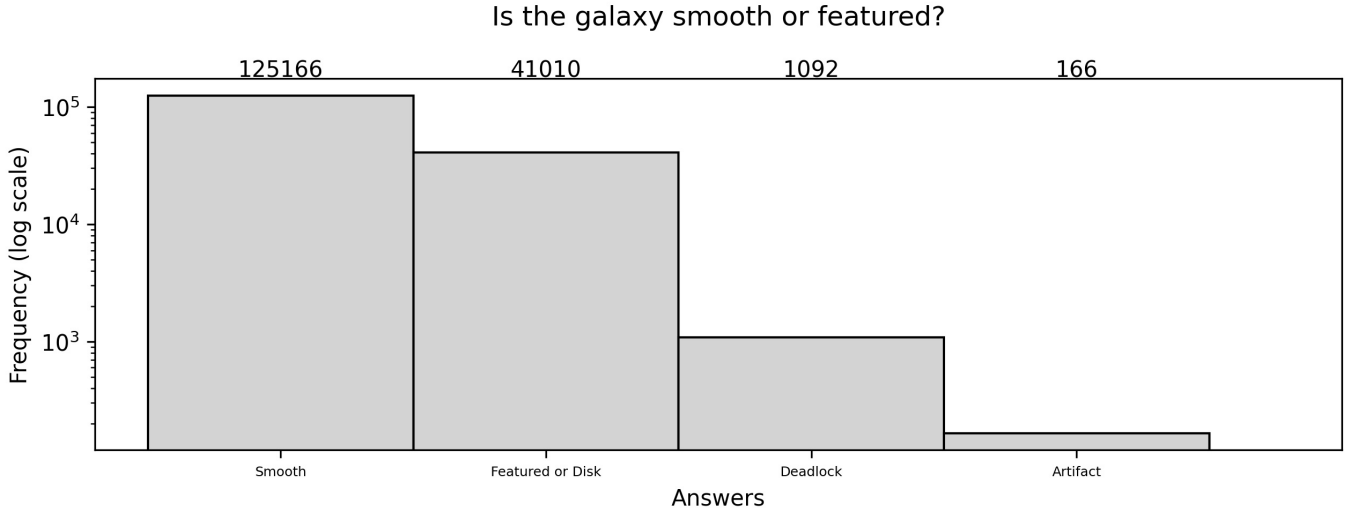
```
ax.set_title('Can the galaxy be seen as edge on?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 4])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------------Histogram of has spiral arms---

has_spiral_arms = df['has-spiral-arms-gz2_semantic']

class_names = ['-','No', 'Yes', 'Deadlock', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(has_spiral_arms, np.arange(5)-0.5, color='skyblue', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('Does the galaxy have spiral arms?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 4])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------------Histogram of bar-----------------

bar = df['bar-gz2_semantic']

class_names = ['-','No', 'Yes', 'Deadlock', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(bar, np.arange(5)-0.5, color='skyblue', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('Does the galaxy have a bar?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 4])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------histogram of bulge size----------------

bulge_size = df['bulge-size-gz2_semantic']

class_names = ['-','Dominant', 'Just Noticable', 'Obvious', 'Deadlock', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(bulge_size, np.arange(6)-0.5, color='skyblue', ec = 'k')

# Set the y-axis to log scale
ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title("Does the Galaxy have a central bulge? If so, what it's size?", pad =24)
```

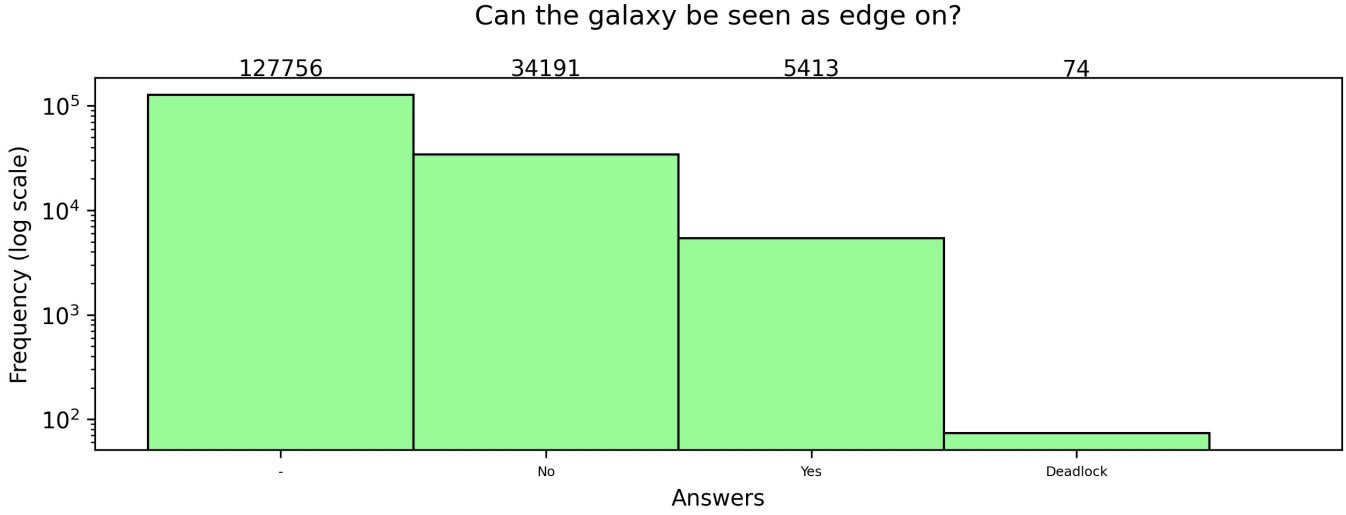
```
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 5])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-----------------------------------------------------------------histogram of something odd-----------------

something_odd = df['something-odd-gz2_semantic']

class_names = ['No','Yes', 'Deadlock', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(something_odd, np.arange(4)-0.5, color='palegreen', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('Does the image of the galaxy have something odd in it?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 3])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------------------histogram of how rounded----

how_rounded = df['how-rounded-gz2_semantic']

class_names = ['Round','In-between', '-', 'Cigar', 'Deadlock',None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(how_rounded, np.arange(6)-0.5, color='palegreen', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('How rounded is the galaxy?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 5])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-----------------------------------------------------------------------histogram of bulge shape----------------

bulge_shape = df['bulge-shape-gz2_semantic']

class_names = ['-', 'Round', 'Deadlock', 'None', 'Boxy', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(bulge_shape, np.arange(6)-0.5, color='skyblue', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('What is the shape of the bulge in the galaxy?', pad =24)
ax.set_xticklabels(class_names)
ax.tick_params(axis='x', which='major', labelsize=6)
```

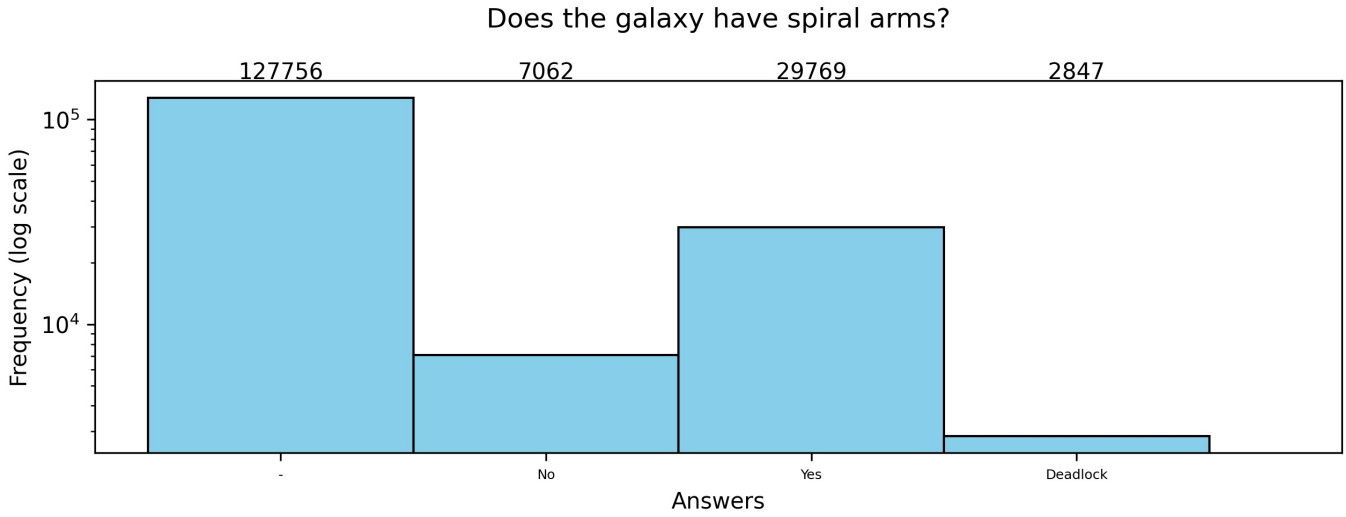
```
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 5])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------histogram of spiral winding---------------

spiral_winding = df['spiral-winding-gz2_semantic']

class_names = ['-', 'Deadlock', 'Loose', 'Medium', 'Tight', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(spiral_winding, np.arange(6)-0.5, color='plum', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('How tightly wound are the spiral arms?', pad =24)
ax.set_xticklabels(class_names, fontsize=6)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 5])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()

#-------------------------------------------------------------------------histogram of spiral arm count---------------

spiral_arm_count = df['spiral-arm-count-gz2_semantic']

class_names = ['-', '1', '2', '4', "Can't tell", '3', 'Deadlock', 'More than 4', None]

fig, ax = plt.subplots(1,1, figsize=(10,3), dpi = 300)

n, bins, patches = ax.hist(spiral_arm_count, np.arange(9)-0.5, color='plum', ec = 'k')

ax.set_yscale('log')
ax.set_xticks(bins+0.5)
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

for n, x in zip(n, bin_centers):
    # Label the frequency
    ax.annotate(str(int(n)), xy=(x, 0), xycoords=('data', 'axes fraction'),
    xytext=(0, 175), textcoords='offset points', va='top', ha='center', color = 'k')

ax.set_title('How many spiral arms are there?', pad =24)
ax.tick_params(axis='x', which='major', labelsize=6)
ax.set_xticklabels(class_names)
ax.set_xticks(ax.get_xticks()[ax.get_xticks() != 8])

plt.xlabel('Answers')
plt.ylabel('Frequency (log scale)')
plt.show()
```

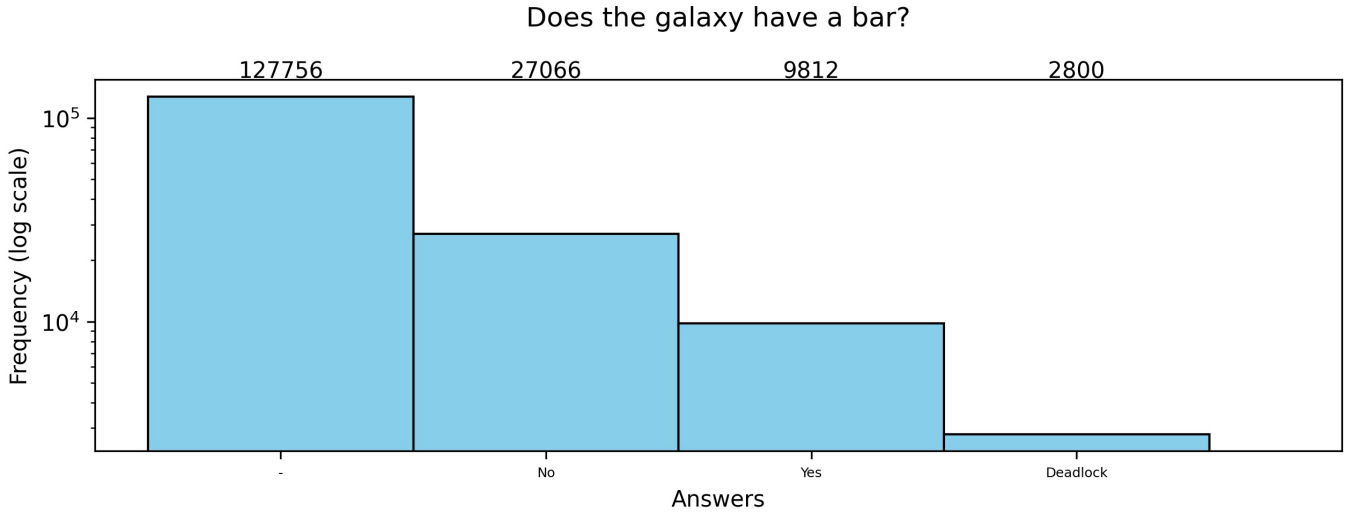
```
Number of Galaxies in the dataset: 167434

class labels:  [-1, 0, 1, 2, 3, 4, 5, 6]

Number of galaxies in each class:

Artifact: 9461 = 5.65 %
Smooth Inbetween: 62857 = 37.54 %
Smooth Round: 43800 = 26.16 %
Smooth Cigar: 15801 = 9.44 %
Edge on Disk: 3729 = 2.23 %
Unbarred Spiral: 20992 = 12.54 %
Barred Spiral: 7346 = 4.39 %
Featured without bar or spiral: 3448 = 2.06 %
```

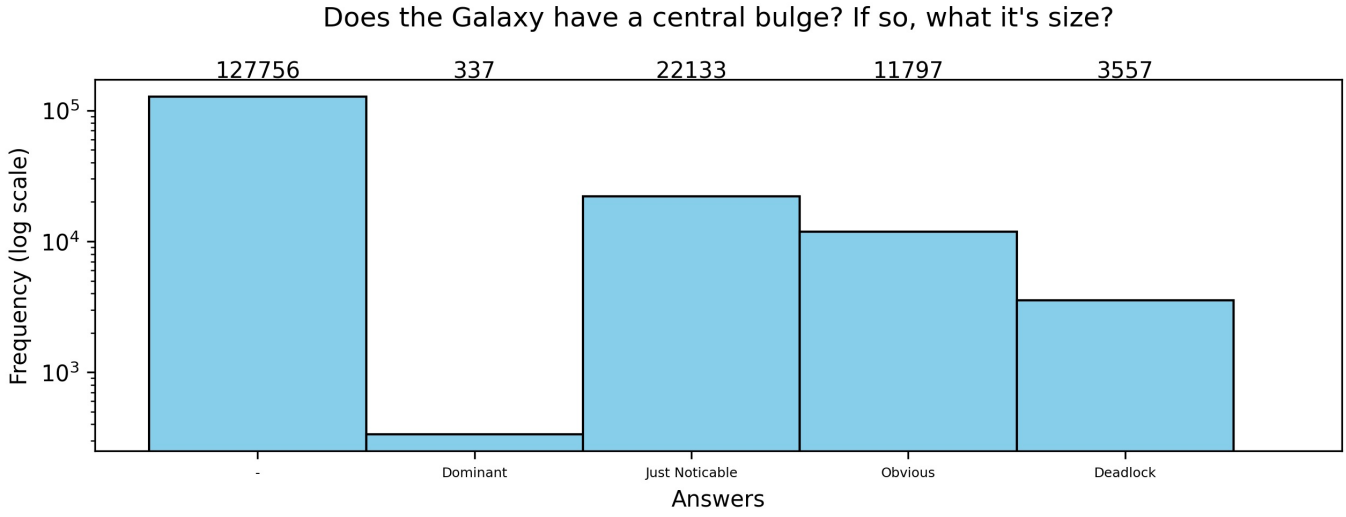

Bar Chart of Galaxy Classes
9461
62857
43800
15801
3729
20992
7346
3448
Frequency (log scale)
10^4
Artifact
Smooth Inbetween
Smooth Round
Smooth Cigar
Edge on Disk
Unbarred Spiral
Barred Spiral
Featured without bar or spiral
Galaxy Classes

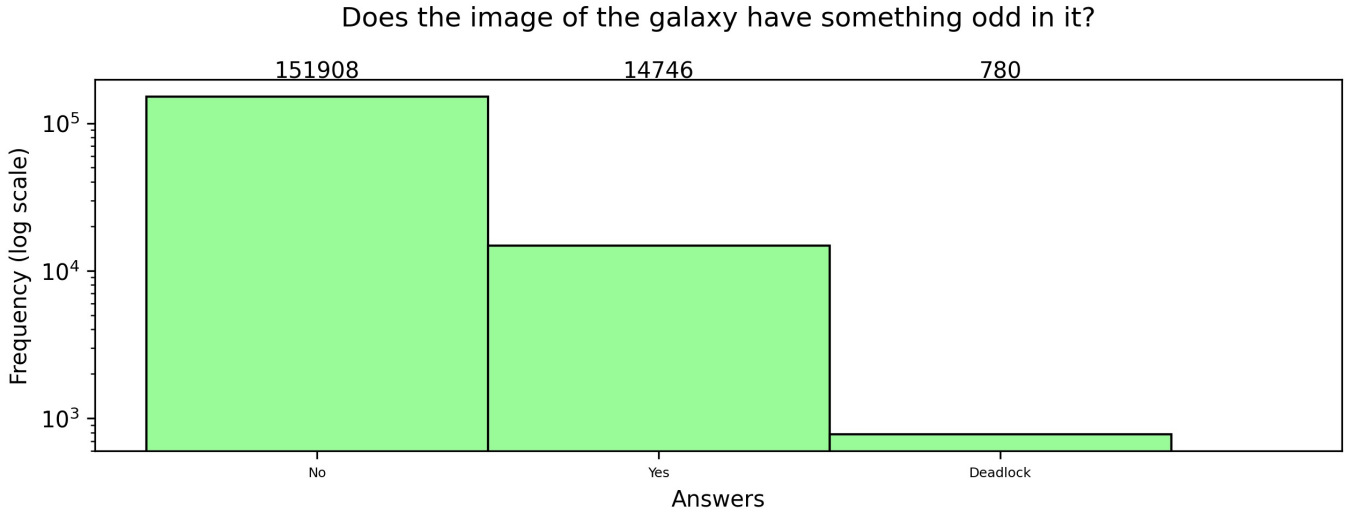

Is the galaxy smooth or featured?
125166
41010
1092
166
Frequency (log scale)
10^5
10^4
10^3
Smooth
Featured or Disk
Deadlock
Artifact
Answers

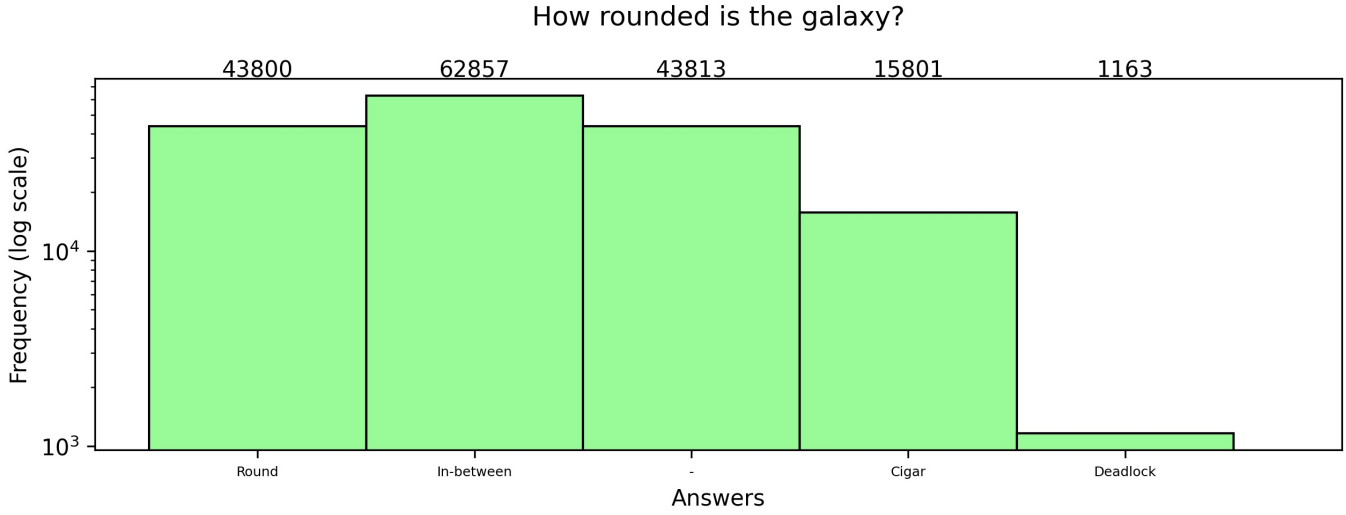

Can the galaxy be seen as edge on?
127756
34191
5413
74
Frequency (log scale)
10^5
10^4
10^3
10^2
-
No
Yes
Deadlock
Answers

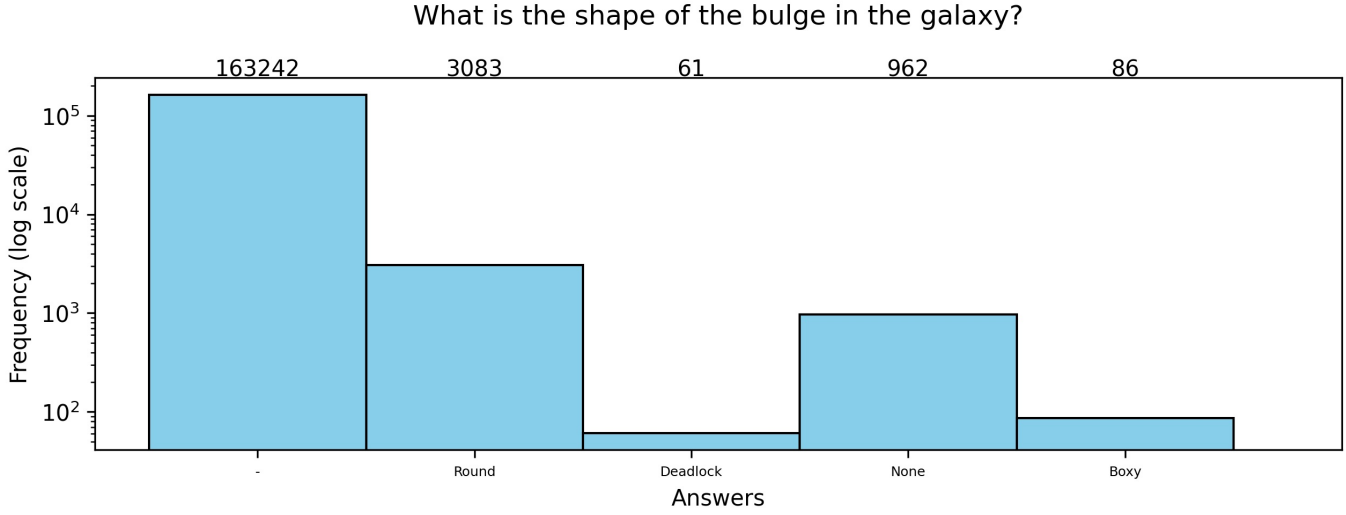

Does the galaxy have spiral arms?
127756
7062
29769
2847
10^5
10^4
Frequency (log scale)
-
No
Yes
Deadlock
Answers

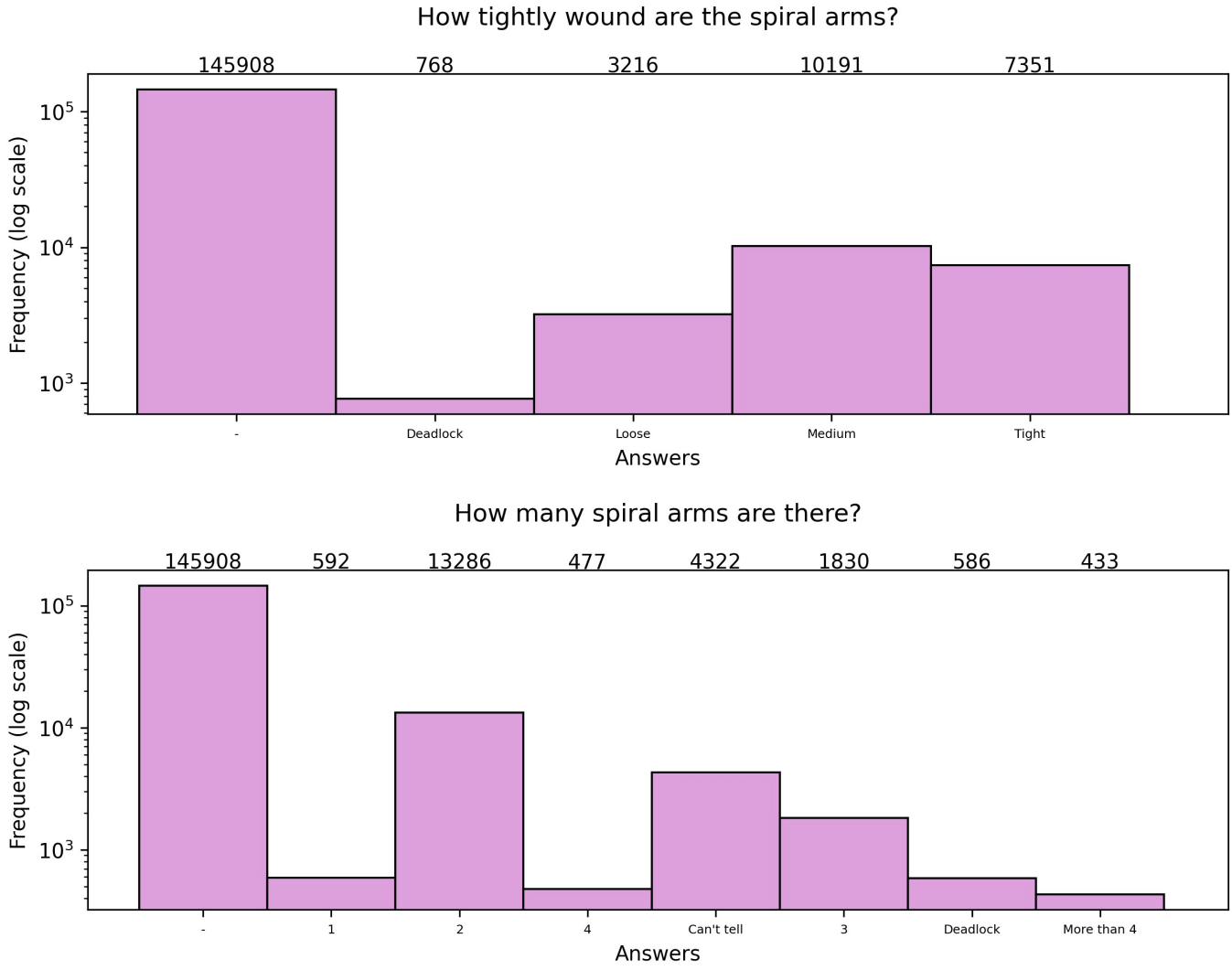

Does the galaxy have a bar?
127756
27066
9812
2800
10^5
10^4
Frequency (log scale)
-
No
Yes
Deadlock
Answers


Does the Galaxy have a central bulge? If so, what it's size?
127756
337
22133
11797
3557
10^5
10^4
10^3
Frequency (log scale)
-
Dominant
Just Noticable
Obvious
Deadlock
Answers


Does the image of the galaxy have something odd in it?
151908
14746
780
Frequency (log scale)
10^5
10^4
10^3
No
Yes
Deadlock
Answers


How rounded is the galaxy?
43800
62857
43813
15801
1163
Frequency (log scale)
10^4
10^3
Round
In-between
-
Cigar
Deadlock
Answers


What is the shape of the bulge in the galaxy?
163242
3083
61
962
86
Frequency (log scale)
10^5
10^4
10^3
10^2
-
Round
Deadlock
None
Boxy
Answers



Distribution of answers for each question asked in the galaxy zoo decision tree.

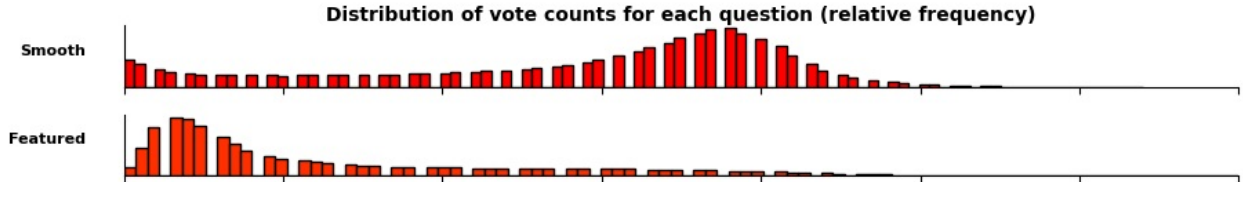
1/04/2024 Malachy - galaxy zoo dataset analysis

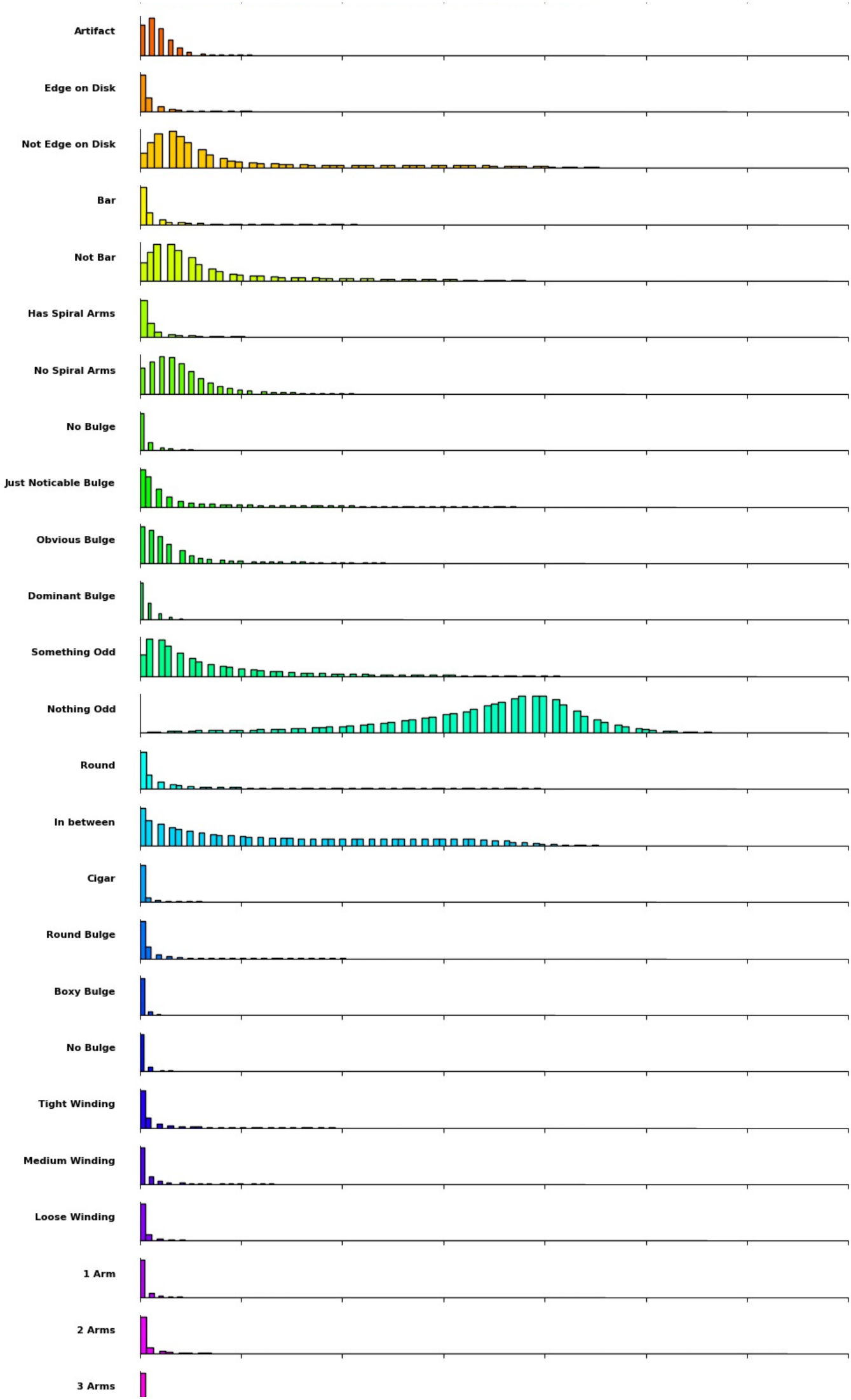
In [ ]:

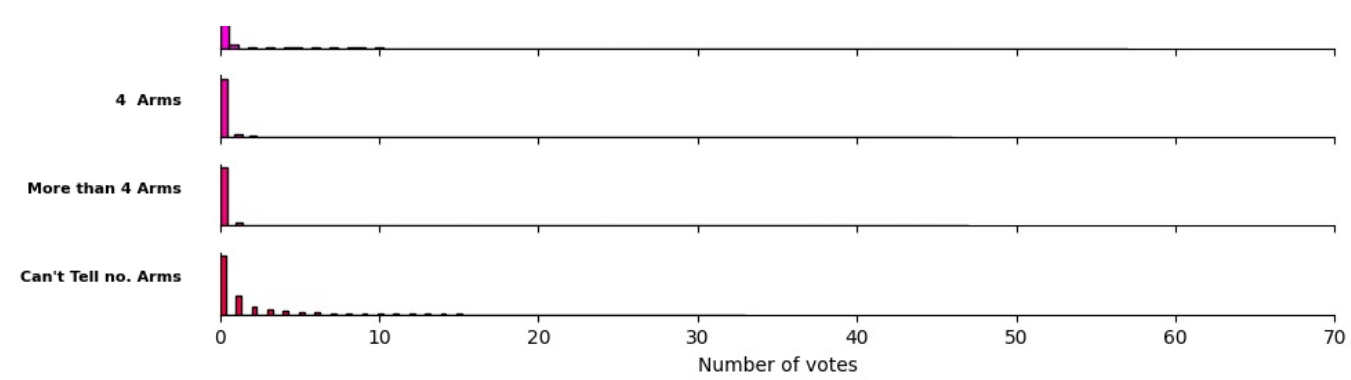
```
import pandas as pd
import os
import matplotlib.pyplot as plt

run = True

if run:
    dataset_root = 'galaxyzoo2-dataset/'
    catalog_name = r'gz2_train_catalog.parquet'

    parquet_file_path = os.path.join(dataset_root, catalog_name)

    # Read the Parquet file
    df = pd.read_parquet(parquet_file_path)

    #Plot answers to every classifying question in the decision tree (index smooth-or-featured-gz2_smooth to sp

    smooth = df['smooth-or-featured-gz2_smooth']
    featured = df['smooth-or-featured-gz2_featured-or-disk']
    artifact = df['smooth-or-featured-gz2_artifact']
    edge_on_yes = df['disk-edge-on-gz2_yes']
    edge_on_no = df['disk-edge-on-gz2_no']
    bar_yes = df['bar-gz2_yes']
    bar_no = df['bar-gz2_no']
    has_spiral_arms_yes = df['has-spiral-arms-gz2_yes']
```

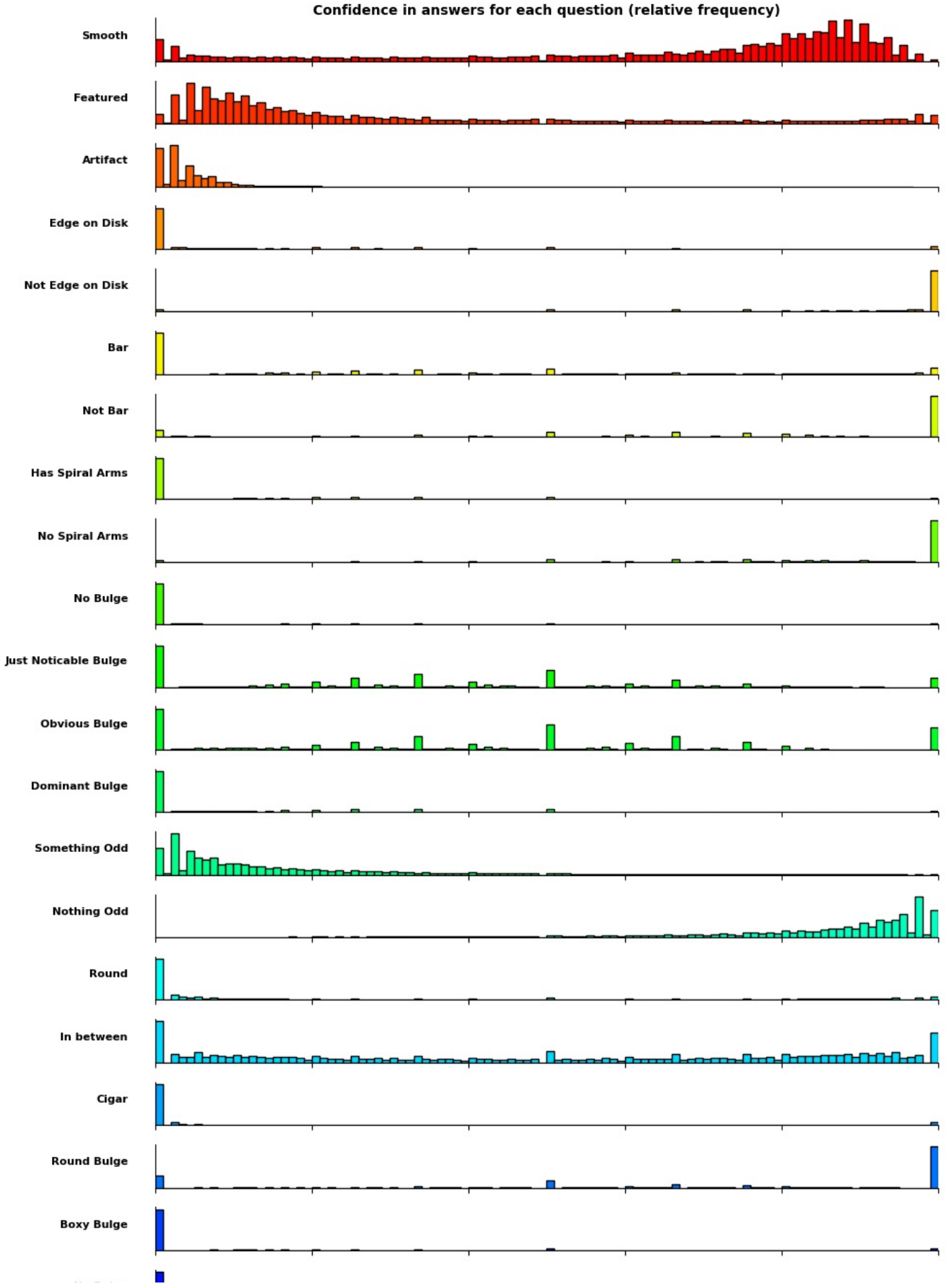
```
has_spiral_arms_no = df['has-spiral-arms-gz2_no']
bulge_no = df['bulge-size-gz2_no']
bulge_just_noticable = df['bulge-size-gz2_just-noticeable']
bulge_obvious = df['bulge-size-gz2_obvious']
bulge_dominant = df['bulge-size-gz2_dominant']
something_odd_yes = df['something-odd-gz2_yes']
something_odd_no = df['something-odd-gz2_no']
round = df['how-rounded-gz2_round']
in_between = df['how-rounded-gz2_in-between']
cigar = df['how-rounded-gz2_cigar']
bulge_round = df['bulge-shape-gz2_round']
bulge_boxy = df['bulge-shape-gz2_boxy']
bulge_no_bulge = df['bulge-shape-gz2_no-bulge']
spiral_winding_tight = df['spiral-winding-gz2_tight']
spiral_winding_medium = df['spiral-winding-gz2_medium']
spiral_winding_loose = df['spiral-winding-gz2_loose']
spiral_count_1 = df['spiral-arm-count-gz2_1']
spiral_count_2 = df['spiral-arm-count-gz2_2']
spiral_count_3 = df['spiral-arm-count-gz2_3']
spiral_count_4 = df['spiral-arm-count-gz2_4']
spiral_count_more_than_4 = df['spiral-arm-count-gz2_more-than-4']
spiral_count_cant_tell = df['spiral-arm-count-gz2_cant-tell']

data = [smooth, featured, artifact, edge_on_yes, edge_on_no, bar_yes, bar_no, has_spiral_arms_yes, has_spira
        bulge_obvious, bulge_dominant, something_odd_yes, something_odd_no, round, in_between, cigar, bulge_
        spiral_winding_medium, spiral_winding_loose, spiral_count_1, spiral_count_2, spiral_count_3, spiral_

class_names = ['Smooth', 'Featured', 'Artifact', 'Edge on Disk', 'Not Edge on Disk', 'Bar', 'Not Bar', 'Has
               'Obvious Bulge', 'Dominant Bulge', 'Something Odd', 'Nothing Odd', 'Round', 'In between', '
               'Medium Winding', 'Loose Winding', '1 Arm', '2 Arms', '3 Arms', '4  Arms', 'More than 4 Arms

fig, axs = plt.subplots(len(data), 1, figsize=(10, 20), sharex=True)

cmap = plt.cm.hsv

for i, ax in enumerate(axs):
    if i == 0:
        ax.set_title('Distribution of vote counts for each question (relative frequency)', fontsize=10, colo

    ax.hist(data[i], bins=100, color=cmap(i/len(data)), edgecolor='black')  # Add edgecolor='black'
    ax.set_ylabel(f'{class_names[i]}', rotation=0, labelpad=20, fontsize=8, fontweight='bold', ha='right')

    # Remove spines and ticks
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(True)
    ax.spines['left'].set_visible(True)
    ax.tick_params(axis='y', which='both', direction='out')
    ax.set_xlim(0, 70)
    ax.set_yticks([])  # Add this line to remove y ticks

plt.xlabel('Number of votes')

plt.tight_layout()  # Add this line to remove gaps between subplots
plt.show()

smooth_frac = df['smooth-or-featured-gz2_smooth_fraction']
disk_frac = df['smooth-or-featured-gz2_featured-or-disk_fraction']
artifact_frac = df['smooth-or-featured-gz2_artifact_fraction']
disk_edge_on_yes_frac = df['disk-edge-on-gz2_yes_fraction']
disk_edge_on_no_frac = df['disk-edge-on-gz2_no_fraction']
has_spiral_arms_yes_frac = df['has-spiral-arms-gz2_yes_fraction']
has_spiral_arms_no_frac = df['has-spiral-arms-gz2_no_fraction']
bar_yes_frac = df['bar-gz2_yes_fraction']
bar_no_frac = df['bar-gz2_no_fraction']
bulge_size_no_frac = df['bulge-size-gz2_no_fraction']
bulge_size_just_noticable_frac = df['bulge-size-gz2_just-noticeable_fraction']
bulge_size_obvious_frac = df['bulge-size-gz2_obvious_fraction']
bulge_size_dominant_frac = df['bulge-size-gz2_dominant_fraction']
something_odd_yes_frac = df['something-odd-gz2_yes_fraction']
something_odd_no_frac = df['something-odd-gz2_no_fraction']
how_rounded_round_frac = df['how-rounded-gz2_round_fraction']
how_rounded_in_between_frac = df['how-rounded-gz2_in-between_fraction']
how_rounded_cigar_frac = df['how-rounded-gz2_cigar_fraction']
bulge_shape_round_frac = df['bulge-shape-gz2_round_fraction']
bulge_shape_boxy_frac = df['bulge-shape-gz2_boxy_fraction']
bulge_shape_no_bulge_frac = df['bulge-shape-gz2_no-bulge_fraction']
spiral_winding_tight_frac = df['spiral-winding-gz2_tight_fraction']
spiral_winding_medium_frac = df['spiral-winding-gz2_medium_fraction']
spiral_winding_loose_frac = df['spiral-winding-gz2_loose_fraction']
spiral_arm_count_1_frac = df['spiral-arm-count-gz2_1_fraction']
spiral_arm_count_2_frac = df['spiral-arm-count-gz2_2_fraction']
spiral_arm_count_3_frac = df['spiral-arm-count-gz2_3_fraction']
```

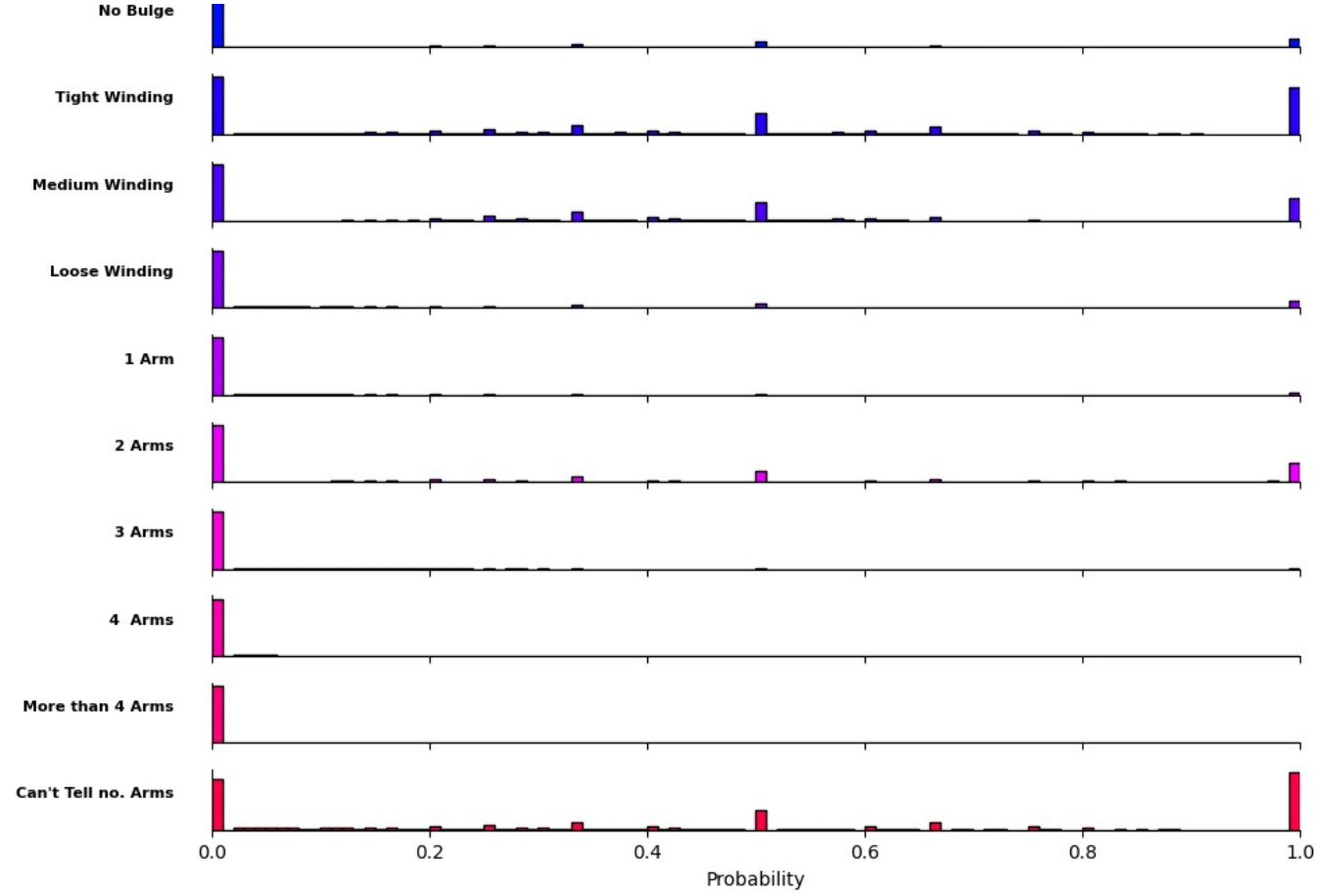
```
spiral_arm_count_4_frac = df['spiral-arm-count-gz2_4_fraction']
spiral_arm_count_more_than_4_frac = df['spiral-arm-count-gz2_more-than-4_fraction']
spiral_arm_count_cant_tell_frac = df['spiral-arm-count-gz2_cant-tell_fraction']

data = [smooth_frac, disk_frac, artifact_frac, disk_edge_on_yes_frac, disk_edge_on_no_frac, has_spiral_arms_
        bulge_size_no_frac, bulge_size_just_noticable_frac, bulge_size_obvious_frac, bulge_size_dominant_fra
        how_rounded_in_between_frac, how_rounded_cigar_frac, bulge_shape_round_frac, bulge_shape_boxy_frac,
        spiral_winding_loose_frac, spiral_arm_count_1_frac, spiral_arm_count_2_frac, spiral_arm_count_3_frac

class_names = ['Smooth', 'Featured', 'Artifact', 'Edge on Disk', 'Not Edge on Disk', 'Bar', 'Not Bar', 'Has
               'Obvious Bulge', 'Dominant Bulge', 'Something Odd', 'Nothing Odd', 'Round', 'In between', '
               'Medium Winding', 'Loose Winding', '1 Arm', '2 Arms', '3 Arms', '4  Arms', 'More than 4 Arms

fig, axs = plt.subplots(len(data), 1, figsize=(10, 20), sharex=True)

cmap = plt.cm.hsv

for i, ax in enumerate(axs):
    if i == 0:
        ax.set_title('Confidence in answers for each question (relative frequency)', fontsize=10, color='k'

    ax.hist(data[i], bins=100, color=cmap(i/len(data)), edgecolor='black')  # Add edgecolor='black'
    ax.set_ylabel(f'{class_names[i]}', rotation=0, labelpad=20, fontsize=8, fontweight='bold', ha='right')

    # Remove spines and ticks
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(True)
    ax.spines['left'].set_visible(True)
    ax.tick_params(axis='y', which='both', direction='out')
    ax.set_yticks([])  # Add this line to remove y ticks
    ax.set_xlim(0, 1)

plt.xlabel('Probability')

plt.tight_layout()  # Add this line to remove gaps between subplots
plt.show()

smooth_or_featured_total = df['smooth-or-featured-gz2_total-votes']
disk_edge_on_total = df['disk-edge-on-gz2_total-votes']
spiral_arms_total = df['has-spiral-arms-gz2_total-votes']
bar_total = df['bar-gz2_total-votes']
bulge_size_total = df['bulge-size-gz2_total-votes']
something_odd_total = df['something-odd-gz2_total-votes']
how_rounded_total = df['how-rounded-gz2_total-votes']
bulge_shape_total = df['bulge-shape-gz2_total-votes']
spiral_winding_total = df['spiral-winding-gz2_total-votes']
spiral_arm_count_total = df['spiral-arm-count-gz2_total-votes']

data = [smooth_or_featured_total, disk_edge_on_total, spiral_arms_total, bar_total, bulge_size_total, somet

class_names = ['Smooth or Featured', 'Disk Edge On', 'Has Spiral Arms', 'Bar', 'Bulge Size', 'Something Odd

fig, axs = plt.subplots(len(data), 1, figsize=(10, 20), sharex=True)

cmap = plt.cm.hsv

for i, ax in enumerate(axs):
    if i == 0:
        ax.set_title('Toal Votes for each question', fontsize=10, color='k', ha='center', va='center', font

    ax.hist(data[i], bins=100, color=cmap(i/len(data)), edgecolor='black')  # Add edgecolor='black'
    ax.set_ylabel(f'{class_names[i]}', rotation=0, labelpad=20, fontsize=8, fontweight='bold', ha='right')

    # Remove spines and ticks
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(True)
    ax.spines['left'].set_visible(True)
    ax.tick_params(axis='y', which='both', direction='out')
    ax.set_xlim(0, 70)

plt.xlabel('Number of votes')

plt.tight_layout()  # Add this line to remove gaps between subplots
plt.show()
```

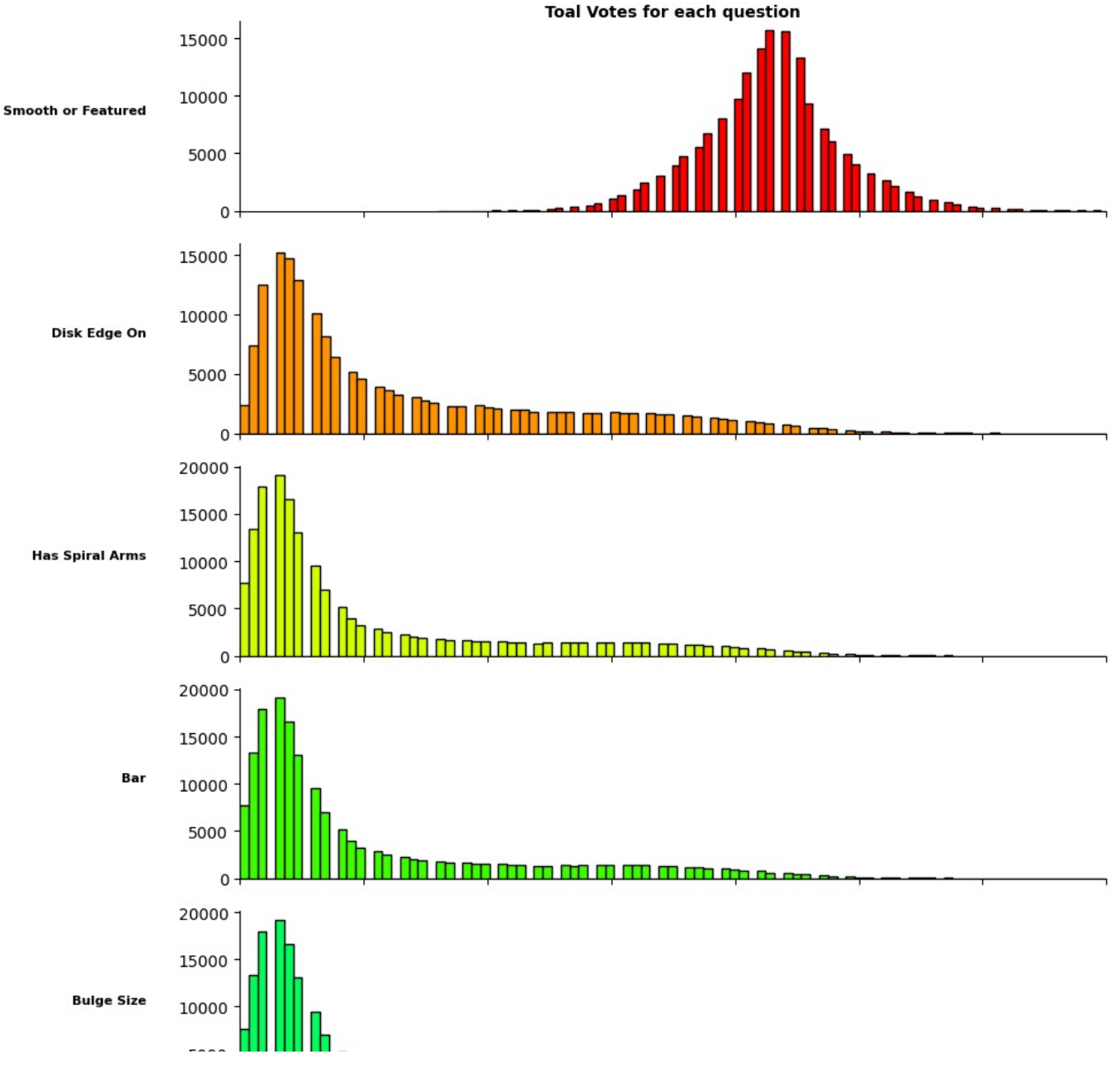


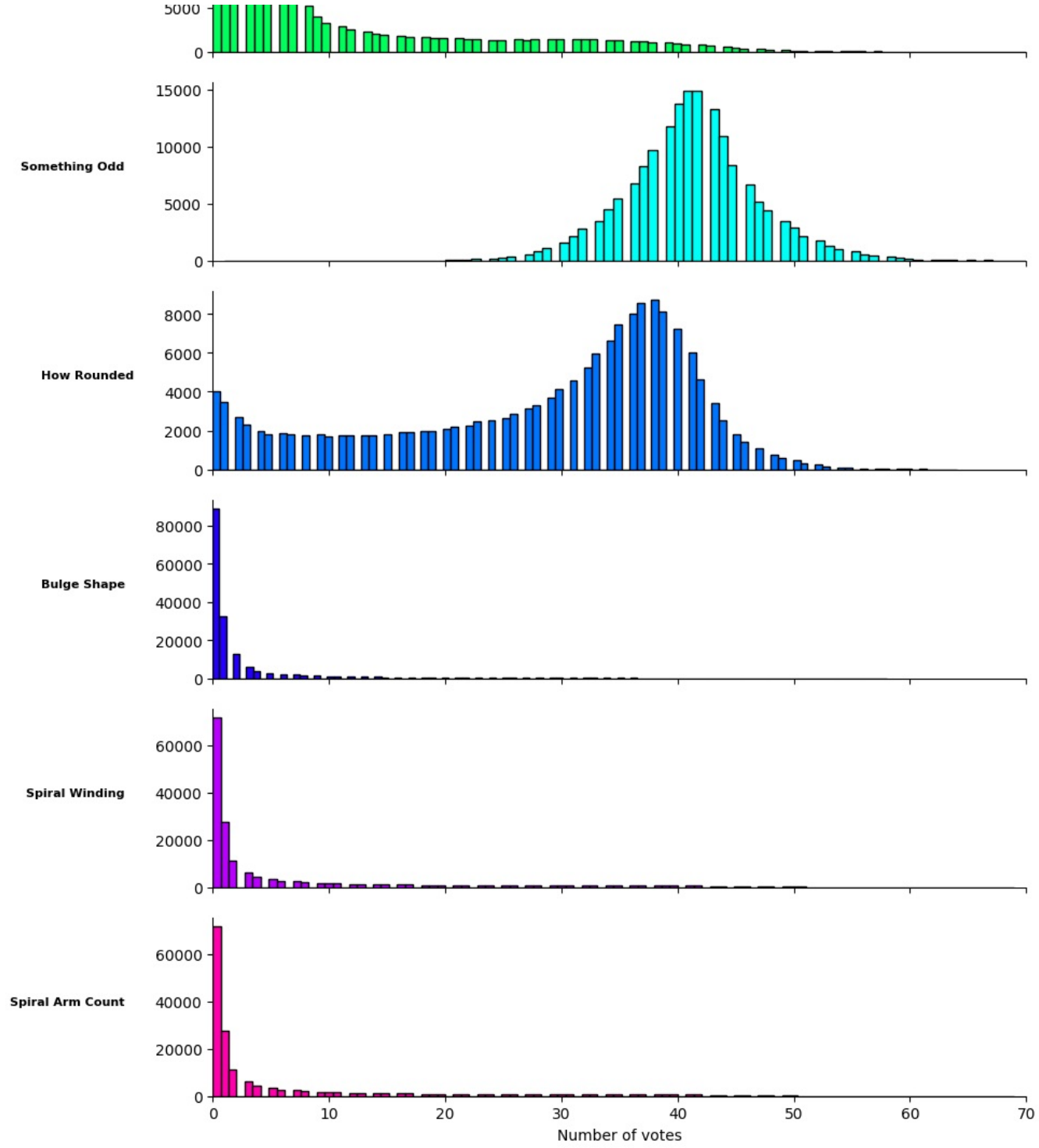

Artifact
Edge on Disk
Not Edge on Disk
Bar
Not Bar
Has Spiral Arms
No Spiral Arms
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge
Something Odd
Nothing Odd
Round
In between
Cigar
Round Bulge
Boxy Bulge
No Bulge
Tight Winding
Medium Winding
Loose Winding
1 Arm
2 Arms
3 Arms


4 Arms
More than 4 Arms
Can't Tell no. Arms
0
10
20
30
40
50
60
70
Number of votes


Confidence in answers for each question (relative frequency)
Smooth
Featured
Artifact
Edge on Disk
Not Edge on Disk
Bar
Not Bar
Has Spiral Arms
No Spiral Arms
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge
Something Odd
Nothing Odd
Round
In between
Cigar
Round Bulge
Boxy Bulge


No Bulge
Tight Winding
Medium Winding
Loose Winding
1 Arm
2 Arms
3 Arms
4 Arms
More than 4 Arms
Can't Tell no. Arms
0.0
0.2
0.4
0.6
0.8
1.0
Probability


Toal Votes for each question
Smooth or Featured
Disk Edge On
Has Spiral Arms
Bar
Bulge Size
0
5000
10000
15000
20000

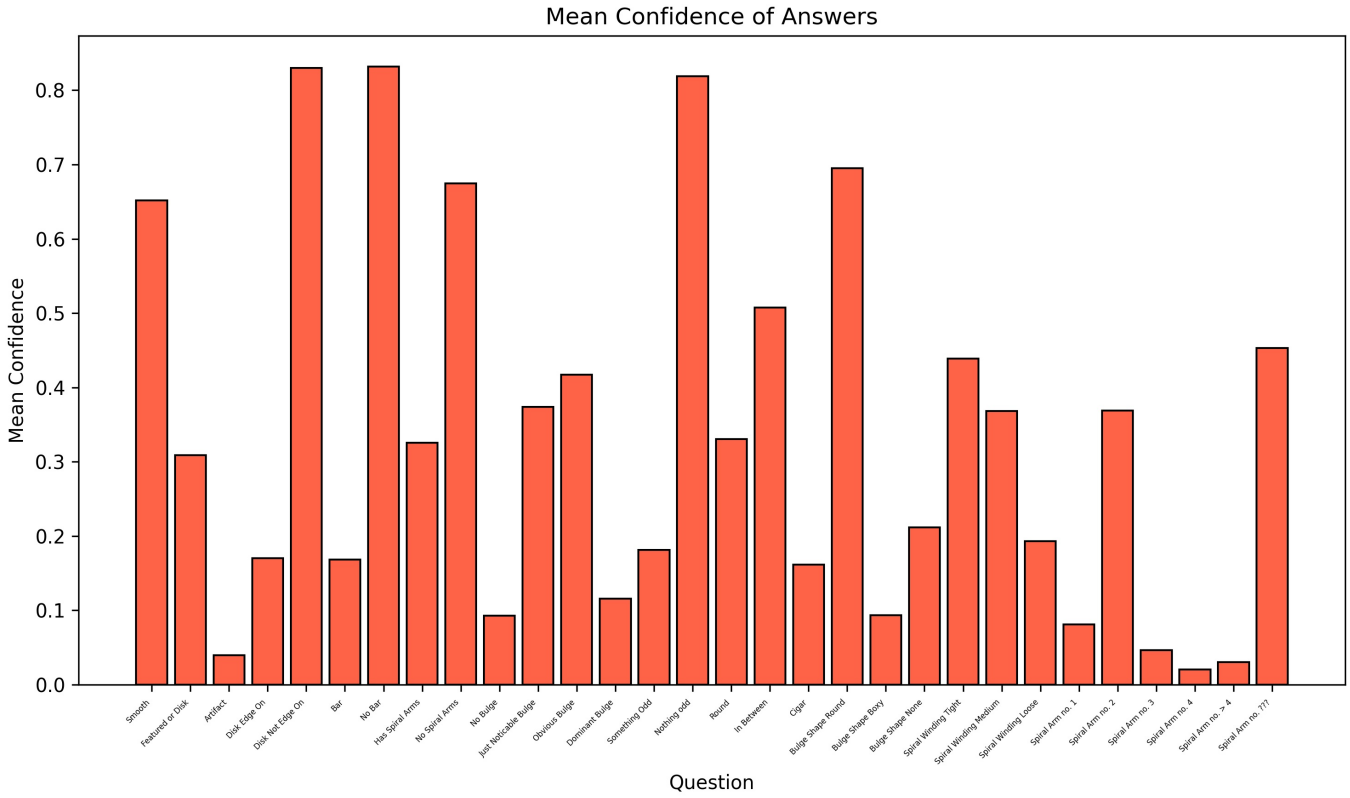


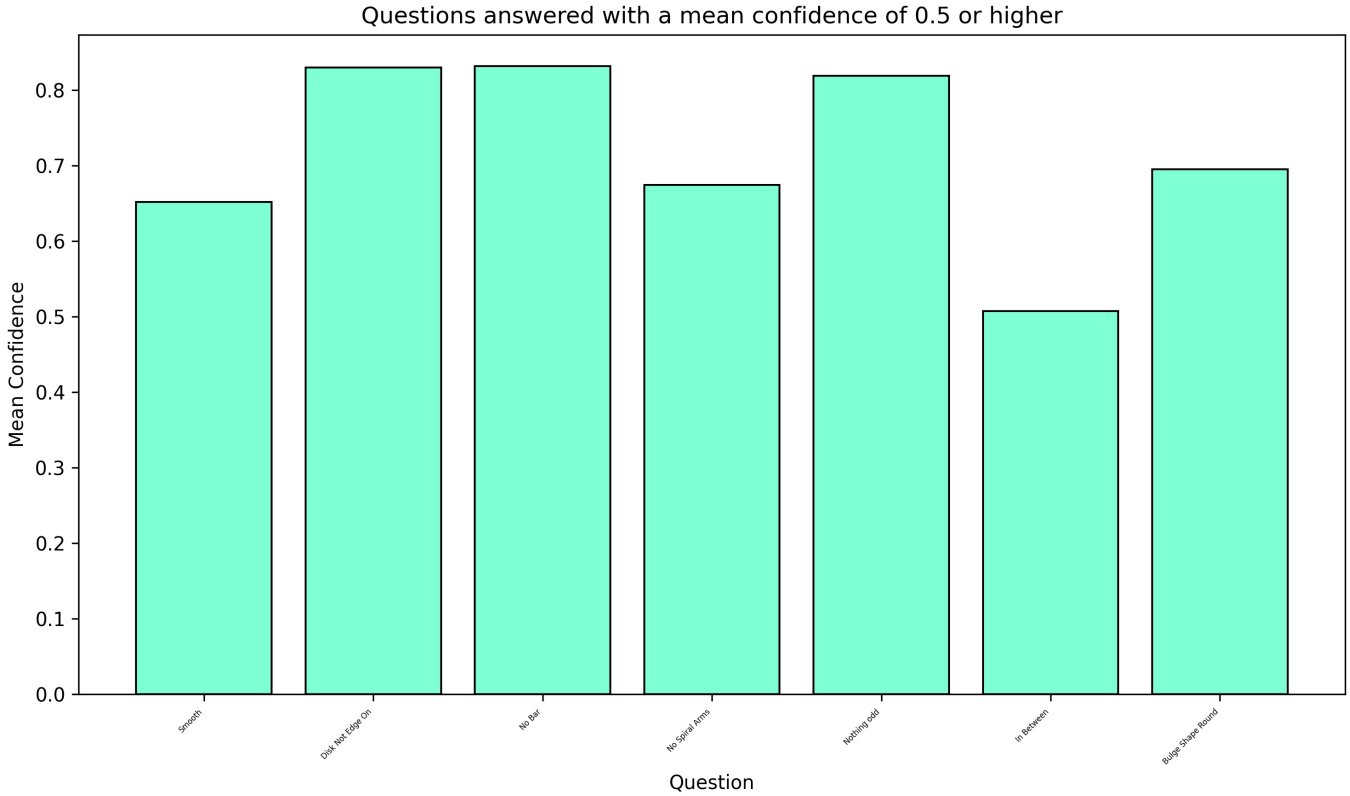
Distribution of different metrics in the dataset for each feature, i.e vote_fraction, total_votes, vote_counts...

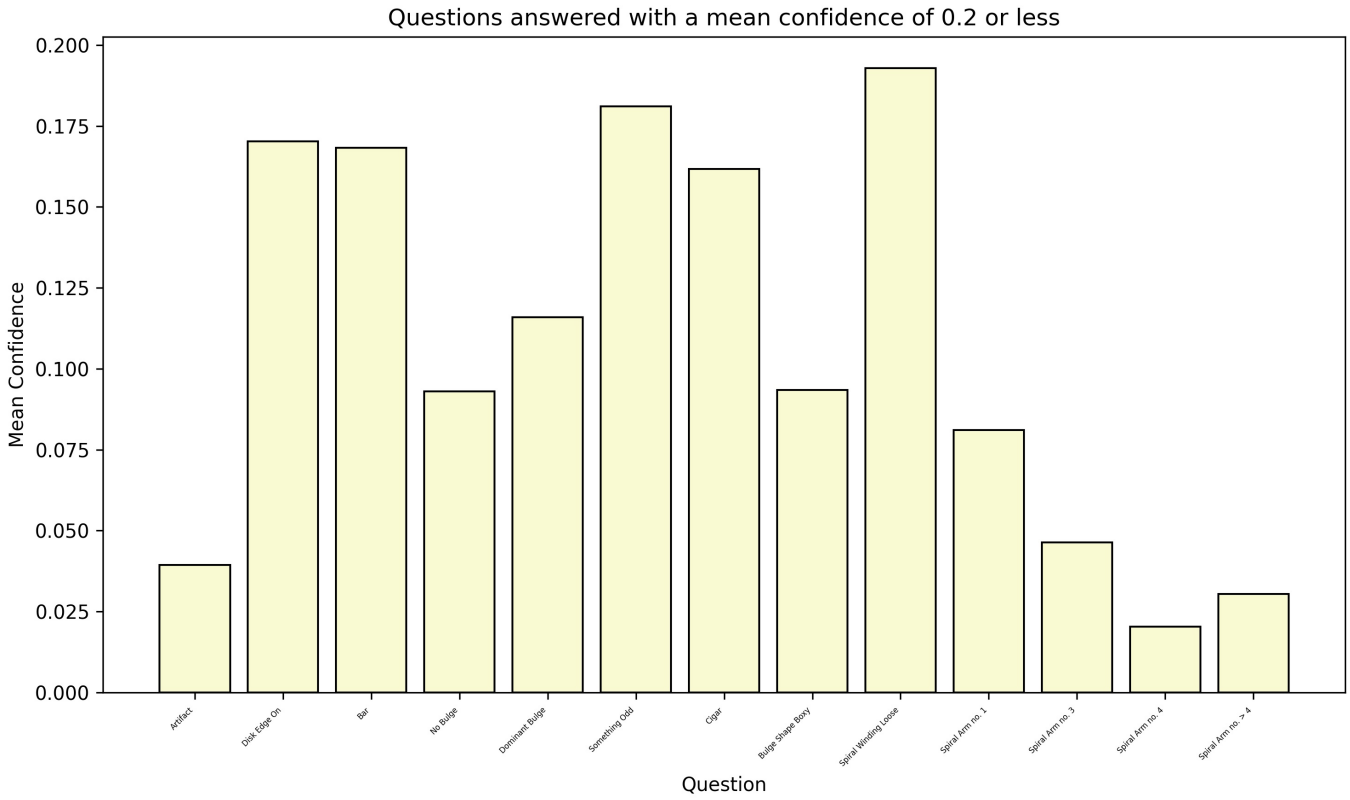
2/2/2024 - Malachy - mean confidence in answers to questions in dataset:

In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt


dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'
parquet_file_path = os.path.join(dataset_root, catalog_name)

# Read the Parquet file
df = pd.read_parquet(parquet_file_path)

mean_confidence_smooth_or_featured_smooth = df['smooth-or-featured-gz2_smooth_fraction'].mean()
mean_confidence_smooth_or_featured_featured = df['smooth-or-featured-gz2_featured-or-disk_fraction'].mean()
mean_confidence_smooth_or_featured_artifact = df['smooth-or-featured-gz2_artifact_fraction'].mean()
mean_confidence_disk_edge_on_yes = df['disk-edge-on-gz2_yes_fraction'].mean()
mean_confidence_disk_edge_on_no = df['disk-edge-on-gz2_no_fraction'].mean()
mean_confidence_bar_yes = df['bar-gz2_yes_fraction'].mean()
mean_confidence_bar_no = df['bar-gz2_no_fraction'].mean()
mean_confidence_has_spiral_arms_yes = df['has-spiral-arms-gz2_yes_fraction'].mean()
mean_confidence_has_spiral_arms_no = df['has-spiral-arms-gz2_no_fraction'].mean()
```

```
mean_confidence_bulge_size_no = df['bulge-size-gz2_no_fraction'].mean()
mean_confidence_bulge_size_just_noticable = df['bulge-size-gz2_just-noticeable_fraction'].mean()
mean_confidence_bulge_size_obvious = df['bulge-size-gz2_obvious_fraction'].mean()
mean_confidence_bulge_size_dominant = df['bulge-size-gz2_dominant_fraction'].mean()
mean_confidence_something_odd_yes = df['something-odd-gz2_yes_fraction'].mean()
mean_confidence_something_odd_no = df['something-odd-gz2_no_fraction'].mean()
mean_confidence_how_rounded_round = df['how-rounded-gz2_round_fraction'].mean()
mean_confidence_how_rounded_in_between = df['how-rounded-gz2_in-between_fraction'].mean()
mean_confidence_how_rounded_cigar = df['how-rounded-gz2_cigar_fraction'].mean()
mean_confidence_bulge_shape_round = df['bulge-shape-gz2_round_fraction'].mean()
mean_confidence_bulge_shape_boxy = df['bulge-shape-gz2_boxy_fraction'].mean()
mean_confidence_bulge_shape_no_bulge = df['bulge-shape-gz2_no-bulge_fraction'].mean()
mean_confidence_spiral_winding_tight = df['spiral-winding-gz2_tight_fraction'].mean()
mean_confidence_spiral_winding_medium = df['spiral-winding-gz2_medium_fraction'].mean()
mean_confidence_spiral_winding_loose = df['spiral-winding-gz2_loose_fraction'].mean()
mean_confidence_spiral_arm_count_1 = df['spiral-arm-count-gz2_1_fraction'].mean()
mean_confidence_spiral_arm_count_2 = df['spiral-arm-count-gz2_2_fraction'].mean()
mean_confidence_spiral_arm_count_3 = df['spiral-arm-count-gz2_3_fraction'].mean()
mean_confidence_spiral_arm_count_4 = df['spiral-arm-count-gz2_4_fraction'].mean()
mean_confidence_spiral_arm_count_more_than_4 = df['spiral-arm-count-gz2_more-than-4_fraction'].mean()
mean_confidence_spiral_arm_count_cant_tell = df['spiral-arm-count-gz2_cant-tell_fraction'].mean()

data = [mean_confidence_smooth_or_featured_smooth, mean_confidence_smooth_or_featured_featured, mean_confidence_
        mean_confidence_disk_edge_on_yes, mean_confidence_disk_edge_on_no, mean_confidence_bar_yes, mean_confid
        mean_confidence_has_spiral_arms_no, mean_confidence_bulge_size_no, mean_confidence_bulge_size_just_noti
        mean_confidence_bulge_size_dominant, mean_confidence_something_odd_yes, mean_confidence_something_odd_n
        mean_confidence_how_rounded_in_between, mean_confidence_how_rounded_cigar, mean_confidence_bulge_shape_
        mean_confidence_bulge_shape_no_bulge, mean_confidence_spiral_winding_tight, mean_confidence_spiral_windi
        mean_confidence_spiral_arm_count_1, mean_confidence_spiral_arm_count_2, mean_confidence_spiral_arm_coun
        mean_confidence_spiral_arm_count_more_than_4, mean_confidence_spiral_arm_count_cant_tell]

class_names = ['Smooth', 'Featured or Disk', 'Artifact', 'Disk Edge On', 'Disk Not Edge On', 'Bar', 'No Bar', '
               'Just Noticable Bulge', 'Obvious Bulge', 'Dominant Bulge', 'Something Odd', 'Nothing odd', 'Roun
               'Cigar', 'Bulge Shape Round', 'Bulge Shape Boxy', 'Bulge Shape None', 'Spiral Winding Tight', 'Sp
               'Spiral Arm no. 1', 'Spiral Arm no. 2', 'Spiral Arm no. 3', 'Spiral Arm no. 4', 'Spiral Arm no. 

#plot the mean confidence of answers as a bar chart

fig, ax = plt.subplots(1,1, figsize=(10,6), dpi = 300)

bars = ax.bar(class_names, data, color='tomato', edgecolor='black')

ax.set_ylabel('Mean Confidence')
ax.set_xlabel('Question')
ax.set_title('Mean Confidence of Answers')
plt.xticks(rotation=(45), ha='right', fontsize=4)
plt.tight_layout()
plt.show()

filtered_data = [d for d in data if d >= 0.5]
filtered_class_names = [cn for d, cn in zip(data, class_names) if d > 0.5]

fig, ax = plt.subplots(1, 1, figsize=(10, 6), dpi=300)

bars = ax.bar(filtered_class_names, filtered_data, color='aquamarine', edgecolor='black')

ax.set_ylabel('Mean Confidence')
ax.set_xlabel('Question')
ax.set_title('Questions answered with a mean confidence of 0.5 or higher')
plt.xticks(rotation=(45), ha='right', fontsize=4)
plt.tight_layout()
plt.show()

filtered_data = [d for d in data if d <= 0.2]
filtered_class_names = [cn for d, cn in zip(data, class_names) if d <= 0.2]

fig, ax = plt.subplots(1, 1, figsize=(10, 6), dpi=300)

bars = ax.bar(filtered_class_names, filtered_data, color='lightgoldenrodyellow', edgecolor='black')

ax.set_ylabel('Mean Confidence')
ax.set_xlabel('Question')
ax.set_title('Questions answered with a mean confidence of 0.2 or less')
plt.xticks(rotation=(45), ha='right', fontsize=4)
plt.tight_layout()
plt.show()
```


Mean Confidence of Answers
Mean Confidence
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Smooth
Featured or Disk
Artifact
Disk Edge On
Disk Not Edge On
Bar
No Bar
Has Spiral Arms
No Spiral Arms
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge
Something Odd
Nothing odd
Round
In Between
Cigar
Bulge Shape Round
Bulge Shape Boxy
Bulge Shape None
Spiral Winding Tight
Spiral Winding Medium
Spiral Winding Loose
Spiral Arm no. 1
Spiral Arm no. 2
Spiral Arm no. 3
Spiral Arm no. 4
Spiral Arm no. > 4
Spiral Arm no. ???
Question


Questions answered with a mean confidence of 0.5 or higher
Mean Confidence
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Smooth
Disk Not Edge On
No Bar
No Spiral Arms
Nothing odd
In Between
Bulge Shape Round
Question



1/ mean confidence in answers to questions in the dataset. 2/ most confidently answered questions 3/ least confidently answered questions

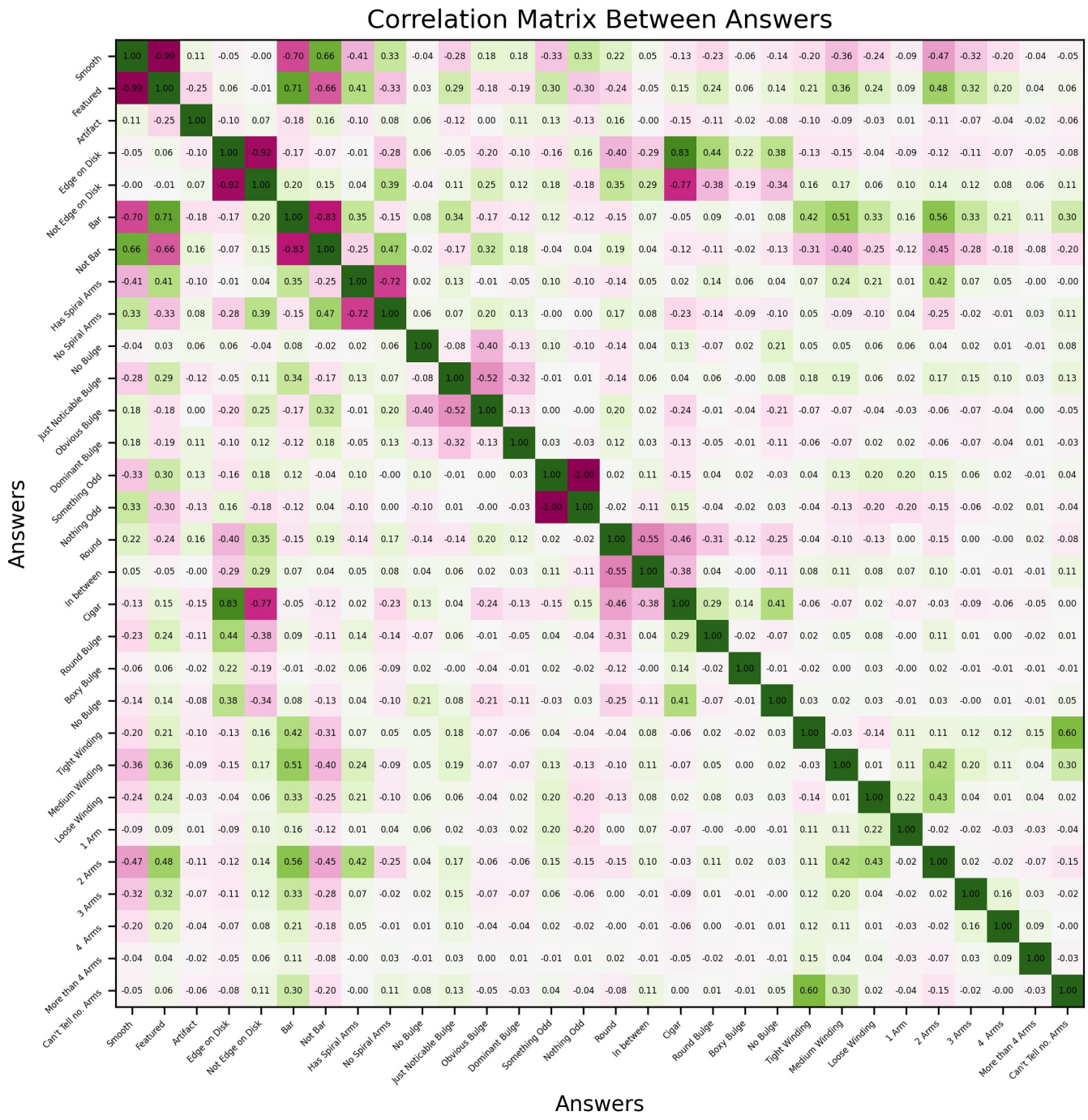
3/2/2024 Malachy - dataset analysis, analsysis behind the confidence of human feature identification

In [ ]:

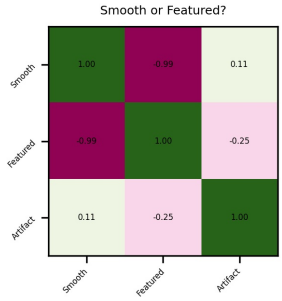
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

# Read the Parquet file
df = pd.read_parquet(parquet_file_path)

# plot a correlation matrix of the dataset

smooth_frac = df['smooth-or-featured-gz2_smooth_fraction']
disk_frac = df['smooth-or-featured-gz2_featured-or-disk_fraction']
artifact_frac = df['smooth-or-featured-gz2_artifact_fraction']
disk_edge_on_yes_frac = df['disk-edge-on-gz2_yes_fraction']
disk_edge_on_no_frac = df['disk-edge-on-gz2_no_fraction']
has_spiral_arms_yes_frac = df['has-spiral-arms-gz2_yes_fraction']
has_spiral_arms_no_frac = df['has-spiral-arms-gz2_no_fraction']
bar_yes_frac = df['bar-gz2_yes_fraction']
bar_no_frac = df['bar-gz2_no_fraction']
bulge_size_no_frac = df['bulge-size-gz2_no_fraction']
bulge_size_just_noticable_frac = df['bulge-size-gz2_just-noticeable_fraction']
bulge_size_obvious_frac = df['bulge-size-gz2_obvious_fraction']
bulge_size_dominant_frac = df['bulge-size-gz2_dominant_fraction']
something_odd_yes_frac = df['something-odd-gz2_yes_fraction']
something_odd_no_frac = df['something-odd-gz2_no_fraction']
how_rounded_round_frac = df['how-rounded-gz2_round_fraction']
how_rounded_in_between_frac = df['how-rounded-gz2_in-between_fraction']
how_rounded_cigar_frac = df['how-rounded-gz2_cigar_fraction']
bulge_shape_round_frac = df['bulge-shape-gz2_round_fraction']
bulge_shape_boxy_frac = df['bulge-shape-gz2_boxy_fraction']
bulge_shape_no_bulge_frac = df['bulge-shape-gz2_no-bulge_fraction']
spiral_winding_tight_frac = df['spiral-winding-gz2_tight_fraction']
spiral_winding_medium_frac = df['spiral-winding-gz2_medium_fraction']
spiral_winding_loose_frac = df['spiral-winding-gz2_loose_fraction']
spiral_arm_count_1_frac = df['spiral-arm-count-gz2_1_fraction']
spiral_arm_count_2_frac = df['spiral-arm-count-gz2_2_fraction']
spiral_arm_count_3_frac = df['spiral-arm-count-gz2_3_fraction']
spiral_arm_count_4_frac = df['spiral-arm-count-gz2_4_fraction']
spiral_arm_count_more_than_4_frac = df['spiral-arm-count-gz2_more-than-4_fraction']
spiral_arm_count_cant_tell_frac = df['spiral-arm-count-gz2_cant-tell_fraction']
```

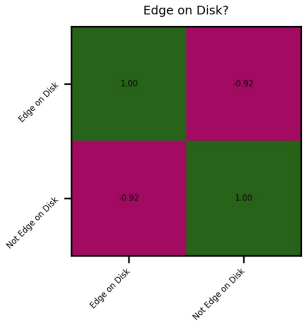
```
data = [smooth_frac, disk_frac, artifact_frac, disk_edge_on_yes_frac, disk_edge_on_no_frac, has_spiral_arms_yes
    bulge_size_no_frac, bulge_size_just_noticable_frac, bulge_size_obvious_frac, bulge_size_dominant_frac, some
    how_rounded_in_between_frac, how_rounded_cigar_frac, bulge_shape_round_frac, bulge_shape_boxy_frac, bulge_s
    spiral_winding_loose_frac, spiral_arm_count_1_frac, spiral_arm_count_2_frac, spiral_arm_count_3_frac, spira

# Remove any NaN values from the data
data = [np.nan_to_num(d) for d in data]

class_names = ['Smooth', 'Featured', 'Artifact', 'Edge on Disk', 'Not Edge on Disk', 'Bar', 'Not Bar', 'Has Spi
        'Obvious Bulge', 'Dominant Bulge', 'Something Odd', 'Nothing Odd', 'Round', 'In between', 'Cigar', 'Rou
        'Medium Winding', 'Loose Winding', '1 Arm', '2 Arms', '3 Arms', '4  Arms', 'More than 4 Arms', "Can't T

questions = ['Smooth?', 'Featured?', 'Artifact?', 'Edge on Disk?', 'Not Edge on Disk?', 'Bar?', 'Not Bar?', 'Ha
            'No Bulge?', 'Just Noticable Bulge?', 'Obvious Bulge?', 'Dominant Bulge?', 'Something Odd?', 'Nothi
            'Round Bulge?', 'Boxy Bulge?', 'No Bulge?', 'Tight Winding?', 'Medium Winding?', 'Loose Winding?',
            'More than 4 Arms?', "Can't Tell no. Arms?"]

fig, ax = plt.subplots(1,1, figsize=(10,10), dpi = 300)

correlation_matrix = np.corrcoef(data)

for i in range(len(class_names)):
    for j in range(len(class_names)):
        ax.text(j, i, f"{correlation_matrix[i, j]:.2f}", ha="center", va="center", color="black", fontsize=4)

cax = ax.matshow(correlation_matrix, cmap='PiYG')

ax.set_xticks(np.arange(len(class_names)))
ax.set_yticks(np.arange(len(class_names)))

ax.set_xticklabels(class_names, rotation = 45, fontsize=4, ha = 'right', va = 'center', rotation_mode = 'anchor
ax.xaxis.set_ticks_position('bottom')
ax.set_xticklabels(class_names, rotation = 45, fontsize=4, ha = 'right', va = 'center', rotation_mode = 'anchor
ax.xaxis.set_ticks_position('bottom')
ax.set_yticklabels(class_names, rotation = 45, fontsize=4, ha = 'right', va = 'center', rotation_mode = 'anchor

ax.set_xlabel('Answers')
ax.set_ylabel('Answers')
ax.set_title('Correlation Matrix Between Answers')

fig.tight_layout(pad=10.0)
plt.show()

groups = [
    [smooth_frac, disk_frac, artifact_frac],
    [disk_edge_on_yes_frac, disk_edge_on_no_frac],
    [bar_yes_frac, bar_no_frac],
    [has_spiral_arms_yes_frac, has_spiral_arms_no_frac],
    [bulge_size_no_frac, bulge_size_just_noticable_frac, bulge_size_obvious_frac, bulge_size_dominant_frac],
    [something_odd_yes_frac, something_odd_no_frac],
    [how_rounded_round_frac, how_rounded_in_between_frac, how_rounded_cigar_frac],
    [bulge_shape_round_frac, bulge_shape_boxy_frac, bulge_shape_no_bulge_frac],
    [spiral_winding_tight_frac, spiral_winding_medium_frac, spiral_winding_loose_frac],
    [spiral_arm_count_1_frac, spiral_arm_count_2_frac, spiral_arm_count_3_frac, spiral_arm_count_4_frac, spiral
]

names = [
    ['Smooth', 'Featured', 'Artifact'],
    ['Edge on Disk', 'Not Edge on Disk'],
    ['Bar', 'Not Bar'],
    ['Has Spiral Arms', 'No Spiral Arms'],
    ['No Bulge', 'Just Noticable Bulge', 'Obvious Bulge', 'Dominant Bulge'],
    ['Something Odd', 'Nothing Odd'],
    ['Round', 'In between', 'Cigar'],
    ['Round Bulge', 'Boxy Bulge', 'No Bulge'],
    ['Tight Winding', 'Medium Winding', 'Loose Winding'],
    ['1 Arm', '2 Arms', '3 Arms', '4  Arms', 'More than 4 Arms', "Can't Tell no. Arms"]

]

titles = [
    'Smooth or Featured?',
    'Edge on Disk?',
    'Bar?',
    'Has Spiral Arms?',
    'Bulge Size?',
    'Something Odd?',
    'How Rounded?',
    'Bulge Shape?',
    'Spiral Winding?',
    'Spiral Arm Count?'
]
```

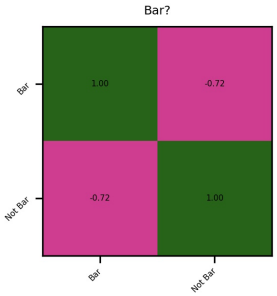
```
fig, ax = plt.subplots(4, 3, figsize=(10,10), dpi=300)

for idx, group in enumerate(groups):
    row = idx // 3
    col = idx % 3

    plt.subplots_adjust(hspace=0.5, wspace=0.5)
    group = [np.nan_to_num(g) for g in group]
    ax[row, col].set_title(titles[idx], fontsize=6)

    corr_matrix = np.corrcoef(group)
    cax = ax[row, col].matshow(corr_matrix, cmap='PiYG', vmin=-1.0, vmax=1.0)

    for i in range(len(group)):
        for j in range(len(group)):
            ax[row, col].text(j, i, f"{corr_matrix[i, j]:.2f}", ha="center", va="center", color="black", fontsi

    ax[row, col].set_xticks(np.arange(len(group)))
    ax[row, col].set_yticks(np.arange(len(group)))

    ax[row, col].set_xticklabels(names[idx], rotation=45, fontsize=4, ha='right', va='center', rotation_mode='a
    ax[row, col].xaxis.set_ticks_position('bottom')

    ax[row, col].set_yticklabels(names[idx], rotation=45, fontsize=4, ha='right', va='center', rotation_mode='a

for idx in range(10, 12):
    ax.flatten()[idx].axis('off')


fig.suptitle('Correlation Matrices of Answers to Each Question', fontsize=12, ha = 'center', va = 'bottom')
fig.tight_layout()
plt.show()
```

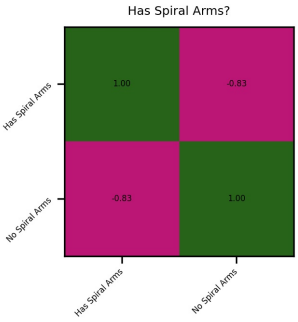

Correlation Matrix Between Answers
Answers
Answers
Smooth
Featured
Artifact
Edge on Disk
Not Edge on Disk
Bar
Not Bar
Has Spiral Arms
No Spiral Arms
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge
Something Odd
Nothing Odd
Round
In between
Cigar
Round Bulge
Boxy Bulge
No Bulge
Tight Winding
Medium Winding
Loose Winding
1 Arm
2 Arms
3 Arms
4 Arms
More than 4 Arms
Can't Tell no. Arms

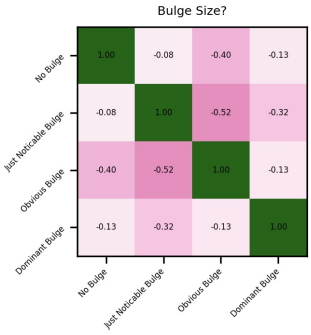
Correlation Matrices of Answers to Each Question

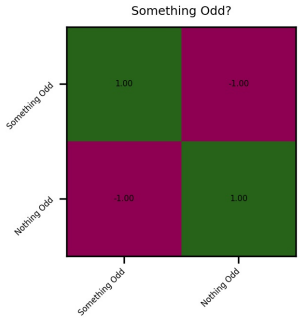


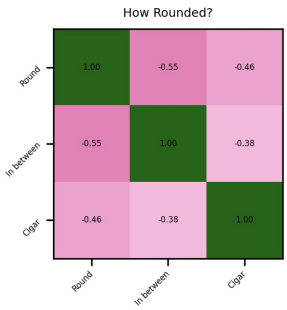


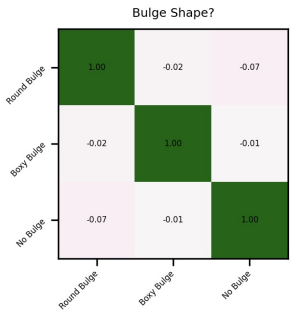


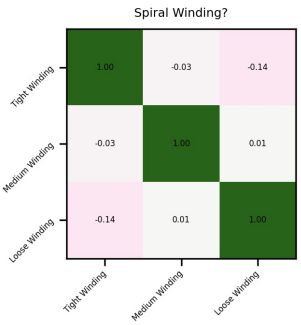


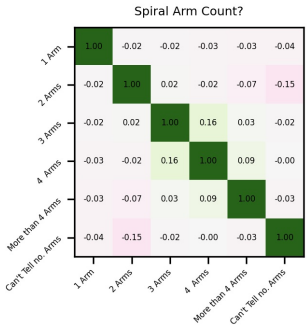


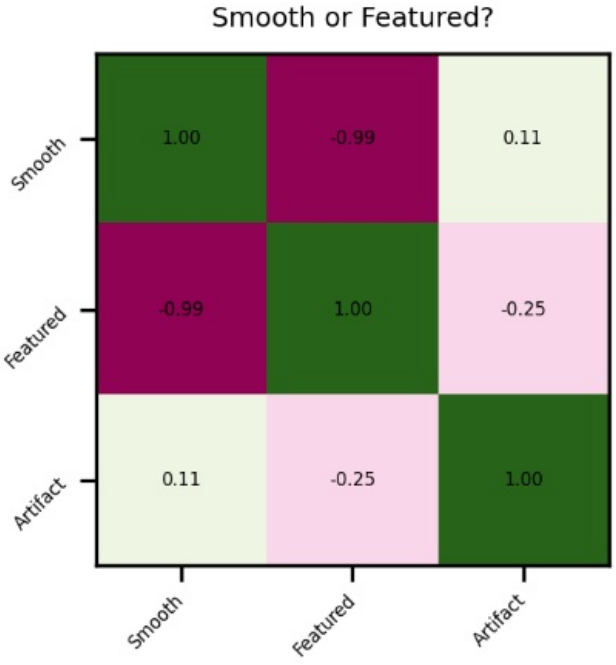




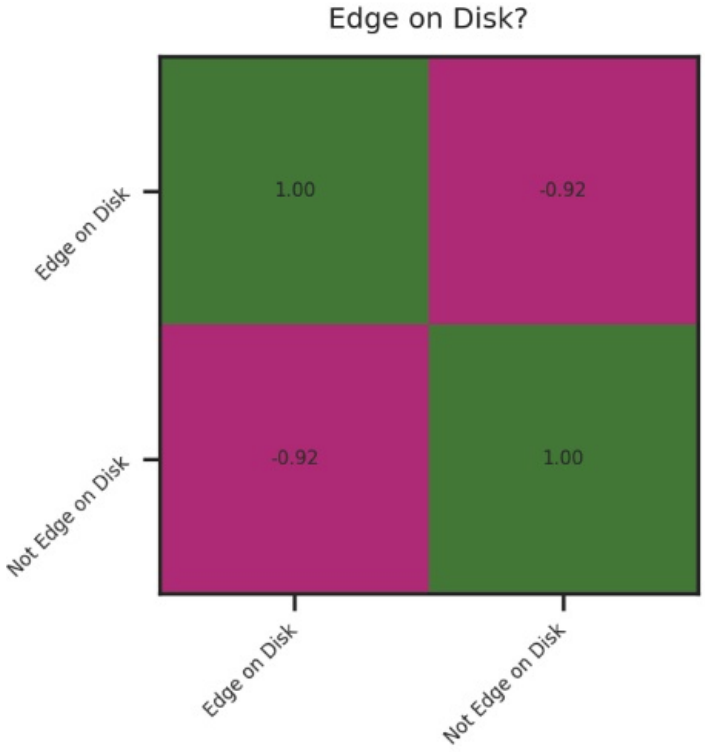




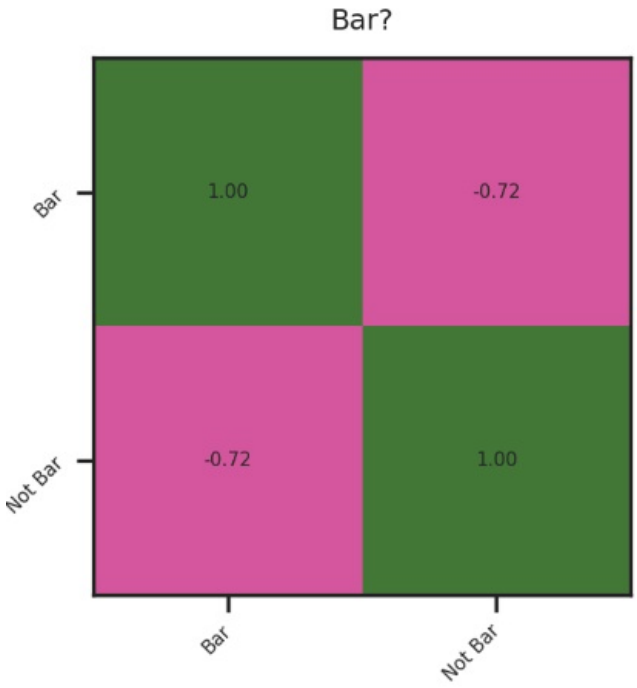


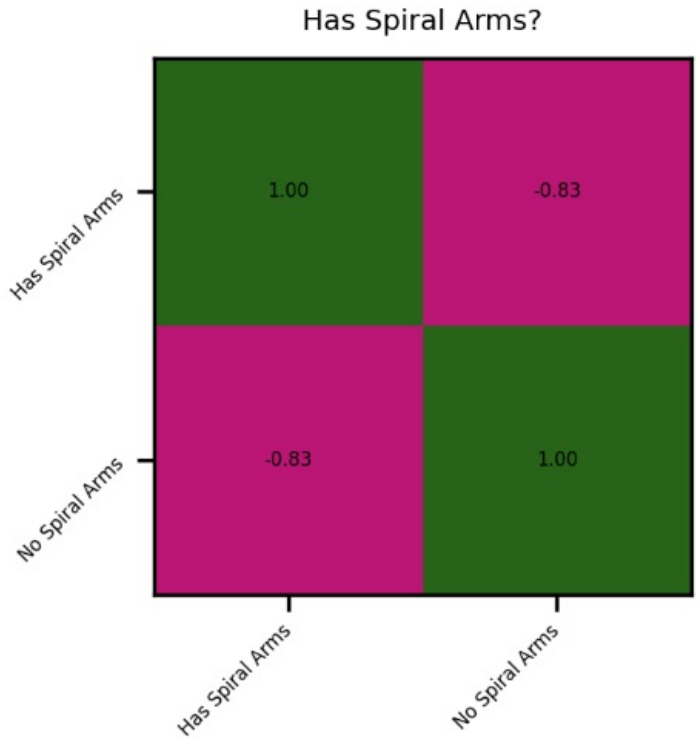


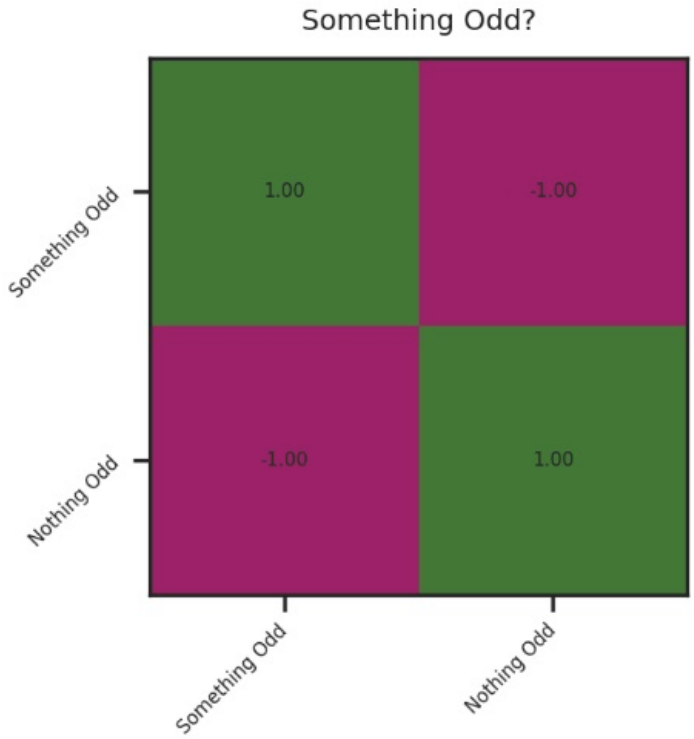
this correlation matrix is what should be expected for smooth and features as they are mutually exclusive and should have negative correlations. however artifact appears to have a 0 correlation between the other features despite it being mutually exclusive in the decison tree, this could mean the human labellers struggle to identify artifacts in an image or instead a large proportion identify a larger star in the image over the smaller central galaxy in the image. (they think the star is the only thing in the image). any model might be bad at classifying artifacts if human labellers struggle with it



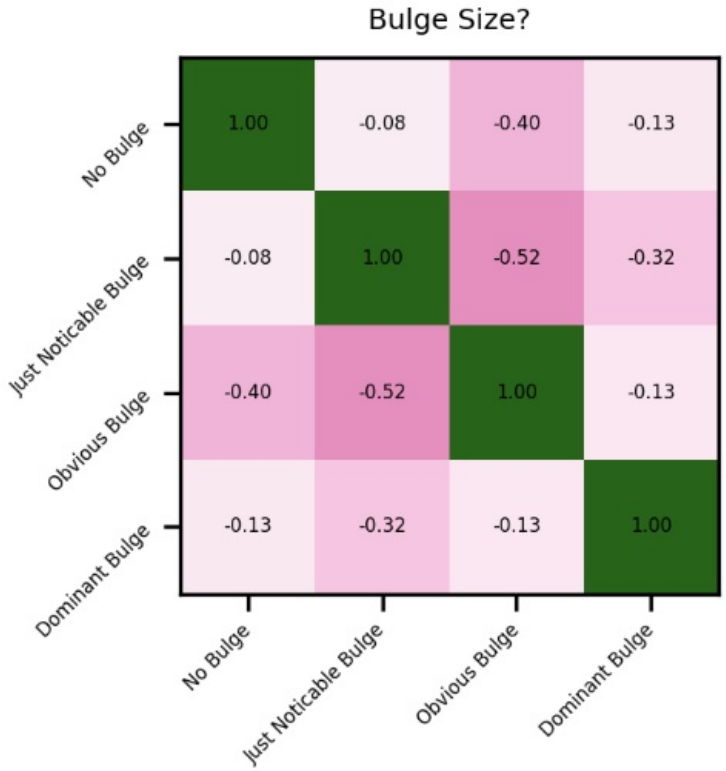
disk edge on and disk face on should be perfectly negatively correlated as they are mutually exclusive in the galaxy zoo decision tree. they human voters correlation matrix also matches the expected results for the correlation matrix meaning the human labellers should be a good approximation to the ground truth for this feature.

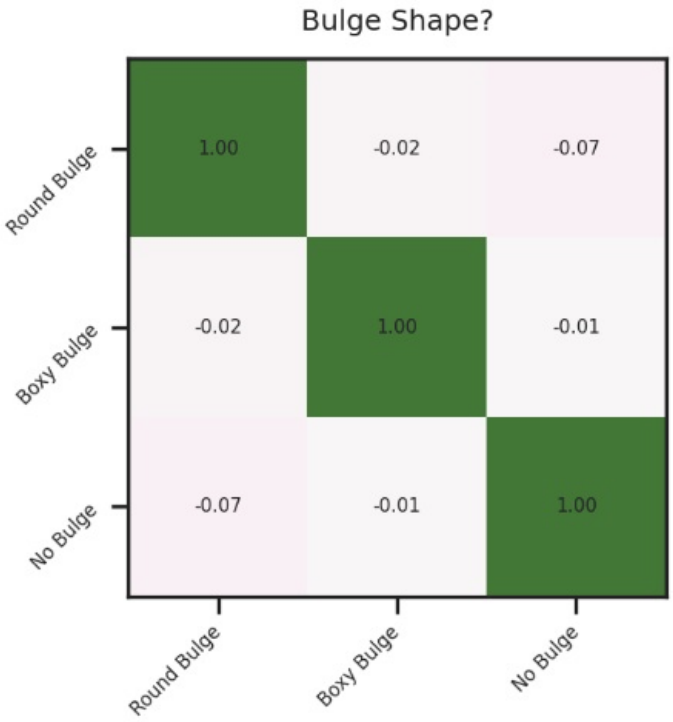


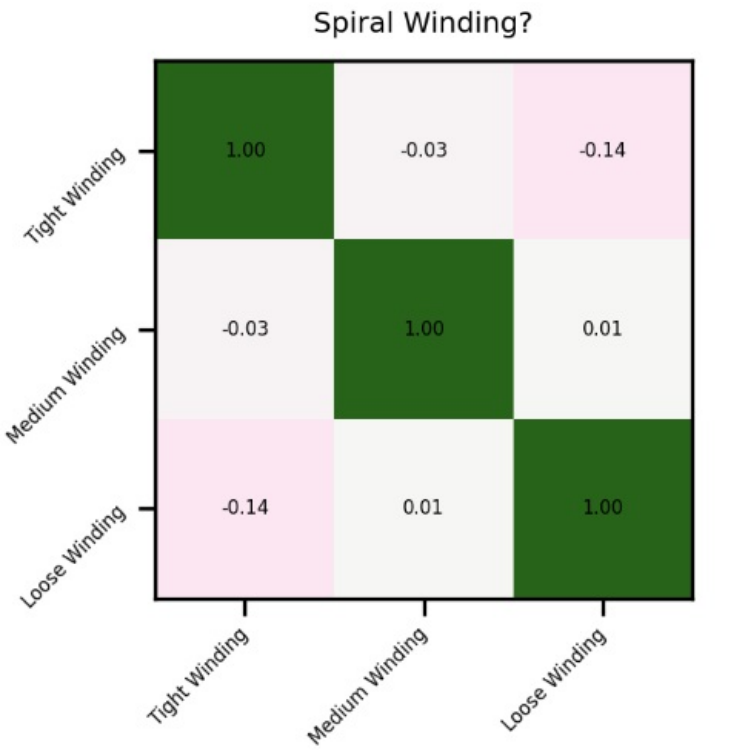


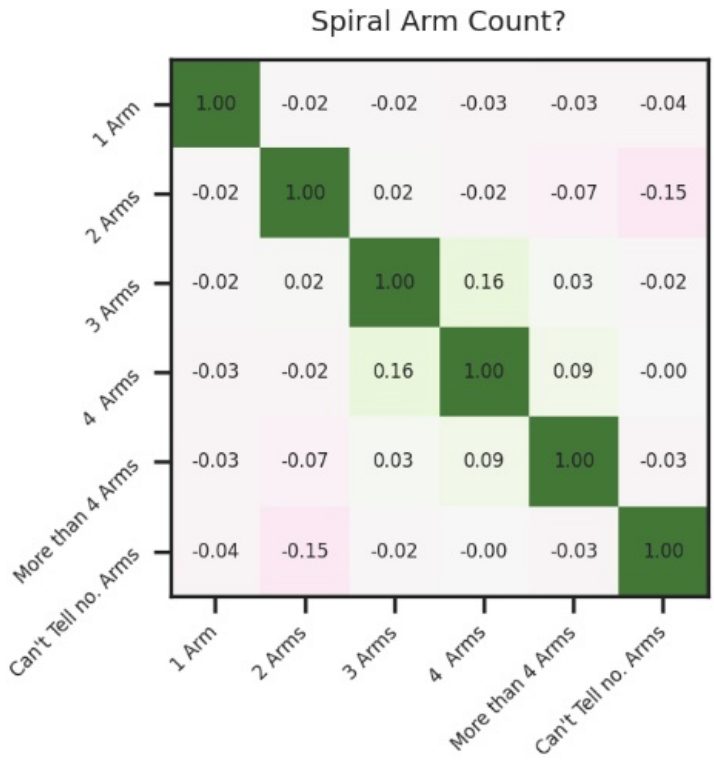


same with identifying if there is a central bar, spiral arms, and if there's something odd (be that dust lane, galaxy merger, overlapping galaxies ect) these could be identified fairly confidently


Bulge Size?
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge
1.00
-0.08
-0.40
-0.13
-0.08
1.00
-0.52
-0.32
-0.40
-0.52
1.00
-0.13
-0.13
-0.32
-0.13
1.00
No Bulge
Just Noticable Bulge
Obvious Bulge
Dominant Bulge


Bulge Shape?
Round Bulge
Boxy Bulge
No Bulge
1.00
-0.02
-0.07
-0.02
1.00
-0.01
-0.07
-0.01
1.00
Round Bulge
Boxy Bulge
No Bulge





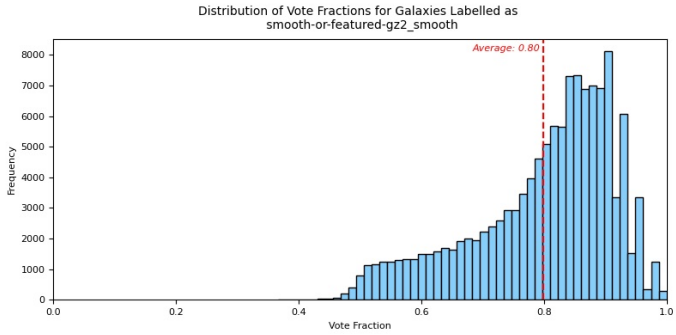
bulge size, bulge shape, spiral widing, and spiral arm amount: all have correlations between the other features with the human labels, this could mean that the human labellers struggle to reach a concensus on the presence of these features as ideally all the features should be perfectly negatively correlated with each other apart from with themselves. and model using the humans as an approximation to the ground truth could stuggle to identify these features if the humans also struggle to identify them.

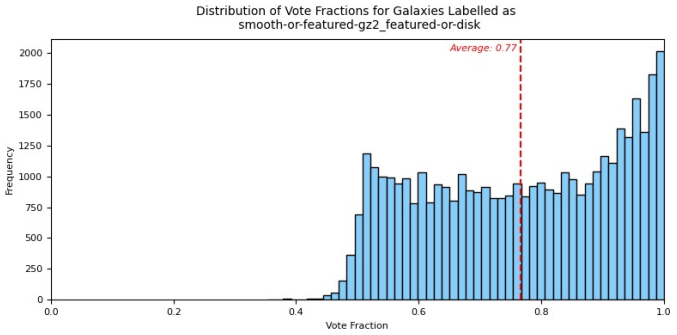
3/2/2024 Malachy - analysis of the distribution of vote fractions for each galaxy feature with the averge vote confidence for that feature

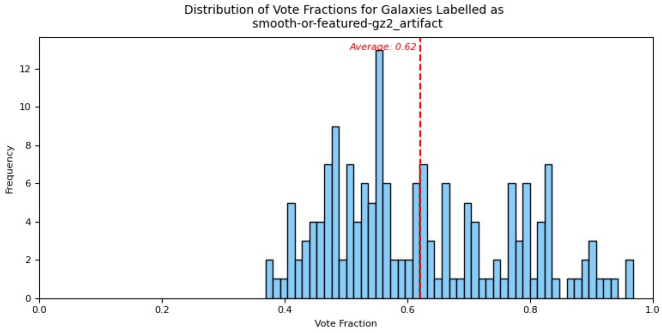
In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

df = pd.read_parquet(parquet_file_path)

#find all the galaxies labelled as having smooth_semantic and find the average smooth_fraction

def plot_label_dist(semantic, labels, fraction, df, colour='lightskyblue'):
```

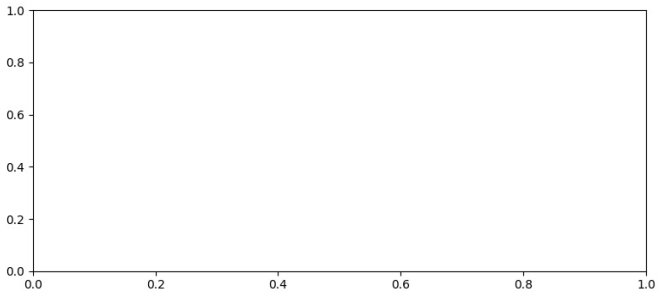
```
    num_labels = len(labels)
    num_rows = int(np.ceil(num_labels / 2))
    num_cols = min(2, num_labels)

    fig, ax = plt.subplots(num_rows, num_cols, figsize=(8 * num_cols, 4 * num_rows), dpi=100)

    for idx, label in enumerate(labels):
        data = [df[fraction[idx]][i] for i in range(df['id_str'].size) if df[semantic][i] == label]

        row = idx // num_cols
        col = idx % num_cols

        if num_rows == 1:
            current_ax = ax[col]
        else:
            current_ax = ax[row, col]

        current_ax.hist(data, bins=50, color=colour, edgecolor='black')

        average_fraction = np.mean(data)

        current_ax.axvline(average_fraction, color='red', linestyle='--')
        current_ax.text(average_fraction, current_ax.get_ylim()[1], f'\n\nAverage: {average_fraction:.2f} ', co

        current_ax.set_xlim(0, 1)
        current_ax.set_ylabel('Frequency', fontsize=8)
        current_ax.set_xlabel('Vote Fraction', fontsize=8)

        # Set the x and y tick label sizes
        current_ax.tick_params(axis='x', labelsize=8)
        current_ax.tick_params(axis='y', labelsize=8)

        current_ax.set_title(f'Distribution of Vote Fractions for Galaxies Labelled as \n {label}', fontsize=10

    plt.subplots_adjust(hspace=0.4, wspace=0.2)
    plt.tight_layout()
    plt.show()

semantic = 'smooth-or-featured-gz2_semantic'
labels = ['smooth-or-featured-gz2_smooth', 'smooth-or-featured-gz2_featured-or-disk', 'smooth-or-featured-gz2_a
fraction = ['smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction', 'smoo
plot_label_dist(semantic, labels, fraction, df)

semantic = 'disk-edge-on-gz2_semantic'
labels = ['disk-edge-on-gz2_yes', 'disk-edge-on-gz2_no']
fraction = ['disk-edge-on-gz2_yes_fraction', 'disk-edge-on-gz2_no_fraction']
plot_label_dist(semantic, labels, fraction, df)

semantic = 'bar-gz2_semantic'
labels = ['bar-gz2_yes', 'bar-gz2_no']
fraction = ['bar-gz2_yes_fraction', 'bar-gz2_no_fraction']
plot_label_dist(semantic, labels, fraction, df)

semantic = 'has-spiral-arms-gz2_semantic'
labels = ['has-spiral-arms-gz2_yes', 'has-spiral-arms-gz2_no']
fraction = ['has-spiral-arms-gz2_yes_fraction', 'has-spiral-arms-gz2_no_fraction']
plot_label_dist(semantic, labels, fraction, df)

semantic = 'bulge-size-gz2_semantic'
labels = ['bulge-size-gz2_no', 'bulge-size-gz2_just-noticeable', 'bulge-size-gz2_obvious', 'bulge-size-gz2_domi
fraction = ['bulge-size-gz2_no_fraction', 'bulge-size-gz2_just-noticeable_fraction', 'bulge-size-gz2_obvious_fra
plot_label_dist(semantic, labels, fraction, df)

semantic = 'something-odd-gz2_semantic'
labels = ['something-odd-gz2_yes', 'something-odd-gz2_no']
fraction = ['something-odd-gz2_yes_fraction', 'something-odd-gz2_no_fraction']
plot_label_dist(semantic, labels, fraction, df)

semantic = 'how-rounded-gz2_semantic'
labels = ['how-rounded-gz2_round', 'how-rounded-gz2_in-between', 'how-rounded-gz2_cigar']
fraction = ['how-rounded-gz2_round_fraction', 'how-rounded-gz2_in-between_fraction', 'how-rounded-gz2_cigar_fra
plot_label_dist(semantic, labels, fraction, df)

semantic = 'bulge-shape-gz2_semantic'
labels = ['bulge-shape-gz2_round', 'bulge-shape-gz2_boxy', 'bulge-shape-gz2_no-bulge']
fraction = ['bulge-shape-gz2_round_fraction', 'bulge-shape-gz2_boxy_fraction', 'bulge-shape-gz2_no-bulge_fracti
plot_label_dist(semantic, labels, fraction, df)

semantic = 'spiral-winding-gz2_semantic'
labels = ['spiral-winding-gz2_tight', 'spiral-winding-gz2_medium', 'spiral-winding-gz2_loose']
fraction = ['spiral-winding-gz2_tight_fraction', 'spiral-winding-gz2_medium_fraction', 'spiral-winding-gz2_loos
plot_label_dist(semantic, labels, fraction, df)
```

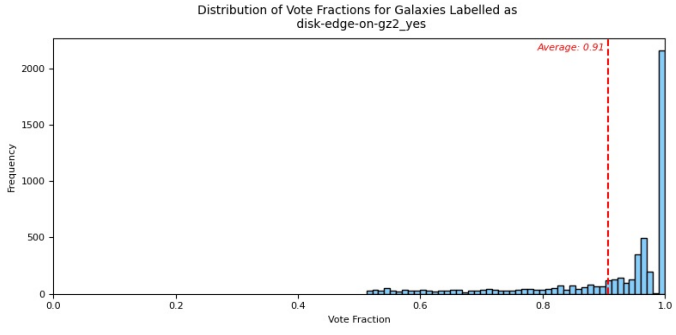
```
semantic = 'spiral-arm-count-gz2_semantic'
labels = ['spiral-arm-count-gz2_1', 'spiral-arm-count-gz2_2', 'spiral-arm-count-gz2_3', 'spiral-arm-count-gz2_4
fraction = ['spiral-arm-count-gz2_1_fraction', 'spiral-arm-count-gz2_2_fraction', 'spiral-arm-count-gz2_3_fracti
plot_label_dist(semantic, labels, fraction, df)
```

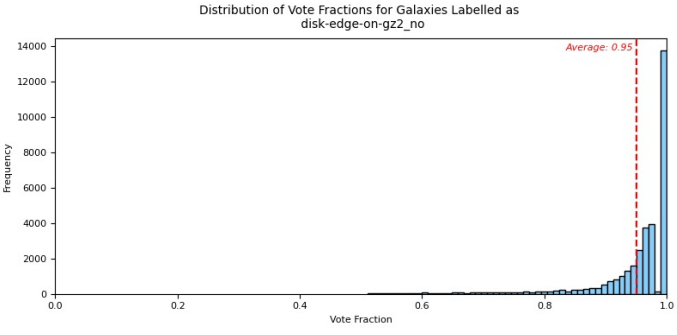


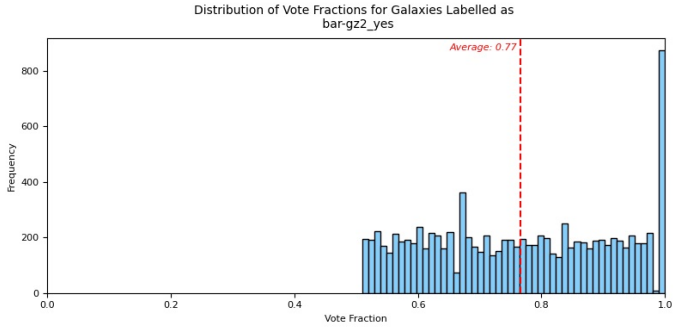


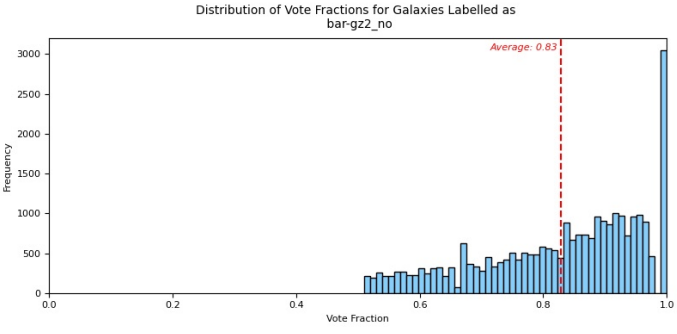


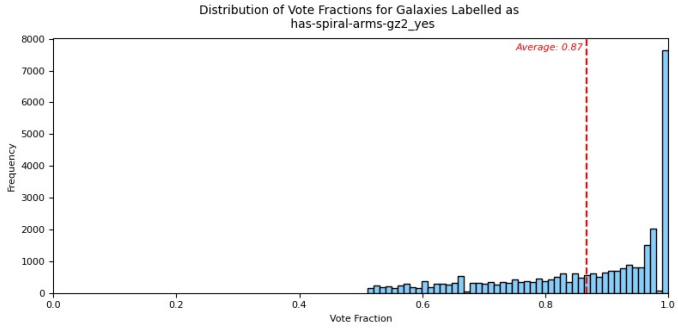


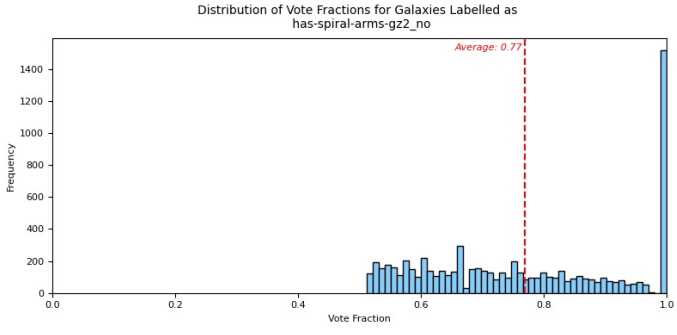


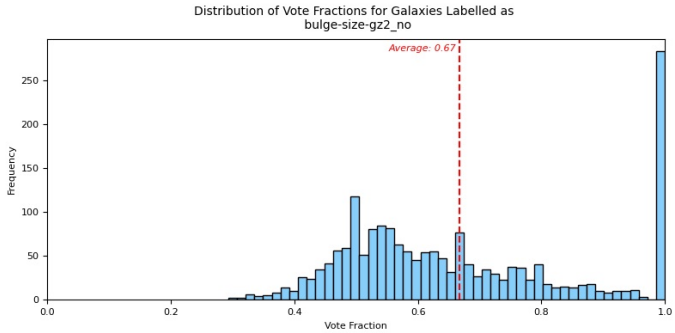


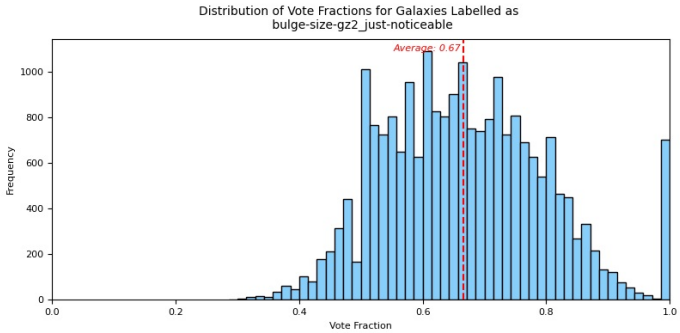


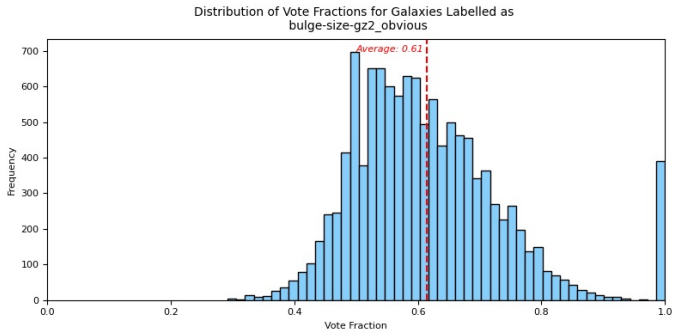


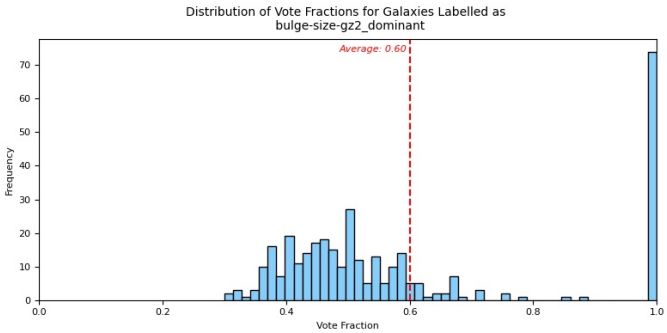


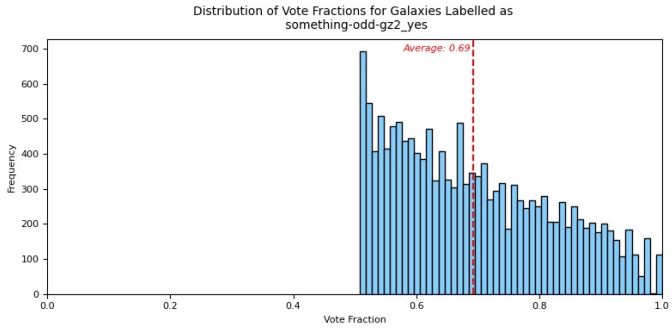


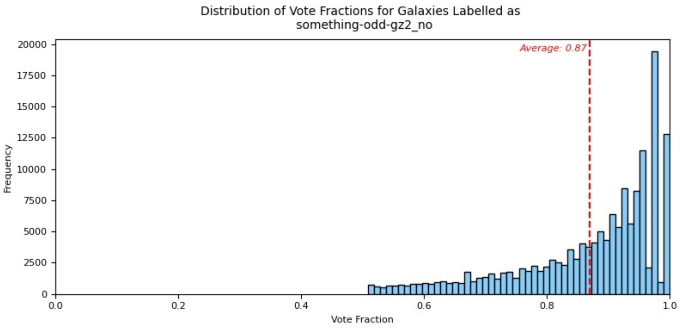

Distribution of Vote Fractions for Galaxies Labelled as
bulge-size-gz2_no
Average: 0.67
Frequency
Vote Fraction

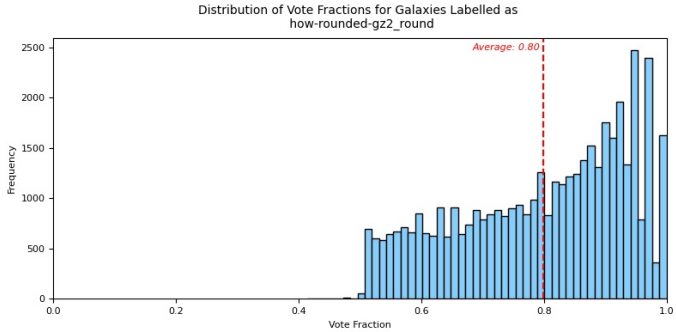

Distribution of Vote Fractions for Galaxies Labelled as
bulge-size-gz2_just-noticeable
Average: 0.67
Frequency
Vote Fraction

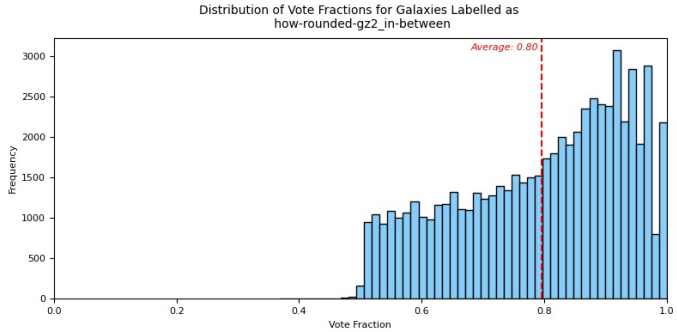

Distribution of Vote Fractions for Galaxies Labelled as
bulge-size-gz2_obvious
Average: 0.61
Frequency
Vote Fraction

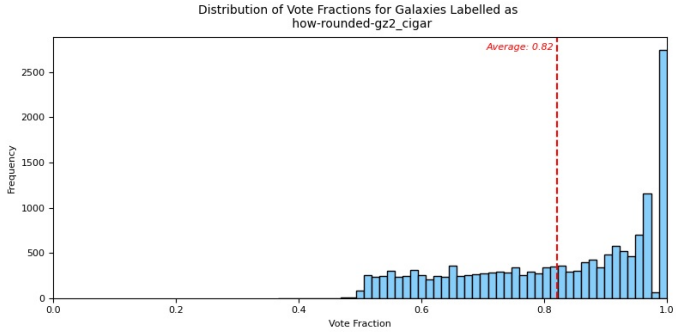

Distribution of Vote Fractions for Galaxies Labelled as
bulge-size-gz2_dominant
Average: 0.60
Frequency
Vote Fraction

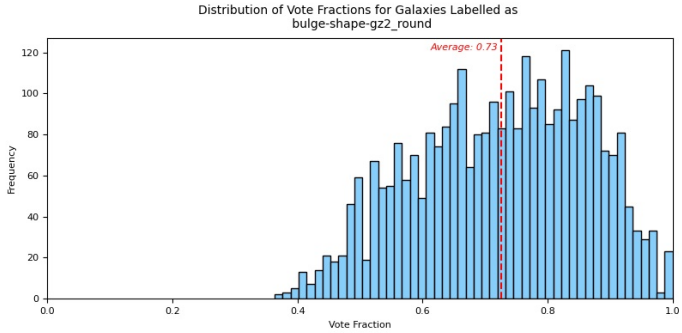

Distribution of Vote Fractions for Galaxies Labelled as
something-odd-gz2_yes
Average: 0.69
Frequency
Vote Fraction

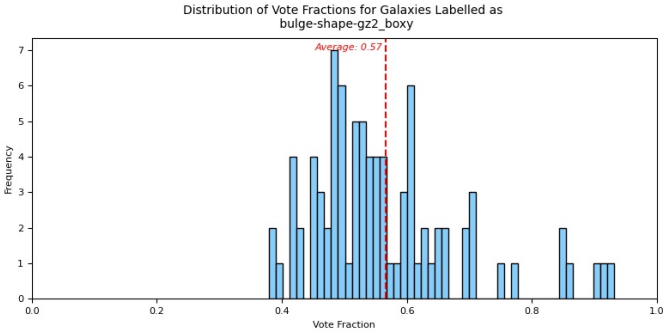

Distribution of Vote Fractions for Galaxies Labelled as
something-odd-gz2_no
Average: 0.87
Frequency
Vote Fraction

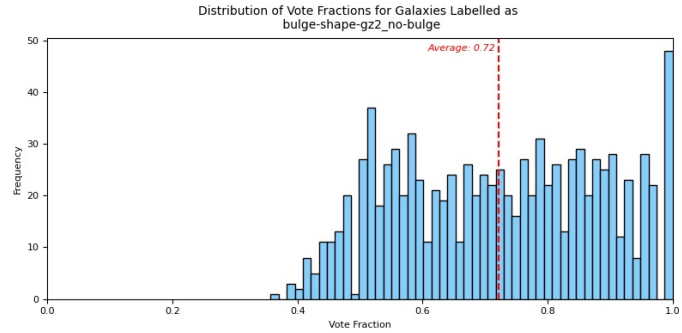

Distribution of Vote Fractions for Galaxies Labelled as
how-rounded-gz2_round
Average: 0.80
Frequency
Vote Fraction

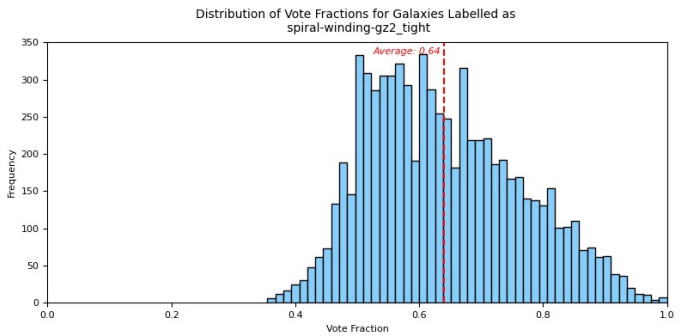

Distribution of Vote Fractions for Galaxies Labelled as
how-rounded-gz2_in-between
Average: 0.80
Frequency
Vote Fraction

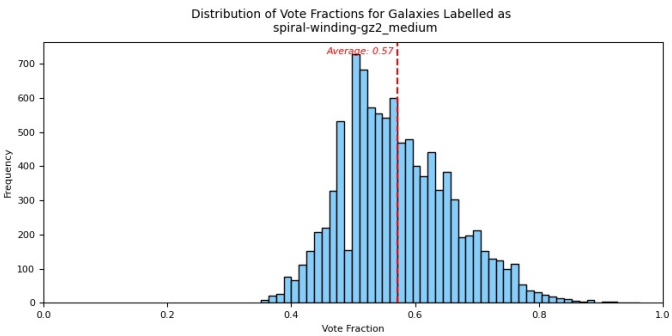

Distribution of Vote Fractions for Galaxies Labelled as
how-rounded-gz2_cigar
Average: 0.82
Frequency
Vote Fraction

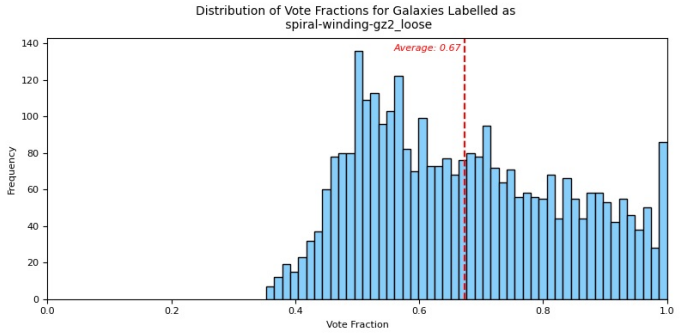

Distribution of Vote Fractions for Galaxies Labelled as
bulge-shape-gz2_round
Average: 0.73
Frequency
Vote Fraction

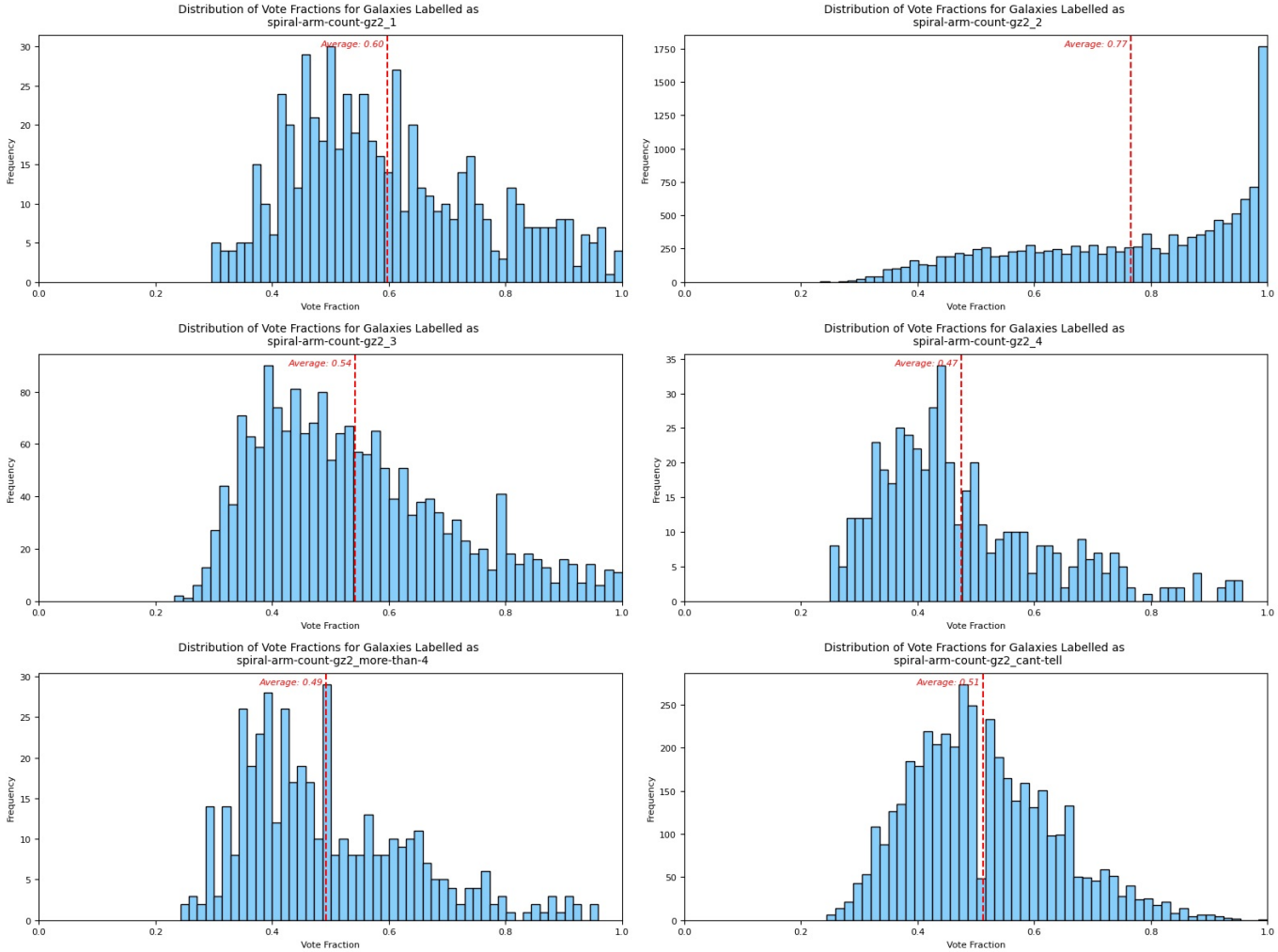

Distribution of Vote Fractions for Galaxies Labelled as
bulge-shape-gz2_boxy
Average: 0.57
Frequency
Vote Fraction


Distribution of Vote Fractions for Galaxies Labelled as
bulge-shape-gz2_no-bulge
Average: 0.72
Frequency
Vote Fraction


Distribution of Vote Fractions for Galaxies Labelled as
spiral-winding-gz2_tight
Average: 0.64
Frequency
Vote Fraction


Distribution of Vote Fractions for Galaxies Labelled as
spiral-winding-gz2_medium
Average: 0.57
Frequency
Vote Fraction


Distribution of Vote Fractions for Galaxies Labelled as
spiral-winding-gz2_loose
Average: 0.67
Frequency
Vote Fraction



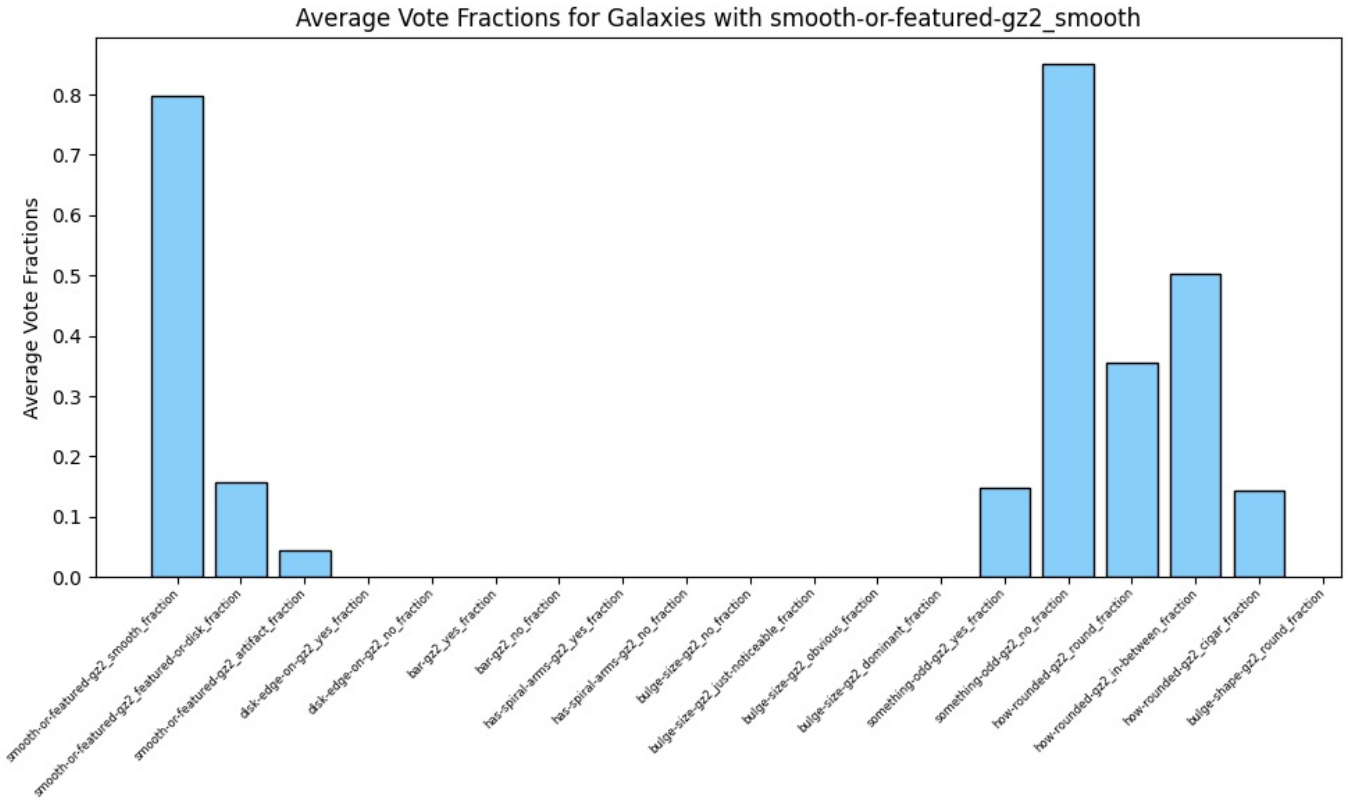
most confident feature is disk edge on no with an average vote fraction of 0.95 least confident feature is spiral arm count 4 with an average vote fraction of 0.47

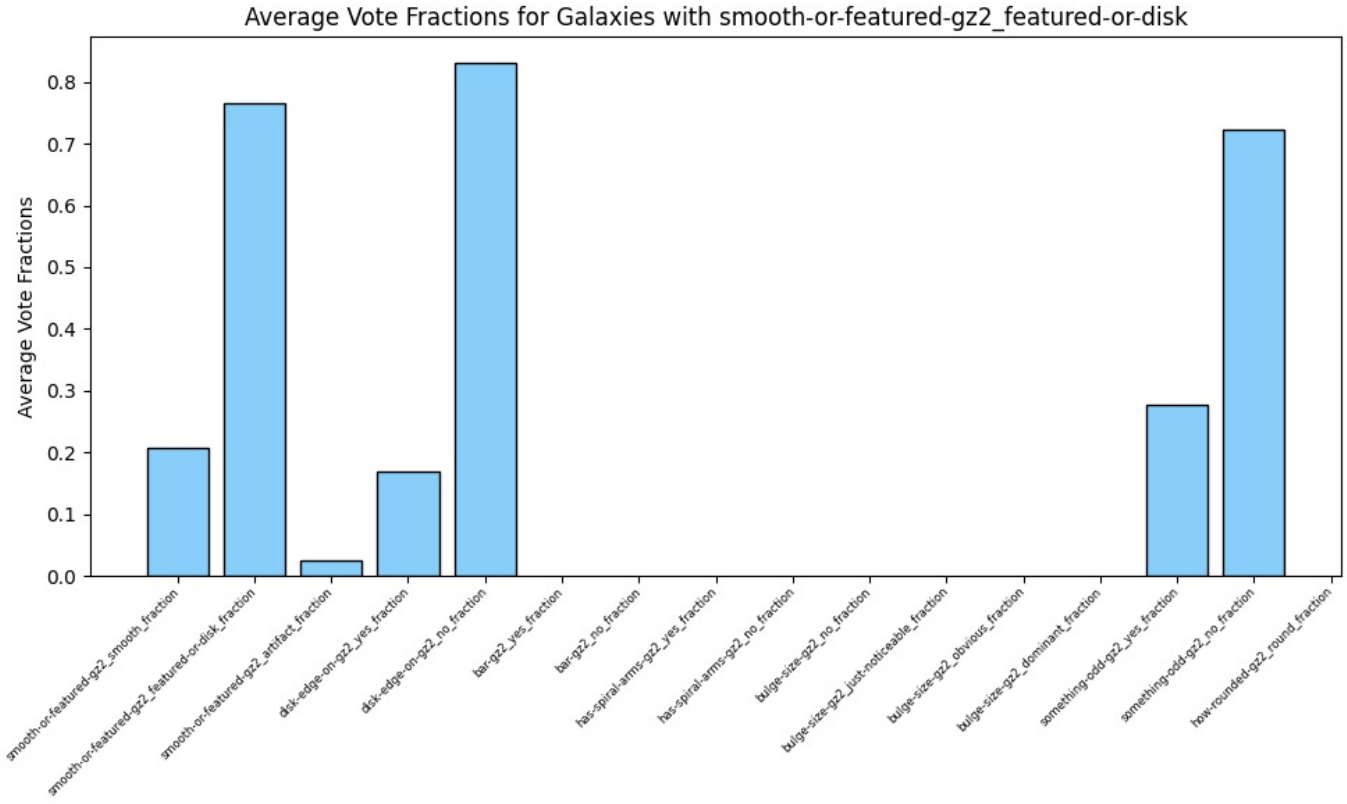
Malachy - average vote fraction for every other feature of galaxies labelled as having another feature.

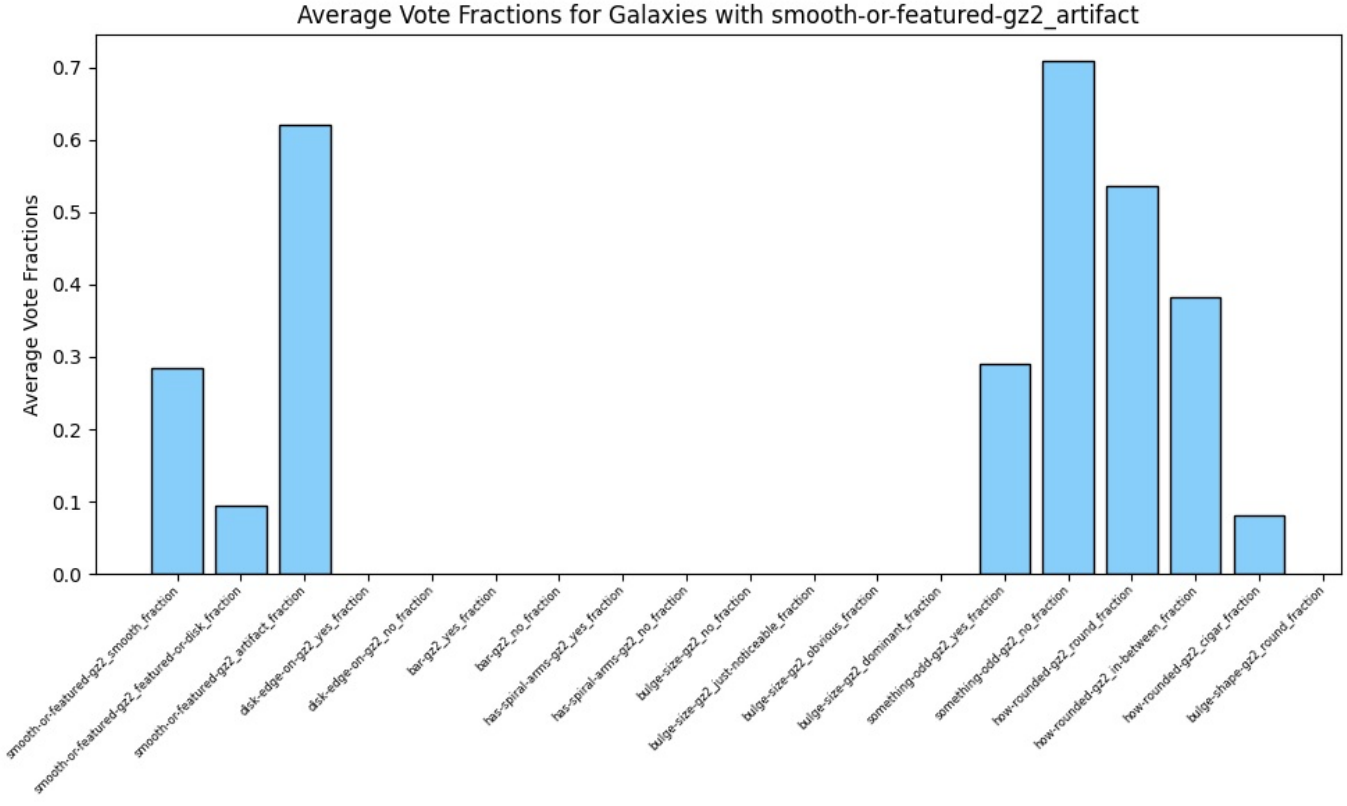
In [ ]:
```
# TODO: Go through every galaxy image, find its corresponding semantic labels, then find the vote fractions for
# i.e smooth-or-featured-gz2_semantic : smooth-or-featured-gz2-smooth - label => smooth-or-featured-smooth_frac
# Then plot the average the of vote fractions for each semantic label

# do the same again but for the 2nd and 3rd most voted label for each galaxy image (or for labels above a certa
# this will show the most commonly mixed up labels?

#IDEA: correlation matrix between semantic labels and vote fractions for each label????

#plot the highest vote fraction for each semantic label and the corresponding image
```

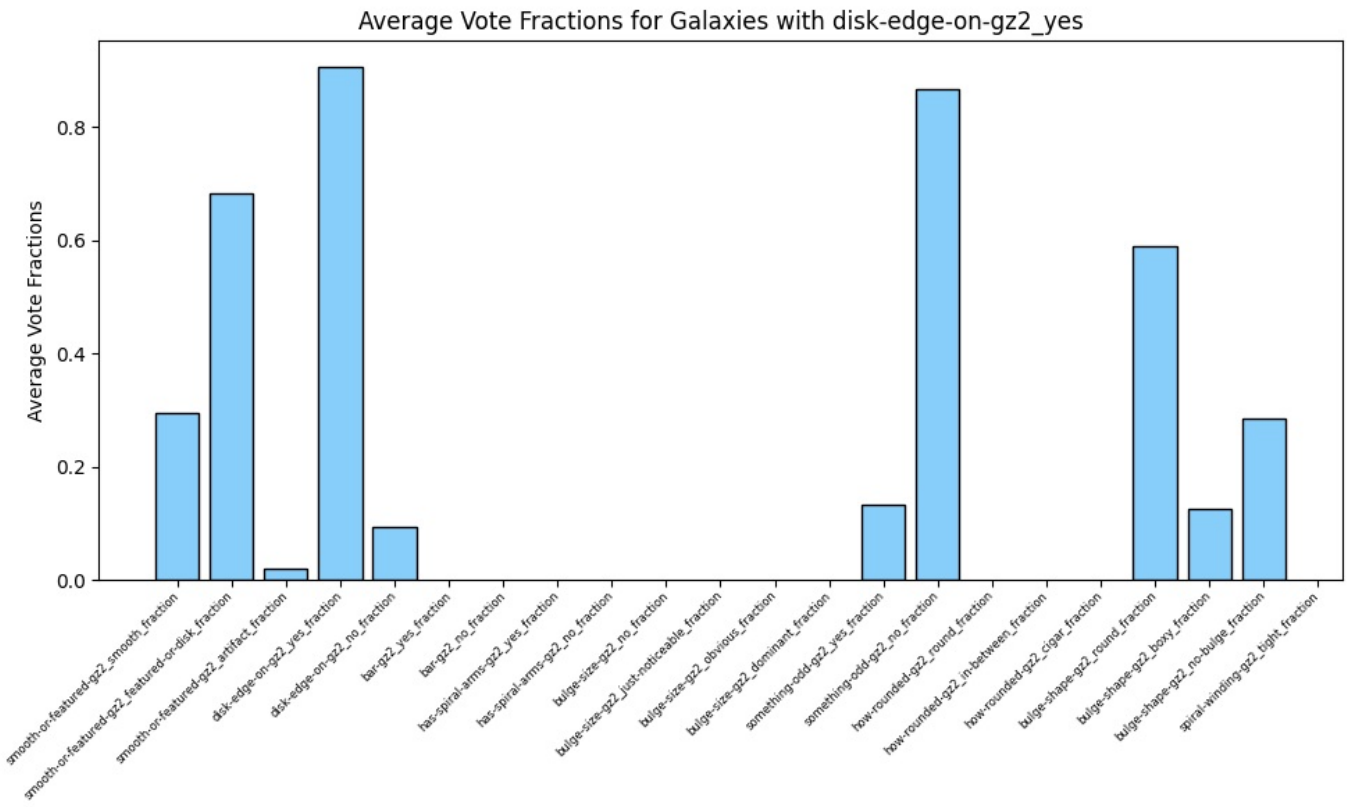
```python
#find the average image of the dataset to help determine cropping bounding box

import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

df = pd.read_parquet(parquet_file_path)

semantic_labels = ['smooth-or-featured-gz2_semantic', 'disk-edge-on-gz2_semantic', 'bar-gz2_semantic', 'has-spi
                   'something-odd-gz2_semantic', 'how-rounded-gz2_semantic', 'bulge-shape-gz2_semantic', 'spira

feature_to_id = {
    'smooth-or-featured-gz2_smooth': [],
    'smooth-or-featured-gz2_featured-or-disk': [],
    'smooth-or-featured-gz2_artifact': [],
    'disk-edge-on-gz2_yes': [],
    'disk-edge-on-gz2_no': [],
    'bar-gz2_yes': [],
    'bar-gz2_no': [],
    'has-spiral-arms-gz2_yes': [],
    'has-spiral-arms-gz2_no': [],
    'bulge-size-gz2_no': [],
    'bulge-size-gz2_just-noticeable': [],
    'bulge-size-gz2_obvious': [],
    'bulge-size-gz2_dominant': [],
    'something-odd-gz2_yes': [],
    'something-odd-gz2_no': [],
    'how-rounded-gz2_round': [],
    'how-rounded-gz2_in-between': [],
    'how-rounded-gz2_cigar': [],
    'bulge-shape-gz2_round': [],
    'bulge-shape-gz2_boxy': [],
    'bulge-shape-gz2_no-bulge': [],
    'spiral-winding-gz2_tight': [],
    'spiral-winding-gz2_medium': [],
    'spiral-winding-gz2_loose': [],
    'spiral-arm-count-gz2_1': [],
    'spiral-arm-count-gz2_2': [],
    'spiral-arm-count-gz2_3': [],
    'spiral-arm-count-gz2_4': [],
    'spiral-arm-count-gz2_more-than-4': [],
    'spiral-arm-count-gz2_cant-tell': [],
    'deadlock': [],
    '-': []
}

keys = [
    'smooth-or-featured-gz2_smooth_fraction',
    'smooth-or-featured-gz2_featured-or-disk_fraction',
    'smooth-or-featured-gz2_artifact_fraction',
    'disk-edge-on-gz2_yes_fraction',
    'disk-edge-on-gz2_no_fraction',
    'bar-gz2_yes_fraction',
    'bar-gz2_no_fraction',
    'has-spiral-arms-gz2_yes_fraction',
    'has-spiral-arms-gz2_no_fraction',
    'bulge-size-gz2_no_fraction',
    'bulge-size-gz2_just-noticeable_fraction',
    'bulge-size-gz2_obvious_fraction',
    'bulge-size-gz2_dominant_fraction',
    'something-odd-gz2_yes_fraction',
    'something-odd-gz2_no_fraction',
    'how-rounded-gz2_round_fraction',
    'how-rounded-gz2_in-between_fraction',
    'how-rounded-gz2_cigar_fraction',
    'bulge-shape-gz2_round_fraction',
    'bulge-shape-gz2_boxy_fraction',
    'bulge-shape-gz2_no-bulge_fraction',
    'spiral-winding-gz2_tight_fraction',
    'spiral-winding-gz2_medium_fraction',
    'spiral-winding-gz2_loose_fraction',
    'spiral-arm-count-gz2_1_fraction',
    'spiral-arm-count-gz2_2_fraction',
    'spiral-arm-count-gz2_3_fraction',
    'spiral-arm-count-gz2_4_fraction',
    'spiral-arm-count-gz2_more-than-4_fraction',
    'spiral-arm-count-gz2_cant-tell_fraction',
```

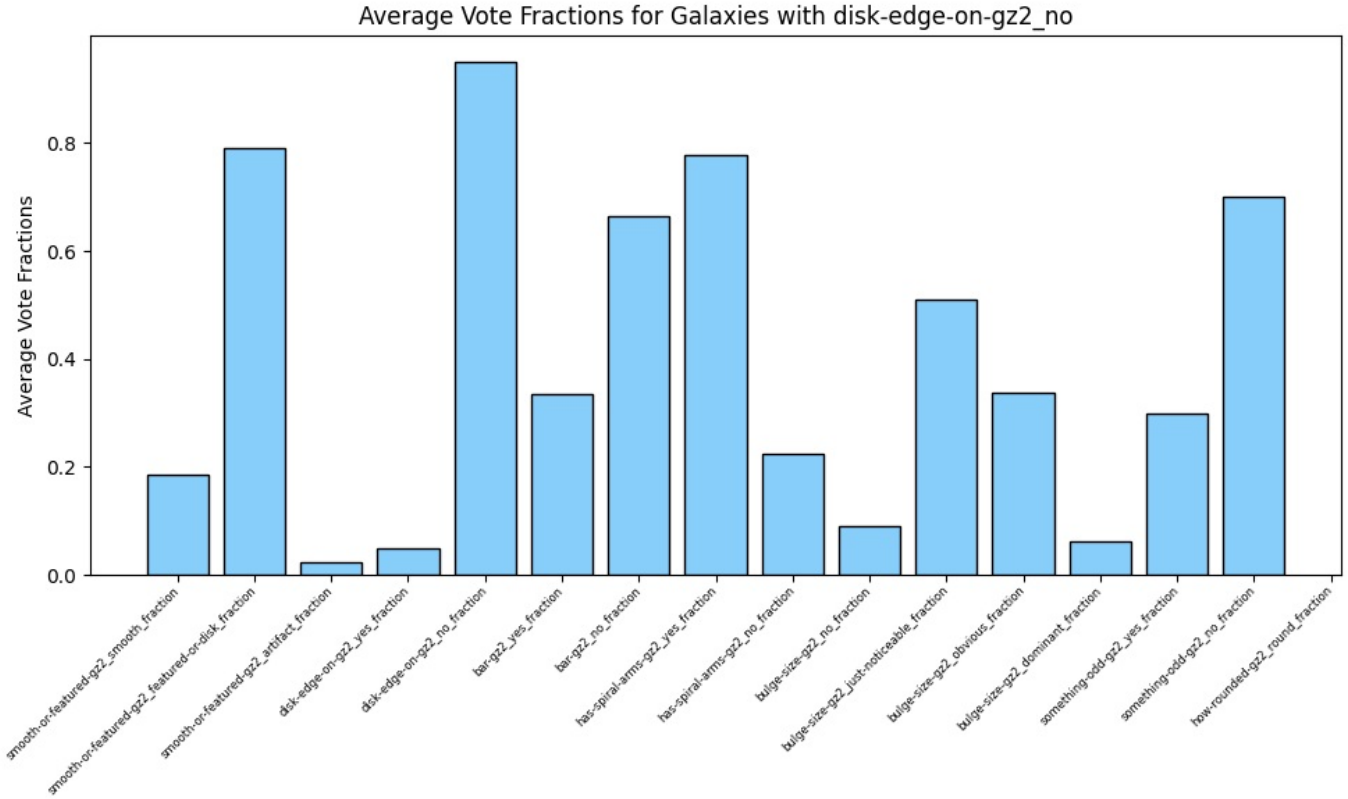
```
]

def plot_label_dist(semantic_labels, df, color='lightskyblue'):
    for semantic_label in semantic_labels:
        feature_to_id[semantic_label] = []

    for idx, _ in enumerate(df['id_str']):
        id = df['id_str'][idx]
        for semantic_label in semantic_labels:
            feature_value = df[semantic_label][idx]
            feature_to_id[feature_value].append(id)

    for feature, galaxies in feature_to_id.items():
        if len(galaxies) == 0:
            continue
        if feature == 'deadlock' or feature == '-':
            return
        fractions = {key: [] for key in keys}

        for galaxy_id in galaxies:
            galaxy_idx = df.index[df['id_str'] == galaxy_id][0]
            for k in keys:
                if df[k].loc[galaxy_idx] is None:
                    fractions[k].append(0)
                else:
                    fractions[k].append(df[k].loc[galaxy_idx])

        avg_fractions = {k: np.mean(v) for k, v in fractions.items()}

        plt.figure(figsize=(10, 6))
        plt.bar(avg_fractions.keys(), avg_fractions.values(), color=color, edgecolor='black')
        plt.ylabel('Average Vote Fractions')
        plt.title(f'Average Vote Fractions for Galaxies with {feature}')

        plt.xticks(rotation=45, fontsize=6, ha='right', va='center', rotation_mode='anchor')
        plt.tight_layout()
        plt.show()

plot_label_dist(semantic_labels, df, color='lightskyblue')
```

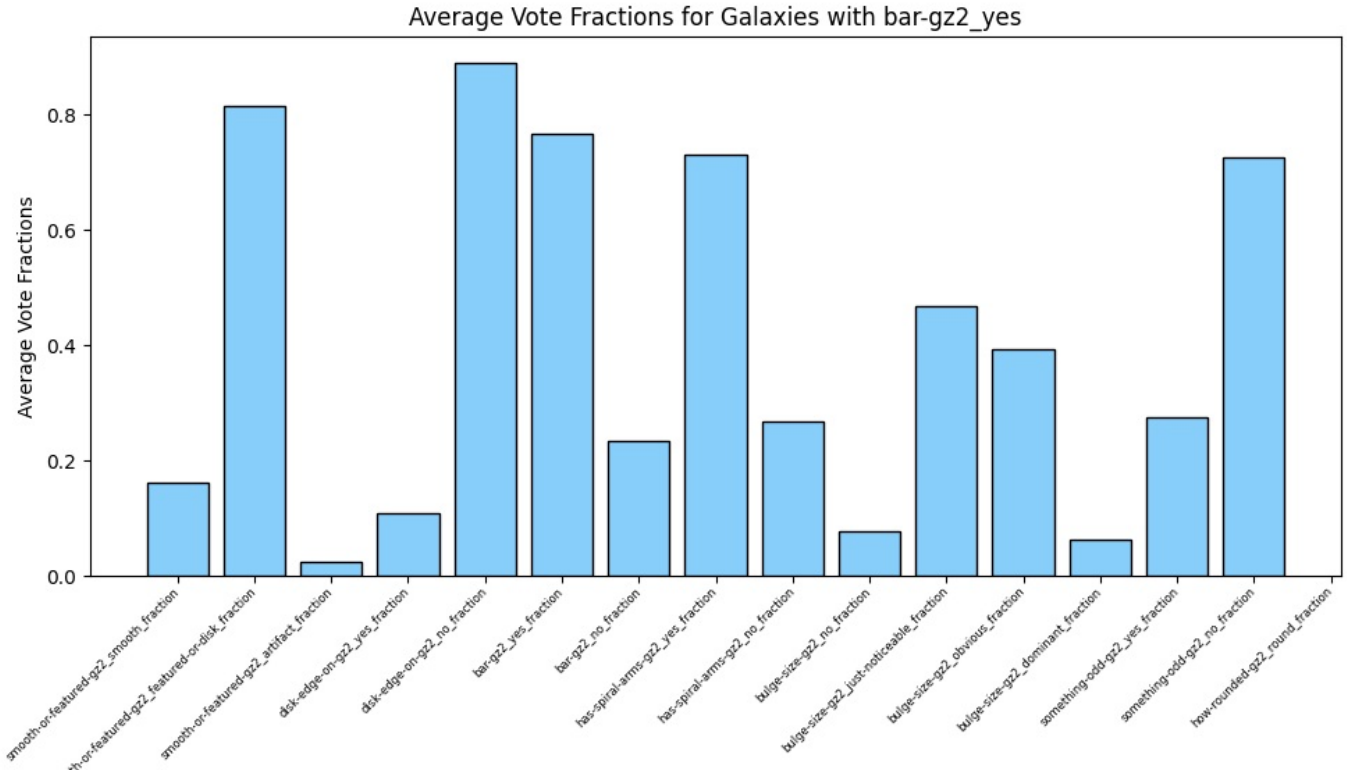


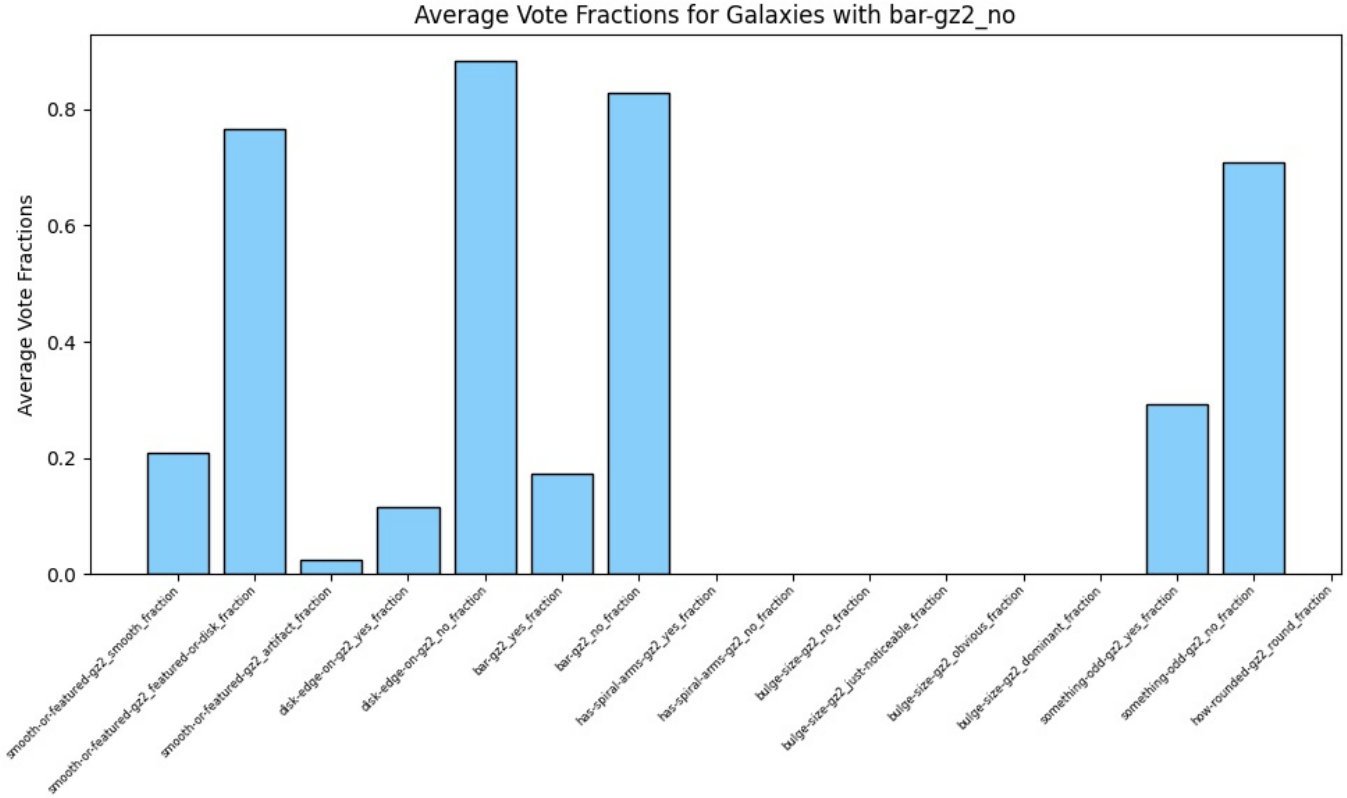

Average Vote Fractions for Galaxies with smooth-or-featured-gz2_featured-or-disk
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

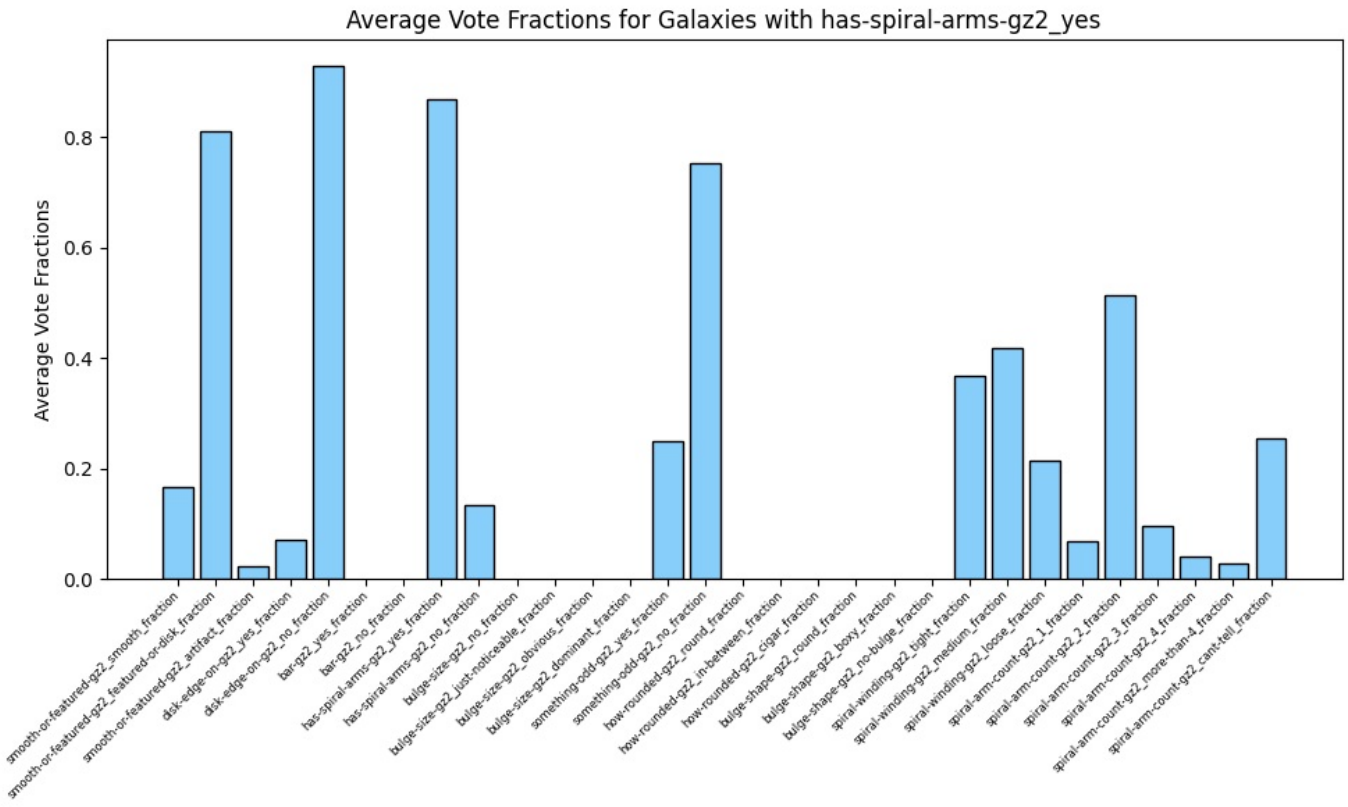

Average Vote Fractions for Galaxies with smooth-or-featured-gz2_artifact
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction

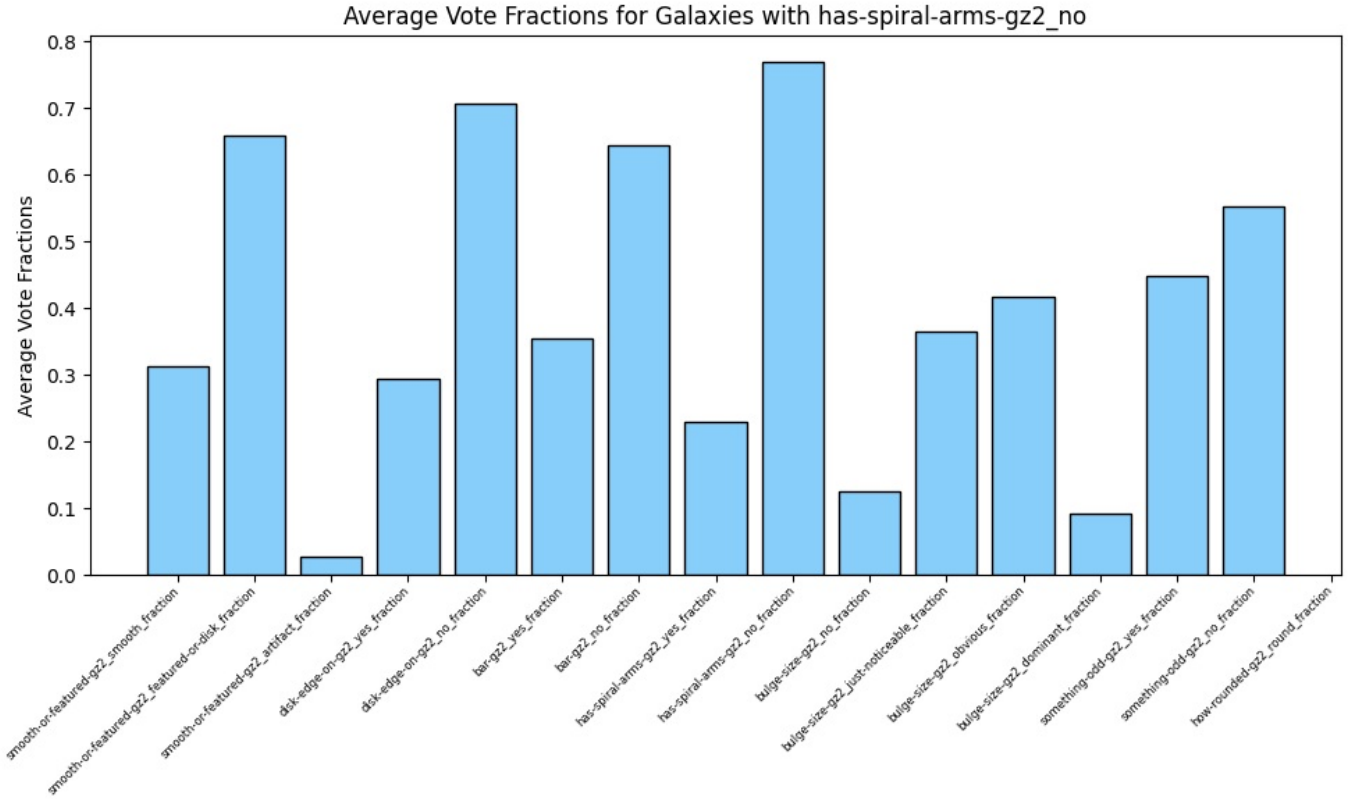

Average Vote Fractions for Galaxies with disk-edge-on-gz2_yes
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction

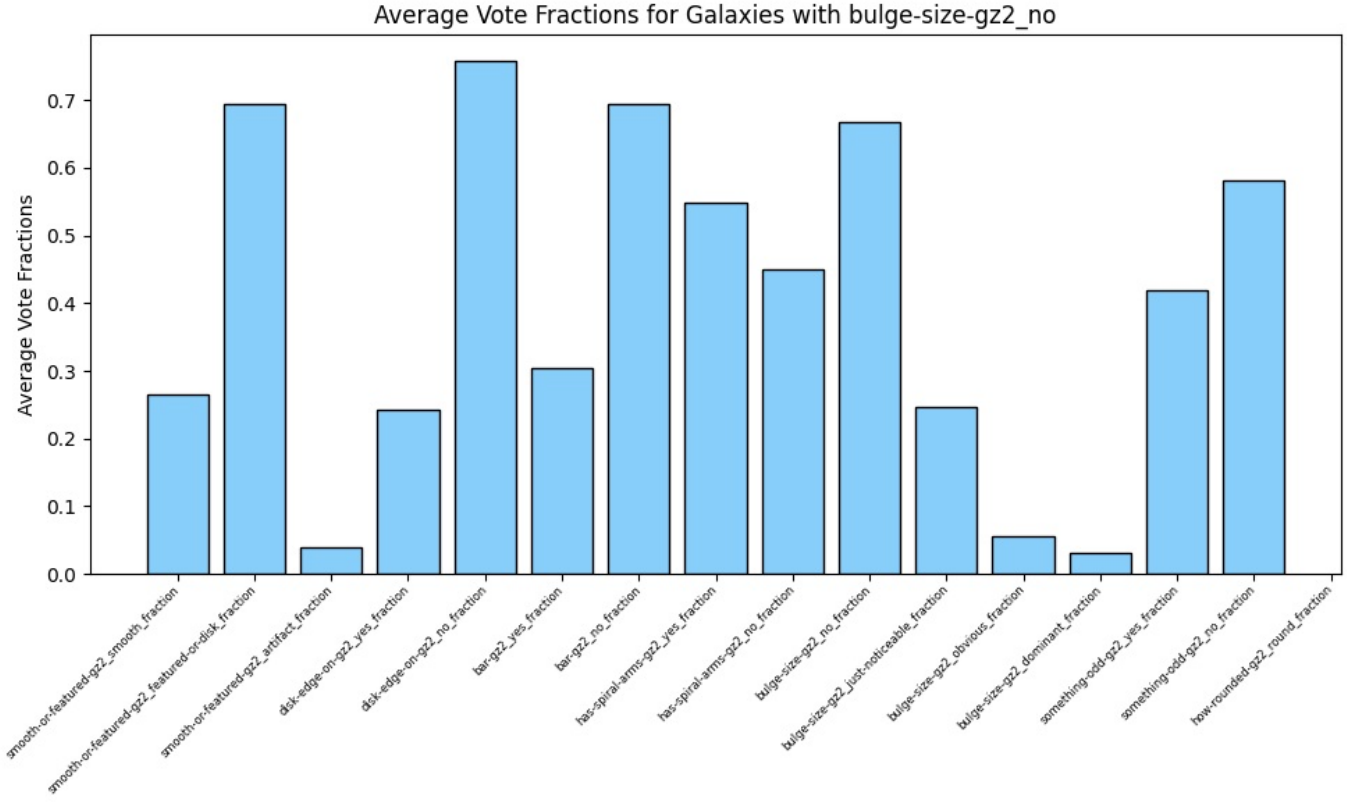

Average Vote Fractions for Galaxies with disk-edge-on-gz2_no
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

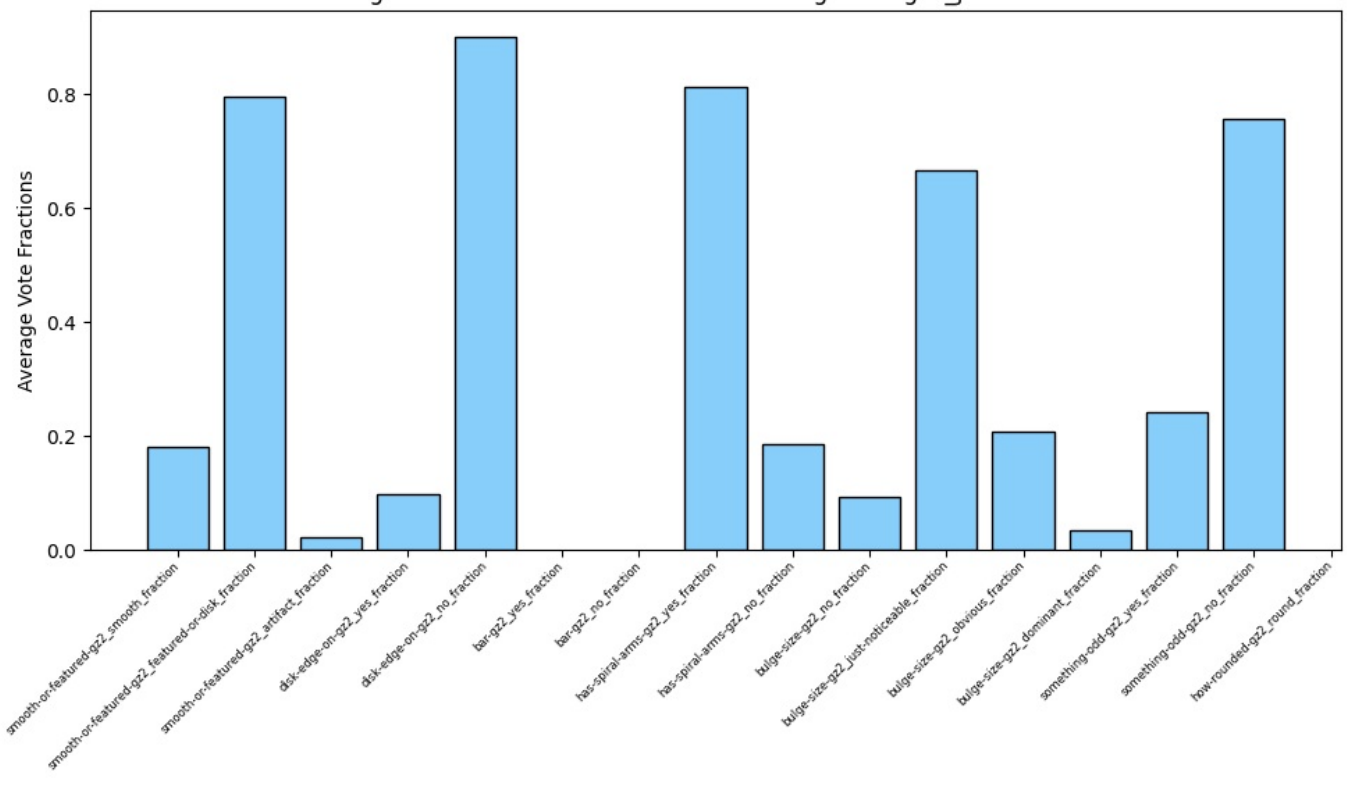

Average Vote Fractions for Galaxies with bar-gz2_yes
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

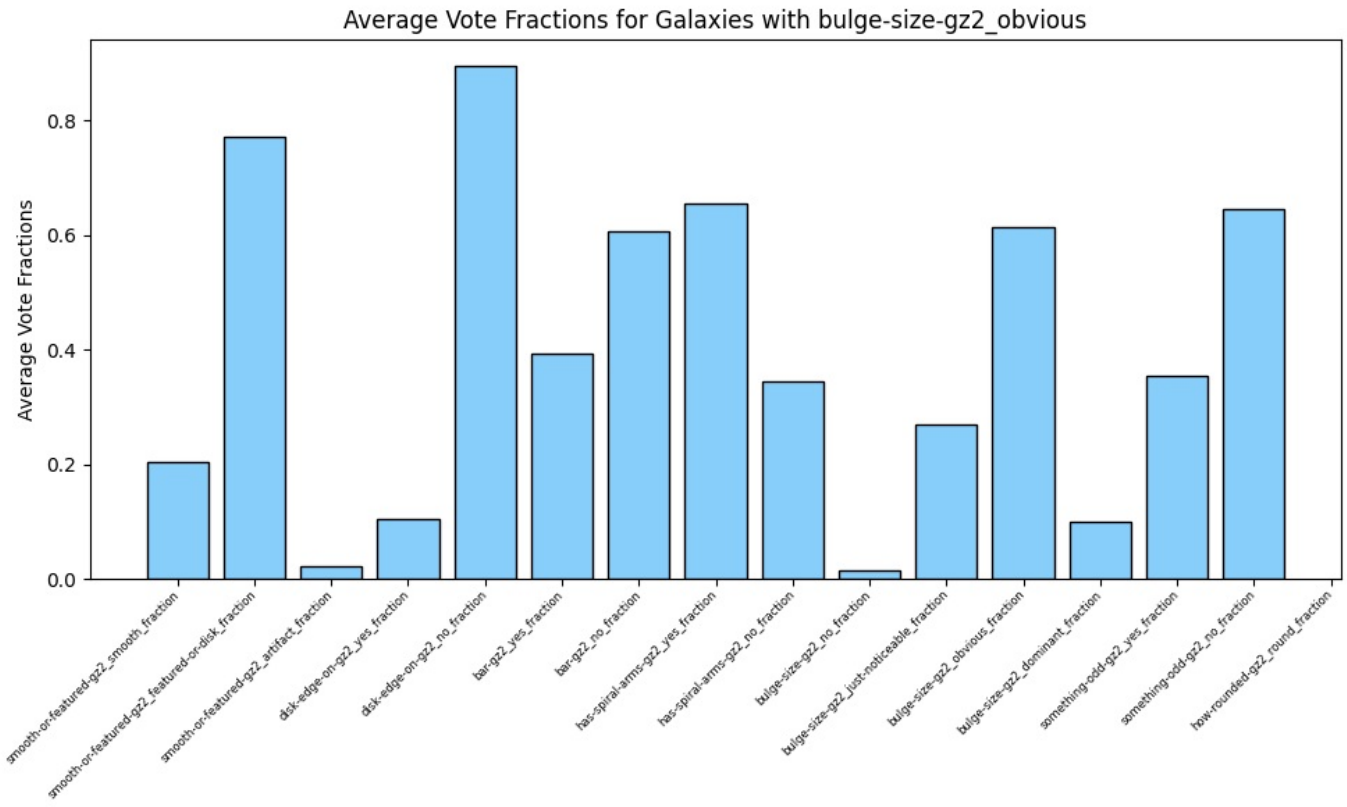

Average Vote Fractions for Galaxies with bar-gz2_no
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

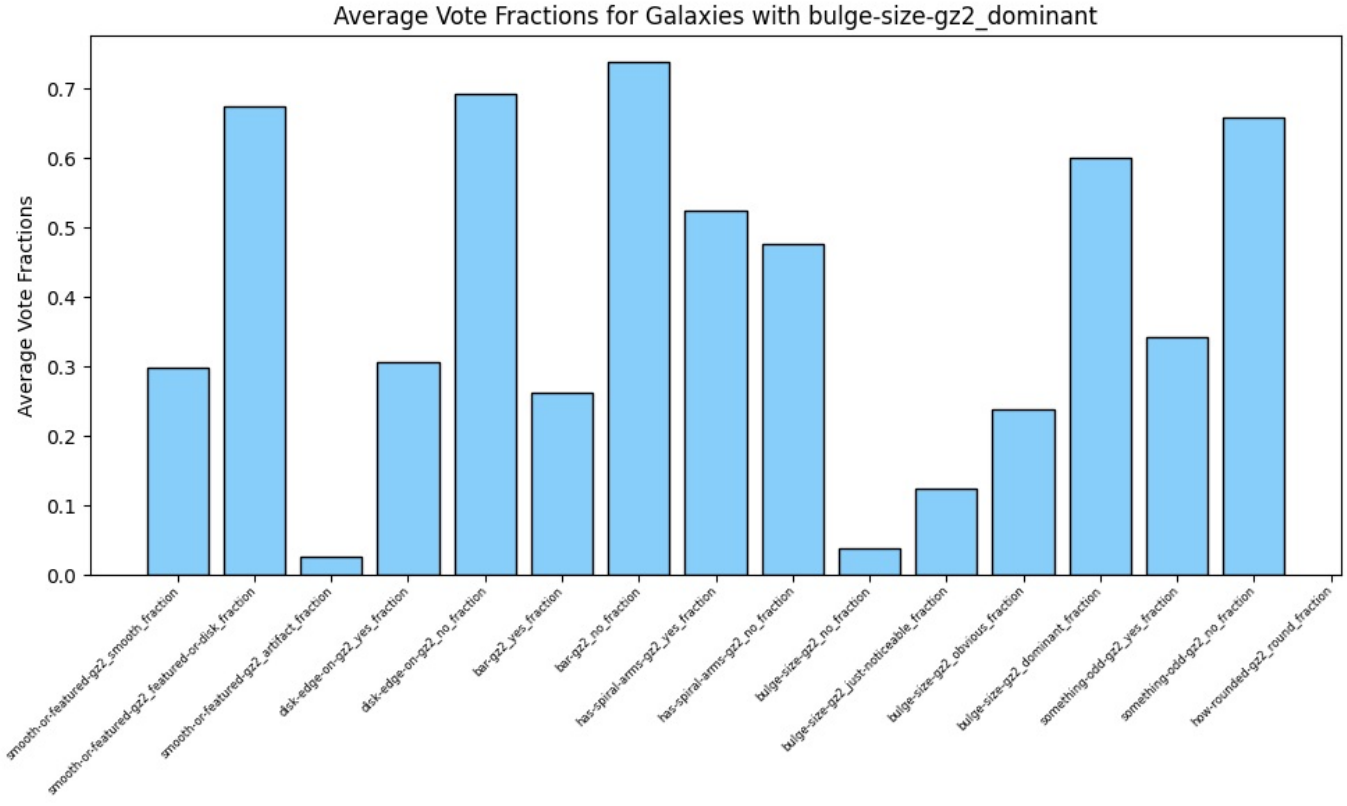

Average Vote Fractions for Galaxies with has-spiral-arms-gz2_yes
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction

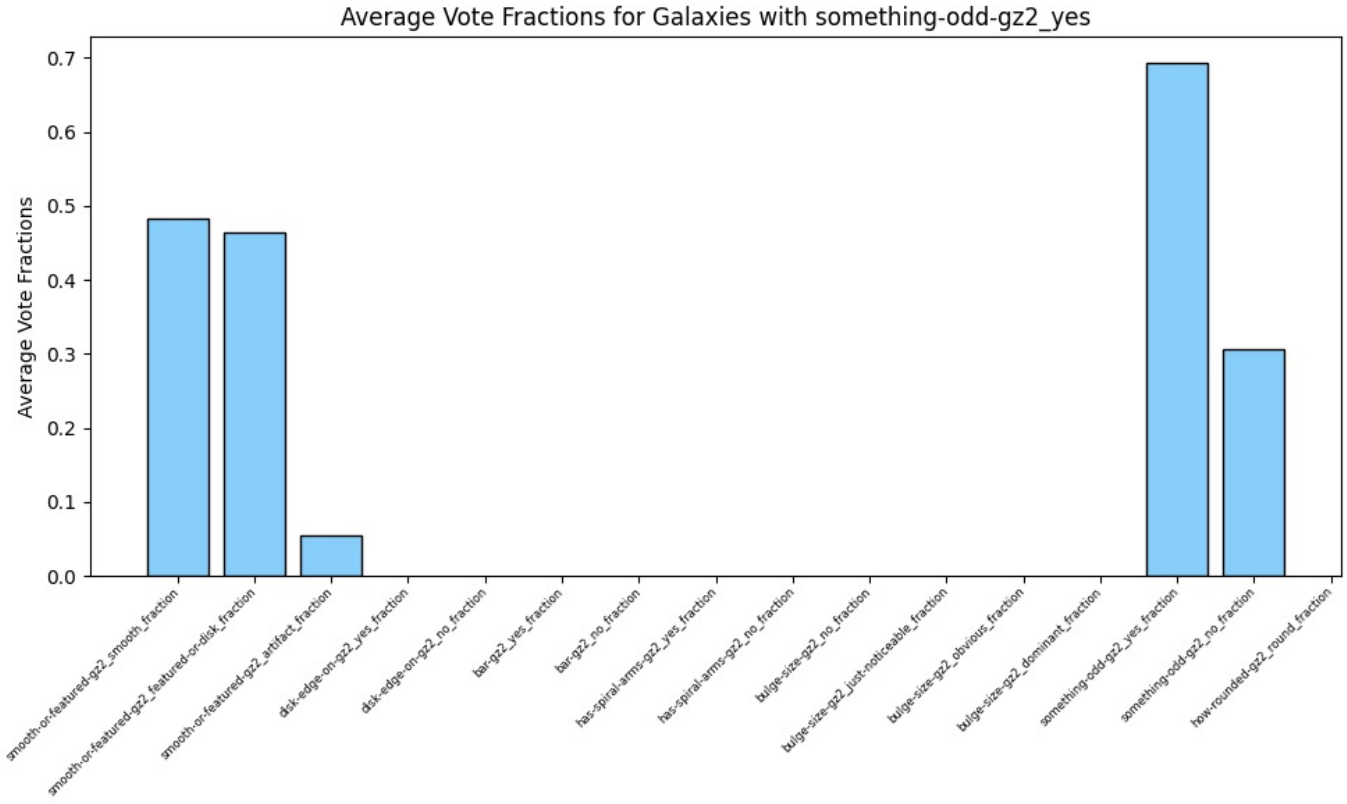

Average Vote Fractions for Galaxies with has-spiral-arms-gz2_no
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

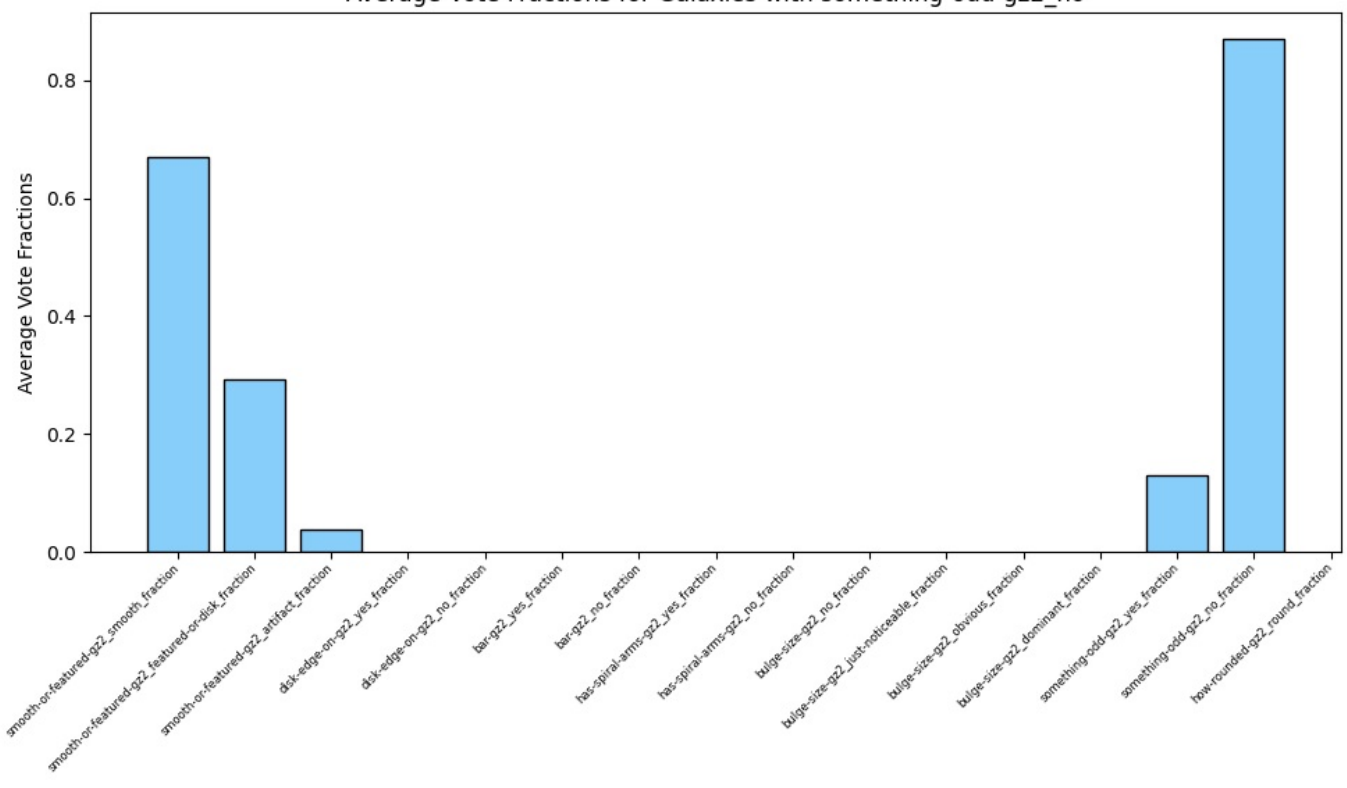

Average Vote Fractions for Galaxies with bulge-size-gz2_no
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

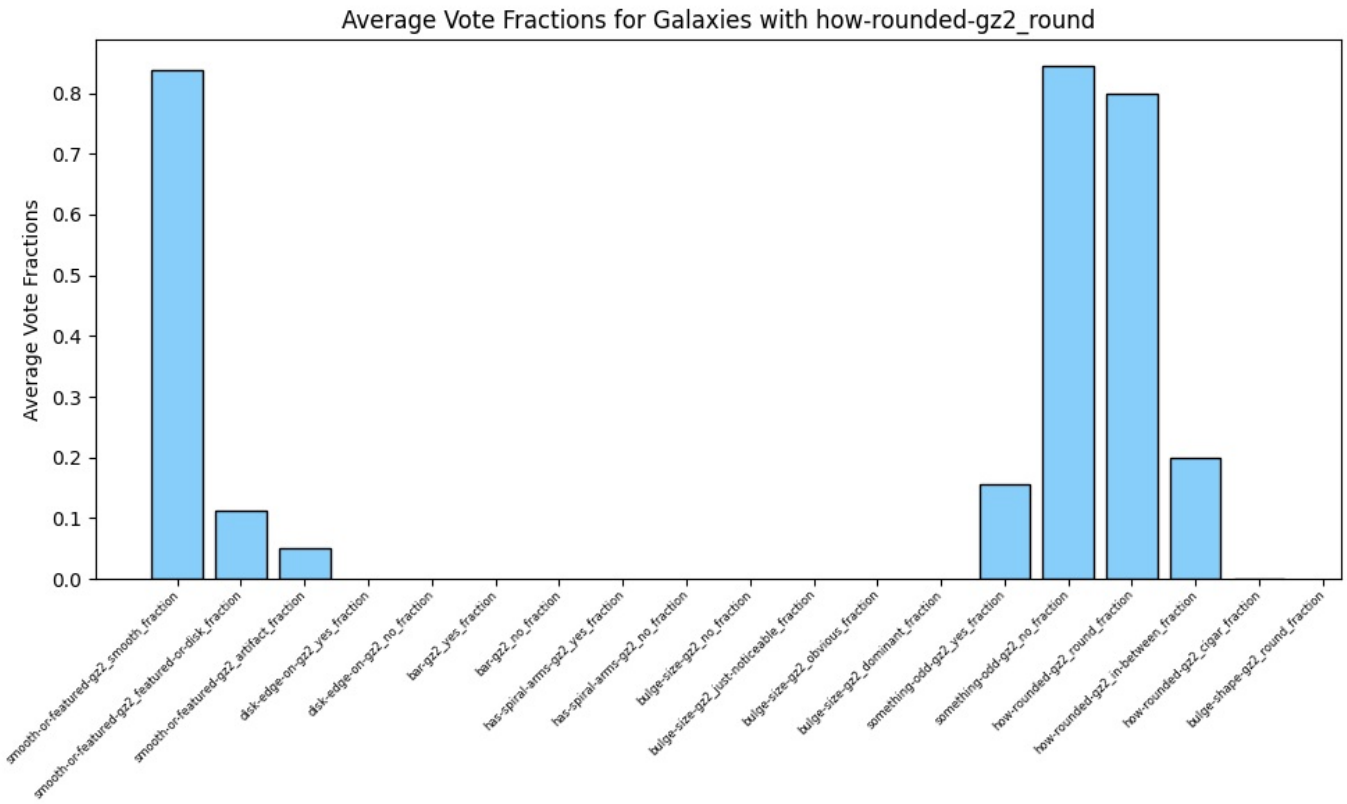

Average Vote Fractions for Galaxies with bulge-size-gz2_just-noticeable
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

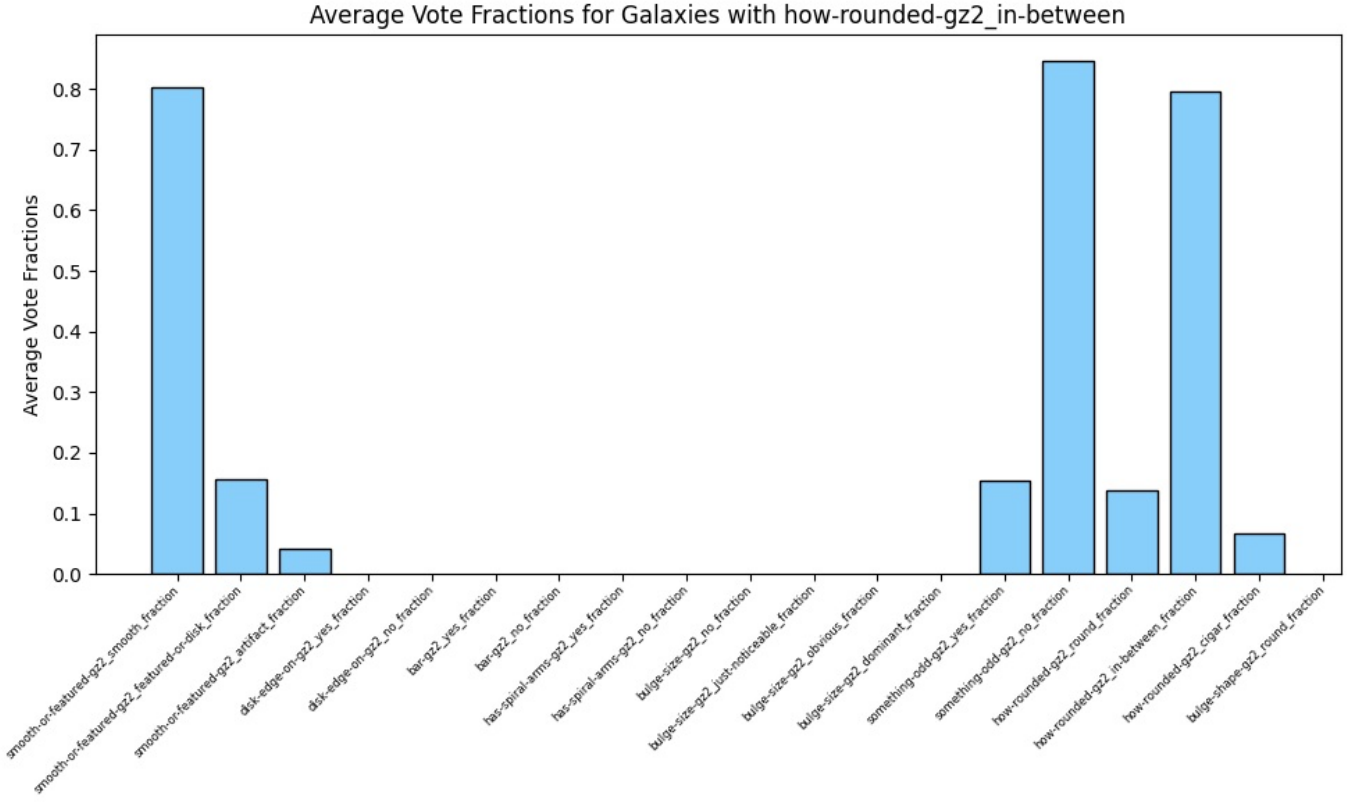

Average Vote Fractions for Galaxies with bulge-size-gz2_obvious
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

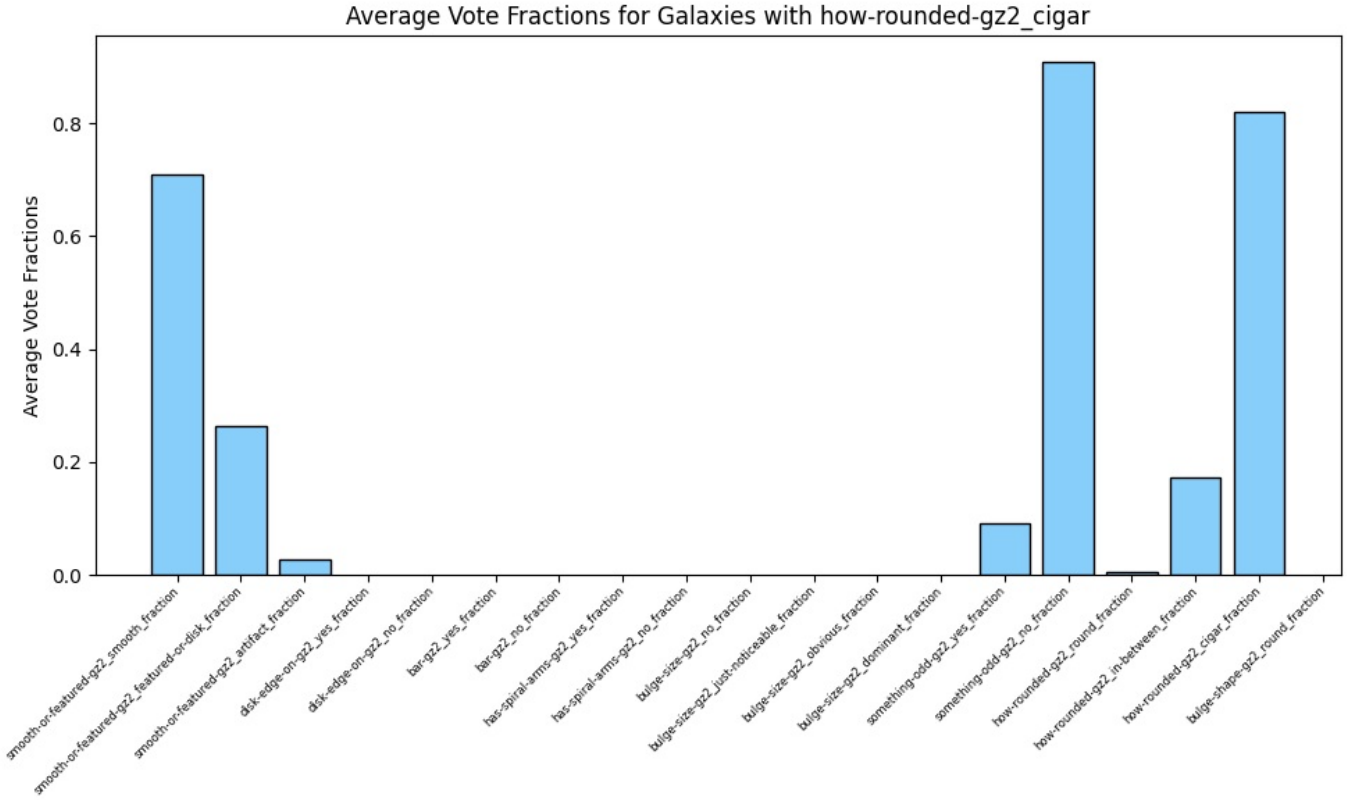

Average Vote Fractions for Galaxies with bulge-size-gz2_dominant
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

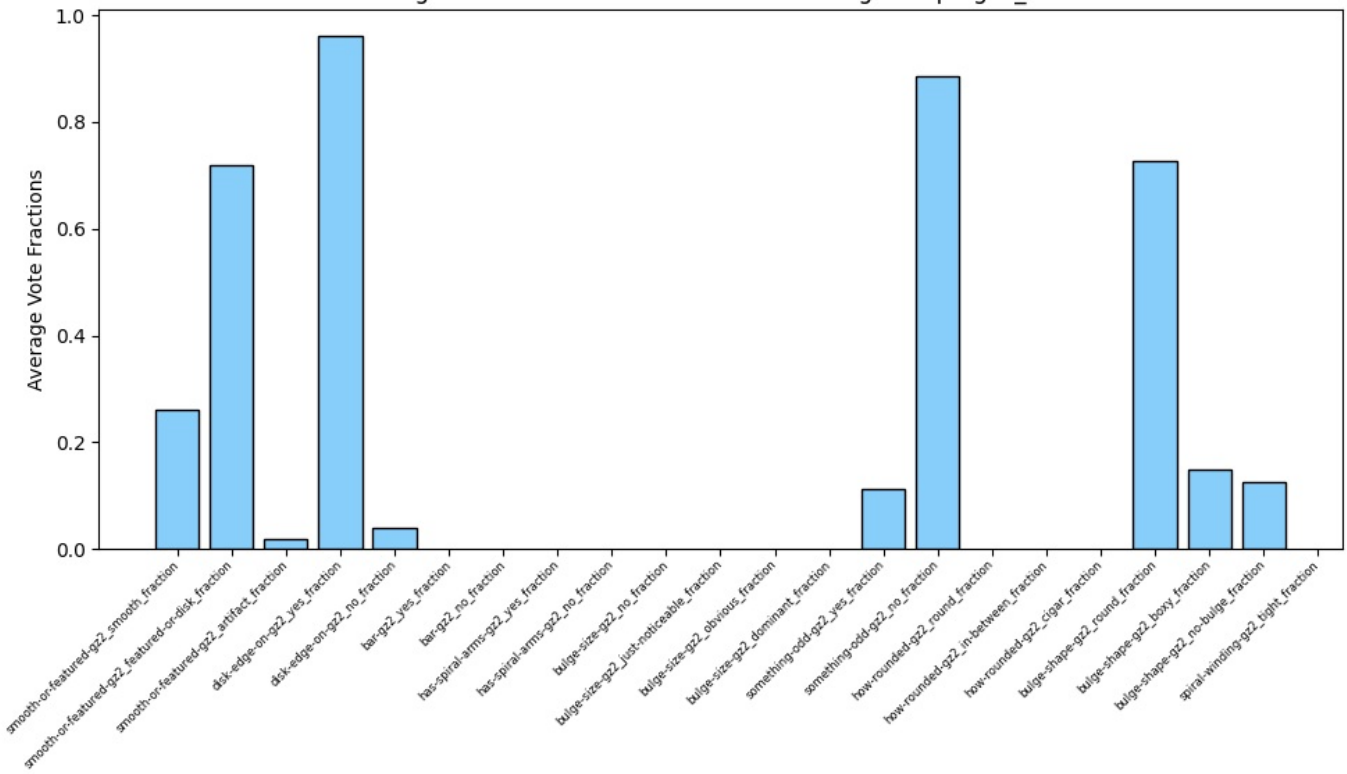

Average Vote Fractions for Galaxies with something-odd-gz2_yes
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

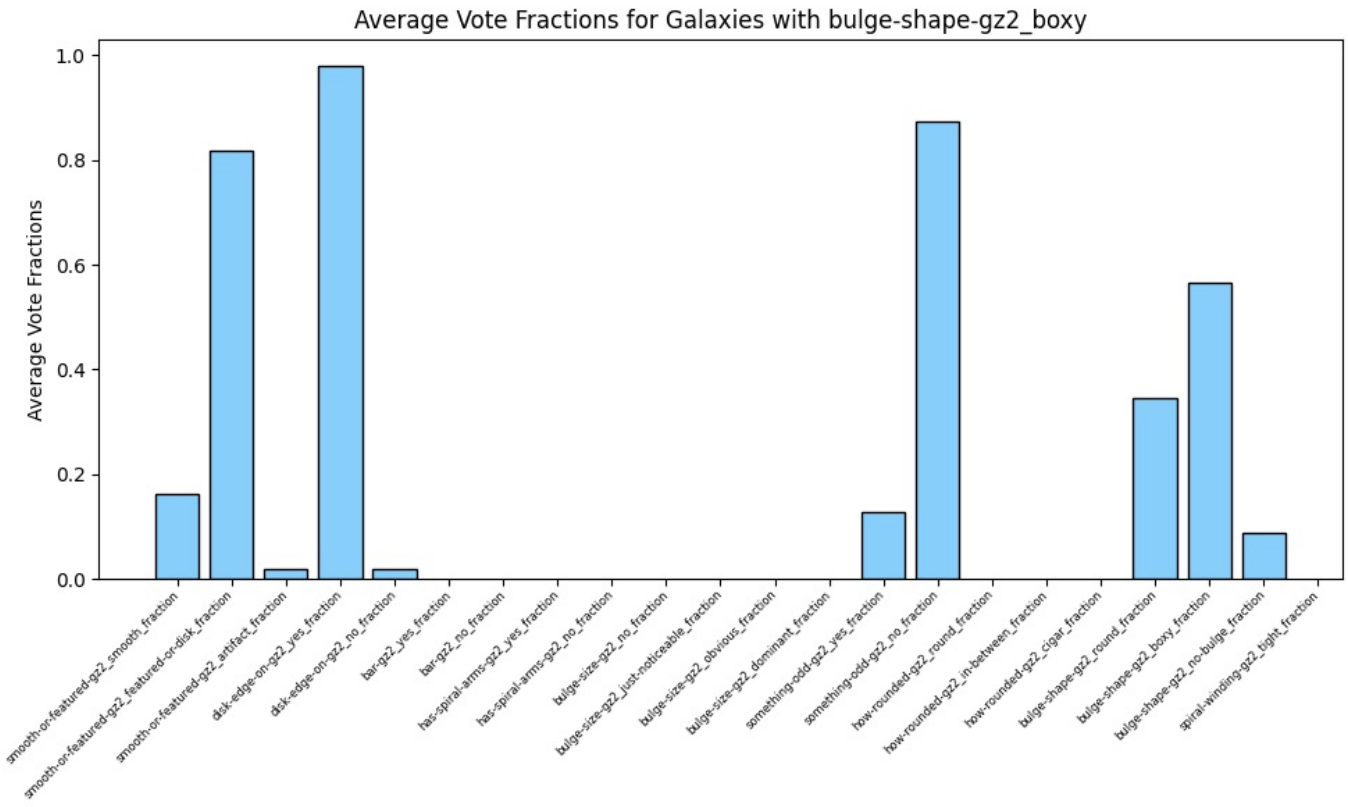

Average Vote Fractions for Galaxies with something-odd-gz2_no
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction

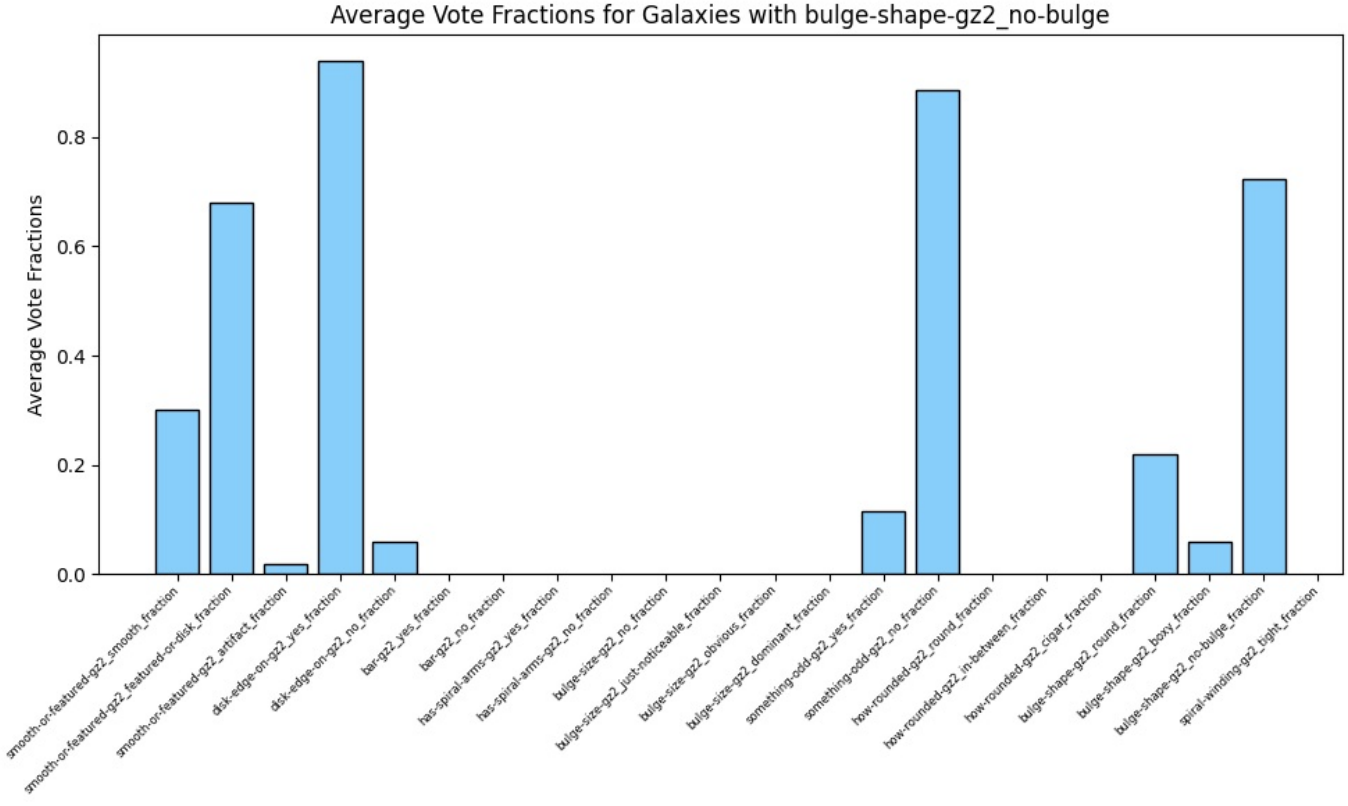

Average Vote Fractions for Galaxies with how-rounded-gz2_round
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction

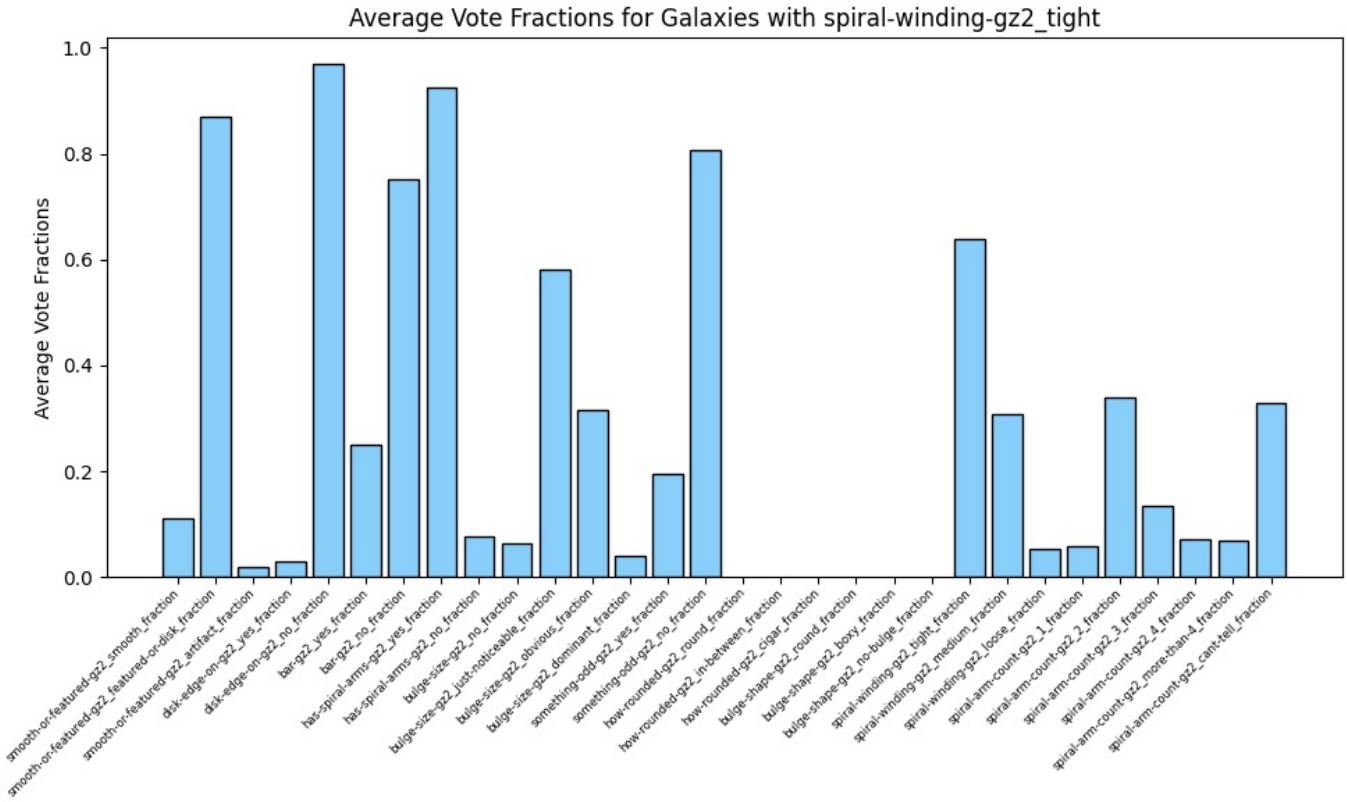

Average Vote Fractions for Galaxies with how-rounded-gz2_in-between
Average Vote Fractions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction

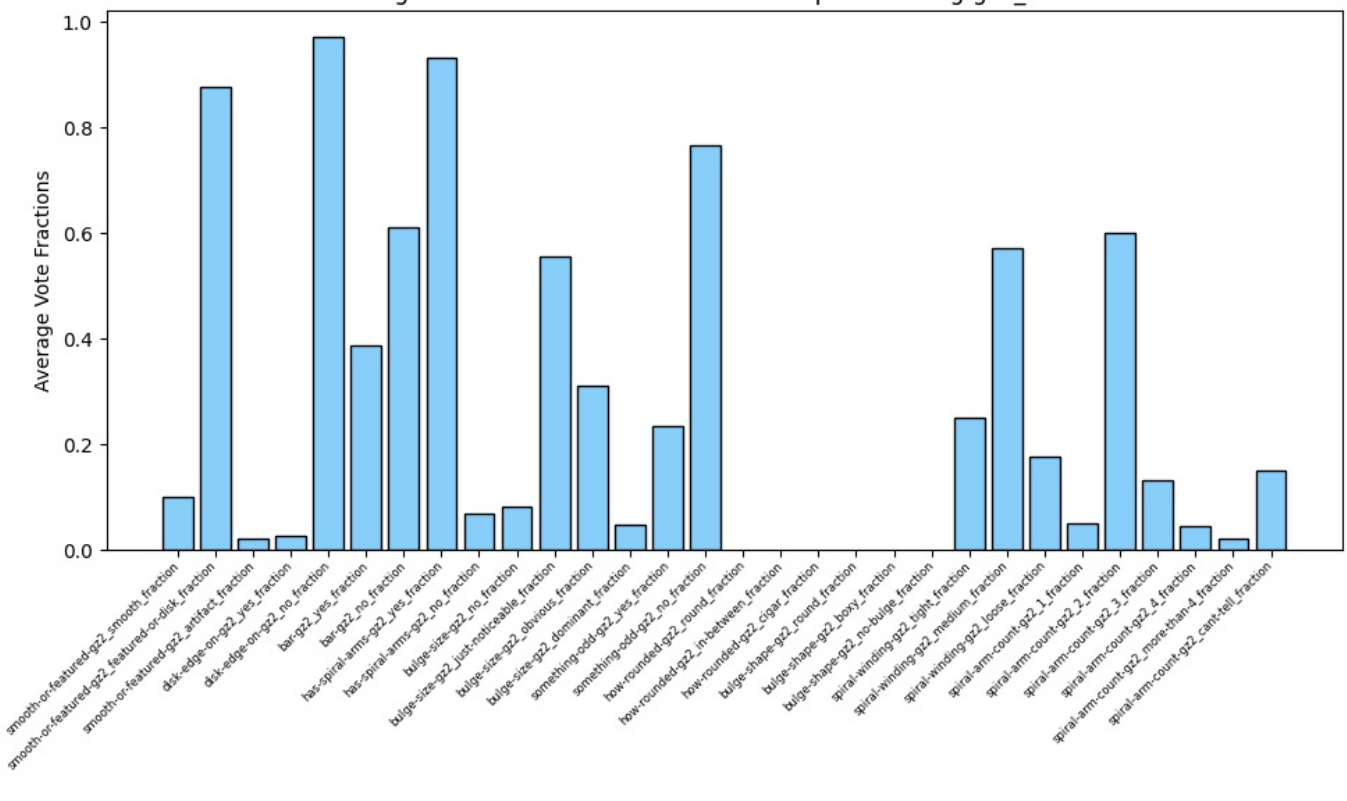

Average Vote Fractions for Galaxies with how-rounded-gz2_cigar
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction

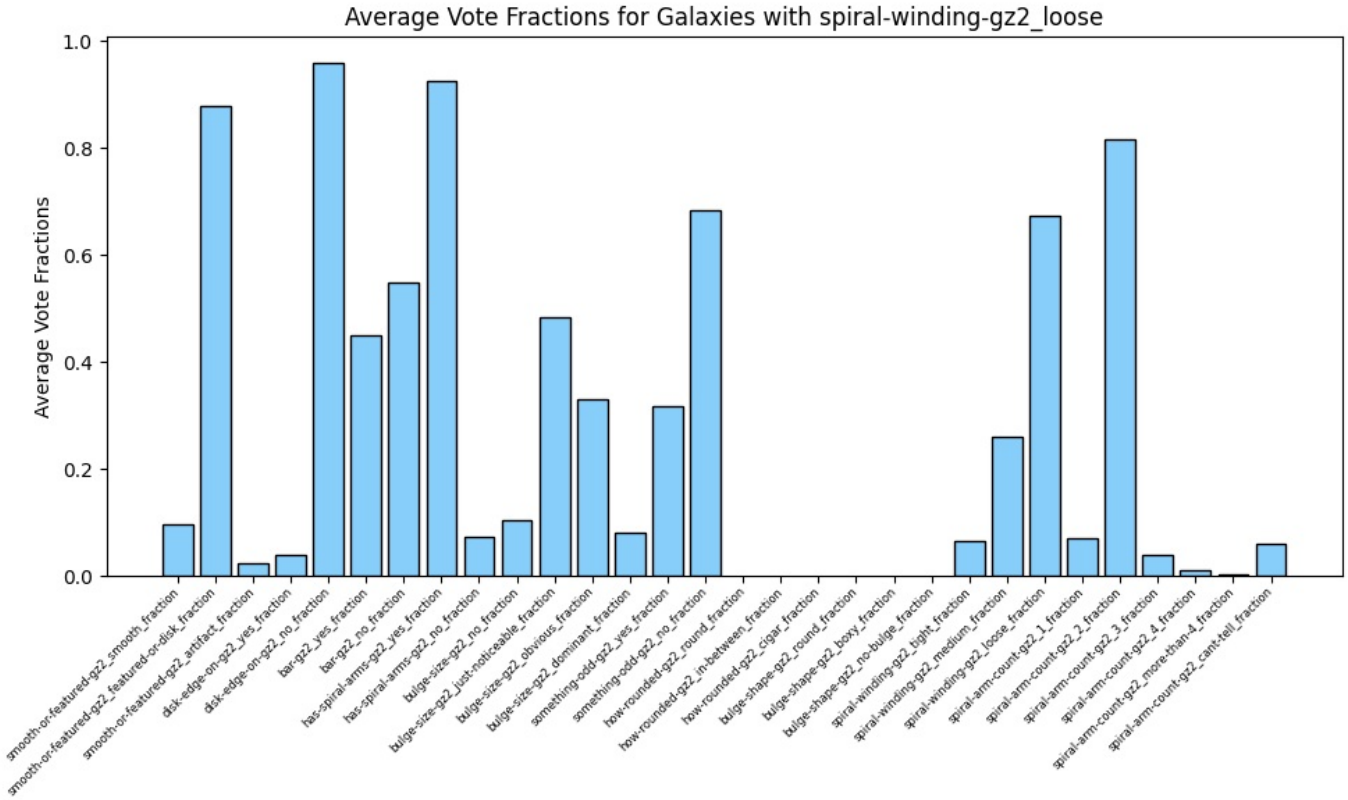

Average Vote Fractions for Galaxies with bulge-shape-gz2_round
Average Vote Fractions
0.0
0.2
0.4
0.6
0.8
1.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction

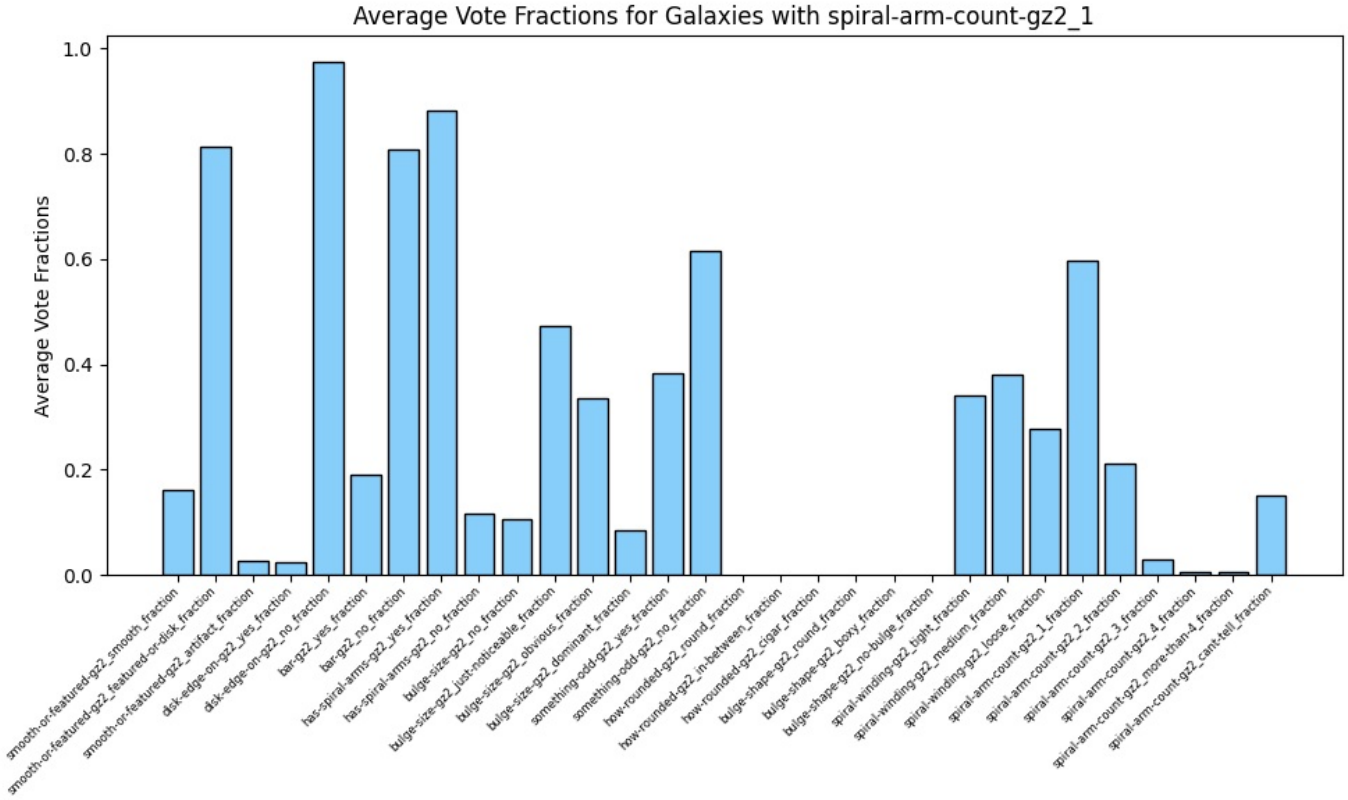

Average Vote Fractions for Galaxies with bulge-shape-gz2_boxy
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction

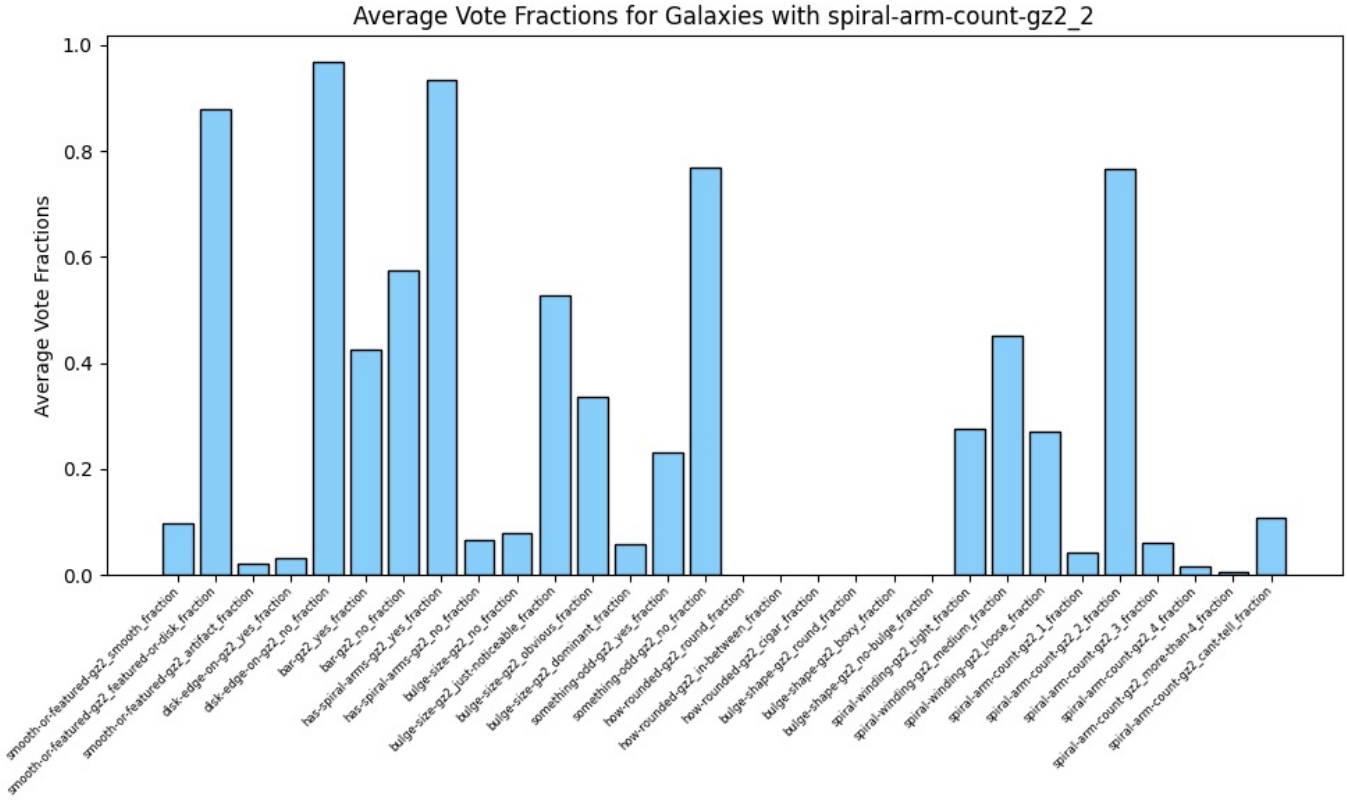

Average Vote Fractions for Galaxies with bulge-shape-gz2_no-bulge
Average Vote Fractions
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction

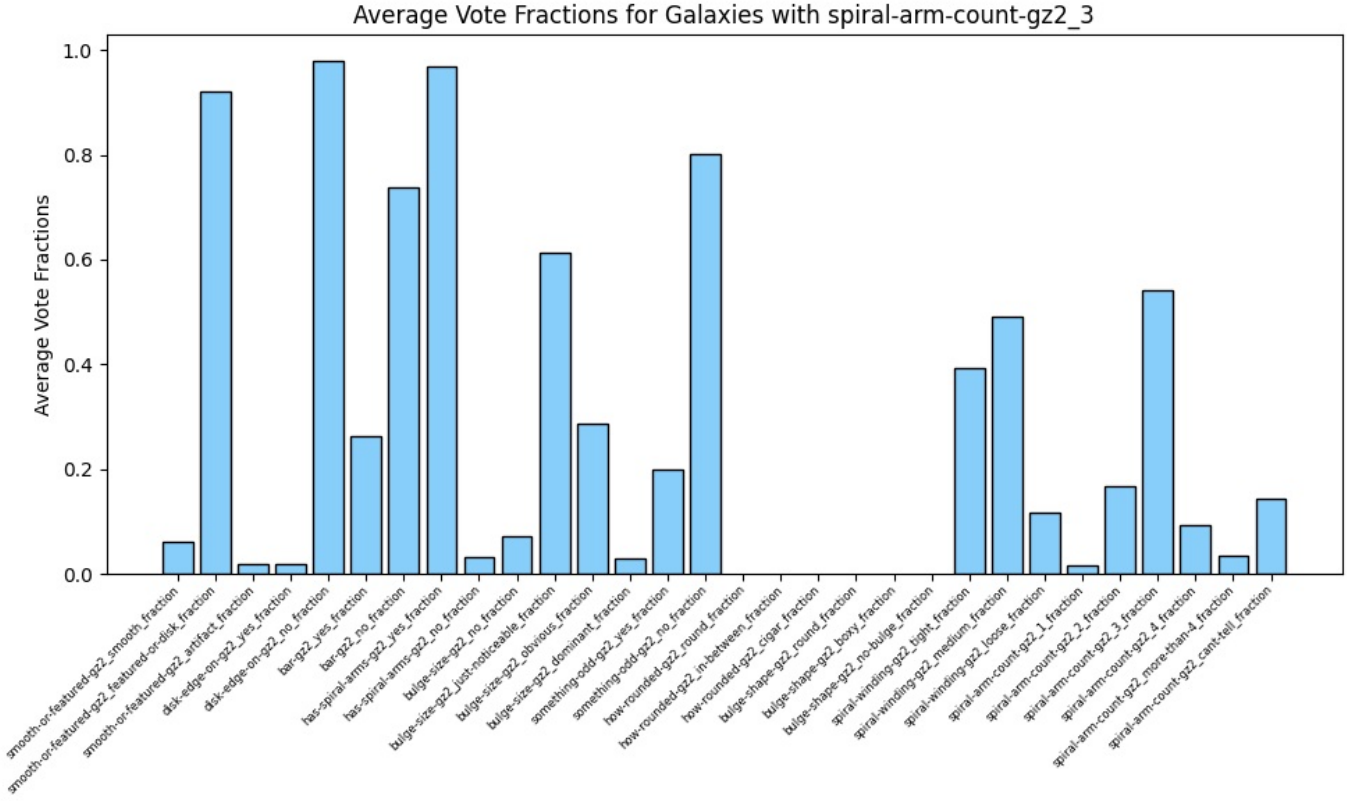

Average Vote Fractions for Galaxies with spiral-winding-gz2_tight
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction

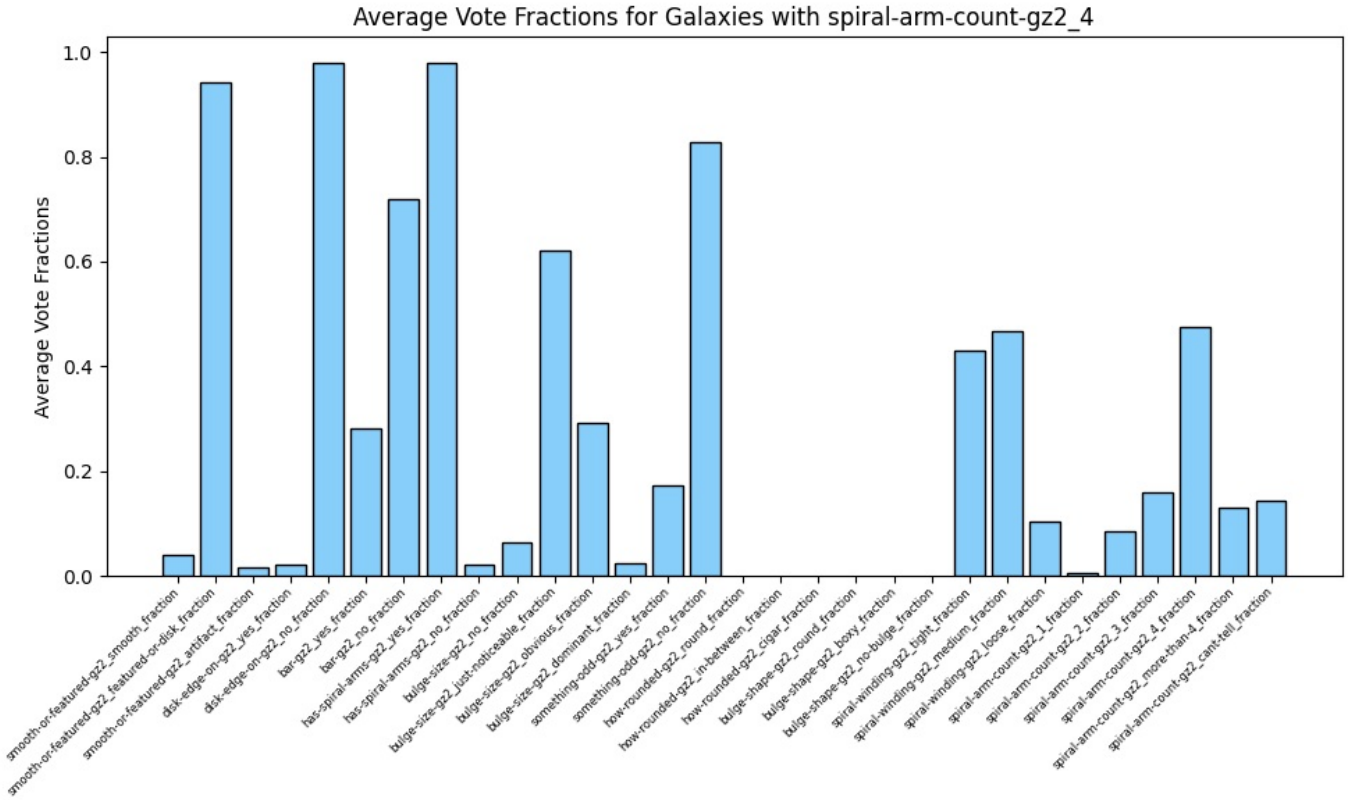

Average Vote Fractions for Galaxies with spiral-winding-gz2_medium
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction

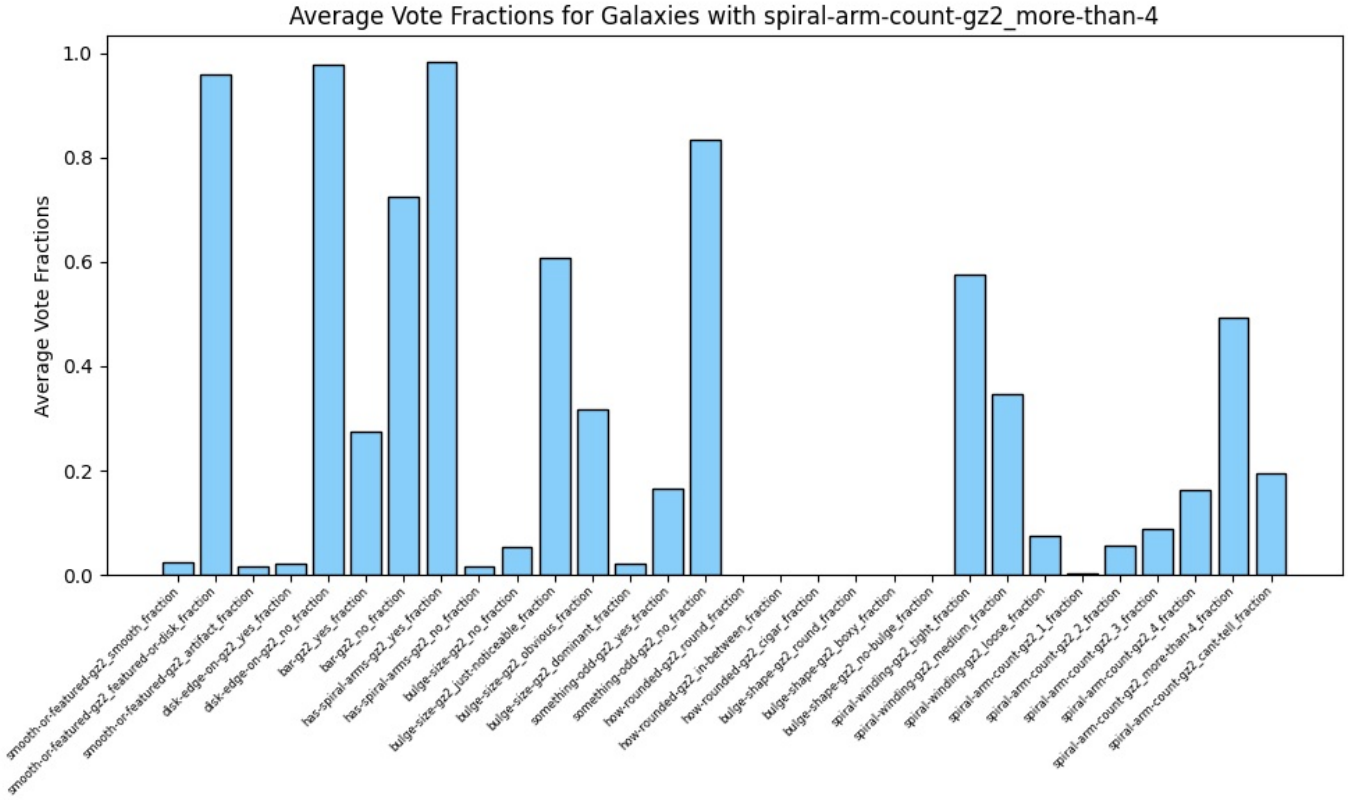

Average Vote Fractions for Galaxies with spiral-winding-gz2_loose
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction

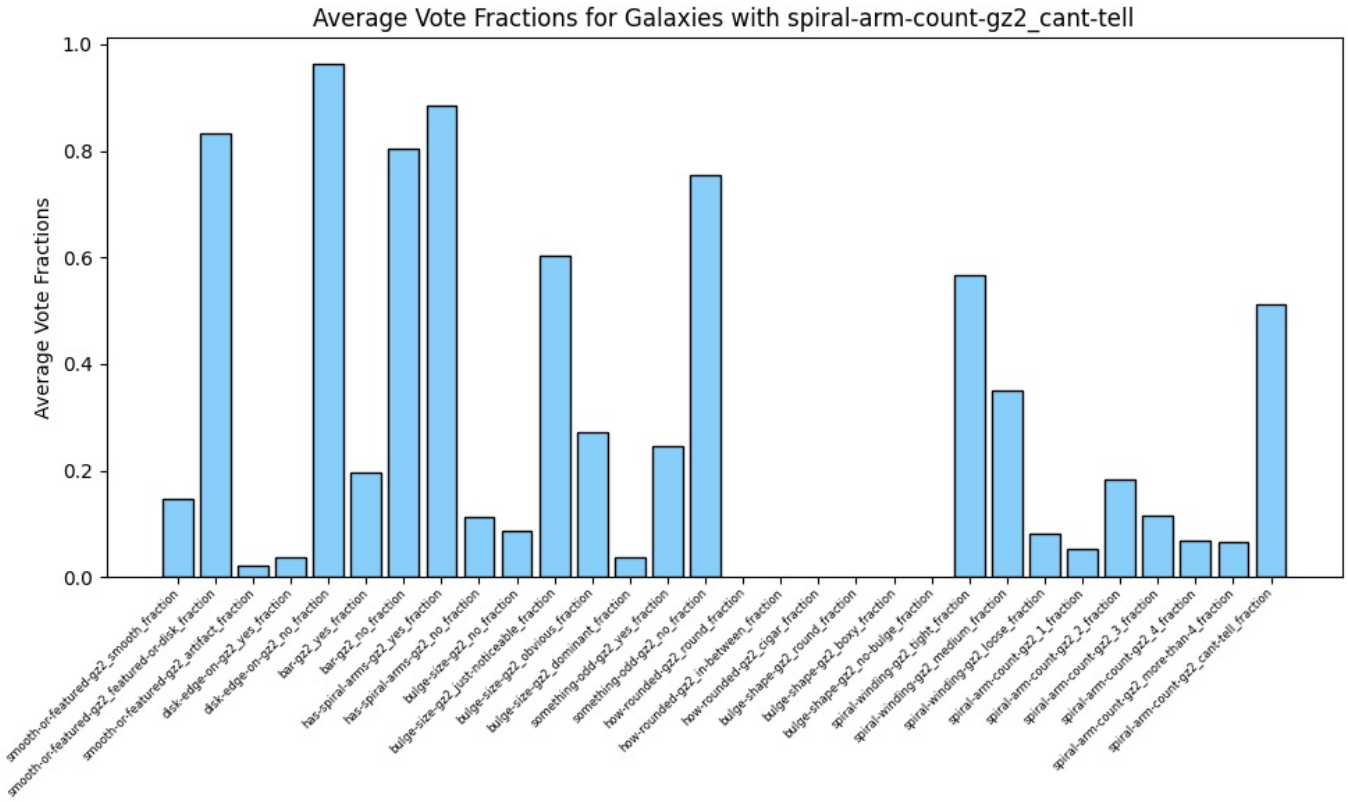

Average Vote Fractions for Galaxies with spiral-arm-count-gz2_1
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction


Average Vote Fractions for Galaxies with spiral-arm-count-gz2_2
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction


Average Vote Fractions for Galaxies with spiral-arm-count-gz2_3
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction


Average Vote Fractions for Galaxies with spiral-arm-count-gz2_4
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction


Average Vote Fractions for Galaxies with spiral-arm-count-gz2_more-than-4
Average Vote Fractions
1.0
0.8
0.6
0.4
0.2
0.0
smooth-or-featured-gz2_smooth_fraction
smooth-or-featured-gz2_featured-or-disk_fraction
smooth-or-featured-gz2_artifact_fraction
disk-edge-on-gz2_yes_fraction
disk-edge-on-gz2_no_fraction
bar-gz2_yes_fraction
bar-gz2_no_fraction
has-spiral-arms-gz2_yes_fraction
has-spiral-arms-gz2_no_fraction
bulge-size-gz2_no_fraction
bulge-size-gz2_just-noticeable_fraction
bulge-size-gz2_obvious_fraction
bulge-size-gz2_dominant_fraction
something-odd-gz2_yes_fraction
something-odd-gz2_no_fraction
how-rounded-gz2_round_fraction
how-rounded-gz2_in-between_fraction
how-rounded-gz2_cigar_fraction
bulge-shape-gz2_round_fraction
bulge-shape-gz2_boxy_fraction
bulge-shape-gz2_no-bulge_fraction
spiral-winding-gz2_tight_fraction
spiral-winding-gz2_medium_fraction
spiral-winding-gz2_loose_fraction
spiral-arm-count-gz2_1_fraction
spiral-arm-count-gz2_2_fraction
spiral-arm-count-gz2_3_fraction
spiral-arm-count-gz2_4_fraction
spiral-arm-count-gz2_more-than-4_fraction
spiral-arm-count-gz2_cant-tell_fraction



these are plots of the average vote fractions for every feature over all the galaxies of a specific feature.

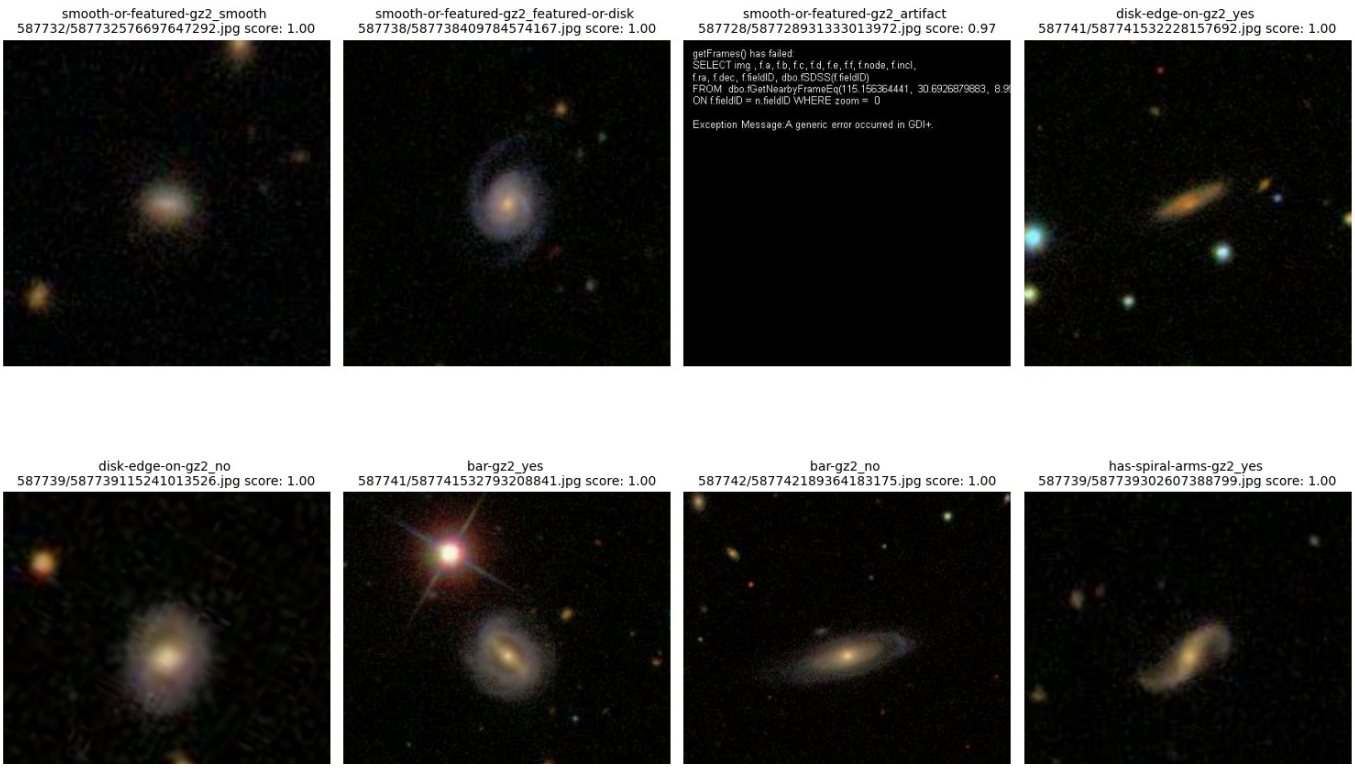
Malachy - Galaxies with the most confidence in their voted features.

In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

"""
The Galaxies with the higest vote confidence for a given feature.
The dataset was filtered such that galaxies with fewer than (the average number of votes plus 2 standard deviati
for a given feature were excluded to prevent galaxies with comically small total votes, and therefore inaccurate
from dominating the dataset.
"""

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

df = pd.read_parquet(parquet_file_path)

features = {
    'smooth-or-featured-gz2_smooth_fraction': [],
    'smooth-or-featured-gz2_featured-or-disk_fraction': [],
    'smooth-or-featured-gz2_artifact_fraction': [],
    'disk-edge-on-gz2_yes_fraction': [],
    'disk-edge-on-gz2_no_fraction': [],
    'bar-gz2_yes_fraction': [],
    'bar-gz2_no_fraction': [],
    'has-spiral-arms-gz2_yes_fraction': [],
    'has-spiral-arms-gz2_no_fraction': [],
    'bulge-size-gz2_no_fraction': [],
    'bulge-size-gz2_just-noticeable_fraction': [],
    'bulge-size-gz2_obvious_fraction': [],
    'bulge-size-gz2_dominant_fraction': [],
```

```
    'something-odd-gz2_yes_fraction': [],
    'something-odd-gz2_no_fraction': [],
    'how-rounded-gz2_round_fraction': [],
    'how-rounded-gz2_in-between_fraction': [],
    'how-rounded-gz2_cigar_fraction': [],
    'bulge-shape-gz2_round_fraction': [],
    'bulge-shape-gz2_boxy_fraction': [],
    'bulge-shape-gz2_no-bulge_fraction': [],
    'spiral-winding-gz2_tight_fraction': [],
    'spiral-winding-gz2_medium_fraction': [],
    'spiral-winding-gz2_loose_fraction': [],
    'spiral-arm-count-gz2_1_fraction': [],
    'spiral-arm-count-gz2_2_fraction': [],
    'spiral-arm-count-gz2_3_fraction': [],
    'spiral-arm-count-gz2_4_fraction': [],
    'spiral-arm-count-gz2_more-than-4_fraction': [],
    'spiral-arm-count-gz2_cant-tell_fraction': [],
}

keys = [
    'smooth-or-featured-gz2_smooth_fraction',
    'smooth-or-featured-gz2_featured-or-disk_fraction',
    'smooth-or-featured-gz2_artifact_fraction',
    'disk-edge-on-gz2_yes_fraction',
    'disk-edge-on-gz2_no_fraction',
    'bar-gz2_yes_fraction',
    'bar-gz2_no_fraction',
    'has-spiral-arms-gz2_yes_fraction',
    'has-spiral-arms-gz2_no_fraction',
    'bulge-size-gz2_no_fraction',
    'bulge-size-gz2_just-noticeable_fraction',
    'bulge-size-gz2_obvious_fraction',
    'bulge-size-gz2_dominant_fraction',
    'something-odd-gz2_yes_fraction',
    'something-odd-gz2_no_fraction',
    'how-rounded-gz2_round_fraction',
    'how-rounded-gz2_in-between_fraction',
    'how-rounded-gz2_cigar_fraction',
    'bulge-shape-gz2_round_fraction',
    'bulge-shape-gz2_boxy_fraction',
    'bulge-shape-gz2_no-bulge_fraction',
    'spiral-winding-gz2_tight_fraction',
    'spiral-winding-gz2_medium_fraction',
    'spiral-winding-gz2_loose_fraction',
    'spiral-arm-count-gz2_1_fraction',
    'spiral-arm-count-gz2_2_fraction',
    'spiral-arm-count-gz2_3_fraction',
    'spiral-arm-count-gz2_4_fraction',
    'spiral-arm-count-gz2_more-than-4_fraction',
    'spiral-arm-count-gz2_cant-tell_fraction',
]

questions = [
    'smooth-or-featured-gz2_smooth',
    'smooth-or-featured-gz2_featured-or-disk',
    'smooth-or-featured-gz2_artifact',
    'disk-edge-on-gz2_yes',
    'disk-edge-on-gz2_no',
    'bar-gz2_yes',
    'bar-gz2_no',
    'has-spiral-arms-gz2_yes',
    'has-spiral-arms-gz2_no',
    'bulge-size-gz2_no',
    'bulge-size-gz2_just-noticeable',
    'bulge-size-gz2_obvious',
    'bulge-size-gz2_dominant',
    'something-odd-gz2_yes',
    'something-odd-gz2_no',
    'how-rounded-gz2_round',
    'how-rounded-gz2_in-between',
    'how-rounded-gz2_cigar',
    'bulge-shape-gz2_round',
    'bulge-shape-gz2_boxy',
    'bulge-shape-gz2_no-bulge',
    'spiral-winding-gz2_tight',
    'spiral-winding-gz2_medium',
    'spiral-winding-gz2_loose',
    'spiral-arm-count-gz2_1',
    'spiral-arm-count-gz2_2',
    'spiral-arm-count-gz2_3',
    'spiral-arm-count-gz2_4',
    'spiral-arm-count-gz2_more-than-4',
    'spiral-arm-count-gz2_cant-tell',
```

```
]

# Find the image with the highest vote fraction for each feature and plot it

num_features = len(features)
num_cols = 4
num_rows = num_features // num_cols + (num_features % num_cols > 0)

average_votes = {}

for question in questions:
    avg_votes_per_question = df[question].mean()
    average_votes[question] = avg_votes_per_question

overall_average_votes = sum(average_votes.values()) / len(questions)

VOTESMIN = int(round(overall_average_votes) + 2 * np.std(list(average_votes.values())))
print(VOTESMIN)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

for i, (question, ax) in enumerate(zip(questions, axes.flatten())):
    # Get the corresponding key for the question
    key = keys[questions.index(question)]

    # Filter dataframe to exclude rows with fewer votes than VOTESMIN for the current question
    df_filtered = df[df[question] > VOTESMIN]

    if not df_filtered.empty:
        # Find the row with the highest vote fraction for the current question
        max_vote_fraction_row = df_filtered[df_filtered[key] == df_filtered[key].max()]

        image_filename = max_vote_fraction_row['filename'].values[0]
        image_root = 'images/'
        image_subfolder = max_vote_fraction_row['subfolder'].values[0] + r'/'
        image_path = os.path.join(dataset_root, image_root, image_subfolder, image_filename)
        question_label = max_vote_fraction_row[question].values[0]

        img = plt.imread(image_path)
        ax.imshow(img)
        ax.set_title(f'{question} \n{os.path.join(image_subfolder, image_filename)} score: {(df[key].max()):.2f}
        ax.axis('off')
    else:
        # If no images meet the minimum vote requirement, display a placeholder
        ax.text(0.5, 0.5, 'No images meet\nminimum vote count', horizontalalignment='center', verticalalignment=
        ax.axis('off')

# Hide empty subplots
for j in range(i + 1, num_rows * num_cols):
    axes.flatten()[j].axis('off')

plt.tight_layout()
plt.show()
```

21

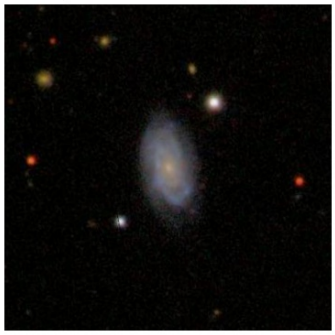




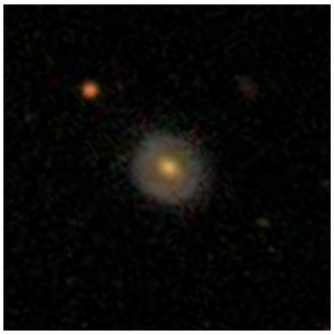

has-spiral-arms-gz2_no
587742/587742062680014916.jpg score: 1.00

bulge-size-gz2_no
587738/587738946129887405.jpg score: 1.00

bulge-size-gz2_just-noticeable
587738/587738950957072565.jpg score: 1.00



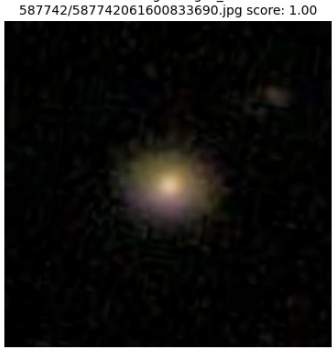
bulge-size-gz2_obvious
587735/587735744227967169.jpg score: 1.00



bulge-size-gz2_dominant
587736/587736807769112603.jpg score: 1.00

something-odd-gz2_yes
587729/587729158982860843.jpg score: 1.00

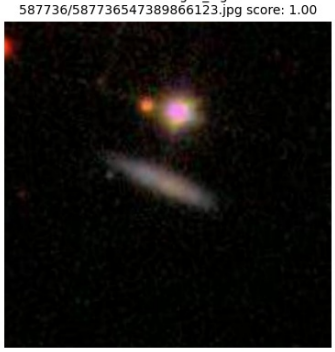
something-odd-gz2_no
587742/587742061600833690.jpg score: 1.00



how-rounded-gz2_round
587739/587739608623218816.jpg score: 1.00

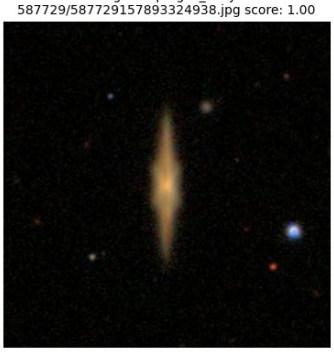
how-rounded-gz2_in-between
587729/587729782276686052.jpg score: 1.00

how-rounded-gz2_cigar
587736/587736547389866123.jpg score: 1.00

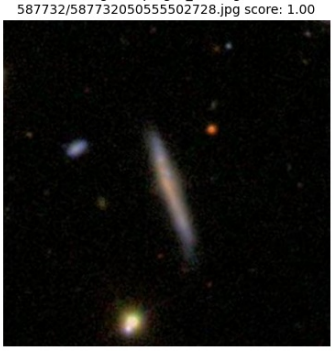


bulge-shape-gz2_round
587733/587733081885507750.jpg score: 1.00

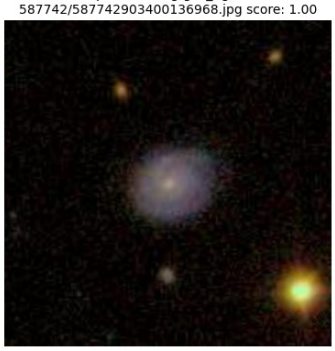
bulge-shape-gz2_boxy
587729/587729157893324938.jpg score: 1.00



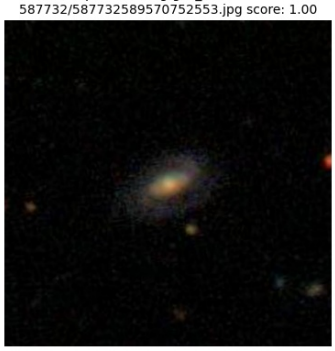
bulge-shape-gz2_no-bulge
587732/587732050555502728.jpg score: 1.00



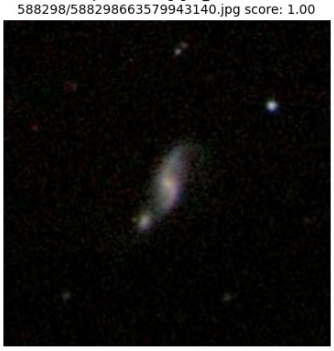
spiral-winding-gz2_tight
587742/587742903400136968.jpg score: 1.00



spiral-winding-gz2_medium
587732/587732589570752553.jpg score: 1.00



spiral-winding-gz2_loose
588298/588298663579943140.jpg score: 1.00

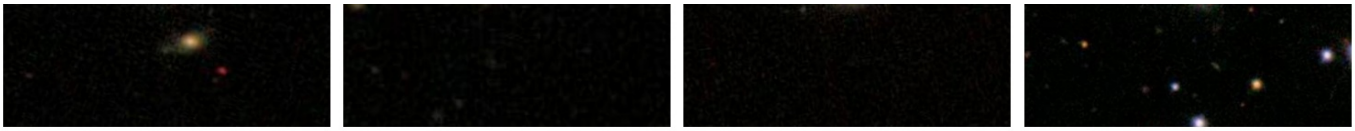


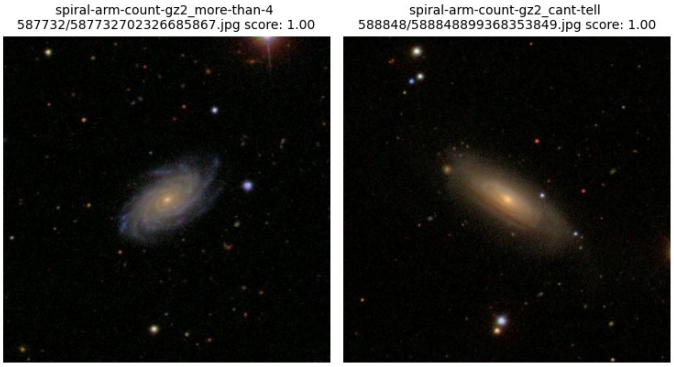
spiral-arm-count-gz2_1
588013/588013382200197365.jpg score: 1.00

spiral-arm-count-gz2_2
587739/587739302607388799.jpg score: 1.00

spiral-arm-count-gz2_3
587739/587739303148257430.jpg score: 1.00

spiral-arm-count-gz2_4
587729/587729653428191602.jpg score: 1.00





These are the galaxy images with the highest vote confidence for a specific feature, they appear to match the labels very well confirming the human labeling must be somewhat accurate. The dataset was filtered such that galaxies with fewer than (the average number of votes plus 2 standard deviations) 21 votes for a given feature were excluded to prevent galaxies with comically small total votes, and therefore inaccurate classifications from dominating the dataset.

NOTE: the artifact galaxy with the highest vote confidence appears to be an image of an SQL error message (very weird???), I have looked at the dataset images on my disk and the error message pictures are there as a .jpg file, so its not a bug with this code its a bug from how the dataset is downloaded/created when downloaded. This has been confirmed as the same galaxy images are these error messages when I redownload the dataset and download it on a new machine.

So I will have to either manually remove these images or try make a script to remove them from the datastet (easiest way would be to check if the bottom half of the image is pure black and then remove it if that's the case).

NOTE: there's also missing images in the dataset also, so will need to error handel those and remove them from the dataset also.

(quality of the PiPy galaxy-datasets gz2 is very questionable)

Malachy - 29/04/2024 -

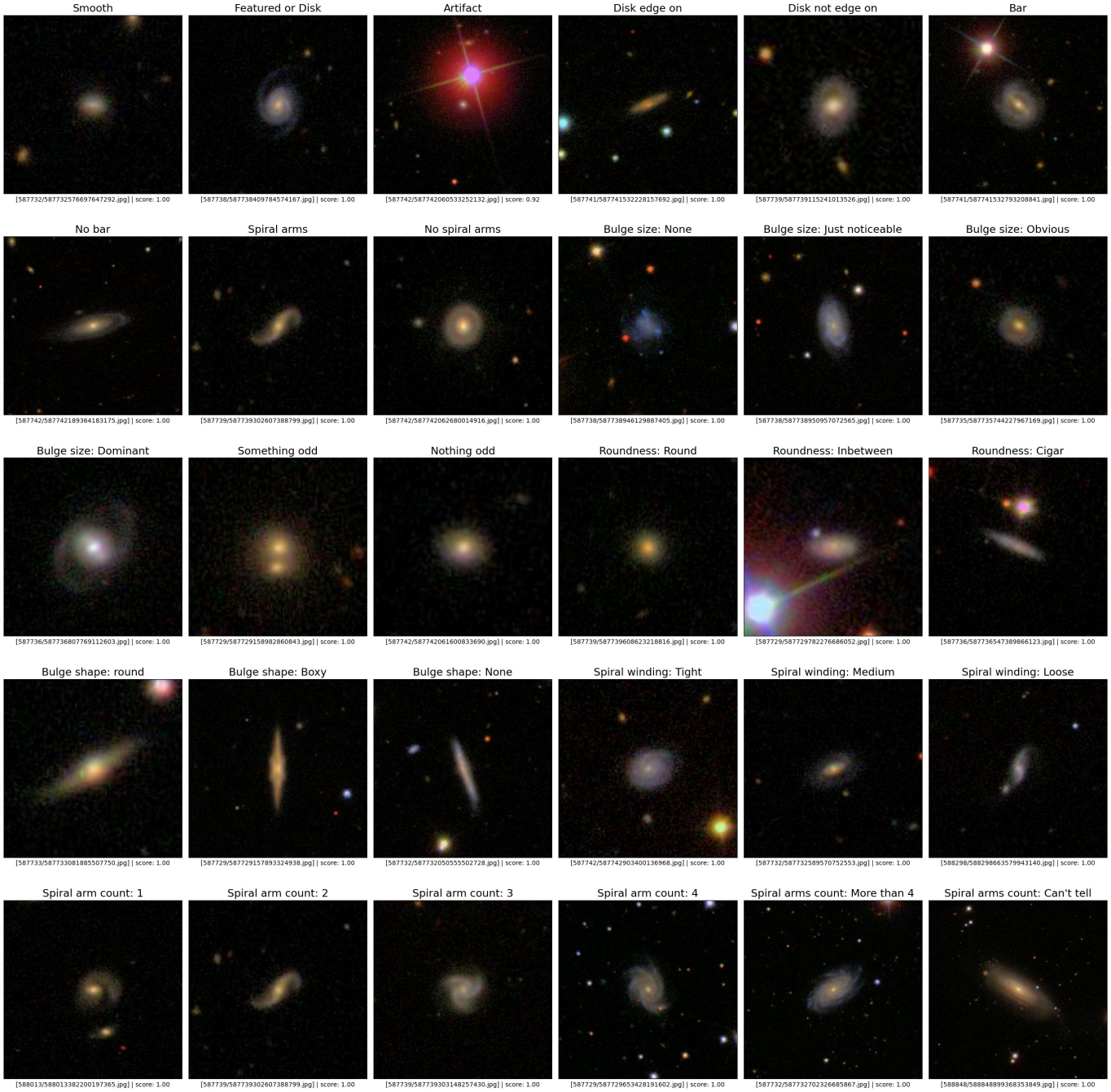
In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

"""
The Galaxies with the higest vote confidence for a given feature.
The dataset was filtered such that galaxies with fewer than (the average number of votes plus 2 standard deviati
for a given feature were excluded to prevent galaxies with comically small total votes, and therefore inaccurat
from dominating the dataset.
"""

dataset_root = 'galaxyzoo2-dataset-augmented/'
catalog_name = r'gz2_train_catalog.parquet'

parquet_file_path = os.path.join(dataset_root, catalog_name)

def build_file_path(row):
    file_loc = os.path.join(dataset_root, 'images/', str(row['subfolder']), str(row['filename']))
    return file_loc if os.path.exists(file_loc) else None

df = pd.read_parquet(parquet_file_path)
df['file_loc'] = df.apply(build_file_path, axis = 1)

df = df.dropna(subset=['file_loc'])
features = {
    'smooth-or-featured-gz2_smooth_fraction': [],
    'smooth-or-featured-gz2_featured-or-disk_fraction': [],
    'smooth-or-featured-gz2_artifact_fraction': [],
    'disk-edge-on-gz2_yes_fraction': [],
    'disk-edge-on-gz2_no_fraction': [],
    'bar-gz2_yes_fraction': [],
    'bar-gz2_no_fraction': [],
    'has-spiral-arms-gz2_yes_fraction': [],
    'has-spiral-arms-gz2_no_fraction': [],
```

```
    'bulge-size-gz2_no_fraction': [],
    'bulge-size-gz2_just-noticeable_fraction': [],
    'bulge-size-gz2_obvious_fraction': [],
    'bulge-size-gz2_dominant_fraction': [],
    'something-odd-gz2_yes_fraction': [],
    'something-odd-gz2_no_fraction': [],
    'how-rounded-gz2_round_fraction': [],
    'how-rounded-gz2_in-between_fraction': [],
    'how-rounded-gz2_cigar_fraction': [],
    'bulge-shape-gz2_round_fraction': [],
    'bulge-shape-gz2_boxy_fraction': [],
    'bulge-shape-gz2_no-bulge_fraction': [],
    'spiral-winding-gz2_tight_fraction': [],
    'spiral-winding-gz2_medium_fraction': [],
    'spiral-winding-gz2_loose_fraction': [],
    'spiral-arm-count-gz2_1_fraction': [],
    'spiral-arm-count-gz2_2_fraction': [],
    'spiral-arm-count-gz2_3_fraction': [],
    'spiral-arm-count-gz2_4_fraction': [],
    'spiral-arm-count-gz2_more-than-4_fraction': [],
    'spiral-arm-count-gz2_cant-tell_fraction': [],
}

keys = [
    'smooth-or-featured-gz2_smooth_fraction',
    'smooth-or-featured-gz2_featured-or-disk_fraction',
    'smooth-or-featured-gz2_artifact_fraction',
    'disk-edge-on-gz2_yes_fraction',
    'disk-edge-on-gz2_no_fraction',
    'bar-gz2_yes_fraction',
    'bar-gz2_no_fraction',
    'has-spiral-arms-gz2_yes_fraction',
    'has-spiral-arms-gz2_no_fraction',
    'bulge-size-gz2_no_fraction',
    'bulge-size-gz2_just-noticeable_fraction',
    'bulge-size-gz2_obvious_fraction',
    'bulge-size-gz2_dominant_fraction',
    'something-odd-gz2_yes_fraction',
    'something-odd-gz2_no_fraction',
    'how-rounded-gz2_round_fraction',
    'how-rounded-gz2_in-between_fraction',
    'how-rounded-gz2_cigar_fraction',
    'bulge-shape-gz2_round_fraction',
    'bulge-shape-gz2_boxy_fraction',
    'bulge-shape-gz2_no-bulge_fraction',
    'spiral-winding-gz2_tight_fraction',
    'spiral-winding-gz2_medium_fraction',
    'spiral-winding-gz2_loose_fraction',
    'spiral-arm-count-gz2_1_fraction',
    'spiral-arm-count-gz2_2_fraction',
    'spiral-arm-count-gz2_3_fraction',
    'spiral-arm-count-gz2_4_fraction',
    'spiral-arm-count-gz2_more-than-4_fraction',
    'spiral-arm-count-gz2_cant-tell_fraction',
]

questions = [
    'smooth-or-featured-gz2_smooth',
    'smooth-or-featured-gz2_featured-or-disk',
    'smooth-or-featured-gz2_artifact',
    'disk-edge-on-gz2_yes',
    'disk-edge-on-gz2_no',
    'bar-gz2_yes',
    'bar-gz2_no',
    'has-spiral-arms-gz2_yes',
    'has-spiral-arms-gz2_no',
    'bulge-size-gz2_no',
    'bulge-size-gz2_just-noticeable',
    'bulge-size-gz2_obvious',
    'bulge-size-gz2_dominant',
    'something-odd-gz2_yes',
    'something-odd-gz2_no',
    'how-rounded-gz2_round',
    'how-rounded-gz2_in-between',
    'how-rounded-gz2_cigar',
    'bulge-shape-gz2_round',
    'bulge-shape-gz2_boxy',
    'bulge-shape-gz2_no-bulge',
    'spiral-winding-gz2_tight',
    'spiral-winding-gz2_medium',
    'spiral-winding-gz2_loose',
    'spiral-arm-count-gz2_1',
    'spiral-arm-count-gz2_2',
```

```
    'spiral-arm-count-gz2_3',
    'spiral-arm-count-gz2_4',
    'spiral-arm-count-gz2_more-than-4',
    'spiral-arm-count-gz2_cant-tell',
]

question_to_name = {
    'smooth-or-featured-gz2_smooth': 'Smooth',
    'smooth-or-featured-gz2_featured-or-disk': 'Featured or Disk',
    'smooth-or-featured-gz2_artifact': 'Artifact',
    'disk-edge-on-gz2_yes': 'Disk edge on',
    'disk-edge-on-gz2_no': 'Disk not edge on',
    'bar-gz2_yes': 'Bar',
    'bar-gz2_no': 'No bar',
    'has-spiral-arms-gz2_yes': 'Spiral arms',
    'has-spiral-arms-gz2_no': 'No spiral arms',
    'bulge-size-gz2_no': 'Bulge size: None',
    'bulge-size-gz2_just-noticeable': 'Bulge size: Just noticeable',
    'bulge-size-gz2_obvious': 'Bulge size: Obvious',
    'bulge-size-gz2_dominant': 'Bulge size: Dominant',
    'something-odd-gz2_yes': 'Something odd',
    'something-odd-gz2_no': 'Nothing odd',
    'how-rounded-gz2_round': 'Roundness: Round',
    'how-rounded-gz2_in-between': 'Roundness: Inbetween',
    'how-rounded-gz2_cigar': 'Roundness: Cigar',
    'bulge-shape-gz2_round': 'Bulge shape: round',
    'bulge-shape-gz2_boxy': 'Bulge shape: Boxy',
    'bulge-shape-gz2_no-bulge': 'Bulge shape: None',
    'spiral-winding-gz2_tight': 'Spiral winding: Tight',
    'spiral-winding-gz2_medium': 'Spiral winding: Medium',
    'spiral-winding-gz2_loose': 'Spiral winding: Loose',
    'spiral-arm-count-gz2_1': 'Spiral arm count: 1',
    'spiral-arm-count-gz2_2': 'Spiral arm count: 2',
    'spiral-arm-count-gz2_3': 'Spiral arm count: 3',
    'spiral-arm-count-gz2_4': 'Spiral arm count: 4',
    'spiral-arm-count-gz2_more-than-4': 'Spiral arms count: More than 4',
    'spiral-arm-count-gz2_cant-tell': "Spiral arms count: Can't tell",
}

# Find the image with the highest vote fraction for each feature and plot it

num_features = len(features)
num_cols = 6
num_rows = num_features // num_cols + (num_features % num_cols > 0)

average_votes = {}

for question in questions:
    avg_votes_per_question = df[question].mean()
    average_votes[question] = avg_votes_per_question

overall_average_votes = sum(average_votes.values()) / len(questions)

VOTESMIN = int(round(overall_average_votes) + 2 * np.std(list(average_votes.values())))
print(VOTESMIN)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 5*num_rows))

for i, (question, ax) in enumerate(zip(questions, axes.flatten())):
    # Get the corresponding key for the question
    key = keys[questions.index(question)]

    # Filter dataframe to exclude rows with fewer votes than VOTESMIN for the current question
    df_filtered = df[df[question] > VOTESMIN]

    if not df_filtered.empty:
        # Find the row with the highest vote fraction for the current question
        max_vote_fraction_row = df_filtered[df_filtered[key] == df_filtered[key].max()]

        image_filename = max_vote_fraction_row['filename'].values[0]
        image_root = 'images/'
        image_subfolder = max_vote_fraction_row['subfolder'].values[0] + r'/'
        image_path = os.path.join(dataset_root, image_root, image_subfolder, image_filename)
        question_label = max_vote_fraction_row[question].values[0]

        name = question_to_name[question]

        img = plt.imread(image_path)
        ax.imshow(img)
        ax.set_title(f'{name}', fontsize = 17)
        ax.set_xlabel(f'[{os.path.join(image_subfolder, image_filename)}] | score: {(df[key].max()):.2f}', fonts
        ax.set_xticks([])
        ax.set_yticks([])
```

```
    else:
        # If no images meet the minimum vote requirement, display a placeholder
        ax.text(0.5, 0.5, 'No images meet\nminimum vote count', horizontalalignment='center', verticalalignment
        ax.axis('off')

# Hide empty subplots
for j in range(i + 1, num_rows * num_cols):
    axes.flatten()[j].axis('off')

plt.tight_layout()
plt.show()
```

21



corrected plots of the most the galaxy confidently answered feature label

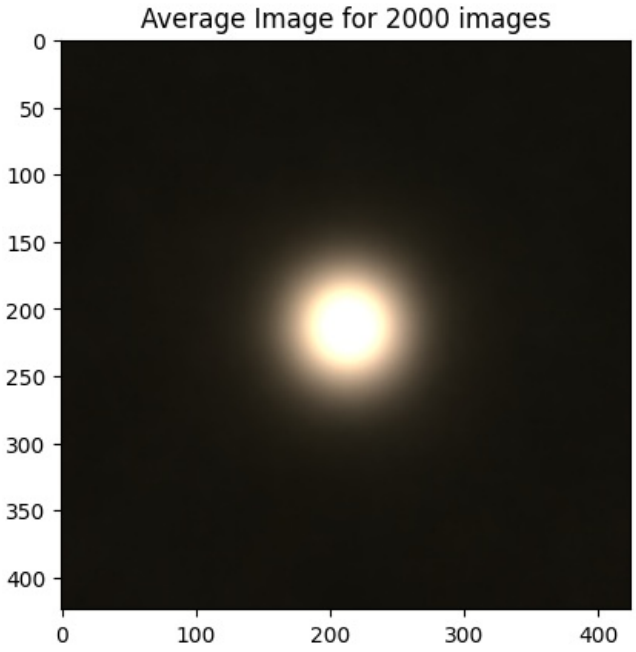
Malachy - determining the optimal cropping bounding box size

wrote a simple script that finds the average colour of 2000 galaxy images, it then shrinks a bounding box centrally by minimising the loss (defined as the distance of the bounding box corners from the brightest pixels) until it reaches pixels above a threshold brightness (then the loss starts to increase).

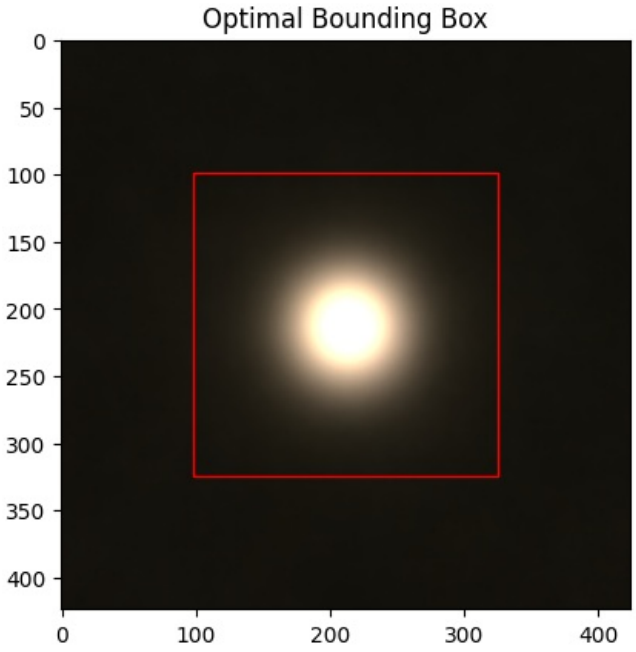
In [ ]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import time as time
import os

dataset_root = 'galaxyzoo2-dataset/'
catalog_name = r'gz2_train_catalog.parquet'
```

```
parquet_file_path = os.path.join(dataset_root, catalog_name)

df = pd.read_parquet(parquet_file_path)

average_image = np.zeros((424, 424, 3))
num_images = 2000

for idx, _ in enumerate(df['id_str']):

    if idx == num_images:
        break
    else:
        id = df['id_str'][idx]
        image_filename = df['filename'][idx]
        image_root = 'images/'
        image_subfolder = df['subfolder'][idx] + r'/'

        image_path = os.path.join(dataset_root, image_root, image_subfolder, image_filename)
        image = plt.imread(image_path)

        average_image += image

average_image /= len(df)
average_image = np.clip(average_image, 0, 1) # Ensure the average image is in the range [0, 1] for matplotlib


plt.imshow(average_image)
plt.title(f'Average Image for {num_images} images')
plt.show()

def loss_function(image, bbox, threshold):
    """
    Loss function to minimize the distance of the corners from the brightest pixels.
    """
    x1, y1, x2, y2 = bbox
    brightest_pixels = np.where(image >= threshold)

    for x, y in zip(brightest_pixels[1], brightest_pixels[0]):
        loss = np.sqrt((x - x1)**2 + (y - y1)**2) + np.sqrt((x - x2)**2 + (y - y2)**2)

    return loss

#use the brightness loss function to find the bounding box for the average image
def find_bounding_box(image, max_iterations, threshold, step):
    """
    Find the bounding box for the image such that the brightness loss
    is less than max_loss.
    """
    iteration = 0
    bbox = [0, 0, image.shape[1] - 1, image.shape[0] - 1]
    loss = loss_function(image, bbox, threshold)
    prev_loss = loss_function(image, bbox, threshold)

    while iteration < max_iterations:

        # If the bounding box corners are on a pixel with a brightness greater than or equal the threshold incr
        if np.any(image[bbox[1], bbox[0]] >= threshold) or np.any(image[bbox[1], bbox[2]] >= threshold) or np.a
            bbox[0] = max(0, int(bbox[0] - step))
            bbox[1] = max(0, int(bbox[1] - step))
            bbox[2] = min(image.shape[1] - 1, bbox[2] + step)
            bbox[3] = min(image.shape[0] - 1, bbox[3] + step)

            # Ensure bounding box coordinates stay within image bounds
            bbox[2] = max(bbox[0], bbox[2])
            bbox[3] = max(bbox[1], bbox[3])

            loss = loss_function(image, bbox, threshold)
            prev_loss = loss

        # decreaase the bounding box size to minimize the loss, the distance from the brightest pixels
        else:
            bbox[0] = max(0, int(bbox[0] + step))
            bbox[1] = max(0, int(bbox[1] + step))
            bbox[2] = min(image.shape[1] - 1, bbox[2] - step)
            bbox[3] = min(image.shape[0] - 1, bbox[3] - step)

            bbox[2] = max(bbox[0], bbox[2])
            bbox[3] = max(bbox[1], bbox[3])

            loss = loss_function(image, bbox, threshold)
            prev_loss = loss
```

```
        iteration += 1
        print(f'iter: {iteration} loss: {prev_loss:.2f} Bounding Box Corners: ({bbox[0]}, {bbox[1]}), ({bbox[2]

    return bbox

bbox = find_bounding_box(average_image, max_iterations=200, threshold=0.085, step=1)

fig, ax = plt.subplots()

ax.imshow(average_image)
rect = plt.Rectangle((bbox[0], bbox[1]), bbox[2] - bbox[0], bbox[3] - bbox[1], linewidth=1, edgecolor='r', face
ax.add_patch(rect)
plt.title('Optimal Bounding Box')
plt.show()
```




```
iter: 200 loss: 515.01 Bounding Box Corners: (98, 98), (325, 325) Side Length: 227
```



-> will probably centrally crop galazy images to 256x256 (227 is a weird number to work with, powers of 2 are nice)

9/03/2024 - Aroushi -

In [ ]:

```
"""

09/03/2024 - Aroushi: CNN Code using method similar to TensorFlow example

-Developed function to import data from parquet files
-Developed function to process image files
-Developed function to convert class label data into one-hot format
-Divided data into test, train and validation sets
-Defined basic CNN Model Architecture

"""
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from PIL import Image


#Function to load the class label data from parquet files, and turning it into one-hot format
def load_class_data(file_path):
    df = pd.read_parquet(file_path)
    class_labels = df['label'] + 1
    class_labels_one_hot = to_categorical(class_labels, num_classes=8) #turns into one-hot
    return pd.DataFrame({'label': class_labels,
                         **{f'class_{i}': class_labels_one_hot[:, i] for i in range(class_labels_one_hot.shape[

#To process the image
def load_image(full_image_path):
    image = tf.io.read_file(full_image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [128, 128])  # Resize as necessary
    image = image / 255.0  # Normalize pixel values if needed
    return image  # Make sure to return the processed image tensor

#Loads all image tensors into a list
def load_images(catalog, image_path):
    images = []
    for relative_path in catalog['file_loc']:
        relative_path = relative_path[-29:]
        full_image_path = os.path.join(image_path, str(relative_path))
        image = load_image(full_image_path)  # This should now return a TensorFlow tensor, not None
        if image is not None:  # Optionally check if image is not None before appending
            images.append(image)
    return images

# Displays individual images for observation
def display_image_from_catalog(catalog, image_path, index):
    full_image_path = os.path.join(image_path, catalog['file_loc'].iloc[index])
    im = Image.open(full_image_path)
    im.show()

#Define paths to relevant files (needs to be changed based on user)
#Functions take care of choosing specific files beyond these paths
file_path = r'/Users/aroushijimulia/Documents/GitHub/Project-72-Classifying-cosmological-data-with-machine-lear
image_path = r'/Users/aroushijimulia/Documents/GitHub/Project-72-Classifying-cosmological-data-with-machine-lea

#For Windows
#file_path = r''
#image_path = r''

#Loads class labels into class_data
class_data = load_class_data(file_path)

#Loads all parquet data into catalog
catalog = pd.read_parquet(file_path)

#Add one hot encoded labels to catalog
#class_data = class_data.merge(catalog[['label', 'file_loc']], on='label', how='left')
#merge_data = class_data.merge(catalog[['label', 'file_loc']], on='label', how='left')

#%%
# Split the data
train_df, test_df = train_test_split(class_data, test_size=0.2, random_state=42)
train_df, val_df = train_test_split(train_df, test_size=0.25, random_state=42)  # 0.25 x 0.8 = 0.2

# Create TensorFlow datasets
train_ds = tf.data.Dataset.from_tensor_slices((train_df['image'], train_df['class_label_one_hot']))
val_ds = tf.data.Dataset.from_tensor_slices((val_df['image'], val_df['class_label_one_hot']))
test_ds = tf.data.Dataset.from_tensor_slices((test_df['image'], test_df['class_label_one_hot']))

# Map the datasets to load the images
train_ds = train_ds.map(lambda x, y: (load_image(x), y))
val_ds = val_ds.map(lambda x, y: (load_image(x), y))
test_ds = test_ds.map(lambda x, y: (load_image(x), y))

# Batch and shuffle the datasets
train_ds = train_ds.shuffle(1000).batch(32)
val_ds = val_ds.batch(32)
test_ds = test_ds.batch(32)
```

```
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(class_data['class_label'].unique()), activation='softmax')  # Number of classes
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

10/03/2024 Malachy -

In [ ]:

```
"""
10/03/2024 - Malachy: CCN code so far.

Created a class for the CNN and a class for the dataframe.
The DataFrame class is used to create a dataframe of the training and test data in the required format, i.e one
The ConvolutionalNeuralNetwork class is used to create a Convolutional Neural Network of a specified architectu
"""
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras as keras
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import MaxPooling3D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv3D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

class ConvolutionalNeuralNetwork():
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, train_class_labels, test_class_
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        """

        self.display_summary = display_summary
        self.architecture = architecture
        self.learning_rate = learning_rate
        self.weight_reg = weight_reg
        self.network = self.__init_network(display_summary, architecture)

        self.representation = representation
        self.train_class_labels = train_class_labels
        self.test_class_labels = test_class_labels
        self.input_dim = input_dim

    def __init_network(self, display_summary: bool = False, architecture: str = "216x64") -> tf.keras.Model:
        lr = self.learning_rate

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)

        if architecture == '128x64xn':

            network = Sequential()
            network.add(Conv2D(filters = 32, kernal_size = (7, 7), strides = 3, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernal_size = (3,3), strides = 1, activation = 'relu', kernel_regu
```

```
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernal_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(Flatten())
            network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
            network.add(Dropout(0.5))
            network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
            network.add(Dropout(0.5))
            network.add(Dense(len(self.class_labels), activation = None, kernel_regularizer = wr))

            network.compile(loss = 'mean_squared_error', optimizer = tf.keras.optimizers.Adam(learning_rate = l

        elif architecture == '400xn':

            pass

        else:
            raise ValueError(f"Invalid architecture parameter provided: {architecture}")

        if (display_summary):
            network.summary()

        return network

    def batch_train():
        pass

    def test():
        pass

class DataFrame():
    def __init__(self, dataset_root, train_catalog, test_catalog, keys, representation, input_dim = 3, seed = 0
        """
        Construct a DataFrame containing the represenation of the training and test data in the required format

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
        representation (str): the representation of the data, i.e one-hot or vote fraction
        """

        self.dataset_root = dataset_root

        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog)
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)

        self.representation = representation
        self.input_dim = input_dim
        self.keys = keys

        self.train_dataset = self.__init_train_dataframe()
        self.test_dataset = self.__init_test_dataframe()

        self.RNG = np.random.default_rng(seed)

    def __init_train_dataframe(self):
        if representation == 'one_hot':

            pandas_df = pd.read_parquet(self.train_parquet_file_path)
            pandas_df = pandas_df.fillna(0)
            class_labels = pandas_df['label'] + 1
            one_hot = pd.get_dummies(class_labels).astype(int) # convert the class labels to one-hot encoding

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str) # delete the information co

            for idx, _ in enumerate(pandas_df['id_str']):  # construct the actual path to the image file and ad
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.concatenate((np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array(one_h

        elif representation == 'vote_fraction':

            pandas_df = pd.read_parquet(self.train_parquet_file_path)
            pandas_df = pandas_df.fillna(0)

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.concatenate((np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array([panda
        else:
```

```
            raise ValueError(f"Training DataFrame: Invalid representation parameter provided: {representation}"

    def __init_test_dataframe(self):
        if representation == 'one_hot':

            pandas_df = pd.read_parquet(self.test_parquet_file_path)
            pandas_df = pandas_df.fillna(0)
            class_labels = pandas_df['label'] + 1
            one_hot = pd.get_dummies(class_labels).astype(int)

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return  np.concatenate((np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array(one_

        elif representation == 'vote_fraction':

            pandas_df = pd.read_parquet(self.test_parquet_file_path)
            pandas_df = pandas_df.fillna(0)
            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return  np.concatenate((np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array([pan
        else:
            raise ValueError(f"Test DataFrame: Invalid representation parameter provided: {representation}")

    def load_image(self, image_path, input_dim):

        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels = input_dim)
        image = tf.image.convert_image_dtype(image, tf.float32) /255  # Convert image dtype from tf.uint8 to tf

        return image

    def shuffle_datset(self, data):
        return self.RNG.choice(data, size=(len(data)), replace=False, axis=0, shuffle=True) # randomly shuffle

    def get_batch(self, data, batch_size, batch):

        num_batches = len(data) // batch_size + (len(data) % batch_size != 0)

        # slice the data array into num_batches batches of size batch_size
        if len(data) % batch_size == 0 :
            batched_ds = data[batch * batch_size : (batch + 1) * batch_size]
        else:
            if batch == num_batches - 1: # case when the last batch is smaller than batch_size
                batched_ds = data[batch * batch_size : len(data)]
            else:
                batched_ds = data[batch * batch_size : (batch + 1) * batch_size]

        batched_labels = batched_ds[:, 1:] # extract the labels from the batch

        image_paths = batched_ds[:, 0] # extract the first column of the batch, the image paths
        batched_images = np.array([self.load_image(image_path, self.input_dim) for image_path in image_paths])

        return batched_images, batched_labels

dataset_root = 'Project-72-Classifying-cosmological-data-with-machine-learning/galaxyzoo2-dataset/'
train_catalog = r'gz2_train_catalog.parquet'
test_catalog = r'gz2_test_catalog.parquet'

representation = 'one_hot'

if representation == 'one_hot':
    keys = [
    0,
    1,
    2,
    4,
    5,
    6,
    7
]
elif representation == 'vote_fraction':
    keys = [
        'smooth-or-featured-gz2_smooth_fraction',
        'smooth-or-featured-gz2_featured-or-disk_fraction',
        'smooth-or-featured-gz2_artifact_fraction',
        'disk-edge-on-gz2_yes_fraction',
```

```
        'disk-edge-on-gz2_no_fraction',
        'bar-gz2_yes_fraction',
        'bar-gz2_no_fraction',
        'has-spiral-arms-gz2_yes_fraction',
        'has-spiral-arms-gz2_no_fraction',
        'bulge-size-gz2_no_fraction',
        'bulge-size-gz2_just-noticeable_fraction',
        'bulge-size-gz2_obvious_fraction',
        'bulge-size-gz2_dominant_fraction',
        'something-odd-gz2_yes_fraction',
        'something-odd-gz2_no_fraction',
        'how-rounded-gz2_round_fraction',
        'how-rounded-gz2_in-between_fraction',
        'how-rounded-gz2_cigar_fraction',
        'bulge-shape-gz2_round_fraction',
        'bulge-shape-gz2_boxy_fraction',
        'bulge-shape-gz2_no-bulge_fraction',
        'spiral-winding-gz2_tight_fraction',
        'spiral-winding-gz2_medium_fraction',
        'spiral-winding-gz2_loose_fraction',
        'spiral-arm-count-gz2_1_fraction',
        'spiral-arm-count-gz2_2_fraction',
        'spiral-arm-count-gz2_3_fraction',
        'spiral-arm-count-gz2_4_fraction',
        'spiral-arm-count-gz2_more-than-4_fraction',
        'spiral-arm-count-gz2_cant-tell_fraction',
    ]
else:
    raise ValueError(f"Invalid representation parameter provided: {representation}")

df = DataFrame(dataset_root, train_catalog, test_catalog, keys, representation, input_dim = 3, seed = 0)
print(f" train ds shape: {df.train_dataset.shape}")
print(f" test ds shape: {df.test_dataset.shape}")

rand_indexes = df.get_train_data_batch(256)
print(f"rand idx shape: {rand_indexes.shape}")
print(rand_indexes)

# NOTE:
# At the start of each epoch call the shuffle_dataset method to shuffle the training dataset
# then call the get_batch method to get the next batch of images and labels, and then train the network on the
# increment the batch counter and repeat the process until the end of the epoch.

# TODO: will need to implement this in the batch_train method of the ConvolutionalNeuralNetwork class
# and possibly look into custom training loops in tensorflow
```

Aroushi -

Week Starting February 26th:

Thursday Febuary 29th 10am-11am: SUPERVISOR MEETING

- Reviewed first attempts at CNNs and discussed next best steps
- Discussed network architectures and previous projects
- Debated structure of decision tree and how the data must be formatted

16/03/2024 Aroushi -

In [ ]:
```
"""

16/03/24 - Aroushi: Researching CNN architectures used in literature


"""

### Architecture 1 ###

#Source: https://github.com/nitarshan/densenet-galaxy-zoo/blob/master/DenseNet%20Galaxy%20Zoo.ipynb

"""

DenseLayer:

Always starts with batch normalization and an activation function.
If a bottleneck is used, it includes a 1x1x1 convolution for dimensionality reduction, followed by batch normal
The main part of the dense layer is a 3D convolution with k filters and a kernel size of 1x3x3, maintaining spa
If dropout is specified, it applies dropout for regularization.
Output: Returns the concatenated tensor of the input and the processed path, increasing the depth (channel dime

"""
def DenseLayer(inputs, k, bn=None, drop=None):
```

```
    x = BatchNormalization()(inputs)
    if bn: # Bottleneck
        x = Activation('relu')(x)
        x = Conv3D(k*bn, (1,1,1))(x)
        x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv3D(k, (1,3,3), padding='same')(x)
    if drop:
        x = Dropout(drop)(x)
    return concatenate([inputs, x], axis=4)


"""

DenseBlock:

Process: Repeatedly applies the DenseLayer function L times on the output of the previous layer.
Output: The output tensor after L dense layers.

"""
def DenseBlock(x, k, L, bn=None, drop=None):
    for l in range(L):
        x = DenseLayer(x, k, bn, drop)
    return x

"""

TransitionLayer:
Process: Includes batch normalization, activation, a 1x1x1 convolution to adjust filter numbers, and an average
Output: The output tensor after transition processing.

"""
def TransitionLayer(x, k):
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv3D(k, (1,1,1))(x)
    x = AvgPool3D((1,2,2), strides=(1,2,2), padding='same')(x)
    return x

"""

DenseNet:
Process: Starts with an initial convolution, followed by batch normalization and pooling.
Then, iterates over B dense blocks, applying a transition layer between each (except after the last block).
Ends with batch normalization, a convolution to adjust to the desired output size, global average pooling, and a
Output: A Keras Model instance with the specified architecture.


"""
def DenseNet(shape, k, bn, theta, drop, B, L, outs):
    img_input = Input(shape=(4,shape,shape,3))
    x = Conv3D(k*2, (1,7,7), strides=(1,2,2), padding='same')(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPool3D((1,3,3), strides=(1,2,2), padding='same')(x)
    for i in range(B):
        x = DenseBlock(x, k, L, bn, drop)
        if i != B-1:
            k = int(k*theta) # compression
            x = TransitionLayer(x, k)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv3D(37, (1,7,7))(x)
    x = GlobalAveragePooling3D()(x)
    x = Activation('sigmoid')(x)
    return Model(inputs=img_input, outputs=x)


### Archiecture 2 ###

#Source: https://link.springer.com/chapter/10.1007/978-3-031-23092-9_1

"""

The paper describes the model in this way:
The model that we are proposing is a ConvNet galaxy architecture consists of one input layer having 16 filters,
followed by 4 hidden layers, 1 penultimate dense layer, along with an Output Softmax layer.
We also included data augmentation such as shear, zoom, rotation, rescaling, and flip.

"""
def build_galaxy_convnet(input_shape=(128, 128, 3)):  # Example input shape, adjust as necessary
    model = models.Sequential([
        # Input layer with data augmentation
```

```
        layers.experimental.preprocessing.Rescaling(1./255, input_shape=input_shape),
        layers.Conv2D(16, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        # Hidden layers
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # Penultimate Dense layer
        layers.Flatten(),
        layers.Dense(64, activation='relu'),

        # Output Softmax layer
        layers.Dense(10, activation='softmax')  # Adjust the number of units to match the number of classes
    ])

    return model

model = build_galaxy_convnet()
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


### Archiecture 3 ###

#AlexNet

def AlexNet(input_shape=(224, 224, 3), num_classes=10):  # Adjust num_classes based on your dataset
    model = Sequential([
        Conv2D(96, (11, 11), strides=4, activation='relu', input_shape=input_shape, padding='same'),
        BatchNormalization(),
        MaxPooling2D(3, strides=2),
        Conv2D(256, (5, 5), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(3, strides=2),
        Conv2D(384, (3, 3), activation='relu', padding='same'),
        Conv2D(384, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(3, strides=2),
        Flatten(),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

### Archiecture 4 ###

#VGGNet

from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense

def VGGNet(input_shape=(224, 224, 3), num_classes=10):  # Adjust num_classes based on your dataset
    # Load the VGG model with ImageNet weights, exclude top layers
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    # Freeze the layers of the base model
    for layer in base_model.layers:
        layer.trainable = False

    # Add custom layers on top for galaxy classification
    x = base_model.output
    x = Flatten()(x)
    x = Dense(4096, activation='relu')(x)
    x = Dense(4096, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    # This is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
### Further architectures can be explored if models stated above do not perform up to standard
```

Malachy, make notes on k fold validation here please:

16/03/2024 - Aroushi:

Defined above are options for model architectures that we will manually test to see how they perform on our dataset. A potential method to test models is k-fold cross validation

K-fold cross-validation is a statistical method used to evaluate the performance of a machine learning model on a limited data sample. It's particularly useful for assessing how well a model is likely to perform when making predictions on unseen data. The process involves dividing the data into "k" equal parts, or folds, then running "k" separate learning experiments. Here's a breakdown of how it works:

Splitting: The entire dataset is randomly divided into "k" equal-sized subsets (or folds).

Training and Validation: For each unique group:

Take one fold as the validation set for testing the model. Take the remaining k-1 folds as the training set. Train the model on the training set and evaluate it on the validation set. Retain the evaluation score and discard the model. Iteration: Repeat this process k times, each time with a different fold as the validation set and the remaining folds as the training set.

Aggregation: Once all k experiments are done, aggregate the evaluation scores from each experiment to come up with a single score that represents the model's overall performance.

Benefits of K-fold Cross-validation: Reduced Bias: Since each data point gets to be in a validation set exactly once and gets to be in a training set k-1 times, the variance and bias are reduced.

Efficient Use of Data: It allows every observation to be used for both training and validation, which is particularly useful when dealing with small datasets.

Generalizability: By using different subsets of the data for training and validation, you can assess how well your model generalizes to unseen data.

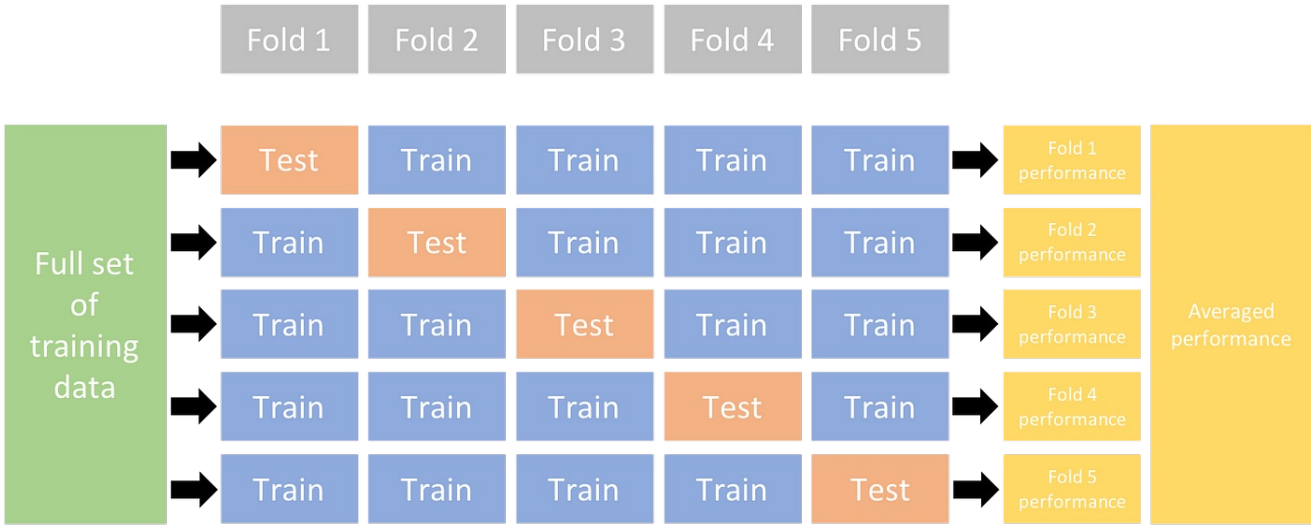
Considerations: Computational Cost: K-fold cross-validation can be computationally expensive as it requires training k models. This might not be feasible with very large datasets or very complex models.

Choosing K: A common choice for "k" is 5 or 10, but the optimal number depends on the size and specifics of the dataset. A larger "k" provides less bias toward overestimating the true expected error (as there's less variance between the training sets), but also increases the computational cost.

Randomness in Splitting: The way the data is split can affect the outcomes, so it's a good practice to perform the cross-validation multiple times with different random splits and average the results for more stable estimates.

K-fold cross-validation is a widely used technique in machine learning for model evaluation, especially when tuning parameters and comparing different models to select the best performing one.

https://machinelearningmastery.com/k-fold-cross-validation/



Aroushi -

Week Starting March 4th:

Thursday March 7th 10am-11am: SUPERVISOR MEETING

- Reviewed improvements made so far and research done and discussed next best steps
- Detailed review of code written
- We were urged to produce some preliminary results so we can assess the kind of accuracies we are getting so far

17/03/2024 Malachy -

In [ ]:
```
"""
Malachy -
17/03/2024

- Added changes to the ConvolutionalNeuralNetwork class and the DataFrame class
    - changed the batching method to use the tensorflow dataset API
    - implemented the training and testing methods for the CNN
    - implemented k-fold cross validation for the CNN
    - implemented validation test splitting of the data
    - implemented the ability to save/load the CNN model to/from a file
    - added command line arguments to the script to allow for the user to specify the architecture, learning ra

- Added additional comments here explaining the code and the changes made for Aroushi to understand.
"""

import pandas as pd
import argparse
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras as keras
import datetime as datetime
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import MaxPooling3D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv3D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

#============================================================== CLASSES AND FUNCTIONS ======================
class ConvolutionalNeuralNetwork(): # this is the ConvolutionalNeuralNetwork class, it contains the CNN and meth
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, train_class_labels, test_class_
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        """

        self.display_summary = display_summary # the "self." parameter refers to the specific instance of the c
        self.architecture = architecture       # i.e self.architecture = architecture gives the object created
        self.learning_rate = learning_rate     # Only the object created has this attribute, not the class itse
        self.weight_reg = weight_reg
        self.network = self.__init_network(display_summary, architecture) # This is a method call, it calls the
                                                                          # methods are functions that belong t
        self.representation = representation
        self.train_class_labels = train_class_labels
        self.test_class_labels = test_class_labels
        self.input_shape = input_shape
        self.input_dim = input_dim

    #------------------------------------------------------------------------ NETWORK INITIALIZATION ---------
    def __init_network(self, display_summary: bool = False, architecture: str = "216x64") -> tf.keras.Model: #
        lr = self.learning_rate                                                                               #

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else: # do weight regularization
            wr = tf.keras.regularizers.l2(self.weight_reg)

        if architecture == '128x64xn': # this if statement checks the architecture parameter and creates the CNN

            network = Sequential()
```

```
            network.add(Conv2D(filters = 32, kernel_size = (7, 7), strides = 3, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(Flatten())
            network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
            network.add(Dropout(0.5))
            network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
            network.add(Dropout(0.5))
            network.add(Dense(len(self.class_labels), activation = None, kernel_regularizer = wr))

            network.compile(loss = 'mean_squared_error', optimizer = tf.keras.optimizers.Adam(learning_rate = l

        elif architecture == '400xn':

            pass # pass is used as a placeholder, it is used when a statement is required syntactically but the

        else:
            raise ValueError(f"Invalid architecture parameter provided: {architecture}") # raise an error if th

        if (display_summary):
            network.summary()

        return network # methods can return values, in this case the method returns the network that was create
                       # this method is used to create the CNN, and is called in the class constructor, when th
    #------------------------------------------------------------------------ NETWORK TRAINING AND TESTING ---
    def train(self, images, labels): # this method is used to train the network on a batch of images and labels
        """
        Train the network on a batch of images and labels.

        args:
        images (np.array): the batch of training images
        labels (np.array): the batch of training labels
        """
        self.network.train_on_batch(images, labels) # trains the network on a single batch
        # see how this method doesn't return anything, it is used to perform an action, that being train the CN
        # calling this method with the correct arguments of images and labels will train the CNN on the batch o
        # this train method will be called in a loop to train the CNN on the entire training dataset, outside t

    def test(self, images, labels): # the self parameter in the method arguments is used to link this method to
        """
        Test the network on a batch of images and labels.

        arrgs:
        images (np.array): the batch of test images
        labels (np.array): the batch of test labels
        """
        test_loss, test_accuracy = self.network.evaluate(images, labels) # tests the network on a single batch

    #------------------------------------------------------------------ NETWORK WEIGHTS AND BIASES SAVING AND LO
    def save(self, checkpoint_path):
        """
        Save the network weights and biases to a file.

        args:
        checkpoint_path (str): the path to the directory to save the network weights and biases
        """
        timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S") # get the current date and time
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)  # create a file path to save the netwo
        self.network.save_weights(path) # save the network weights and biases to the file


    def load(self, checkpoint_path, timestamp):
        """
        Load the network weights and biases from a file.

        args:
        checkpoint_path (str): the path to the directory to load the network weights and biases
        """
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp) # create a file path to load the networ
        self.network.load_weights(path) # load the network weights and biases from the file

class DataFrame(): # this class is used to create a DataFrame object, which contains the training and test data
    def __init__(self, dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
        """
        Construct a DataFrame containing the represenation of the training and test data in the required format

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
```

```
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_shape (tuple): the input shape of the data, x, y
        input_dim (int): the input dimension of the data, i.e 3 for RGB image
        """

        self.dataset_root = dataset_root

        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog) # create the file path to
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)  # create the file path to

        self.representation = representation # representation of the data, i.e one-hot or vote fraction
        self.input_shape = input_shape
        self.input_dim = input_dim
        self.keys = self.__get_keys(representation) # this calls the __get_keys method to get the keys of the c

        self.train_dataset = self.__init_train_dataset(self.keys) # this cretes the training dataset
        self.test_dataset = self.__init_test_dataset(self.keys) # this creates the test dataset

        self.RNG = np.random.default_rng(seed)

    #------------------------------------------------------------------------ DATASET(S) INITIALIZATION ----------
    def __init_train_dataset(self, keys): # this method is identical functionally to the __init_test_dataset me
        """
        Initialise the training dataset, containing paths to the image files and the labels in the required rep

        returns:
        train_dataset (tf.data.Dataset): the training dataset in the required format, file path column then cla
        """
        if self.representation == 'one-hot':

            pandas_df = pd.read_parquet(self.train_parquet_file_path) # read the training catalog file into a Pa
            pandas_df = pandas_df.fillna(0) # remove all NaN values from the DataFrame
            class_labels = pandas_df['label'] + 1
            one_hot = pd.get_dummies(class_labels).astype(int) # convert the class labels to one-hot encoding

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str) # delete the information co

            for idx, _ in enumerate(pandas_df['id_str']):  # construct the actual path to the image file and ad
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array(one_hot) # return the

        elif self.representation == 'vote-fraction':

            pandas_df = pd.read_parquet(self.train_parquet_file_path) # read the training catalog file into a Pa
            pandas_df = pandas_df.fillna(0) # remove all NaN values from the DataFrame

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str) # delete the information co

            for idx, _ in enumerate(pandas_df['id_str']): # construct the actual path to the image file and add
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array([pandas_df[key] for k
        else:
            raise ValueError(f"Training DataFrame: Invalid representation parameter provided: {self.representat

    def __init_test_dataset(self, keys): # this is functionally identical to the __init_train_dataset method, i
        """
        Initialise the test dataset, containing paths to the image files and the labels in the required represe

        returns:
        test_dataset (tf.data.Dataset): the test dataset in the required format, file path column then class la
        """
        if self.representation == 'one-hot':

            pandas_df = pd.read_parquet(self.test_parquet_file_path)
            pandas_df = pandas_df.fillna(0)
            class_labels = pandas_df['label'] + 1
            one_hot = pd.get_dummies(class_labels).astype(int)

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return  np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array(one_hot) # return th

        elif self.representation == 'vote-fraction':

            pandas_df = pd.read_parquet(self.test_parquet_file_path)
            pandas_df = pandas_df.fillna(0)
            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)
```

```
            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return  np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array([pandas_df[key] for
        else:
            raise ValueError(f"Test DataFrame: Invalid representation parameter provided: {self.representation}

    def __get_keys(self, representation): # this method is used to obtain the keys of the class labels when acc
        """
        Obtain the keys of the class labels when accessing the class labels from a Pandas DataFrame.
        """
        if representation == 'one-hot': # one-hot class labels
            keys = [
            0,
            1,
            2,
            4,
            5,
            6,
            7
        ]
            return keys
        elif representation == 'vote-fraction': # vote fraction class labels
            keys = [
                'smooth-or-featured-gz2_smooth_fraction',
                'smooth-or-featured-gz2_featured-or-disk_fraction',
                'smooth-or-featured-gz2_artifact_fraction',
                'disk-edge-on-gz2_yes_fraction',
                'disk-edge-on-gz2_no_fraction',
                'bar-gz2_yes_fraction',
                'bar-gz2_no_fraction',
                'has-spiral-arms-gz2_yes_fraction',
                'has-spiral-arms-gz2_no_fraction',
                'bulge-size-gz2_no_fraction',
                'bulge-size-gz2_just-noticeable_fraction',
                'bulge-size-gz2_obvious_fraction',
                'bulge-size-gz2_dominant_fraction',
                'something-odd-gz2_yes_fraction',
                'something-odd-gz2_no_fraction',
                'how-rounded-gz2_round_fraction',
                'how-rounded-gz2_in-between_fraction',
                'how-rounded-gz2_cigar_fraction',
                'bulge-shape-gz2_round_fraction',
                'bulge-shape-gz2_boxy_fraction',
                'bulge-shape-gz2_no-bulge_fraction',
                'spiral-winding-gz2_tight_fraction',
                'spiral-winding-gz2_medium_fraction',
                'spiral-winding-gz2_loose_fraction',
                'spiral-arm-count-gz2_1_fraction',
                'spiral-arm-count-gz2_2_fraction',
                'spiral-arm-count-gz2_3_fraction',
                'spiral-arm-count-gz2_4_fraction',
                'spiral-arm-count-gz2_more-than-4_fraction',
                'spiral-arm-count-gz2_cant-tell_fraction',
            ]
            return keys # return the keys of the class labels
        else:
            raise ValueError(f"Invalid representation parameter provided: {representation}")
    #------------------------------------------------------------------------------- IMAGE PROCESSING -------
    def load_image(self, image_path, input_shape, input_dim): # load and image for the CNN to process
        """
        Load an image from a file path and convert it to a tf tensor.

        args:
        image_path (str): path to the image file
        input_dim (int): the input dimension of the data

        returns:
        image (tf.tensor): the image as a tf.tensor
        """

        image = tf.io.read_file(image_path) # read the image from the file path
        image = tf.image.decode_jpeg(image, channels = input_dim) # decode the image from a jpeg file to a tf.t
        image = tf.image.convert_image_dtype(image, tf.float32) /255  # Convert image dtype from tf.uint8 to tf
        image = tf.image.resize(image, input_shape) # resize the image to the input shape of the CNN

        return image # returns the image as a tf.tensor, this method is used to load an image from a file path a
    #TODO: implement image augmentation methods here, i.e rotation, flipping, scaling, translation, brightness,
    #------------------------------------------------------------------------------- VALIDATION DATA SPLITTING --
    def get_train_val_split(self, data, val_percent, shuffle, seed): # this method is used to split the training
        """
        Split the training dataset into a training and validation dataset.
```

```
        args:
        data (tuple): the training dataset to be split
        val_percent (float): the percentage of the training dataset to be used as the validation dataset range
        shuffle (bool): if to shuffle the data before splitting
        """
        images, labels = data

        if shuffle: # if to randomly shuffle the data before splitting
            images, labels = tf.random.shuffle(images, seed = seed), tf.random.shuffle(labels, seed = seed)

        val_size = int(len(images) * val_percent) # get the size of the validation dataset

        train_images, train_labels = images[val_size:], labels[val_size:] # split the training dataset into a t
        val_images, val_labels = images[:val_size], labels[:val_size] # split the training dataset into a train

        train_images, train_labels = tf.data.Dataset.from_tensor_slices(np.array(train_images)), tf.data.Dataset
        val_images, val_labels = tf.data.Dataset.from_tensor_slices(np.array(val_images)), tf.data.Dataset.from_

        train_split, validation_split = tf.data.Dataset.zip((train_images, train_labels)), tf.data.Dataset.zip(

        return train_split, validation_split # returns the training and validation datasets as a tuple

    def get_kfolds(self, data, k, split, RNGstate, shuffle): # this method is used to obtain a split of the data
        """
        Obtain a split of the dataset using k-fold cross validation, containing the k-1 training folds and 1 va

        args:
        data (tuple): the dataset to be split into k-folds
        k (int): the number of folds to be used
        split (int): the split to be returned
        RNGstate (int): random integer from self.RNG to set the random state of the k-fold cross validation
        shuffle (bool): if to shuffle the data before splitting
        """
        images, labels = data

        kFold = KFold(n_splits = k, shuffle = shuffle, random_state = RNGstate) # create a k-fold cross validat

        for split_num, (train_indices, val_indices) in enumerate(kFold.split(images, labels)): # get the indice
            if split_num == split: # convert the indices to the image paths and labels for the desired split

                train_images, train_labels = tf.data.Dataset.from_tensor_slices(np.array(images[train_indices])
                val_images, val_labels = tf.data.Dataset.from_tensor_slices(np.array(images[val_indices])), tf.

                train_split, validation_split = tf.data.Dataset.zip((train_images, train_labels)), tf.data.Data

        return train_split, validation_split # NOTE: perform batching on the returned split and then pass to th

    #------------------------------------------------------------------------ DATA BATCHING --------------------
    def batch_data(self, data, batch_size, drop): # this method is used to batch the dataset into batches of a
        """
        Batch a dataset into batches of a specified batch size.

        args:
        data (tf.data.Dataset): the dataset to be batched
        batch_size (int): the size of the batch
        drop (bool): drop the remaining data if it is less than the batch size
        """
        return data.batch(batch_size, drop_remainder = drop).prefetch(tf.data.experimental.AUTOTUNE) # Enable p

    def process_batch(self, data, batch): # this method is used to process a specified batch of images and labe
        """
        Process a specified batch of images and labels from a batched dataset.

        args:
        data (tf.data.Dataset): the batched dataset
        batch (int): the batch number to be returned

        returns:
        batched_images (np.array): the batch of images
        batched_labels (np.array): the batch of labels
        """

        for images, labels in data.skip(batch).take(1): # Get the specified batch
            images = tf.squeeze(images, axis=1) # remove unnecessary singleton dimension from tensor
            batched_images = [self.load_image(image_path, self.input_shape, self.input_dim).numpy() for image_p
            batched_labels = labels.numpy() # convert the labels to a np.array

        return batched_images, batched_labels # returns the batch of images and labels as a tuple

#============================================================================ MAIN FUNCTION ================
def main(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, display_summary, ar
    # the main function is ran when the script is executed, this is where the training loop and testing loop wi
    df = DataFrame(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
```

```
    RNGstate = df.RNG.integers(0, 1000)
    train_split, validation_split = df.get_kfolds(df.train_dataset, k = 10, split = 0, RNGstate = RNGstate, shu

    batched_train_split = df.batch_data(train_split, batch_size, drop = False)
    train_images, train_labels = df.process_batch(batched_train_split, 0)
    print(train_labels)

    #TODO: implement basic training loop here, i.e batch_data, process_batch, train, save, metrics
    # can then try a loop that implements basic 20% val 80% train split with early stopping.
    # then can implement a loop that implements k-fold cross validation with early stopping.

#================================================================== COMMAND LINE ARGUMENT(S) PARSER =========
if __name__ == "__main__": # this is the command line argument parser, it is used to parse the command line argu
    """run `python network.py --help` for argument information"""
    # ensure you are in the correct directory where py file is located, can change directory by running `cd "<di
    # i.e run `cd "C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmological-data-with-machin

    parser = argparse.ArgumentParser(prog = "network.py")

    parser.add_argument("-dim",       "--inputdim", type = int,   default = 3,          help = "(int) The input
    parser.add_argument("-ixy",     "--inputshape", type = tuple, default = (424, 424), help = "(tuple) The inpu

    parser.add_argument("-r",             "--root", type = str,   default = r"C:/Users/Ma1a_/Desktop/3rd Year P
    parser.add_argument("-l",           "--labels", type = str,   default = r"gz2_train_catalog.parquet", help =
    parser.add_argument("-tl",      "--testlabels", type = str,   default = r"gz2_test_catalog.parquet",  help =
    parser.add_argument("-rep", "--representation", type = str,   default = "vote-fraction",             help =

    parser.add_argument("-sum",        "--summary", type = bool,  default = False,      help = "(bool) Display
    parser.add_argument("-a",     "--architecture", type = str,   default = "128x64xn", help = "(str) The archi
    parser.add_argument("-lr",    "--learningrate", type = float, default = 0.001,      help = "(float) The lea
    parser.add_argument("-wr",       "--weightreg", type = str,   default = None,       help = "(str) The weigh
    parser.add_argument("-bs",       "--batchsize", type = int,   default = 256,        help = "(int) The batch
    parser.add_argument("-e",          "--epochs", type = int,   default = 100,        help = "(int) The numbe
    parser.add_argument("-s",            "--seed", type = int,   default = 0,          help = "(int) The seed

    args = parser.parse_args()

    main(
        input_dim = args.inputdim,
        input_shape = args.inputshape,
        dataset_root = args.root,
        train_catalog = args.labels,
        test_catalog = args.testlabels,
        representation = args.representation,
        display_summary = args.summary,
        architecture = args.architecture,
        learning_rate = args.learningrate,
        weight_reg = args.weightreg,
        batch_size = args.batchsize,
        epochs = args.epochs,
        seed = args.seed
    )

# NOTE:
# At the start of each epoch call the shuffle_dataset method to shuffle the training dataset
# then call the get_batch method to get the next batch of images and labels, and then train the network on the
# increment the batch counter and repeat the process until the end of the epoch.

# TODO:
#       - implement train method and training loop
#       - implement the test method of the ConvolutionalNeuralNetwork class to test the network on the test data
#       - implement the save and load methods of the ConvolutionalNeuralNetwork class to save and load the netw
#       - implement training and testing metrics, i.e accuracy, precision, recall, f1-score, confusion matrix
```

17/03/2024 - Malachy -

In [ ]:

```
"""
Malachy - 17/03/2021
-First attempt at basic training loop.
-Is really slow as the images are loaded from the dataset sequentially in the training loop.
-Would take over 9 days to train the network on the entire dataset. (need to speed it up by an order of magnitu
"""

import pandas as pd
import argparse
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras as keras
import time as time
import datetime as datetime
```

```
import concurrent.futures
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import MaxPooling3D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv3D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

#============================================================== CLASSES AND FUNCTIONS ======================
class ConvolutionalNeuralNetwork():
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, representation, input_shape, i
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        """

        self.display_summary = display_summary
        self.architecture = architecture
        self.learning_rate = learning_rate
        self.weight_reg = weight_reg

        self.representation = representation
        self.label_length = len(get_keys(representation))

        self.input_x, self.input_y = input_shape
        self.input_dim = input_dim

        self.network = self.__init_network(display_summary, architecture)
    #------------------------------------------------------------------------ NETWORK INITIALIZATION ---------
    def __init_network(self, display_summary: bool = False, architecture: str = "216x64") -> tf.keras.Model:
        lr = self.learning_rate

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)

        if architecture == '128x64xn':

            network = Sequential()
            network.add(Conv2D(filters = 32, kernel_size = (7, 7), strides = 3, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(Flatten())
            network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
            network.add(Dropout(0.5))
            network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
            network.add(Dropout(0.5))
            network.add(Dense(self.label_length, activation = None, kernel_regularizer = wr))

            network.compile(loss = 'mean_squared_error', optimizer = tf.keras.optimizers.Adam(learning_rate = l

        elif architecture == '400xn':

            pass

        else:
            raise ValueError(f"Invalid architecture parameter provided: {architecture}")

        if (display_summary):
            network.summary()

        return network
    #------------------------------------------------------------------------ NETWORK TRAINING AND TESTING ---
    def train(self, images, labels):
        """
        Train the network on a batch of images and labels.
```

```
        args:
        images (np.array): the batch of training images
        labels (np.array): the batch of training labels
        """
        return self.network.train_on_batch(images, labels, return_dict = True) # trains the network on a single

    def test(self, images, labels):
        """
        Test the network on a batch of images and labels.

        arrgs:
        images (np.array): the batch of test images
        labels (np.array): the batch of test labels
        """
        return self.network.test_on_batch(images, labels, return_dict = True) # tests the network on a single ba

    #---------------------------------------------------------------- NETWORK WEIGHTS AND BIASES SAVING AND LOA
    def save(self, checkpoint_path):
        """
        Save the network weights and biases to a file.

        args:
        checkpoint_path (str): the path to the directory to save the network weights and biases
        """
        timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.save_weights(path)


    def load(self, checkpoint_path, timestamp):
        """
        Load the network weights and biases from a file.

        args:
        checkpoint_path (str): the path to the directory to load the network weights and biases
        """
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.load_weights(path)

class DataFrame():
    def __init__(self, dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
        """
        Construct a DataFrame containing the represenation of the training and test data in the required format

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_shape (tuple): the input shape of the data, x, y
        input_dim (int): the input dimension of the data, i.e 3 for RGB image
        """

        self.dataset_root = dataset_root

        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog)
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)

        self.representation = representation
        self.input_shape = input_shape
        self.input_dim = input_dim
        self.keys = get_keys(representation)

        self.train_dataset = self.__init_dataset(self.keys, self.train_parquet_file_path)
        self.test_dataset = self.__init_dataset(self.keys, self.test_parquet_file_path)

        self.RNG = np.random.default_rng(seed)

    #---------------------------------------------------------------- DATASET(S) INITIALIZATION ----------
    def __init_dataset(self, keys, catalog_path):
        """
        Initialise a dataset, containing paths to the image files and the labels in the required representation

        returns:
        dataset (tf.data.Dataset): a dataset in the required format, file path column then class label columns
        """
        if self.representation == 'one-hot':

            pandas_df = pd.read_parquet(catalog_path)
            pandas_df = pandas_df.fillna(0)
            class_labels = pandas_df['label'] + 1
```

```
            one_hot = pd.get_dummies(class_labels).astype(int) # convert the class labels to one-hot encoding

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str) # delete the information co

            for idx, _ in enumerate(pandas_df['id_str']):  # construct the actual path to the image file and ad
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array(one_hot)

        elif self.representation == 'vote-fraction':

            pandas_df = pd.read_parquet(catalog_path)
            pandas_df = pandas_df.fillna(0)

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

            return np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)), np.array([pandas_df[key] for k
        else:
            raise ValueError(f"Training DataFrame: Invalid representation parameter provided: {self.representat

    #------------------------------------------------------------------------------- IMAGE PROCESSING -------
    def load_image(self, image_path, input_shape, input_dim):
        """
        Load an image from a file path and convert it to a tf tensor.

        args:
        image_path (str): path to the image file
        input_dim (int): the input dimension of the data

        returns:
        image (tf.tensor): the image as a tf.tensor
        """

        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels = input_dim)
        image = tf.image.convert_image_dtype(image, tf.float32) /255  # Convert image dtype from tf.uint8 to tf
        image = tf.image.resize(image, input_shape)

        return image
    #TODO: implement image augmentation methods here, i.e rotation, flipping, scaling, translation, brightness,
    #------------------------------------------------------------------------------- VALIDATION DATA SPLITTING --
    def get_train_val_split(self, data, val_percent, shuffle, seed):
        """
        Split the training dataset into a training and validation dataset.

        args:
        data (tuple): the training dataset to be split
        val_percent (float): the percentage of the training dataset to be used as the validation dataset
        shuffle (bool): if to shuffle the data before splitting
        """
        images, labels = data

        if shuffle:
            state = self.RNG.integers(low=0, high=65535, size=1)[0]
            combined = tf.data.Dataset.from_tensor_slices((images, labels))
            combined = combined.shuffle(buffer_size = len(images), seed = state)
            images, labels = zip(*combined)

        val_size = int(len(images) * val_percent)

        train_images, train_labels = images[val_size:], labels[val_size:]
        val_images, val_labels = images[:val_size], labels[:val_size]

        train_images, train_labels = tf.data.Dataset.from_tensor_slices(np.array(train_images)), tf.data.Dataset
        val_images, val_labels = tf.data.Dataset.from_tensor_slices(np.array(val_images)), tf.data.Dataset.from_

        train_split, validation_split = tf.data.Dataset.zip((train_images, train_labels)), tf.data.Dataset.zip(

        return train_split, validation_split

    def get_kfolds(self, data, k, split, RNGstate, shuffle):
        """
        Obtain a split of the dataset using k-fold cross validation, containing the k-1 training folds and 1 va

        args:
        data (tuple): the dataset to be split into k-folds
        k (int): the number of folds to be used
        split (int): the split to be returned
        RNGstate (int): random integer from self.RNG to set the random state of the k-fold cross validation
        shuffle (bool): if to shuffle the data before splitting
```

```
        """
        images, labels = data

        kFold = KFold(n_splits = k, shuffle = shuffle, random_state = RNGstate)

        for split_num, (train_indices, val_indices) in enumerate(kFold.split(images, labels)): # get the indices
            if split_num == split: # convert the indices to the image paths and labels for the desired split

                train_images, train_labels = tf.data.Dataset.from_tensor_slices(np.array(images[train_indices]))
                val_images, val_labels = tf.data.Dataset.from_tensor_slices(np.array(images[val_indices])), tf.

                train_split, validation_split = tf.data.Dataset.zip((train_images, train_labels)), tf.data.Data

        return train_split, validation_split # NOTE: perform batching on the returned split and then pass to th

    #------------------------------------------------------------------------ DATA BATCHING --------------------
    def batch_data(self, data, batch_size, drop, shuffle):
        """
        Batch a dataset into batches of a specified batch size.

        args:
        data (tf.data.Dataset): the dataset to be batched
        batch_size (int): the size of the batch
        drop (bool): drop the remaining data if it is less than the batch size
        """
        if shuffle:
            state = self.RNG.integers(low=0, high=65535, size=1)[0]
            batches = data.batch(batch_size, drop_remainder = drop).shuffle(buffer_size = data.cardinality(), s
        else:
            batches = data.batch(batch_size, drop_remainder = drop).prefetch(tf.data.experimental.AUTOTUNE)

        return batches # Enable prefetching to allow for the data to be batched in parallel with the training o

    def process_batch(self, data, batch):
        """
        Process a specified batch of images and labels from a batched dataset.

        args:
        data (tf.data.Dataset): the batched dataset
        batch (int): the batch number to be returned

        returns:
        batched_images (np.array): the batch of images
        batched_labels (np.array): the batch of labels
        """

        batched_images, batched_labels = [], []

        for images, labels in data.skip(batch).take(1):
            images = tf.squeeze(images, axis=1)
            image_paths = images.numpy()
            labels = labels.numpy()

            for idx, image_path in enumerate(image_paths):
                image = self.load_image(image_path, self.input_shape, self.input_dim).numpy() if image_path is
                if image is not None:
                    batched_images.append(image)
                    batched_labels.append(labels[idx])

        return np.array(batched_images), np.array(batched_labels)

def get_keys(representation):
    """
    Obtain the keys of the class labels when accessing the class labels from a Pandas DataFrame.
    """
    if representation == 'one-hot':
        keys = [
        0,
        1,
        2,
        4,
        5,
        6,
        7
    ]
        return keys
    elif representation == 'vote-fraction':
        keys = [
            'smooth-or-featured-gz2_smooth_fraction',
            'smooth-or-featured-gz2_featured-or-disk_fraction',
            'smooth-or-featured-gz2_artifact_fraction',
            'disk-edge-on-gz2_yes_fraction',
            'disk-edge-on-gz2_no_fraction',
```

```
            'bar-gz2_yes_fraction',
            'bar-gz2_no_fraction',
            'has-spiral-arms-gz2_yes_fraction',
            'has-spiral-arms-gz2_no_fraction',
            'bulge-size-gz2_no_fraction',
            'bulge-size-gz2_just-noticeable_fraction',
            'bulge-size-gz2_obvious_fraction',
            'bulge-size-gz2_dominant_fraction',
            'something-odd-gz2_yes_fraction',
            'something-odd-gz2_no_fraction',
            'how-rounded-gz2_round_fraction',
            'how-rounded-gz2_in-between_fraction',
            'how-rounded-gz2_cigar_fraction',
            'bulge-shape-gz2_round_fraction',
            'bulge-shape-gz2_boxy_fraction',
            'bulge-shape-gz2_no-bulge_fraction',
            'spiral-winding-gz2_tight_fraction',
            'spiral-winding-gz2_medium_fraction',
            'spiral-winding-gz2_loose_fraction',
            'spiral-arm-count-gz2_1_fraction',
            'spiral-arm-count-gz2_2_fraction',
            'spiral-arm-count-gz2_3_fraction',
            'spiral-arm-count-gz2_4_fraction',
            'spiral-arm-count-gz2_more-than-4_fraction',
            'spiral-arm-count-gz2_cant-tell_fraction',
        ]
        return keys
    else:
            raise ValueError(f"Invalid representation parameter provided: {representation}")

#================================================================================ MAIN FUNCTION ================
def main(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, display_summary, ar

    gpus = tf.config.experimental.list_physical_devices('GPU')

    if gpus:
        for gpu in gpus:
            os.environ["TF_GPU_ALLOCATOR"] = "cuda_malloc_async"
            tf.config.experimental.set_memory_growth(gpu, True)
            tf.config.experimental.set_virtual_device_configuration(gpu, [tf.config.experimental.VirtualDeviceC

    df = DataFrame(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
    cnn = ConvolutionalNeuralNetwork(display_summary, architecture, learning_rate, weight_reg, representation, 

    train_split, val_split = df.get_train_val_split(df.train_dataset, val_percent = 0.2, shuffle = True, seed =
    epoch_time_hist = []

    # NOTE: most of the runtime is spent on loading the images from disk, can try to preload as many batches of
    for epoch in range(0,epochs):

        epoch_start_time = time.time()
        iteration = 0
        val_loss, val_accuracy = 0, 0 # no val metrics at start of epoch
        iter_time_hist = []

        if epoch == 0:
            pred_runtime = 0
        else:
            pred_runtime = np.mean(epoch_time_hist) * (epochs - epoch)

        train_batches = df.batch_data(train_split, batch_size, drop = False, shuffle = True)

        for iteration in range(0, (len(train_batches))): # perform training on each batch

            iter_start_time = time.time()
            time_fetch_start = time.time()
            train_images, train_labels = df.process_batch(train_batches, iteration)
            time_fetch_end = time.time()
            history = cnn.train(train_images, train_labels)
            loss, accuracy = history['loss'], history['accuracy']
            iter_end_time = time.time()

            # Code to display the progress of the training as a progress bar
            iter_time = iter_end_time - iter_start_time
            iter_time_hist.append(iter_time)

            bar_length = 50
            progress_percentage = ((iteration + 1) / len(train_batches)) * 100
            num_chars = int(bar_length * (progress_percentage / 100))

            print(
                f'Epoch: {epoch}/{epochs} | Iter: {iteration}/{len(train_batches)} | Loss: {loss:.5f} | Accuracy
                f'{"■" * num_chars}{" " * (bar_length - num_chars)}] {progress_percentage:.1f} %',
```

```
                f' | It: {iter_time:.2E} s | Imft: {time_fetch_end - time_fetch_start:.2E} s ({(((time_fetch_en
                end = '\r',
                flush = True
            )

        # perform validation at the end of each epoch
        val_batches = df.batch_data(val_split, batch_size, drop = False, shuffle = True)
        val_images, val_labels = df.process_batch(val_batches, epoch)
        val_history = cnn.test(val_images, val_labels)
        val_loss, val_accuracy = val_history['loss'], val_history['accuracy']

        epoch_end_time = time.time()
        runtime = epoch_end_time - epoch_start_time
        epoch_time_hist.append(runtime)

        # print val metrics
        print(
            f' | Val loss: {val_loss:.5f} | Val accuracy: {val_accuracy:.5f} | Epoch runtime {runtime:.2E} s | 
            end = '\n',
            flush = True
            )

    #TODO: implement basic training loop here, i.e batch_data, process_batch, train, save, metrics
    # can then try a loop that implements basic 20% val 80% train split with early stopping.
    # then can implement a loop that implements k-fold cross validation with early stopping.

#================================================================== COMMAND LINE ARGUMENT(S) PARSER =========
if __name__ == "__main__":
    """run `python network.py --help` for argument information"""
    # ensure you are in the correct directory where py file is located, can change directory by running `cd "<d
    # i.e run `cd "C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmological-data-with-machin

    parser = argparse.ArgumentParser(prog = "network.py")

    parser.add_argument("-dim",       "--inputdim", type = int,   default = 3,          help = "(int) The input
    parser.add_argument("-ixy",     "--inputshape", type = tuple, default = (424, 424), help = "(tuple) The inp

    parser.add_argument("-r",             "--root", type = str,   default = r"C:/Users/Ma1a_/Desktop/3rd Year P
    #parser.add_argument("-r",             "--root", type = str,   default = r"/Users/aroushijimulia/Downloads/
    parser.add_argument("-l",           "--labels", type = str,   default = r"gz2_train_catalog.parquet", help =
    parser.add_argument("-tl",      "--testlabels", type = str,   default = r"gz2_test_catalog.parquet",  help =
    parser.add_argument("-rep", "--representation", type = str,   default = "vote-fraction",             help =

    parser.add_argument("-sum",        "--summary", type = bool,  default = False,      help = "(bool) Display 
    parser.add_argument("-a",     "--architecture", type = str,   default = "128x64xn", help = "(str) The archi
    parser.add_argument("-lr",    "--learningrate", type = float, default = 0.001,      help = "(float) The lea
    parser.add_argument("-wr",       "--weightreg", type = str,   default = None,       help = "(str) The weigh
    parser.add_argument("-bs",       "--batchsize", type = int,   default = 32,        help = "(int) The batch 
    parser.add_argument("-e",           "--epochs", type = int,   default = 500,        help = "(int) The numbe
    parser.add_argument("-s",             "--seed", type = int,   default = 0,          help = "(int) The seed 

    args = parser.parse_args()

    main(
        input_dim = args.inputdim,
        input_shape = args.inputshape,
        dataset_root = args.root,
        train_catalog = args.labels,
        test_catalog = args.testlabels,
        representation = args.representation,
        display_summary = args.summary,
        architecture = args.architecture,
        learning_rate = args.learningrate,
        weight_reg = args.weightreg,
        batch_size = args.batchsize,
        epochs = args.epochs,
        seed = args.seed
    )

# TODO:
#       - implement train method and training loop
#       - implement the test method of the ConvolutionalNeuralNetwork class to test the network on the test dat
#       - implement the save and load methods of the ConvolutionalNeuralNetwork class to save and load the netw
#       - implement training and testing metrics, i.e accuracy, precision, recall, f1-score, confusion matrix

# preload as many batches of size 32 of images and labels that fit into memory -> then train on each batch unti
# use tensorboard to store the training/test metrics and display them in a graph
```

22/03/2024 - Malachy -

In [ ]:

```
"""
22/03/2024
Malachy -
```

```
-Got the basic training/testing working, methods are implemented in the ConvolutionalNeuralNetwork class to tra
-Had to do refactoring to use tensorflow datasets API along with model.fit() and model.evaluate() methods to tr
dataset is now loaded in parallel when training and testing the network, this allows for faster training and te
"""

import pandas as pd
import argparse
import os
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras as keras
import time as time
import datetime as datetime
import concurrent.futures
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import MaxPooling3D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv3D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

#============================================================== CLASSES AND FUNCTIONS ======================
class ConvolutionalNeuralNetwork():
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, representation, input_shape, i
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        """

        self.display_summary = display_summary
        self.architecture = architecture
        self.learning_rate = learning_rate
        self.weight_reg = weight_reg

        self.representation = representation
        self.label_length = len(get_keys(representation))

        self.input_x, self.input_y = input_shape
        self.input_dim = input_dim

        self.network = self.__init_network(display_summary, architecture)

        self.train_history = None
        self.test_history = None
    #------------------------------------------------------------------------------ NETWORK INITIALIZATION ---------
    def __init_network(self, display_summary: bool = False, architecture: str = "216x64") -> tf.keras.Model:
        lr = self.learning_rate

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)

        if architecture == '128x64xn':

            network = Sequential()
            network.add(Conv2D(filters = 32, kernel_size = (7, 7), strides = 3, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(MaxPooling2D(pool_size = (2,2)))
            network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
            network.add(Flatten())
            network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
            network.add(Dropout(0.5))
            network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
            network.add(Dropout(0.5))
            network.add(Dense(self.label_length, activation = None, kernel_regularizer = wr))
```

```
            network.compile(loss = 'mean_squared_error', optimizer = tf.keras.optimizers.Adam(learning_rate = l

        elif architecture == '400xn':

            pass

        else:
            raise ValueError(f"Invalid architecture parameter provided: {architecture}")

        if (display_summary):
            network.summary()

        return network
    #------------------------------------------------------------------------- NETWORK TRAINING AND TESTING ---
    def train(self, train_data, validation_data, dataframe, train_batch_size, val_batch_size, val_steps, epochs

        train_data = train_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE # allow for tensorflow to dynamically decide the
        ).batch(
            train_batch_size # batch the dataset into batches of size batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE # allow for prefetching to parallelize the loading of the images and
        )

        validation_data = validation_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE
        ).batch(
            val_batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE
        )

        self.train_history = self.network.fit(
            train_data,
            epochs = epochs,
            verbose = 1,
            callbacks = None,
            validation_data = validation_data,
            shuffle = True,
            initial_epoch = initial_epoch,
            steps_per_epoch = None,
            validation_steps = val_steps,
            validation_batch_size = val_batch_size,
            validation_freq = 1,
            max_queue_size = 10,
            workers = 1,
            use_multiprocessing = True
        )

    def test(self, test_data, dataframe, test_batch_size, test_steps):

        test_data = test_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE
        ).batch(
            test_batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE
        )

        self.test_history = self.network.evaluate(
            test_data,
            batch_size = test_batch_size,
            verbose = 1,
            steps = test_steps,
            callbacks = None,
            max_queue_size = 10,
            workers = 1,
            use_multiprocessing = True,
            return_dict = True
        )

    #------------------------------------------------------------------ NETWORK WEIGHTS AND BIASES SAVING AND LO
    def save(self, checkpoint_path):
        """
        Save the network weights and biases to a file.

        args:
        checkpoint_path (str): the path to the directory to save the network weights and biases
        """
        timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

```
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.save_weights(path)


    def load(self, checkpoint_path, timestamp):
        """
        Load the network weights and biases from a file.

        args:
        checkpoint_path (str): the path to the directory to load the network weights and biases
        """
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.load_weights(path)

class DataFrame():
    def __init__(self, dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
        """
        Construct a DataFrame containing the represenation of the training and test data in the required format

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_shape (tuple): the input shape of the data, x, y
        input_dim (int): the input dimension of the data, i.e 3 for RGB image
        """

        self.dataset_root = dataset_root

        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog)
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)

        self.representation = representation
        self.input_shape = input_shape
        self.input_dim = input_dim
        self.keys = get_keys(representation)

        self.train_dataset = self.__init_dataset(self.keys, self.train_parquet_file_path)
        self.test_dataset = self.__init_dataset(self.keys, self.test_parquet_file_path)

        tf.random.set_seed(seed)

    #------------------------------------------------------------------------ DATASET(S) INITIALIZATION ----------
    def __init_dataset(self, keys, catalog_path):
        """
        Initialise a dataset, containing paths to the image files and the labels in the required representation

        returns:
        dataset (tf.data.Dataset): a dataset in the required format, file path column then class label columns
        """
        if self.representation == 'one-hot':

            pandas_df = pd.read_parquet(catalog_path)
            pandas_df = pandas_df.fillna(0)
            class_labels = pandas_df['label'] + 1
            one_hot = pd.get_dummies(class_labels).astype(int) # convert the class labels to one-hot encoding

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str) # delete the information co

            for idx, _ in enumerate(pandas_df['id_str']):  # construct the actual path to the image file and ad
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf

                if not os.path.exists(pandas_df['file_loc'][idx]): # check if the file exists at the constructe
                    pandas_df = pandas_df.drop(idx)
                    one_hot = one_hot.drop(idx)

            image_path_ds = tf.data.Dataset.from_tensor_slices(np.array(pandas_df['file_loc']).reshape((len(pan
            labels_ds = tf.data.Dataset.from_tensor_slices(np.array(one_hot))

            return tf.data.Dataset.zip((image_path_ds, labels_ds))

        elif self.representation == 'vote-fraction':

            pandas_df = pd.read_parquet(catalog_path)
            pandas_df = pandas_df.fillna(0)

            pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

            for idx, _ in enumerate(pandas_df['id_str']):
                pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subf
```

```
                if not os.path.exists(pandas_df['file_loc'][idx]):
                    pandas_df = pandas_df.drop(idx)

            image_path_ds = tf.data.Dataset.from_tensor_slices(np.array(pandas_df['file_loc']).reshape((len(pan
            labels_ds = tf.data.Dataset.from_tensor_slices(np.array([pandas_df[key] for key in keys]).reshape((

            return tf.data.Dataset.zip((image_path_ds, labels_ds))
        else:
            raise ValueError(f"Training DataFrame: Invalid representation parameter provided: {self.representat

    #------------------------------------------------------------------------------- IMAGE PROCESSING -------
    def fetch_image_label_pair(self, image_path, label):

        image_path = tf.squeeze(image_path)
        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels = self.input_dim)
        image = tf.image.convert_image_dtype(image, tf.float32) /255  # Convert image dtype from tf.uint8 to tf
        image = tf.image.resize(image, self.input_shape)

        return image, label

    #TODO: implement image augmentation methods here, i.e rotation, flipping, scaling, translation, brightness,
    #------------------------------------------------------------------------------- VALIDATION DATA SPLITTING --
    def get_train_val_split(self, data, val_fraction, shuffle, seed):
        """
        Split the training dataset into a training and validation dataset.

        args:
        data (tf.data.Dataset): the input dataset to be split
        val_fraction (float): the fraction of the dataset to be used for validation [0-1]
        shuffle (bool): shuffle the dataset before splitting
        seed (int): the seed for the random number generator

        returns:
        (tuple) of training and validation datasets
            train_split (tf.data.Dataset): the training dataset
            validation_split (tf.data.Dataset): the validation dataset
        """
        train_split, validation_split = tf.keras.utils.split_dataset(data, left_size = 1 - val_fraction, right_

        return train_split, validation_split

    def get_kfolds(self):
        pass

def get_keys(representation):
    """
    Obtain the keys of the class labels when accessing the class labels from a Pandas DataFrame.

    args:
    representation (str): the representation of the data, i.e one-hot or vote fraction

    returns:
    keys (list): the keys of the class labels
    """
    if representation == 'one-hot':
        keys = [
        0,
        1,
        2,
        4,
        5,
        6,
        7
    ]
        return keys
    elif representation == 'vote-fraction':
        keys = [
            'smooth-or-featured-gz2_smooth_fraction',
            'smooth-or-featured-gz2_featured-or-disk_fraction',
            'smooth-or-featured-gz2_artifact_fraction',
            'disk-edge-on-gz2_yes_fraction',
            'disk-edge-on-gz2_no_fraction',
            'bar-gz2_yes_fraction',
            'bar-gz2_no_fraction',
            'has-spiral-arms-gz2_yes_fraction',
            'has-spiral-arms-gz2_no_fraction',
            'bulge-size-gz2_no_fraction',
            'bulge-size-gz2_just-noticeable_fraction',
            'bulge-size-gz2_obvious_fraction',
            'bulge-size-gz2_dominant_fraction',
            'something-odd-gz2_yes_fraction',
            'something-odd-gz2_no_fraction',
```

```
            'how-rounded-gz2_round_fraction',
            'how-rounded-gz2_in-between_fraction',
            'how-rounded-gz2_cigar_fraction',
            'bulge-shape-gz2_round_fraction',
            'bulge-shape-gz2_boxy_fraction',
            'bulge-shape-gz2_no-bulge_fraction',
            'spiral-winding-gz2_tight_fraction',
            'spiral-winding-gz2_medium_fraction',
            'spiral-winding-gz2_loose_fraction',
            'spiral-arm-count-gz2_1_fraction',
            'spiral-arm-count-gz2_2_fraction',
            'spiral-arm-count-gz2_3_fraction',
            'spiral-arm-count-gz2_4_fraction',
            'spiral-arm-count-gz2_more-than-4_fraction',
            'spiral-arm-count-gz2_cant-tell_fraction',
        ]
        return keys
    else:
        raise ValueError(f"Invalid representation parameter provided: {representation}")

#============================================================================== MAIN FUNCTION ================
def main(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, display_summary, ar

    gpus = tf.config.experimental.list_physical_devices('GPU')

    if gpus:
        for gpu in gpus:
            os.environ["TF_GPU_ALLOCATOR"] = "cuda_malloc_async"
            tf.config.experimental.set_memory_growth(gpu, True)
            tf.config.experimental.set_virtual_device_configuration(gpu, [tf.config.experimental.VirtualDeviceC

    df = DataFrame(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed)
    CNN = ConvolutionalNeuralNetwork(display_summary, architecture, learning_rate, weight_reg, representation, 

    train_split, val_split = df.get_train_val_split(df.train_dataset, val_fraction = 0.2, shuffle = True, seed =
    CNN.train(train_data= train_split, validation_data = val_split, dataframe = df, train_batch_size = batch_si

    #TODO: implement autosaving and loading of the network weights and biases
    # implement kfold cross validation with tf.datasets or sklearn
    # implement training and testing metrics, i.e accuracy, precision, recall, f1-score, confusion matrix
    # implement plotting of the training and testing metrics

#================================================================== COMMAND LINE ARGUMENT(S) PARSER =========
if __name__ == "__main__":
    """run `python network.py --help` for argument information"""
    # ensure you are in the correct directory where py file is located, can change directory by running `cd "<d
    # i.e run `cd "C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmological-data-with-machin

    parser = argparse.ArgumentParser(prog = "network.py")

    parser.add_argument("-dim",       "--inputdim", type = int,   default = 3,          help = "(int) The input
    parser.add_argument("-ixy",     "--inputshape", type = tuple, default = (424, 424), help = "(tuple) The inp

    parser.add_argument("-r",             "--root", type = str,   default = r"C:/Users/Ma1a_/Desktop/3rd Year P
    parser.add_argument("-l",           "--labels", type = str,   default = r"gz2_train_catalog.parquet", help =
    parser.add_argument("-tl",      "--testlabels", type = str,   default = r"gz2_test_catalog.parquet",  help =
    parser.add_argument("-rep", "--representation", type = str,   default = "vote-fraction",             help =

    parser.add_argument("-sum",        "--summary", type = bool,  default = False,      help = "(bool) Display 
    parser.add_argument("-a",     "--architecture", type = str,   default = "128x64xn", help = "(str) The archi
    parser.add_argument("-lr",    "--learningrate", type = float, default = 0.001,      help = "(float) The lea
    parser.add_argument("-wr",       "--weightreg", type = str,   default = None,       help = "(str) The weigh
    parser.add_argument("-bs",       "--batchsize", type = int,   default = 32,        help = "(int) The batch 
    parser.add_argument("-e",           "--epochs", type = int,   default = 500,        help = "(int) The numbe
    parser.add_argument("-s",             "--seed", type = int,   default = 0,          help = "(int) The seed 

    args = parser.parse_args()

    main(
        input_dim = args.inputdim,
        input_shape = args.inputshape,
        dataset_root = args.root,
        train_catalog = args.labels,
        test_catalog = args.testlabels,
        representation = args.representation,
        display_summary = args.summary,
        architecture = args.architecture,
        learning_rate = args.learningrate,
        weight_reg = args.weightreg,
        batch_size = args.batchsize,
        epochs = args.epochs,
        seed = args.seed
    )
```

Aroushi -

Week Starting March 11th:

Thursday March 14th 10am-11am: SUPERVISOR MEETING

- Review of work done so far
- Since Malachy is working on code from scatch, Adam suggested that Aroushi try a different approach and use higher level keras API functions for the same goal. In this way, the results achieved by the simpler approach and more detailed approach can be compared at the end

Malachy -

```
df = DataFrame(dataset_root, train_catalog, test_catalog, representation, input_shape, input_dim, seed) # create the DataFrame object
CNN = ConvolutionalNeuralNetwork(display_summary, architecture, learning_rate, weight_reg, representation, input_shape, input_dim) # create the Convolutional Neural Network object

train_split, val_split = df.get_train_val_split(df.train_dataset, val_fraction = 0.2, shuffle = True, seed = seed) # create the training and validation datasets
CNN.train(train_data= train_split, validation_data = val_split, dataframe = df, train_batch_size = batch_size, val_batch_size = 16, val_steps = 2, epochs = epochs, initial_epoch = 0) # train the network
```

Basic use of main.py file as of 22/03/2024

26/03/2024

had bug where network wouldn't learn, loss either exploded towards infinity or hovered around value.

```
4186/4186 [==============================] - 162s 37ms/step - loss: 0.1347 - accuracy: 0.0324
Epoch 2/500
4186/4186 [==============================] - 149s 35ms/step - loss: 0.1333 - accuracy: 0.0313
Epoch 3/500
4186/4186 [==============================] - 142s 34ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 4/500
4186/4186 [==============================] - 140s 34ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 5/500
4186/4186 [==============================] - 141s 34ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 6/500
4186/4186 [==============================] - 137s 33ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 7/500
4186/4186 [==============================] - 169s 40ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 8/500
4186/4186 [==============================] - 139s 33ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 9/500
4186/4186 [==============================] - 143s 34ms/step - loss: 0.1333 - accuracy: 0.0311
Epoch 10/500
 511/4186 [==>...........................] - ETA: 2:10 - loss: 0.1337 - accuracy: 0.0322
```

loss hovering around value

```
_________________________________________________________________
Epoch 1/500
2024-03-26 16:38:54.518871: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8100
1126/5233 [=====>........................] - ETA: 2:19 - loss: 1383590658048.0000 - accuracy: 0.3360
```

exploding loss

issue was with how dataset was being constructed.

```
image_path_ds = tf.data.Dataset.from_tensor_slices(np.array(pandas_df['file_loc']).reshape((len(pandas_df), 1)))
labels_ds = tf.data.Dataset.from_tensor_slices(np.array([pandas_df[key] for key in keys]).reshape((len(pandas_df), len(keys))))
```

problematic lines

gave every image the a part of the column corresponding to a label key up to a the number of labels. So it looked correct shape/length wise but instead every image was getting like 3 smooth vote fractions then the next was getting 3 featured-disk votes ect (absolute nonsense labels).

managed to fix it!!

```
Epoch 1/500
2024-03-26 17:32:35.344841: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8100
4186/4186 [==============================] - ETA: 0s - loss: 0.7758 - accuracy: 0.7523
Epoch 1: val_loss improved from inf to 0.79057, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmological-
4186/4186 [==============================] - 148s 34ms/step - loss: 0.7758 - accuracy: 0.7523 - val_loss: 0.7906 - val_accuracy: 0.6250
Epoch 2/500
4185/4186 [============================>.] - ETA: 0s - loss: 0.7702 - accuracy: 0.7539
Epoch 2: val_loss improved from 0.79057 to 0.68380, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmolog
4186/4186 [==============================] - 147s 35ms/step - loss: 0.7702 - accuracy: 0.7539 - val_loss: 0.6838 - val_accuracy: 0.8438
Epoch 3/500
4186/4186 [==============================] - ETA: 0s - loss: 0.7695 - accuracy: 0.7540
Epoch 3: val_loss did not improve from 0.68380
4186/4186 [==============================] - 145s 34ms/step - loss: 0.7695 - accuracy: 0.7540 - val_loss: 0.7869 - val_accuracy: 0.7812
Epoch 4/500
 282/4186 [=>............................] - ETA: 2:09 - loss: 0.7692 - accuracy: 0.7548
```

main code is now a collection of seperate python files that interact.

imports.py contains the imports used throughout the files dataframe.py constructs a dataframe and peforms operations on it related to loading of images and creating training and validation splits of the dataset keys.py contains the class labels, keys, used when acessing the catalog files network.py constructs a CNN of a specified architecture and performs training and testing of the network, along with saving and loading of weights

these files interact through main.py, which contaians command line arguments to allow variables to be changed.

Aroushi -

While Malachy works on building neural network from scratch, I am developing code using keras API functions, such that we can compare the different methods and the advantages of each one

In [ ]:

```
"""

CNN using Keras Draft 2 - 20/03/2024
This model currently uses only the top layers of the decision tree (3 classes)
Problem to be solved: T4 GPU on CoLab times out and does not let the training complete
so we only have results from the first two epochs
The next draft will attempt to make code more efficient and add more classes

"""

#Imported libraries and downloaded dataset as usual


#Functions

def load_class_data(file_path):
    """
    Summary: Loads class label data in one-hot format
    Input: Path to parquet file
    Output: Pandas dataframe with one hot class labels
    """
    df = pd.read_parquet(file_path)
    class_labels = df['label'] + 1
    class_labels_one_hot = to_categorical(class_labels, num_classes=8) #turns into one-hot
    return class_labels_one_hot


def load_image(full_image_path):
    """
    Summary: Loads and resizes image
    Input: Path to image file
    Output: Image tensor
    """
    image = tf.io.read_file(full_image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [224, 224])
    image = image / 255.0
    return image

def load_images(catalog, image_path):
    """
    Summary: Loads image tensors into a list
    Input: Main pandas dataframe which includes image locations, image paths
    Output: List of image tensors
    """
    images = []
    for relative_path in catalog['file_loc']:
        relative_path = relative_path[-29:] #Picks out relevant bit of full path
        full_image_path = os.path.join(image_path, str(relative_path))
        image = load_image(full_image_path)
        images.append(image)
    return images
```

```
def display_image_from_catalog(catalog, image_path, index):
    """
    Summary: Can be used to open individual images if needed
    Input: Main pandas dataframe, image path, image index
    Output: Displays image
    """
    full_image_path = os.path.join(image_path, catalog['file_loc'].iloc[index])
    im = Image.open(full_image_path)
    im.show()

#Testing if catalog is downloaded as expected

import matplotlib.pyplot as plt

def plot_images(catalog, num_images=6):
  fig, axes = plt.subplots(1,num_images, sharex=True, sharey=True, figsize=(10,5))
  for i in range(num_images):
    image = Image.open(catalog.iloc[i], 'r')
    axes[i].imshow(np.asarray(image))
    axes[i].set_xticklabels([])
    axes[i].set_yticklabels([])
  plt.subplots_adjust(wspace=0, hspace=0)
  plt.show()


batch_size=32
validation_split=0.2
original_size=424

train_catalog, validation_catalog = train_test_split(catalog, test_size=validation_split, random_state=42)

# Training data generator with augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=90,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest')

# Validation data generator (only rescaling)
validation_datagen = ImageDataGenerator(rescale=1./255)

target_columns = ['smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction',

train_iterator = train_datagen.flow_from_dataframe(
    dataframe=train_catalog,
    x_col='file_loc',
    y_col=target_columns,
    batch_size=batch_size,
    target_size=(224,224),
    class_mode='raw'  # 'raw' because your labels are continuous values
)


validation_iterator = validation_datagen.flow_from_dataframe(
    dataframe=validation_catalog,
    x_col='file_loc',
    y_col=target_columns,
    target_size=(224,224),
    batch_size=batch_size,
    class_mode='raw'
)

checkpoint_filepath = '/tmp/checkpoint_layer1'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1)

#Model

def build_model():
  resnet_model = Sequential()

  pretrained_model = tf.keras.applications.ResNet50(include_top=False,
                                                     input_shape=(224,224,3),
                                                     pooling='avg',
```

```
                                                weights='imagenet')
  for layer in pretrained_model.layers:
    layer.trainable = True

  resnet_model.add(pretrained_model)
  resnet_model.add(Flatten())
  resnet_model.add(Dense(512, activation='relu'))
  resnet_model.add(Dense(3, activation='softmax'))

  resnet_model.compile(loss='categorical_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])

  return resnet_model

model = build_model()

#Training

train_val_catalog, test_catalog = train_test_split(catalog, test_size=0.1, random_state=42)

train_catalog, val_catalog = train_test_split(train_val_catalog, test_size=0.2, random_state=42)

history = model.fit(train_iterator,
                    steps_per_epoch = len(train_catalog) // batch_size,
                    validation_data = validation_iterator,
                    validation_steps=len(val_catalog,) // batch_size,
                    verbose=1,

                    epochs=10)
```

In [ ]:
```
"""

Results so far:

Epoch 1/10
3767/3767 [==============================] - 2222s 577ms/step - loss: 0.6033 - accuracy: 0.9120 - val_loss: 0.5
Epoch 2/10
104/3767 [..............................] - ETA: 32:51 - loss: 0.5944 - accuracy: 0.9246

"""
```

26/03/2024 Malachy -

In [ ]:
```
"""
imports.py - 26/03/2024
"""

import matplotlib.pyplot as plt
import numpy as np
import time as time
import datetime as datetime
import os
import pandas as pd
import argparse

import tensorflow as tf
import keras as keras

from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import MaxPooling3D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv3D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model

from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
```

26/03/2024 Malachy -

In [ ]:
```
"""
keys.py - 26/03/2024
"""

def get_keys(class_labels):
    """
```

```
    Obtain the keys of the class labels when accessing the class labels from a Pandas DataFrame.

    args:
    class_labels (str): the class labels to consider.

    returns:
    keys (list): the keys of the class labels
    """
    if class_labels == 'binary':
        keys = [
        'smooth-or-featured-gz2_smooth_fraction',
        ]
        return keys

    elif class_labels == 'Q1':
        keys = [
            'smooth-or-featured-gz2_smooth_fraction',
            'smooth-or-featured-gz2_featured-or-disk_fraction',
            'smooth-or-featured-gz2_artifact_fraction',
        ]
        return keys

    elif class_labels == 'all':
        keys = [
            'smooth-or-featured-gz2_smooth_fraction',
            'smooth-or-featured-gz2_featured-or-disk_fraction',
            'smooth-or-featured-gz2_artifact_fraction',
            'disk-edge-on-gz2_yes_fraction',
            'disk-edge-on-gz2_no_fraction',
            'bar-gz2_yes_fraction',
            'bar-gz2_no_fraction',
            'has-spiral-arms-gz2_yes_fraction',
            'has-spiral-arms-gz2_no_fraction',
            'bulge-size-gz2_no_fraction',
            'bulge-size-gz2_just-noticeable_fraction',
            'bulge-size-gz2_obvious_fraction',
            'bulge-size-gz2_dominant_fraction',
            'something-odd-gz2_yes_fraction',
            'something-odd-gz2_no_fraction',
            'how-rounded-gz2_round_fraction',
            'how-rounded-gz2_in-between_fraction',
            'how-rounded-gz2_cigar_fraction',
            'bulge-shape-gz2_round_fraction',
            'bulge-shape-gz2_boxy_fraction',
            'bulge-shape-gz2_no-bulge_fraction',
            'spiral-winding-gz2_tight_fraction',
            'spiral-winding-gz2_medium_fraction',
            'spiral-winding-gz2_loose_fraction',
            'spiral-arm-count-gz2_1_fraction',
            'spiral-arm-count-gz2_2_fraction',
            'spiral-arm-count-gz2_3_fraction',
            'spiral-arm-count-gz2_4_fraction',
            'spiral-arm-count-gz2_more-than-4_fraction',
            'spiral-arm-count-gz2_cant-tell_fraction',
        ]
        return keys
    else:
        raise ValueError(f"Invalid representation parameter provided: {class_labels}")
```

26/03/2024 Malachy -

In [ ]:

```
"""
dataframe.py - 26/03/2024
"""

from packages.imports import os, tf, pd, np
from dataframe.keys import get_keys

class DataFrame():
    def __init__(self, dataset_root, train_catalog, test_catalog, classes, one_hot, input_shape, input_dim, see
        """
        Construct a DataFrame containing the represenation of the training and test data

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
        class_labels (str): the class labels of the dataset to use, i.e smooth-or-featured-gz2_smooth_fraction,
        one_hot (bool): whether to one-hot encode the labels
        input_shape (tuple): the input shape of the data, x, y
        input_dim (int): the input dimension of the data, i.e 3 for RGB image
        seed (int): the seed for the random number generator
```

```
        ds_fraction (float): the fraction of the dataset to use, range: [0, 1]
        """

        self.dataset_root = dataset_root

        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog)
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)

        self.classes = classes
        self.input_shape = input_shape
        self.input_dim = input_dim
        self.keys = get_keys(classes)
        self.one_hot = one_hot
        self.seed = seed

        if ds_fraction < 0 or ds_fraction > 1:
            raise ValueError("The fraction of the dataset to use must be in the range [0, 1]")
        else:
            self.ds_fraction = ds_fraction

        self.train_dataset = self.__init_dataset(self.keys, self.train_parquet_file_path, self.one_hot)
        self.test_dataset = self.__init_dataset(self.keys, self.test_parquet_file_path, self.one_hot)

        tf.random.set_seed(seed)

    #------------------------------------------------------------------- DATASET(S) INITIALIZATION ----------
    def __init_dataset(self, keys, catalog_path, one_hot):
        """
        Initialise a dataset, containing paths to the image files and the labels.

        args:
        keys (list): the keys of the class labels
        catalog_path (str): the path to the catalog file
        one_hot (bool): whether to one-hot encode the labels

        returns:
        dataset (tf.data.Dataset): a zipped dataset containing a tuple of a tf.data.Dataset of image paths and a
        """

        pandas_df = pd.read_parquet(catalog_path)
        pandas_df = pandas_df.fillna(0)

        pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

        for idx, _ in enumerate(pandas_df['id_str']):
            pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subfolde

            if not os.path.exists(pandas_df['file_loc'][idx]):
                pandas_df = pandas_df.drop(idx)

        image_path_ds = tf.data.Dataset.from_tensor_slices(pandas_df['file_loc'])
        labels_ds = tf.data.Dataset.from_tensor_slices(pandas_df[keys])

        if one_hot: # round the labels to the nearest integer to encode as one-hot (0 or 1)
            labels_ds = labels_ds.map(tf.round)

        return tf.data.Dataset.zip((image_path_ds, labels_ds)).take(int(len(labels_ds) * self.ds_fraction)) # ta

    #------------------------------------------------------------------------------- IMAGE PROCESSING -------
    def fetch_image_label_pair(self, image_path, label):
        """
        Fetches an image and its corresponding label from the dataset.

        args:
        image_path (tf.Tensor): the path to the image file
        label (tf.Tensor): the label of the image

        returns:
        (tuple): of the image and its corresponding label
            image (tf.Tensor): the requested image
            label (tf.Tensor): the requested label
        """
        image_path = tf.squeeze(image_path)
        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels = self.input_dim)
        image = tf.image.convert_image_dtype(image, tf.float32) # Convert image dtype from tf.uint8 to tf.float
        #image = tf.image.resize(image, self.input_shape)

        return image, label

    #TODO: implement image augmentation methods here, i.e rotation, flipping, scaling, translation, brightness,
    #------------------------------------------------------------------------------ VALIDATION DATA SPLITTING --
    def get_train_val_split(self, data, val_fraction, shuffle, seed):
```

```
        """
        Split the training dataset into a training and validation dataset.

        args:
        data (tf.data.Dataset): the input dataset to be split
        val_fraction (float): the fraction of the dataset to be used for validation, range: [0, 1]
        shuffle (bool): shuffle the dataset before splitting
        seed (int): the seed for the random number generator

        returns:
        (tuple): of training and validation datasets
            train_split (tf.data.Dataset): the training dataset
            validation_split (tf.data.Dataset): the validation dataset
        """
        if val_fraction < 0 or val_fraction > 1:
            raise ValueError("The fraction of the dataset to use must be in the range [0, 1]")
        else:
            if shuffle:
                data = data.shuffle(buffer_size = len(data), seed = seed)

            train_size = int(len(data) * (1 - val_fraction))

        return data.take(train_size), data.skip(train_size)

    def get_kfolds(self):
        pass
```

26/03/2024 - Malachy -

In [ ]:

```
"""
network.py - 26/03/2024
"""

from packages.imports import os, datetime, tf, keras, MaxPooling2D, MaxPooling3D, Conv2D, Conv3D, Sequential, F
from dataframe.keys import get_keys

class ConvolutionalNeuralNetwork():
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, representation, input_shape, i
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        """

        self.display_summary = display_summary
        self.architecture = architecture
        self.learning_rate = learning_rate
        self.weight_reg = weight_reg

        self.representation = representation
        self.label_length = len(get_keys(representation))

        self.input_x, self.input_y = input_shape
        self.input_dim = input_dim

        self.network = self.__init_network(display_summary, architecture)

        self.train_history = None
        self.test_history = None
    #------------------------------------------------------------------------------ NETWORK INITIALIZATION ---------
    def __init_network(self, display_summary: bool = False, architecture: str = "216x64") -> tf.keras.Model:
        """
        Initalise a Convolutional Neural Network of a specified architecture using the Keras Sequential API.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        """
        lr = self.learning_rate

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)
```

```
    if architecture == '128x64xn':

        network = Sequential()
        network.add(Conv2D(filters = 32, kernel_size = (7, 7), strides = 3, activation = 'relu', kernel_regu
        network.add(MaxPooling2D(pool_size = (2,2)))
        network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
        network.add(MaxPooling2D(pool_size = (2,2)))
        network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_regu
        network.add(Flatten())
        network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
        network.add(Dropout(0.5))
        network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
        network.add(Dropout(0.5))
        network.add(Dense(self.label_length, activation = 'softmax', kernel_regularizer = wr))

        network.compile(loss = 'categorical_crossentropy', optimizer = tf.keras.optimizers.Adam(learning_ra
        # TRY CROSS ENTROPY LOSS
    elif architecture == '400xn':

        pass

    elif architecture == 'ResNet50':

        network = Sequential()

        pretrained_model = ResNet50(include_top=False, input_shape = (self.input_x, self.input_y, self.inpu

        for layer in pretrained_model.layers:
            layer.trainable = True

        network.add(pretrained_model)
        network.add(Flatten())
        network.add(Dense(512, activation = 'relu'))
        network.add(Dense(self.label_length, activation = 'softmax'))

        network.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

    else:
        raise ValueError(f"Invalid architecture parameter provided: {architecture}")

    if (display_summary):
        network.summary()

    return network
#------------------------------------------------------------------------------ NETWORK TRAINING AND TESTING ---
def train(self, train_data, validation_data, dataframe, train_batch_size, val_batch_size, val_steps, epochs
    """
    Train the network on the training data using model.fit().

    args:
    train_data (tf.data.Dataset): the training dataset
    validation_data (tf.data.Dataset): the validation dataset
    dataframe (DataFrame): An instance of DataFrame containing the representation of the training and test
    train_batch_size (int): the batch size of the training data
    val_batch_size (int): the batch size of the validation data
    val_steps (int): the number of steps to run the validation data
    epochs (int): the number of epochs to train the network
    initial_epoch (int): the initial epoch to start training the network from
    """
    train_data = train_data.map(
        lambda image, label: dataframe.fetch_image_label_pair(image, label),
        num_parallel_calls = tf.data.experimental.AUTOTUNE # allow for tensorflow to dynamically decide the
    ).batch(
        train_batch_size # batch the dataset into batches of size batch_size
    ).prefetch(
        tf.data.experimental.AUTOTUNE # allow for prefetching to parallelize the loading of the images and
    )

    validation_data = validation_data.map(
        lambda image, label: dataframe.fetch_image_label_pair(image, label),
        num_parallel_calls = tf.data.experimental.AUTOTUNE
    ).batch(
        val_batch_size
    ).prefetch(
        tf.data.experimental.AUTOTUNE
    )

    self.train_history = self.network.fit(
        train_data,
        epochs = epochs,
        verbose = 1,
        callbacks = [self.__checkpoint_callback(checkpointfolder)],
```

```
            validation_data = validation_data,
            shuffle = True,
            initial_epoch = initial_epoch,
            steps_per_epoch = None,
            validation_steps = val_steps,
            validation_batch_size = val_batch_size,
            validation_freq = 1,
            # max_queue_size = 10,
            # workers = 1,
            # use_multiprocessing = True
        )

    def test(self, test_data, dataframe, test_batch_size, test_steps):
        """
        Evaluate the network on the test data using model.evaluate().

        args:
        test_data (tf.data.Dataset): the test dataset
        dataframe (DataFrame): An instance of DataFrame containing the representation of the training and test d
        test_batch_size (int): the batch size of the test data
        test_steps (int): the number of steps to run the test data
        """

        test_data = test_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE
        ).batch(
            test_batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE
        )

        self.test_history = self.network.evaluate(
            test_data,
            batch_size = test_batch_size,
            verbose = 1,
            steps = test_steps,
            callbacks = None,
            # max_queue_size = 10,
            # workers = 1,
            # use_multiprocessing = True,
            return_dict = True
        )

    #------------------------------------------------------------------- NETWORK WEIGHTS AND BIASES SAVING AND LOA
    def save(self, checkpoint_path):
        """
        Save the network weights and biases to a file.

        args:
        checkpoint_path (str): the path to the directory to save the network weights and biases
        """
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.save_weights(path)


    def load(self, checkpoint_path, timestamp):
        """
        Load the network weights and biases from a file.

        args:
        checkpoint_path (str): the path to the directory to load the network weights and biases
        """
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.load_weights(path)

    def __checkpoint_callback(self, checkpointfolder):
        """
        A ModelCheckpoint callback to save the best model during training.

        args:
        checkpointfolder (str): the path to the directory to save the model checkpoint

        returns:
        tf.keras.callbacks.ModelCheckpoint: a ModelCheckpoint callback
        """

        return tf.keras.callbacks.ModelCheckpoint(
            filepath = os.path.join(checkpointfolder, "model-checkpoint-" + datetime.datetime.now().strftime("%
            save_weights_only = False,
            save_freq = 'epoch',
            monitor = 'val_loss',
```

```
            mode = 'min',
            save_best_only = True,
            verbose = 1
        )
```

26/03/2024 - Malachy -

In [ ]:

```
"""
main.py - 26/03/2024
"""

from packages.imports import argparse, os
from dataframe.dataframe import DataFrame
from network.network import ConvolutionalNeuralNetwork

def main(root, dataset_root, checkpointroot, train_catalog, test_catalog, classes, one_hot, ds_fraction, input_s

    if not os.path.exists(os.path.join(root, checkpointroot)):
        os.makedirs(os.path.join(root, checkpointroot))

    dataset_root = os.path.join(root, dataset_root)
    checkpoint_path = os.path.join(root, checkpointroot)

    df = DataFrame(dataset_root, train_catalog, test_catalog, classes, one_hot, input_shape, input_dim, seed, ds
    CNN = ConvolutionalNeuralNetwork(display_summary, architecture, learning_rate, weight_reg, classes, input_sh

    train_split, val_split = df.get_train_val_split(df.train_dataset, val_fraction = 0.2, shuffle = True, seed =
    CNN.train(train_data = train_split, validation_data = val_split, dataframe = df, train_batch_size = batch_si

    #TODO: implement loading of the network weights and biases
    # implement kfold cross validation with tf.datasets or sklearn
    # implement training and testing metrics, i.e accuracy, precision, recall, f1-score, confusion matrix
    # implement plotting of the training and testing metrics

if __name__ == "__main__":
    """run `python network.py --help` for argument information"""
    # ensure you are in the correct directory where py file is located, can change directory by running `cd "<di
    # i.e run `cd "C:/Users/Ma1a_/Desktop/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine

    parser = argparse.ArgumentParser(prog = "network.py")

    parser.add_argument("-dim",       "--inputdim", type = int,   default = 3,          help = "(int) The input
    parser.add_argument("-ixy",     "--inputshape", type = tuple, default = (424, 424), help = "(tuple) The inpu

    parser.add_argument("-r",             "--root", type = str,   default = r"C:/Users/Ma1a_/Desktop/3rd Year P
    parser.add_argument("-dsr",    "--datasetroot", type = str,   default = r"galaxyzoo2-dataset/",       help =
    parser.add_argument("-l",          "--labels", type = str,   default = r"gz2_train_catalog.parquet", help =
    parser.add_argument("-tl",      "--testlabels", type = str,   default = r"gz2_test_catalog.parquet",  help =
    parser.add_argument("-cpr", "--checkpointroot", type = str,   default = r"checkpoints/", help = "(str) The p
    parser.add_argument("-c",         "--classes", type = str,   default = "Q1",       help = "(str) The classe
    parser.add_argument("-oh",         "--onehot", type = bool,  default = False,      help = "(bool) If to one
    parser.add_argument("-dsf",     "--dsfraction", type = float, default = 0.5,        help = "(float) The fra

    parser.add_argument("-sum",        "--summary", type = bool,  default = True,       help = "(bool) Display
    parser.add_argument("-a",     "--architecture", type = str,   default = "128x64xn", help = "(str) The archi
    parser.add_argument("-lr",    "--learningrate", type = float, default = 0.001,      help = "(float) The lea
    parser.add_argument("-wr",       "--weightreg", type = str,   default = None,       help = "(str) The weight
    parser.add_argument("-bs",       "--batchsize", type = int,   default = 32,         help = "(int) The batch
    parser.add_argument("-e",          "--epochs", type = int,   default = 500,        help = "(int) The numbe
    parser.add_argument("-s",            "--seed", type = int,   default = 0,          help = "(int) The seed

    args = parser.parse_args()

    main(
        input_dim = args.inputdim,
        input_shape = args.inputshape,
        root = args.root,
        dataset_root = args.datasetroot,
        train_catalog = args.labels,
        test_catalog = args.testlabels,
        checkpointroot = args.checkpointroot,
        classes = args.classes,
        one_hot = args.onehot,
        ds_fraction = args.dsfraction,
        display_summary = args.summary,
        architecture = args.architecture,
        learning_rate = args.learningrate,
        weight_reg = args.weightreg,
        batch_size = args.batchsize,
        epochs = args.epochs,
        seed = args.seed
    )
```

Aroushi -

Week Starting March 18th:

Thursday Febuary 1st 10am-11am: SUPERVISOR MEETING

- Discussed results obtained so far and areas of improvement
- Adam suggested some successful network architectures to try out
- Adam also helped us debug code to solve Malachy's error with exploding loss function gradients

26/03/2024 - Malachy -

Trained for 75 epochs, 20% val split, 32 batch size, 0.001 learning rate, 128x64xn architecture, 3 input dimension, vote-fraction representation, 424x424 input shape, no weight regularization Used keys: 'smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction', 'smooth-or-featured-gz2_artifact_fraction',

(first question in decision tree)

output:

In [ ]:

```
"""
PS C:\Users\Ma1a_\Desktop\3rd Year Project> & C:/Users/Ma1a_/AppData/Local/Programs/Python/Python310/python.exe
2024-03-26 17:52:32.913764: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optir
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-03-26 17:52:33.463858: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device /job:localh
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Convolutional_1 (Conv2D)    (None, 140, 140, 32)      4736

 max_pooling2d (MaxPooling2D  (None, 70, 70, 32)       0
 )

 Convolutional_2 (Conv2D)    (None, 68, 68, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 34, 34, 64)       0
 2D)

 Convolutional_3 (Conv2D)    (None, 32, 32, 64)        36928

 flatten (Flatten)           (None, 65536)             0

 Dense_1 (Dense)             (None, 128)               8388736

 dropout (Dropout)           (None, 128)               0

 Dense_2 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense (Dense)               (None, 3)                 195

=================================================================
Total params: 8,457,347
Trainable params: 8,457,347
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
2024-03-26 17:52:47.244629: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8100
2093/2093 [==============================] - ETA: 0s - loss: 0.6844 - accuracy: 0.8290
Epoch 1: val_loss improved from inf to 0.59793, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Project
2093/2093 [==============================] - 60s 27ms/step - loss: 0.6844 - accuracy: 0.8290 - val_loss: 0.5979
Epoch 2/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.6343 - accuracy: 0.8776
Epoch 2: val_loss did not improve from 0.59793
2093/2093 [==============================] - 57s 27ms/step - loss: 0.6343 - accuracy: 0.8776 - val_loss: 0.6510
Epoch 3/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.6203 - accuracy: 0.8929
Epoch 3: val_loss did not improve from 0.59793
2093/2093 [==============================] - 55s 26ms/step - loss: 0.6203 - accuracy: 0.8930 - val_loss: 0.6620
Epoch 4/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.6142 - accuracy: 0.9018
Epoch 4: val_loss improved from 0.59793 to 0.54981, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Pro
2093/2093 [==============================] - 59s 28ms/step - loss: 0.6142 - accuracy: 0.9018 - val_loss: 0.5498
Epoch 5/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.6079 - accuracy: 0.9083
Epoch 5: val_loss did not improve from 0.54981
2093/2093 [==============================] - 56s 27ms/step - loss: 0.6079 - accuracy: 0.9083 - val_loss: 0.5513
Epoch 6/500
```

```
2093/2093 [==============================] - ETA: 0s - loss: 0.6045 - accuracy: 0.9138
Epoch 6: val_loss did not improve from 0.54981
2093/2093 [==============================] - 57s 27ms/step - loss: 0.6045 - accuracy: 0.9138 - val_loss: 0.6043
Epoch 7/500
2093/2093 [==============================] - ETA: 0s - loss: 0.6010 - accuracy: 0.9173
Epoch 7: val_loss improved from 0.54981 to 0.50042, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Pro
2093/2093 [==============================] - 58s 28ms/step - loss: 0.6010 - accuracy: 0.9173 - val_loss: 0.5004
Epoch 8/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5989 - accuracy: 0.9206
Epoch 8: val_loss did not improve from 0.50042
2093/2093 [==============================] - 58s 28ms/step - loss: 0.5989 - accuracy: 0.9206 - val_loss: 0.5894
Epoch 9/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5967 - accuracy: 0.9245
Epoch 9: val_loss did not improve from 0.50042
2093/2093 [==============================] - 60s 28ms/step - loss: 0.5968 - accuracy: 0.9245 - val_loss: 0.6108
Epoch 10/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5950 - accuracy: 0.9253
Epoch 10: val_loss did not improve from 0.50042
2093/2093 [==============================] - 63s 30ms/step - loss: 0.5950 - accuracy: 0.9253 - val_loss: 0.5850
Epoch 11/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5936 - accuracy: 0.9290
Epoch 11: val_loss did not improve from 0.50042
2093/2093 [==============================] - 64s 30ms/step - loss: 0.5936 - accuracy: 0.9290 - val_loss: 0.5309
Epoch 12/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5913 - accuracy: 0.9312
Epoch 12: val_loss did not improve from 0.50042
2093/2093 [==============================] - 63s 30ms/step - loss: 0.5913 - accuracy: 0.9312 - val_loss: 0.5724
Epoch 13/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5899 - accuracy: 0.9330
Epoch 13: val_loss did not improve from 0.50042
2093/2093 [==============================] - 62s 30ms/step - loss: 0.5900 - accuracy: 0.9329 - val_loss: 0.5684
Epoch 14/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5887 - accuracy: 0.9352
Epoch 14: val_loss did not improve from 0.50042
2093/2093 [==============================] - 62s 30ms/step - loss: 0.5887 - accuracy: 0.9352 - val_loss: 0.5553
Epoch 15/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5876 - accuracy: 0.9347
Epoch 15: val_loss did not improve from 0.50042
2093/2093 [==============================] - 62s 29ms/step - loss: 0.5876 - accuracy: 0.9348 - val_loss: 0.5179
Epoch 16/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5871 - accuracy: 0.9353
Epoch 16: val_loss did not improve from 0.50042
2093/2093 [==============================] - 62s 30ms/step - loss: 0.5871 - accuracy: 0.9353 - val_loss: 0.5757
Epoch 17/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5861 - accuracy: 0.9377
Epoch 17: val_loss did not improve from 0.50042
2093/2093 [==============================] - 62s 30ms/step - loss: 0.5861 - accuracy: 0.9377 - val_loss: 0.5635
Epoch 18/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5854 - accuracy: 0.9380
Epoch 18: val_loss did not improve from 0.50042
2093/2093 [==============================] - 61s 29ms/step - loss: 0.5854 - accuracy: 0.9380 - val_loss: 0.5957
Epoch 19/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5851 - accuracy: 0.9388
Epoch 19: val_loss did not improve from 0.50042
2093/2093 [==============================] - 60s 29ms/step - loss: 0.5851 - accuracy: 0.9388 - val_loss: 0.6050
Epoch 20/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5847 - accuracy: 0.9376
Epoch 20: val_loss did not improve from 0.50042
2093/2093 [==============================] - 60s 29ms/step - loss: 0.5847 - accuracy: 0.9376 - val_loss: 0.6469
Epoch 21/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5842 - accuracy: 0.9401
Epoch 21: val_loss did not improve from 0.50042
2093/2093 [==============================] - 60s 28ms/step - loss: 0.5842 - accuracy: 0.9401 - val_loss: 0.6039
Epoch 22/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5835 - accuracy: 0.9414
Epoch 22: val_loss did not improve from 0.50042
2093/2093 [==============================] - 63s 30ms/step - loss: 0.5835 - accuracy: 0.9414 - val_loss: 0.5377
Epoch 23/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5822 - accuracy: 0.9415
Epoch 23: val_loss did not improve from 0.50042
2093/2093 [==============================] - 64s 30ms/step - loss: 0.5822 - accuracy: 0.9415 - val_loss: 0.5087
Epoch 24/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5813 - accuracy: 0.9420
Epoch 24: val_loss did not improve from 0.50042
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5813 - accuracy: 0.9420 - val_loss: 0.5586
Epoch 25/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5810 - accuracy: 0.9419
Epoch 25: val_loss did not improve from 0.50042
2093/2093 [==============================] - 58s 27ms/step - loss: 0.5810 - accuracy: 0.9419 - val_loss: 0.5870
Epoch 26/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5811 - accuracy: 0.9430
Epoch 26: val_loss did not improve from 0.50042
2093/2093 [==============================] - 60s 28ms/step - loss: 0.5811 - accuracy: 0.9429 - val_loss: 0.5825
```

```
Epoch 27/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5802 - accuracy: 0.9430
Epoch 27: val_loss improved from 0.50042 to 0.49558, saving model to C:/Users/Ma1a_/Desktop/3rd Year Project/Pr
2093/2093 [==============================] - 63s 30ms/step - loss: 0.5802 - accuracy: 0.9430 - val_loss: 0.4956
Epoch 28/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5808 - accuracy: 0.9437
Epoch 28: val_loss did not improve from 0.49558
2093/2093 [==============================] - 59s 28ms/step - loss: 0.5809 - accuracy: 0.9438 - val_loss: 0.5429
Epoch 29/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5801 - accuracy: 0.9445
Epoch 29: val_loss did not improve from 0.49558
2093/2093 [==============================] - 59s 28ms/step - loss: 0.5801 - accuracy: 0.9445 - val_loss: 0.6043
Epoch 30/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5794 - accuracy: 0.9454
Epoch 30: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5794 - accuracy: 0.9454 - val_loss: 0.5670
Epoch 31/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5795 - accuracy: 0.9440
Epoch 31: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5795 - accuracy: 0.9440 - val_loss: 0.5460
Epoch 32/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5782 - accuracy: 0.9445
Epoch 32: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5782 - accuracy: 0.9445 - val_loss: 0.5701
Epoch 33/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5786 - accuracy: 0.9458
Epoch 33: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5786 - accuracy: 0.9458 - val_loss: 0.6732
Epoch 34/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5780 - accuracy: 0.9461
Epoch 34: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5780 - accuracy: 0.9461 - val_loss: 0.5661
Epoch 35/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5780 - accuracy: 0.9444
Epoch 35: val_loss did not improve from 0.49558
2093/2093 [==============================] - 72s 34ms/step - loss: 0.5780 - accuracy: 0.9445 - val_loss: 0.5659
Epoch 36/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5770 - accuracy: 0.9449
Epoch 36: val_loss did not improve from 0.49558
2093/2093 [==============================] - 58s 27ms/step - loss: 0.5770 - accuracy: 0.9449 - val_loss: 0.5969
Epoch 37/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5776 - accuracy: 0.9458
Epoch 37: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5776 - accuracy: 0.9458 - val_loss: 0.5058
Epoch 38/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5773 - accuracy: 0.9454
Epoch 38: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5773 - accuracy: 0.9454 - val_loss: 0.6048
Epoch 39/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5769 - accuracy: 0.9471
Epoch 39: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5769 - accuracy: 0.9471 - val_loss: 0.5573
Epoch 40/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5760 - accuracy: 0.9467
Epoch 40: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5760 - accuracy: 0.9467 - val_loss: 0.5073
Epoch 41/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5762 - accuracy: 0.9472
Epoch 41: val_loss did not improve from 0.49558
2093/2093 [==============================] - 58s 28ms/step - loss: 0.5762 - accuracy: 0.9472 - val_loss: 0.6121
Epoch 42/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5767 - accuracy: 0.9460
Epoch 42: val_loss did not improve from 0.49558
2093/2093 [==============================] - 58s 27ms/step - loss: 0.5767 - accuracy: 0.9460 - val_loss: 0.6336
Epoch 43/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5757 - accuracy: 0.9474
Epoch 43: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5758 - accuracy: 0.9474 - val_loss: 0.5659
Epoch 44/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5760 - accuracy: 0.9481
Epoch 44: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5760 - accuracy: 0.9481 - val_loss: 0.6135
Epoch 45/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5761 - accuracy: 0.9491
Epoch 45: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5761 - accuracy: 0.9491 - val_loss: 0.5865
Epoch 46/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5755 - accuracy: 0.9475
Epoch 46: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5755 - accuracy: 0.9475 - val_loss: 0.5311
Epoch 47/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5751 - accuracy: 0.9475
Epoch 47: val_loss did not improve from 0.49558
```

```
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5751 - accuracy: 0.9475 - val_loss: 0.5840
Epoch 48/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5752 - accuracy: 0.9489
Epoch 48: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5752 - accuracy: 0.9489 - val_loss: 0.5302
Epoch 49/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5746 - accuracy: 0.9486
Epoch 49: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5746 - accuracy: 0.9486 - val_loss: 0.5163
Epoch 50/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5748 - accuracy: 0.9476
Epoch 50: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5749 - accuracy: 0.9476 - val_loss: 0.6176
Epoch 51/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5745 - accuracy: 0.9481
Epoch 51: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5745 - accuracy: 0.9481 - val_loss: 0.5083
Epoch 52/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5737 - accuracy: 0.9484
Epoch 52: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5737 - accuracy: 0.9484 - val_loss: 0.5986
Epoch 53/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5745 - accuracy: 0.9469
Epoch 53: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5745 - accuracy: 0.9469 - val_loss: 0.5899
Epoch 54/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5739 - accuracy: 0.9480
Epoch 54: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5739 - accuracy: 0.9480 - val_loss: 0.5440
Epoch 55/500
2091/2093 [============================>.] - ETA: 0s - loss: 0.5738 - accuracy: 0.9491
Epoch 55: val_loss did not improve from 0.49558
2093/2093 [==============================] - 57s 27ms/step - loss: 0.5738 - accuracy: 0.9490 - val_loss: 0.5072
Epoch 56/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5741 - accuracy: 0.9490
Epoch 56: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5741 - accuracy: 0.9490 - val_loss: 0.5391
Epoch 57/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5733 - accuracy: 0.9500
Epoch 57: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5733 - accuracy: 0.9500 - val_loss: 0.5576
Epoch 58/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5734 - accuracy: 0.9505
Epoch 58: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5734 - accuracy: 0.9505 - val_loss: 0.5337
Epoch 59/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5742 - accuracy: 0.9489
Epoch 59: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5742 - accuracy: 0.9489 - val_loss: 0.5577
Epoch 60/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5737 - accuracy: 0.9500
Epoch 60: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5737 - accuracy: 0.9500 - val_loss: 0.5684
Epoch 61/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5734 - accuracy: 0.9492
Epoch 61: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5734 - accuracy: 0.9492 - val_loss: 0.5480
Epoch 62/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5731 - accuracy: 0.9495
Epoch 62: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5731 - accuracy: 0.9495 - val_loss: 0.6172
Epoch 63/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5728 - accuracy: 0.9502
Epoch 63: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5728 - accuracy: 0.9502 - val_loss: 0.5477
Epoch 64/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5723 - accuracy: 0.9487
Epoch 64: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5723 - accuracy: 0.9487 - val_loss: 0.5474
Epoch 65/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5723 - accuracy: 0.9505
Epoch 65: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5723 - accuracy: 0.9505 - val_loss: 0.5268
Epoch 66/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5732 - accuracy: 0.9500
Epoch 66: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5732 - accuracy: 0.9500 - val_loss: 0.5385
Epoch 67/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5723 - accuracy: 0.9497
Epoch 67: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5723 - accuracy: 0.9497 - val_loss: 0.5370
Epoch 68/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5722 - accuracy: 0.9500
```

```
Epoch 68: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5722 - accuracy: 0.9500 - val_loss: 0.5685
Epoch 69/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5719 - accuracy: 0.9515
Epoch 69: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5718 - accuracy: 0.9515 - val_loss: 0.5527
Epoch 70/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5720 - accuracy: 0.9510
Epoch 70: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5720 - accuracy: 0.9510 - val_loss: 0.5720
Epoch 71/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5721 - accuracy: 0.9519
Epoch 71: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5721 - accuracy: 0.9519 - val_loss: 0.5641
Epoch 72/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5725 - accuracy: 0.9490
Epoch 72: val_loss did not improve from 0.49558
2093/2093 [==============================] - 56s 27ms/step - loss: 0.5725 - accuracy: 0.9490 - val_loss: 0.5603
Epoch 73/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5714 - accuracy: 0.9509
Epoch 73: val_loss did not improve from 0.49558
2093/2093 [==============================] - 61s 29ms/step - loss: 0.5714 - accuracy: 0.9509 - val_loss: 0.5978
Epoch 74/500
2093/2093 [==============================] - ETA: 0s - loss: 0.5716 - accuracy: 0.9507
Epoch 74: val_loss did not improve from 0.49558
2093/2093 [==============================] - 58s 28ms/step - loss: 0.5716 - accuracy: 0.9507 - val_loss: 0.5538
Epoch 75/500
2092/2093 [============================>.] - ETA: 0s - loss: 0.5717 - accuracy: 0.9510
Epoch 75: val_loss did not improve from 0.49558
2093/2093 [==============================] - 59s 28ms/step - loss: 0.5717 - accuracy: 0.9510 - val_loss: 0.5444
"""
```

27/03/2024 Malachy -

need to implement:

- loading of saved network
- early stopping
- saving network weights once training has finished
- displaying of metrics in tensorboard or with matplotlib
- kfold-cross validation using sklearn

In [ ]:

```
"""
PS C:\Users\Ma1a_\Desktop\3rd Year Project> & C:/Users/Ma1a_/AppData/Local/Programs/Python/Python310/python.exe
2024-03-26 19:43:41.523633: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optir
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-03-26 19:43:42.074554: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device /job:localh
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Convolutional_1 (Conv2D)    (None, 140, 140, 32)      4736

 max_pooling2d (MaxPooling2D  (None, 70, 70, 32)       0
 )

 Convolutional_2 (Conv2D)    (None, 68, 68, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 34, 34, 64)       0
 2D)

 Convolutional_3 (Conv2D)    (None, 32, 32, 64)        36928

 flatten (Flatten)           (None, 65536)             0

 Dense_1 (Dense)             (None, 128)               8388736

 dropout (Dropout)           (None, 128)               0

 Dense_2 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense (Dense)               (None, 30)                1950

=================================================================
Total params: 8,459,102
Trainable params: 8,459,102
Non-trainable params: 0
_________________________________________________________________
Epoch 1/500
```

```
2024-03-26 19:43:55.782316: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8100
419/419 [==============================] - ETA: 0s - loss: 13394953371648.0000 - accuracy: 0.1460
Epoch 1: val_loss improved from inf to 61264613081088.00000, saving model to C:/Users/Ma1a_/Desktop/3rd Year Pr
419/419 [==============================] - 15s 28ms/step - loss: 13394953371648.0000 - accuracy: 0.1460 - val_l
Epoch 2/500
417/419 [============================>.] - ETA: 0s - loss: 1885763507585024.0000 - accuracy: 0.1710
Epoch 2: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 1901604554932224.0000 - accuracy: 0.1713 - val_
Epoch 3/500
418/419 [============================>.] - ETA: 0s - loss: 26762351390752768.0000 - accuracy: 0.1662
Epoch 3: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 26787747532374016.0000 - accuracy: 0.1661 - va
Epoch 4/500
416/419 [============================>.] - ETA: 0s - loss: 1504299410522112000.0000 - accuracy: 0.1716
Epoch 4: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 1510647887581675520.0000 - accuracy: 0.1716 - va
Epoch 5/500
418/419 [============================>.] - ETA: 0s - loss: 5331460414830018560.0000 - accuracy: 0.1647
Epoch 5: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 5335916529298964480.0000 - accuracy: 0.1646 - va
Epoch 6/500
418/419 [============================>.] - ETA: 0s - loss: 1444021743546007552.0000 - accuracy: 0.1695
Epoch 6: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 1445483956571996160.0000 - accuracy: 0.1694 - v
Epoch 7/500
416/419 [============================>.] - ETA: 0s - loss: 3238939978713006080.0000 - accuracy: 0.1720
Epoch 7: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 3245738259107545088.0000 - accuracy: 0.1717 - v
Epoch 8/500
417/419 [============================>.] - ETA: 0s - loss: 6346505910092824576.0000 - accuracy: 0.1687
Epoch 8: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 6356006240312623104.0000 - accuracy: 0.1688 - v
Epoch 9/500
419/419 [==============================] - ETA: 0s - loss: 11274639621331353600.0000 - accuracy: 0.1703
Epoch 9: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 11274639621331353600.0000 - accuracy: 0.1703 -
Epoch 10/500
418/419 [============================>.] - ETA: 0s - loss: 18903872630027190272.0000 - accuracy: 0.1678
Epoch 10: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 18904367410259689472.0000 - accuracy: 0.1677 -
Epoch 11/500
418/419 [============================>.] - ETA: 0s - loss: 29680854596929454080.0000 - accuracy: 0.1663
Epoch 11: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 29697369261578649600.0000 - accuracy: 0.1663 -
Epoch 12/500
418/419 [============================>.] - ETA: 0s - loss: 44594309258687807488.0000 - accuracy: 0.1685
Epoch 12: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 44614878922220240896.0000 - accuracy: 0.1685 -
Epoch 13/500
418/419 [============================>.] - ETA: 0s - loss: 64916912528359948288.0000 - accuracy: 0.1696
Epoch 13: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 64921816350219829248.0000 - accuracy: 0.1698 -
Epoch 14/500
418/419 [============================>.] - ETA: 0s - loss: 90862629368711086080.0000 - accuracy: 0.1650
Epoch 14: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 90863051581176152064.0000 - accuracy: 0.1650 -
Epoch 15/500
418/419 [============================>.] - ETA: 0s - loss: 125591003199178801152.0000 - accuracy: 0.1669
Epoch 15: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 125589059262620893184.0000 - accuracy: 0.1669
Epoch 16/500
418/419 [============================>.] - ETA: 0s - loss: 167267595625842081792.0000 - accuracy: 0.1683
Epoch 16: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 167389104854850863104.0000 - accuracy: 0.1686
Epoch 17/500
418/419 [============================>.] - ETA: 0s - loss: 220263686648738349056.0000 - accuracy: 0.1719
Epoch 17: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 12s 28ms/step - loss: 220305151431245037568.0000 - accuracy: 0.1718
Epoch 18/500
416/419 [============================>.] - ETA: 0s - loss: 284267736245207040000.0000 - accuracy: 0.1665
Epoch 18: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 284613106041631014912.0000 - accuracy: 0.1669
Epoch 19/500
416/419 [============================>.] - ETA: 0s - loss: 364103785711417688064.0000 - accuracy: 0.1667
Epoch 19: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 364299446004603682816.0000 - accuracy: 0.1665
Epoch 20/500
418/419 [============================>.] - ETA: 0s - loss: 459777306217230106624.0000 - accuracy: 0.1661
Epoch 20: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 12s 27ms/step - loss: 460126229635235053568.0000 - accuracy: 0.1661
Epoch 21/500
417/419 [============================>.] - ETA: 0s - loss: 572899460529336090624.0000 - accuracy: 0.1708
Epoch 21: val_loss did not improve from 61264613081088.00000
```

```
419/419 [==============================] - 12s 27ms/step - loss: 573077810111454380032.0000 - accuracy: 0.1708
Epoch 22/500
417/419 [============================>.] - ETA: 0s - loss: 707578200010967220224.0000 - accuracy: 0.1658
Epoch 22: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: 708141501808109420544.0000 - accuracy: 0.1659
Epoch 23/500
418/419 [============================>.] - ETA: 0s - loss: 867701714437288230912.0000 - accuracy: 0.1641
Epoch 23: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 26ms/step - loss: 867645208335713566720.0000 - accuracy: 0.1642
Epoch 24/500
419/419 [==============================] - ETA: 0s - loss: 1047811852166158614528.0000 - accuracy: 0.1686
Epoch 24: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 12s 28ms/step - loss: 1047811852166158614528.0000 - accuracy: 0.1686
Epoch 25/500
419/419 [==============================] - ETA: 0s - loss: 1266519316445221879808.0000 - accuracy: 0.1713
Epoch 25: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 12s 28ms/step - loss: 1266519316445221879808.0000 - accuracy: 0.1713
Epoch 26/500
419/419 [==============================] - ETA: 0s - loss: 1511639548113882972160.0000 - accuracy: 0.1656
Epoch 26: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 13s 32ms/step - loss: 1511639548113882972160.0000 - accuracy: 0.1656
Epoch 27/500
418/419 [============================>.] - ETA: 0s - loss: 1794668829645935214592.0000 - accuracy: 0.1627
Epoch 27: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 13s 32ms/step - loss: 1794539773369113378816.0000 - accuracy: 0.1628
Epoch 28/500
419/419 [==============================] - ETA: 0s - loss: 2131412403196147007488.0000 - accuracy: 0.1710
Epoch 28: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 13s 31ms/step - loss: 2131412403196147007488.0000 - accuracy: 0.1710
Epoch 29/500
418/419 [============================>.] - ETA: 0s - loss: 2468068860241066852352.0000 - accuracy: 0.1658
Epoch 29: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 13s 31ms/step - loss: 2468033675868978020352.0000 - accuracy: 0.1658
Epoch 30/500
417/419 [============================>.] - ETA: 0s - loss: 2892569435392731774976.0000 - accuracy: 0.1733
Epoch 30: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 13s 30ms/step - loss: 2892813192722563203072.0000 - accuracy: 0.1732
Epoch 31/500
417/419 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.0114
Epoch 31: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 11s 27ms/step - loss: nan - accuracy: 0.0113 - val_loss: nan - val_a
Epoch 32/500
418/419 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.0062
Epoch 32: val_loss did not improve from 61264613081088.00000
419/419 [==============================] - 12s 28ms/step - loss: nan - accuracy: 0.0062 - val_loss: nan - val_a
"""
```

31/03/2024: - Malachy:

implemented label binarisation: labels for features are binarised by splitting label vectors into the constituent questions, Q0,Q1,Q2... the maximum value for a feature per question is taken to be 1 the rest are zero. i.e. for Q1 could have 0.1,0.1,0.8 -> 0,0,1 so full label vector then beomes -> 0,0,1,0,1,1,0,...,0,0,1,0,0 ect encoded as a binary bit string.

ran this for 10 epochs with binary_crosentropy loss with the following 9 labels...

- Q0/ Smooth or Featured? 'smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction', 'smooth-or-featured-gz2_artifact_fraction',
- Q1/ Disk Edge On? 'disk-edge-on-gz2_yes_fraction', 'disk-edge-on-gz2_no_fraction',
- Q2/ Has spiral arms? 'has-spiral-arms-gz2_yes_fraction', 'has-spiral-arms-gz2_no_fraction',
- Q3/ Bar? 'bar-gz2_yes_fraction', 'bar-gz2_no_fraction',

achived 56% accuracy with 128x64xn architecture.

In [ ]:

```
"""
Model: "sequential"
```

Layer (type)	Output Shape	Param #
Convolutional_1 (Conv2D)	(None, 140, 140, 32)	4,736
max_pooling2d (MaxPooling2D)	(None, 70, 70, 32)	0
Convolutional_2 (Conv2D)	(None, 68, 68, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 34, 34, 64)	0
Convolutional_3 (Conv2D)	(None, 32, 32, 64)	36,928

```
| flatten (Flatten)                    | (None, 65536)               |              0 |
|--------------------------------------|-----------------------------|----------------|
| Dense_1 (Dense)                      | (None, 128)                 |      8,388,736 |
|--------------------------------------|-----------------------------|----------------|
| dropout (Dropout)                    | (None, 128)                 |              0 |
|--------------------------------------|-----------------------------|----------------|
| Dense_2 (Dense)                      | (None, 64)                  |          8,256 |
|--------------------------------------|-----------------------------|----------------|
| dropout_1 (Dropout)                  | (None, 64)                  |              0 |
|--------------------------------------|-----------------------------|----------------|
| dense (Dense)                        | (None, 9)                   |            585 |
 Total params: 8,457,737 (32.26 MB)
 Trainable params: 8,457,737 (32.26 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/500
2024-03-31 19:07:40.632657: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117] Plugi
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.3683 - loss: 0.4275
Epoch 1: val_loss improved from inf to 0.36607, saving model to /Users/malachy/Documents/3rd Year Project/Proje
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 258s 61ms/step - accuracy: 0.3684 - loss: 0.4275 - val_accuracy: 0.6250 - val_los
Epoch 2/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.5325 - loss: 0.3256
Epoch 2: val_loss improved from 0.36607 to 0.33331, saving model to /Users/malachy/Documents/3rd Year Project/P
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 249s 60ms/step - accuracy: 0.5325 - loss: 0.3256 - val_accuracy: 0.6875 - val_los
Epoch 3/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.5506 - loss: 0.3074
Epoch 3: val_loss improved from 0.33331 to 0.24896, saving model to /Users/malachy/Documents/3rd Year Project/P
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 250s 60ms/step - accuracy: 0.5506 - loss: 0.3074 - val_accuracy: 0.5312 - val_los
Epoch 4/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.5549 - loss: 0.2943
Epoch 4: val_loss improved from 0.24896 to 0.24350, saving model to /Users/malachy/Documents/3rd Year Project/P
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 250s 60ms/step - accuracy: 0.5549 - loss: 0.2943 - val_accuracy: 0.6562 - val_los
Epoch 5/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.5652 - loss: 0.2851
Epoch 5: val_loss did not improve from 0.24350
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 247s 59ms/step - accuracy: 0.5652 - loss: 0.2851 - val_accuracy: 0.5938 - val_los
Epoch 6/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 488ms/step - accuracy: 0.5645 - loss: 0.2781
Epoch 6: val_loss improved from 0.24350 to 0.22548, saving model to /Users/malachy/Documents/3rd Year Project/P
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 2042s 488ms/step - accuracy: 0.5645 - loss: 0.2781 - val_accuracy: 0.6875 - val_
Epoch 7/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 386ms/step - accuracy: 0.5692 - loss: 0.2684
Epoch 7: val_loss did not improve from 0.22548
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 1617s 386ms/step - accuracy: 0.5692 - loss: 0.2684 - val_accuracy: 0.5938 - val_
Epoch 8/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.5705 - loss: 0.2625
Epoch 8: val_loss did not improve from 0.22548
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 251s 60ms/step - accuracy: 0.5705 - loss: 0.2625 - val_accuracy: 0.6562 - val_los
Epoch 9/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.5734 - loss: 0.2552
Epoch 9: val_loss improved from 0.22548 to 0.19378, saving model to /Users/malachy/Documents/3rd Year Project/P
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 251s 60ms/step - accuracy: 0.5734 - loss: 0.2552 - val_accuracy: 0.5000 - val_los
Epoch 10/500
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.5708 - loss: 0.2473
Epoch 10: val_loss improved from 0.19378 to 0.16957, saving model to /Users/malachy/Documents/3rd Year Project/
4186/4186 ━━━━━━━━━━━━━━━━━━━━ 255s 61ms/step - accuracy: 0.5708 - loss: 0.2473 - val_accuracy: 0.5625 - val_los
"""
```

31/03/2024 Malachy - current state of dataframe.py and network.py:

In [ ]:

```python
from packages.imports import os, tf, pd, np
from dataframe.keys import get_keys, class_labels_to_question

class DataFrame():
    def __init__(self, dataset_root, train_catalog, test_catalog, classes, binarised_labels, input_shape, input_
        """
        Construct a DataFrame containing the represenation of the training and test data

        args:
        dataset_root (str): path to the dataset root directory
        train_catalog (str): name of the training catalog file
        test_catalog (str): name of the test catalog file
        keys (list): list of the keys to be used when acessing the class labels
        classes (str): the class labels of the dataset to use, i.e smooth-or-featured-gz2_smooth_fraction, spira
        binarised_labels (bool): whether to binarise the labels
        input_shape (tuple): the input shape of the data, x, y
        input_dim (int): the input dimension of the data, i.e 3 for RGB image
        seed (int): the seed for the random number generator
        ds_fraction (float): the fraction of the dataset to use, range: [0, 1]
        """
        # RNG
        self.seed = seed
        self.RNG = np.random.default_rng(seed)
```

```
        tf.random.set_seed(seed)

        # File paths
        self.dataset_root = dataset_root
        self.train_parquet_file_path = os.path.join(self.dataset_root, train_catalog)
        self.test_parquet_file_path = os.path.join(self.dataset_root, test_catalog)

        # Class labels to use
        self.keys = get_keys(classes)
        self.question_dict = class_labels_to_question()
        self.group_sizes = self.__init_group_sizes()

        # Data properties
        self.input_shape = input_shape
        self.input_dim = input_dim

        # Dataset properties
        self.binarised_labels = binarised_labels

        if ds_fraction <= 0 or ds_fraction > 1:
            raise ValueError("The fraction of the dataset to use must be in the range (0, 1]")
        else:
            self.ds_fraction = ds_fraction

        # Initialise the datasets
        self.train_dataset = self.__init_dataset(self.train_parquet_file_path)
        self.test_dataset = self.__init_dataset(self.test_parquet_file_path)

    #------------------------------------------------------------------------ DATASET(S) INITIALIZATION ----------
    def __init_dataset(self, catalog_path):
        """
        Initialise a dataset, containing paths to the image files and the labels.

        args:
        catalog_path (str): the path to the catalog file

        returns:
        dataset (tf.data.Dataset): a zipped dataset containing a tuple of a tf.data.Dataset of image paths and a
        """

        print(f"Initialising dataset [{catalog_path}]...")
        pandas_df = pd.read_parquet(catalog_path)
        pandas_df = pandas_df.fillna(0)

        pandas_df['file_loc'] = np.empty(len(pandas_df['id_str']), dtype = str)

        for idx, _ in enumerate(pandas_df['id_str']):
            pandas_df.loc[idx, 'file_loc'] = os.path.join(self.dataset_root, 'images/', str(pandas_df['subfolde

            if not os.path.exists(pandas_df['file_loc'][idx]):
                pandas_df = pandas_df.drop(idx)

        image_path_ds = tf.data.Dataset.from_tensor_slices(pandas_df['file_loc'])

        labels = pandas_df[self.keys]

        if self.binarised_labels:

            print("Binarising labels... (this may take a while)")
            labels = labels.apply(self.__to_binary, axis = 1, result_type = 'broadcast')
            print("Labels binarised.")

        labels_ds = tf.data.Dataset.from_tensor_slices(labels)

        print("Dataset initialised.")

        return tf.data.Dataset.zip((image_path_ds, labels_ds)).take(int(len(labels_ds) * self.ds_fraction)) # ta

    def __init_group_sizes(self):
        """
        Initialise the group sizes of the class labels to be used for label binarisation.

        returns:
        group_sizes (np.array): the group sizes of the class labels
        """
        values = np.zeros(10, dtype=int) # there are 10 unique questions in the decision tree

        for key in self.keys:
            values[self.question_dict[key]] += 1

        group_sizes = values[values != 0]
        print(f"Group sizes: {group_sizes}")
```

```
        return group_sizes

    def __to_binary(self, label):
        """
        Split each label vector into each of its constituent questions, i.e. Q1, Q2,..
        for each question convert the answer to binary, i.e. 0.1,0.1,0.8 -> 0,0,1

        args:
        label (pd.Series): the label to be binarised

        returns:
        new_label (pd.Series): the binarised label
        """
        new_label = np.zeros(len(label))
        offset = 0

        for size in self.group_sizes:

            group = label[offset:offset + size]
            max_indices = np.flatnonzero(group == np.max(group))
            # takes the first instance of the maximum value for simplicity, should maybe choose randomly from t
            max_idx = max_indices[0]
            new_label[max_idx + offset] = 1

            offset += size

        return pd.Series(new_label)

    #------------------------------------------------------------------------------- IMAGE PROCESSING -------
    def fetch_image_label_pair(self, image_path, label):
        """
        Fetches an image and its corresponding label from the dataset.

        args:
        image_path (tf.Tensor): the path to the image file
        label (tf.Tensor): the label of the image

        returns:
        (tuple): of the image and its corresponding label
            image (tf.Tensor): the requested image
            label (tf.Tensor): the requested label
        """
        image_path = tf.squeeze(image_path)
        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels = self.input_dim)
        image = tf.image.convert_image_dtype(image, tf.float32) # Convert image dtype from tf.uint8 to tf.float
        image = tf.image.resize(image, self.input_shape)

        return image, label

    #TODO: implement image augmentation methods here, i.e rotation, flipping, scaling, translation, brightness,
    #------------------------------------------------------------------------------ VALIDATION DATA SPLITTING --
    def get_train_val_split(self, data, val_fraction, shuffle, seed):
        """
        Split the training dataset into a training and validation dataset.

        args:
        data (tf.data.Dataset): the input dataset to be split
        val_fraction (float): the fraction of the dataset to be used for validation, range: [0, 1]
        shuffle (bool): shuffle the dataset before splitting
        seed (int): the seed for the random number generator

        returns:
        (tuple): of training and validation datasets
            train_split (tf.data.Dataset): the training dataset
            validation_split (tf.data.Dataset): the validation dataset
        """
        if val_fraction < 0 or val_fraction > 1:
            raise ValueError("The fraction of the dataset to use must be in the range [0, 1]")
        else:
            if shuffle:
                data = data.shuffle(buffer_size = len(data), seed = seed)

            train_size = int(len(data) * (1 - val_fraction))

        return data.take(train_size), data.skip(train_size)

    def get_kfolds(self, data, k, shuffle, seed):
        """
        Split the training dataset into k-1 folds for training and 1 fold for validation
        to perform k-fold cross validation.

        args:
```

```
        data (tf.data.Dataset): the input dataset to be split
        k (int): the number of folds
        shuffle (bool): shuffle the dataset before splitting
        seed (int): the seed for the random number generator

        returns:
        """
        if shuffle:
            data = data.shuffle(buffer_size = len(data), seed = seed)

        folds = []

        for split in range(k):
            validation_data = data.skip(split * (len(data) // k)).take(len(data) // k)
            training_data = data.take(split * (len(data) // k)).concatenate(data.skip((split + 1) * (len(data)
            folds.append((training_data, validation_data))

        return folds
```

Aroushi -

Week Starting March 25th:

Wednesday March 27th 10am-11am: SUPERVISOR MEETING

- Reviewed progress so far and compared with initial project plan
- Discussed actions for Easter
- To improve results and advance the project, we discussed on the fly augmentation vs pre processed augmentations

31/03/2024 Malachy -

In [ ]:
```
from packages.imports import os, datetime, tf, keras, MaxPooling2D, MaxPooling3D, Conv2D, Conv3D, Sequential, F
from dataframe.keys import get_keys

class ConvolutionalNeuralNetwork():
    def __init__(self, display_summary, architecture, learning_rate, weight_reg, representation, input_shape, i
        """
        Construct a Convolutional Neural Network of a specified architecture.

        args:
        display_summary (bool): display the summary of the network
        architecture (str): the architecture of the network
        learning_rate (float): the learning rate of the network
        weight_reg (str): the weight regularization of the network
        train_class_labels (list): the class labels of the training data
        test_class_labels (list): the class labels of the test data
        representation (str): the representation of the data, i.e one-hot or vote fraction
        input_dim (int): the input dimension of the data
        loss (str): the loss function to use
        """
        # network parameters
        self.display_summary = display_summary
        self.architecture = architecture
        self.learning_rate = learning_rate
        self.weight_reg = weight_reg
        self.loss = loss

        # data parameters
        self.label_length = len(get_keys(representation))
        self.input_x, self.input_y = input_shape
        self.input_dim = input_dim

        # network
        self.network = self.__init_network()
        self.train_history = None
        self.test_history = None
    #------------------------------------------------------------------------- NETWORK INITIALIZATION ---------
    def __init_network(self) -> tf.keras.Model:
        """
        Initalise a Convolutional Neural Network of a specified architecture using the Keras Sequential API.

        returns:
        network (tf.keras.Model): the Convolutional Neural Network
        """
        lr = self.learning_rate

        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)
```

```
        match self.architecture: # initalise weights and biases to random values
            case '128x64xn':
                network = Sequential()
                network.add(Conv2D(filters = 32, kernel_size = (7, 7), strides = 3, activation = 'relu', kernel_
                network.add(MaxPooling2D(pool_size = (2,2)))
                network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_
                network.add(MaxPooling2D(pool_size = (2,2)))
                network.add(Conv2D(filters = 64, kernel_size = (3,3), strides = 1, activation = 'relu', kernel_
                network.add(Flatten())
                network.add(Dense(units = 128, activation = 'relu', kernel_regularizer = wr, name = "Dense_1"))
                network.add(Dropout(0.5))
                network.add(Dense(units = 64, activation = 'relu', kernel_regularizer = wr, name = "Dense_2"))
                network.add(Dropout(0.5))
                network.add(Dense(self.label_length, activation = 'softmax', kernel_regularizer = wr))

                network.compile(loss = self.loss, optimizer = tf.keras.optimizers.Adam(learning_rate = lr, epsi

            case '400xn':

                network = Sequential()
                network.add(Conv2D(filters = 8, kernel_size = (7, 7), strides = 4, activation = 'relu', kernel_
                network.add(MaxPooling2D(pool_size = (2, 2)))
                network.add(Conv2D(filters = 16, kernel_size = (3, 3), strides = 1, activation = 'relu', kernel_
                network.add(MaxPooling2D(pool_size = (2, 2)))
                network.add(Flatten())
                network.add(Dense(400, activation = 'relu', kernel_regularizer = wr, name = "Dense1"))
                network.add(Dense(self.label_length, activation = 'softmax', kernel_regularizer = wr))

                network.compile(loss = self.loss, optimizer = tf.keras.optimizers.Adam(learning_rate=lr, epsilo

            case 'ResNet50':

                network = Sequential()

                pretrained_model = ResNet50(include_top=False, input_shape = (self.input_x, self.input_y, self.

                for layer in pretrained_model.layers:
                    layer.trainable = True

                network.add(pretrained_model)
                network.add(Flatten())
                network.add(Dense(512, activation = 'relu'))
                network.add(Dense(self.label_length, activation = 'softmax'))

                network.compile(loss = self.loss, optimizer = 'adam', metrics = ['accuracy'])

            case _:
                raise ValueError(f"Invalid architecture parameter provided: {self.architecture}")

        if (self.display_summary):
            network.summary()

        return network
    #------------------------------------------------------------------------------ NETWORK TRAINING AND TESTING ---
    def train(self, train_data, validation_data, dataframe, train_batch_size, val_batch_size, val_steps, epochs
        """
        Train the network on the training data using model.fit().

        args:
        train_data (tf.data.Dataset): the training dataset
        validation_data (tf.data.Dataset): the validation dataset
        dataframe (DataFrame): An instance of DataFrame containing the representation of the training and test
        train_batch_size (int): the batch size of the training data
        val_batch_size (int): the batch size of the validation data
        val_steps (int): the number of steps to run the validation data
        epochs (int): the number of epochs to train the network
        initial_epoch (int): the initial epoch to start training the network from
        """
        train_data = train_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE # allow for tensorflow to dynamically decide the
        ).batch(
            train_batch_size # batch the dataset into batches of size batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE # allow for prefetching to parallelize the loading of the images and
        )

        validation_data = validation_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE
        ).batch(
            val_batch_size
        ).prefetch(
```

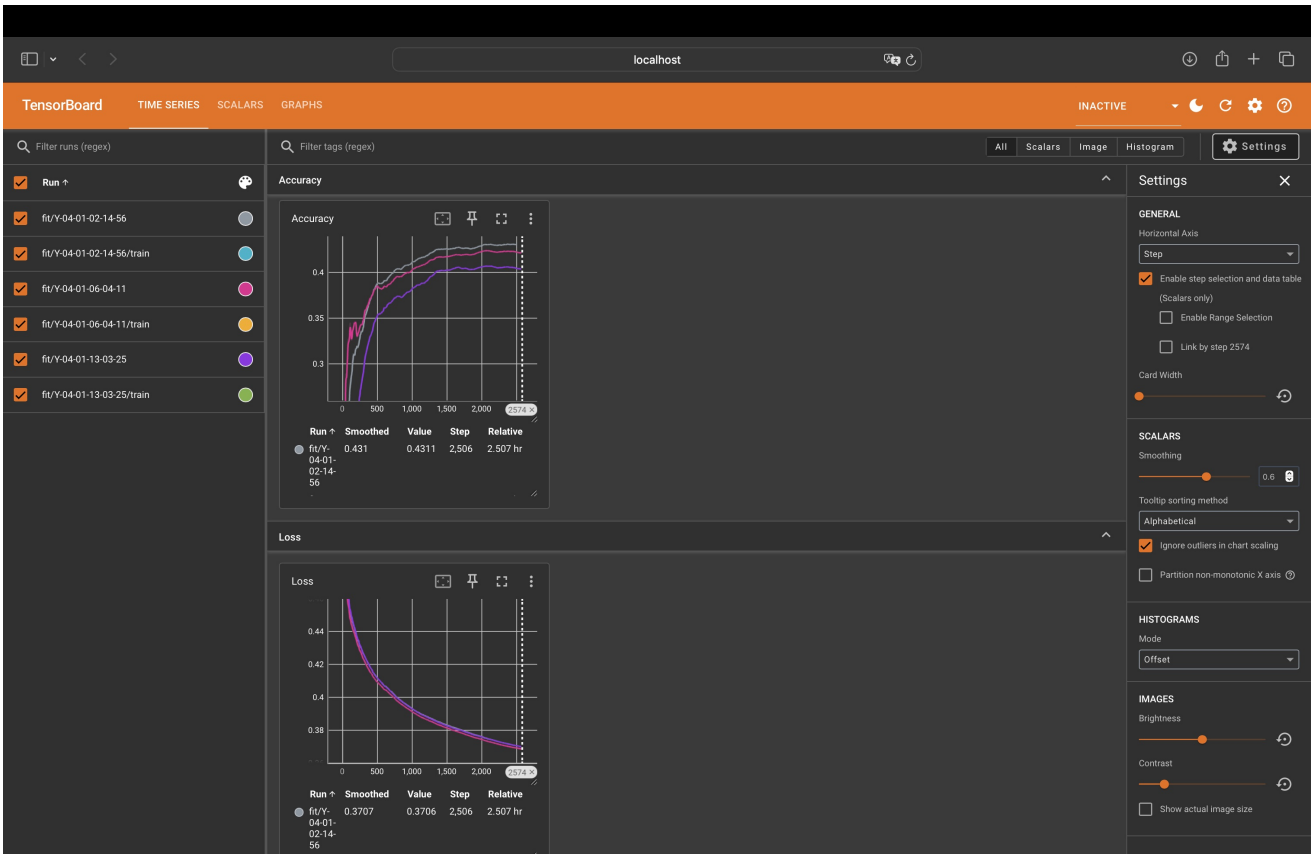
```
            tf.data.experimental.AUTOTUNE
        )

        self.train_history = self.network.fit(
            train_data,
            epochs = epochs,
            verbose = 1,
            callbacks = [self.__checkpoint_callback(checkpointfolder)],
            validation_data = validation_data,
            shuffle = True,
            initial_epoch = initial_epoch,
            steps_per_epoch = None,
            validation_steps = val_steps,
            validation_batch_size = val_batch_size,
            validation_freq = 1,
            # max_queue_size = 10,
            # workers = 1,
            # use_multiprocessing = True
        )

    def test(self, test_data, dataframe, test_batch_size, test_steps):
        """
        Evaluate the network on the test data using model.evaluate().

        args:
        test_data (tf.data.Dataset): the test dataset
        dataframe (DataFrame): An instance of DataFrame containing the representation of the training and test d
        test_batch_size (int): the batch size of the test data
        test_steps (int): the number of steps to run the test data
        """

        test_data = test_data.map(
            lambda image, label: dataframe.fetch_image_label_pair(image, label),
            num_parallel_calls = tf.data.experimental.AUTOTUNE
        ).batch(
            test_batch_size
        ).prefetch(
            tf.data.experimental.AUTOTUNE
        )

        self.test_history = self.network.evaluate(
            test_data,
            batch_size = test_batch_size,
            verbose = 1,
            steps = test_steps,
            callbacks = None,
            # max_queue_size = 10,
            # workers = 1,
            # use_multiprocessing = True,
            return_dict = True
        )

    #------------------------------------------------------------------- NETWORK WEIGHTS AND BIASES SAVING AND LOA
    def save(self, checkpoint_path):
        """
        Save the network weights and biases to a file.

        args:
        checkpoint_path (str): the path to the directory to save the network weights and biases
        """
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.save_weights(path)


    def load(self, checkpoint_path, timestamp):
        """
        Load the network weights and biases from a file.

        args:
        checkpoint_path (str): the path to the directory to load the network weights and biases
        """
        path = os.path.join(checkpoint_path, 'checkpoint-' + timestamp)
        self.network.load_weights(path)

    def __checkpoint_callback(self, checkpointfolder):
        """
        A ModelCheckpoint callback to save the best model during training.

        args:
        checkpointfolder (str): the path to the directory to save the model checkpoint

        returns:
```

```
        tf.keras.callbacks.ModelCheckpoint: a ModelCheckpoint callback
        """

        return tf.keras.callbacks.ModelCheckpoint(
            filepath = os.path.join(checkpointfolder, "model-checkpoint-" + datetime.datetime.now().strftime("%
            save_weights_only = False,
            save_freq = 'epoch',
            monitor = 'val_loss',
            mode = 'min',
            save_best_only = True,
            verbose = 1
        )
```

full problem might be solvable with a more complex architecture like resnet30 or resnet50 with data augmentation applied. still need to clean up dataset from the error message images

need to focus on getting tensorboard working with this now so I can visualise different training metrics.

1/04/2024 - Malachy -



Tensorboard displaying results of 3 different runs of the neural network for approx 2500 batches of size 32 for each run using the pretrained ResNet50 model architecture. Used all 30 class labels but converted into binary labels.

Accuracy appears to level off at around 0.45 Loss appears to level off at round 0.37

8-4-2024 - Malachy- Created image preprocessing and image augmentation scripts:

In [ ]:

```
"""
Malachy -
I have created a new script to remove missing paths from the catalog dataframe and remove any images from the da
"""

# go through the dataset and remove any images from the dataframe and folders that do not have a file path that
# remove any images from the dataframe and folders where the bottom half of the image is only black
import os, cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def remove_missing_paths(dataset_root, catalog_path):
    """
    Remove missing paths from the catalog dataframe and return a new dataframe with the missing paths removed

    Args:
    dataset_root (str): path to the root of the dataset
    catalog_path (str): path to the catalog dataframe
```

```
    Returns:
    pandas_df (pd.DataFrame): new dataframe with missing paths removed
    """
    pandas_df = pd.read_parquet(catalog_path)

    missing_indices = []
    missing_paths = 0

    for idx, _ in enumerate(pandas_df['id_str']):
        file_loc = os.path.join(dataset_root, 'images/', str(pandas_df['subfolder'][idx]) + '/', str(pandas_df[
        pandas_df.at[idx, 'file_loc'] = file_loc  # write new file_loc to file_loc column

        if not os.path.exists(file_loc):
            missing_indices.append(idx)
            missing_paths += 1
            print(f"Missing path: {pandas_df['subfolder'][idx]}/{pandas_df['filename'][idx]}", end = '\r', flus

    print("\n")
    for idx in missing_indices:
        pandas_df = pandas_df.drop(idx)

    pandas_df = pandas_df.reset_index(drop=True)

    print("Missing paths removed")
    print(f"Number of missing paths removed: {missing_paths}")

    return pandas_df

def remove_error_msg_images(pandas_df, show_images, delete_paths):
    """
    Remove error message images from the dataset

    Args:
    pandas_df (pd.DataFrame): dataframe to remove error message images from
    show_images (bool): show images that are removed
    delete_paths (bool): delete the paths of the removed images

    Returns:
    pandas_df (pd.DataFrame): new dataframe with error message images removed
    """

    print("Removing error message images")
    erorr_count = 0
    removed_paths = []

    for idx, _ in enumerate(pandas_df['id_str']):
        print(f"Reading: {pandas_df.loc[idx, 'subfolder']}/{pandas_df.loc[idx, 'filename']}", end = '\r', flush
        image_path = os.path.join(dataset_root, 'images/', str(pandas_df.loc[idx, 'subfolder']) + '/', str(panda

        if os.path.exists(image_path):
            image = cv2.imread(image_path)
            cropped_image = image[image.shape[0]//2:, :]

            if np.all(cropped_image == 0):
                removed_paths.append(image_path)
                print(f"Error message image found: {pandas_df.loc[idx, 'subfolder']}/{pandas_df.loc[idx, 'filena
                if show_images:
                    plt.imshow(image)
                    plt.title(f"{pandas_df['subfolder'][idx]}/{pandas_df['filename'][idx]}")
                    plt.show()

                erorr_count += 1

    for image_path in removed_paths:
        if delete_paths:
            print("Removing: ", image_path, end = '\r', flush = True)
            try:
                os.remove(image_path)
                print(f"Deleted: {image_path}")
            except Exception as e:
                print(f"Failed to delete {image_path}: {e}")

    pandas_df = pandas_df.reset_index(drop=True)

    print(f"Number of error message images removed: {erorr_count}")

    return pandas_df

def build_new_catalog(pandas_df, catalog_path):
    """
    Build a new catalog dataframe and save it to a parquet file

    Args:
```

```
    pandas_df (pd.DataFrame): dataframe to build new catalog from
    catalog_path (str): path to save the new catalog
    """
    print("Building new catalog")
    os.remove(catalog_path)
    pandas_df = pandas_df.reset_index(drop=True)
    pandas_df['iauname'] = pandas_df['iauname'].astype(str)
    pandas_df['summary'] = pandas_df['summary'].astype(str)
    pandas_df.to_parquet(catalog_path)
    print(f"New catalog saved to: {catalog_path}")

dataset_root = "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
train_catalog = "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machin
test_catalog = "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine

paths_dataframe = remove_missing_paths(dataset_root, train_catalog)
processed_dataframe = remove_error_msg_images(paths_dataframe, show_images = False, delete_paths = True)
build_new_catalog(processed_dataframe, catalog_path = train_catalog)

paths_dataframe = remove_missing_paths(dataset_root, test_catalog)
processed_dataframe = remove_error_msg_images(paths_dataframe, show_images = False, delete_paths = True)
build_new_catalog(processed_dataframe, catalog_path = test_catalog)
```

Aroushi -

Developed code to pre process images using randomised augmentations

In [ ]:

```
"""
Aroushi image augmentation code
"""

import pandas as pd
import os
import numpy as np
import tensorflow as tf
from skimage import io, transform
from skimage.util import random_noise
from scipy import fftpack
import random
from PIL import Image

#Define file paths
root_dir = r"/Users/aroushijimulia/Downloads/GitHub/Project-72-Classifying-cosmological-data-with-machine-learn
parquet_file = r'/Users/aroushijimulia/Downloads/GitHub/Project-72-Classifying-cosmological-data-with-machine-l
aug_images_dir = '/Users/aroushijimulia/Downloads/GitHub/Project-72-Classifying-cosmological-data-with-machine-

# Load image file locations
catalog = pd.read_parquet(parquet_file)
# Use the first 100 images for testing augmentation
file_locs = catalog['file_loc'].str[-29:].head(100).tolist()

 def crop_image(image, crop_amount):
     return tf.image.crop_to_bounding_box(image, crop_amount, crop_amount, image.shape[0]-2*crop_amount, image.


 def flip_vertically(image):
     return tf.image.flip_up_down(image).numpy()


 def denoise_fft(image): #will need to be adjusted, values are arbitrary
     # Convert to frequency domain
     im_fft = fftpack.fft2(image)
     # Remove frequencies below a threshold
     keep_fraction = 0.1
     im_fft2 = im_fft.copy()
     r, c = im_fft2.shape
     im_fft2[int(r*keep_fraction):int(r*(1-keep_fraction))] = 0
     im_fft2[:, int(c*keep_fraction):int(c*(1-keep_fraction))] = 0
     # Convert back to time domain
     im_new = fftpack.ifft2(im_fft2).real
     return im_new


def flip_image(image):
    if random.choice([True, False]):
        return image.transpose(Image.FLIP_LEFT_RIGHT)
    return image

def rotate_image(image):
    angles = [45, 90, -45, -90]
    return image.rotate(random.choice(angles))

def add_noise(image):
```

```
    # Convert to numpy array for noise operations
    image_array = np.asarray(image)
    noisy_image = random_noise(image_array)
    # Convert back to Image
    return Image.fromarray((noisy_image * 255).astype(np.uint8))


def apply_random_augmentation(image):
    aug_methods = {
        'no_aug': lambda x: (x, ''),
        'flip': lambda x: (flip_image(x), 'flip'),
        'rotate': lambda x: (rotate_image(x), 'rotate'),
        'noise': lambda x: (add_noise(x), 'noise')
    }
    aug_choice = random.choice(list(aug_methods.keys()))
    return aug_methods[aug_choice](image)


for idx, rel_path in enumerate(file_locs):
    full_path = os.path.join(root_dir, rel_path)
    image = Image.open(full_path)

    aug_description = 'orig'  # Default description for no augmentation
    if idx >= 20:  # Apply augmentations only to 80% of the images
        image, aug_description = apply_random_augmentation(image)

    # Save the augmented image with a descriptive filename
    aug_image_filename = f"aug_image_{idx}_{aug_description}.jpg"
    aug_image_path = os.path.join(aug_images_dir, aug_image_filename)
    image.save(aug_image_path)


# Can easily add augmentations
# Next steps, need to adjust it so that multiple augmentations are applied to the same image at random
# Also need to adjust noise parameters
```

In [ ]:

```
"""
Malachy -
Image augmentation script, pre-processing the images in the dataset by applying augmentations to them.
"""

import os, cv2
import pandas as pd
import numpy as np
import time

# duplicate images in dataset by applying augmentations to them.
# create new subfolder with A appended to the start of the subfolder name and save the augmented images there.
# do this for all images in the dataset, 4 times for each image, add an extra A to the subfolder name each time
# append the new images to the end of the catalog, keeping the original data associated with the image, but rep
# can apply multiple augmentations to the same image, i.e rotation by random angle, flipping, zoom, adding noise

def read_parquet(path):
    """
    Read the parquet file at the path.

    args:
    path (str): The path to the parquet file.

    returns:
    pandas_df (pd.DataFrame): The dataframe containing the data from the parquet file.
    """
    return pd.read_parquet(path)

def open_image(image_path):
    """
    Open the image at the image_path.

    args:
    image_path (str): The path to the image.

    returns:
    image (tf.Tensor): The image.
    """
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image

def get_augmentations(image, RNG):
    """
    Get a random set of 3 augmentations to apply to the image.

    args:
```

```
    image (tf.Tensor): The image to apply augmentations to.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (tf.Tensor): The image with the augmentations applied.
    aug_1 (str): The first augmentation applied.
    aug_2 (str): The second augmentation applied.
    aug_3 (str): The third augmentation applied.
    """

    augmentations = np.random.choice(['None', 'Flip', 'Rotation', 'Scale', 'Blurr', 'Noise', 'Translation'], 3,
    aug_1, aug_2, aug_3 = augmentations

    for aug in augmentations:
        match aug:
            case 'None':
                image = no_augmentations(image)
            case 'Flip':
                image = random_flip(image, RNG)
            case 'Rotation':
                image = random_rotation(image, RNG)
            case 'Scale':
                image = random_zoom(image, RNG)
            case 'Hue':
                image = random_hue(image, RNG)
            case 'Blurr':
                image = random_blurr(image, RNG)
            case 'Noise':
                image = noise_denoise(image, RNG)
            case 'Translation':
                image = random_translation(image, RNG)

    image = cv2.convertScaleAbs(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image, aug_1, aug_2, aug_3

def no_augmentations(image):
    """
    Do not apply any augmentations to the image.

    args:
    image (tf.Tensor): The image to not apply augmentations to.

    returns:
    image (tf.Tensor): The original image.
    """
    return image

def random_flip(image, RNG):
    """
    Flip the image along a random axis.
    axis = 0 -> flip up-down
    axis = 1 -> flip left-right

    args:
    image (np.array): The image to flip.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The flipped image.
    """
    axis = RNG.integers(0, 1, endpoint = True)
    image = cv2.flip(image, axis)
    return image

def random_rotation(image, RNG):
    """
    Rotate the image by a random angle between -180 and 180 degrees.

    args:
    image (tf.Tensor): The image to rotate.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The rotated image
    """
    rows, cols, dim = image.shape
    angle = RNG.uniform(-180, 180)
    M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    image = cv2.warpAffine(image, M, (cols, rows))
    return image

def random_zoom(image, RNG):
```

```
    """
    Zoom the image by a random value between -0.5 and 0.5.
    Ensure the aspect ratio of the zoomed image is square.

    args:
    image (np.array): The image to zoom.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The zoomed image.
    """
    scale = RNG.uniform(-0.75, 1.5)
    rows, cols, dim = image.shape
    M = cv2.getRotationMatrix2D((cols/2, rows/2), 0, scale)
    image = cv2.warpAffine(image, M, (cols, rows))
    return image

def random_hue(image, RNG):
    """
    Change the hue of the image by a random value between -180 and 180.

    args:
    image (np.array): The image to change the hue of.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The image with the hue changed.
    """
    delta = RNG.uniform(-180, 180)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(image)
    h = np.clip(h + delta, 0, 255)
    image = cv2.merge((h, s, v))
    image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
    return image

def random_blurr(image, RNG):
    """
    Apply a random blurring filter to the image.

    args:
    image (np.array): The image to apply the blurring filter to.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The image with the blurring filter applied.
    """
    k = RNG.choice([3, 5, 7])
    image = cv2.GaussianBlur(image, ksize = (k, k), sigmaX = 0, sigmaY = 0)
    return image

def noise_denoise(image, RNG):
    """
    Add random noise to the image or denoise the image.
    50% chance of adding noise, 50% chance of denoising the image.

    args:
    image (np.array): The image to add noise to or denoise.
    RNG (np.random.Generator): The random number generator.

    returns:
    image (np.array): The image with noise added or denoised.
    """
    choice = RNG.integers(0, 1, endpoint=True)

    if choice == 0:
        noise = np.random.normal(0, 1, image.shape)
        image = image + noise
        image = np.clip(image, 0, 255)
        return image
    else:
        image = cv2.fastNlMeansDenoisingColored(image, h = 10, hColor = 10, templateWindowSize = 7, searchWindo
        return image

def random_translation(image, RNG):
    """
    Translate the image by a random value between -0.5 and 0.5.

    args:
    image (np.array): The image to translate.
    RNG (np.random.Generator): The random number generator.

    returns:
```

```
    image (np.array): The translated image.
    """
    rows, cols, dim = image.shape
    tx = RNG.uniform(-0.25, 0.25) * cols
    ty = RNG.uniform(-0.25, 0.25) * rows
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    image = cv2.warpAffine(image, M, (cols, rows))
    return image

def augment_images(pandas_df, max_iter, dataset_root, catalog, RNG):
    """
    Augment the images in the pandas_df by applying random augmentations to them.
    Save the augmented images to a new subfolder in the images directory.
    Append the new image data to the end of the pandas_df.
    Repeat this process max_iter times.

    Parameters:
    pandas_df (pd.DataFrame): The dataframe containing the image data.
    max_iter (int): The number of times to augment the images.
    dataset_root (str): The root directory of the dataset.
    catalog (str): The name of the catalog.
    RNG (np.random.Generator): The random number generator.
    """

    times = []

    for iter in range(1, max_iter + 1):

        new_pandas_df = pd.DataFrame()

        for idx, _ in enumerate(pandas_df['file_loc']):

            start_time = time.time()
            image = pandas_df.loc[idx, 'file_loc']
            image_data = pandas_df.loc[idx].copy()
            image = open_image(image)
            image, aug_1, aug_2, aug_3  = get_augmentations(image, RNG)

            image_subfolder = f"Augmented-{iter}-{image_data['subfolder']}"
            image_name = f"A{iter}-{aug_1}-{aug_2}-{aug_3}-{image_data['filename']}"

            sub_folder_path = os.path.join(dataset_root, 'images/', image_subfolder)

            if not os.path.exists(sub_folder_path):
                os.mkdir(sub_folder_path)

            image_path = os.path.join(sub_folder_path, image_name)

            image_data['subfolder'] = image_subfolder
            image_data['filename'] = image_name
            image_data['file_loc'] = image_path

            new_pandas_df = new_pandas_df._append(image_data, ignore_index = True)

            cv2.imwrite(image_data['file_loc'], image)

            end_time = time.time()
            iter_time = end_time - start_time
            times.append(iter_time)

            if len(times) > 1024:
                times = times[:len(times)//2]

            if times == []:
                mean_time = None
            else:
                mean_time = np.mean(times)

            print(f"iter: {iter}/{max_iter} \t | {idx}/{len(pandas_df['file_loc'])} \t | ETA: {mean_time * (len
            = True)

        # save the new_pandas_df to file
        new_pandas_df.to_parquet(f"{dataset_root}/augmented_{iter}_{catalog}")

dataset_root = "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
train_catalog = "gz2_train_catalog.parquet"
test_catalog = "gz2_test_catalog.parquet"

train_catalog_path = os.path.join(dataset_root, train_catalog)
test_catalog_path = os.path.join(dataset_root, test_catalog)

train_df = read_parquet(train_catalog_path)
test_df = read_parquet(test_catalog_path)
```

```
max_iter = 1

seed = 1
RNG = np.random.default_rng(seed)

augment_images(pandas_df = train_df, max_iter = max_iter, dataset_root = dataset_root, catalog = train_catalog,
augment_images(pandas_df = test_df, max_iter = max_iter, dataset_root = dataset_root, catalog = test_catalog, RI

# should probably manually delete the duplicate catalogs created by the augmentation process, but
# I won't make a script to do that as its also handy to keep them around incase something breaks.
```

The image augmentation script is SLOW. not really much I can do about that, will take me a few days to pre augment all my data but it will save time during training so it is potentially worth it.

I will create 6 different views of each image + the original for a total 7 different views for every image, giving me well over 1M images to train off.

11-4-2024 Malachy - Implemented on the fly augmentation in training dataset pipline map function.

In [ ]:
```
def fetch_image_label_pair(self, image_path, label, architecture, augmentation):
    """
    Fetches an image and its corresponding label from the dataset.

    args:
    image_path (tf.Tensor): the path to the image file
    label (tf.Tensor): the label of the image
    architecture (str): the architecture of the model to be used
    augmentation (bool): whether to apply image augmentation

    returns:
    (tuple): of the image and its corresponding label
        image (tf.Tensor): the requested image
        label (tf.Tensor): the requested label
    """
    image_path = tf.squeeze(image_path)
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels = self.input_dim)
    image = tf.image.convert_image_dtype(image, tf.float32) # Convert image dtype from tf.uint8 to tf.float32 wi
    image = tf.image.resize(image, self.input_shape)

    if augmentation:
        augs = tf.random.shuffle(['None', 'flip-lr', 'flip-ud', 'rot90'])[:1]
        for a in augs:
            if a == 'None':
                image = image
            elif a == 'flip-lr':
                image = tf.image.random_flip_left_right(image)
            elif a == 'flip-ud':
                image = tf.image.random_flip_up_down(image)
            elif a == 'rot90':
                image = tf.image.rot90(image, k = tf.random.uniform(shape = [], minval = 0, maxval = 4, dtype =

    # Pre trained models require the input to be preprocessed before passing to the model
    if architecture == "ResNet50":
        image = tf.keras.applications.resnet.preprocess_input(image)
    elif architecture == "ResNet50V2":
        image = tf.keras.applications.resnet_v2.preprocess_input(image)
    elif architecture == "VGG16":
        image = tf.keras.applications.vgg16.preprocess_input(image)
    elif architecture == "VGG19":
        image = tf.keras.applications.vgg19.preprocess_input(image)
    elif architecture == "MobileNetV3Small":
        image = tf.keras.applications.MobileNetV3Small.preprocess_input(image)
    elif architecture == "MobileNetV3Large":
        image = tf.keras.applications.MobileNetV3Large.preprocess_input(image)

    return image, label
```

Aroushi -

Week Starting April 22nd:

Thursday April 25th 10am-11am: SUPERVISOR MEETING

- Once again, reviewed progress so far and compared with initial project plan
- Started to discuss how we might start to find some final results for the report
- Adam suggested looking into transformer models as an alternative model architecture (inspired by the winner of the galaxy zoo kaggle competition)

25/04/2024 Malachy - tensorflow only supports basic flipping and rotation by 90 degrees. I tried implementing a custom augmentation

script by decorating functions with @tf.function or wrapping in tf.py_functions however i kept running into issues with graph execution and compatibility with tensorflow and libaries like open cv2. have decided it is easier to have basic on the fly augmentation here and more complex methods such as denoising can be done as a preprocessing step.

Ensemble of CNNs could work: have CNNs in parallel that identify if the features are present that correlate to the top layer of the decision tree, i.e. is there spiral arms? is there a central bulge? ect then can have in series to those networks, networks that are built to identify the specifics of those features, i.e how many spiral arms? bulge size? bulge shape? ect.

right now the labels are this:

Q0/ Smooth or Featured? 'smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction', 'smooth-or-featured-gz2_artifact_fraction',

Q1/ Disk Edge On? 'disk-edge-on-gz2_yes_fraction', 'disk-edge-on-gz2_no_fraction',

Q2/ Has spiral arms? 'has-spiral-arms-gz2_yes_fraction', 'has-spiral-arms-gz2_no_fraction',

Q3/ Bar? 'bar-gz2_yes_fraction', 'bar-gz2_no_fraction',

Q4/ Bulge size? 'bulge-size-gz2_dominant_fraction', 'bulge-size-gz2_obvious_fraction', 'bulge-size-gz2_just-noticeable_fraction', 'bulge-size-gz2_no_fraction',

Q5/ Something odd? 'something-odd-gz2_yes_fraction', 'something-odd-gz2_no_fraction',

Q6/ Round? 'how-rounded-gz2_round_fraction', 'how-rounded-gz2_in-between_fraction', 'how-rounded-gz2_cigar_fraction',

Q7/ Bulge shape? 'bulge-shape-gz2_round_fraction', 'bulge-shape-gz2_boxy_fraction', 'bulge-shape-gz2_no-bulge_fraction',

Q8/ Spiral winding? 'spiral-winding-gz2_tight_fraction', 'spiral-winding-gz2_medium_fraction', 'spiral-winding-gz2_loose_fraction',

Q9/ Spiral arms count? 'spiral-arm-count-gz2_1_fraction', 'spiral-arm-count-gz2_2_fraction', 'spiral-arm-count-gz2_3_fraction', 'spiral-arm-count-gz2_4_fraction', 'spiral-arm-count-gz2_more-than-4_fraction', 'spiral-arm-count-gz2_cant-tell_fraction',

now for the labels: 'bulge-size-gz2_no_fraction' and 'bulge-shape-gz2_no-bulge_fraction', remove them from questions Q4 and Q7 and instead create a new question:

i.e Q4/ Is there a central bulge? with the labels: 'bulge-gz2-yes_fraction', 'bulge-gz2-no_fraction' where yhe no_fraction is the average of 'bulge-size-gz2_no_fraction' and 'bulge-shape-gz2_no-bulge_fraction' and 'bulge-gz2-yes_fraction' = 1 - 'bulge-gz2-no_fraction' then remove the redundant columns from the dataframe and insert the new columns.

so then the possible questions and answers will be

Q0/ Smooth or Featured? 'smooth-or-featured-gz2_smooth_fraction', 'smooth-or-featured-gz2_featured-or-disk_fraction', 'smooth-or-featured-gz2_artifact_fraction',

Q1/ Disk Edge On? 'disk-edge-on-gz2_yes_fraction', 'disk-edge-on-gz2_no_fraction',

Q2/ Has spiral arms? 'has-spiral-arms-gz2_yes_fraction', 'has-spiral-arms-gz2_no_fraction',

Q3/ Bar? 'bar-gz2_yes_fraction', 'bar-gz2_no_fraction',

Q4/ Is there a central bulge? 'bulge-gz2-yes_fraction', 'bulge-gz2-no_fraction'

Q5/ Bulge size? 'bulge-size-gz2_dominant_fraction', 'bulge-size-gz2_obvious_fraction', 'bulge-size-gz2_just-noticeable_fraction'

Q6/ Something odd? 'something-odd-gz2_yes_fraction', 'something-odd-gz2_no_fraction',

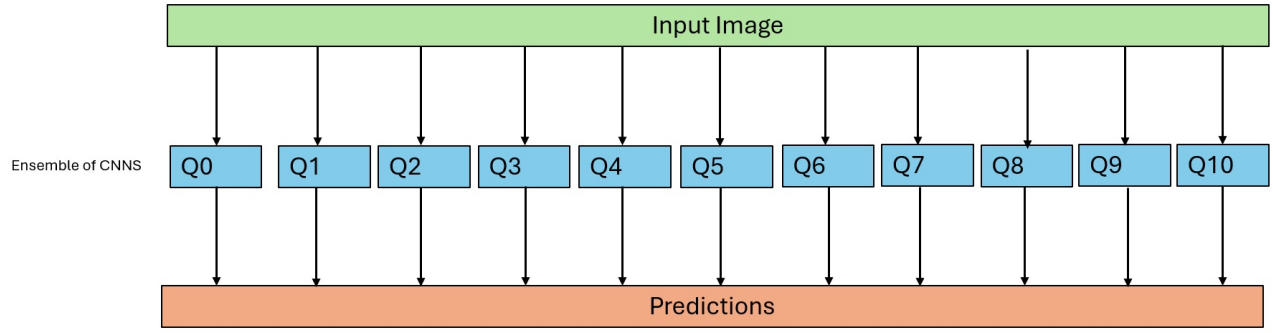
Q7/ Round? 'how-rounded-gz2_round_fraction', 'how-rounded-gz2_in-between_fraction', 'how-rounded-gz2_cigar_fraction',

Q8/ Bulge shape? 'bulge-shape-gz2_round_fraction', 'bulge-shape-gz2_boxy_fraction',

Q9/ Spiral winding? 'spiral-winding-gz2_tight_fraction', 'spiral-winding-gz2_medium_fraction', 'spiral-winding-gz2_loose_fraction',

Q10/ Spiral arms count? 'spiral-arm-count-gz2_1_fraction', 'spiral-arm-count-gz2_2_fraction', 'spiral-arm-count-gz2_3_fraction', 'spiral-arm-count-gz2_4_fraction', 'spiral-arm-count-gz2_more-than-4_fraction', 'spiral-arm-count-gz2_cant-tell_fraction',

i.e now the network ensemble structure can be like:



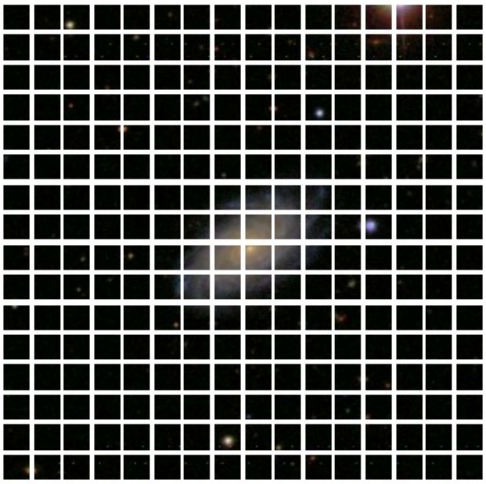
Transformer Model could also work.

25/04/2024 Malachy -

In [ ]:
```python
import keras
import tensorflow as tf
from keras import layers
from keras import ops
import numpy as np
import matplotlib.pyplot as plt

num_patches = 16
patch_size = 424 // num_patches
image_path = "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning/gal

class Patches(layers.Layer):
    def __init__(self, patch_size):
        super().__init__()
        self.patch_size = patch_size

    def call(self, images):

        input_shape = ops.shape(images)
        batch_size = input_shape[0]
        height = input_shape[1]
        width = input_shape[2]
        channels = input_shape[3]
        num_patches_h = height // self.patch_size
        num_patches_w = width // self.patch_size

        patches = keras.ops.image.extract_patches(images, size = self.patch_size)
        patches = ops.reshape(
            patches,
            (
                batch_size,
                num_patches_h * num_patches_w,
                self.patch_size * self.patch_size * channels,
            ),
        )
        return patches

    def get_config(self):
        config = super().get_config()
        config.update({"patch_size": self.patch_size})
        return config

plt.figure(figsize=(8, 8))
image = tf.io.read_file(image_path)
image = tf.image.decode_jpeg(image, channels = 3)
image = tf.cast(image, tf.float32)
image = tf.image.resize(image, (424, 424))

image_size = image.shape
image = tf.expand_dims(image, 0)

patches = Patches(patch_size)(image)

print(f"Image size: {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))

for i, patch in enumerate(patches[0]):
```

```
    ax = plt.subplot(n, n, i + 1)
    patch_img = ops.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(ops.convert_to_numpy(patch_img).astype("uint8"))
    plt.axis("off")

plt.show()
```

```
2024-04-25 05:41:15.584114: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optim
ized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate com
piler flags.
2024-04-25 05:41:16.289390: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not fi
nd TensorRT
2024-04-25 05:41:17.322642: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.345099: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.345154: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.348659: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.348729: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.348770: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.482828: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.482898: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.482905: I tensorflow/core/common_runtime/gpu/gpu_device.cc:2019] Could not identify NUMA nod
e of platform GPU id 0, defaulting to 0.  Your kernel may not have been built with NUMA support.
2024-04-25 05:41:17.482952: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:984] could not open f
ile to read NUMA node: /sys/bus/pci/devices/0000:0a:00.0/numa_node
Your kernel may have been built without NUMA support.
2024-04-25 05:41:17.482972: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1928] Created device /job:localho
st/replica:0/task:0/device:GPU:0 with 5714 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 2080 SUPER, pci bu
s id: 0000:0a:00.0, compute capability: 7.5
2024-04-25 05:41:29.004718: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:465] Loaded cuDNN version
8907
Image size: (424, 424, 3)
Patch size: 26 X 26
Patches per image: 256
Elements per patch: 2028
<Figure size 800x800 with 0 Axes>
```



Splitting an image into patches with keras to pass into a transformer model.

25/04/2024 Malachy - transformer model implementation:

- transformer.py :

In [ ]:
```
import keras
from keras import layers
from keras import ops
import numpy as np
```

```
# credit for implementation: https://keras.io/examples/vision/image_classification_with_vision_transformer/#bui

def mlp(x, hidden_units, dropout_rate):
    """
    Multilayer Perceptron (MLP)
    """
    for units in hidden_units:
        x = layers.Dense(units, activation = keras.activations.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

class Patches(layers.Layer):
    def __init__(self,  CNNreference, num_patches):
        """
        Construct a keras layer that splits images into patches to be passed into a transformer model

        args:
        CNNreference (ConvolutionalNeuralNetwork): The Convolutional Neural Network object
        num_patches (int): the number of patches to split the image into
        """
        super().__init__()
        self.CNNreference = CNNreference
        self.num_patches = int(np.sqrt(num_patches))
        self.patch_size = CNNreference.input_x // self.num_patches

    def call(self, images):

        input_shape = ops.shape(images)
        batch_size = input_shape[0]
        size = input_shape[1]
        channels = input_shape[3]
        num_patches = size // self.patch_size

        patches = keras.ops.image.extract_patches(images, size = self.patch_size)
        patches = ops.reshape(
            patches,
            (
                batch_size,
                num_patches**2,
                self.patch_size * self.patch_size * channels,
            ),
        )
        return patches

    def get_config(self):
        config = super().get_config()
        config.update({"patch_size": self.patch_size})
        return config

class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        """
        Construct a Patch Encoding Layer.
        Linearly transforms a patch by projecting it into a vector of size projection_dim and adds a learnable p
        embedding to the projected vector.

        args:
        num_patches (int): the total number of patches the image is divided into
        projection_dim (int): the dimensionality of the projected vector space.
        """
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units = projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim = num_patches, output_dim = projection_dim
        )

    def call(self, patch):
        positions = ops.expand_dims(
            ops.arange(start=0, stop = self.num_patches, step = 1 ), axis = 0
        )
        projected_patches = self.projection(patch)
        encoded = projected_patches + self.position_embedding(positions)
        return encoded

    def get_config(self):
        config = super().get_config()
        config.update({"num_patches": self.num_patches})
        return config
```

ViT Model:

In [ ]:
```
case 'ViT':
```

```
    inputs = keras.Input(shape = self.input_shape)
    patches = Patches(self.num_patches)(inputs)
    encoded_patches = PatchEncoder(self.num_patches, self.projection_dim)(patches)

    for _ in range(self.transformer_layers):
        x1 = layers.layerNormalization(epsilon = 1e-6)(encoded_patches) # layer normalisation 1
        attention_output = layers.MultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim,

        # skip connection 1
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon = 1e-6)(x2) # layer normalisation 2

        x3 = mlp(x3, hidden_units = self.transformer_units, dropout_rate = 0.1) #mlp

        # skip connection 2
        encoded_patches = layers.Add()([x3, x2])

    # create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon = 1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)

    features = mlp(representation, hidden_units = self.mlp_head_units, dropout_rate = 0.5)
    logits = layers.Dense(self.label_length)(features)

    network = Model(inputs = inputs, outputs = logits)

    self.__compile_network(network)
```

26/04/2024 Malachy -

added loss function weighting. weights uncommon classes more and weights common classes less in the loss function, makes the network pay more attention to the less common features. 3 methods: inverse: 1/fraction where fraction is the fraction of galaxies of having that feature in the datset. inverse square root: 1/sqrt(fraction) effective num samples: weights the features based off the effective number of samples of the feature in the dataset, based off the idea that as a network sees more examples of a feature the importance of each example has exponentially diminishing returns. for more info: https://openaccess.thecvf.com/content_CVPR_2019/papers/Cui_Class-Balanced_Loss_Based_on_Effective_Number_of_Samples_CVPR_2019_paper.pdf

In [ ]:
```
def get_feature_weights(self):
    """
    Get a dictionary containing the weights to be used in the loss function for each class label.

    returns:
    weight_dict (dict): a dictionary mapping of class indices (integers) to a weight (float)
    """
    return self.class_weights

def __create_weights(self, pandas_dataframe):
    """
    create class weights for the dataset by finding the fraction of galaxies that have a specific feature (= 1 |

    args:
    pandas_dataframe (pd.DataFrame): the dataframe containing the class labels

    returns:
    weights (list): a list of the weights for each class label
    """
    weights = []

    for key in self.keys:
        num = (pandas_dataframe[key] == 1).sum()
        fraction = num / len(pandas_dataframe[key])

        if self.weight_mode == 'inverse':
            weight = 1 / fraction

        elif self.weight_mode == 'inverse-sqrt':
            weight = 1 / np.sqrt(fraction)

        elif self.weight_mode == 'effective_num_samples':
            effective_num = (1 - self.beta**num)
            weight = (1 - self.beta) / effective_num

        elif self.weight_mode is None:
            weight = 1

        else:
            raise ValueError(f"\033[31mInvalid weight_mode: {self.weight_mode}\033[0m")

        weights.append(weight)
```

```
    return {idx: weight for idx, weight in enumerate(weights)}
```

27/4/2024 Malachy - function to construct proposed bulge present question, this function is now a part of cleandataset.py inside image_preprocessing. (makes sense to build the question here if I'm already going to pass the catalogs through this file)

In [ ]:
```
def construct_bulge_present_question(pandas_dataframe):
    """
    Q4/ Is there a central bulge?
    with the labels: 'bulge-gz2-yes_fraction', 'bulge-gz2-no_fraction'

    Take the columns 'bulge-size-gz2_no_fraction' and 'bulge-shape-gz2_no-bulge_fraction' from the dataframe an
    Then 'bulge-gz2-yes_fraction' = 1 - 'bulge-gz2-no_fraction'. Then insert the new columns bulge-gz2-yes_frac
    """
    pandas_dataframe['bulge-gz2-no_fraction'] = (pandas_dataframe['bulge-size-gz2_no_fraction'] + pandas_datafra
    pandas_dataframe['bulge-gz2-yes_fraction'] = 1 - pandas_dataframe['bulge-gz2-no_fraction']

    return pandas_dataframe
```

```
case 'all':
    keys = [
        # Q0/ Smooth or Featured?
        'smooth-or-featured-gz2_smooth_fraction',
        'smooth-or-featured-gz2_featured-or-disk_fraction',
        # 'smooth-or-featured-gz2_artifact_fraction', remove as it's so uncommon its basically impossible to classify well

        # Q1/ Disk Edge On?
        'disk-edge-on-gz2_yes_fraction',
        'disk-edge-on-gz2_no_fraction',

        # Q2/ Has spiral arms?
        'has-spiral-arms-gz2_yes_fraction',
        'has-spiral-arms-gz2_no_fraction',

        # Q3/ Bar?
        'bar-gz2_yes_fraction',
        'bar-gz2_no_fraction',

        # Q4/ Bulge size?
        'bulge-size-gz2_dominant_fraction',
        'bulge-size-gz2_obvious_fraction',
        'bulge-size-gz2_just-noticeable_fraction',

        # Q5/ Something odd?
        'something-odd-gz2_yes_fraction',
        'something-odd-gz2_no_fraction',

        # Q6/ Round?
        'how-rounded-gz2_round_fraction',
        'how-rounded-gz2_in-between_fraction',
        'how-rounded-gz2_cigar_fraction',

        # Q7/ Bulge shape?
        'bulge-shape-gz2_round_fraction',
        'bulge-shape-gz2_boxy_fraction',

        # Q8/ Spiral winding?
        'spiral-winding-gz2_tight_fraction',
        'spiral-winding-gz2_medium_fraction',
        'spiral-winding-gz2_loose_fraction',

        # Q9/ Spiral arms count?
        'spiral-arm-count-gz2_1_fraction',
        'spiral-arm-count-gz2_2_fraction',
        'spiral-arm-count-gz2_3_fraction',
        'spiral-arm-count-gz2_4_fraction',
        'spiral-arm-count-gz2_more-than-4_fraction',
        'spiral-arm-count-gz2_cant-tell_fraction',

        # Q10/ Bulge?
        'bulge-gz2-yes_fraction',
        'bulge-gz2-no_fraction',
    ]
    return keys
```

implemented bulge presence question

since the network should classify the presence of a feature instead of replicating the human distribution of votes and part of this requirement is to use binary labels as either the feature is present or it isn't. another thing that can be done is to simplfy the questions and corresponding labels that are binary and mutually exclusive to a single binary label.

i.e. 'bulge-gz2-yes_fraction' and 'bulge-gz2-no_fraction' should be a single label where 1 = 'bulge-gz2-yes_fraction' and 0 = 'bulge-gz2-no_fraction'.

this will simplify the feature space and possibly improve performance of the models.

i.e the labels will now be:

In [ ]:
```
keys = [
    # Q0/ Smooth or Featured?
    'smooth-or-featured-gz2_smooth_fraction', # 1 =  'smooth-or-featured-gz2_smooth_fraction' 0 = 'smooth-or-fea

    # Q1/ Disk Edge On?
    'disk-edge-on-gz2_yes_fraction', # 1 = 'disk-edge-on-gz2_yes_fraction' 0 = 'disk-edge-on-gz2_no_fraction',

    # Q2/ Has spiral arms?
    'has-spiral-arms-gz2_yes_fraction', # 1 = 'has-spiral-arms-gz2_yes_fraction', 0 = 'has-spiral-arms-gz2_no_fr

    # Q3/ Bar?
    'bar-gz2_yes_fraction', # 1 = 'bar-gz2_yes_fraction', 0 = 'bar-gz2_no_fraction'

    # Q4/ Bulge size?
    'bulge-size-gz2_dominant_fraction',
    'bulge-size-gz2_obvious_fraction',
    'bulge-size-gz2_just-noticeable_fraction',

    # Q5/ Something odd?
    'something-odd-gz2_yes_fraction', # 1 = 'something-odd-gz2_yes_fraction', 0 = 'something-odd-gz2_no_fractio

    # Q6/ Round?
    'how-rounded-gz2_round_fraction',
    'how-rounded-gz2_in-between_fraction',
    'how-rounded-gz2_cigar_fraction',

    # Q7/ Bulge shape?
    'bulge-shape-gz2_round_fraction', # 1 = 'bulge-shape-gz2_round_fraction', 0 = 'bulge-shape-gz2_boxy_fractio

    # Q8/ Spiral winding?
    'spiral-winding-gz2_tight_fraction',
    'spiral-winding-gz2_medium_fraction',
    'spiral-winding-gz2_loose_fraction',

    # Q9/ Spiral arms count?
    'spiral-arm-count-gz2_1_fraction',
    'spiral-arm-count-gz2_2_fraction',
    'spiral-arm-count-gz2_3_fraction',
    'spiral-arm-count-gz2_4_fraction',
    'spiral-arm-count-gz2_more-than-4_fraction',
    'spiral-arm-count-gz2_cant-tell_fraction',

    # Q10/ Bulge?
    'bulge-gz2-yes_fraction', # 1 = 'bulge-gz2-yes_fraction', 0 = 'bulge-gz2-no_fraction'
]
```

this reduces the number of labels from 30 to 22, about a 1/3 rd reduction in labels.

Q5/ Something odd? isn't really a specific feature, could be dust lane, galaxy merger, lensing event, overlapping, ring, irrelgular, or other. in the other galaxy zoo datasets there is 37 features, for some reason in this one the something odd question has been reduced to a binary yes/no concatentation of multiple features this means the multiple features have no commonality between themselves and are quite rare meaning this question will be very hard for the network to classify so it's probably best to just remove this label as this problem is a galaxy classification problem and not a lensing event classification problem ect.

finalised keys that i'm going to use will be labled 'all-features':

In [ ]:
```
keys = [
    # Q0/ Smooth or Featured?
    'smooth-or-featured-gz2_smooth_fraction', # 1 =  'smooth-or-featured-gz2_smooth_fraction' 0 = 'smooth-or-fea

    # Q1/ Disk Edge On?
    'disk-edge-on-gz2_yes_fraction', # 1 = 'disk-edge-on-gz2_yes_fraction' 0 = 'disk-edge-on-gz2_no_fraction',

    # Q2/ Has spiral arms?
    'has-spiral-arms-gz2_yes_fraction', # 1 = 'has-spiral-arms-gz2_yes_fraction', 0 = 'has-spiral-arms-gz2_no_fr

    # Q3/ Bar?
    'bar-gz2_yes_fraction', # 1 = 'bar-gz2_yes_fraction', 0 = 'bar-gz2_no_fraction'

    # Q4/ Bulge size?
    'bulge-size-gz2_dominant_fraction',
    'bulge-size-gz2_obvious_fraction',
    'bulge-size-gz2_just-noticeable_fraction',
```

```
    # Q6/ Round?
    'how-rounded-gz2_round_fraction',
    'how-rounded-gz2_in-between_fraction',
    'how-rounded-gz2_cigar_fraction',

    # Q7/ Bulge shape?
    'bulge-shape-gz2_round_fraction', # 1 = 'bulge-shape-gz2_round_fraction', 0 = 'bulge-shape-gz2_boxy_fractio

    # Q8/ Spiral winding?
    'spiral-winding-gz2_tight_fraction',
    'spiral-winding-gz2_medium_fraction',
    'spiral-winding-gz2_loose_fraction',

    # Q9/ Spiral arms count?
    'spiral-arm-count-gz2_1_fraction',
    'spiral-arm-count-gz2_2_fraction',
    'spiral-arm-count-gz2_3_fraction',
    'spiral-arm-count-gz2_4_fraction',
    'spiral-arm-count-gz2_more-than-4_fraction',
    'spiral-arm-count-gz2_cant-tell_fraction',

    # Q10/ Bulge?
    'bulge-gz2-yes_fraction', # 1 = 'bulge-gz2-yes_fraction', 0 = 'bulge-gz2-no_fraction'
]
```

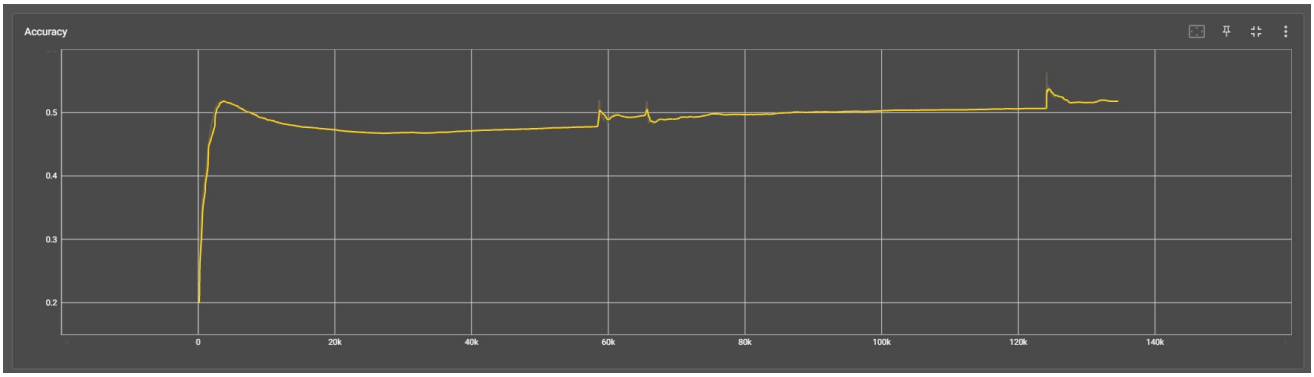
Aroushi -

Week Starting May 6th:

Thursday May 9th 10am-11am: SUPERVISOR MEETING

- Discussed end of project and reflected on final steps
- Reviewed data formatting and its impact on how we might make confusion matrices
- Compared models used and discussed why some models perform significantly better than others
- Below are some of the results obtained along with the dates they were obtained on

27/04/2024 - Malachy - ViT model training:



meta.json:

In [ ]:

```
{
    "seed": 0,
    "time_stamp": "2024-04-28-00-57-42",
    "global_epoch": 2,
    "global_batch": 131072,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "inverse",
    "beta": 0.9,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        424,
        424
    ],
    "input_dim": 3,
```

```
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 64,
    "projection_dim": 64,
    "num_heads": 8,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        424,
        424
    ]
}
```


Notes:

really happy with how this is performing, could be very promising if i have way more data issue is i can only augment my data so much as it has diminishing returns, that and it could lead over fitting. 50% accuracy is still way better than chance for 8 labels.

need to try different transformer hyperparameters, more attention heads and a larger projection dim might work better. still unsure on the number of patches to use and the image crop size to use.

tried using 256 patches on a 256x256 image -> patch size = 16x16 although was using different class labels so need to try this again and compare performance -> could be good as paper on ViT's "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" uses 16x16 patches sizes.

need to get a balance between number of patches & image size/crop size as larger image = more computational cost, larger patches = more spatial information per patch for the transformers to understand however there are less patches to be encoded into the projection dimension for the transformers to learn correlations between.

more attention heads could possibly allow for the model to be able to extract and pay attention to more features.

from my earlier experimentation - smaller models seemed to perform better than larger ones, as the larger ones seemed to underfit heavily as there simply is not enough training data, but the smaller models lack the complexity to fully capture and solve the problem.

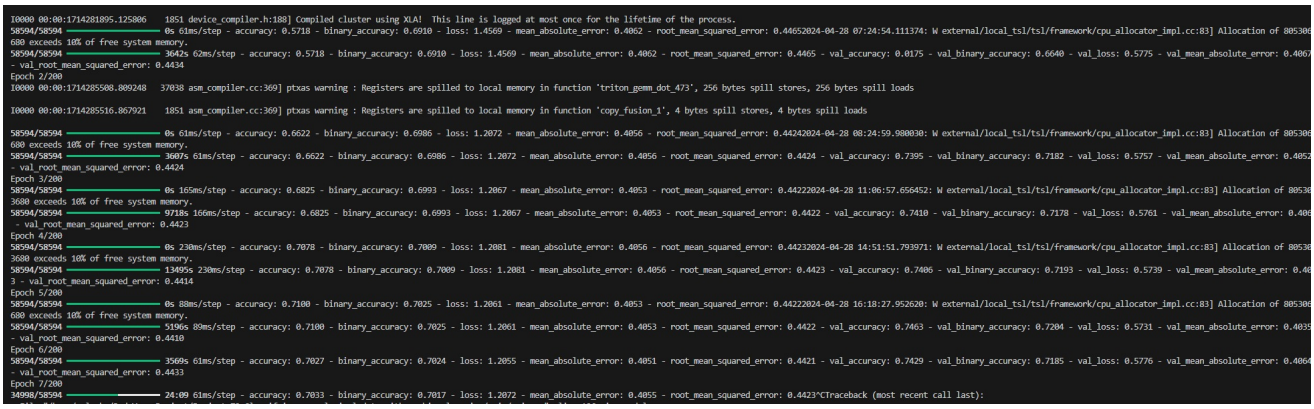
TODO:

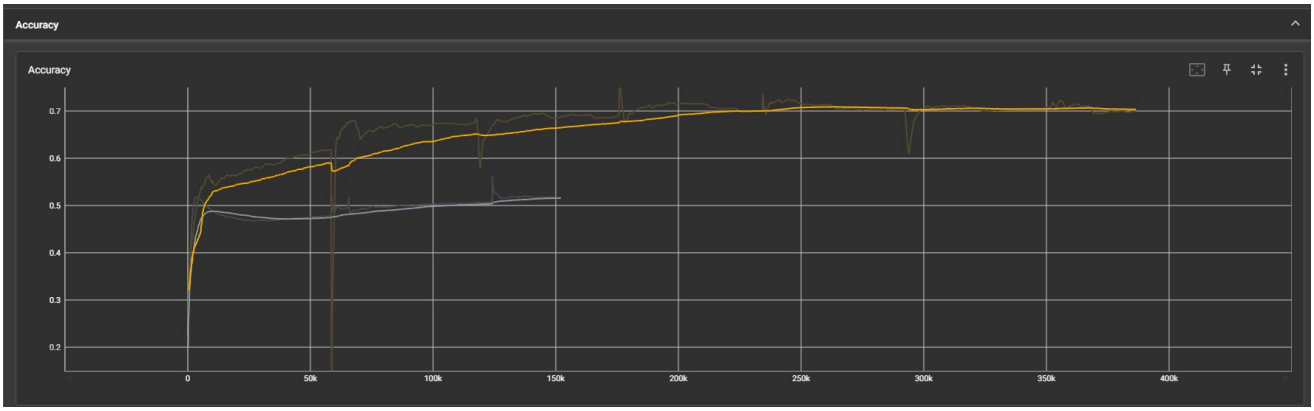
- Testing script, like main.py - loading of model & weights, performing test (should probably rename main.py to train.py)
- ability to see transformer model attention map on images
- ability to plot confusion/correlation matricies between predictions & truth labels
- metrics like precision, f1, recall, false positive, false negative, AUC of the ROC curve.

Malachy 27/04/2024 - transformer model with 256x256 cropped images and 256 16x16 patches

I have disabled dropout within the MLP's and attention heads for these test as I was experiencing underfitting:

can renable it when i find model architecture and hyperparameters that perform well and if I run into overfitting issues.





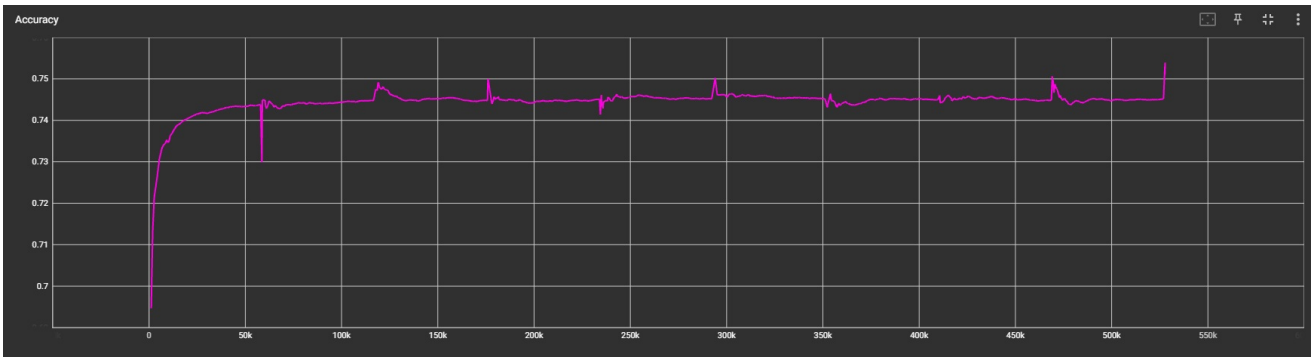
meta.json:

In [ ]:


```
{
"seed": 0,
"time_stamp": "2024-04-28-06-06-44",
"global_epoch": 6,
"global_batch": 360448,
"saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-lea
"save_step": 32768,
"root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
"dataset_root": "galaxyzoo2-dataset-augmented/",
"train_catalog": "combined-train_catalog.parquet",
"test_catalog": "combined-test_catalog.parquet",
"checkpoint_root": "checkpoints/",
"log_root": "logs/fit/",
"classes": "reduced",
"weight_mode": "inverse",
"beta": 0.9,
"build_paths": true,
"binarised_labels": true,
"ds_fraction": 1,
"val_fraction": 0.2,
"input_shape": [
    256,
    256
],
"input_dim": 3,
"summary": true,
"architecture": "ViT",
"learning_rate": 0.001,
"loss_function": "binary_crossentropy",
"activation_function": "sigmoid",
"weight_reg": null,
"num_patches": 256,
"projection_dim": 64,
"num_heads": 8,
"transformer_layers": 12,
"mlp_head_units": [
    2048,
    1024
],
"train_batch_size": 16,
"val_batch_size": 16,
"val_steps": 512,
"epochs": 200,
"steps_per_epoch": 58594,
"augmentation": true,
"crop": true,
"crop_size": [
    256,
    256
]
}
```


28/04/2024 - Malachy: 256x256 cropped images, 256 16x16 patches. adjusted the number of transformer heads from 8 -> 12. used effective number of samples weighting (unsure about how this performs) NOTE: (this was done with an incorrectly functioning version of effective number of samples weighting)

```
I0000 00:00:1714343962.653473  407536 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7370 - binary_accuracy: 0.7222 - loss: 0.0625 - mean_absolute_error: 0.3706 - root_mean_squared_error: 0.43342024-04-29 01:14:10.329221: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 5737s 97ms/step - accuracy: 0.7370 - binary_accuracy: 0.7222 - loss: 0.0625 - mean_absolute_error: 0.3706 - root_mean_squared_error: 0.4334 - val_accuracy: 0.7473 - val_binary_accuracy: 0.7256 - val_loss: 0.5509 - val_mean_absolute_error: 0.3680
- val_root_mean_squared_error: 0.4298
Epoch 2/200
I0000 00:00:1714349666.281262  429636 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'triton_gemm_dot_473', 256 bytes spill stores, 256 bytes spill loads

I0000 00:00:1714349675.212966  407540 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_51', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_50', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_49', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'copy_fusion_1', 4 bytes spill stores, 4 bytes spill loads

58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7441 - binary_accuracy: 0.7246 - loss: 0.0552 - mean_absolute_error: 0.3703 - root_mean_squared_error: 0.43042024-04-29 02:28:49.079422: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4479s 76ms/step - accuracy: 0.7441 - binary_accuracy: 0.7246 - loss: 0.0552 - mean_absolute_error: 0.3703 - root_mean_squared_error: 0.4304 - val_accuracy: 0.7450 - val_binary_accuracy: 0.7265 - val_loss: 0.5512 - val_mean_absolute_error: 0.3703
- val_root_mean_squared_error: 0.4299
Epoch 3/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7455 - binary_accuracy: 0.7248 - loss: 0.0552 - mean_absolute_error: 0.3700 - root_mean_squared_error: 0.43022024-04-29 03:43:05.769336: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4460s 76ms/step - accuracy: 0.7455 - binary_accuracy: 0.7248 - loss: 0.0552 - mean_absolute_error: 0.3700 - root_mean_squared_error: 0.4302 - val_accuracy: 0.7389 - val_binary_accuracy: 0.7233 - val_loss: 0.5538 - val_mean_absolute_error: 0.3715
- val_root_mean_squared_error: 0.4312
Epoch 4/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.7448 - binary_accuracy: 0.7243 - loss: 0.0552 - mean_absolute_error: 0.3705 - root_mean_squared_error: 0.43052024-04-29 04:57:05.336521: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4439s 76ms/step - accuracy: 0.7448 - binary_accuracy: 0.7243 - loss: 0.0552 - mean_absolute_error: 0.3705 - root_mean_squared_error: 0.4305 - val_accuracy: 0.7435 - val_binary_accuracy: 0.7259 - val_loss: 0.5514 - val_mean_absolute_error: 0.3699
- val_root_mean_squared_error: 0.4300
Epoch 5/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.7454 - binary_accuracy: 0.7248 - loss: 0.0552 - mean_absolute_error: 0.3702 - root_mean_squared_error: 0.43032024-04-29 06:10:58.014685: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4434s 76ms/step - accuracy: 0.7454 - binary_accuracy: 0.7248 - loss: 0.0552 - mean_absolute_error: 0.3702 - root_mean_squared_error: 0.4303 - val_accuracy: 0.7523 - val_binary_accuracy: 0.7303 - val_loss: 0.5471 - val_mean_absolute_error: 0.3692
- val_root_mean_squared_error: 0.4278
Epoch 6/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4442s 76ms/step - accuracy: 0.7456 - binary_accuracy: 0.7249 - loss: 0.0552 - mean_absolute_error: 0.3700 - root_mean_squared_error: 0.4302 - val_accuracy: 0.7490 - val_binary_accuracy: 0.7260 - val_loss: 0.5489 - val_mean_absolute_error: 0.3711
- val_root_mean_squared_error: 0.4290
Epoch 7/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4432s 76ms/step - accuracy: 0.7448 - binary_accuracy: 0.7247 - loss: 0.0552 - mean_absolute_error: 0.3702 - root_mean_squared_error: 0.4303 - val_accuracy: 0.7466 - val_binary_accuracy: 0.7245 - val_loss: 0.5532 - val_mean_absolute_error: 0.3701
- val_root_mean_squared_error: 0.4308
Epoch 8/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4443s 76ms/step - accuracy: 0.7451 - binary_accuracy: 0.7245 - loss: 0.0552 - mean_absolute_error: 0.3701 - root_mean_squared_error: 0.4303 - val_accuracy: 0.7482 - val_binary_accuracy: 0.7202 - val_loss: 0.5508 - val_mean_absolute_error: 0.3704
- val_root_mean_squared_error: 0.4297
Epoch 9/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4502s 77ms/step - accuracy: 0.7451 - binary_accuracy: 0.7246 - loss: 0.0552 - mean_absolute_error: 0.3702 - root_mean_squared_error: 0.4303 - val_accuracy: 0.7480 - val_binary_accuracy: 0.7248 - val_loss: 0.5526 - val_mean_absolute_error: 0.3715
- val_root_mean_squared_error: 0.4306
```



meta.json:

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-04-28-23-19-39",
    "global_epoch": 8,
    "global_batch": 527346,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": 0.9,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
```

```
    "crop_size": [
        256,
        256
    ]
}
```

Increasing the number of transformer heads has appeared to improve the models ability to classify quite well although it does slow down the training a bit.

29/04/2024 Malachy - corrected effective number of sampling weighting.

now will perform a training run of the CNN model to see how it perfoms.

In [ ]:
```
def get_feature_weights(self):
        """
        Get a dictionary containing the weights to be used in the loss function for each class label.

        returns:
        weight_dict (dict): a dictionary mapping of class indices (integers) to a weight (float)
        """
        return self.class_weights

    def __create_weights(self, pandas_dataframe):
        """
        create class weights for the dataset by finding the fraction of galaxies that have a specific feature (=

        args:
        pandas_dataframe (pd.DataFrame): the dataframe containing the class labels

        returns:
        weights (list): a list of the weights for each class label
        """
        weights = []

        if self.beta is None: # calculate beta based off the formula beta = (N - 1) / N (4. Class-Balanced Loss
            self.beta = (len(pandas_dataframe) - 1) / len(pandas_dataframe)

        for key in self.keys:
            num = (pandas_dataframe[key] == 1).sum()
            fraction = num / len(pandas_dataframe[key])

            if self.weight_mode == 'inverse':
                weight = 1 / fraction

            elif self.weight_mode == 'inverse-sqrt':
                weight = 1 / np.sqrt(fraction)

            # https://openaccess.thecvf.com/content_CVPR_2019/papers/Cui_Class-Balanced_Loss_Based_on_Effective_
            # 4. Class-Balanced Loss:
            elif self.weight_mode == 'effective_num_samples': # beta = 0 -> no reweighting. beta = 1 -> reweigh
                effective_num = (1 - self.beta**num) / (1 - self.beta)
                weight = - 1 / effective_num # 4.2. Class-Balanced Sigmoid Cross-Entropy Loss: CB = -1/weights

            elif self.weight_mode is None:
                weight = 1

            else:
                raise ValueError(f"\033[31mInvalid weight mode!: {self.weight_mode}\033[0m")

            weights.append(weight)

        if self.weight_mode == 'effective_num_samples': # 4. Class-Balanced Loss: sum of weights = num classes
            total = np.sum(weights)
            weights = [(weight / total) * len(self.keys) for weight in weights]

        return {idx: weight for idx, weight in enumerate(weights)}
```

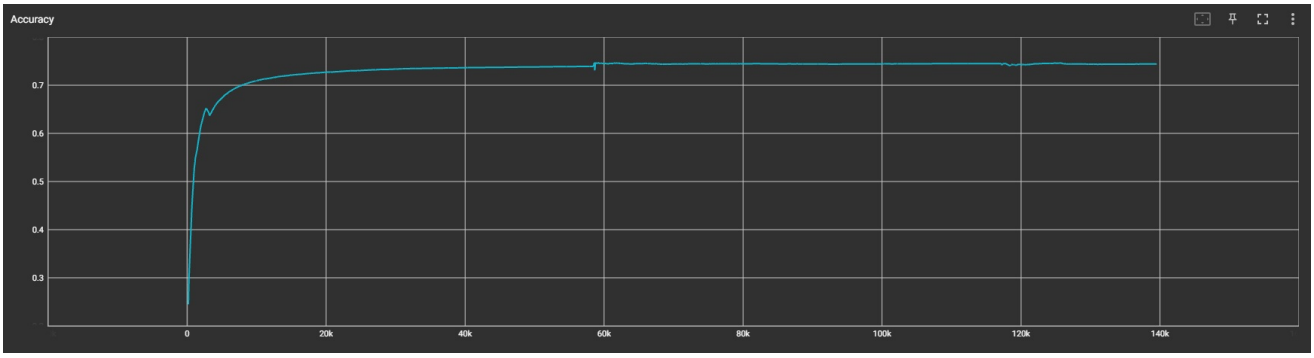
29/04/2024 Malachy -

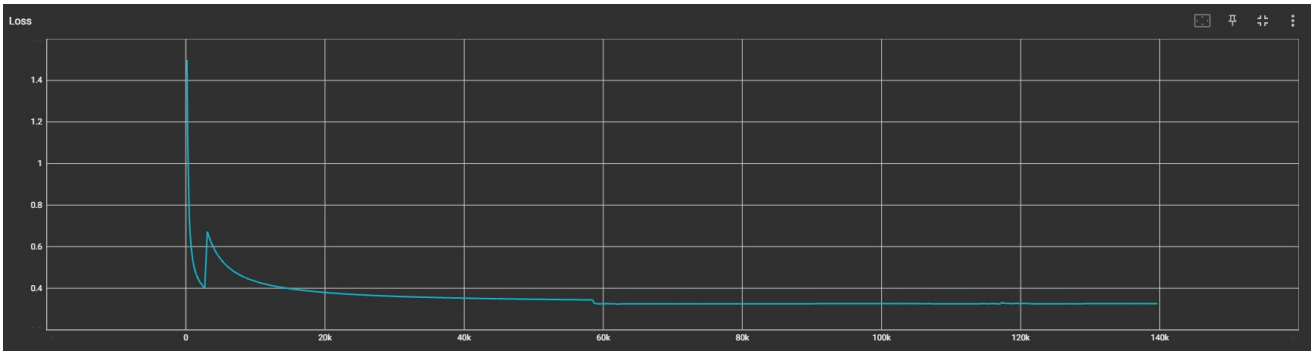
the weightings used for the "reduced" labels:

'smooth-or-featured-gz2_smooth_fraction', 'disk-edge-on-gz2_yes_fraction', 'has-spiral-arms-gz2_yes_fraction', 'bar-gz2_yes_fraction', 'how-rounded-gz2_round_fraction', 'how-rounded-gz2_in-between_fraction', 'how-rounded-gz2_cigar_fraction', 'bulge-gz2-yes_fraction'

loss function weights: {0: 0.4070826856251515, 1: 1.5601711800347748, 2: 0.8015602801702779, 3: 1.7976638308200183, 4: 0.7670370007813275, 5: 0.5288567578317966, 6: 1.4888872862120155, 7: 0.6487409785246386}

using the same transformer hyperparameters as the previous run.






```
I0000 00:00:1714398939.296838  622060 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.7163 - binary_accuracy: 0.7172 - loss: 0.3994 - mean_absolute_error: 0.3982 - root_mean_squared_error: 0.44112024-04-29 16:11:58.293070: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 4631s 78ms/step - accuracy: 0.7163 - binary_accuracy: 0.7172 - loss: 0.3994 - mean_absolute_error: 0.3982 - root_mean_squared_error: 0.4411 - val_accuracy: 0.7452 - val_binary_accuracy: 0.7203 - val_loss: 0.5675 - val_mean_absolute_error: 0.3983
 - val_root_mean_squared_error: 0.4380
Epoch 2/200
I0000 00:00:1714403544.828481  646416 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'triton_gemm_dot_473', 256 bytes spill stores, 256 bytes spill loads

I0000 00:00:1714403553.613222  622060 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_51', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_50', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_49', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'copy_fusion_1', 4 bytes spill stores, 4 bytes spill loads

58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.7445 - binary_accuracy: 0.7213 - loss: 0.3252 - mean_absolute_error: 0.3973 - root_mean_squared_error: 0.43742024-04-29 17:55:42.142050: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 805306
3680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 6226s 106ms/step - accuracy: 0.7445 - binary_accuracy: 0.7213 - loss: 0.3252 - mean_absolute_error: 0.3973 - root_mean_squared_error: 0.4374 - val_accuracy: 0.7472 - val_binary_accuracy: 0.7216 - val_loss: 0.5623 - val_mean_absolute_error: 0.3938
 - val_root_mean_squared_error: 0.4355
Epoch 3/200
22480/58594 ━━━━━━━━━━━━━━━━━━━━ 46:48 78ms/step - accuracy: 0.7439 - binary_accuracy: 0.7218 - loss: 0.3258 - mean_absolute_error: 0.3972 - root_mean_squared_error: 0.4374^CTraceback (most recent call last):
```

meta.json:

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-04-29-14-36-48",
    "global_epoch": 2,
    "global_batch": 131072,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
```

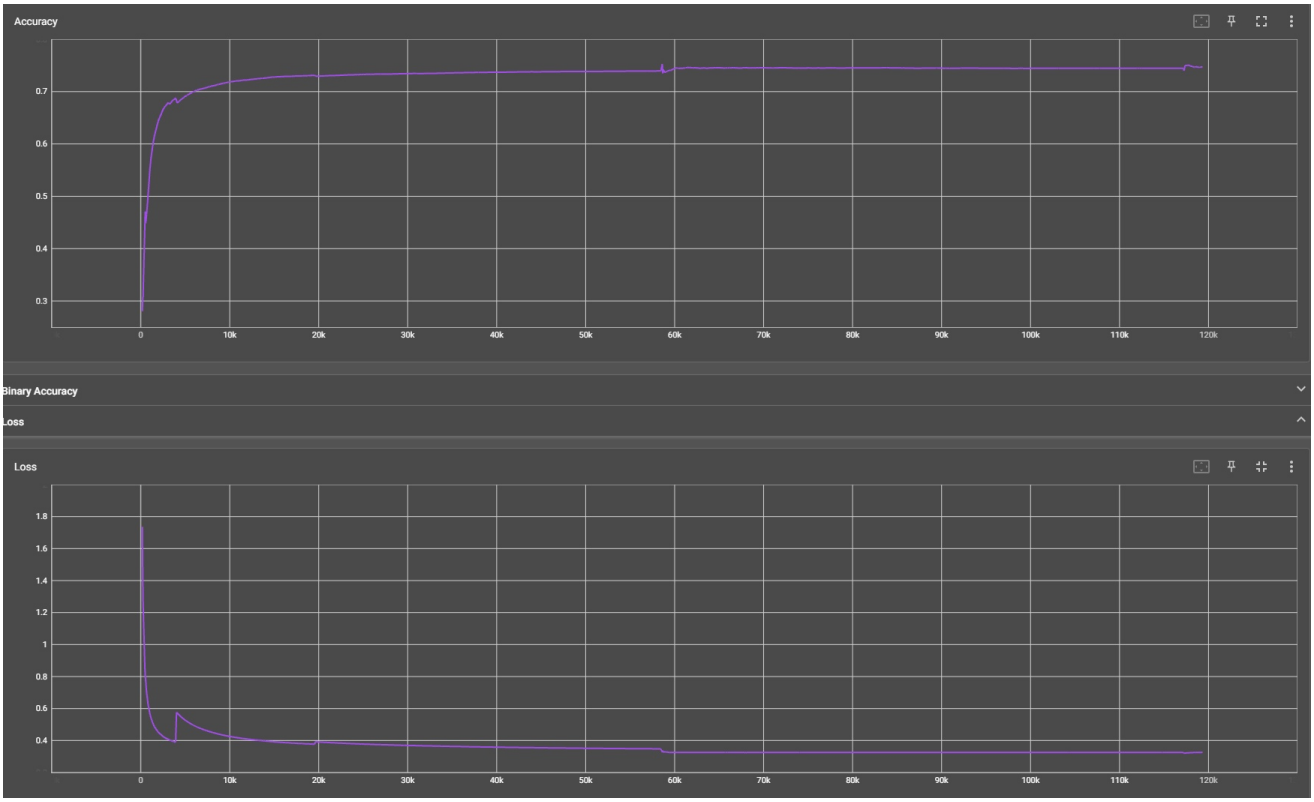
```
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```


using effective num_samples didnt change the accuracy or model performance noticably, made the learning more stable and less noisy compared to inverse weighting, so I will stick with it. reached about 75% val accuracy. hypothesis: more attention heads results in better classification performance. will now try 16 heads.

- ideas:
- find transformer hyper parameters that perform well on this reduced problem.
- train transformer model with dropout re-added then:
- try apply transfer learning to this model -> try get it to classify all the possible features.
- or have ensemble of models this one for the main general features, 2nd for finer detail features, i.e spiral arm amount/winding.

16 heads:


```
I0000 00:00:1714412913.997415  675632 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7212 - binary_accuracy: 0.7180 - loss: 0.4017 - mean_absolute_error: 0.3981 - root_mean_squared_error: 0.44102024-04-29 20:19:57.451046: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 5529s 94ms/step - accuracy: 0.7212 - binary_accuracy: 0.7180 - loss: 0.4017 - mean_absolute_error: 0.3981 - root_mean_squared_error: 0.4410 - val_accuracy: 0.7446 - val_binary_accuracy: 0.7196 - val_loss: 0.5689 - val_mean_absolute_error: 0.4002
- val_root_mean_squared_error: 0.4384
Epoch 2/200
I0000 00:00:1714418413.986861  700966 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'triton_gemm_dot_473', 256 bytes spill stores, 256 bytes spill loads

I0000 00:00:1714418422.457626  675634 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_46', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_45', 500 bytes spill stores, 500 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'copy_fusion_1', 4 bytes spill stores, 4 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_38', 500 bytes spill stores, 500 bytes spill loads

58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7447 - binary_accuracy: 0.7210 - loss: 0.3256 - mean_absolute_error: 0.3977 - root_mean_squared_error: 0.43762024-04-29 21:51:49.408420: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063
680 exceeds 10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 5515s 94ms/step - accuracy: 0.7447 - binary_accuracy: 0.7210 - loss: 0.3256 - mean_absolute_error: 0.3977 - root_mean_squared_error: 0.4376 - val_accuracy: 0.7454 - val_binary_accuracy: 0.7213 - val_loss: 0.5681 - val_mean_absolute_error: 0.3993
- val_root_mean_squared_error: 0.4383
```

comparable performance to 12, but took longer to train. approx 90m/s a step. transformer models seem to learn pretty quickly but seem to reach a maximum and level out. (that or the learning rate is so small after due to the lack of data)

instead will try more transformer layers/larger projection dim

meta.json:

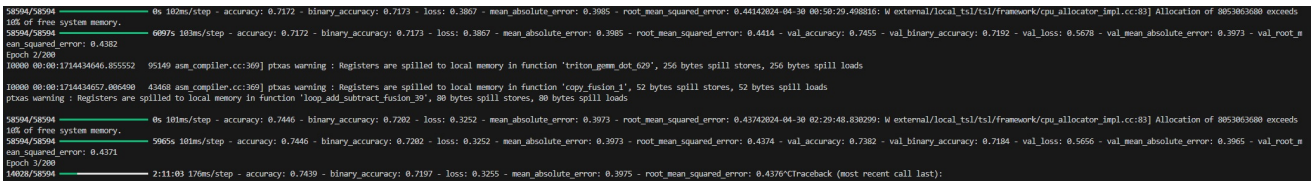
In [ ]:

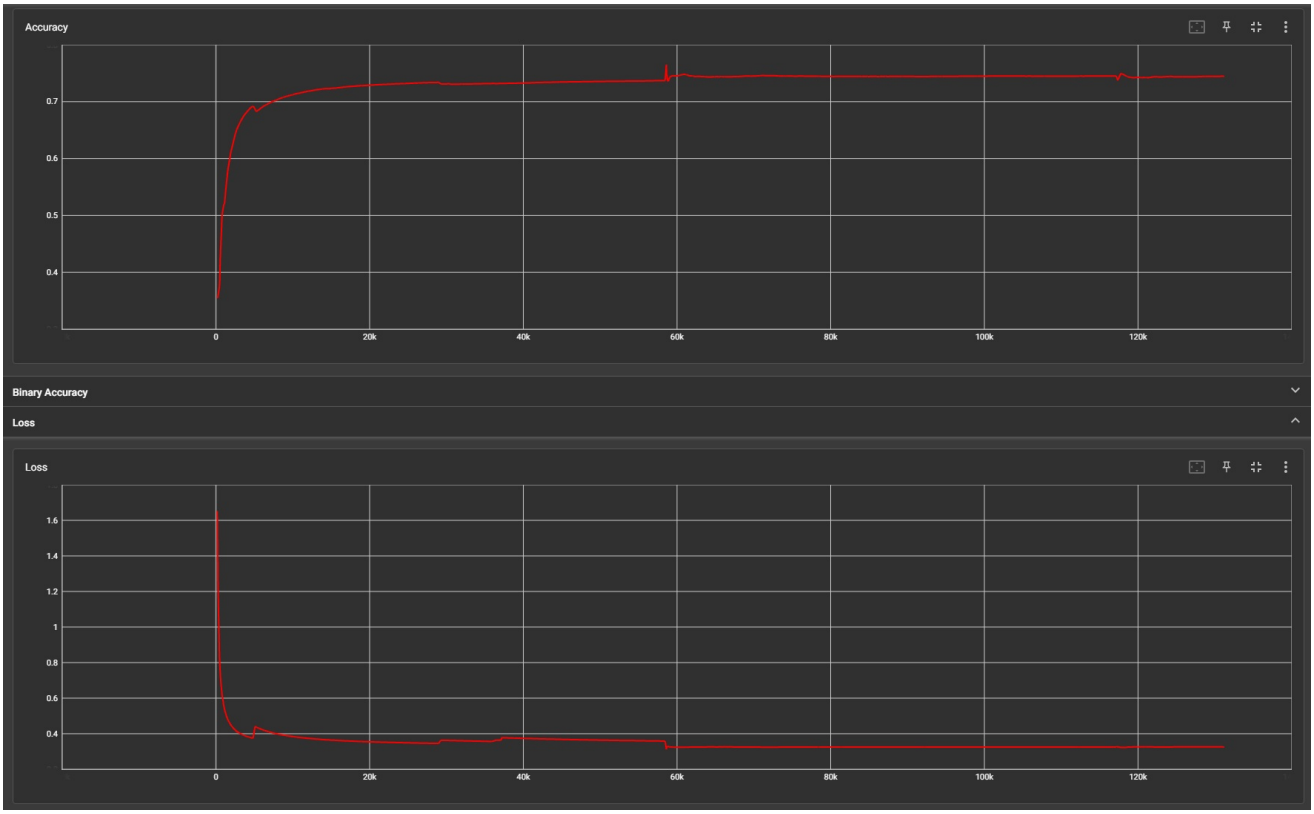

```
{
    "seed": 0,
    "time_stamp": "2024-04-29-18-29-31",
    "global_epoch": 1,
    "global_batch": 117188,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
```

```
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 16,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```


- 12 transformer layers -> 16 transformer layers
- 16 attention heads -> 12 attention heads

30/04/2024 Malachy -





12 -> 16 transformer layers didnt change the performance of the network in a significant way.

meta.json

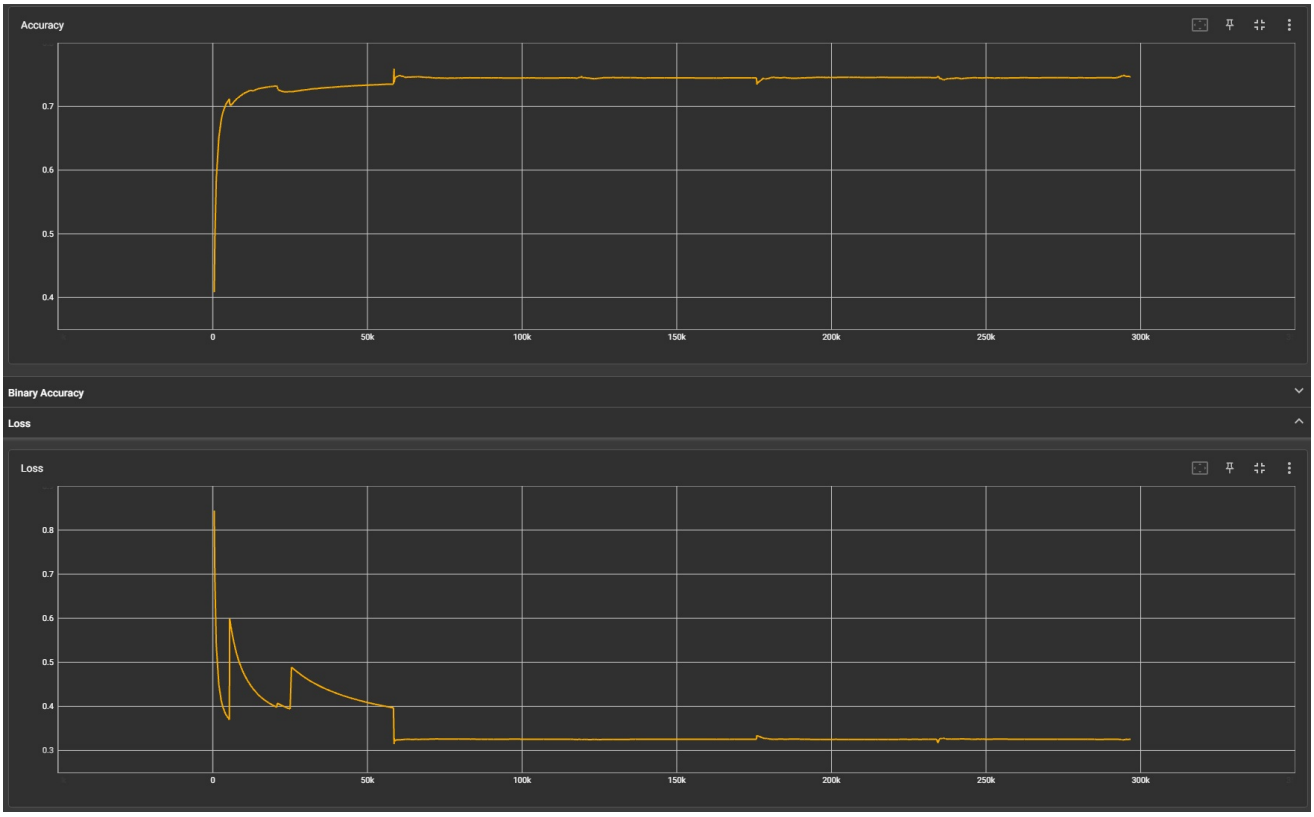
In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-04-29-22-49-37",
    "global_epoch": 2,
    "global_batch": 131072,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 16,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
```

```
        256,
        256
    ]
}
```

will try 16 attention heads and 16 transformer layers. can then experiment with different projection dimensions

```
I0000 00:00:1714444574.248264  149558 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step - accuracy: 0.7184 - binary_accuracy: 0.7181 - loss: 0.4504 - mean_absolute_error: 0.3986 - root_mean_squared_error: 0.44132024-04-30 05:38:55.400686: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063680 exceeds
10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 7417s 126ms/step - accuracy: 0.7184 - binary_accuracy: 0.7181 - loss: 0.4504 - mean_absolute_error: 0.3986 - root_mean_squared_error: 0.4413 - val_accuracy: 0.7465 - val_binary_accuracy: 0.7191 - val_loss: 0.5675 - val_mean_absolute_error: 0.3979 - val_root_m
ean_squared_error: 0.4381
Epoch 2/200
I0000 00:00:1714451952.287811  203751 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'triton_gemm_dot_629', 256 bytes spill stores, 256 bytes spill loads

I0000 00:00:1714451962.191402  149559 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'copy_fusion_1', 52 bytes spill stores, 52 bytes spill loads

58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step - accuracy: 0.7451 - binary_accuracy: 0.7203 - loss: 0.3254 - mean_absolute_error: 0.3975 - root_mean_squared_error: 0.43752024-04-30 07:39:36.154499: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063680 exceeds
10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 7241s 123ms/step - accuracy: 0.7451 - binary_accuracy: 0.7203 - loss: 0.3254 - mean_absolute_error: 0.3975 - root_mean_squared_error: 0.4375 - val_accuracy: 0.7429 - val_binary_accuracy: 0.7199 - val_loss: 0.5688 - val_mean_absolute_error: 0.3987 - val_root_m
ean_squared_error: 0.4386
Epoch 3/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step - accuracy: 0.7448 - binary_accuracy: 0.7203 - loss: 0.3253 - mean_absolute_error: 0.3974 - root_mean_squared_error: 0.43752024-04-30 09:39:32.208581: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063680 exceeds
10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 7196s 123ms/step - accuracy: 0.7448 - binary_accuracy: 0.7203 - loss: 0.3253 - mean_absolute_error: 0.3974 - root_mean_squared_error: 0.4375 - val_accuracy: 0.7422 - val_binary_accuracy: 0.7195 - val_loss: 0.5691 - val_mean_absolute_error: 0.3990 - val_root_m
ean_squared_error: 0.4389
Epoch 4/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step - accuracy: 0.7453 - binary_accuracy: 0.7206 - loss: 0.3254 - mean_absolute_error: 0.3974 - root_mean_squared_error: 0.43742024-04-30 11:39:39.148913: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063680 exceeds
10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 7209s 123ms/step - accuracy: 0.7453 - binary_accuracy: 0.7206 - loss: 0.3254 - mean_absolute_error: 0.3974 - root_mean_squared_error: 0.4374 - val_accuracy: 0.7463 - val_binary_accuracy: 0.7197 - val_loss: 0.5664 - val_mean_absolute_error: 0.3980 - val_root_m
ean_squared_error: 0.4375
Epoch 5/200
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step - accuracy: 0.7449 - binary_accuracy: 0.7202 - loss: 0.3255 - mean_absolute_error: 0.3977 - root_mean_squared_error: 0.43752024-04-30 14:12:22.153803: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 8053063680 exceeds
10% of free system memory.
58594/58594 ━━━━━━━━━━━━━━━━━━━━ 9162s 156ms/step - accuracy: 0.7449 - binary_accuracy: 0.7202 - loss: 0.3255 - mean_absolute_error: 0.3977 - root_mean_squared_error: 0.4375 - val_accuracy: 0.7389 - val_binary_accuracy: 0.7160 - val_loss: 0.5708 - val_mean_absolute_error: 0.4001 - val_root_m
ean_squared_error: 0.4397
```



meta.json :

In [ ]:

```
{
    "seed": 0,
    "time_stamp": "2024-04-30-03-16-03",
    "global_epoch": 5,
    "global_batch": 294912,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
```

```
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 16,
    "transformer_layers": 16,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```

again no real substantial change in model performance.

so 12 transformer heads and 12 transformer layers seems to be an ideal amount that balances model performance and training time

Summary of final architectures tested

In [ ]:
```python
def __init_network(self) -> tf.keras.Model:
        """
        Initalise a Convolutional Neural Network of a specified architecture using the Keras Sequential API.

        returns:
        network (tf.keras.Model): the Convolutional Neural Network
        """
        # Weight regularization
        if self.weight_reg is None:
            wr = None
        else:
            wr = tf.keras.regularizers.l2(self.weight_reg)

        match self.architecture:
            case 'ViT':
                transformer_units = [
                    self.projection_dim * 2,
                    self.projection_dim,
                ]

                inputs = keras.Input(shape = (self.input_x, self.input_y, self.input_dim))
                augs = augmentation_layers(self.input_x, self.input_y)
                augs = augs(inputs)
                patches = Patches(self.input_x, self.num_patches)(augs)
                encoded_patches = PatchEncoder(self.num_patches, self.projection_dim)(patches)

                for _ in range(self.transformer_layers):
                    x1 = LayerNormalization(epsilon = 1e-6)(encoded_patches) # layer normalisation 1
                    attention_output = MultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_

                    # skip connection 1
                    x2 = layers.Add()([attention_output, encoded_patches])
                    x3 = LayerNormalization(epsilon = 1e-6)(x2) # layer normalisation 2

                    x3 = mlp(x3, hidden_units = transformer_units, dropout_rate = 0.1)

                    # skip connection 2
                    encoded_patches = layers.Add()([x3, x2])

                # create a [batch_size, projection_dim] tensor.
                representation = LayerNormalization(epsilon = 1e-6)(encoded_patches)
                representation = Flatten()(representation)
                representation = Dropout(0.5)(representation)

                features = mlp(representation, hidden_units = self.mlp_head_units, dropout_rate = 0.5)
                logits = Dense(self.label_length, activation = self.activation_function, kernel_regularizer = w

                network = Model(inputs = inputs, outputs = logits)

                self.__compile_network(network)

            case 'CvT':
```

```
                units = [
                    self.projection_dim * 2,
                    self.projection_dim,
                ]

                inputs = keras.Input(shape=(self.input_x, self.input_y, self.input_dim))
                convolutional_tokeniser = ConvolutionalEmbedding(patch_size = 3, embed_dim = self.projection_di
                encoded_patches = convolutional_tokeniser(inputs)

                TransformerBlock1 = Sequential([
                    LayerNormalization(epsilon = 1e-6),
                    SeparableConvMultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim, 
                    LayerNormalization(epsilon = 1e-6),
                    MLP(hidden_units = units, dropout_rate = 0.1),
                ], name ='Transformer_Block_1'
                )

                TransformerBlock2 = Sequential([
                    LayerNormalization(epsilon = 1e-6),
                    SeparableConvMultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim, 
                    LayerNormalization(epsilon = 1e-6),
                    MLP(hidden_units = units, dropout_rate = 0.1),
                ], name ='Transformer_Block_2'
                )

                TransformerBlock3 = Sequential([
                    AddCLSToken(cls_token = tf.zeros(shape=(1, self.projection_dim))),
                    LayerNormalization(epsilon = 1e-6),
                    SeparableConvMultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim, 
                    LayerNormalization(epsilon = 1e-6),
                    MLP(hidden_units = units, dropout_rate = 0.1)
                ], name ='Transformer_Block_3'
                )

                x = TransformerBlock1(encoded_patches)
                x = TransformerBlock2(x)
                x = TransformerBlock3(x)

                representation = LayerNormalization(epsilon=1e-5)(x)
                representation = Flatten()(representation)
                representation = Dropout(0.5)(representation)

                features = mlp(x = representation, hidden_units = self.mlp_head_units, dropout_rate = 0.5)
                logits = Dense(self.label_length, activation = self.activation_function, kernel_regularizer = w

                network = Model(inputs = inputs, outputs = logits)

                self.__compile_network(network)

            case 'CCT':
                transformer_units = [
                self.projection_dim,
                self.projection_dim,
                ]

                stochastic_depth_rate = 0.1

                inputs = layers.Input(shape = (self.input_x, self.input_y, self.input_dim))

                covolutional_tokeniser = ConvolutionalTokeniser(kernel_size = 3, stride = 1, padding = 1, poolin
                encoded_patches = covolutional_tokeniser(inputs)

                if self.positional_embedding:
                    sequence_length = encoded_patches.shape[1]
                    encoded_patches = PositionEmbedding(sequence_length = sequence_length)(encoded_patches)

                dpr = [x for x in np.linspace(0, stochastic_depth_rate, self.transformer_layers)]  # stochastic

                for i in range(self.transformer_layers):
                    x1 = LayerNormalization(epsilon = 1e-6)(encoded_patches) # normalisation layer 1
                    attention_output = MultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_

                    # skip connection 1
                    attention_output = StochasticDepth(dpr[i])(attention_output)
                    x2 = layers.Add()([attention_output, encoded_patches])
                    x3 = LayerNormalization(epsilon = 1e-6)(x2) # normalisation layer 2
                    x3 = mlp(x3, hidden_units = transformer_units, dropout_rate = 0.1)

                    # skip connection 2
                    x3 = StochasticDepth(dpr[i])(x3)
                    encoded_patches = layers.Add()([x3, x2])

                # Apply sequence pooling.
```

```
                representation = LayerNormalization(epsilon=1e-5)(encoded_patches)
                weighted_representation = SequencePooling()(representation)
                weighted_representation = mlp(x = weighted_representation, hidden_units = self.mlp_head_units, 

                logits = Dense(self.label_length, activation = self.activation_function, kernel_regularizer = w
                network = keras.Model(inputs=inputs, outputs=logits)

                self.__compile_network(network)

            case 'ResNet50':
                ResNet50_model = ResNet50(include_top = False, weights = 'imagenet', input_shape = (self.input_
                ResNet50_output = Flatten()(ResNet50_model.output)
                Dense_layer = Dense(1000, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", kerne
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = ResNet50_model.input, outputs = Classifier)

                for layer in network.layers:
                        layer.trainable = True

                self.__compile_network(network)

            case 'ResNet50V2':
                ResNet50V2_model = ResNet50V2(include_top = False, weights = 'imagenet', input_shape = (self.inp
                ResNet50V2_output = Flatten()(ResNet50V2_model.output)
                Dense_layer = Dense(1000, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", kerne
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = ResNet50V2_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case 'VGG16':
                VGG16_model = VGG16(include_top = False, weights = 'imagenet', input_shape = (self.input_x, sel
                VGG16_output = Flatten()(VGG16_model.output)
                Dense_layer_1 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", ker
                Dense_layer_2 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_2", ker
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = VGG16_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case 'VGG19':
                VGG19_model = VGG19(include_top = False, weights = 'imagenet', input_shape = (self.input_x, sel
                VGG19_output = Flatten()(VGG19_model.output)
                Dense_layer_1 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", ker
                Dense_layer_2 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_2", ker
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = VGG19_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case 'MobileNetV3Small':
                    MobileNetV3Small_model = MobileNetV3Small(include_top = False, weights = 'imagenet', input_
                    MobileNetV3Small_output = Flatten()(MobileNetV3Small_model.output)
                    Dense_layer_1 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1",
                    Dense_layer_2 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_2",
                    Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regular

                    network = Model(inputs = MobileNetV3Small_model.input, outputs = Classifier)

                    for layer in network.layers:
                        layer.trainable = True

                    self.__compile_network(network)

            case 'MobileNetV3Large':
                MobileNetV3Large_model = MobileNetV3Large(include_top = False, weights = 'imagenet', input_shap
                MobileNetV3Large_output = Flatten()(MobileNetV3Large_model.output)
                Dense_layer_1 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", ker
                Dense_layer_2 = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_2", ker
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer
```

```
                network = Model(inputs = MobileNetV3Large_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case 'Xception':
                Xception_model = Xception(include_top = False, weights = 'imagenet', input_shape = (self.input_
                Xception_output = Flatten()(Xception_model.output)
                Dense_layer = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", kerne
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = Xception_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case 'InceptionV3':
                InceptionV3_model = InceptionV3(include_top = False, weights = 'imagenet', input_shape = (self.
                InceptionV3_output = Flatten()(InceptionV3_model.output)
                Dense_layer = Dense(4096, activation = 'relu', kernel_regularizer = wr, name = "Dense_1", kerne
                Classifier = Dense(self.label_length, activation = self.activation_function, kernel_regularizer

                network = Model(inputs = InceptionV3_model.input, outputs = Classifier)

                for layer in network.layers:
                    layer.trainable = True

                self.__compile_network(network)

            case _:
                raise ValueError(f"\033[31mInvalid architecture parameter provided: {self.architecture} \033[31m

        if (self.display_summary):
            network.summary()

        return network

    def __compile_network(self, network):
        """
        Compiles the optimiser of a network.
        If the network is being loaded from a checkpoint, we overwrite the randomly initialised weights with the

        args:
        network (tf.keras.Model): the network to compile
        """
        if self.saved_weights_path:
            self.load(network)
            print(f"\033[32mLoaded save state [{self.saved_weights_path}]! \033[0m")

        network.compile(loss = self.loss_function, optimizer = tf.keras.optimizers.Adam(learning_rate = self.lea
```

FINAL RESULTS

01/05/2024 - Malachy:

Pre trained CNN training. tried VGG19 & ResNet50V2 on different sets of labels

VGG19:

labels = 'all-features'

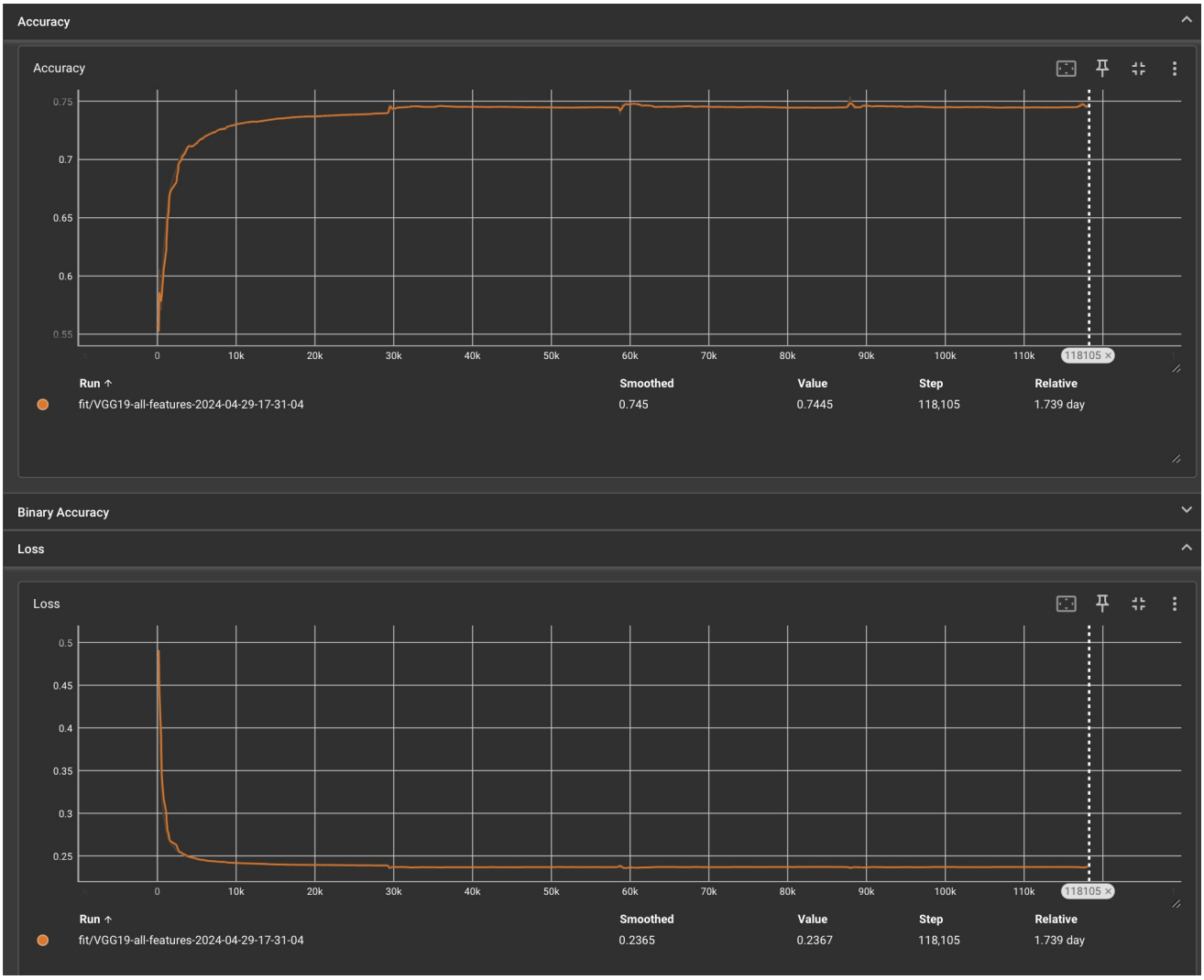
In [ ]:

```
"""
(base) malachy@Malachys-MacBook-Pro ~ % conda activate projectenv
(projectenv) malachy@Malachys-MacBook-Pro ~ % cd "/Users/malachy/Documents/3rd Year Project/Project-72-Classify
(projectenv) malachy@Malachys-MacBook-Pro main % python main.py --save_dir "/Users/malachy/Documents/3rd Year P
Reading json from: [/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-mac
Group sizes: [1 1 1 1 3 3 1 3 6 1]
Initialising dataset [/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-ma
Building file paths... (This may take a while)
2024-04-30 17:42:00.393866: I metal_plugin/src/device/metal_device.cc:1154] Metal device set to: Apple M3 Pro
2024-04-30 17:42:00.393890: I metal_plugin/src/device/metal_device.cc:296] systemMemory: 18.00 GB
2024-04-30 17:42:00.393894: I metal_plugin/src/device/metal_device.cc:313] maxCacheSize: 6.00 GB
2024-04-30 17:42:00.394092: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] (
2024-04-30 17:42:00.394105: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] (
Binarising labels... (this may take a while)
Labels binarised.
```

```
loss function weights: {0: 0.3257563781130054, 1: 1.2484741568960998, 2: 0.6414248001447729, 3: 1.4385259140593(
Dataset initialised.
2024-04-30 17:53:02.489060: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117] Plugi
2024-04-30 17:53:02.543255: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting w
Loaded training state: global_epoch = 1, global_train_batch = 58594
Loading weights and biases from [/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-
Loaded weights and biases!
Loaded save state [/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-mach
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_max_pooling2d (GlobalMaxPooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
Dense_1 (Dense)	(None, 4096)	2,101,248
Dense_2 (Dense)	(None, 4096)	16,781,312
Predictions (Dense)	(None, 21)	86,037

```
 Total params: 38,992,981 (148.75 MB)
 Trainable params: 38,992,981 (148.75 MB)
 Non-trainable params: 0 (0.00 B)
WARNING:tensorflow:AutoGraph could not transform <bound method DataFrame.fetch_image_label_pair of <dataframe.da
Cause: mangled names are not yet supported
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Epoch 2/200
29297/29297 ━━━━━━━━━━━━━━━━━━━━ 30003s 1s/step - accuracy: 0.7453 - binary_accuracy: 0.7341 - loss: 0.2363 - m
Epoch 3/200
29297/29297 ━━━━━━━━━━━━━━━━━━━━ 32495s 1s/step - accuracy: 0.7451 - binary_accuracy: 0.7339 - loss: 0.2363 - m
"""
```



meta.json:

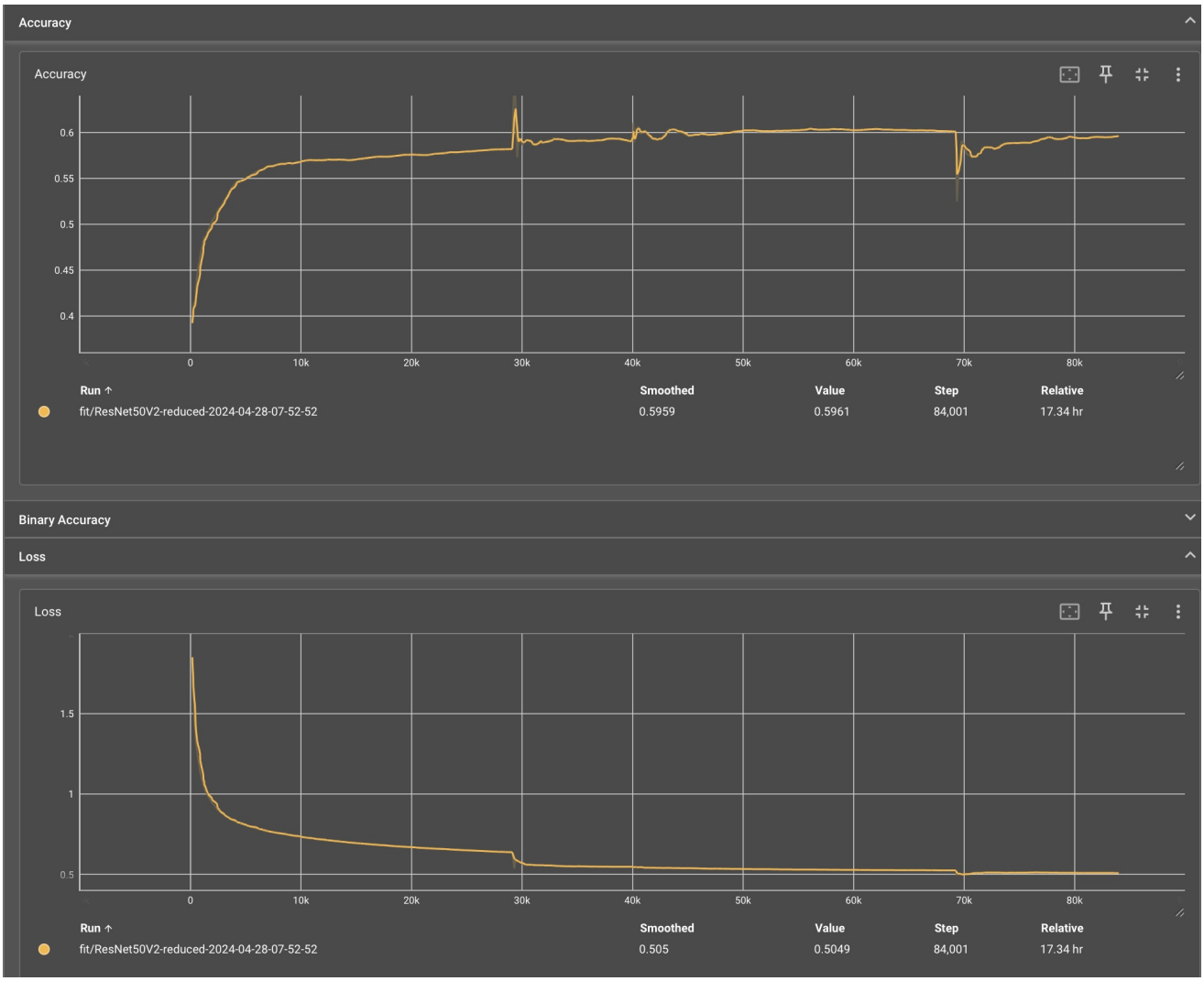
In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-04-29-17-31-04",
    "global_epoch": 2,
    "global_batch": 117188,
    "saved_weights_path": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-w
    "save_step": 8192,
    "root": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-le
    "dataset_root": "galaxyzoo2-dataset-no-translation-scale/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "all-features",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        224,
        224
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "VGG19",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
```

```
    "train_batch_size": 32,
    "val_batch_size": 32,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 29297,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        224,
        224
    ]
}
```


ResNet50V2:

labels = 'reduced'



meta.json:

In [ ]:

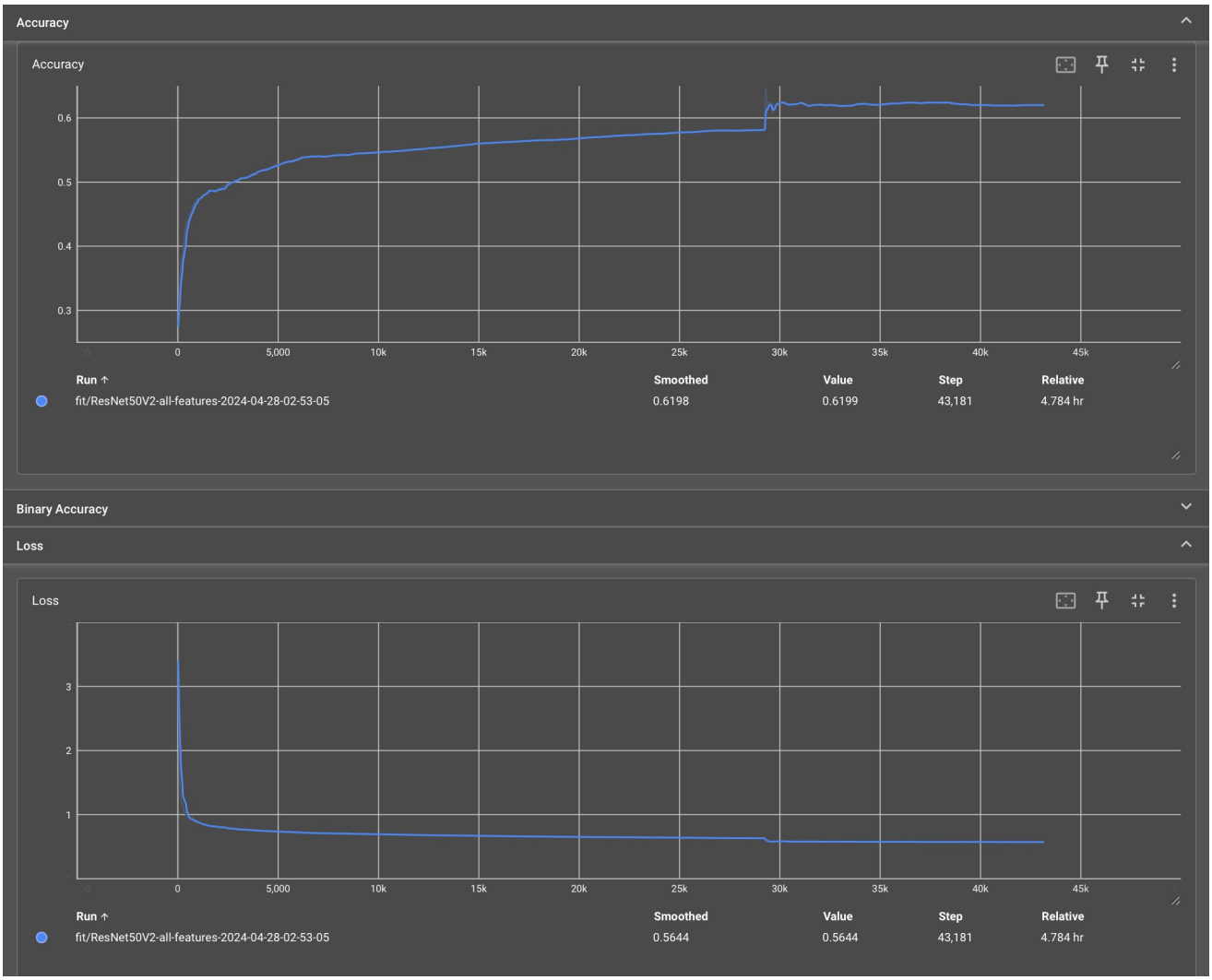

```
{
    "seed": 0,
    "time_stamp": "2024-04-28-07-52-52",
    "global_epoch": 2,
    "global_batch": 80000,
    "saved_weights_path": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-w
    "save_step": 10000,
    "root": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-le
    "dataset_root": "galaxyzoo2-dataset-no-translation-scale/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "inverse",
    "beta": 0.9,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        224,
```

```
        224
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ResNet50V2",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 64,
    "projection_dim": 64,
    "num_heads": 8,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 29297,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        224,
        224
    ]
}
```


worse performance than transformer model.

ResNet50V2:

labels = 'all-features'



meta.json:

In [ ]:

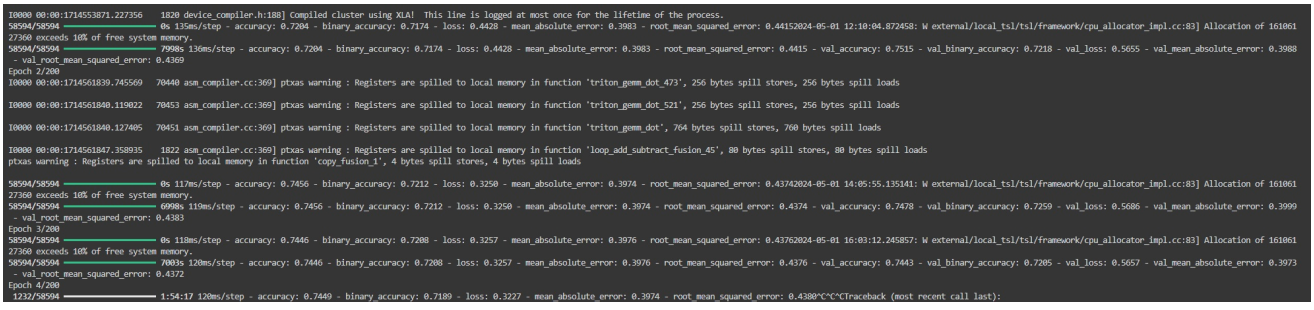

```
{
    "seed": 0,
    "time_stamp": "2024-04-28-02-53-05",
```

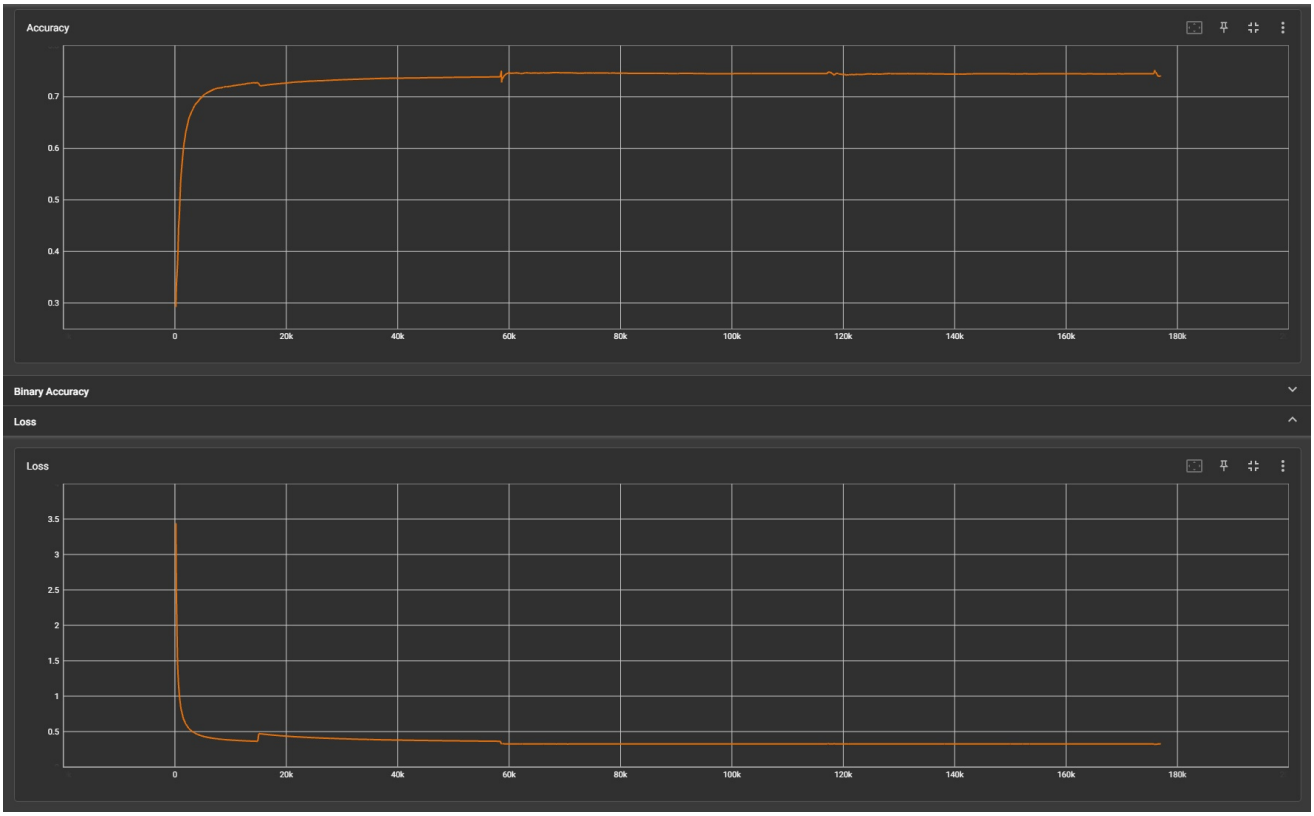
```
    "global_epoch": 1,
    "global_batch": 40000,
    "saved_weights_path": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-w
    "save_step": 10000,
    "root": "/Users/malachy/Documents/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-le
    "dataset_root": "galaxyzoo2-dataset-no-translation-scale/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "all-features",
    "weight_mode": "inverse",
    "beta": 0.9,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        224,
        224
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ResNet50V2",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 64,
    "projection_dim": 64,
    "num_heads": 8,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 29297,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        224,
        224
    ]
}
```


worse performance than VGG19

FINAL RESULTS (TRANSFORMERS)

tried increasing projection dim of ViT from 64 to 128:





no apparent change in model performance compared to previous attempts.

meta.json :

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-05-01-09-39-50",
    "global_epoch": 2,
    "global_batch": 175782,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 128,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
```

```
        256,
        256
    ]
}
```


Malachy -

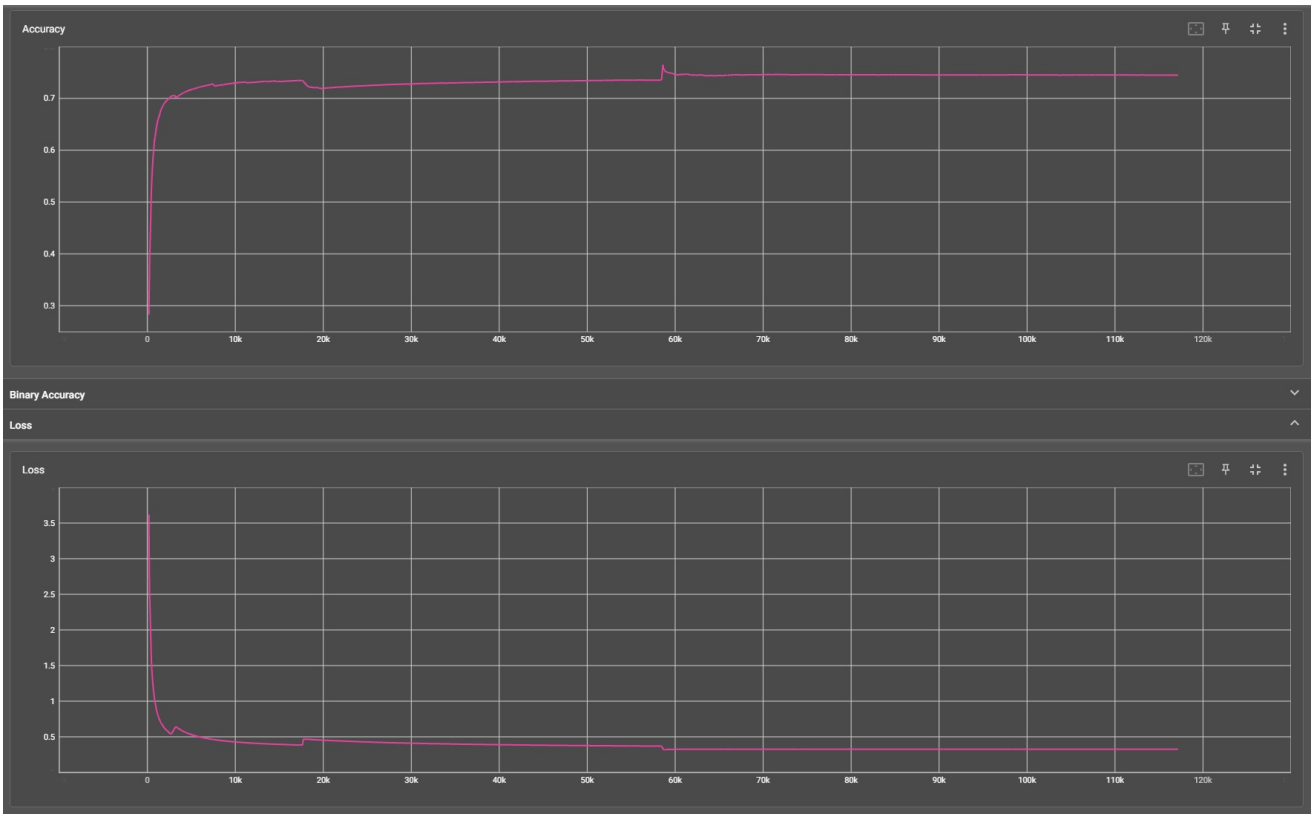
ViT:

increased mlp units from:

[2048, 1024]

to:

[4096, 2048]



had no noticeable affect on model performance.

meta.json :

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-05-01-17-02-45",
    "global_epoch": 1,
    "global_batch": 117188,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
```

```
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        4096,
        2048
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```

Malachy -

- CVT model:

In [ ]:

```
# credit for implementation: https://keras.io/examples/vision/cct/
class ConvolutionalTokeniser(layers.Layer):
    def __init__(self, kernel_size, stride, padding, pooling_kernel_size, pooling_stride, num_conv_layers, num_

        super().__init__()
        self.positional_embedding = positional_embedding
        self.conv_model = keras.Sequential()

        for i in range(num_conv_layers):

            self.conv_model.add(layers.Conv2D(num_output_channels[i], kernel_size, stride, padding = "valid", us
            self.conv_model.add(layers.ZeroPadding2D(padding))
            self.conv_model.add(layers.MaxPooling2D(pooling_kernel_size, pooling_stride, "same"))

    def call(self, images):
        outputs = self.conv_model(images)
        # After passing the images through the convolutional tokeniser the spatial dimensions are flattened to
        reshaped = keras.ops.reshape(
            outputs,
            (
                -1,
                keras.ops.shape(outputs)[1] * keras.ops.shape(outputs)[2],
                keras.ops.shape(outputs)[-1],
            ),
        )
        return reshaped

class PositionEmbedding(keras.layers.Layer):
    def __init__(self, sequence_length, initializer = "glorot_uniform"):
        super().__init__()

        if sequence_length is None:
            raise ValueError(f"\033[31m`sequence_length` must be an Integer! Received: `{type(sequence_length)}

        self.sequence_length = int(sequence_length)
        self.initializer = keras.initializers.get(initializer)

    def get_config(self):
        config = super().get_config()
        config.update({"sequence_length": self.sequence_length, "initializer": keras.initializers.serialize(sel
        return config

    def build(self, input_shape):

        feature_size = input_shape[-1]
        self.position_embeddings = self.add_weight(name = "embeddings", shape = [self.sequence_length, feature_s

        super().build(input_shape)

    def call(self, inputs, start_index=0):
        shape = keras.ops.shape(inputs)
        feature_length = shape[-1]
        sequence_length = shape[-2]
        # trim to match the length of the input sequence, as it might be less than the sequence_length of the la
        position_embeddings = keras.ops.convert_to_tensor(self.position_embeddings)
        position_embeddings = keras.ops.slice(position_embeddings, (start_index, 0), (sequence_length, feature_
```

```
        return keras.ops.broadcast_to(position_embeddings, shape)

    def compute_output_shape(self, input_shape):
        return input_shape
```

In [ ]:
```
case 'CvT':
    transformer_units = [
        self.projection_dim * 2,
        self.projection_dim,
    ]

    inputs = keras.Input(shape = (self.input_x, self.input_y, self.input_dim))
    covolutional_tokeniser = ConvolutionalTokeniser(self, kernel_size = 3, stride = 1, padding = 1, pooling_ker
    encoded_patches = covolutional_tokeniser(inputs)

    if self.positional_embedding:
        sequence_length = encoded_patches.shape[1]
        encoded_patches = PositionEmbedding(sequence_length)(encoded_patches)

    for i in range(self.transformer_layers):
        x1 = LayerNormalization(epsilon = 1e-6)(encoded_patches) # normalisation layer 1
        attention_output = MultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim, dropou

        # skip connection 1
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = LayerNormalization(epsilon = 1e-6)(x2) # layer normalisation 2

        x3 = mlp(x3, hidden_units = transformer_units, dropout_rate = 0.1)

        # skip connection 2
        encoded_patches = layers.Add()([x3, x2])

    representation = LayerNormalization(epsilon = 1e-6)(encoded_patches)
    representation = Flatten()(representation)
    representation = Dropout(0.5)(representation)

    features = mlp(representation, hidden_units = self.mlp_head_units, dropout_rate = 0.5)
    logits = Dense(self.label_length, activation = self.activation_function, kernel_regularizer = wr, name = "P

    network = Model(inputs = inputs, outputs = logits)

    self.__compile_network(network)
```

Malachy - CTT model:

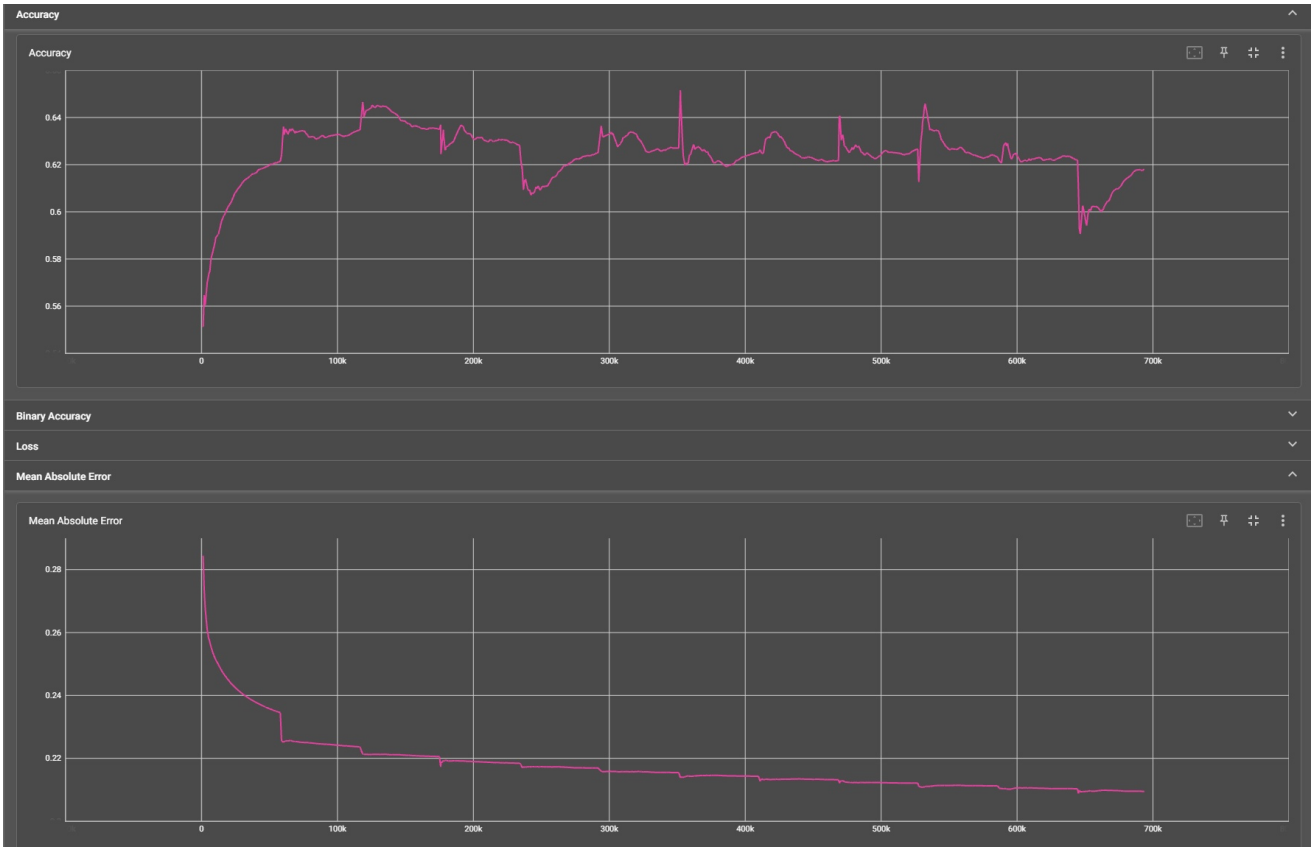
In [ ]:
```
class SequencePooling(layers.Layer): # Sequence Pooling is used in Compact Covolutional Vision Transformers.
    def __init__(self):
        super().__init__()
        self.attention = layers.Dense(1)

    def call(self, x):
        attention_weights = keras.ops.softmax(self.attention(x), axis = 1)
        attention_weights = keras.ops.transpose(attention_weights, axes = (0, 2, 1))
        weighted_representation = keras.ops.matmul(attention_weights, x)
        return keras.ops.squeeze(weighted_representation, -2)

class StochasticDepth(layers.Layer): # Referred from: github.com:rwightman/pytorch-image-models.
    def __init__(self, drop_prop):
        super().__init__()
        self.drop_prob = drop_prop
        self.seed_generator = keras.random.SeedGenerator(0)

    def call(self, x, training=None):
        if training:
            keep_prob = 1 - self.drop_prob
            shape = (keras.ops.shape(x)[0],) + (1,) * (len(x.shape) - 1)
            random_tensor = keep_prob + keras.random.uniform(shape, 0, 1, seed=self.seed_generator)
            random_tensor = keras.ops.floor(random_tensor)
            return (x / keep_prob) * random_tensor

        return x
```

In [ ]:
```
case 'CCT':
    transformer_units = [
    self.projection_dim,
    self.projection_dim,
    ]

    stochastic_depth_rate = 0.1

    inputs = layers.Input(shape = (self.input_x, self.input_y, self.input_dim))
```

```
    covolutional_tokeniser = ConvolutionalTokeniser(self, kernel_size = 3, stride = 1, padding = 1, pooling_ker
    encoded_patches = covolutional_tokeniser(inputs)

    if self.positional_embedding:
        sequence_length = encoded_patches.shape[1]
        encoded_patches = PositionEmbedding(sequence_length)(encoded_patches)

    dpr = [x for x in np.linspace(0, stochastic_depth_rate, self.transformer_layers)]  # stochastic depth decay

    for i in range(self.transformer_layers):
        x1 = LayerNormalization(epsilon = 1e-6)(encoded_patches) # normalisation layer 1
        attention_output = MultiHeadAttention(num_heads = self.num_heads, key_dim = self.projection_dim, dropou

        # skip connection 1
        attention_output = StochasticDepth(dpr[i])(attention_output)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = LayerNormalization(epsilon = 1e-6)(x2) # normalisation layer 2
        x3 = mlp(x3, hidden_units = transformer_units, dropout_rate = 0.1)

        # skip connection 2
        x3 = StochasticDepth(dpr[i])(x3)
        encoded_patches = layers.Add()([x3, x2])

    # Apply sequence pooling.
    representation = LayerNormalization(epsilon=1e-5)(encoded_patches)
    weighted_representation = SequencePooling()(representation)

    logits = Dense(self.label_length, activation = self.activation_function, kernel_regularizer = wr, name = "P
    network = keras.Model(inputs=inputs, outputs=logits)

    self.__compile_network(network)
```

MobileNetV3Large:

labels = 'all-features'



meta.json :

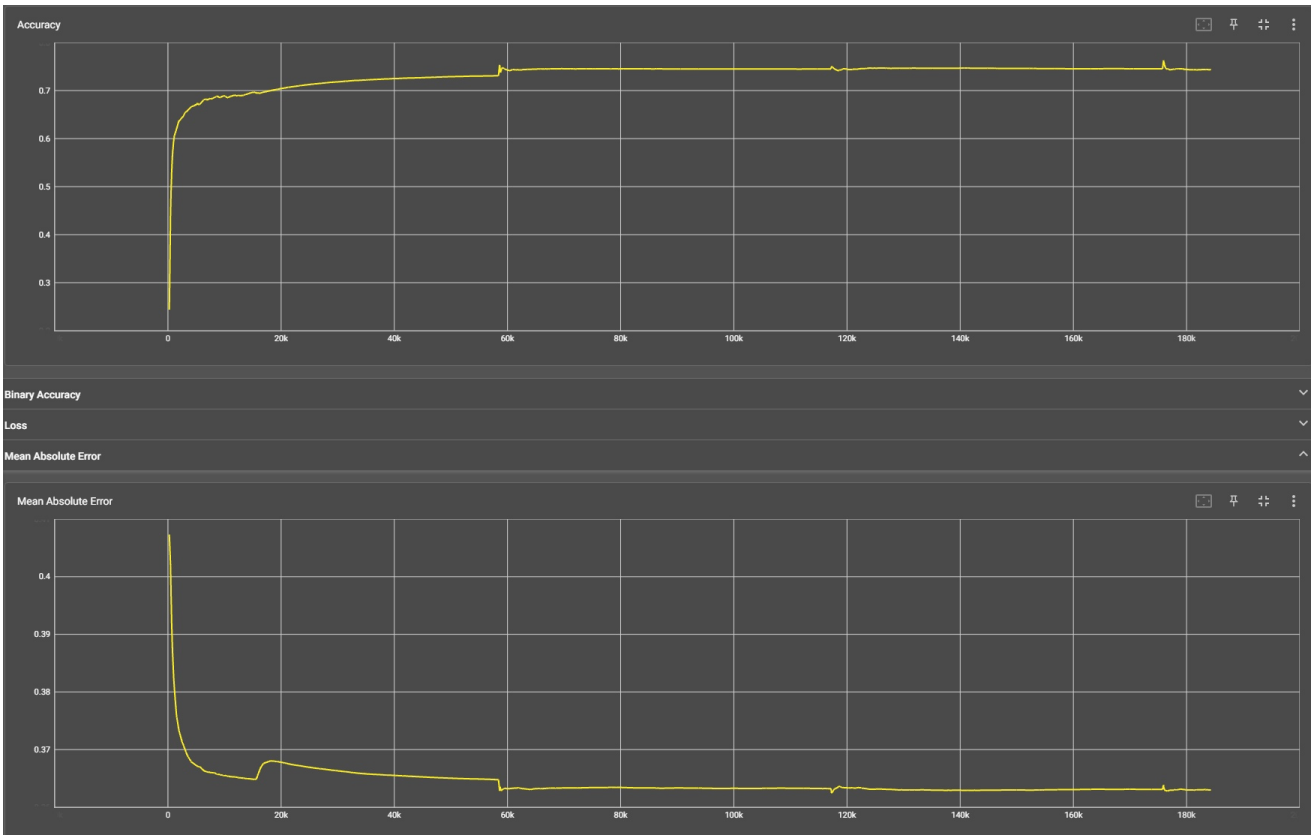
In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-05-02-01-14-35",
    "global_epoch": 11,
    "global_batch": 688128,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
```

```
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "all-features",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "MobileNetV3Large",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 8,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 32,
    "val_batch_size": 32,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```


ViT:

labels = 'all-features'



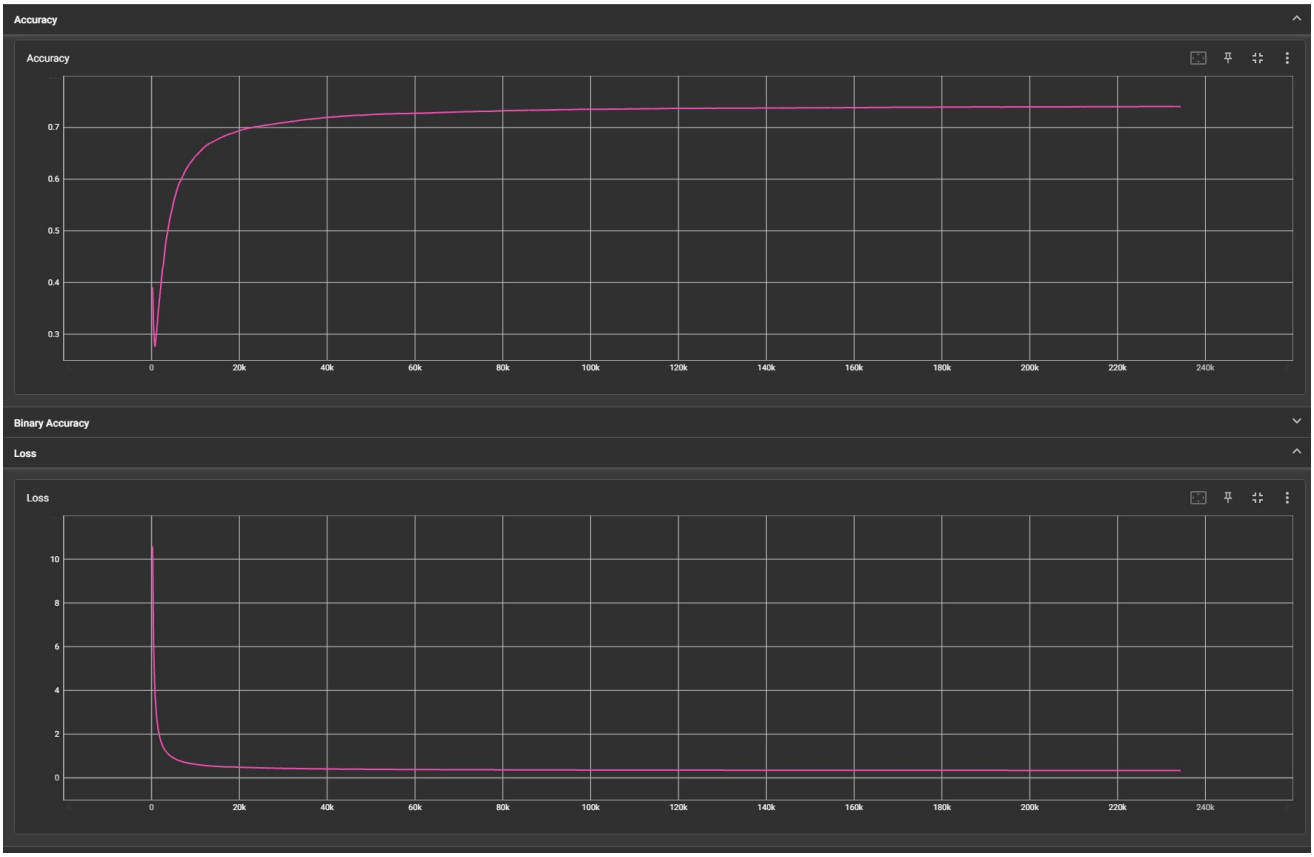
meta.json :

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-05-03-05-23-31",
    "global_epoch": 2,
    "global_batch": 175782,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "all-features",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        256,
        256
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "ViT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 12,
    "transformer_layers": 12,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 16,
    "val_batch_size": 16,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 58594,
    "augmentation": true,
    "crop": true,
    "crop_size": [
        256,
        256
    ]
}
```


CvT:

labels = 'reduced'




```
I0000 00:00:1714796605.797827    2066 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
234376/234376 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.7216 - binary_accuracy: 0.7158 - loss: 0.4299 - mean_absolute_error: 0.4003 - root_mean_squared_error: 0.4430I0000 00:00:1714811581.285725
```

meta.json:

In [ ]:

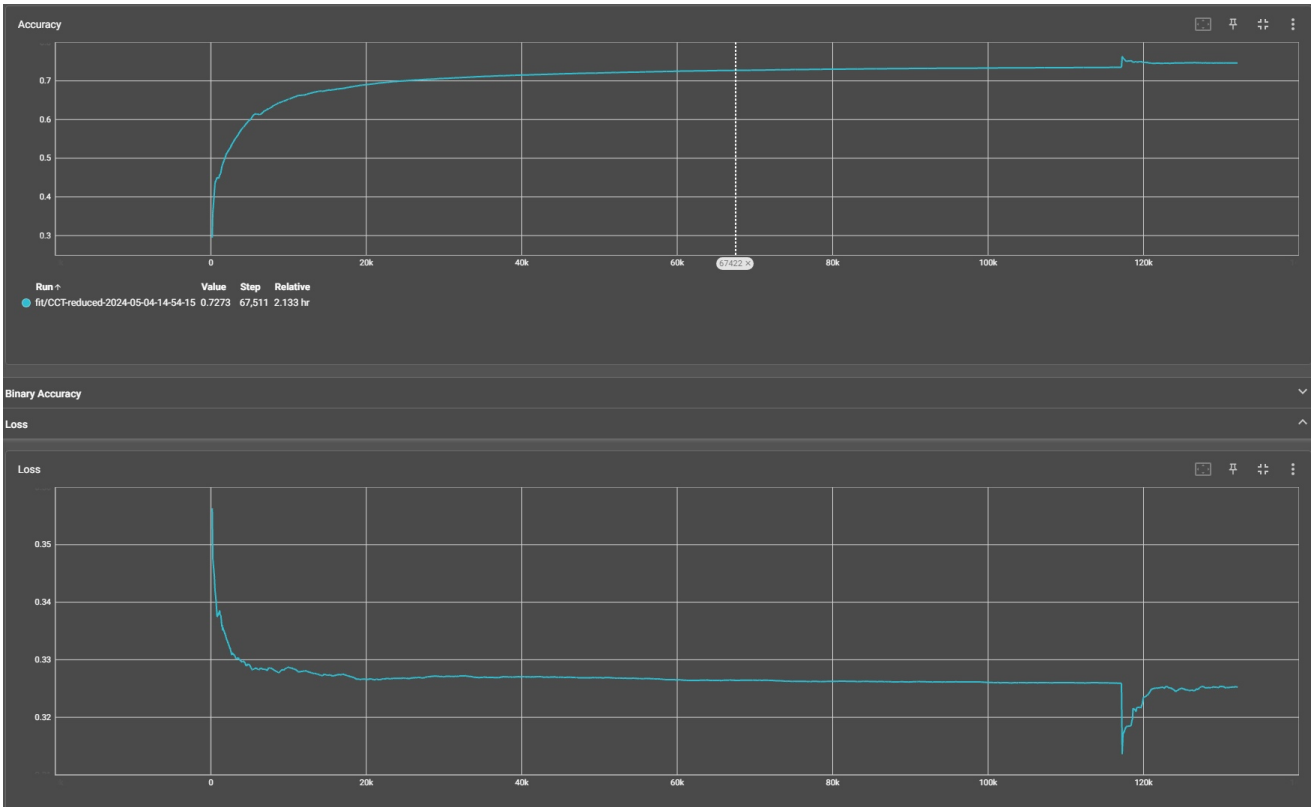

```
{
    "seed": 0,
    "time_stamp": "2024-05-04-05-03-58",
    "global_epoch": 0,
    "global_batch": 234376,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        128,
        128
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "CvT",
    "learning_rate": 0.001,
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 64,
    "num_heads": 2,
    "transformer_layers": 2,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 4,
    "val_batch_size": 4,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 234376,
```

```
    "augmentation": true,
    "crop": false,
    "crop_size": [
        128,
        128
    ]
}
```

CCT:

labels = 'reduced'

```
I0000 00:00:1714832038.157348   30069 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
117188/117188 ━━━━━━━━━━━━━━━━━━━━ 13468s 114ms/step - accuracy: 0.7064 - binary_accuracy: 0.7177 - loss: 0.3270 - mean_absolute_error: 0.3988 - root_mean_squared_error: 0.4391 - val_accuracy: 0.7512 - val_binary_accuracy: 0.7246 - val_loss: 0.5664 - val_mean_absolute_error: 0.3963 - val_root_mean_squared_error: 0.4374
Epoch 2/200
     1/117188 ━━━━━━━━━━━━━━━━━━━━ 3:46:43 116ms/step - accuracy: 0.6250 - binary_accuracy: 0.6875 - loss: 0.2920 - mean_absolute_error: 0.3904 - root_mean_squared_error: 0.4305I0000 00:00:1714845471.588947  139897 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'triton_gemm_dot_263', 256 bytes spill stores, 256 bytes spill loads

I0000 00:00:1714845500.758517   30066 asm_compiler.cc:369] ptxas warning : Registers are spilled to local memory in function 'loop_add_subtract_fusion_28', 16 bytes spill stores, 16 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_16', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_15', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_14', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_13', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_12', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_11', 20 bytes spill stores, 20 bytes spill loads
ptxas warning : Registers are spilled to local memory in function 'input_add_multiply_reduce_fusion_10', 160 bytes spill stores, 124 bytes spill loads

14847/117188 ━━━━━━━━━━━━━━━━━━━━ 3:23:36 119ms/step - accuracy: 0.7471 - binary_accuracy: 0.7214 - loss: 0.3240 - mean_absolute_error: 0.3963 - root_mean_squared_error: 0.4367
```



meta.json :

In [ ]:


```
{
    "seed": 0,
    "time_stamp": "2024-05-04-14-54-15",
    "global_epoch": 1,
    "global_batch": 131072,
    "saved_weights_path": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine
    "save_step": 32768,
    "root": "/home/malachy/3rd Year Project/Project-72-Classifying-cosmological-data-with-machine-learning",
    "dataset_root": "galaxyzoo2-dataset-augmented/",
    "train_catalog": "combined-train_catalog.parquet",
    "test_catalog": "combined-test_catalog.parquet",
    "checkpoint_root": "checkpoints/",
    "log_root": "logs/fit/",
    "classes": "reduced",
    "weight_mode": "effective_num_samples",
    "beta": null,
    "build_paths": true,
    "binarised_labels": true,
    "ds_fraction": 1,
    "val_fraction": 0.2,
    "input_shape": [
        128,
        128
    ],
    "input_dim": 3,
    "summary": true,
    "architecture": "CCT",
    "learning_rate": 0.001,
```

```
    "loss_function": "binary_crossentropy",
    "activation_function": "sigmoid",
    "weight_reg": null,
    "num_patches": 256,
    "projection_dim": 32,
    "num_heads": 8,
    "transformer_layers": 8,
    "mlp_head_units": [
        2048,
        1024
    ],
    "train_batch_size": 8,
    "val_batch_size": 8,
    "val_steps": 512,
    "epochs": 200,
    "steps_per_epoch": 117188,
    "augmentation": true,
    "crop": false,
    "crop_size": [
        128,
        128
    ]
}
```