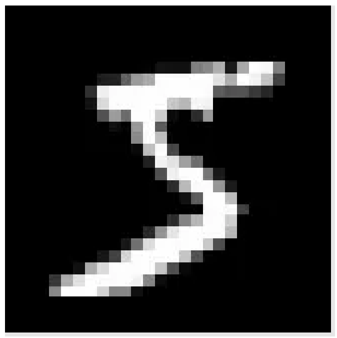
COMP 3301 Project Report

Introduction

Our project combines machine learning methods with image processing techniques to solve the problem of handwritten digit recognition. To motivate this topic, suppose that you would like to deposit a handwritten cheque into an ATM machine. The machine must invoke some algorithm in order to classify the correct amount of money to be deposited into your account. Such an algorithm involves an agent classifying handwritten digits. We would like our algorithm to be as highly accurate as possible so that the probability of the agent misclassifying the digits is minimized. In our example, a misclassification of the digits could lead to an incorrect amount of funds being deposited into the bank account. Other motivations include applications such as postal address recognition, writing to text conversion, etc.

To solve the problem specified, we trained an agent on a data set of 10000 grayscale images of the digits zero through nine, each image 28 by 28 in dimension (see figure). The data set which our model was trained on was far from diverse and, consequently, our model was not very adaptive. To handle this issue, we developed a program to pre-process images before feeding them to our model. The program applies image processing techniques learned in class such as thresholding, edge detection, filtering, addition, etc. The purpose of the program is to process the image so that it would correlate with the images in the data set that our model trained on and thus our model could accurately classify the image. Part of this process includes resizing input images down to 28 by 28.



Sample image of 5 from training set

We conducted two experiments to test the strength of our model and methods. In our first experiment, we wrote each of the digits zero through nine on a clean chalkboard and took pictures of each one to test on. The pictures taken were approximately 3000 by 3000 in dimensions and were coloured. We found that our methods were very successful on such images and our model was able to classify each picture with high accuracy. Our second experiment involved testing our model and methods on the digits zero through nine which we drew on a dirty chalkboard. In this case, our methods and model underperformed in contrast with our first experiment. The dirty chalkboard had a large impact on our experiment as it introduced an abundance of noise during the processing phase. Therefore, our model could not classify such images as accurately as in our first experiment due to the fact that the processed images did not resemble the images in the training set. The experiment was not a complete failure, however, because despite our comparatively less effective processing, our model could still classify the images with some degree of accuracy.

Neural Network Model

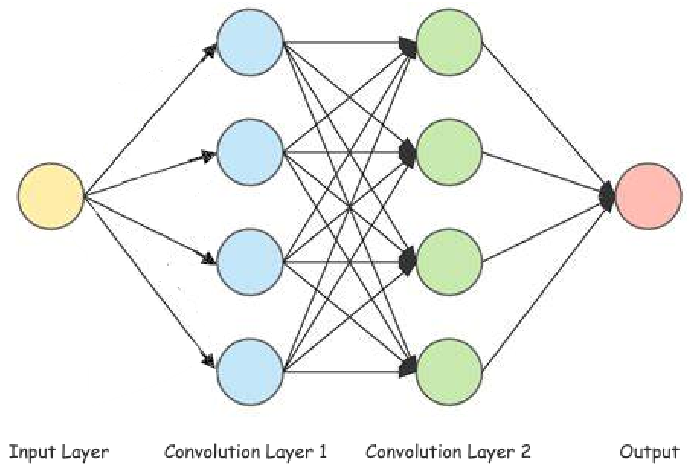
Our project uses a neural network-based machine learning approach to classify images. This structure is inspired by the human brain, imitating the way biological neurons send signals. Extensive training data is required for this model to enhance the accuracy and yield valid results. For our purposes, we used the *TensorFlow* python library.

In particular, our project makes use of a *convolution neural network* or *CNN* as this is an algorithm designed to recognize patterns in images. To begin, we break down the parts of a neural network into its fundamental components[3].

1. A *classifier* is a function in machine learning that categorizes input data into specific categories or classes based on certain criteria or learned patterns.
2. A *tensor* is an n-dimensional matrix. For our purposes, tensors will be familiar 2-dimensional matrices.
3. A *neuron* behaves as a function providing a unique output for a given input. This is consistent with our intuition with the biological analogy.
4. A *layer* refers to a group of neurons performing the same operation.
5. The *kernel weight* is the value of the kernel entries. This is tuned during the training phase and allows the classifier to adapt to the dataset provided.

Starting with the input layer. This layer consists of one or more input images that we are looking to identify. In principle, this could include multiple viewing angles or separate RGB channels. For our purposes, the input layer consisted of a single grayscale image of a digit that we feed into the CNN. From here, the CNN uses several *convolution layers*. These are special layers that are designed for image classification. Each neuron in this layer has a unique, corresponding kernel. This kernel performs elementwise dot product on each pixel of the input image and produces an output.

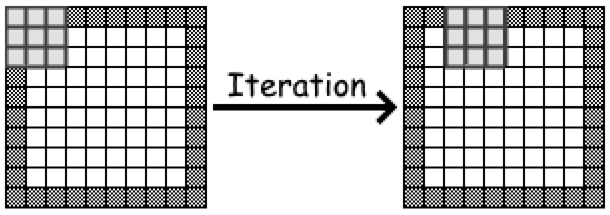
The first neuron in the subsequent convolution layer repeats this process with every output from the previous layer, by performing elementwise dot product on each of these outputs using its own unique kernel. The results of all these calculations are then summed up to produce this neuron’s output. This procedure is repeated for all the neurons in this layer, allowing these outputs to serve as inputs for the next layer and so on. At a much higher level, each layer of the network processes the image in stages, gradually extracting more complex features.



Convolution Neural Network[1]

Modern CNNs often use a vast multitude of convolution layers to achieve the best results at the cost of computation time[3]. In addition, the following global parameters called *hyperparameters* can be tweaked:

1. *Kernel size.* In general, a smaller kernel size creates a closer attention to detail in exchange for increased computation time. However, there are some instances where "the big picture" or larger details is what we want to identify. In this case, we would consider using a large kernel size. For the purposes of our project, we are interested in very small distinctions between digits and are not concerned with computation time. For this reason, we made use of a small, 3x3 kernel.
2. *Stride*. Stride indicates the rate that a kernel scans an input image, i.e., how many pixels to traverse after an iteration of the kernel operation. This parameter has a similar effect to the kernel size. That is, a smaller stride results in greater attention to detail while a large stride increases computation time with more of a focus on inputs as a whole. By default, stride is 1 and this is what it remained for our purposes.
3. *Padding*. Padding images is often necessary when kernel size exceeds the edges of an input image, causing loss of data at the borders. We made use of the zero padding technique for simplicity.



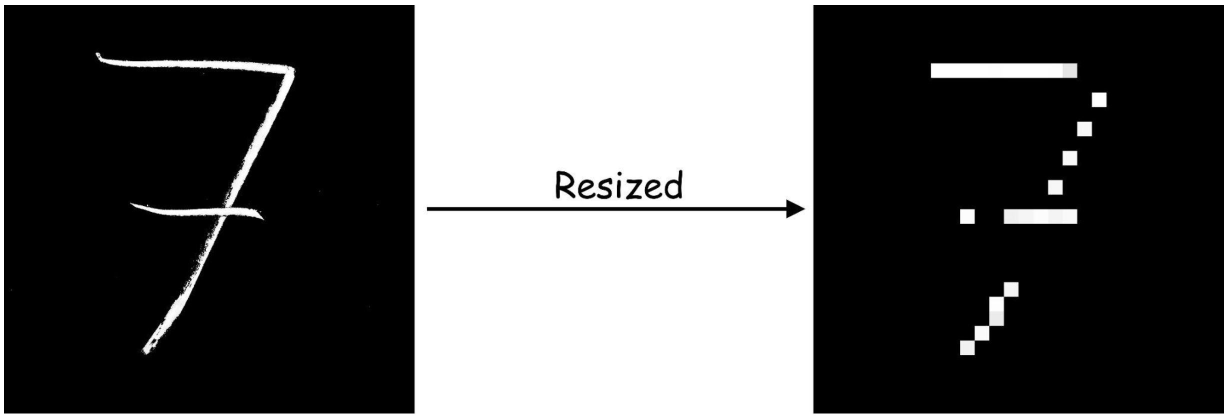
3x3 kernel iteration on padded input with stride of 2

To further improve the performance of our model, we used a *Rectified Linear Unit* (ReLU) and a *Pooling* operation. In general, these are complicated topics that could justify projects of their own. To put it loosely, ReLU cuts out unwanted results in the output of each layer for an appreciable increase in performance and Pooling discards unwanted outputs as they appear to cut down on computation time. Again, these are complicated topics and we made use of built in functions to achieve the best results for our project.

Finally, the results are *flattened* which converts all outputs into a single dimensional vector. We then used the *softmax* function which normalizes the outputs and converts the result into a probability. Finally, we see which digit our input image is most likely to be.

Image Processing

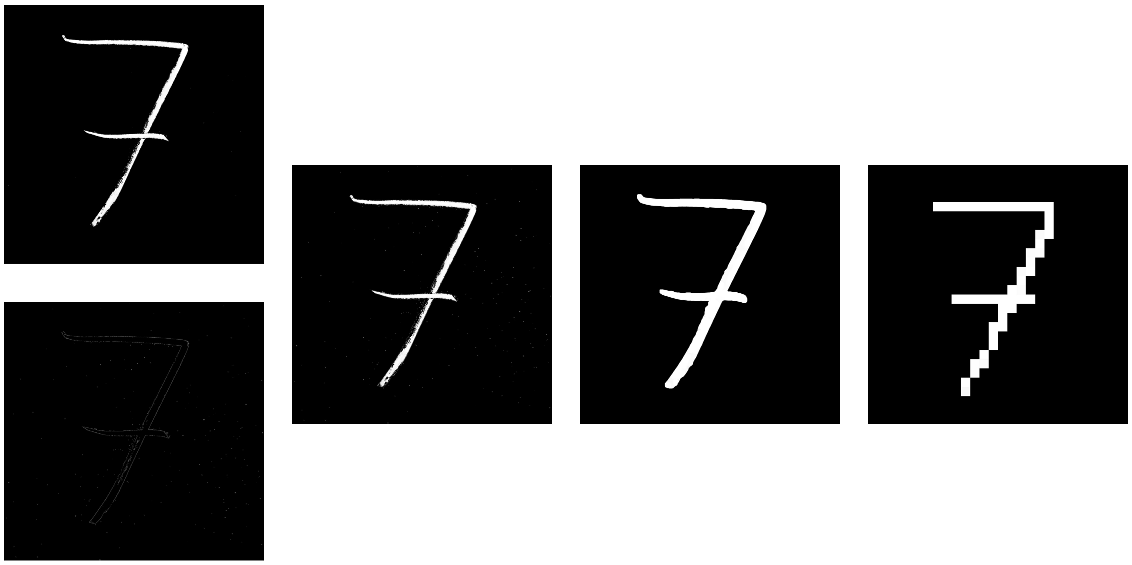
For our problem, we first had to deal with preprocessing the images we plan on giving to our model. To be used effectively with our model, images need to be grayscale with a dark background and dimensions of 28x28. To capture maximal detail of our hand drawn numbers, we took high quality photos of dimension 3024x3024. Resizing these images to 28x28 proved problematic as much of the thin chalk lines were consistently lost.



3024x3024 (Thresholded) 28x28

Therefore, our images required further processing. The steps for processing our images are listed below.

1. Create a duplicate of our given image and perform thresholding on it to create a binary image
2. Create a second duplicate image and perform canny edge detection on it to get an image of the digits edges
3. Add the two images together to create a thickened binary image of our digit
4. Use closing to remove any gaps left in the digit
5. Now that the digit is thick enough and no information should be lost, use a median blur to remove salt and pepper noise in the image
6. Dilate the image so that more of the digit is preserved when resizing
7. Resize the image to 28x28



Threshold and Edge detection   Sum (Noisy)   Dilated and Median Blurred   Resize to 28x28

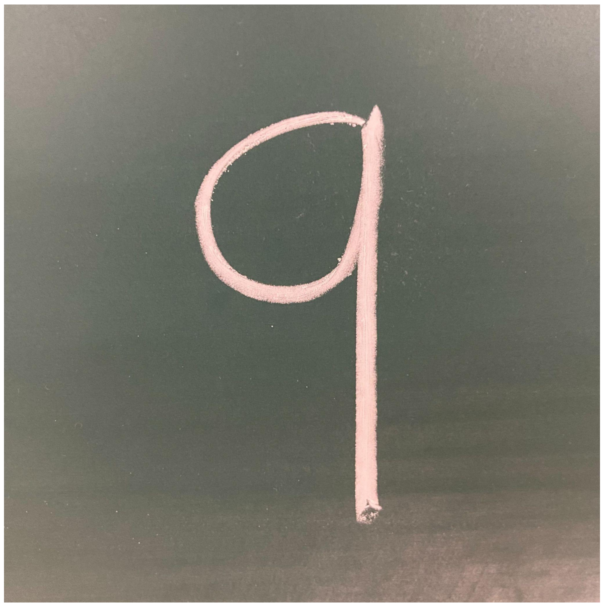
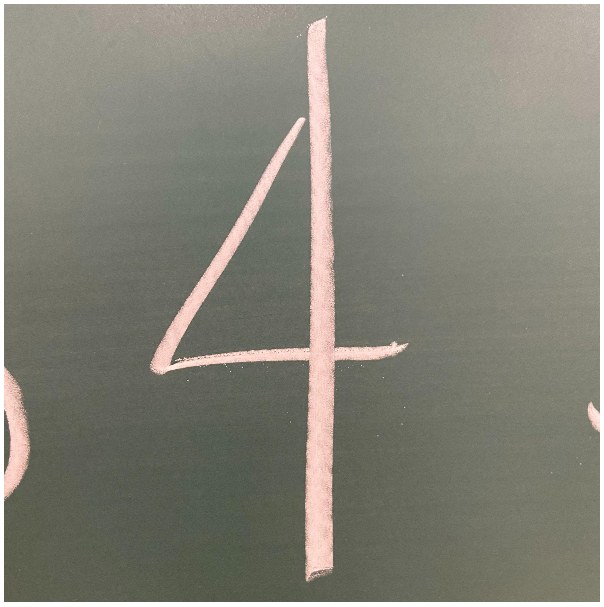
Our program still struggles to process images with noisy backgrounds but we believe that we may be able to propose some possible solutions that we could implement in future iterations of the program.

The first potential solution is a bilateral filtering algorithm. We attempted to implement this but as it wasn't learned in class and it's more complicated than the other filters, we would be unable to implement it effectively or be able to explain it briefly during our presentation. When we attempted to add it, it showed some improvements for recognizing digits written on an unclean chalkboard, but caused a loss of small, important details on those written on a clean chalkboard.

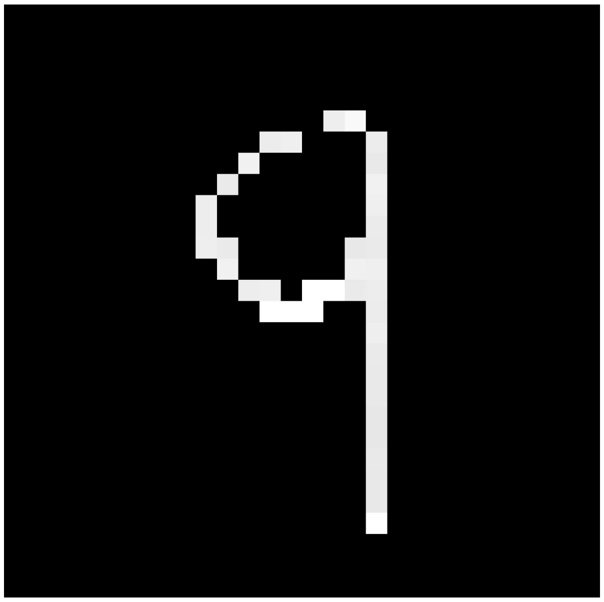
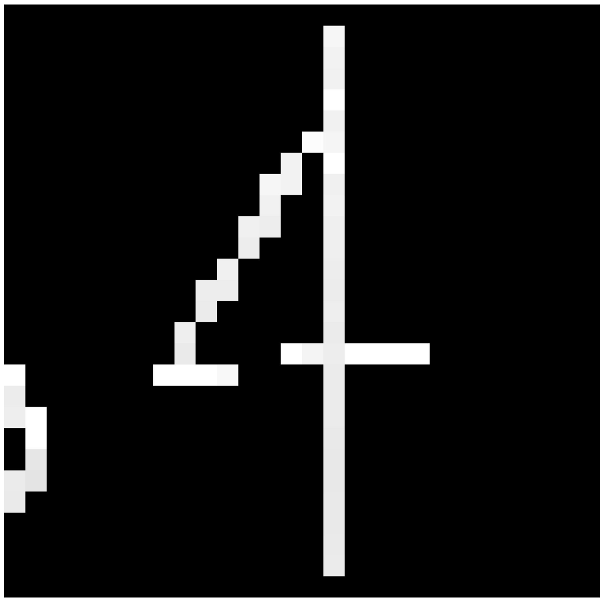
Further potential solutions could be to diversify our data set to create more variation in what the model accepts as input, or general hardware improvements to make the training process more efficient.

Experiments

The first experiment we conducted involved drawing the digits zero through nine on a clean chalkboard, photographing these drawings, applying our pre-processing algorithm on the images, and then feeding the processed images into our model. Included below are a few sample drawings from our experiment.

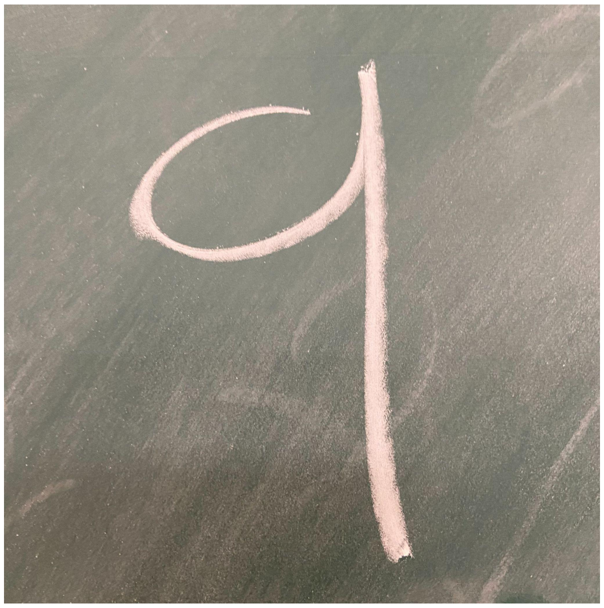



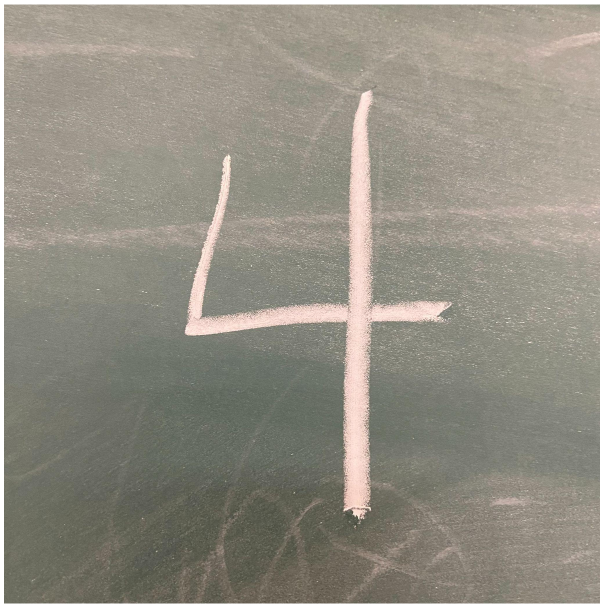
As we can see, the chalkboard that these digits were drawn on is fairly clean, although not perfect as we can still see some white marks around the edges. In the case of the four, there are parts of other drawings included on the left and right sides of the image, especially on the left. We fed these images into our pre-processing algorithm which would hopefully convert the images to grayscale, downsize the images to 28 by 28, and remove any noise gained along the way. Included below are the results of the images presented after applying our image processing algorithms to them.

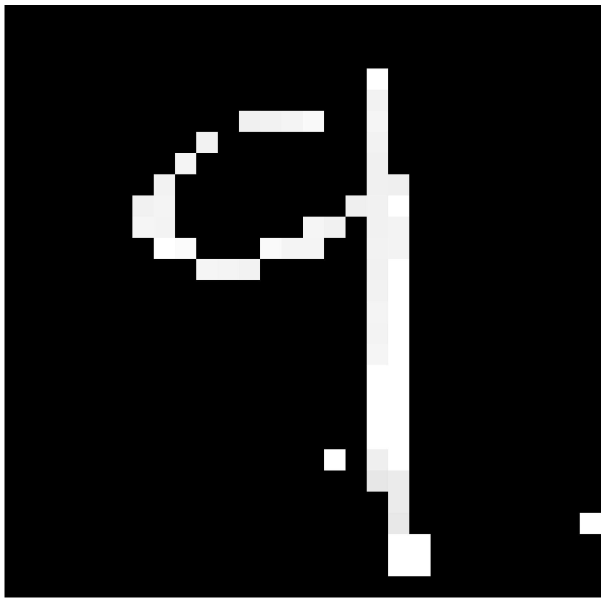
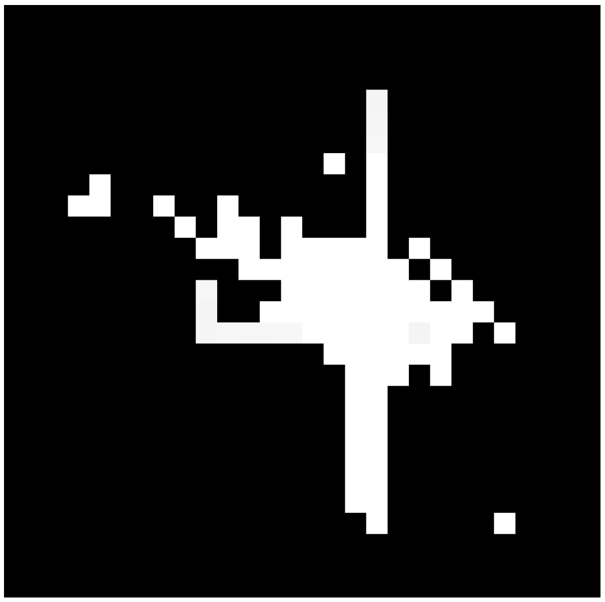
After applying our image processing techniques, we see that our results closely resemble a 28 by 28 version of the input images. The thresholding was able to clean up the background of nine, but in the case of the four, the right writing on the right edge was preserved. This is because the intensity of this segment is very high and consequently, it surpassed our threshold used to remove noise. Our model correctly identified each image after we input these processed images since they resemble the images that our model used to train on.

The second experiment we conducted involved drawing the digits zero through nine on a dirty chalkboard, photographing these drawings, applying our pre-processing algorithm on the images, and then feeding the processed images into our model. Included below are a few sample drawings from our experiment.





In this case, the chalkboard we drew the digits on is severely dirty and the streaks are much more noticeable and are of higher intensity. In the case of the four, we even see additional numbers and letters faded in the background such as an x above and below the four and a 7 in the lower left corner. In this case, we expected our pre-processing algorithm to perform much worse on these images given the circumstances. Included below are the results after inputting the above images into our program.

After observing the resulting output images of our program, we can clearly see the impact that writing on a dirty chalkboard has on our program. The program performed exceptionally well given the circumstances although the images are not perfect. There are outlier pixels on the bottom right of the image and towards the end of the tail of the nine. It is not surprising that the four did not perform very well. The background on the original image was very dirty and so our thresholding was not able to weed out all of the noise. Furthermore, the closing of the image enhanced the noise which is why we see a hotspot of noise in the center of the image. Surprisingly, our model was able to identify both images. It seems reasonable that our model could classify the nine but the four is counter-intuitive. We suspect that the model was able to identify the four through the identification of subtle details. In particular, the tail and the arm of the four. It was very beneficial to conduct this experiment as it challenged our intuition and made us think of potential ways to further adapt our algorithm to work on a larger class of images.

Summary

In summary, a neural network approach to the problem of handwriting recognition, while not perfect, is impressively practical in a primitive state and is further enhanced by the assistance of image processing techniques. Our model could additionally undergo more training with larger datasets alongside numerous potential enhancements to our image processing algorithm that could significantly increase its adaptability.

References

1. Quddoos, Talha. “Computer Vision 101: Build a Digit Recognizer with Tensorflow | Artificialis.” *Medium* https://medium.com/artificialis/get-started-with-computer-vision-by-building-a-digit-recognition-model-with-tensorflow-b2216823b90a.
2. “What are Neural Networks?” *IBM*, https://www.ibm.com/topics/neural-networks.
3. Sorokina, Ksenia. “Image Classification with Convolutional Neural Networks | by Ksenia Sorokina.” *Medium*, 19 November 2017, https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8.
4. “CNN Explainer.” Polo Club of Data Science, https://poloclub.github.io/cnn-explainer/.
5. “A Gentle Introduction to the Rectified Linear Unit (ReLU) - MachineLearningMastery.com.” *Machine Learning Mastery*, 20 August 2020, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Contributions

All group members contributed equally to the writing and the editing of slides and had an equal speaking part in the presentation.

Logan designed image processing code and Michael and Patrick worked together on the design of the neural network code.

In the report, Logan wrote the section of Image Processing, Michael wrote the section on Neural Networks, and Patrick wrote the Introduction and the section on Experiments. All members contributed to the review and editing of the report.