Galaxy Zoo: Morphological Classifications for Galaxies from the Sloan Digital Sky Survey

Lexi Torres, Jennifer Mei, Praveen Bandla
Department of Mathematics, University of California, Los Angeles

Abstract

To gain a better understanding of how galaxies form and evolve, it is imperative to differentiate between the primary morphological classes of systems. These systems include early types, late types, and mergers. This paper introduces the Galaxy Zoo project, which provides visual morphological classifications for nearly 1 million galaxies taken from the Sloan Digital Sky Survey (SDSS). Due to the large amount of image data, it makes visual inspection of each galaxy impractical for individual astronomers. For our project, we employ Keras, a popular deep learning library, to build a convolutional neural network (CNN) for the classification task. Our results highlight the significance of finetuning hyperparameters along with showing drawbacks in experimentation with limited datasets and computational processing.

1. Introduction

Galaxies are fundamental building blocks of the universe. The study of galaxies provides insights into the formation, evolution, and structure of the universe. To study such phenomena, Hubble Space Telescope was created. It was designed to explore the universe in visible, ultraviolet, and infrared wavelengths. To date, the telescope has more than one million observations. In recent years, the James Webb Space Telescope was launched to continue the journey of exploration in infrared astronomy. For most of the twentieth century, the classification of many cosmic objects, including galaxies was compiled by small teams of astronomers or individuals. As modern surveys, such as the Sloan Digital Sky Survey (SDSS), now contain hundreds of thousands of galaxies, this approach has become impractical. To anticipate the issues that could arise from surveys, Lahav et al. (1995) examined classifications made by experts who evaluated slightly over 800 galaxies. Their objective was to create a training set for neural networks, which could automate the classification process. In 1926, Edwin Hubble developed a classification scheme for galaxies known as the Hubble tuning fork diagram. The diagram divides galaxies into elliptical (early-type) and spiral galaxies (late-type).

Over the duration of this class, we have studied the significance of numerous hyperparameters including but not limited to, the optimizer, the number of epochs, the batch size, the learning rate, amongst others. In addition to the significance, we studied the impact of hyperparameter fine-tuning. For our project, we thus decided to investigate the impact of tuning said hyperparameters, including the model itself on a given dataset. We decided to investigate one model each, and a myriad of hyperparameter configurations to test and document the results of. This enabled us to further our understanding of the material through understanding firsthand an instance of the ramifications of changing our hyperparameters. Moreover, this enabled us to conduct an understudy on identifying the best configuration.

1.1 The Problem

Galaxy Zoo is a project that employs the collective intelligence of a large group of people to analyze the morphology of galaxies by utilizing images. Participants were presented with questions such as "To what degree is the galaxy round?" and "Is there a bulge at the center?". These questions stem from the Hubble tuning fork method. The answers provided by the participants are then used to determine the subsequent question to be asked. The questions are organized in a decision tree format, as illustrated in the figure below, sourced from Willett et al. 2013.

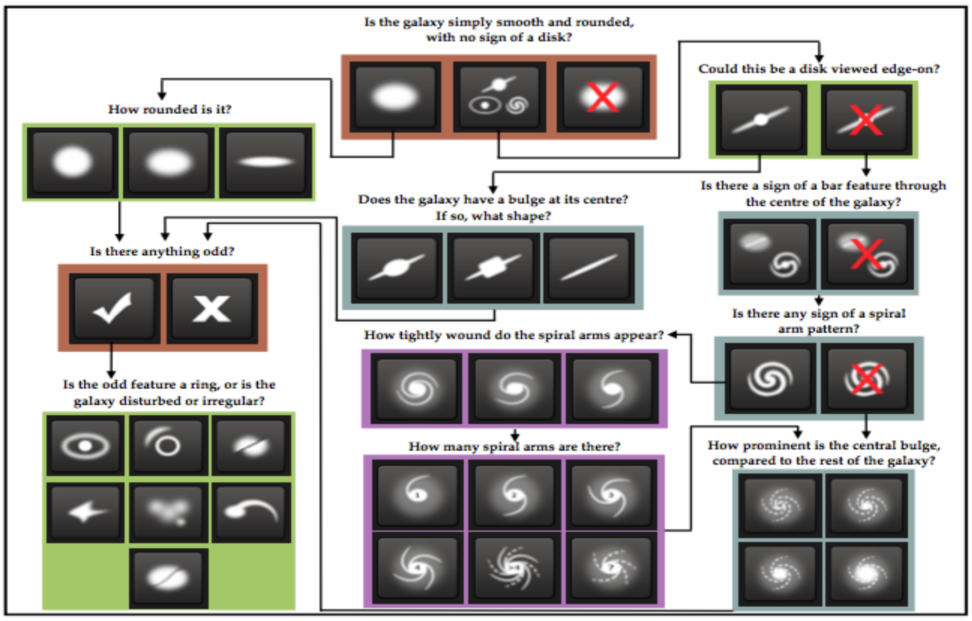

Is the galaxy simply smooth and rounded, with no sign of a disk?
Could this be a disk viewed edge-on?
How rounded is it?
Is there a sign of a bar feature through the centre of the galaxy?
Does the galaxy have a bulge at its centre? If so, what shape?
Is there anything odd?
Is there any sign of a spiral arm pattern?
How tightly wound do the spiral arms appear?
Is the odd feature a ring, or is the galaxy disturbed or irregular?
How many spiral arms are there?
How prominent is the central bulge, compared to the rest of the galaxy?

Figure 1. Flowchart of the classification tasks for GZ2, beginning at the top centre. Tasks are colour-coded by their relative depths in the decision tree. Tasks outlined in brown are asked of every galaxy. Tasks outlined in green, blue, and purple are (respectively) one, two or three steps below branching points in the decision tree. Table 2 describes the responses that correspond to the icons in this diagram.

In this paper, we will focus on designing a convolutional neural network to classify images using the decision tree shown above. To develop the most optimal CNN architecture, we researched and experimented with different models. We mainly focused on experimenting with three different models. The first model we experimented with was VGG16. The next model we experimented with was VGG19. The last model we experimented with was much smaller compared to the previous models. In the "methods" section of the paper, each group member will discuss in detail the other CNN architectures they experimented with, as well as the final model and why the final model was chosen.

2. Background

For our project, we chose to reference “Galaxy Zoo Classification with Keras” by James Lawlor. Lawlor used a VGG16 CNN architecture to perform the galaxy classification. The first part of the project starts with prepping the data. There are approximately 60,000 images in the training set and 70,000 images in the test set. Each image is 424x424 color JPEG. From inspection, Lawlor gathered that only the central part of the image is useful. The reason for this being is that the image contains a galaxy and surrounding the galaxy is black space. Therefore, Lawlor deemed we only want to focus on the middle of the image, cutting the black space. This led to Lawlor cropping the images from 424x424 to 212x212. In addition to cropping the image, Lawlor down sampled the image to a half resolution to reduce the number of parameters that must be tuned.

3. Methods

As described above each member of the group was responsible for researching and employing a model of their choice to see which model may be best for the final classification task. The model I researched and chose to test was VGG16. The next sections are based on my research and findings after experimenting with the model:

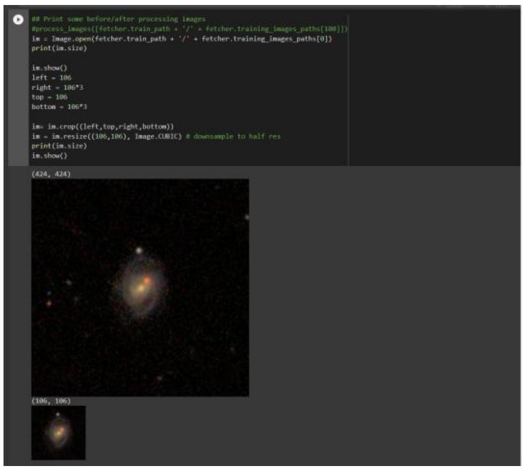
3.1 Model Selection: VGG16

The VGG16 model is a Convolution Neural Network (CNN) introduced by K.Simonyan and A. Zisserman from Oxford University in a paper called Very Deep Convolutional Networks for Large-Scale Image Recognition. It is based on the AlexNet architecture, making improvements by replacing the large filters (including a 11x11 receptive field) used in AlexNet with a sequence of smaller 3x3 filters. The proposed architecture consists of a total of 16 layers, including 13 convolutional layers and 3 fully connected layers (see section 3.3 for a detailed description). The results of testing the proposed architecture were outstanding: the VGG16 model achieved a test accuracy of 92.7% in ImageNet – a dataset containing more than 14 million training images across 1000 object classes.

The dataset was made publicly available on Kaggle following an open competition. Upon inspecting several of the high ranking papers/projects on the data, we noticed that the VGG16 was abundantly used. We thus decided to make one of the models that we were to test the VGG16, finetuning its hyper-parameters. The complexity of the model was a good fit with our multi-class classification task. We also acknowledge that the research regarding the potency of model architectures was beyond our knowledge, so we decided to use the model architecture as given without changing.

3.2 Download and Preprocessing of Data

The primary dataset includes 60,000 training images, with the test data hidden – the competition was set up such that each contestant would train the model with the given train data, and upon submission, the performance on the test score would be revealed to the contestant. From the Kaggle competition website, we were able to download the 60,000 images as a zip. However, in our storage in google drive, we encountered several issues resulting in only a partial 17,000 images being extracted into our folder unzipped. Due to the additional fact that we were constrained by limited processing, we realized we could not even make use of all 17,000 images for our experiment, and had to use a constrained subset of it. Thus, we felt that having only 17 of the 60 thousand images available did not hold us back in any way. Since we did not have access to the test data, we decided to create our own splits from the original training images to include training/test/validation subsets.



```
## Print some before/after processing images
#process_images([fetcher.train_path + '/' + fetcher.training_images_paths[100]])
im = Image.open(fetcher.train_path + '/' + fetcher.training_images_paths[0])
print(im.size)

im.show()
left = 106
right = 106*3
top = 106
bottom = 106*3

im= im.crop((left,top,right,bottom))
im = im.resize((106,106), Image.CUBIC) # downsample to half res
print(im.size)
im.show()
```

```
(424, 424)
```

```
(106, 106)
```

Each of the loaded images had a (424x424x3) native dimension. Because the default input for VGG16 was (224x224x3) and accepts no smaller than (200x200x3), we realized that it would be beneficial to resize the images to half their size. Since the images were of galaxies, a singular object, with most of them being in the center of their respective images, we decided to crop and resize – taking inspiration from Lawor's work. The coloring of the image remained unchanged when fed as an input to the model.

3.3 Data Management

As we were using Google Colab, we found that storing our data on Google Drive was the natural choice. This meant that we had to create many helper functions `batch_create`, `image_titles`, `generate_splits_and_files`, and many methods in class `data_getter` in order to facilitate handling our files/file paths. This proved to be far more tedious than initially expected. Moreover, we frequently were timed out by Google Drive or had our session blocked for excessive access. This added substantial time complexity to our models and slowed us down significantly. In hindsight, we could have preprocessed each image (thereby reducing the imagesize) globally before any further operation, instead of reading in the image, preprocessing it for the sake of our model, feeding it in and terminating. For future endeavors, we would look into more efficient image data management solutions including compressing.

3.4 Processing constraints

As highlighted in the section above, we were extremely constrained by the size of our GPU processing ability. Because we intended to run numerous experiments, tuning the hyperparameters differently each time, we needed to run many iterations of the models. Thus, for the most part, we only operated with a sample size of 2000-5000 of the images, using either an (80,10,10) or (75,15,15) split for train/test/validation and running on average for 5-10 minutes per configuration. This proved to be extremely limiting and added a lens of skepticism to the reliability of our accuracy results. In future endeavors, we hope to operate with larger RAM functionalities!

3.5 Hyperparameter configuration

This section outlines what was the core of our investigation for the VGG16 model. The hyperparameters we considered beyond the model itself were (along with the used values) were as follows:

- Learning rate: 0.01, 0.0001, 0.00001
- Optimizer: Adam, RMSprop
- Sample size: 2000,5000
- Ratio: (80,10,10) and (75,10,10)
- Batch size: 16, 32, 64

The things we kept constant were:

- Loss function: MSE
- Epochs: 100

Given that our computational processing was limited, we could not try each permutation of the hyperparameters (3x2x2x2x3 = 72 models). Thus, we adopted a binary tree structure of experimentation where at each stage, keeping every other hyperparameter constant, we noted the results of changing one hyperparameter to its available configurations and determined the best.

While this mechanism is not perfect, it meant we only had to test (3+2+2+2+1 = 12 models) a sixth of the total number of runs we wouldn't have to compute otherwise.
We decided to keep the loss function constant as we noted that the ideal hyperparameter configuration would completely change on differing loss functions and the binary tree mechanism of identifying the best set would not work. Thus, we decided to hold it as constant. Moreover, we noticed that RMSE was used to measure accuracy in the competition. In order to best ready our models to compare to the leaderboards, we decided to simply use MSE. In addition, given that we were plotting the history markers for each of the epochs, we felt that 100 epochs was effectively a time period for the model to test its convergence. We noted that nothing substantially changed after 100 epochs, so felt comfortable setting the limit to 100.

In evaluating each simulation, we looked at (between epochs 50-100):

- The average loss value: the lower the better
- The average variation in loss: the less varied the better
- Any convergence of the loss function: convergence was viewed favorably.
- If convergence, rate of convergence: quicker convergence was viewed favorably
- The difference between the Train and Validation accuracy: if they were close, it was viewed favorably
- The stability of the loss value: the more stable the better
- Computation time: lower times were viewed favorably.

For each simulation, we evaluated by collectively discussing based on the visuals we generated. We conducted one trial of each, and would have loved to expand that to say 3, if computation permitted. Any simulations that did not run within reasonable time were terminated. We also used the same GPU configurations for each as to not skew the results. First, we found the ideal configuration for each model, and then compared only the best between models. For the sake of concision, we chose not to include every single experiment result (only the most noteworthy ones are included). Moreover, in each of our papers, we included the results from our own models, and the result of the final chosen model after all comparisons.

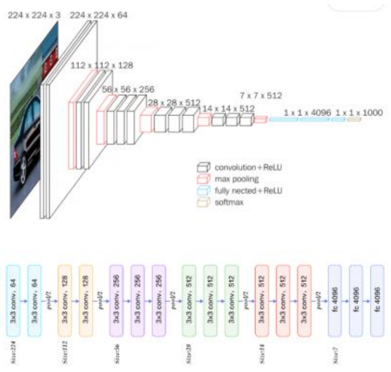
Before we go over the experiment results, we will detail the theory of the VGG16 architecture.

4. Theory – VGG16 model architecture

As described briefly above the model architecture we decided to implement for the final classification task is the VGG16 CNN architecture. The breakdown of the architecture is as follows:

- **Input layer:** The input to the network is an image of size 224 x 224 x 3 (i.e., width, height, and RGB color channels) by default, though the minimum accepted size is (200x200)
- **Convolutional layers:** There are thirteen convolutional layers. All of the layers use 3x3 filters and a stride of 1. The number of filters increases as you go deeper into the network, from 64 in the first layer to 512 in the last layer. Each convolutional layer is followed by a rectified linear unit (ReLU) activation function, which introduces non-linearity into the model.
- **Max pooling layers:** There are several pooling layers all of which use max pooling. Each of these layers uses a 2x2 filter with a stride of 2, which reduces the spatial dimensions of the feature maps by a factor of 2. The purpose of max pooling is to down sample the feature maps while retaining the most important information.

- **Fully connected layers:** The final layers of our VGG19 architecture are fully connected layers. There are three of these layers, two with 4096 neurons, and one with 37 neurons. The output of the last fully connected layer is fed into a sigmoid activation function, which outputs a probability distribution over the possible classes.



Note that the last layer's output was changed to 37 to account for the 37 class labels we have in this case.

5. Dataset Description

5.1 Sloan Digital Sky Survey

The Sloan Digital Sky Survey (SDSS) is a major astronomical survey that has been ongoing since 2000. It is one of the largest surveys of the sky, covering roughly a quarter of the celestial sphere. The SDSS uses a dedicated 2.5-meter telescope located at Apache Point Observatory in New Mexico, equipped with a suite of imaging and spectroscopic instruments. The main goals of the SDSS are to map the large-scale structure of the Universe. It studies properties of galaxies as well as other astrophysical objects and measures cosmological parameters that describe the evolution of the Universe. To study what was defined above the SDSS has collected data on millions of celestial objects, including stars, galaxies, quasars, and other exotic objects. The SDSS has produced a vast database of astronomical data, which is publicly available and widely used by astronomers around the world. The data is organized into various data releases, each of which includes new observations and improvements to the data reduction and analysis pipelines.

5.2 Galaxy Zoo Kaggle

The Galaxy Zoo: The Galaxy Challenge dataset is a collection of images of galaxies taken from the Sloan Digital Sky Survey (SDSS) as described above. The dataset was created as part of a Kaggle competition called the Galaxy Zoo Challenge, which was designed to test machine learning models in classifying galaxies based on their visual features.

The dataset contains a total of 61578 images of galaxies, each with a unique identifier. The images are divided into training and testing sets, with 54903 images in the training set and 6675 images in the testing set. Each galaxy in the training set has been labeled with one or more of 37 different classes based on its morphological features, such as the shape and structure of its arms and disk. These labels were generated by citizen scientists who participated in the Galaxy Zoo project, which aimed to classify many galaxies through a crowdsourcing platform.

The images in the dataset have been pre-processed to have a standardized resolution of 424 x 424 pixels and have been normalized to have zero mean and unit variance. In addition to the images, the dataset also includes a table of metadata for each galaxy, which contains information such as its redshift, the confidence level of its classification, and its location in the sky.
The goal of the Galaxy Zoo Challenge is to use machine learning models to predict the classes of galaxies in the testing set based on their images. The competition used a weighted multi-label log loss metric to evaluate the performance of the models.

6. Results

See the appendix for visualizations of the results.

6.1 The first configuration involved using the following hyper-parameters:

Learning rate: 0.0001; optimizer = Adam; sample size = 2000; ratio = (80,10,10); batch size = 32

We observe that the loss function hovers around 0.03, which implies that the RMSE averages at 0.173. The results appear to be stable and converge relatively quickly. Moreover, each training step averages around 2 seconds (though there is a process of things that need to finish before training the model). We conclude that this model performs relatively well.

6.2 Using RMSprop as the optimizer instead of Adam

Learning rate: 0.0001; optimizer = RMSprop; sample size = 2000; ratio = (80,10,10); batch size = 32

We note that the accuracy appears to be unstable, as there is extreme variability in the accuracy. Moreover, we noticed that the computation time of the process increased. We thus conclude that the Adam optimizer performs better than the RMSprop.

6.3 Reducing the batch size from 32 to 16

Learning rate: 0.0001; optimizer = Adam; sample size = 2000; ratio = (80,10,10); batch size = 16

We see that the accuracy has a healthy progression over the epochs (time). It converges relatively quick, just as in section 6.1, and the accuracy value is around the same. Moreover, there is stability in the accuracy – the stability here is the highest it is, even more than 6.1. However, the model took significantly longer to run. In the screencaps of the epoch history, the runtime can be seen to be around 3seconds opposed to the 2seconds of 6.1. Though the decision was not clear-cut, we conclude that 6.1 continues to be the best performing model

6.4 Increasing the learning rate from 0.0001 to 0.001

Learning rate: 0.001; optimizer = Adam; sample size = 2000; ratio = (80,10,10); batch size = 32

Not only was the loss really high, but unstable as well. We thus leave the learning rate at 1e-4

6.5 Lastly, decreasing the learning rate from 0.0001 to 0.00001:

Learning rate: 0.00001; optimizer = Adam; sample size = 2000; ratio = (80,10,10); batch size = 32

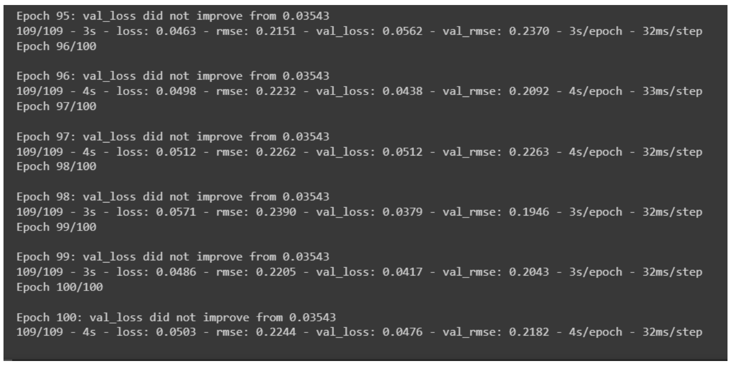
While the accuracy value and its stability are promising, the model appears to be taking longer to converge. The code also took significantly longer to run – probably as a result of having to constantly update the best weights with the convergence being more slow.

6.6 Final model

Upon deliberating as a group, we came to the conclusion that one of the specified configurations of the VGG19 was best to run. Here are the hyperparameters as listed:

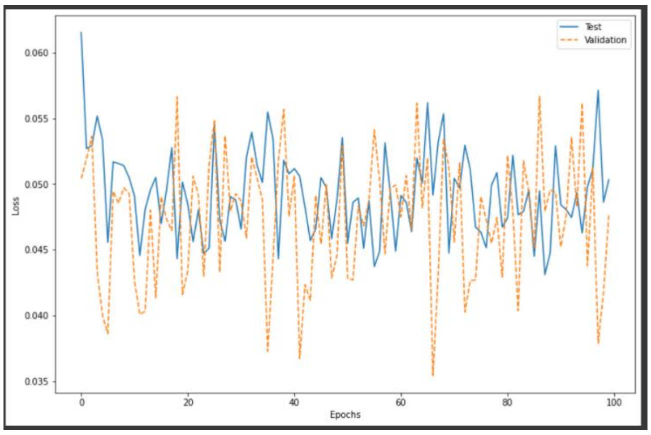
Learning rate: 0.0001; optimizer = Adam; sample size = 5000; ratio = (80,10,10); batch size = 32; model = VGG_19

While we felt that the model did slightly worse on the smaller 2000 sample, we felt that given its complexity and limited data in the first comparison, that it would perform better on a larger dataset of 5000 images, more than double the number of the first run. The last few epochs looked as such, and the final visualization is as follows:



```
Epoch 95: val_loss did not improve from 0.03543
109/109 - 3s - loss: 0.0463 - rmse: 0.2151 - val_loss: 0.0562 - val_rmse: 0.2370 - 3s/epoch - 32ms/step
Epoch 96/100

Epoch 96: val_loss did not improve from 0.03543
109/109 - 4s - loss: 0.0498 - rmse: 0.2232 - val_loss: 0.0438 - val_rmse: 0.2092 - 4s/epoch - 33ms/step
Epoch 97/100

Epoch 97: val_loss did not improve from 0.03543
109/109 - 4s - loss: 0.0512 - rmse: 0.2262 - val_loss: 0.0512 - val_rmse: 0.2263 - 4s/epoch - 32ms/step
Epoch 98/100

Epoch 98: val_loss did not improve from 0.03543
109/109 - 3s - loss: 0.0571 - rmse: 0.2390 - val_loss: 0.0379 - val_rmse: 0.1946 - 3s/epoch - 32ms/step
Epoch 99/100

Epoch 99: val_loss did not improve from 0.03543
109/109 - 3s - loss: 0.0486 - rmse: 0.2205 - val_loss: 0.0417 - val_rmse: 0.2043 - 3s/epoch - 32ms/step
Epoch 100/100

Epoch 100: val_loss did not improve from 0.03543
109/109 - 4s - loss: 0.0503 - rmse: 0.2244 - val_loss: 0.0476 - val_rmse: 0.2182 - 4s/epoch - 32ms/step
```

(correction: in the graph below, the blue line represents train, not test!)



7. Conclusion

The final results were rather surprising. We saw that VGG19 performed significantly worse on the larger data set than it did on the smaller one. Not only was the loss value significantly higher, but also the accuracy was unstable and did not appear to be converging by any stretch. Upon deliberating further, we concluded that that we would need more trials for each simulation to be able to understand the general case. The trends for the accuracy for any randomly selected subset of our data – given that said subsets are small – falls on a probability distribution of sorts, and one random sample is likely to not give us the full information we need to evaluate the performance of the model. We thus detail our simulation with the asterisk that our results are not robustly reliable. In a further outline of our study, we hope to have a higher GPU processing capability and are thereby able to run multiple trials of each simulation over a significantly larger training sample.

Within the configurations of VGG16, we see that the results of increasing and decreasing the learning rate, and reducing the batch size all had the hypothesized effect. Decreasing the learning rate presumably caused the model to develop an overshooting problem, escaping critical points and thereby not converging. Increasing the learning rate resulted in the model reaching the same conclusion, just slower – both in line with the material we discussed in this course. When we reduced the batch size, we are increasing the number of groups the model has to be computed over for each epoch and thereby extending the number of steps needed for the training process. This did not improve the quality of our results and we therefore conclude that leaving the batch size at 32 remained our best option. We also conclude, based on all our experiments, that Adam outperforms RMSprop in most aspects.

8. Reflection

Though the final model's result was generally disappointing, this project offered a fruitful ground for growth and learning. I was responsible for the coding section of the project and building the program end-to-end taught me a lot! I gained exposure to things I feel I would not have otherwise been made privy to, including information about model architectures, GPU performances, model checkpoints, and many more. Here are three takeaways/learnings from this project:

1. Efficiency and computation processing are far more important than I thought: in many ways, the success of our project was hindered by having the basic GPU performances. Moreover, a more efficient file management system would have likely sped up the process of things exponentially
2. Experimentation is a necessity: it is very difficult to determine what works and doesn't work in machine learning without same baseline experimentation. Configurations are extremely case dependent.
3. ML includes a lot of black boxes: There are several things including model architectures, optimizers (Adam especially), where there is much research done to identify improvements. For most of us, for all intents and purposes, we take things as a given without giving it much thought. Upon my own reading that followed, I realized that this is especially true for NLP related things, where so many of the embedding models are used by people who do not need to have an understanding of why they perform well, just that they do.

9. Data Availability

We retrieved our data from "Galaxy Zoo: The Galaxy Challenge" on Kaggle. It is not our own data. The data was provided by Galaxy Zoo.

10. Contributions Statements

Lexi was responsible for effective work planning and organization, setting clear objectives and prioritizing tasks for the group. She contributed to the communication and collaboration with team members to ensure everyone was aligned on project objectives and that work was being completed efficiently. Jen and Lexi developed the hypothesis and reviewed existing literature, as well as ensuring that the data collection and analysis were focused on answering the research question. They were also both responsible for data research, understanding the importance of starting with a clear understanding of the research question and making decisions based on data sources and methods. Praveen has an integral part in the development and implementation of the code. He was responsible for writing and testing code, as well as fixing any errors that arose in Jen's and Lexi's code. In addition to creating plots and models, Jen, Praveen, and Lexi all contributed to constructive criticism on the code, documentation, and writing for the report. All group members were responsible for the analysis of different models. We each selected different models to implement on our dataset, described above in methods, to see which model performed most optimally before choosing which model to implement in the final report. All group members contributed to writing parts of the final report; as a group we recognized the purpose and audience of the report, as well as the data and evidence used to support our conclusions. Each individual member was responsible for a common section of the report to maximize time and efficiency.

References

Blurredmachine. (2020, July 28). *VGGNet-16 architecture: A complete guide*. Kaggle. Retrieved March 10, 2023, from https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide

Galaxy Zoo - the galaxy challenge. Kaggle. (n.d.). Retrieved March 10, 2023, from https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge

Simonyan, K., & Zisserman, A. (2015, April 10). *Very deep convolutional networks for large-scale image recognition*. arXiv.org. Retrieved March 10, 2023, from https://arxiv.org/abs/1409.1556

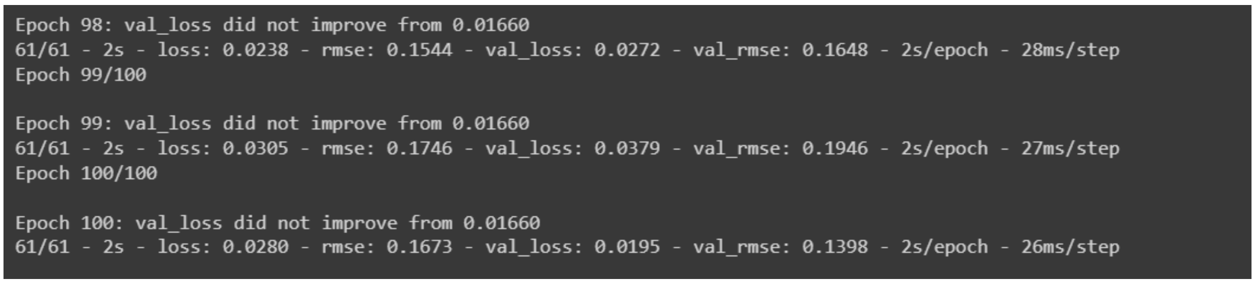
Understanding VGG16: Concepts, architecture, and performance. Datagen. (n.d.). Retrieved March 10, 2023, from https://datagen.tech/guides/computer-vision/vgg16/

Appendix

Note: The labels of the graphs are incorrect! The y-axis is Loss, while the blue line indicates train_loss, and the dotted orange line indicates validation_loss.

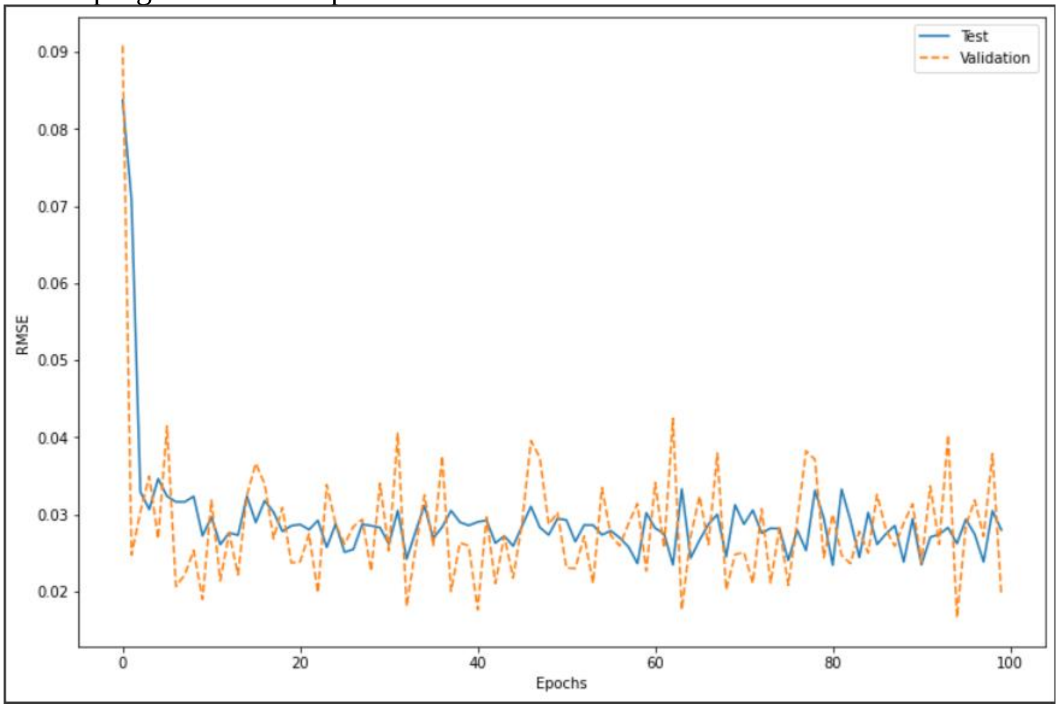
6.1

The model trained on these and here we can see the results of the last few epochs:



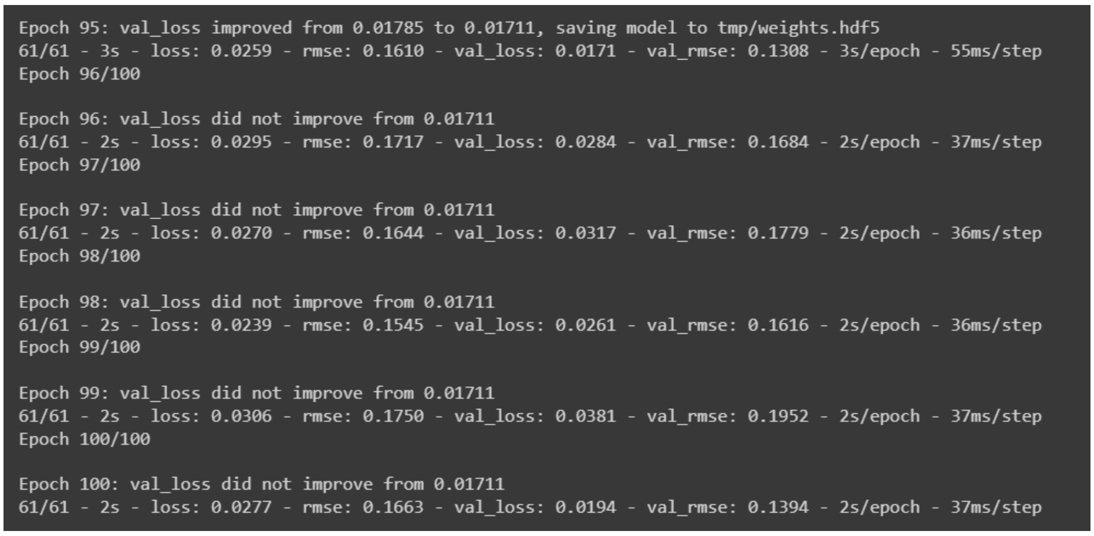
```
Epoch 98: val_loss did not improve from 0.01660
61/61 - 2s - loss: 0.0238 - rmse: 0.1544 - val_loss: 0.0272 - val_rmse: 0.1648 - 2s/epoch - 28ms/step
Epoch 99/100

Epoch 99: val_loss did not improve from 0.01660
61/61 - 2s - loss: 0.0305 - rmse: 0.1746 - val_loss: 0.0379 - val_rmse: 0.1946 - 2s/epoch - 27ms/step
Epoch 100/100

Epoch 100: val_loss did not improve from 0.01660
61/61 - 2s - loss: 0.0280 - rmse: 0.1673 - val_loss: 0.0195 - val_rmse: 0.1398 - 2s/epoch - 26ms/step
```

A visualization of the progression over epochs:



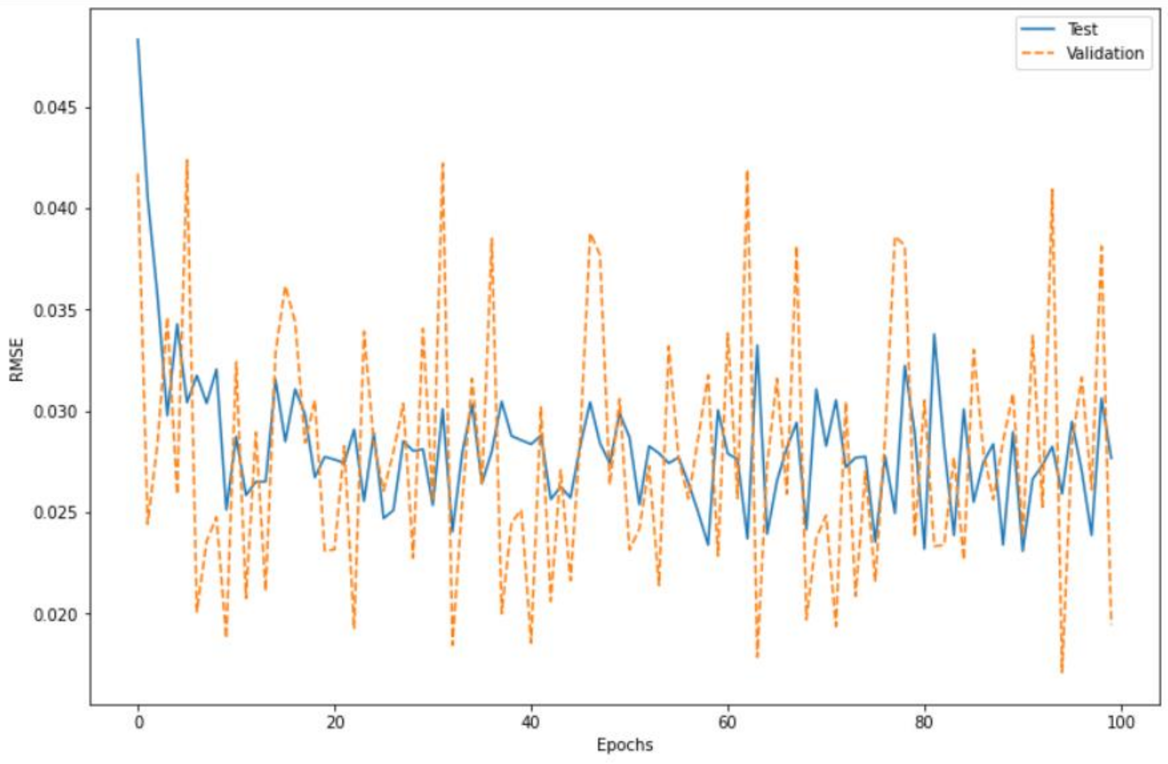
6.2

The model trained on these and here we can see the results of the last few epochs:



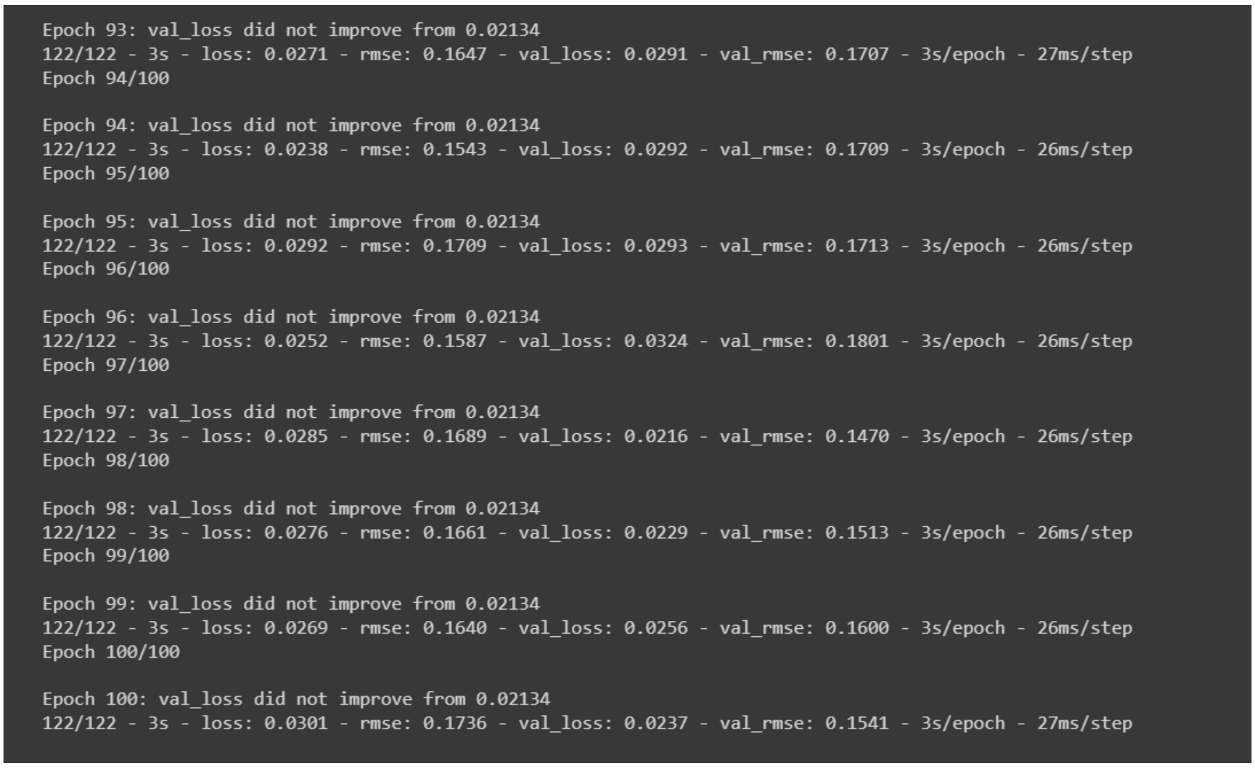
```
Epoch 95: val_loss improved from 0.01785 to 0.01711, saving model to tmp/weights.hdf5
61/61 - 3s - loss: 0.0259 - rmse: 0.1610 - val_loss: 0.0171 - val_rmse: 0.1308 - 3s/epoch - 55ms/step
Epoch 96/100

Epoch 96: val_loss did not improve from 0.01711
61/61 - 2s - loss: 0.0295 - rmse: 0.1717 - val_loss: 0.0284 - val_rmse: 0.1684 - 2s/epoch - 37ms/step
Epoch 97/100

Epoch 97: val_loss did not improve from 0.01711
61/61 - 2s - loss: 0.0270 - rmse: 0.1644 - val_loss: 0.0317 - val_rmse: 0.1779 - 2s/epoch - 36ms/step
Epoch 98/100

Epoch 98: val_loss did not improve from 0.01711
61/61 - 2s - loss: 0.0239 - rmse: 0.1545 - val_loss: 0.0261 - val_rmse: 0.1616 - 2s/epoch - 36ms/step
Epoch 99/100

Epoch 99: val_loss did not improve from 0.01711
61/61 - 2s - loss: 0.0306 - rmse: 0.1750 - val_loss: 0.0381 - val_rmse: 0.1952 - 2s/epoch - 37ms/step
Epoch 100/100

Epoch 100: val_loss did not improve from 0.01711
61/61 - 2s - loss: 0.0277 - rmse: 0.1663 - val_loss: 0.0194 - val_rmse: 0.1394 - 2s/epoch - 37ms/step
```

A visualization of the progression over epochs:



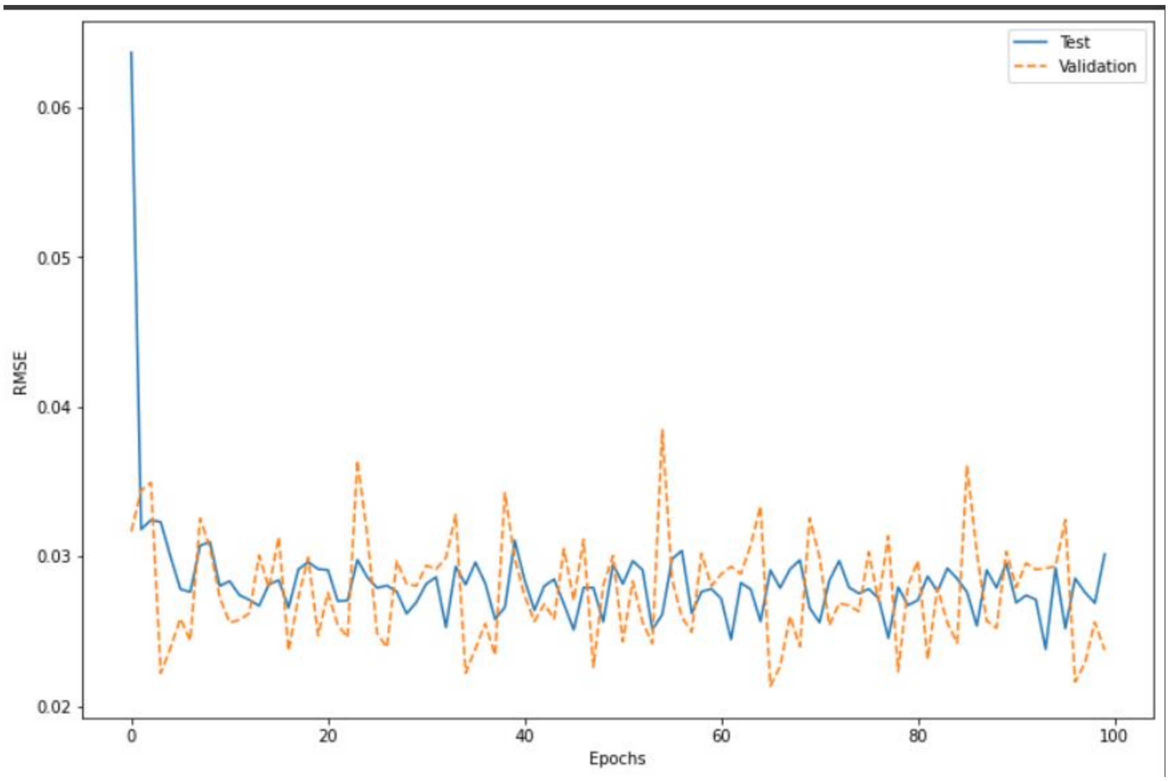
6.3

The model trained on these and here we can see the results of the last few epochs:



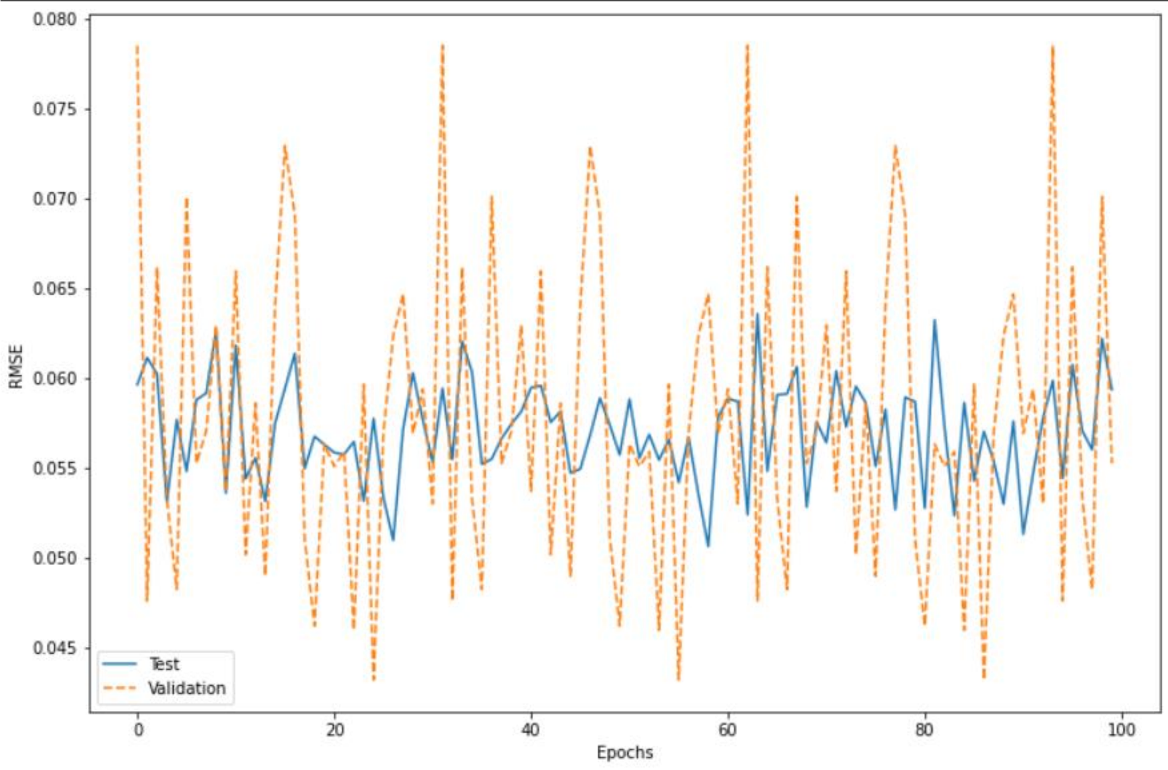
```
Epoch 93: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0271 - rmse: 0.1647 - val_loss: 0.0291 - val_rmse: 0.1707 - 3s/epoch - 27ms/step
Epoch 94/100

Epoch 94: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0238 - rmse: 0.1543 - val_loss: 0.0292 - val_rmse: 0.1709 - 3s/epoch - 26ms/step
Epoch 95/100

Epoch 95: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0292 - rmse: 0.1709 - val_loss: 0.0293 - val_rmse: 0.1713 - 3s/epoch - 26ms/step
Epoch 96/100

Epoch 96: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0252 - rmse: 0.1587 - val_loss: 0.0324 - val_rmse: 0.1801 - 3s/epoch - 26ms/step
Epoch 97/100

Epoch 97: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0285 - rmse: 0.1689 - val_loss: 0.0216 - val_rmse: 0.1470 - 3s/epoch - 26ms/step
Epoch 98/100

Epoch 98: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0276 - rmse: 0.1661 - val_loss: 0.0229 - val_rmse: 0.1513 - 3s/epoch - 26ms/step
Epoch 99/100

Epoch 99: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0269 - rmse: 0.1640 - val_loss: 0.0256 - val_rmse: 0.1600 - 3s/epoch - 26ms/step
Epoch 100/100

Epoch 100: val_loss did not improve from 0.02134
122/122 - 3s - loss: 0.0301 - rmse: 0.1736 - val_loss: 0.0237 - val_rmse: 0.1541 - 3s/epoch - 27ms/step
```

A visualization of the progression over epochs:



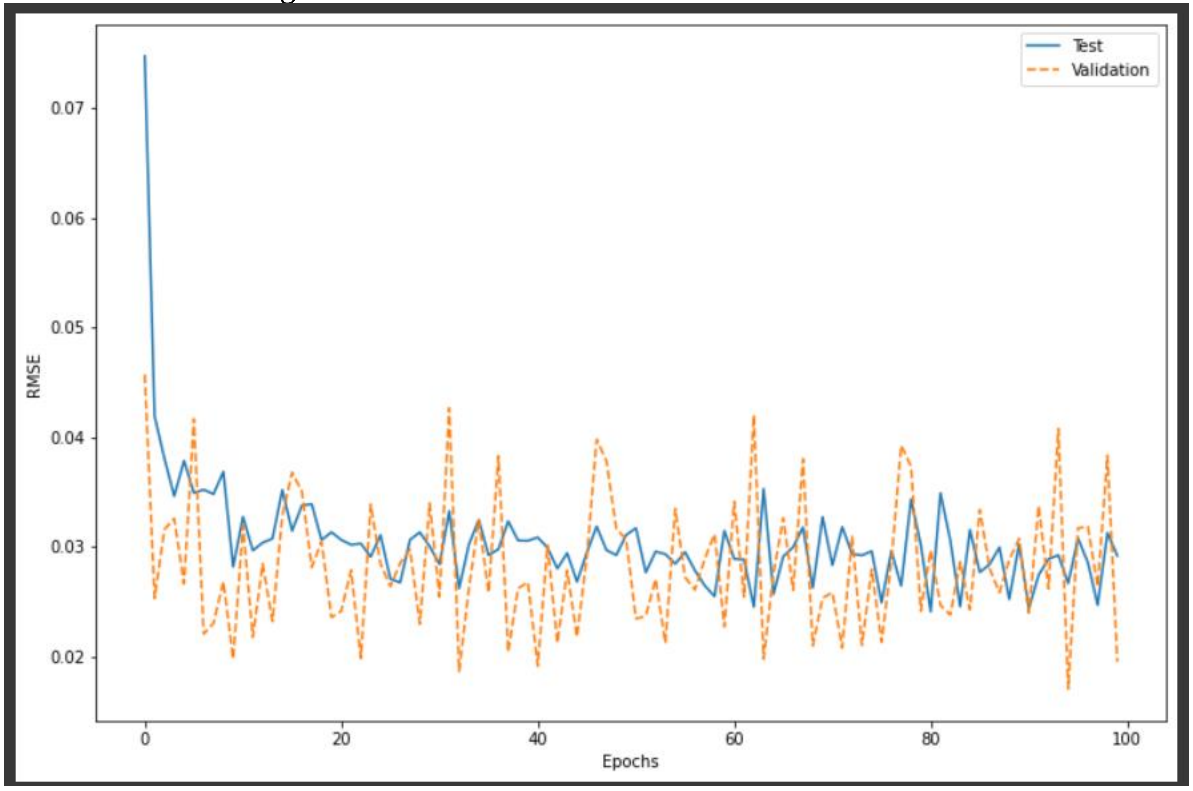
6.4

Increasing the learning rate by a factor of ten had the following effect:



6.5

This had the following effect:


RMSE
Epochs
Test
Validation