Lab 3: Galaxy Image Classification and the Galaxy Merger Rate

Andrew Goh

Dec 2022

Abstract

We analyze the efficacy of a conventional CNN (convolutional neural network) and the ResNet50 model in classifying features in galaxy images from the Sloan Digital Sky Survey (SDSS). Labels for the training data are obtained from the Galaxy Zoo 2 project (Willett et al., 2013) [4]. A validation RMSE loss of ≈ 0.1 was reached for each of the models, with the ResNet50 model performing slightly better. We then use the ResNet trained models to estimate the fraction of merger galaxies at redshift z = 0, in which an upper bound fraction of around 3% was concluded. This value is within reasonable agreement with the results published by Lotz et al., 2011 [3].

1 Introduction

1.1 Galaxy Classification

Classifying galaxies is vital for inferring galaxy evolution over large timescales. We can observe galaxies at different redshifts z, in which higher redshifts indicate galaxies that are far away from us, and thus correspond to earlier objects in cosmic time. Essentially, by looking in deep into space, we have snapshots of galaxies at different points in time of our universe, allowing us to probe the mechanisms of galaxy formation.

Galaxy mergers as a cause of many galaxy formations is a consistent subject of interest. It is believed to be a dominant phenomenon in the early universe, as gravity from dark matter halos that continuously accrete mass will in turn pull galaxies together to merge. However, parameterization of the the galaxy merger rate as a function of redshift have been uncertain, so classifying observed galaxy morphologies is important for constraining theoretical models of galaxy formation.

The task of galaxy classification for research would normally be done by scientists, but due to the tremendous number of images, it would be an infeasible task. Lintott et al. 2008 [2] handles this by allowing the general public to classify galaxy images over the internet as volunteers. The Galaxy Zoo 2 presents for over 4×10^7 classifications done by volunteers over 300,000 galaxy images from SDSS. These final classifications denote probabilities of the galaxy containing features (smooth, odd, spiral arms, etc.) and the classification process involves a decision tree, where an outcome of a node would be weighted by the previous decision. There are 11 possible tasks with a total of 37 possible responses for each galaxy image presented to the volunteer. Biases and outlier classifications are taken into account as well and are described in Willet et al., 2013 [4].

1.2 Deep Neural Networks

Although outsourcing to volunteers for research efforts is an efficient way to handle classifying large number of images, it can still be somewhat time consuming to curate the data. Thus, we look towards neural networks that specialize in image classification. Image classification is an extremely popular task in machine learning, and many algorithms have been developed to optimize classification accuracy. In this lab, we implement a standard Convolutional Neural Network (CNN) and ResNet50 model pretrained by the ImageNet Dataset using Tensorflow's Keras framework. The Galaxy Zoo project provides us with a large training data set of feature labels that correspond with galaxy IDs associated with the images from SDSS. A detailed description of the neural network models and the training procedure is presented in section 3 and 4.

2 Training Dataset

2.1 Galaxy Images and Labels

The images from SDSS are 424×424 pixels with three RGB color channels. All of the images provided from SDSS are properly processed and normalized. Galaxy objects all have comparable brightness values and sit in the center of the image. However there are a few artifacts in some of the images, in which these also undergo a specific classification label. Example images are shown in figure 1. These images will be downsized for efficiency purposes as described in section 3, in which the downsized image are going to be passed into our neural network models.

The content of the 37 possible labels are visualized in figure 2. Unlike common classification problems such as dealing with the CIFAR-10 dataset, our classifications of galaxies are not meant to be discrete, but instead attempt to capture the confidence levels of features as provided by the processed dataset from the Galaxy Zoo Project. In addition, certain labels, or from a volunteer classifiers point of view, "answers", are dependent on prior answers determining the property of the galaxy. For example, in order to have answered a question determining the number of spiral arms a galaxy has, the volunteer must have answered whether or not the galaxy appeared to have spiral arms in the first place. In terms of encoding this information into the labels, the total probabilities of the different number of spiral arms must equal the total probability that it has spiral arms.

We can observe 'protoypes' of the labels, which are galaxy images that have the highest probability of a certain label. We expect these images to be very clear in its respective feature, corresponding to a high confidence of it possessing that feature. Prototypes are shown in figure 3.

2.2 Correlation between labels

We can inspect the labels further to determine the quality of them. In theory, we expect there to be no correlation between labels within the same 'class'. These classes correspond to a set of possible answers proposed in a single question. For example, a galaxy cannot both be smooth and have features, corresponding to answers in Class 1, or a galaxy cannot have different numbers of spiral arms, corresponding to answers in Class 11. However, we do expect correlations between answers of different classes. We can observe the correlation between the labels with the following equation:

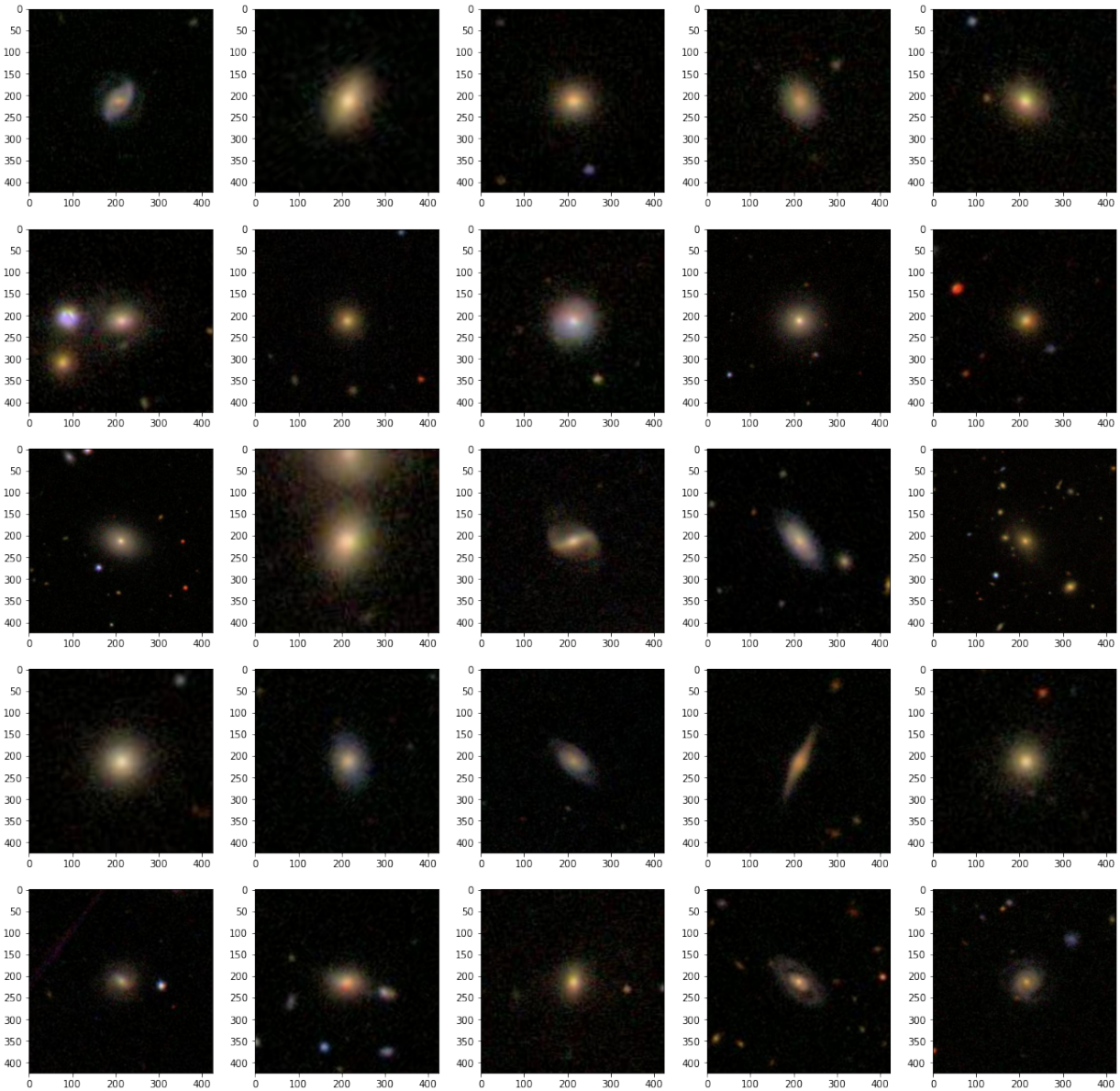

Figure 1: Example galaxy images from SDSS. A variety of features can be seen from these images, e.g. smooth, spiral, bulges, in which an image classification task is feasible.

$$\rho_{ij} = \frac{\langle ij \rangle - \langle i \rangle \langle j \rangle}{\sqrt{\langle i^2 \rangle - \langle i \rangle^2} \sqrt{\langle j^2 \rangle - \langle j \rangle^2}} \tag{1}$$

By calculating this quantity for each 37×37 combinations, we obtain a correlation matrix which can be visualized by a heatmap. The matrix is plotted in figure 4. There are nonzero correlations between labels that are expected to be non correlated within its respective class, which indicates uncertainties between different answers. For example, there are correlations between the different number of spiral arms, indicating a significant uncertainty in determining spiral arm features from the images.

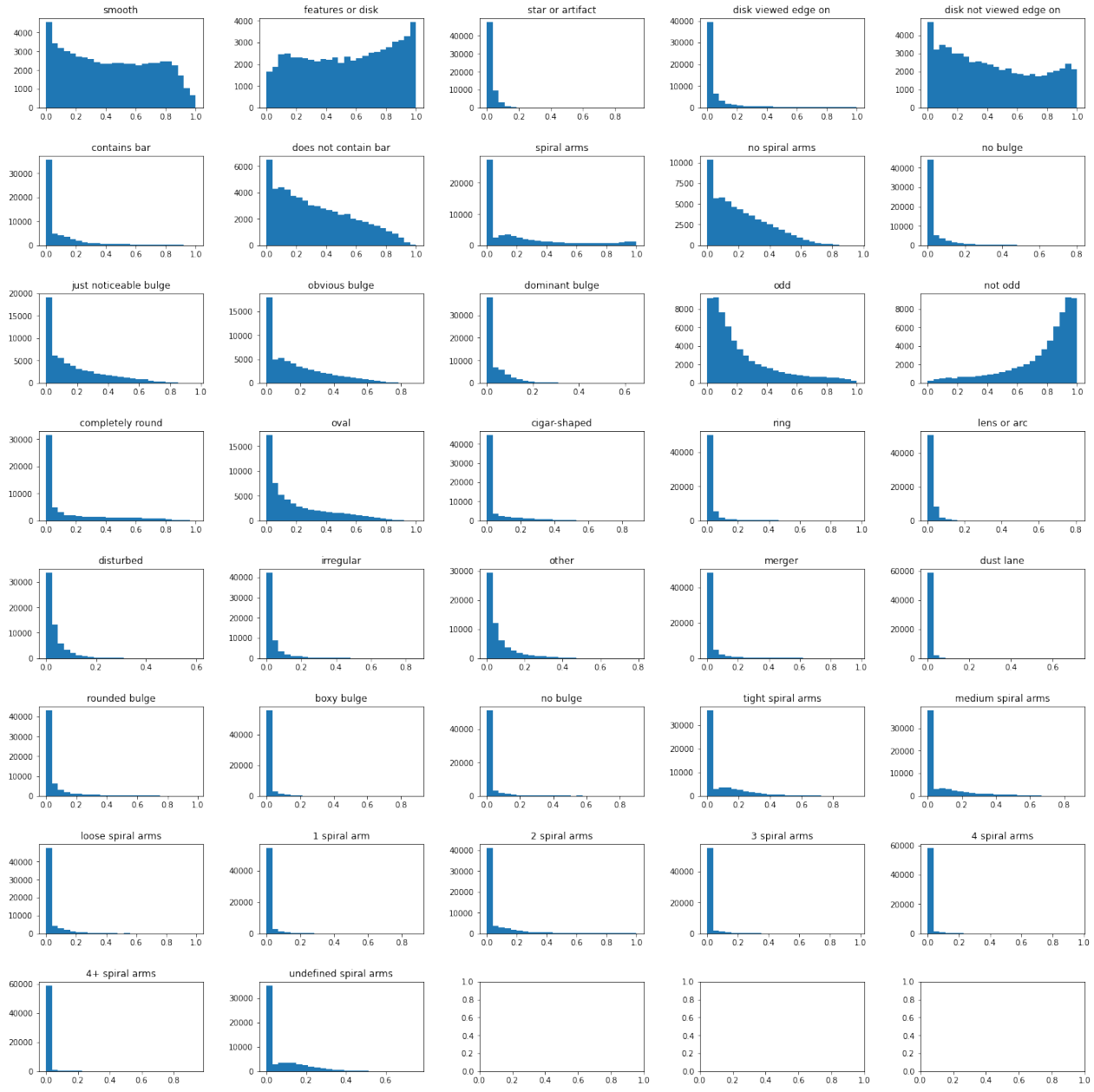

Figure 2: Histogram of the training labels. These essentially represent how often a specific feature of an image is classified with a certain probability. For example, left skewed distributions indicate that there are more galaxies that do not posses that feature, as many of these label histograms indicate. More uniform distributions such as the 'smooth', 'features or disk' and 'disk not viewed edge on' labels indicate that classification of these features are often uncertain and blurred.

2.3 Handling Memory

The SDSS galaxy images are 424×424 pixels with three color channels, making up 539328 input parameters. If we are to load all of these images into memory, it would require ≈ 33 GB. The resolution is quite high and there is alot of unnecessary and "verbose" information in our images. This would correspond to a large scaling of more parameters in our deep learning model

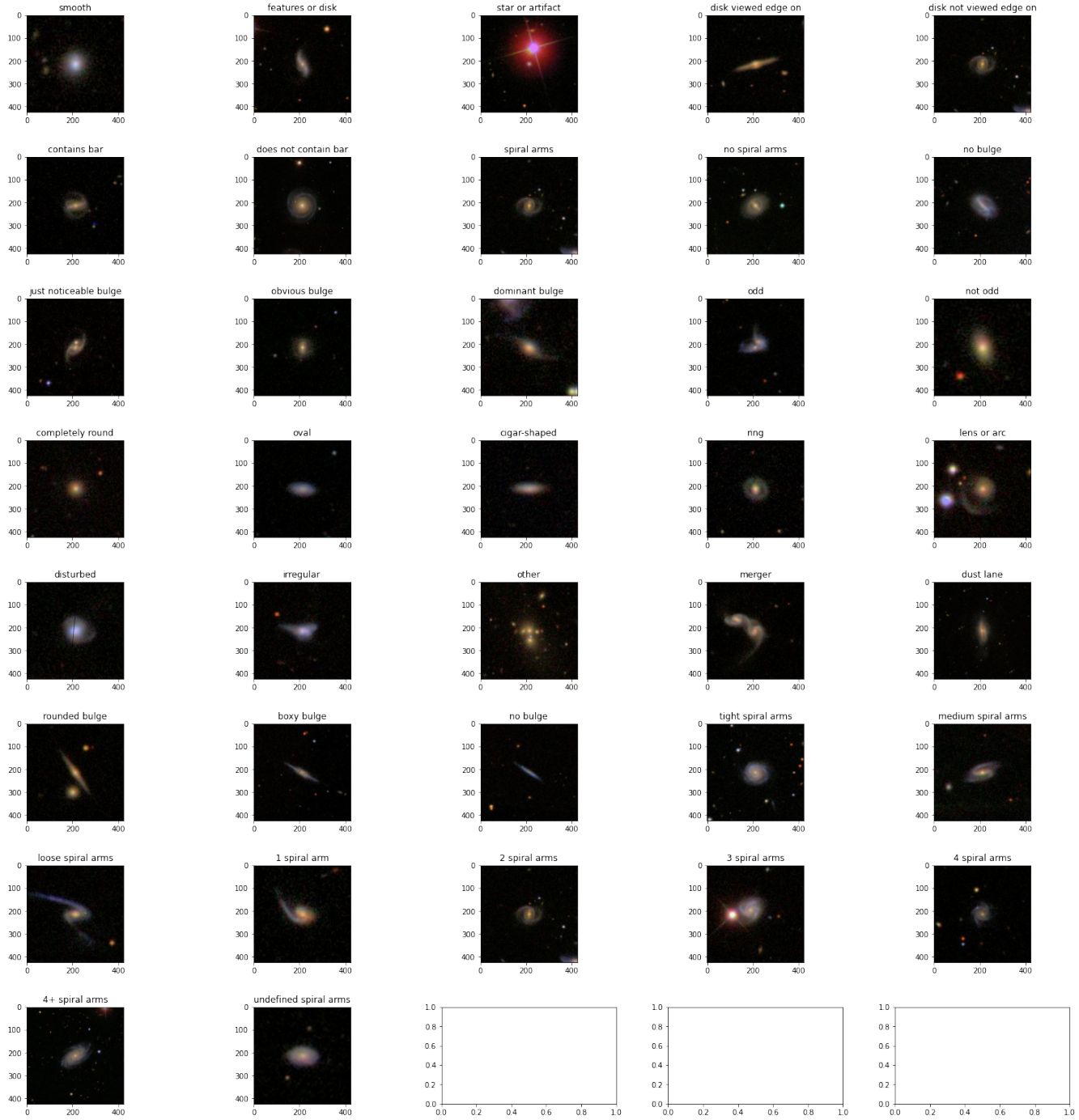

Figure 3: Prototype images of each of the 37 labels. The title of each image corresponds to its most confident feature, which should also be very agreeable to our common judgements. We should expect our neural networks to be able to, at least, confidently classify these images or at least very similar ones as its respective label extrema.

than necessary for the model to make a competent classification. So we direct our attention to reducing the amount of input parameters. Because most of the image is black (empty space), and that the galaxy is centered in the image, cropping is applied to the images so the galaxy

takes up more space in the image. Furthermore, we downscale the resolution of the image to a point where features can still be discernible by the human eye. The three color channels are maintained in the data and no normalization is needed, as the galaxies images have similar brightness values as already processed by SDSS. The final size of our images are 68×68 with three color channels, making 13872 total input parameters. Loading the downsized images into memory would now be ≈ 0.86 GB.

To further preserve memory without having to load all images into memory, we can utilize generator functions, which yields a fraction of the data set. This serves as the direct input into our deep learning models. The generator function yields an infinite batch of images upon being continuously called, looping through the training data set. The same is done for a validation set.

2.4 Loss Function and Guess Model

To understand the loss function used to optimize our models, we first input our data into a crude guessing model, in which our 'model' guesses each image to be the average value of each of the labels of the training set. The loss function used is the root-mean-squared-error (RMSE):

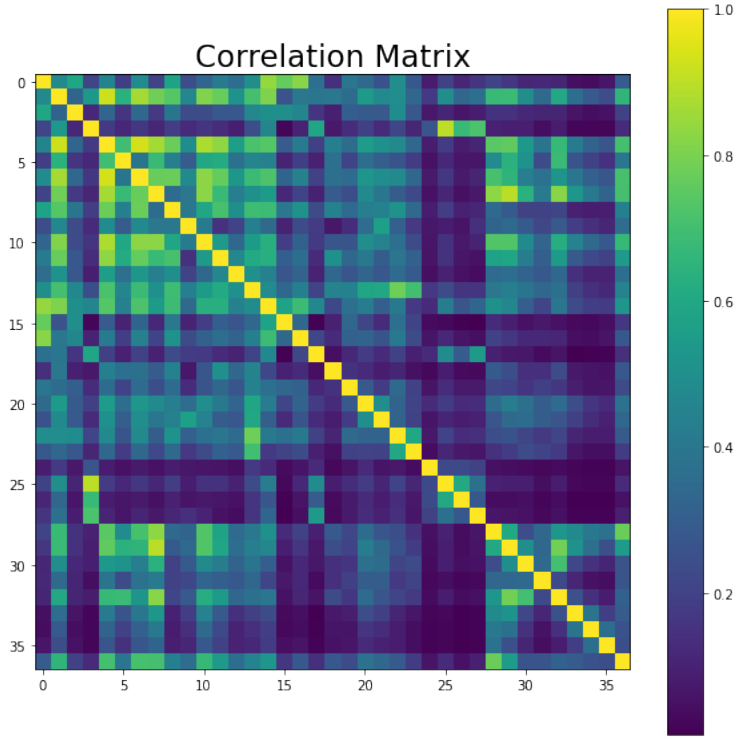

Figure 4: Correlation matrix of the labels visualized as a heat map. As expected, specific feature labels are correlated with themselves, as shown by the yellow diagonal elements. Because features of galaxies are not uniformly distributed, we can also expect there to be correlations between features that are more common than others.

$$L_{RMSE} = \sqrt{\langle(\vec{l}_{true} - \vec{l}_{predicted})^2\rangle} = \sqrt{\frac{1}{N_{galaxies} N_{labels}} \sum_{i}^{N_{galaxies}} \sum_{j}^{N_{labels}} (l_{true,ij} - l_{predicted,ij})^2} \quad (2)$$

We split 80% of our data (about 49,000 images) to a training set and 20% (about 12,000 images) to a testing set. We ensure that the feature label distributions are similar between the training and validation set by inspecting the plotted histogram as similarly shown in figure 2. Using the crude guess model, an RMSE loss of ≈ 0.16 (0.0256 MSE) is obtained. We expect that a trained neural network will perform much better, in other words, reach a lower loss. The loss metrics returned by the Keras models are MSE so it is the metric we will be using for further analysis on model performance.

3 Convolutional Neural Network (CNN)

3.1 Model Architecture

The first deep learning architecture we use is a convolutional neural network (CNN). The core element of a CNN is the usage of filters, represented as square or rectangular grid of trainable weights. This filter is applied across the image, hence a convolution. Each section of the image that the filter covers will correspond to a mapping to a single output neuron. The motivation for this structure is so that trained weights could learn shapes and features that span from high scale structures to finer, more detailed shapes within the image. Here, we outline the architecture of our CNN.

First we describe the initial 'block' of our CNN. The inputs are first passed into a convolutional layer, using 32 filters with size 3×3 and stride of 1. The stride indicates how many pixels the filter 'steps' to make another computation to map to another output neuron. Thus, each filter has 9 trainable weights from the convolution filter and one trainable weight of the bias. Accounting the three color channels, this makes a total of 896 trainable parameters. The output of the convolutional layer of is then passed into a batch normalization layer that normalizes the input to being between -1 and 1 by dividing by a learned mean of the training dataset. This prevents a so-called covariate shift, in which the model possibly changes the distribution of the input data while training, causing over-fitting. Batch normalization also prevents the gradient in backpropagation from becoming too small. Then, the output of the batch normalization layer is passed through an activation function, in which the specific function used is the rectified linear unit, or "ReLU". Activation functions are a simple way introduce nonlinearity to our model, which allows our model to fit for more complex and nuanced functions. The output of the activation function is then passed through a max pooling layer, which down-scales the output by half, by taking the max value of 2×2 sections of a two dimensional input. This simply reduces the scale of computation and training time. Finally a dropout layer is introduced at the end of the block. This layer randomly turns off the output of a fraction of neurons at the end of the block, which we set this fraction to be 0.1. This also prevents the model from over-fitting on the training set, and can even lead the model to make more general and robust predictions on the validation/testing set.

The output of the first block is passed through the second block which contains the same layers except for batch normalization. The choice to omit batch normalization in subsequent

blocks was more or less arbitrary, but a motive can be to reduce unnecessary complexity of our model. The second block's convolutional layer contains 64 filters with size 3×3 and stride of 1. We then have a final block that has 128 filters of size 6×6 with a stride, and consists of the same layers as the second block. After the convolutional blocks, the output is "flattened", which reduces the dimensionality of the output and passed into a "Dense" layer, or fully connected layer, which a specified number of neurons is linear combination of every input from the previous layer in addition to a bias parameter. This Dense layer is set to have 128 output neurons and passed through a ReLU activation function. Adding a dense layer increases the number of trainable parameters by a large scale so a dropout layer of 0.15 is used to reduce complexity which prevents over-fitting and computational time. The output of this dense layer is then passed through another dense layer with 64 output neurons with another ReLU activation function and dropout layer. A final dense layer is used to map the inputs into 37 classes, which corresponds to the galaxy feature labels of interest. A sigmoid activation function is used to constrain the outputs from being between 0 to 1, that would correspond to denoting probabilities as our labels suggest. The total number of trainable parameters of the CNN is near 2 million.

3.2 Training Process

Backpropagation is the method in the training process that updates the weights of the model. In the training process, the input images are passed through the network and outputs 37 values for each input image. These 37 values are compared to the true labels that we want our model predict. The loss function, RMSE, is then calculated to determine how each trainable weight in our model should be updated by "propagating" the error through the layers, to reduce the RMSE. The optimization algorithm we use to update the weights is the Adaptive Moment Estimation, or Adam. Adam uses stochastic gradient descent, which essentially adds noise to the parameter space that the model is trying to minimize. This allows for quicker model convergence, as it will be less likely to be stuck in local minima due to the noise. Specifically, the Adam optimizer automatically tunes the learning rate for each trainable weight, which decides how much the weight should change with respect to the propagated error.

The batch size is a hyperparameter we can tune to optimize model training. It determines the number of images that is passed through the network until it calculates the error gradients to update the weights of the model. A large batch size will calculate a more accurate error gradient but it will take longer to compute, where a smaller batch size will improve model time but not necessarily model performance. A batch size of 128 is used.

The number of epochs dictates the amount of times the whole training set is passed through the model. Training the model for longer epochs can improve model performance, but it can eventually be over-fitting to the training set. We initially set the model to train on 100 epochs, but we use an early stop callback that terminates the training once the model stagnates on improving its performance on the validation set.

3.3 Callbacks

While the model is training, we want it to update the user or perform certain operations. The ReduceLROnPlateau callback function reduces the learning rate by a manually tuned factor if the validation loss does not improve after a certain number of epochs. This can prevent our model from jumping out of desired minima and converge with greater stability. We also utilize the CSVLogger callback function, which logs training loss, training accuracy, validation loss,

and validation into a csv file for each epoch. The model checkpoint callback is used to save the model in our local directory after an epoch only if its validation loss has reached a new low. The early stopping callback stops the model from training if the validation loss does not improve within a certain number of epochs.

4 ResNet50

4.1 Model Architecture

The ResNet (Residual Network) is a type of deep convolutional neural network that have been specialized for image classification. The model used within the Keras framework is the ResNet50, which includes 50 layers. He et al. 2015 [1] describes the architecture of ResNet models in detail. The motivation for designing the ResNet models is that as neural networks become deeper, they become subject to the vanishing gradient problem, where the error being backpropagated through the network become so small, that the trainable weights stop being updated. The core design that fights against this problem is the usage of skip connections. Skip connections are identity mappings between the input of a block and its output; in other words, the input is directly passed into the output along with the mappings of convolutional layers. These skip connections forces the model to look at the comparison between the input and the output of the block. This keeps the error gradient stable and ensures convergence. The ResNet50 model has about 23 million trainable parameters.

The ResNet model provided by Keras allows us to initialize the model with pre-trained weights from the ImageNet dataset. This dataset is a collection of over 14 million images of everyday objects, and the pre-trained ResNet50 model is trained to classify each object. It turns out that image classification models for different tasks are trained very similarly, as feature and shape detection is universal in identifying what is contained in an image. By initializing these weights, we are performing "transfer learning", which is the process of utilizing pre-trained weights on larger datasets, but it is then fine tuned for a more specialized task, which in our case is galaxy classification.

4.2 Data Augmentation

To help reduce over fitting, we can implement data augmentation, a common method to artificially increasing the size of the training set. Our model should be able to predict galaxy features regardless of its orientation in the image, so we implement generator functions that yield images from the training set that are rotated by a random angle prior to being downsized.

4.3 Rescaling

Although the sigmoid activation function at the end of our network allows our model to represent probabilties of features, it does not encode class dependencies as the true labels do. For example, class 1.1 denotes that the galaxy is smooth. If the user chooses this, they will be directed to class 7, a question that asks how rounded the galaxy is, where the possible answers are 'completely round', 'in between', and 'cigar-shaped'. The confidence of an answer in class 7 is dependent on the confidence in class 1.1, so the total probabilites of the answers in class 7 must add up to the probability of class 1.1. We implement a custom activation function using the get_custom_objects module within Keras to encode this information into our model.

5 Results

5.1 CNN

The implemented CNN was trained for 60 epochs in about an hour using a Google Colab GPU. The lowest validation MSE loss reached while training is 0.0106. As stated as before, the model is initialized to train for 100 epochs until the validation loss stagnates for 20 epochs. The initial learning rate was set to be 0.001. The ReduceLROnPlateau callback function was initialized with a patience of 6 and a reduction factor of 0.5. The learning rate scheduler has a minor reduction rate as the Adam optimizer automatically fine tunes the learning rate of each weights. The training and validation loss is plotted below.

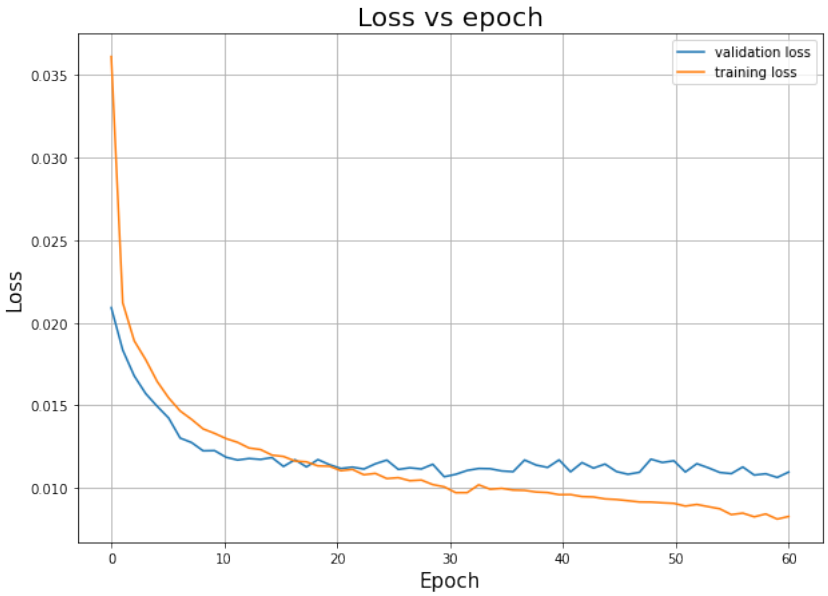

Figure 5: Graph showing loss versus epoch of the CNN model. After 20 epochs, the training loss overtakes the validation loss and the validaton loss stagnates, a sign of over-fitting to the training set.

5.2 Out-of-the-box ResNet50

To implement a default ResNet50 model, we firstv define a sequential model and add the ResNet50 model to the sequential model. Layers can now be tacked on following the ResNet. Then we simply flatten the output of the ResNet model and pass it into a dense layer with 37 outputs corresponding to our feature labels with a sigmoid activation function. The ResNet50 model also trained for an hour with 27 epochs. We expected that it takes longer to train an epoch on the ResNet model because of the large scale of the number parameters. The lowest validation MSE loss reached is 0.009102. The learning rate scheduler is identical to the one instantiated for the CNN.

5.3 Optimized ResNet50

The optimized ResNet50 model includes two additional dense layers with outputs 200 and 100 with drop out layers before being passed into the 37 final output labels. We also implement

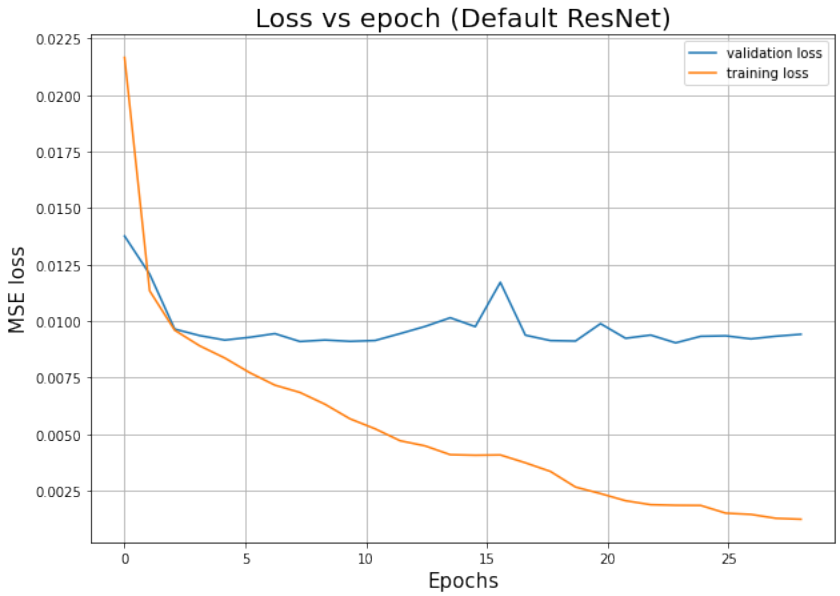

Figure 6: Graph showing loss versus epoch of the out of the box ResNet50 model. The validation loss quickly reaches a low at 3 epochs and stagnates for the rest of the training while training loss continues to decrease. The ResNet model is more subject to fluctuations in performance since the ratio between trainable parameters and input parameters in a batch is smaller than the CNN.

data augmentation generators that the model trains on and rescaling of weights via a custom activation function. The lowest validation loss reached was an MSE of 0.009899. The learning rate rescheduler has a patience of 4 and a reduction factor of 0.4. The patience for the early stop callback is 15. This was due to a repeatedly observed early stagnation of the default ResNet50 model. However it seems that it could be trained further as the validation loss has a continuously decreasing downwards trend after the initial dip.

5.4 Comparison

The validation loss and accuracy for each of the models are plotted in figure 8 and 9 respectively. The default ResNet50 model has the lowest validation loss, but its accuracy is comparable to the CNN model. The optimized ResNet50 reaches a much higher accuracy than both the CNN model and the default ResNet50 model despite performing comparably in the validation loss metric. This indicates that the optimized ResNet50 model is able to accurately label a good amount of the validation data with more precision than the other models. However, the default ResNet50 model, having a lower loss but lower accuracy, indicates that is slightly less precise, but is able to generalize better than the optimized ResNet50.

5.5 Model Predictions

We use default ResNet50 model, which reached the lowest loss, as our model prediction model. We predict on the validation set and plot one-to-one scatter plots in figure 10. The scatter plots are more spread out the closer it is to 0.5 probabilities, indicating uncertainty in the label, which is to be somewhat expected. However, the densities are high either at 0 and 1 probabilities for all of the labels, indicating that for many images, it is able to make more confident and discrete

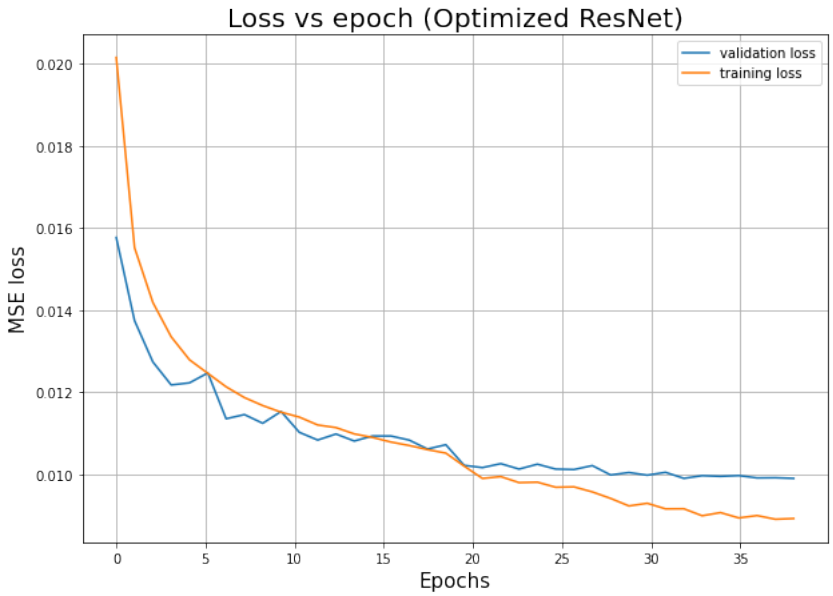

Figure 7: Graph showing loss versus epoch of the optimized ResNet50 model. The performance of the model over epochs is similar to the CNN model but reaches lower validation loss with fewer number of epochs. The ResNet model is still subject to fluctuations, but a decrease in the loss is consistent. If trained for more epochs, we can expect the model to perform even better.

	Validation MSE Loss	Validation Accuracy
CNN	0.0106	0.76770
Default ResNet50	.009102	0.765508
Optimized ResNet50	0.009899	0.818529

Table 1: Tabulated values of the most optimized metrics reached for the three models with about an hour of training time.

predictions on whether that feature corresponding to the label was present in the image or not.

5.6 Estimated Galaxy Merger Rate

Here, we outline the process of making a rough estimate of the galaxy merger rate at redshift z = 0. This is the redshift of interest as it approximately the distance of all the galaxies in our data set. We are given a set of 79,975 images to make predictions on, in contrast to our 49,000 images it trains on. This is a common usage of model deployment, where the model is expected to make predictions on data sets that are much larger than what it is trained on. We first decide a rough threshold of the merger probability by eye, by simply plotting galaxy images of the validation set with a merger probability greater than a certain value. The probability threshold was decided to be > 0.3, meaning that any probability above 0.3 would confidently categorize the image as a merger galaxy. Example plots of mergers from the validation set are shown in figure 12.

We employ this probability threshold to the predictions of the 79,975 testing images, and obtained 2324 confident mergers. This yields a galaxy merger fraction of ≈ 0.0290. Because

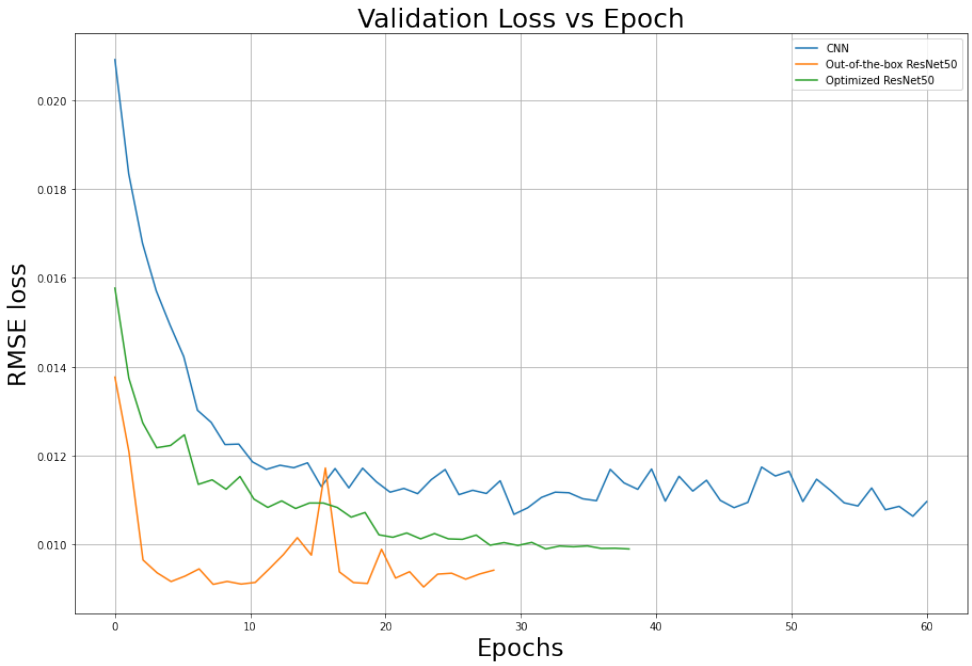

Figure 8: Plot of loss vs epoch for the three models of interest: CNN, default ResNet50, and optimized ResNet50. The default ResNet50 model fluctuates more than the other models, despite converging to a low loss much quicker. As shown, the optimized ResNet50 model can reach a lower loss with further training. It is unsure if the same could be said for the CNN model.

we can not validate that our model is perfectly accurate given the probability threshold, this value is simply a rough upper bound to the galaxy merger rate. However, this value is in good agreement to the results published by Lotz et al., 2011 [3] in figure 13.

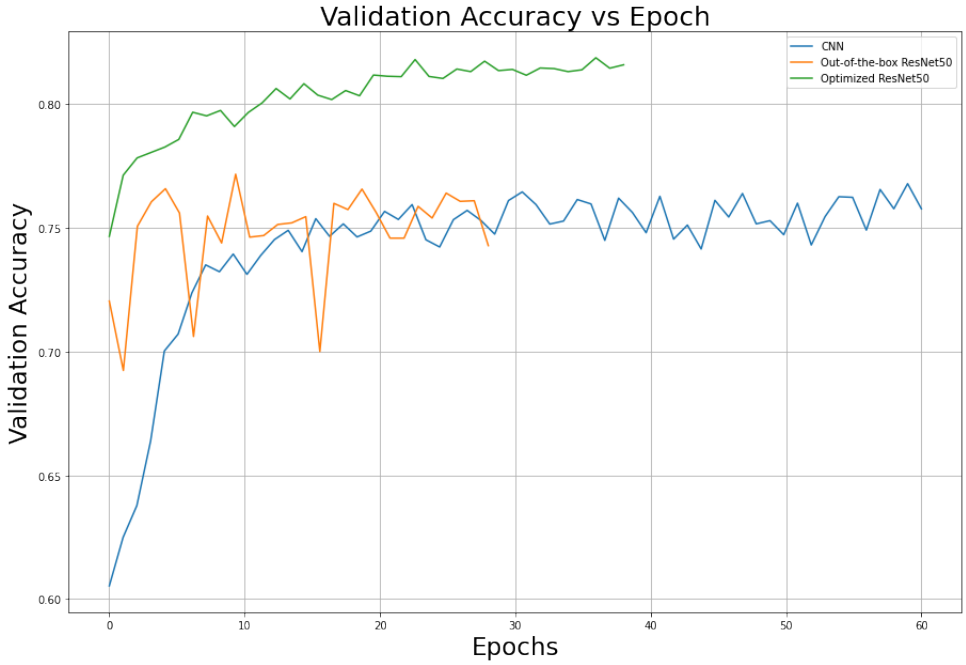

Figure 9: Plot of validation accuracy vs epoch for the three models of interest: CNN, default ResNet50, and optimized ResNet50. The optimized ResNet50 model has a much higher accuracy than the two other models, which those two models are comparable in accuracy. This likely due to our output rescaling to better match the class dependency logic of the true labels.

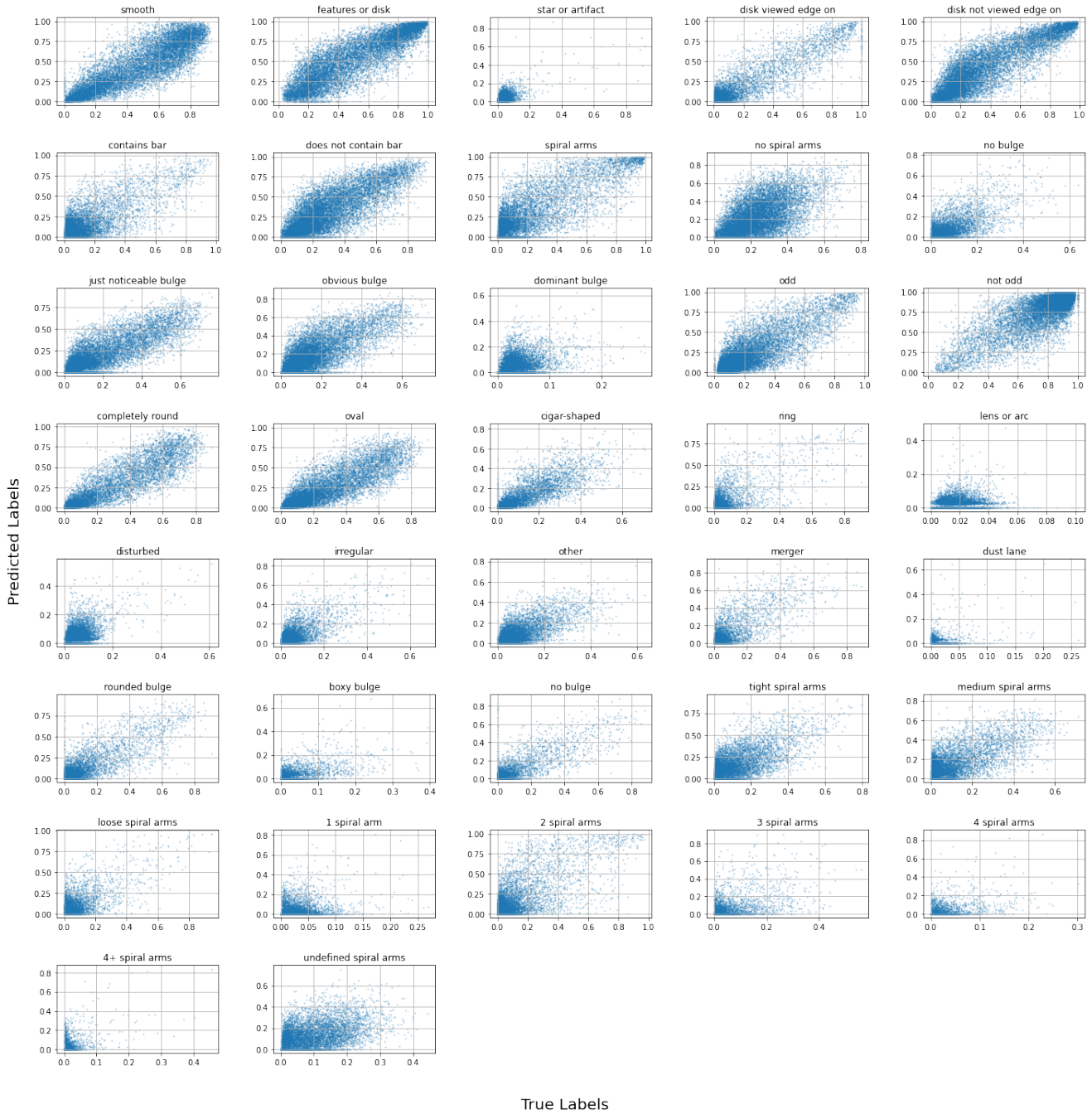

Figure 10: Plot of one to one scatter plots of predicted labels vs. true labels for all 37 labels. Higher densities in the 0 and 1 probabilities indicate that there are more confident classifications than not, meaning that our model is reasonable to an extent.

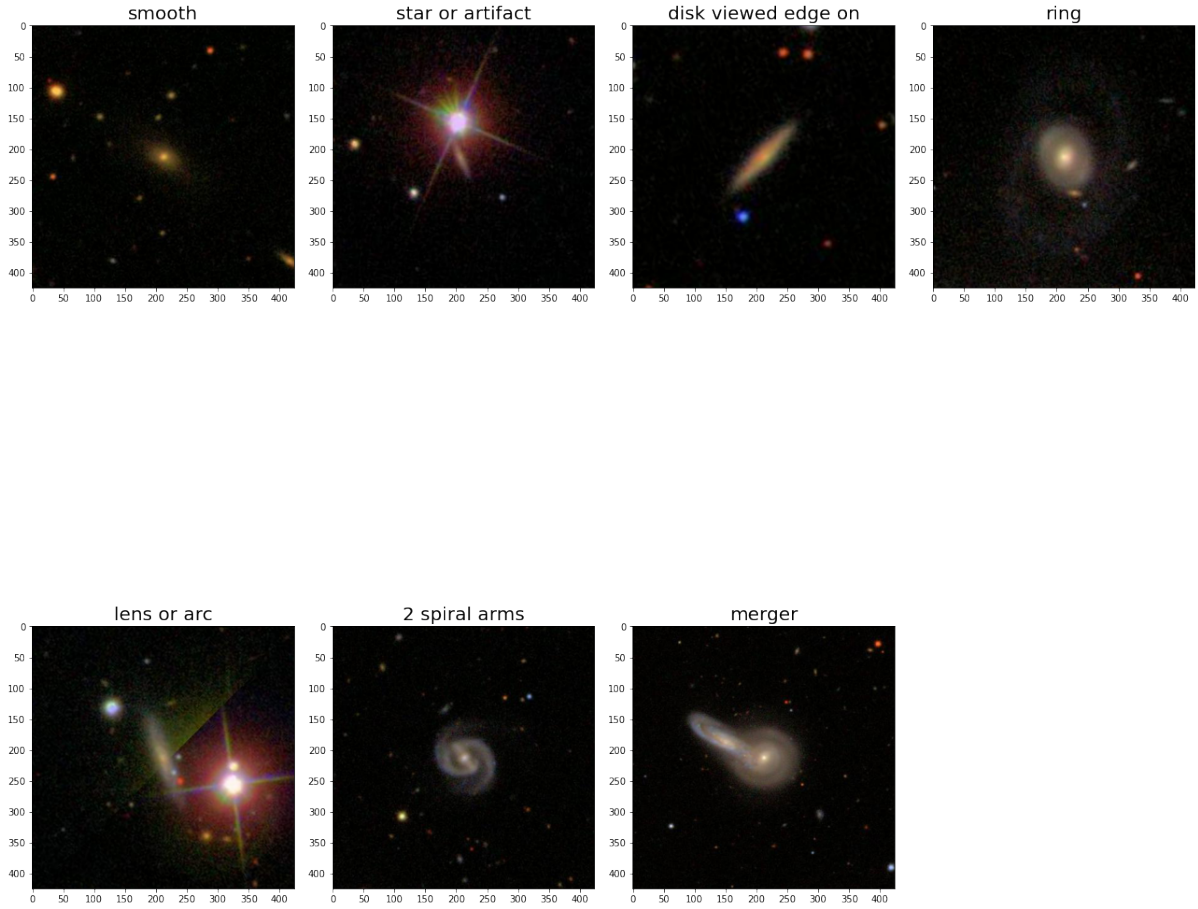

Figure 11: Images in the validation set in which the model characterizes as the most confident of the displayed label. This means that we can trust our model when it makes high confidence predictions, as the images display its corresponding feature very clearly.

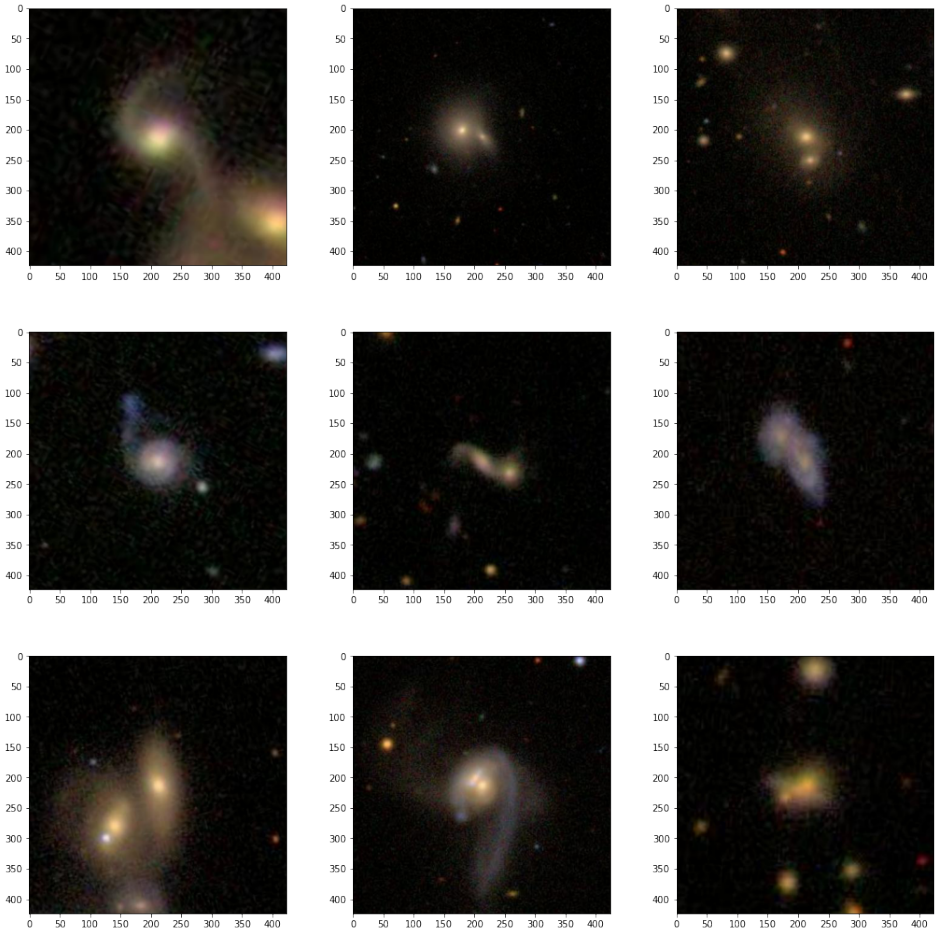

Figure 12: Images in the validation set in which the model predicts as a merger with probability > 0.3. The model is able to capture certain attributes of merging galaxies, such as the accretion of mass, even if two merging galaxies are not exactly discernible by eye.

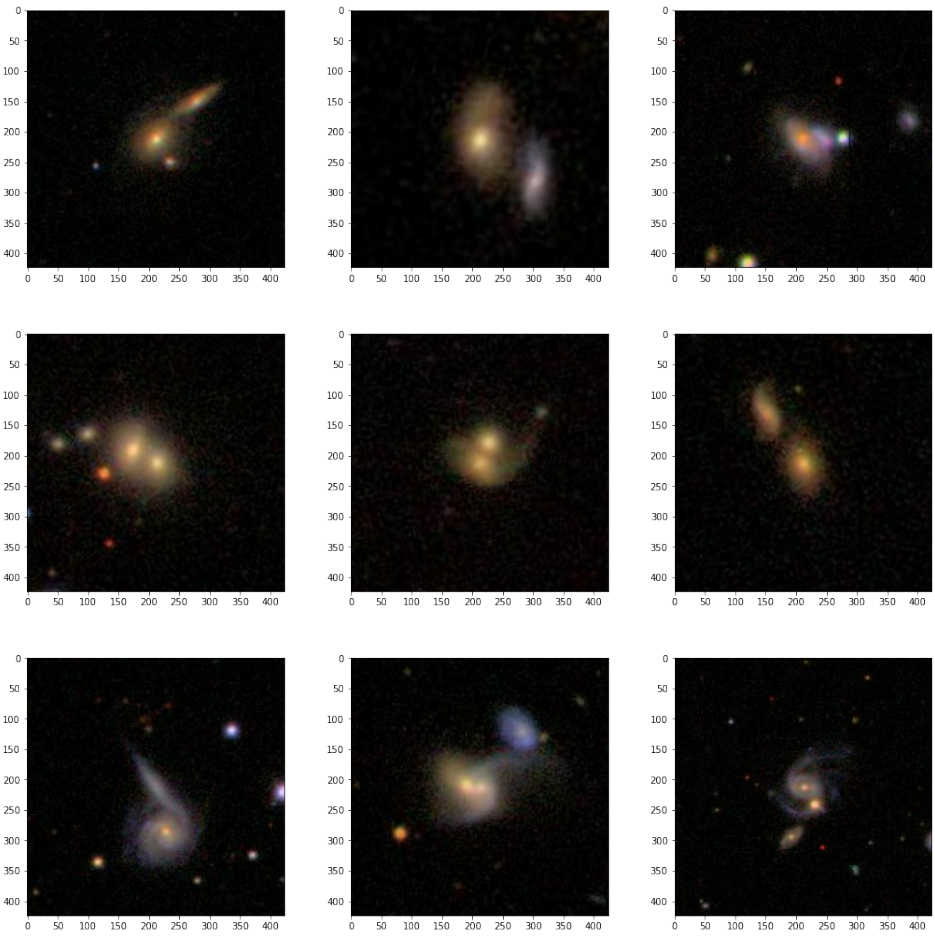

Figure 13: Images in the testing set in which the model predicts as a merger with probability > 0.3.

6 Conclusion

Comparing the custom CNN, default ResNet50, and optimized ResNet50 model, it is interesting that the default ResNet50 has a much faster and steeper convergence than the optimized ResNet50 model. This could be due to the addition of dense layers that complexified the optimized ResNet50 model. Although each model reached a comprable validation loss of ≈ 0.01, the optimized ResNet50 model reached a much higher accuracy (above 80 percent) compared to the accuracy of the other numbers. This likely due to the rescaling of the outputs which eased the model precision. The models can be optimized further, through training for more epochs, or further tuning of hyperparameters. Re-training the model can also improve the performance due to different instantiation of weights by sheer luck as well. In conclusion, this general classification task was satisfactory, as it was able to provide a reasonable galaxy merger rate at z = 0. However, there is still

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[2] Chris J. Lintott et al. "Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Surveysup72/sup". In: *Monthly Notices of the Royal Astronomical Society* 389.3 (Sept. 2008), pp. 1179–1189. DOI: 10.1111/j.1365-2966.2008.13689.x.

[3] Jennifer M. Lotz et al. "THE MAJOR AND MINOR GALAXY MERGER RATES ATiz/i&lt 1.5". In: *The Astrophysical Journal* 742.2 (Nov. 2011), p. 103. DOI: 10.1088/0004-637x/742/2/103.

[4] Kyle W. Willett et al. "Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey". In: *Monthly Notices of the Royal Astronomical Society* 435.4 (Sept. 2013), pp. 2835–2860. DOI: 10.1093/mnras/stt1458.