ANIMAL IMAGE CLASSIFICATION

Group 4:

Isaiah Martinez, Joycelyn Tuazon

TABLE OF CONTENTS

DATASET OVERVIEW

Guiding Questions:

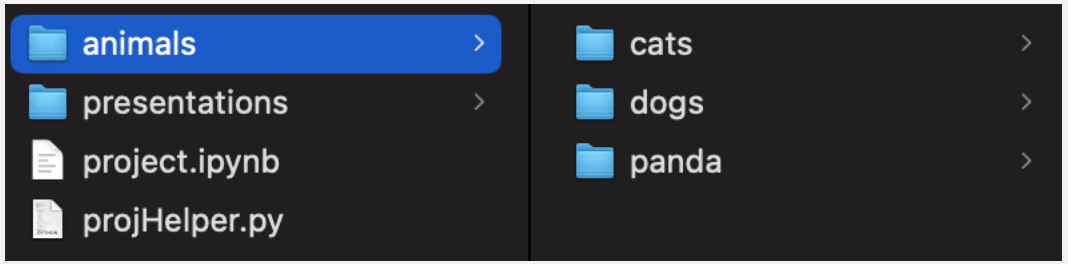
- Where did the given data come from?
- What did the given data look like?
- What is the structure of the given data?

DATASET OVERVIEW

- Dataset name: **Animal Image Dataset (DOG, CAT and PANDA)***

- Link to Dataset:
  - https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda

- 3 folder structure:
  - cats
  - dogs
  - panda

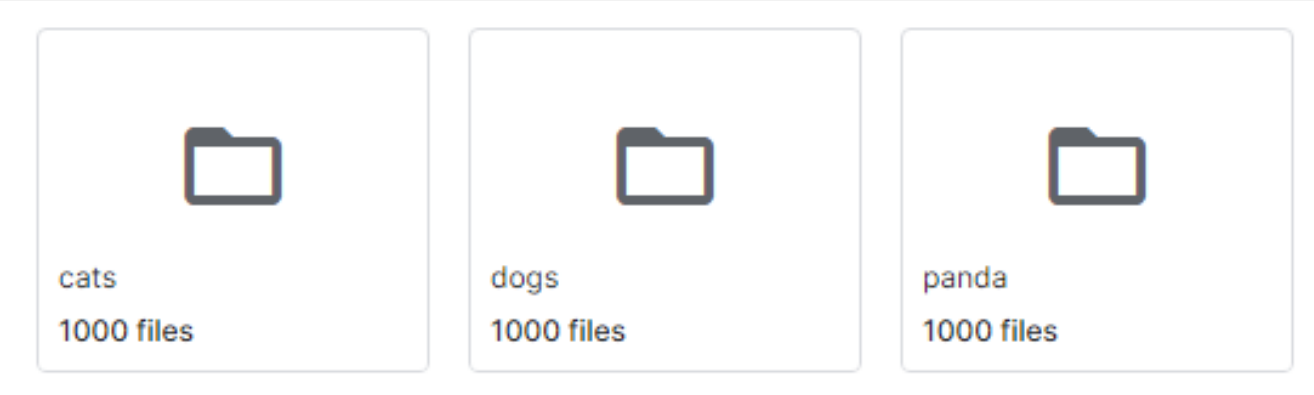


Folder Structure

DATASET OVERVIEW

- Each animal/directory had 1000 images
- Image size was highly varied:
  - Smallest images were 55 x 75 pixels and 94 x 69 pixels
  - Largest image was 1600 x 1200 pixels
  - Avg image dimensions were

    431 x 372 pixels



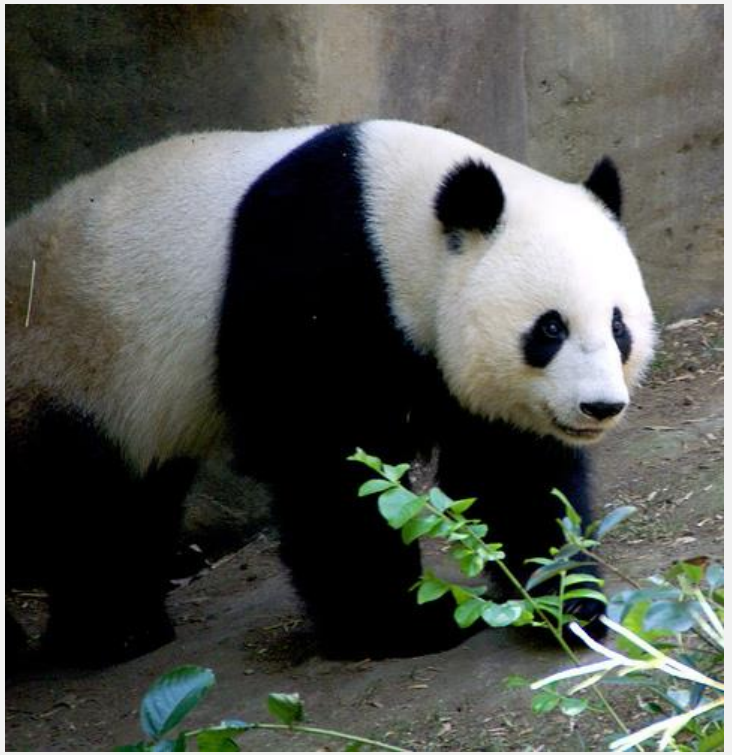
Files per Directory

DATA VISUALIZATION

DATA VISUALIZATION

Cat Image 1

Dog Image 1



Panda Image 1

DATA CLEANING

Guiding Questions:

- Were there "bad" images?
- What was the criteria used?
- What are some examples of "bad" images?

DATA CLEANING

- Cleaned manually
- Got rid of "bad" images meeting any of the following criteria:
  - Multiple interested animal species within the same image
  - Too small of image size (< 100 x 100)
  - Difficult to view subject
  - Image does not contain subject

Cat image 374

Panda Image 475

Cat image 556

DATA PREPARATION

Guiding Questions:

- Could the cleaned images be used as is?
- What are some examples of the images used to train the models?

DATA PREPARATION

- Several ideas employed:
  - Resize to 500 x 500 pixels
  - Convert to Grayscale
  - Convert images to flattened arrays
  - Resize to 224 x 224 pixels
  - Generate new images using image augmentation

Cat Image 1

GS Cat Image 1

DATA PREPARATION

Sample conversion to array:

Panda Image 880

```
[[[159 155 128] [162 161 131] [165 169 134] ...
[120 141 162] [118 138 162] [119 139 163]] [[136
136 112] [154 157 128] [158 165 131] ... [124
144 168] [123 143 167] [125 145 169]] [[124 133
112] [127 137 112] [148 163 132] ... [132 154
178] [133 155 179] [135 157 181]] ... [[ 31 51
58] [ 26 46 53] [ 22 42 51] ... [ 53 55 42] [ 49
52 41] [ 48 51 40]] [[ 33 52 59] [ 29 48 55] [
24 44 53] ... [ 50 52 41] [ 47 50 41] [ 47 50
41]] [[ 34 53 60] [ 29 48 55] [ 25 45 54] ... [
45 47 36] [ 42 45 36] [ 43 46 37]]]
```

Panda Image 880 as an array

DATA PREPARATION

- Each image holds (Length x Width x Color Mode) features:
  - 500 x 500 Images x 1 brightness (Grayscale) => 250,000 Features
  - 500 x 500 Images x 3 color intensity (RGB) => 750,000 Features
  - 224 x 224 Images x 3 color intensity (RGB) => 150,528 Features
- 2952 images were used
  - 250,001 Features x 2952 tuples
  - 750,001 Features x 2952 tuples
  - 150,529 Features x 2952 tuples
  - Recall: 1 Feature is the animal class within the given image

ML ALGORITHMS

List of ML algorithms used:

- Isaiah:
  - SVM Classifier
  - Res Net 50
  - Zero-Shot Prediction with CLIP **
- Joycelyn:
  - CNN
  - VGG16

MODEL: SVM CLASSIFIER

- Lots of usage with image classification
- Easy to implement and update with better performing parameters
- Took a long time to run
  - > 1-2 days in many cases
- Highly sensitive to the data size:
  - $\Omega(D)^{***} \sim \Omega(N^2) \sim O(N^2 \times D) \sim \prod(N^3)$
  - For N = Number of tuples, D = Number of Features
    - N = 2952
    - D = [750,000, 250,000]
- Kfold = O(KN) = O(N) for K = Number of Folds
- Grid Search = (Kfold x SVM x Combinations of Parameter List)
  - $O(N^2 \times N \times P)$ for P = Number of Parameter Combinations
  - $O(N^3)$

MODEL: SVM CLASSIFIER

```
###     Final Pipeline
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler

pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('classifier', svm.SVC(cache_size = 16000, max_iter = 5000, decision_function_shape = 'ovr', verbose = True))
])

params = {
    'classifier__C' : [5, 1]
}

gridSearch = GridSearchCV(pipeline, params, cv = 5, n_jobs = 1, verbose = 4)

gridSearch.fit(xTrain, yTrain)

#1750 minute runtime (initial 750,000) (250 fits)
#100 minute runtime   (initial 250,000) (1 fit)
#417 minute runtime   (initial 250,000) (15 fits)
#1051 minute runtime   (initial 250,000) (45 fits)
```

Pipeline + GridSearch

1750 minutes =
29.17 hours =
1.22 days

MODEL EVALUATION: SVM

```python
#prediction and metrics

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

yPred = gridSearchModel.predict(xTest)

acc = accuracy_score(yTest, yPred)
f1 = f1_score(yTest, yPred, average = 'weighted')
prec = precision_score(yTest, yPred, average = 'weighted')
recall = recall_score(yTest, yPred, average = 'weighted')

print(acc, f1, prec, recall)
#4 minute runtime

#initial run stats:
#0.5854483925549916 0.5885099209700294 0.5983072203381199 0.5854483925549916

#12 minute runtime 1 fit

#initial run stats: 1 fit
#0.6091370558375635 0.6054727828929725 0.6030864613210003 0.6091370558375635

#12 minute runtime 15 fits

#initial run stats: 1 fit
#0.6328257191201354 0.6299543290220251 0.6287454462588863 0.6328257191201354
```

Evaluation Metrics SVM

- ~63% consistent metrics
- No Loss Plot
- No Confusion Matrix

MODEL: SVM

```
SVC
SVC(C=10, cache_size=6000, decision_function_shape='ovo', gamma='auto',
    max_iter=5000, verbose=True)
```

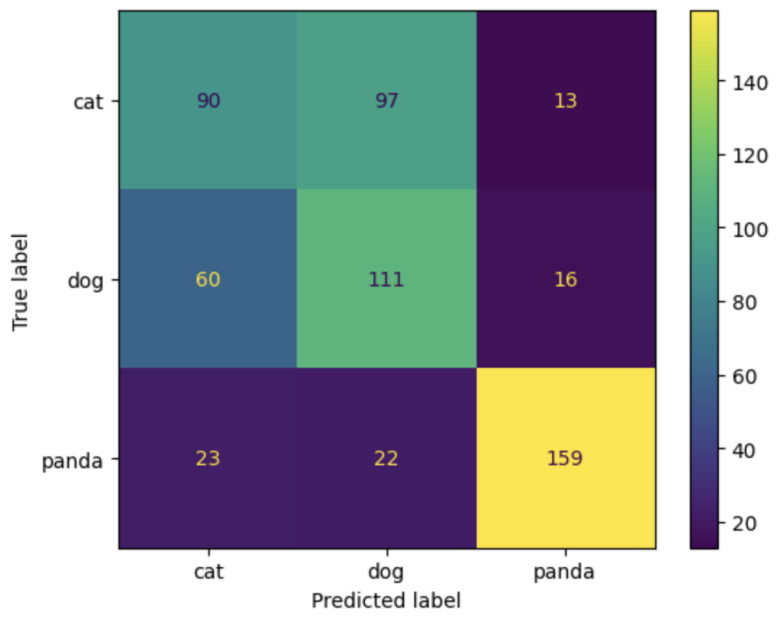
Model Parameters

```
0.6091370558375635 0.6117741083946036 0.6206869437481218 0.6091370558375635
```

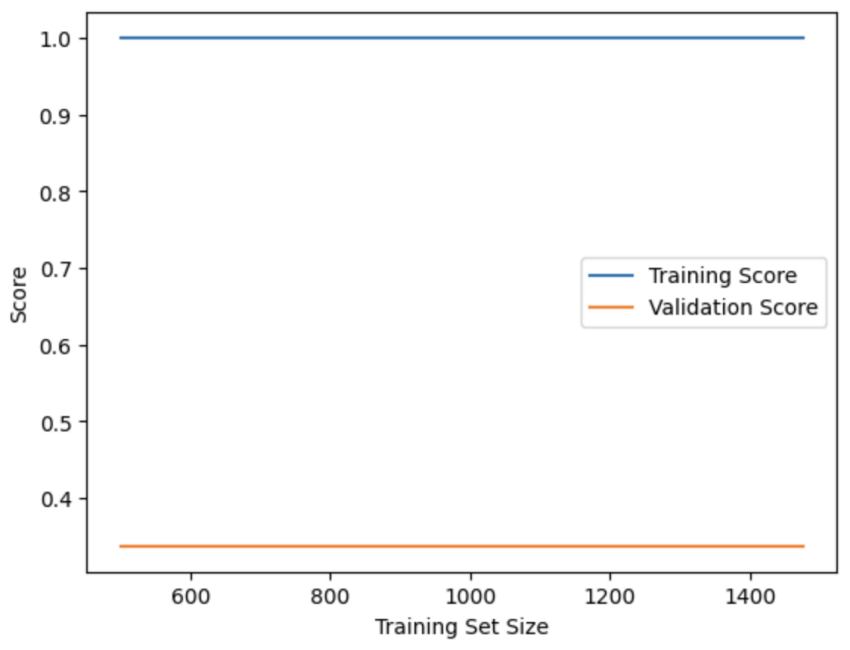
Evaluation Metrics

Accuracy F1 Score Precision Recall

MODEL EEVALUATION: SVM



CM SVM

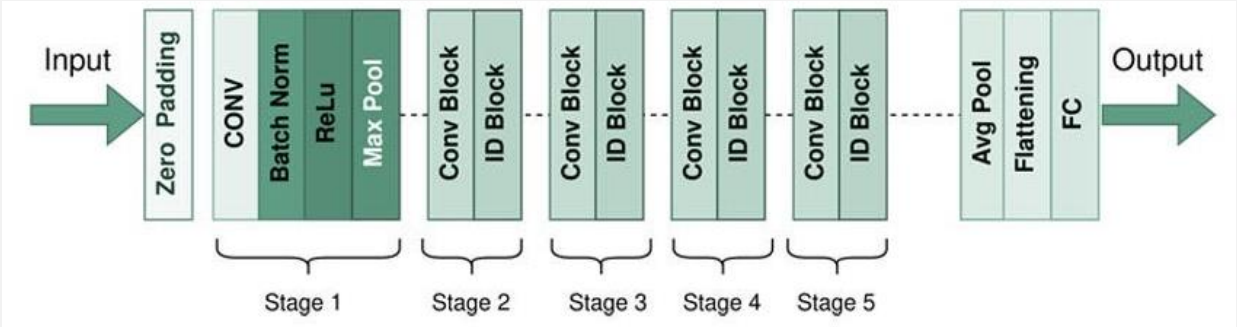


Score vs Training Size

- EXTREME overfitting
- Long time to process

MODEL: RES NET 50

- A type of CNN that is 50 layers deep
- Has shortcuts allowing the gradient to flow easily during training
- Very powerful for DL and ML tasks regarding images
- Can perform 2D Convolutions and Max Pooling to obtain images and extract features to improve performance



Res Net 50 example

MODEL: RES NET 50

Ran 3 separate models

V1

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_1 (Flatten)         (None, 100352)            0

 dense_3 (Dense)             (None, 256)               25690368

 dropout_2 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 3)                 771

=================================================================
Total params: 49278851 (187.98 MB)
Trainable params: 25691139 (98.00 MB)
Non-trainable params: 23587712 (89.98 MB)
```

Model 1

V2

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_2 (Flatten)         (None, 100352)            0

 dense_5 (Dense)             (None, 256)               25690368

 dropout_3 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 256)               65792

 dropout_4 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 256)               65792

 dropout_5 (Dropout)         (None, 256)               0

 dense_8 (Dense)             (None, 3)                 771

=================================================================
Total params: 49410435 (188.49 MB)
Trainable params: 25822723 (98.51 MB)
Non-trainable params: 23587712 (89.98 MB)
```

Model 2

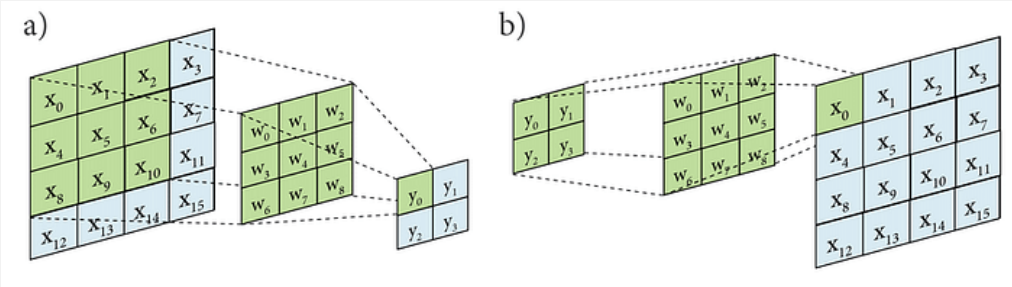
V3

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 conv2d (Conv2D)             (None, 5, 5, 128)         2359424

 max_pooling2d (MaxPooling2  (None, 2, 2, 128)         0
 D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 3)                 99

=================================================================
Total params: 25982147 (99.11 MB)
Trainable params: 2394435 (9.13 MB)
Non-trainable params: 23587712 (89.98 MB)
```

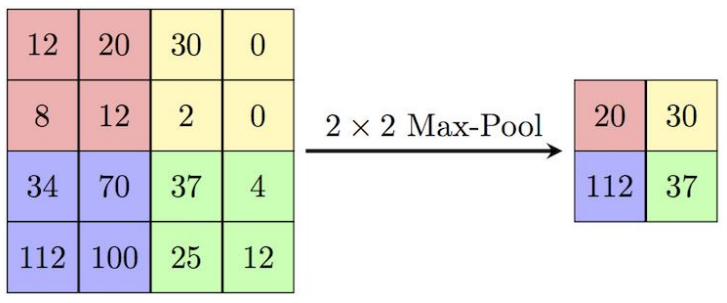
Model 3

MODEL: RES NET 50

- V1 was the simplest
  - 1 hidden layer
- V2 was simplistic and designed based off of V1
  - 3 hidden layers
    - All had the same number of neurons and dropout value
- V3 was designed to be complex and different from V1 and V2
  - Used Conv2D layers and MaxPooling2D layers
  - Different dropouts per layer
  - Strictly decreasing neuron count per layer
- All models:
  - used ReLu activation function
  - Had a single output layer with 3 neurons – 1 per animal



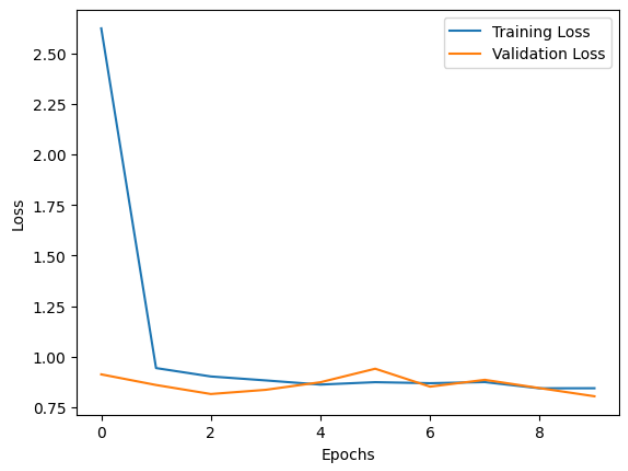
Convolution 2D



Max Pool 2D

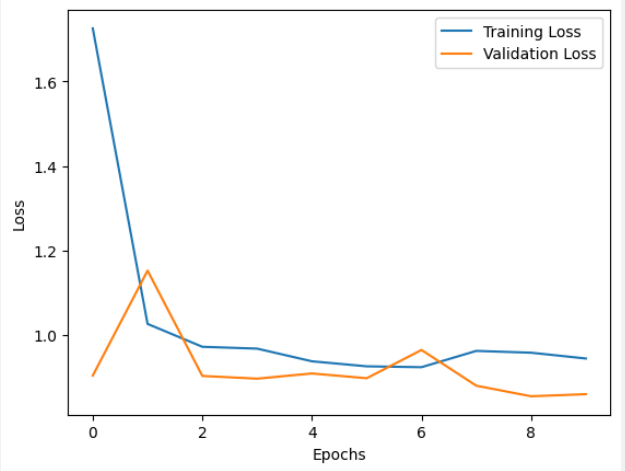
MODEL EVALUATION: RES NET 50

V1



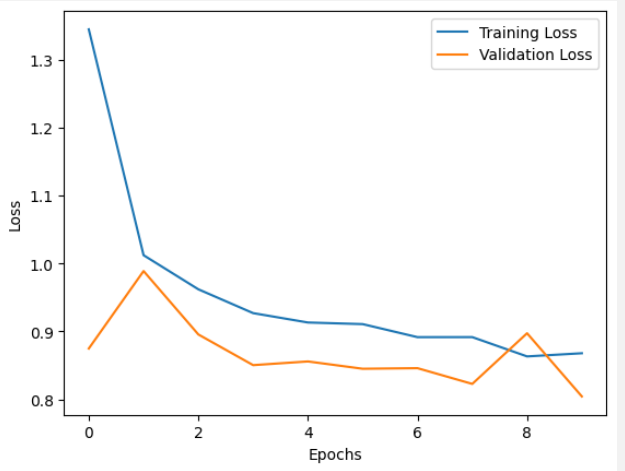
Loss V1

V2



Loss V2

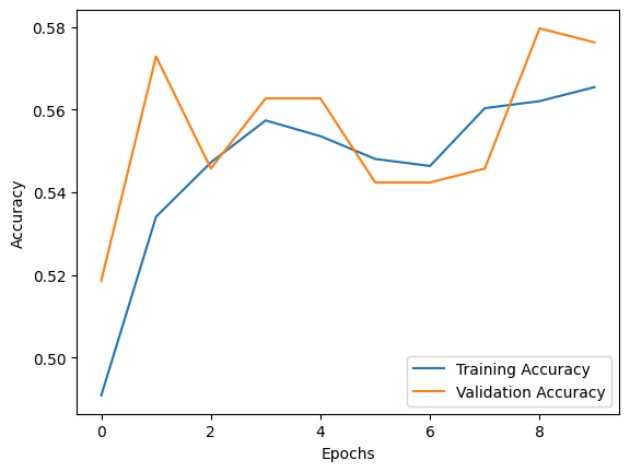
V3



Loss V3

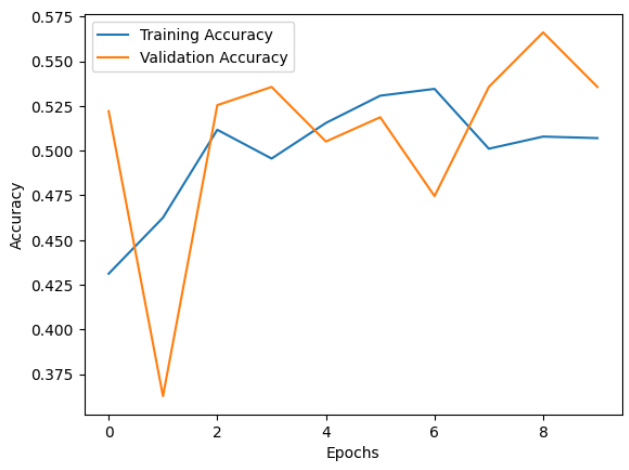
MODEL EVALUATION: RES NET 50

V1



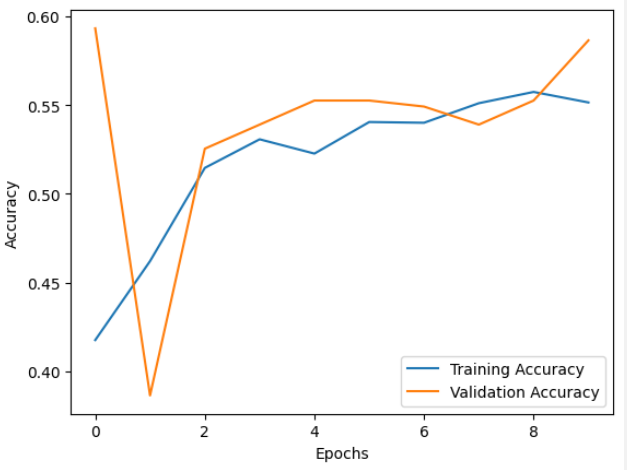
Accuracy V1

V2



Accuracy V2

V3



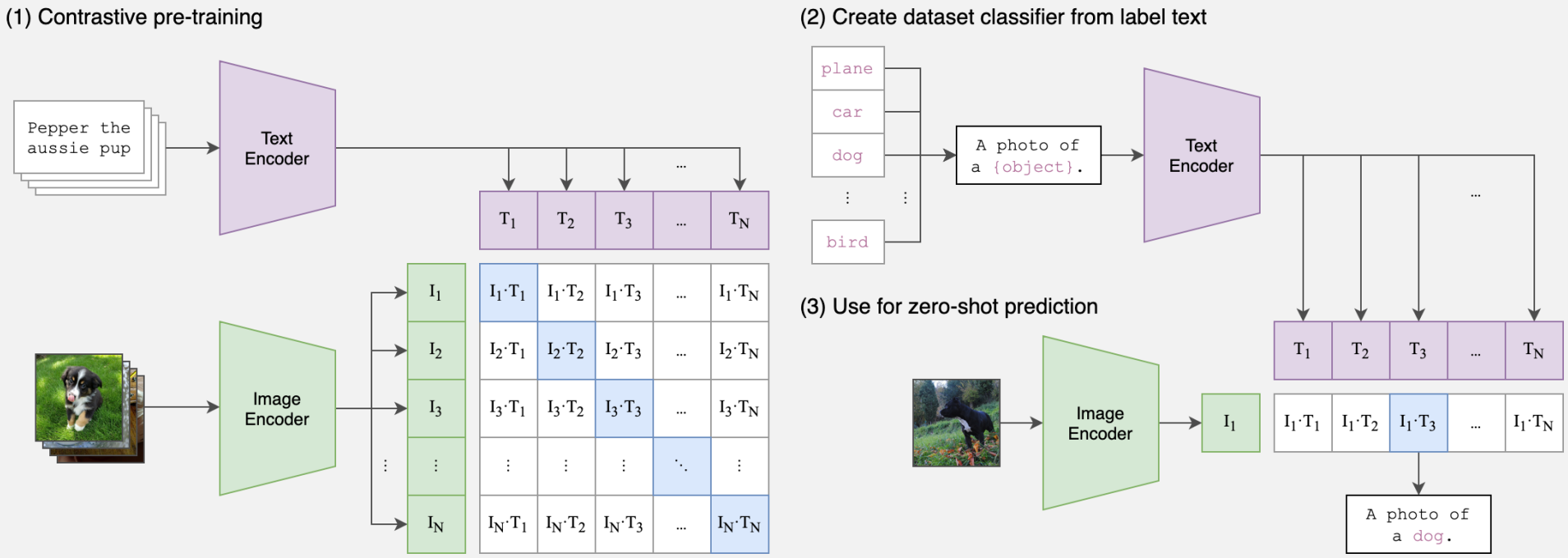
Accuracy V3

MODEL EVALUATION: RES NET 50

- Resulting Accuracies:
  - V1: ~ 61%
  - V2: ~ 29%
  - V3: ~ 55%

- Thus, V1 yielded the best performance despite model modification

MODEL: CLIP



How CLIP works with ZSP

MODEL: CLIP

- Zero-Shot Prediction:
  - Can a model make accurate predictions about instances without being trained on any part of the desired dataset?
  - Operates like the following example:
    - Step 1: Show an image of an animal to a person
    - Step 2: Ask them to determine what animal is being shown
  - An extreme alternative to not using labeled data

(Cat Image 461)

Visual Example

Code Sample on Cat image 1 using ViT-B/32 Model:

```python
import torch
import clip
from PIL import Image
import numpy as np

#make device with cuda (GPU) if available, otherwise uses the CPU
device = "cuda" if torch.cuda.is_available() else "cpu"

#loads the model ViT-B/32, and preprocess
model, preprocess = clip.load("ViT-B/32", device = device)

#preprocess the image and load it into the device
img = preprocess(Image.open("animals_cleaned/cats/cats_00001.jpg")).unsqueeze(0).to(device)

#categories to be used by model
animals = ['cat', 'dog', 'panda']

#add text (categories) to device
text = clip.tokenize(animals).to(device)

#read the image and text to obtain the Logits
#use the Logits as input for softmax function to obtain probabilities
#Note: `_` denotes a var/output we dont care about
with torch.no_grad():
    logitsPerImg, _ = model(img, text)
    probs = np.array(logitsPerImg.softmax(dim = -1).cpu().numpy())

#take largest probability as label
label = np.argmax(probs[0])

#print probability of highest class
print(probs[0][label])

#print predicted class
print(animals[label])

#list of each probability for each class
print(probs[0])
```

Entirety of code for predicting 1st cat image

Cat Image 1

```
0.9885767
cat
[0.9885767  0.00973916 0.00168417]
```

Print statement results

MODEL EVALUATION: CLIP

- CLIP doesn't yield Evaluation Metrics
- These Confusion Matrices were computed manually
  - Columns refer to the predicted class: 1 = Cat, 2 = Dog, 3 = Panda
  - Rows refer to the actual class: 1 = Cat, 2 = Dog, 3 = Panda

- Using ViT-B/32 Model

```
[[991   1   2]
 [  4 986   0]
 [  0   1 967]]
```

CM ViT-B/32

- Using RN50x16 Model

```
[[993   1   0]
 [  2 988   0]
 [  0   0 968]]
```

CM RN50x16

MODEL: CNN

- CNN models apply convolution operations to input data to learn features and identify patterns
  - Filters and kernels in the convoluted layers capture these features and patterns
  - MaxPooling layers retain important features and patterns
  - Connected layers combine these features for the model to learn relationships between them
- CNN models are good for image classification because of these methods of capturing and learning features and patterns

MODEL: CNN

Before applying the CNN model, we must pre-process the images and classes

```
# reshape 500*500*1 matrices into vectors
xTrain_reshaped = np.reshape(xTrain, (2361, 250000))
xTest_reshaped = np.reshape(xTest, (591, 250000))

# normalize the images to be within [0-1] instead of [0-255]
xTrain_reshaped = pjH.scalePixels(xTrain_reshaped)
xTest_reshaped = pjH.scalePixels(xTest_reshaped)

# convert string labels to integer labels
label_encoder = LabelEncoder()
yTrain_encoded = label_encoder.fit_transform(yTrain)
yTest_encoded = label_encoder.transform(yTest)
# modify classes (cat, dog, panda) to be one-hot encoded vectors
classes = 3 # there are 3 classes (cat, dog, panda)
yTrain_one_hot = to_categorical(yTrain_encoded, classes)
yTest_one_hot = to_categorical(yTest_encoded, classes)
```

Pre-processing steps

Additionally, apply data augmentation to the dataset to increase variance within the data

```
from keras.preprocessing.image import ImageDataGenerator
# data augmentation

datagen = ImageDataGenerator (
    rotation_range=20, # rotate images randomly up to 20 degrees
    width_shift_range=0.2, # shift images horizontally up to 20% of the width
    height_shift_range=0.2, # shift images veritcally by up to 20% of the height
    shear_range=0.2, # apply shear transformations
    zoom_range=0.2, # zoom in or out by up to 20%
    horizontal_flip=True, # flip images horizontally
    fill_mode="nearest" # fill in missing pixels with the nearest available pixel
)
```

Augmented Image Generator

MODEL: CNN

```
# build CNN model structure
model = Sequential()

# Layer 1 (Conv2D)
model.add(Conv2D(16, (3, 3), activation="relu", input_shape=(500, 500, 1)))

# Layer 2 (MaxPooling2D)
model.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 3 (Conv2D)
model.add(Conv2D(32, (3 ,3), activation="relu"))

# Layer 4 (MaxPooling2D)
model.add(MaxPooling2D(pool_size=(2,2)))

#Layer 5 (Conv2D)
model.add(Conv2D(64, (3, 3), activation="relu"))

# Layer 6 (MaxPooling2D)
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))

# Output Dense layer
model.add(Dense(3, activation="softmax"))
```

- Input layer is a Conv2D layer with 16 filters of size 3x3
- Next layer is a MaxPooling layer with window size 2x2
  - MaxPooling retains the max value within the 2x2 window, and discards the rest
  - Retaining important features and discarding the rest makes the model more computationally efficient
- Dropout layers are used to help prevent overfitting by randomly dropping neurons during training
- We flatten the output from the previous convoluted layer into a 1-Dimensional vector before moving onto the fully connected layer
- Dense adds a fully connected layer
- Output layer is a fully connected layer with 3 neurons:
  - one for each class: cat, dog, panda

MODEL: CNN

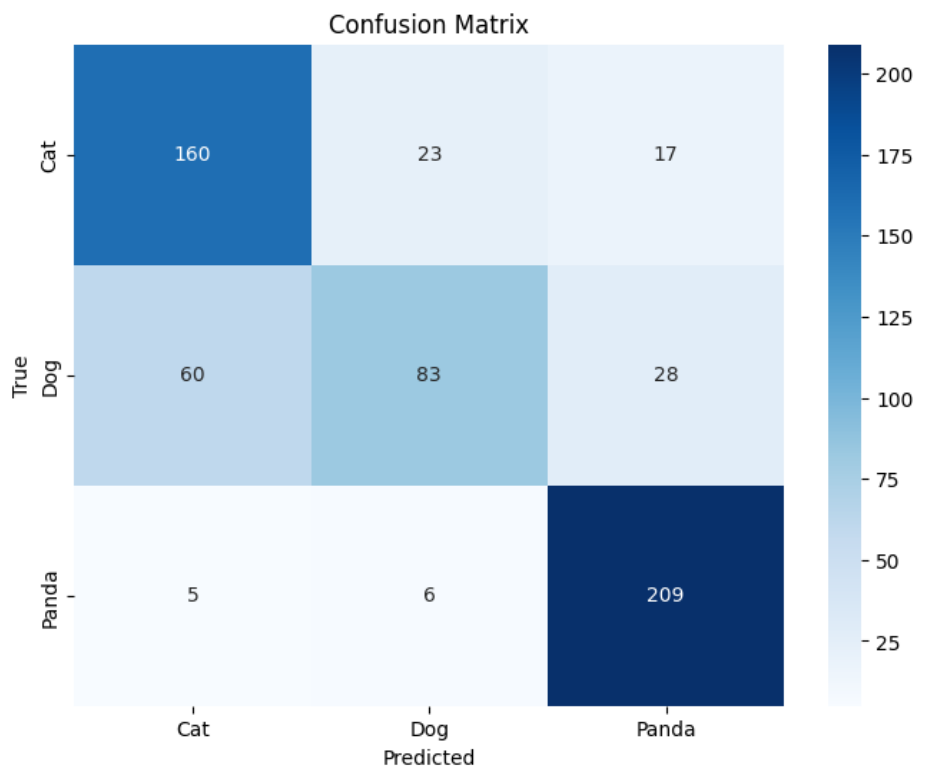
```
# create a generator for training data with data augmentation
train_generator = datagen.flow(xTrain_original_shape, yTrain_one_hot, batch_size=new_batch_size)
```

Train generator is used to generate augmented batches of data

```
from keras.callbacks import EarlyStopping

# implement early stopping to prevent overfitting
# stop training if accuracy does not improve for 3 consecutive epochs
# restore the model's weights to the best configuration observed during training
early_stopping = EarlyStopping(monitor="val_accuracy", patience=5, restore_best_weights=True)

# fit the model on training data (with data augmentation)
history = model.fit(train_generator, epochs=25, verbose=1, validation_data=(xTest_original_shape, yTest_one_hot), callbacks=[early_stopping])
```

Fit the model and implement early stopping with a significant epoch size to prevent overfitting

MODEL EVALUATION: CNN



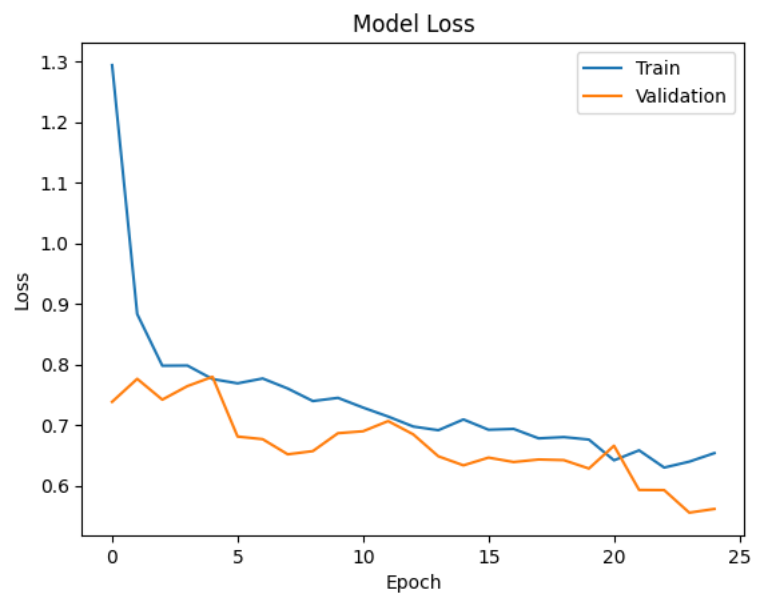
CM CNN

```
Test accuracy (calculated): 0.7648
Precision: 0.7614
Recall: 0.7648
F1 Score: 0.7528
```

Evaluation Metrics CNN

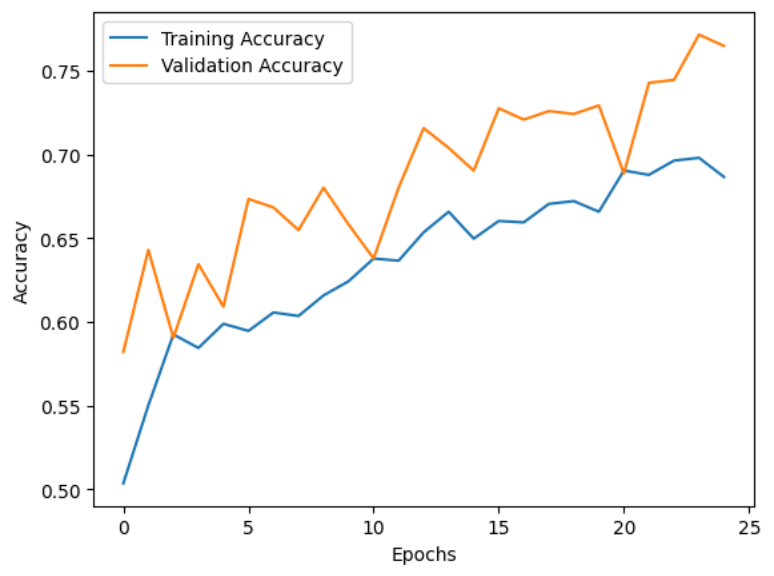
- Most runs would reach between 60-70% accuracy
  - Applying data augmentation seemed to be what increased performance the most
  - Model consistently had the most difficulty classifying dogs and the most success classifying pandas

MODEL EVALUATION: CNN



Loss vs Epoch

- No clear sign of stabilization in general
- However, when they converge, it stabilizes



Accuracy vs Epoch

- Generally positive trend
- Oscillations in validation accuracy may indicate unstable learning

MODEL: VGG16

- VGG16 is a pretrained image classification CNN model that can be accessed through TensorFlow
  - Pretrained on large datasets such as ImageNet
  - 16 layers (13 convoluted layers, 3 fully connected layers)
  - Input layer designed to take in images of size (224, 224, 3)
    - 224x224 pixels with 3 color channels (RGB)

MODEL: VGG16

- We must resize the images in preprocessing to be 224 x 224 pixels and in RGB
- VGG16 is computationally expansive
  - If we don't resize then training time takes longer

```
import cv2

# resize and repeat to create pseudo-RGB images
xTrain_resized = [cv2.resize(img, (224, 224)) for img in xTrain]
xTrain_rgb = np.repeat(np.expand_dims(xTrain_resized, axis=-1), 3, axis=-1) # replicate the single channel to create pseudo RGB images

xTest_resized = [cv2.resize(img, (224, 224)) for img in xTest]
xTest_rgb = np.repeat(np.expand_dims(xTest_resized, axis=-1), 3, axis=-1)

# normalize the images to be within [0-1] instead of [0-255]
xTrain_rgb = xTrain_rgb / 255.0
xTest_rgb = xTest_rgb / 255.0


# convert string labels to integer labels
label_encoder = LabelEncoder()
yTrain_encoded = label_encoder.fit_transform(yTrain)
yTest_encoded = label_encoder.transform(yTest)
# modify classes (cat, dog, panda) to be one-hot encoded vectors
classes = 3 # there are 3 classes (cat, dog, panda)
yTrain_one_hot = to_categorical(yTrain_encoded, classes)
yTest_one_hot = to_categorical(yTest_encoded, classes)
```

Pre-processing for VGG16

MODEL: VGG16

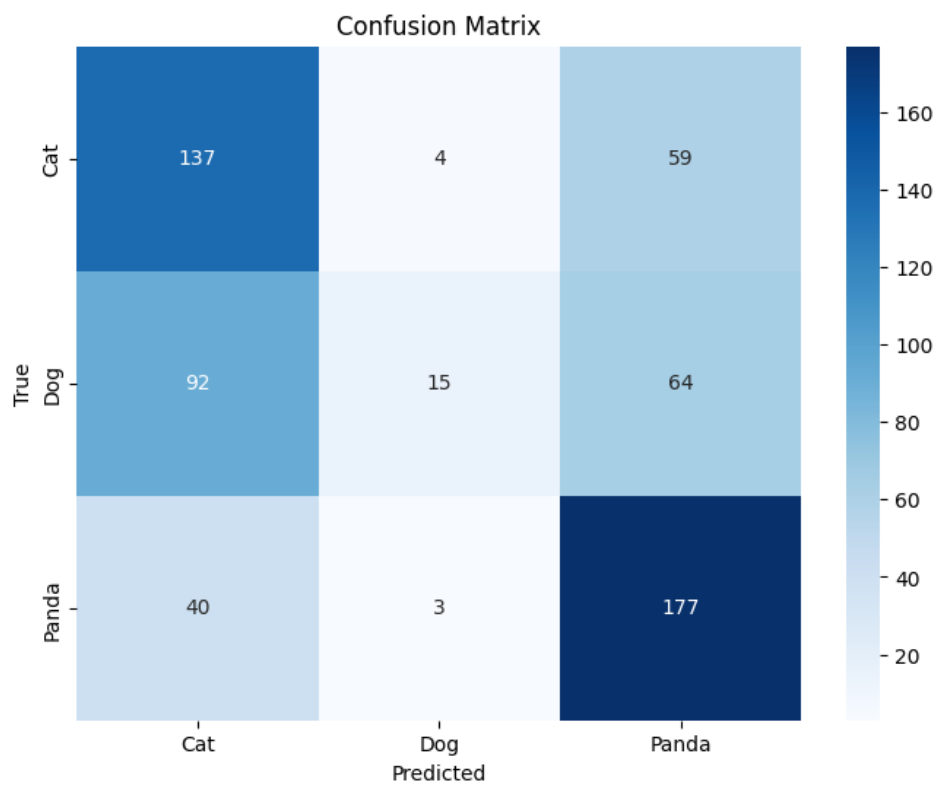
```
# load VGG16 base model without top layers
base_model = VGG16(weights=None, include_top=False, input_shape=(224, 224, 3))

# load weights manually
weights_path = "/                                                    /group_proj/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
base_model.load_weights(weights_path)

# build model
model = Sequential()
model.add(base_model)

# add custom layers on top
model.add(Flatten())
model.add(Dense(64, activation="relu", kernel_initializer="he_uniform"))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))  # 3 classes (cat, dog, panda)
```

(parameter) kernel_initializer: str

Creation of VGG16 Model

- Load base model from TensorFlow without the top layers
- Load the pretrained weights manually
- Try to keep the top layers simple as we have a smaller dataset

MODEL EVALUATION: VGG16



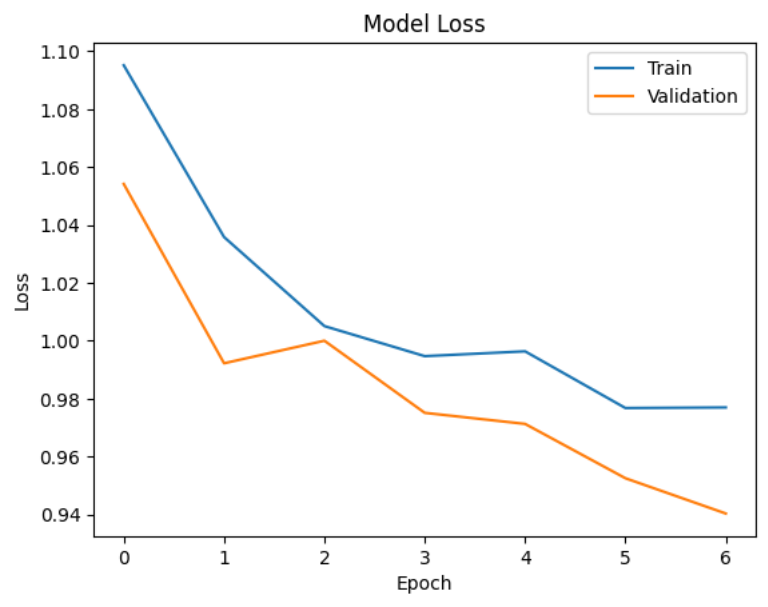
CM VGG16

```
Accuracy (calculated): 0.5567
Precision: 0.5893
Recall: 0.5567
F1 Score: 0.4961
```

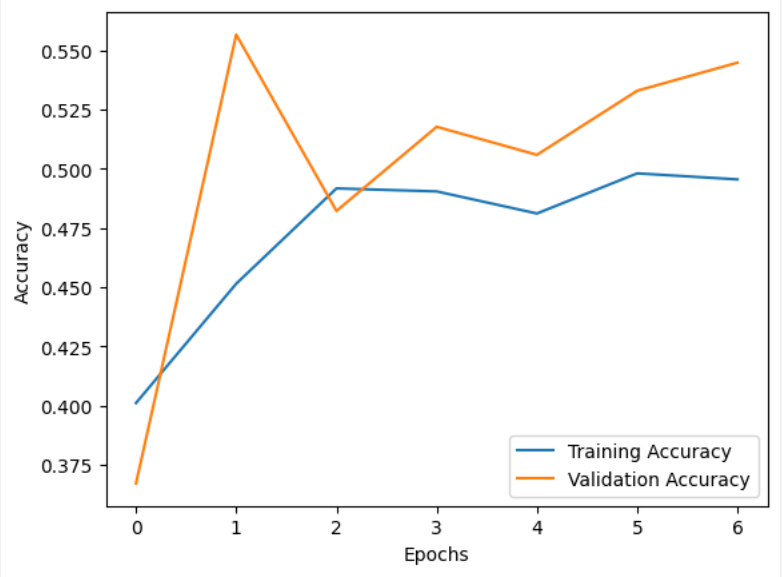
Evaluation Metrics VGG16

- Most runs stayed between 50-60% accuracy
- Overall performance not great
  - This may be because our dataset is small which makes the model prone to overfitting
  - VGG16 was pretrained on images with 3 color channels

MODEL EVALUATION: VGG16



Loss vs Epoch



Accuracy vs Epoch

- Looks like the training process may be unstable
  - Possibly due to our dataset being too small for a complex model like VGG16

- The dramatic increase in validation accuracy before dropping back down and plateauing may indicate instability

ML PERFORMANCE COMPARISON

List of ML algorithms used:

- Isaiah:
  - SVM Classifier
  - Res Net 50
  - Zero-Shot Prediction with CLIP **
- Joycelyn:
  - CNN
  - VGG16

ML PERFORMANCE COMPARISON

- SVM:
  - Overfitting
  - Accuracy: ~60%
- Res Net 50:
  - (V1) Accuracy: ~61%
- CNN:
  - Accuracy: ~76%
- VGG16:
  - Accuracy: ~55%

FUTURE WORKS

FUTURE WORKS

- Increase dataset size and diversity
  - A larger dataset may allow CNN models to achieve learning stability, increasing overall performance
    - This would particularly be useful while applying the VGG16 model, as it is more computationally expansive and was pretrained on larger datasets
- For CNN models built from scratch, more layers, filters, and neurons may need to be added to increase complexity, ensuring that the intricate features and patterns of the images were captured
- Stricter set of criteria for clean images
- Additional augmentations of images
- For SVM, use a smaller subset of the data to perform training

CONCLUSION

CONCLUSION

- Learned a lot about how image classification works
- Image classification is both difficult and time consuming
- Pre-trained models exist and are available for public usage
- Neural Networks are fantastic at Image Classification

RELATED WORKS

1. https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda
2. https://marcovirgolin.github.io/extras/details_time_complexity_machine_learning_algorithms/
3. https://stats.stackexchange.com/questions/94596/computational-complexity-of-prediction-using-svm-and-nn
4. https://medium.com/@arashserej/resnet-50-83b3ff33be7d
5. https://medium.com/analytics-vidhya/2d-convolution-using-python-numpy-43442ff5f381
6. https://paperswithcode.com/method/max-pooling
7. https://github.com/openai/CLIP/tree/main
8. https://deepchecks.com/glossary/zero-shot-learning/
9. https://blog.roboflow.com/zero-shot-learning-computer-vision/
10. https://www.freeiconspng.com/img/1682
11. https://www.ibm.com/topics/convolutional-neural-networks
12. https://www.geeksforgeeks.org/vgg-16-cnn-model/

QUESTIONS?

THANK YOU