"Convolutional Neural Networks For Galaxy Morphology Classification" Machine Learning (course 1DT071) Uppsala University – Spring 2018

Diego Castillo Ankur Shukla Tristan Wright

October 24, 2018

Abstract

In 2014, Kaggle organized the Galaxy Zoo challenge to analyze images of galaxies from the Sloan Digital Sky Survey in order to automate the tagging of morphological attributes. For a machine to automatically learn the features directly from data, a deep convolutional neural network was implemented based on an existing architecture that placed highly in a competitive image classification competition a number of years prior. The dataset consisted of 79 975 test images and 61 578 training images which were cropped and downsampled for dimensionality reduction and memory efficiency. The model was trained with a variety of batch sizes and input formats. These input formats included further dimensionality reductions, data augmentation, and color space conversions. One of the methods ultimately attained a Top 100 score on the Kaggle leaderboards. Further advances can be done to improve the performance of the model by tuning other hyperparameters such as learning rate and number of epochs.

1 Introduction

A galaxy is a system of stars, dust, and matter grouped together by some center of mass. By studying galaxies, astronomers search for answers to questions like "are all galaxies the same size?" and "how and when did galaxies form?" Through the search for these questions scientists learn about the history of the universe and about the origins of our own galaxy.

As a consequence of this search for questions regarding heavenly bodies, astronomy is a research field ripe with large datasets [12]. Simultaneous to the enormous datasets, machine learning methods are rapidly becoming a go-to tool for automating data-intensive processes which normally require days or weeks of –often tedious– human processing. Due to the incomprehensible size of the Universe, classifying celestial bodies is an excellent candidate for the application of machine learning.

The Sloan Digital Sky Survey (SDSS) has made machine learning methods more viable by releasing large datasets to the public. Since SDSS' inception in 1998 and the first data release in 2000, the group has released over 14 datasets for public use. These datasets have been used for numerous applications of machine learning in the classification of astronomical bodies. For example, Ross Fadely in 2012 used support vector machines to classify galaxies and stars. However, they found in the end that they only preformed marginally better than template matching, a traditional image analysis method [18]. Decision trees have also been employed on the star-galaxy classification problem [1]. Thanks to more available computing resources in this century, artificial neural networks have been trained to classify galaxies and even detect black holes by analyzing images taken on the X-ray spectrum [10]. Convolutional neural networks (CNN's) have been a popular choice for classifying astronomical images due to their ability to extract features from images which humans might not otherwise select [8].

This paper demonstrates how a CNN can be constructed to analyze images of galaxies to automatically detect metrics that reproduce the probability distributions derived from expert and crowd-sourced human classifications.

The dataset comes from the Dark Energy Camera Legacy Survey (DECaLS), a subset of an SDSS dataset for specifically galaxies called "Galaxy Zoo". DECaLS uses a larger telescope, so it is 10 times

more sensitive to light than the survey that supplied images to the first iteration of Galaxy Zoo. This means there is more detail [23]. In 2014 this same dataset was used as the premise for a Kaggle competition.

2 The Dataset

The "Galaxy Zoo" dataset consists of a total 141 553 images. These are split into 61 578 images for training –each with their respective probability distributions for the classifications for each of the inputs– and 79 975 images for testing.

As a crowd-sourced volunteer effort, images of the dataset were classified across 11 different categories. Each of categories have attributes which volunteers can rank, there are 37 attributes in total. Some categories are dependent on the presence of others, for example number of spiral arms and spiral tightness is dependent on if the galaxy has spiral shape. The votes on these volunteer categorizations are normalized to a floating point number between 0 and 1 inclusive. A number close to 1 indicates many users identified this category for the galaxy image with a high level of confidence, while numbers close to 0 indicate otherwise. These numbers represent the overall morphology of a galaxy in 37 attributes.

Figure 2 shows how each galaxy's classification is the result of a specific path down the decision tree. The root of the tree consists of more general questions, and as the tree grows, the questions become more specific. The appearance of subsequent questions is dependent upon previous responses. Classifications performed by multiple users for the same galaxy result in multiple paths along the tree which generate probabilities for each question. The classification probability for each node sums up to 1.

Due to these probabilities being tied to each image this is a regression problem rather than a classification problem. Therefore, the task is to determine the degree to which a galaxy has certain attributes and mimic how people would classify the same galaxy.

3 Theory

3.1 Neural Network

A standard neural network (NN) consists of many simple, connected processors called neurons. Each neuron produces a sequence of real-valued activations. A neuron can be activated from an input or another neuron's activation through its weighted connections from a previous layer [19].

Figure 3.1 shows how a NN receives a number of inputs which are connected to an activation function through weighted edges. A NN can have any number of inputs, hidden layers, or outputs.

An activation function computes a weighted sum of its input, adds a bias and decides whether the neuron's value should be propagated or not. A common choice for an activation function is the Rectified Linear Unit (ReLU) [16]. A ReLU is defined as: $\sigma(x) = \max(0, x)$, where x is the weighted sum of the inputs. It returns an output x if $\sigma(x)$ is positive, 0 otherwise. NN's using primarily ReLU activation functions have been shown to enable faster training even with many layers [8].

It is possible for ReLU to introduce dead neurons whose output is always zero, this problem is known as the dying ReLU problem. To mitigate this issue, a slightly different version of ReLU known as leaky ReLU can be used. Leaky ReLU is defined in equation 3.1.

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise} \end{cases} \tag{3.1}$$

A loss function defines the difference between the target and actual output [9]. A NN uses the loss function to correct weights after a feed forward operation, this process of correction is called back propagation. The error is propagated backwards from the output layer through the hidden layers to the input layer, wherein the weights and biases are modified in such a way that the error for the most recent input is minimized. Over many training samples the loss function will minimize error and the value of the weights for the whole network will begin to converge. The curve which the error rate of the network experiences is controlled through a gradient descent method. This method can help determine whether the network should be trained further or to stop early.

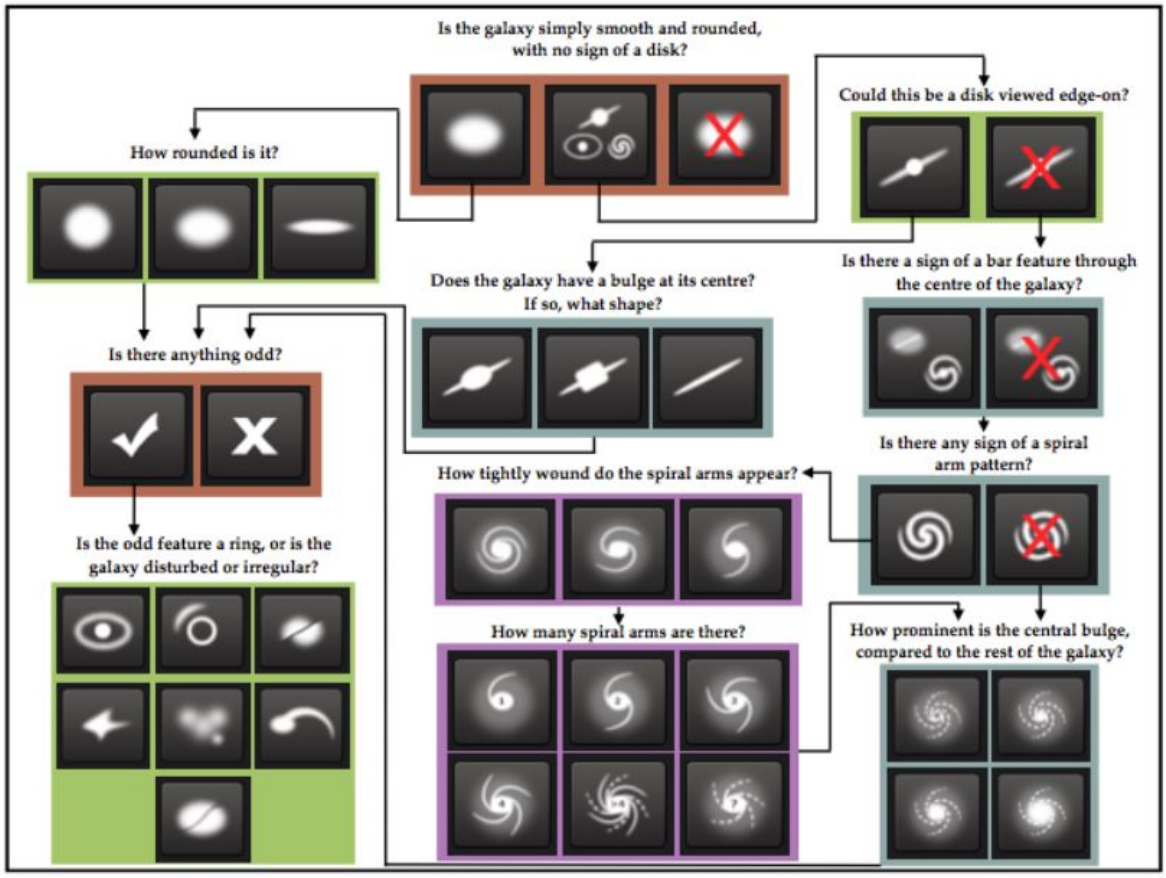

Figure 2.1: Flowchart of the classification tasks for the "Galaxy Zoo" dataset. Tasks are color-coded by their relative depths in the decision tree. Tasks outlined in brown are asked for every galaxy. Tasks outlined in green, blue, and purple are one, two or three steps respectively below branching points in the decision tree [22].

3.2 Deep Learning

Deep learning is a set of algorithms in machine learning that attempt to learn in multiple levels, corresponding to different levels of abstraction. It is based on learning several levels of representations, corresponding to a hierarchy of features, where higher-level concepts are defined from lower-level ones, and the same lower level concepts can help to define many higher-level concepts. It typically uses artificial neural networks. An observation (e.g., an image) can be represented in many ways (e.g., a vector of pixels), but some representations make it easier to learn tasks of interest through examples (e.g., is this the image of a human face?). Research in this area attempts to define what makes better representations and the best methods for a NN to approach learning them [6].

3.3 CNNs

A CNN is a type of deep feed-forward neural network [8] which is able to extract elementary visual features from its input. The creation of CNNs was motivated by Hubel and Wiesel's discovery in [4], where they were able to find that a cat's visual cortex has locally sensitive, orientation-selective neurons. CNNs were first introduced in 1999 [15], and since then have been applied to solve numerous different type of problems in natural language processing [3], image recognition [8], and recommendation systems [21].

A CNN takes an image as an input and feeds it through several layers; usually convolutional layers with ReLU activation functions, pooling layers, and a fully-connected layer (FCL). Convolutions are the

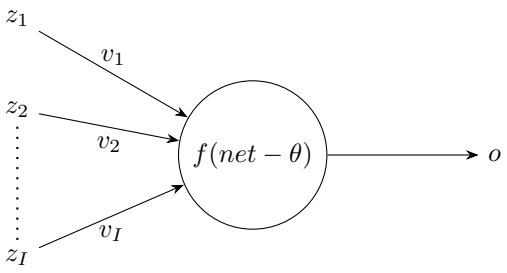

Figure 3.1: A NN which receives the inputs $z_1, z_2, ..., z_I$ and connects them to the neuron through the weighted edges $v_1, v_2, .., v_I$. The input values are multiplied by their respective weights and subtracted from a threshold (θ). The result is passed to an activation function which produces the output [9].

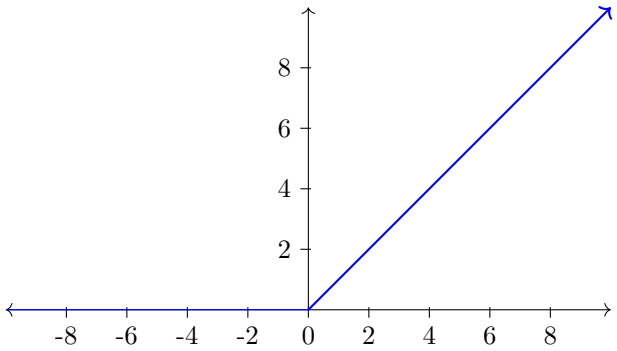

Figure 3.2: A ReLU function [5].

primary operation of a CNN and what makes them distinct from other type of networks. A convolutional layer parameters is made up of a set of small learnable weights known as filters. A filter has a local receptive field, and given an image as an input to a convolutional layer, it convolves each filter across the image's width and height using a specified stride size and produces outputs called feature maps [8]. Filters are what allow a CNN to learn to extract visual clues from its input such as edges, lines, and corners [15]. Equation 3.2 from [8] gives the mathematical definition of a feature map k. The summation is performed over the set of input feature maps, the symbol $*$ describes the convolution operator, and $\boldsymbol{x}_m$ the filters.

$$y^k = \sigma\left(\sum_m \boldsymbol{w}_m^k * \boldsymbol{x}_m + b^k\right) \tag{3.2}$$

The filters of a CNN compute linear element-wise multiplication and additions to create the feature maps. In order to add non-linearity, it is common for the convolutional layers to use ReLUs as the activation function which additionally allow for faster training in networks with many layers [8].

Feature maps are then fed through pooling layers. A pooling layer is typically of size 2×2 [14], and its job is to essentially reduce the resolution of the previous feature map. A pooling layer performs an operation such as selecting the maximum value within its range, thus acting as a regularization technique to avoid overfitting.

Finally, convolutional and pooling layers are followed up by a FCL. The FCL size needs to be the same as the number of classes or outputs the network has to learn to identify [2], and its goal is to simply act as a classification layer which outputs the probability for each class.

Once the architecture of a CNN has been specified, its filters and weights are initialized to small random values. Next, given an image as an input, it is fed through the convolutional, pooling operations, and the FCL. The output probabilities of the network are then used to compute the total error, and finally gradient descent is used to update the filter and weight values with respect to their contribution to the total error. This process is repeated with other images in the dataset until satisfactory results are achieved.

4 Implementation

4.1 Data Preprocessing

All images in the dataset are of size 424×424 and the object of interest is always centered. In order to reduce the dimensionality of the images, during preprocessing images are cropped to 212×212, half their original size, images are also down sampled to half size again 106×106, discarding unnecessary information in each image which could impair the network. Down sampling can help the CNN learn which regions are related to each specific expression as well as improve performance when training [11]. Reducing the size of each image has the secondary quality of being more memory efficient which is beneficial for training and network size.

4.1.1 Data Augmentation

While there is a sufficient count of images to train on, it is possible to increase the performance of the network by augmenting the training data, increasing the apparent size of the dataset to the CNN, thereby reducing the chances of overfitting. Because the galaxies are rotationally invariant this dataset is an ideal candidate for applying data augmentation to. Therefore, a random transformation was performed online to different images across epochs. This transformation could be any combination of rotation by 90 degree increments or a flip on the horizontal or vertical axes, or no transformation at all (one in four chance that there is no rotation at all). This brings 16 possible configurations for each image. Trained over 50 epochs there is a good likelihood that the network will see every configuration of any given image.

4.2 CNN Architecture - VGG-16

VGG-16 is a 16 weight-layer CNN used for image recognition. Created by the visual geometry group at Oxford in 2014 VGG-16 got second place to Google's GoogLeNet in the Large Scale Visual Recognition Challenge (ILSVRC) 2014 [13]. With the trained model released to the public, it is possible to build on top of the weights that are already set by the initial training phase. The advantage behind this is that the model doesn't need to be trained from scratch, this is a form of transfer learning.

Other pre-trained CNN architectures exist such as GoogLeNet and AlexNet. GoogLeNet uses average pooling instead of FCL's [20] and beat VGG with an error rate of 6.67% over VGG's 6.8%. AlexNet won ILSVRC in 2012 and was novel for stacking multiple convolutional (conv) layers in a row before pooling their results [14].

Table 1 lays out the full VGG-16 architecture. VGG also made 11, 13, and 19 weight-layer variants which use different counts and sizes of conv layers. For the variant implemented here, all conv layers are of size 3×3 and have a stride of one. The second number after "conv3" in table 1 represents the number of image channels, which increase as the architecture gets deeper and the image input more filtered.

VGG is a good choice over GoogLeNet and AlexNet because the weights from the competition are available to the public for use and AlexNet and GoogLeNet are not as readily available. Furthermore, VGG's fully connected nature without any special layers and widgets like AlexNet or GoogLeNet make implementation and management much more simple.

While VGG-16 was trained on 1000 categories of images (classes), the Galaxy Zoo dataset has an expected output dimensionality of 37. Therefore, several changes had to be made to the network architecture to support the new input and output sizes. The last layer is set to 37 neurons. On the input layer of VGG-16, because the data preprocessing methods crop and down-sample from 424×424 pixels to 106×106 pixels the input layer needs to be one of size $3 \times 106 \times 106$ (three because the images have three color channels: red, green, blue). Another change made for this problem was changing the output layer activation function from soft-max to sigmoid. With this new network architecture the pre-trained weights from the ILSVRC competition released to the public by VGG were unusable, this meant training from

Input 224×224
Conv3-64 Conv3-64
maxpool
Conv3-128 Conv3-128
maxpool
Conv3-256 Conv3-256 Conv3-256
maxpool
Conv3-512 Conv3-512 Conv3-512
maxpool
Conv3-512 Conv3-512 Conv3-512
maxpool
FC 4096
FC 4096
FC 1000
soft-max output

Table 1: VGG-16 architecture [13].

scratch was necessary. Fortunately the hardware to do such training was available (see 4.5).

4.3 Dropout

Dropout is a method to prevent NN's from overfitting [17]. It does this by randomly deactivating some neurons in one –or many– layers. This makes the remaining active neurons compensate their weights twice as much in the back-propagation phase. Neurons are randomly activated and deactivated per update; here, each time a batch has been fully fed through the network. By activating and deactivating neurons in layers, dropout keeps the neurons of a network robust to the training data by never letting them fully fit to the training data. In [17] the authors recommend to keep dropout ratio between 0.5 and 1. In the implementation of VGG-16 the 50% dropout rate –which the original implementors chose– was unchanged. In this VGG-16 implementation dropout only applies to the final two FCL layers. This means that during any given batch only 2048 neurons are actually active in each layer during any given training step.

4.4 Hyperparameters

A learning rate of 10^{-6} was a adopted, this is much lower than the original VGG-16 learning rate, which was trained with a learning rate of 10^{-2} and was decreased by a factor of 10 when the validation accuracy stopped improving [13]. This decrease was because the dataset was much smaller, the ImageNet training dataset is 1.3 million images, a higher learning rate would be advantageous with that much data because you would want to take very large improvements in the beginning. Momentum of 0.9 was used. There was no learning rate decay. Early validation splits were very high, at 90% training and 10% validation (90/10) this split was producing models which were overfit as evidenced by poor performance as well as the training loss on the model becoming –sometimes significantly– less than the validation loss. Later, this was changed to a more traditional validation split of 80/20. The test set was provided entirely separately by Kaggle and is discussed in section 5.1. In each back propagation call after a step, the minimizing function was a mean square error of the expected versus actual output.

Dimensions of the images can be reduced dramatically and still be useful to the network. Two image sizes were tried, 106×106 and 69×69. The 106×106 image size is the center of the image cropped to 212×212 and downsampled. The 69×69 size was prompted from the winning submission to the Kaggle competition, the author crops at 207 and then downsamples 3x to 69×69 [7]. Different batch sizes were tried too: 16, 32. These batch sizes determine how fast the weights of the network get updated. The smaller the batches get to one the closer the network gets to using simple gradient descent methods. Because the dataset and the number of parameters in the network are both large, adjusting weights with an extreme frequency could lead to getting stuck in minima or over-adjusting and missing the target function.

Another change from the presented VGG-16 implementation was the output layer activation function. Sigmoid activation function was chosen over soft-max, this was an important change because the soft-max function has difficulty predicting hard zeros and ones [7]. Throughout the dataset there are many hard zero values which represent the galaxy not having that attribute at all.

4.5 Software and Hardware

For the software implementations there are two readily available and widely used machine learning frameworks Keras and TensorFlow. Keras is a high-level machine learning library written in Python. It can be used with a range of backend drivers which makes it very portable. TensorFlow is a backend driver used with Keras which allows access to low level hardware for training.

A NVidia GeForce 970 graphics processing unit (GPU) was available, it was used for training each of the models described in results section. GPU's allow for embarrassingly parallel computations and NN's can be placed in that domain of being embarrassingly parallel. Using a GPU decreased training time by an order of magnitude over traditional CPU's.

	69×69	106×106
Batches of 16	0.11538 (95)	0.11062 (78)
Batches of 32	0.11476 (94)	0.11288 (88)

Table 2: Spread of results for various input and batch sizes. In parentheses after each score denote the place it would have earned in the Kaggle competition.

5 Results

Each model was trained for at most fifty epochs on the full training dataset. Early runs were trained with a 90/10 training-validation split, there didn't seem to be much room for improvement based on the separation between the validation and training loss. Thus, an 80/20 split was settled on, results were much better. Early stopping would be invoked if validation loss did not improve for five epochs. To make sure the results could be equally quantified the dataset was shuffled with the same random seed each time.

5.1 Benchmarks

The Galaxy Zoo Kaggle competition has a test image set of 79 975 images. After training the model the network uses these images to produce predictions. These predictions are saved as a CSV file and uploaded to Kaggle. Kaggle then scores the predictions using the root mean square error (RMSE) between the model predictions and what the actual values are (these actual values for the test set are kept secret), the lower scores are better. There are certain basic thresholds which can be used to determine if the model behaves as a random classifier or worse. The first is an All Zeros benchmark, this is equivalent to a network that always outputs zeros for all attributes. Similarly, there is an All Ones benchmark; these two benchmarks represent random classifications which, while not entirely trivial to pass, require some rational architectural choices. The final benchmark is harder, a Central Pixel benchmark, which are the predictions if the outputs were simply the central pixel RGB value in each training image. Very early implementations of the VGG-16 model could pass the All Zero benchmark, later–with the right hyper-parameters– the model could pass the All Ones and Central Pixel benchmarks. Results through the rest of this paper are based on the RMSE score Kaggle given for a particular model.

5.2 Performance

Before immediately attempting data-augmentation methods, a few hyperparameter configurations were run to get a better idea of the best input size and the best batch size for moving forward (figure 2).

Results where image size was 69×69 and where the batch size was 32 or 16 the loss graph and validation loss exhibited traits of overfitting (figure 5.1). The results from Kaggle may, in fact, back this up, as the score gets better the larger the batch size. Although, two points do not necessarily make a trend, it could also be that larger batch sizes may be necessary when the images (input sizes) are smaller.

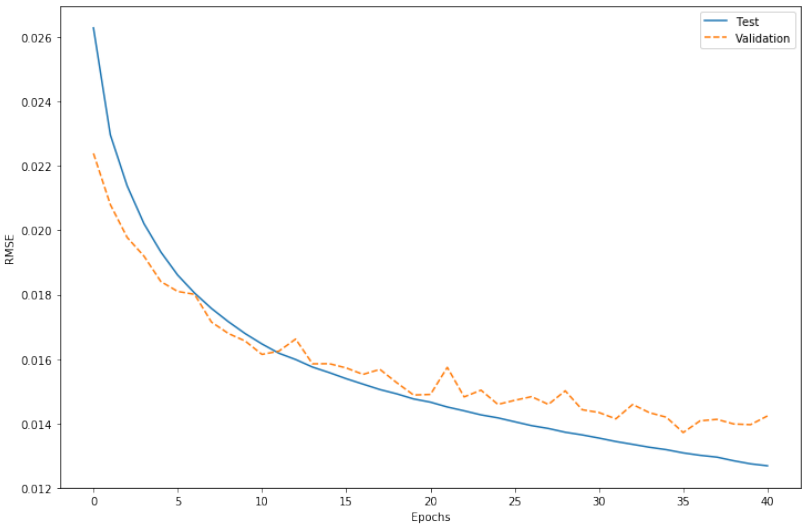

Figure 5.1: The training loss line being lower than the validation loss line is characteristic of overfitting. Stopped before 50 epochs.

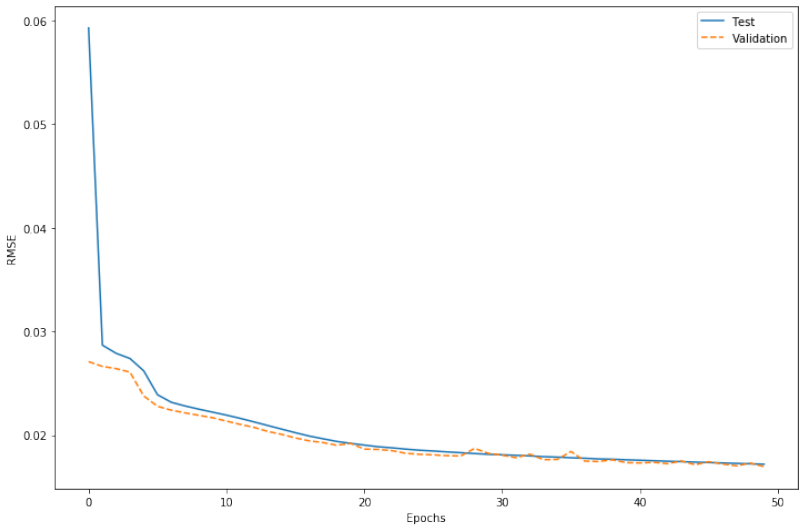

Figure 5.2: A run with a 90/10 training-validation split, the score from this run was admissible but ultimately beat by 80/20 splits. The Kaggle score from this run was 0.13070.

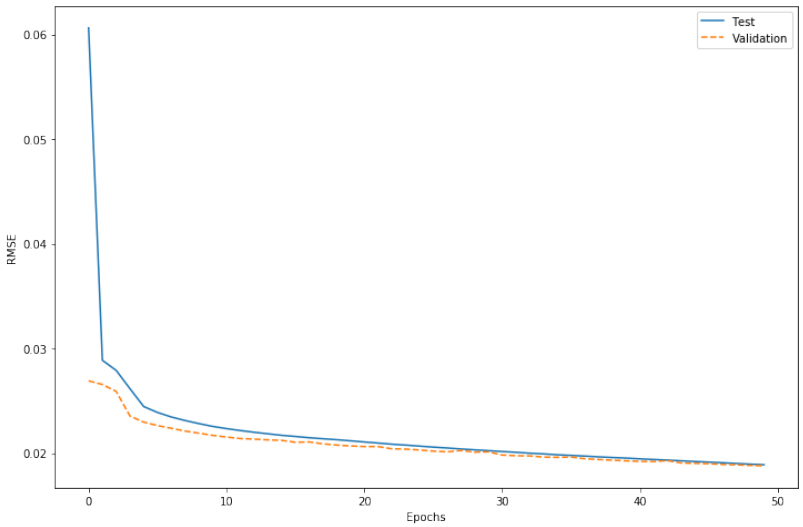

Figure 5.3: A run with 80/20 training-validation split. Notice the smoother curve versus a run with the same parameters but a 90/10 split.

Results for the larger images are more in line with expectations. Interestingly, the smaller batches of 16 worked better than 32. This may be surprising compared to the original VGG-16 implementation where input sizes were 224×224 and their batch was 256. The low learning rate is potentially the parameter solving this. Again, the ImageNet dataset is a little more than an order of magnitude larger than the Galaxy Zoo one, in this aspect the chosen architecture may be overkill and a smaller VGG variant should have been tried out first.

The best run was a simple improvement from an early run where the data was split 90/10 (figure 5.2). This was a rather high split, while it could be trained further the jitter of the validation loss line is cause for concern. It was eventually settled on using the same split which VGG did: 80/20. The results of this run produce an error curve smoother than the one a 90/20 split produced and one which, if desired, could continue to be trained (figure 5.3).

5.3 HSV Input format

In digital imagery pixels are stored as combinations of three values red, green, and blue (RGB). However, there are many more ways to represent color; in printing for example: cyan, magenta, yellow, black (CMYK) are optimal values because of how light reflects on printed surfaces. Hue, saturation, value (HSV) is a different color space where the color is stored in hue, color intensity is stored in saturation, and the brightness is stored in a value. There were hypothesis if converting the galaxy images to this color format might enhance responses in the network, stars being bright, the star pixels will emit a high V value and thus create stronger weights.

The results from this color space conversion were not terrific. With a batch of 32 with images the image size of 106×106 got a score of 0.13778 worse than any RGB result. The score was worse due to two issues over which there was much rumination. While V may have been regularly high for the features the model is trying to detect, H may have been too varied for the network to converge on. For S, the color intensity of images was not very high, and so most colors were close to zero on the saturation scale. The combination of a rapidly fluctuating variable H, a regularly small value of S made V comparatively worthless to the network. This score would have earned 178th in the Kaggle competition.

5.4 Data Augmentation

After examining the results on 16 and 32 batches it was decided to move forward with applying data augmentation to image sizes of 106×106. Augmenting with the transformations discussed in section 4.1.1 increases the size of the training dataset by a great amount. In an 80/20 split there will be 49 262

	106×106
Batches of 16	0.11331
Batches of 32	0.11673

Table 3: Spread of results for various batch sizes with data augmentation. Augmentation was a random combination of reflection on the horizontal or vertical axis and being rotated by 90 degree increments.

All results, sorted best to worst	
image size*, *batch size	**Score**
106×106, 16	0.11062
106×106, 32	0.11288
106×106, 16, augmented	0.11331
69×69, 32	0.11476
69×69, 16	0.11538
106×106, 32, augmented	0.11673
106×106, 32, 90/10 split	0.13070
106×106, 32, HSV format	0.13778

Table 4: All results sorted by Kaggle score. Runs are using 80/20 split unless specified otherwise.

training images, when this is augmented there will effectively be 788 198 images. The validation dataset is not augmented as it is needed as unaltered images are best used for a baseline.

This reinforces a forming hypothesis that smaller batch sizes for the 106×106 image sizes are more optimal. These scores would have earned 91st and 97th respectively. There may be too much augmentation going on and not enough hyperparameter adjustment. There are almost a three quarters of a million images with these transformations, it might have been best to perform some learning rate studies; as is, the network might not have taken the large strides it needed to attain better score than the un-augmented datasets.

6 Discussion

From the get-go, there were a lot of models and solutions for this problem available online to trawl over and gain knowledge from. In that sense shoulders of giants were stood upon, and what tall giants they were. To summarize, a sixteen layer CNN was implemented in Keras with Tensorflow and was trained on a dataset of tens of thousands of images. These images were cropped and down-sampled to reduce dimensionality and make them a manageable size for the network. From those initial scores a series of other methods were tried and explored from data-augmentation to color space shifting. In the end the initial trials were the most successful, a full table of the results is available in table 4.

6.1 Further Work

A top 100 score on Kaggle is an accomplishment but there is plenty of improvement to be done, the scores are in the bottom quartile after all. Hyperparameter studies beyond batch sizes and training splits were not performed, it is likely that hyperparameters such as learning rate and learning rate decay could have helped improve the network past current thresholds. A few props were taken from the winning Kaggle submission, what else could be garnered from their solution? A smaller network for one, feeding images of 69×69 through such a large network might have been the folly of those runs which ultimately led to overfitting, there were simply too many network parameters and not enough input parameters to fill them. Follies with data-augmentation may not have been negligible either, while the images were certainly rotationally invariant, too many translations may have been performed and made the apparent dataset too large for the network to settle on. Tweaking the learning rate and number of epochs could have worked well, though in that scenario –with resources limited as they were– there might not have been time to sufficiently explore that line of experimentation which would have involved running 8-12 hour jobs. Ultimately, many runs with varying methods producing varying results –instead of trying to

laser in on a single great score by perpetually improving a single good result– may have given a breadth but not depth of material to present.

With the network as it is, could there be other astronomical problems to tackle? In the coming years the James Webb space telescope will soon be launched (or hopefully, it has already been delayed multiple times this decade). An infrared telescope, it will be able to see objects in the Universe which conventional visible light telescopes cannot. This could open up whole new bodies previously obscured by dust and nebulae. Notable here, it will be able to detect the earliest galaxies in the Universe.

6.2 Contributions

Prior work has demonstrated the effectiveness of machine learning techniques applied to astronomical datasets. This is further demonstrated here with a sufficiently large CNN which uses a very low learning rate. Present results indicate that data-augmentation may not work well with such a low learning rate and that other hyperparameters –such as learning-rate decay and batch sizes– may be open to adjustment. The present research is therefore intended to make contributions towards the study of applications of machine learning on large, image based, astronomical datasets.

6.3 Acknowledgments

Gustaf Borgström, the teaching assistant assigned to us, for keeping us on schedule and advising on the structure of this paper.

References

[1] N. M. Ball, R. J. Brunner, A. D. Myers, and D. Tcheng. Robust Machine Learning Applied to Astronomical Data Sets. I. Star-Galaxy Classification of the Sloan Digital Sky Survey DR3 Using Decision Trees. *Astrophysical Journal*, 650:497–509, October 2006. doi: 10.1086/507440.

[2] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification, 2011.

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL `http://doi.acm.org/10.1145/1390156.1390177`.

[4] T. N. Wiesel D. H. Hubel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160, 2015. ISSN 0022-3751.

[5] Dansbecker. Rectified linear units (relu) in deep learning. `https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning/code`.

[6] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL `http://dx.doi.org/10.1561/2000000039`.

[7] Sander Dieleman. My solution for the galaxy zoo challenge. `https://benanne.github.io/2014/04/05/galaxy-zoo.html`.

[8] Robert J Brunner Edward J. Kim. Star-galaxy classification using deep convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 464:4463–4475, 02 2017.

[9] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007. ISBN 0470035617.

[10] Stanislav Fort. Towards understanding feedback from supermassive black holes using convolutional neural networks. *Workshop on Deep Learning for Physical Sciences (DLPS 2017)*, 2017. URL `https://arxiv.org/pdf/1712.00523.pdf`.

[11] Mehdi Ghayoumi. A quick review of deep learning in facial expression. *Journal of Communication and Computer*, 14:34–38, 2017.

[12] Tony Hey, Stewart Tansley, and Kristin Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, October 2009. ISBN 978-0-9825442-0-4.

[13] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[15] Yann Lecun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Contour and Grouping in Computer Vision*. Springer, 1999.

[16] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL `http://dl.acm.org/citation.cfm?id=3104322.3104425`.

[17] Alex Krizhevsky Ilya Sutskever Ruslan Salakhutdinov Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[18] Beth Willman Ross Fadely, David W. Hogg. Star–galaxy classification in multi-band optical imaging. *The Astrophysical Journal*, 760(1):15, November 2012.

[19] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *the official journal of the International Neural Network Society*, 61, 2015. ISSN 1879-2782.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

[21] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf`.

[22] Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, and Daniel Thomas. Galaxy zoo 2: detailed morphological classifications for 304 122 galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 435(4):2835–2860, 2013. doi: 10.1093/mnras/stt1458. URL `http://dx.doi.org/10.1093/mnras/stt1458`.

[23] Zooniverse. The science behind the site. `https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/about/research`.