Neural Networks: Assignment 2

Spring semester 2019

1 Task 1: Proof of Understanding

To begin this assignment we present some preliminary explorations of the following neural network architectures: convolutional neural networks, recurrent neural neworks, and autoencoders.

1.1 Convolutional Neural Networks

Convolutational neural networks have more than one "hidden" layer and these layers have one of two purposes: convolution and pooling. By convolving the input data with a kernel (which is optimized like any other set of weights) there is a dramatic reduction in the number of network parameters that need to be optimized. Additionally, convolution makes the network equivariant, i.e. a change in the input leads to the exact same change in the output. Pooling the outputs from neighboring nodes prevents overfitting; in the case of analyzing a particular set of images it is more important to discern holistic features rather than a pixel-by-pixel analysis of how certain features relate to others. A helpful way of thinking about convolution and pooling is that we are, in a sense, endowing infinitely strong priors on certain parameters within the network (e.g. neighboring nodes will have certain correlations and the weight associated with an unactivated edge will automatically be zero) such that we can avoid the computational expense of optimizing them.

We explored the architecture of a fairly simple convolutional neural net developed by TensorFlow (see `https://www.tensorflow.org/alpha/tutorials/images/intro_to_cnns`) and its ability to classify the familiar MNIST dataset. Figure 1 shows the arrangement of convolutional, pooling, and dense layers. The provided activation functions were the rectified linear unit for every non-pooling layer except the second dense layer which used the 'softmax' function. (A dense layer applies an element-wise evaluation of the expression `output = activation((dot(input, kernel) + bias)`.) We found using the hyperbolic tangent as an activation function rather than the rectified linear unit (which are described in more detail in § 2.3.1) led to a drastically less accurate network. Additionally, the version of `keras` we downloaded had a bug associated with 'softmax' so we used a similar function called 'softplus'. With a classification accuracy of 98.74 % this simple CNN outperformed any of the classifiers we developed during assignment 1.

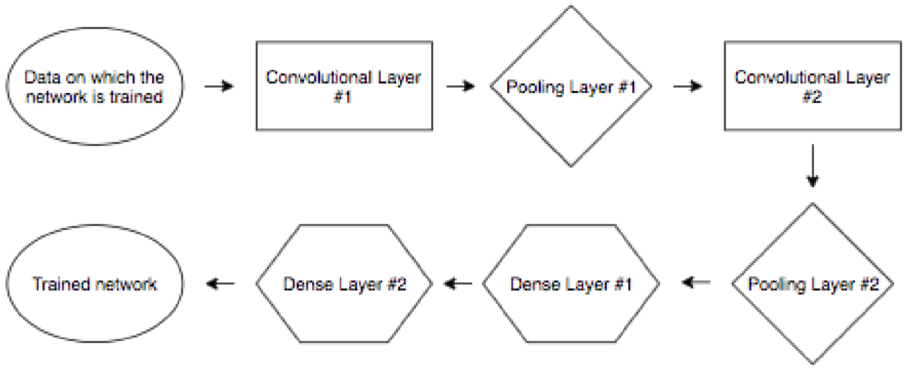


Figure 1: Convolutional neural network layers.

1.2 Recurrent Neural Networks

Recurrent neural networks (RNN) differ from basic feedforward neural networks as layer output is both provided as input to the next layer, as well as given back as input into the previous layer. Whereas hidden layers in a basic feedforward neural network just provide output to the next layer, RNN's use their internal 'memory' state to learn sequences of inputs. To illustrate this process, Figure 2 shows a simplified schema of a recurrent neural network.

Since the networks' internal memory enables sequence processing, RNN's are most often used for prediction in time series. Text sequence prediction, weather forecasting and similar time series analysis problems are often solved well using an RNN.

In order to gain understanding of RNNs we implemented a word generation network as proposed by Siraj Raval (see `https://github.com/llSourcell/recurrent_neural_network`), which takes a plain textfile as input to train the network in generating of words and sentences in style of the input text using just one hidden layer. We experimented with the amount of nodes in the hidden layer and measured performance using the negative log likelihood loss function with regards to a textfile of the book 'Pride and Prejudice' written by Jane Austen. We found that using a higher amount of nodes in the hidden layer resulted in slower convergence of the loss. The default amount of nodes was 100, which resulted in a loss converging to about 45 after 20,000 iterations, whereas using 200 nodes in the hidden layers seemed to converge in a loss of 55 after 30.000 iterations. Thus, as for this small experiment can be concluded that too many nodes in the hidden layer has a negative impact on the performance.

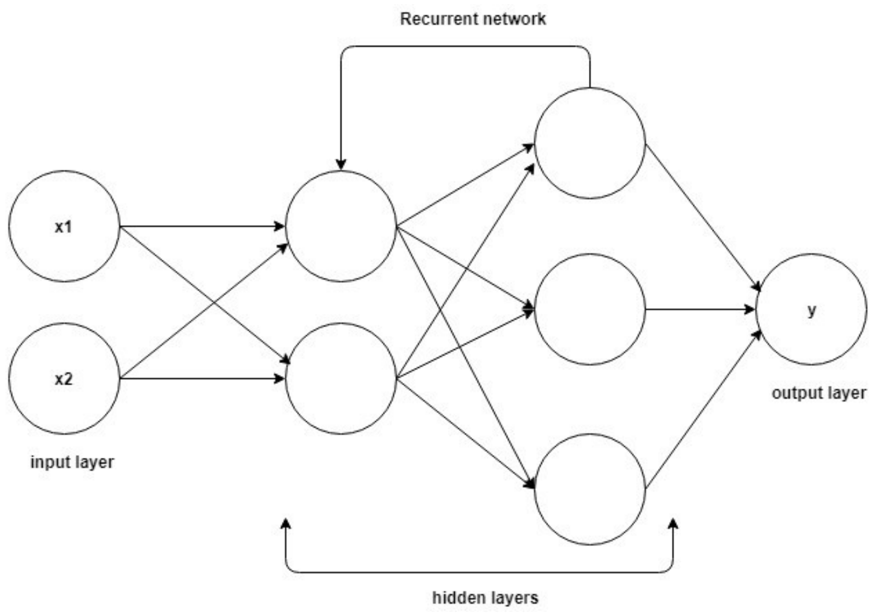


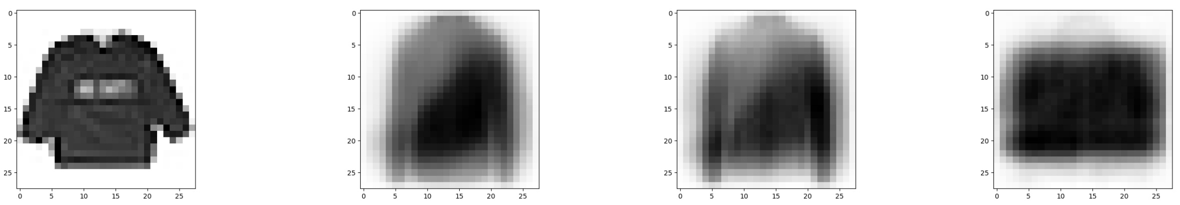
Figure 2: Recurrent neural network schema

1.3 Autoencoders

An autoencoder is a type of artificial neural network that is used to learn efficient data encodings by encoding input data into a simplified representation of the data in an unsupervised manner. Also, a decoding method is learned to efficiently recreate a decoded representation of the encoded simple middle layer.

In order to give a proof of understanding, we implemented an existing autoencoder on the MNIST fashion dataset as proposed by Soumya Ghosh (see `https://medium.com/@connectwithghosh`). The MNIST fashion datset consists of 70,000 images of zalando garments represented as a 28×28 matrix containing grayscale values, where every image is associated with a label that corresponds to one of the ten classes which representing different pieces of clothing, such as 'dress' or 'shoe'. In the default implementation as proposed by Ghosh, two hidden layers are used and the images are encoded to and decoded from a binary vector of size 32, which is small enough to enable fast computation on an ordinary 1.6 GHz Macbook Air with 4 gigabytes of RAM.

We were interested in the effect of the encoded layer size in terms of loss and graphical accuracy of decoded images. We experimented with the size of the encoded representations as we trained the network using an encoded representation vector of size 16, 32 and 64. In our experiments, we trained the networks for 100 epochs using 100 images per batch. Using a vector of size 16, the loss converged to $\sim$ 190,000 with a representation vector of size 32, the loss converged to $\sim$ 157,000 and with the vector of size 64 the loss converged to $\sim$ 130,000. This would mean that a recurrent neural net with a bigger representation vector size performs best. However, when looking at the decoded images provided in Figure 3, the performance seems to be quite ambiguous as in case of vector sizes 16 and 32 the shape of a sweater can be clearly recognized, whereas in case of vector size 64 there is no recognition of a sweater possible at all without a lot of fantasy, and looks more like a bag.



(a) Original image (b) Decoded 16 bit (c) Decoded 32 bit (d) Decoded 64 bit

Figure 3: Original image compared with decoded images.

2 Task 2: Deep Learning Challenge

2.1 Introduction

Our task for the second part of this assignment was to find an original problem/dataset and apply deep learning to it. In order to do so, we implemented a neural network with convolutional and dense layers to classify galaxies by shape.

Systematically classifying galaxies based on their morphology has been a key goal in the field of astronomy for more than a century. One popular scheme was proposed by Hubble and Sandage in the 1920s; it is repeated as Figure 4. Initially this process (erroneously) attempted to show how galaxies evolve from elliptical to spiral galaxies with an array of features (e.g., a bar bisecting the galactic center in the SBx galaxies or the degree to which the arms are wrapped around the center). Hubble also found that there were a large number of galaxies that had no discerning characteristics and classifying them would consequently be a dubious concept. These anomalous galaxies often arise after mergers. (As this is a report on using a neural network to classify images this would be analogous to having a 3 and 7 collide and then including the result in the MNIST data set.) Additionally, these classifications only depend on visible characteristics of the galaxy and that omits a significant portion of information about the galaxy, e.g. the mass of its dark matter halo. Nonetheless, it is instructive to be able to classify galaxies as different phenomena present in these images often emerge from interesting things happening within the galaxy itself that are more difficult to infer.

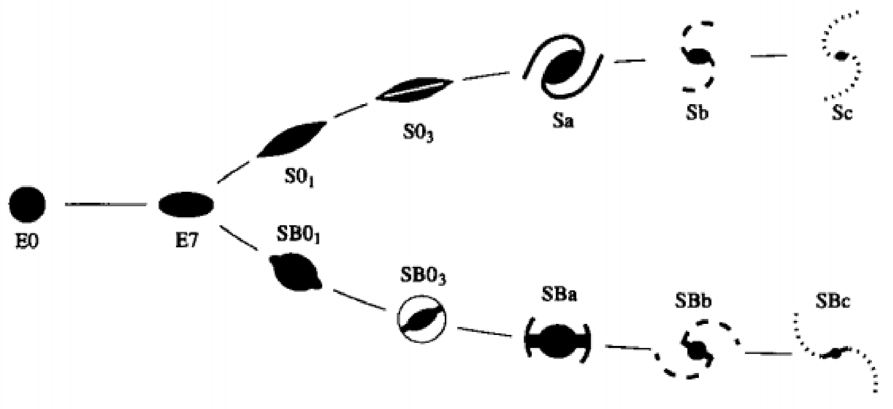


Figure 4: The Hubble "tuning fork", borrowed from Marijn Franx's lecture notes.

Nearly a century after Hubble's work on galaxy classification the Sloan Digital Sky Survey (SDSS) took pictures of thousands of galaxies distributed throughout the cosmos. Given the number of galaxies there was not an obvious solution about how to go about classifying them. One proposed method, called the Galaxy Zoo project (see `http://zoo1.galaxyzoo.org/Project.aspx` for more details), would be to crowdsource the task. Humans are exceptionally good at pattern recognition (which is why, for example, there are a myriad of constellations found from the seemingly random distribution of stars in the night sky) so it stood to reason that once explained the classification rules regular people would be just as good if not better than computers at carrying out this classification task.

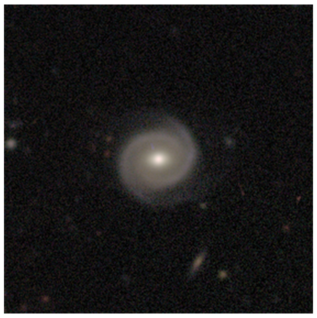
Our goal is not to see if computers can exceed the performance of humans in the task of classifying galaxies; it is not as if a galaxy is endowed with an exact classification scheme in the same way that other images waiting to be classified are. Instead, we propose employing a Convolutional Neural Network (CNN) to see how well a computer can recreate the predictions made by thousands of people critically looking at galaxies, many of whom were doing so for the first time. If the network is successful in achieving the same classifications as the Galaxy Zoo volunteers it would be reasonable to proceed with applying this network to galaxy images found in future astronomical surveys.

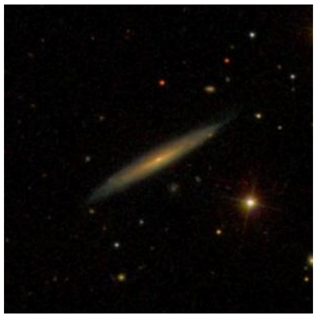
2.2 Methods

2.2.1 Dataset

The Galaxy Zoo classification dataset has been compiled and made open to the public (see *The Galaxy Challenge*) on Kaggle; The Galaxy Challenge invited programmers to create a network that could best reproduce the efforts of the human volunteers. The challenge ran from December 2013 to April 2014 where the best solutions were awarded up to $10,000 USD.

The dataset consists of a total of 141,553, 424×424 `.jpg` galaxy images. These images have been preëmptively split up into a training set (61,578 galaxies) and test set (79,975 galaxies). Some examples of galaxy images are provided in Figure 5.


(a) Spiral


(b) Disk

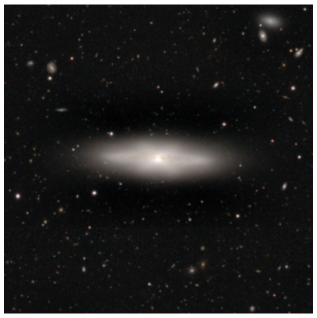

(c) X-shaped [1]

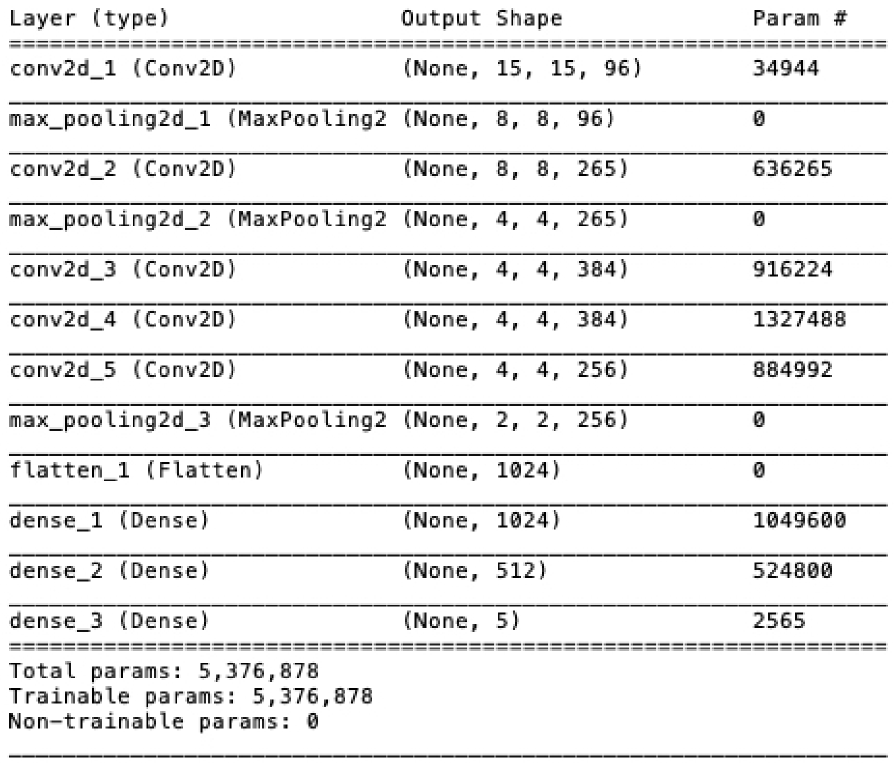
Figure 5: Examples of Galaxy Zoo images.

The volunteers were given the opportunity to classify each galaxy as one of 37 possible subclasses. A recent paper published in the Monthly Notices of the Royal Astronomical Society (see `https://arxiv.org/pdf/1807.10406.pdf` for more details) created their own classification scheme of the Galaxy Zoo data where there are four possible elliptical galaxy classifications and one spiral classification. It would be an interesting exercise to see if our neural network could achieve a similar performance but this is perhaps beyond the scope of this assignment.

The actual classes are only available for the training images from the Galaxy Challenge, so we decided to leave the test set out of our experiments in order to be able to test the accuracy of our model. We created a new distribution of training and test set where 70 % of the data is used as training data and 30 % as test data. Solutions in the image set contain a probability distribution for all images, which shows the certainty of belonging to each of the classes. We only consider the class with the highest probability to be the descriptive class for the corresponding image; in other words, whichever classification was chosen by a plurality of the volunteers who looked at that particular galaxy will be the classification with which we work. After this processing scheme, only five distinct classes remain in the training data.

2.2.2 Convolutional Neural Network Structure

In order to use the given set of images as an input for the network we initially transformed the `.jpg` images to $60 \times 60 \times 3$ `numpy` arrays where the three array slices contain the RGB values. We found that this input format was prohibitively expensive; additionally, the physical implication of keeping the colorized versions of these images is not particularly profound as the volunteers were instructed to be primarily concerned with morphology rather than color. Thus, we found that using the grayscale capabilities of the `cv2` Python library reduced the number of parameters that required optimization within the network while simultaneously keeping the most important features of each image.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 15, 15, 96)        34944
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 96)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 265)         636265
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 265)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 384)         916224
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 384)         1327488
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 256)         884992
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 2565
=================================================================
Total params: 5,376,878
Trainable params: 5,376,878
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: A schematic of AlexNet; the neural net we have developed has a very similar structure.

The structure of the CNN that we implemented is similar to the CNN as used in the proof of understanding section on the MNIST dataset, existing of an input layer, several convolutional and pooling layers, two dense layers and an output layer. Our proposed structure is based on the AlexNet CNN structure, which is an often used structure for image classification in which the first, second and last convolutional layers are followed by a maxpooling layer. A schematic overview of the basic network structure is provided in Figure 6.

The initial layer of the network takes the (normalized and 2D) grayscale image arrays as input and applies a convolution with a kernel of length 11. After the convolution a pooling layer is applied with 3 pooling windows. With the exception of appropriately changing the dimensionality of the output space in the convolution step, the third and fourth layers of each network simply repeats the convolution and pooling of the first two layers. Once these initial layers have been implemented, the next portion of the network's architecture is a variable number of convolution layers after which a single pooling layer is applied. We will discuss the motivation for the inclusion of this network feature in § 2.3. This concludes the portion of the network with adjustable parameters, as all that remains are a flattening layer (changes the format of the previous layer's outputs) and three dense layers with fixed activation functions.

2.3 Tunable parameters

While the AlexNet structure has had success in image classification we thought it would be fruitful to tune two network attributes: the choice of activation function within the convolution layers as well as the number of convolution layers applied after the first two convolution/pooling pairs. Given the relatively peculiar nature of our dataset (which will be elaborated upon more in the conclusions section) it will behoove us to consider several options.

2.3.1 Activation functions

As discussed in the previous assignment activation functions play a critical role in neural networks. The inputs for a particular node $\{\mathscr{I}\}$ are fed into a function f such that the output of the node succinctly describes $f(\{\mathscr{I}\})$. This was done in assignment 1 as digits in the MNIST dataset were evaluated based on their distance to a discriminating hyperplane and consequently determined to be (or not to be) included in the associated class.

According to the activation functions Wikipedia page there are several desirable attributes of activation functions that ensure stability and efficiency in network training: nonlinearity, monotonicity, and approximating $f(x) \approx x$ for small values of x. Our choices of activation functions are shown in Figure 7; the most successful was either the rectified linear unit or hyperbolic tangent:

$$f(x) = \begin{cases} 0 & x < 0, \\ x & x \geq 0. \end{cases} \tag{1}$$

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2}$$

2.4 Results

We now have twelve networks to consider (four choices of convolution layer activation function and either 0, 1, or 3 extra convolutional layers) and there are two main metrics with which we would like to evaluate them: accuracy and computational expense. The first is done in the standard way (train the network using the training set and see to what extent the classifications within the testing set can be replicated) and the second is done using the runtime. Each of the networks used a single GPU (see the `bashrc` file for more details) so the time it took to train the network is directly correlated with the complexity.

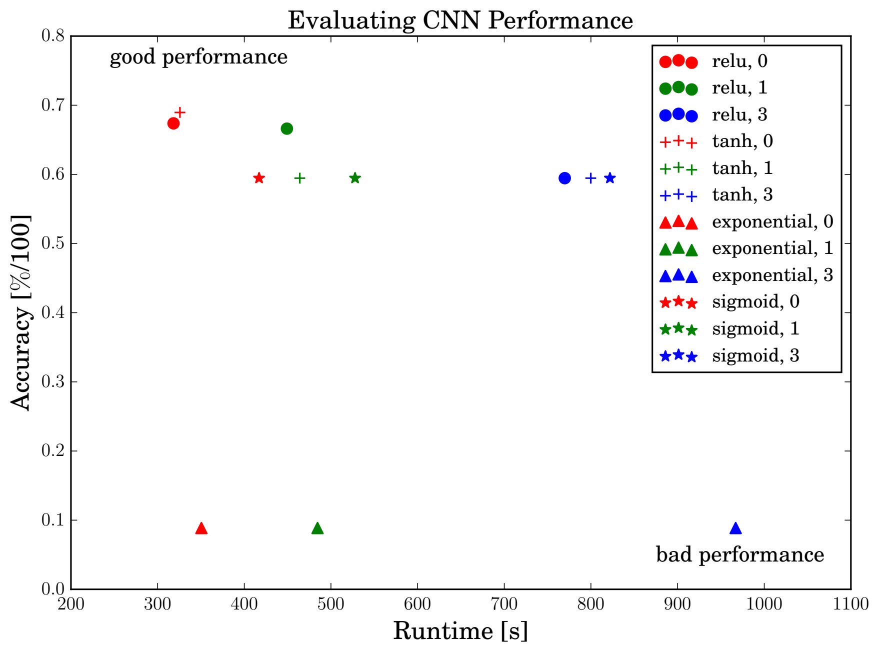


Figure 7: The accuracy and runtime associated with each proposed network configuration when applied to the testing set we used; see § 2.2.1 for more details.

While it is expected that networks with additional convolutional layers took longer to train it is perhaps unexpected that the more complex networks performed worse than their comparatively simple counterparts. Introducing no additional convolutional/pooling layers (beyond the first four) and choosing either the 'tanh' or 'relu' activation functions yields the best network, namely a network that can achieve 69.0% accuracy on the testing set with a training time of a little more than five minutes. This accuracy is comparable to that of the top-1 scores (a metric similar to our evaluation scheme in which the most probable class proffered by the network matches the testing data) achieved by ENet, GoogLeNet, and ResNet-18. The network with convolutional layers applying the 'exponential' activation function had an undefined loss so it is unwise to trust the outputs of this network.

2.5 Conclusions

It may come as a surprise that our best performing network only achieved a $\sim 70\%$ accuracy, but this deserves further explanation. Our network was not trying to infer the well-defined class of an image like a cat or the number 7; it was trying to recreate the aggregate thought process of volunteers whose task was to analyze images that are notoriously difficult to classify. While beyond the scope of this assignment it would be interesting to know which galaxy features the network picked up on and how they would compare to those of a volunteer. For example, the network may have placed higher emphasis on relational attributes (e.g., the symmetry of possible spiral arms or the ratio of galactic bulge-to-disk brightness) while the volunteers focused on the most apparent feature of each galaxy. Additionally, the Galaxy Zoo produced a probability distribution associated with potential classes rather than a single choice; future work could be devoted to developing a network that provides outputs with similar structure.

One could argue the most important attribute of our network (and the one that would make it attractive to researchers interested in applying a galaxy classification scheme) is its relative simplicity. With publicly available machine learning libraries (namely, `tensorflow` and `keras`) and a GPU we were able to qualitatively reproduce the efforts of a years-long project comprised of thousands of volunteers in a matter of minutes. It is worth noting, however, that amateur astronomers have always been an integral role in advancing the field and as a result their efforts are tremendously appreciated. However, given the proliferation of neural networks (and GPU technological advances) it stands to reason that efforts similar to ours will supplant crowdsourcing efforts such as the Galaxy Zoo project in the future.