Report for Spring 2021 EE445 Handwritten Digit Recognition

Author: Kyle Neubauer, Paula Penular

Date: May 14, 2021

Introduction

The aim of this project is by implementing a classification algorithm, there would be the ability to recognize handwritten digits from the numbers 0 to 9. The scope of this project will be to test different classifiers in evaluation of their accuracy and error rate in the performance of the particulars of handwritten digit classification. This report will present the group's implementations of convolutional neural networks, CNN, and support vector machines, SVM, in the linear and non-linear varieties.

Motivation and Background

Handwritten digit recognition is to simply put it the process of identifying handwritten numbers. It is true that in the past, starting since the 1980s, computers have struggled and failed to correctly identify written text on paper documents[3]. This was the proposed challenge, to convert written characters into a identifiable digital counterpart. This type of machine learning falls into the image recognition category deep learning. One of great importance, handwritten digit recognition has been seen — online handwriting recognition on computer tablets, recognizing zip codes on mail for postal mail sorting, processing bank check amounts, and numeric entries in forms filled up by hand such as the likes of tax forms[7]. There are also examples that students come across everyday in their college career. These are most notably seen in note taking in programs like oneNote and Notability where even mathematical formulas can be translated to digital form.

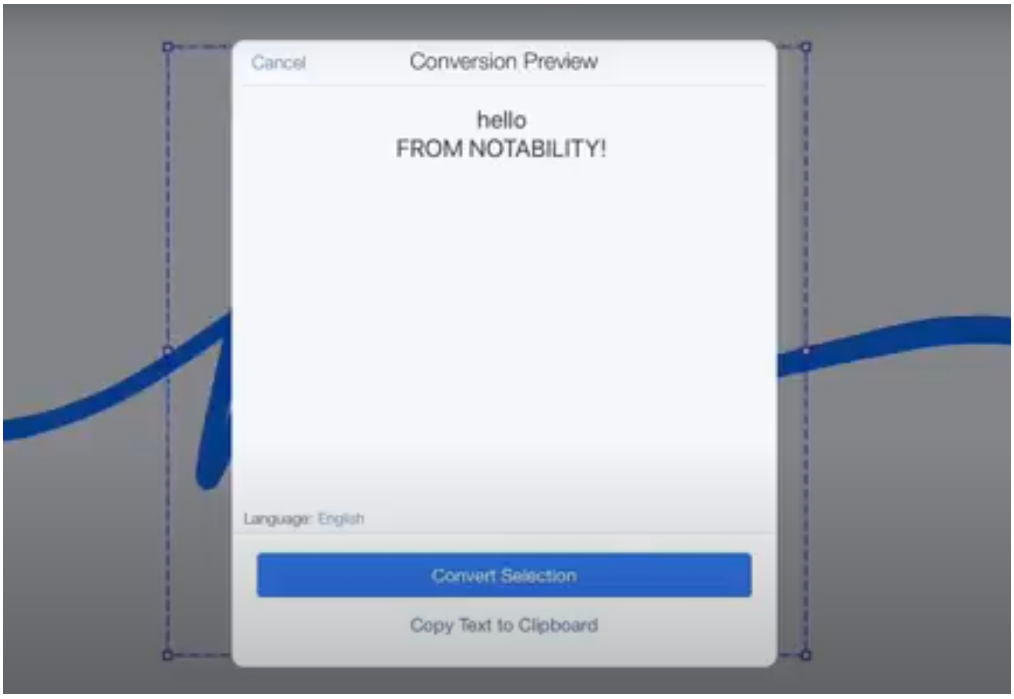

Figure 1: Notability Handwritten Conversion to Digital Figures

There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins[3]. Even with the same person writing, these differences occur that are only exasperated when added to the assortment of different persons and writing styles. The group aims to clear the confusion and achieve the perceptions to the digital world.

Model Set

The model set that was used in this project is the Modified National Institute of Standards and Technology database, or better known as MNIST. Created in 1998, this is a database of handwritten digits[6]. Written by various highschool students and employees of the United States Census Bureau, a training set of 60,000 examples and a test set of 10,000 examples was made up[3]. The training examples are annotated by humans with the correct identification. Each MNIST image is comprised of 28 by 28 pixels in grayscale[3].

The general problem predicted in classifying the given digits are on the grounds of the similarity between the digits like 1 and 7, 5 and 6, 3 and 8, 9 and 8 etc[7]. This also takes in account that people write the same digit in many different ways. This is often seen in the digit ‘1’, where it could be depicted as a straight line, slanted line, or line with a flick at the top[7]. Similarly 7 may be written as an acute angle, with a flick, or drastically acute with a sharp slant[7]. Finally, as stated previously the shear uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digits[3].

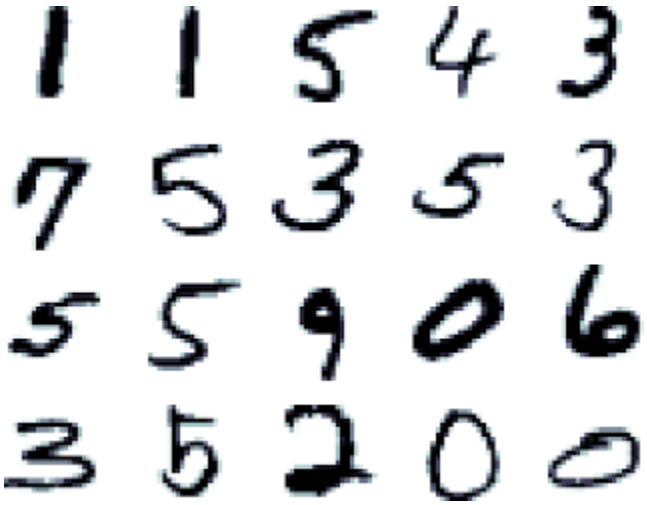

Figure 2: Subset of numbers found in MNIST

This data set is commonly used for training various image processing systems and has been widely at use for training and testing in machine learning. It was found that back in 2004, a best case error rate of 0.42 was achieved by researchers utilizing a new classifier called the LIRA, which is a neural classifier with three neuron layers based on Rosenblatt's perceptron principles[6]. Then in 2011, the error rate was improved to 0.27 percent using a similar system of neural networks. Next, in 2013, a 0.21 error rate was achieved on a regularization of neural networks using DropConnect [6]. In 2016, the single convolutional neural network's best performance was 0.25 percent error rate.Fast forward to 2018, the best recorded error rate of 0.18

was announced by researchers from Department of System and Information Engineering, University of Virginia [6]. This was done using simultaneously stacked three kinds of neural networks of fully connected, recurrent and convolutional neural networks.

According to the HandWiki, there are many different classifiers that can be used to do image recognition on the MNIST [6]. However the one that stood out as possessing the least amount of error rate was convolutional neural networks. Our group will strive to see for ourselves in this process would give way to accuracy. To compare, the group will also use a linear and nonlinear SVM that was taught in class to find the accuracy and error rates. In our process, the group did not use the whole data set for the classifications.

CNN

Next up, we have the convolutional neural network, or better known as CNN. First introduced in the 1980s by Yann LeCun, this is a deep learning algorithm which can take in an input image and assign importance, learnable weights and biases, to various aspects or objects in the image and be able to differentiate one from the other[2]. The preprocessing required in a CNN is much lower as compared to other classification algorithms [2]. While in primitive methods filters are hand-engineered, with enough training, CNN has the ability to learn these filters and characteristics. The architecture of a CNN is comparable to that of the connectivity pattern of neurons in the human brain[8]. The inspiration of CNN was the organization of the visual cortex of the brain where individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field.

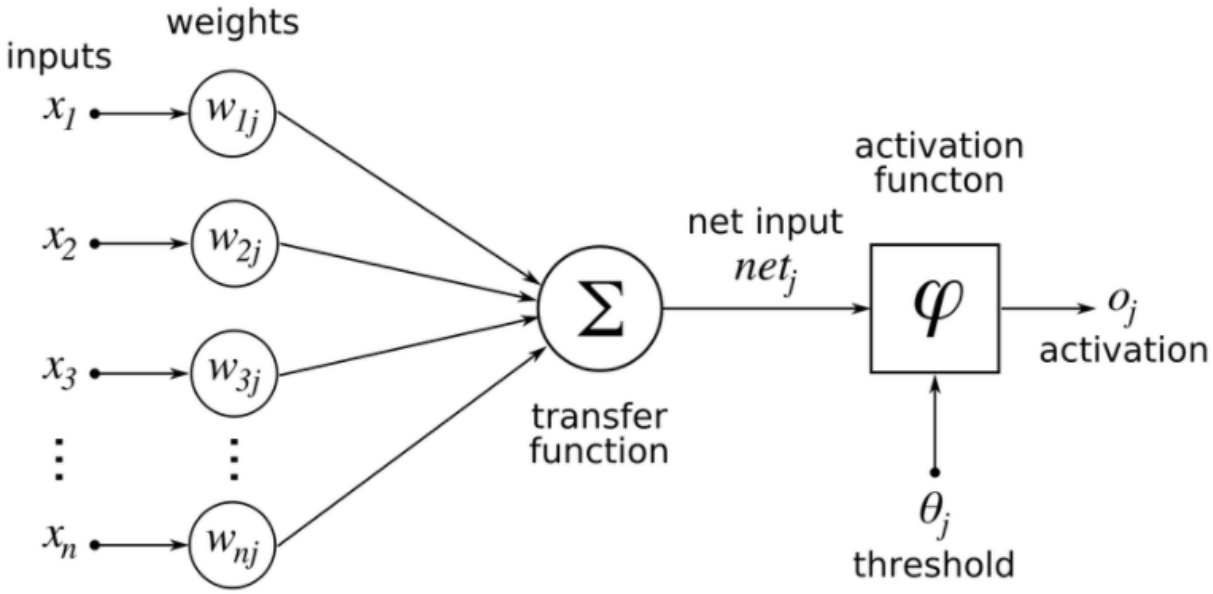

Figure3: Structure of an Artificial Neuron, Basic Component of Artificial Neural Networks

CNN is also able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture was observed to prove a better performance in fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. To simply put it, the network can be trained to understand the sophistication of the image better. It can be said that the aspect of CNN is the ability to reduce images into a more straightforward configuration to work with, without losing the integrity of features that prove critical in getting a more accurate response. In general, when an image is imputed to CNN each layer generates several activation maps. The first layer of the CNN usually detects basic features such as horizontal, vertical, and diagonal edges. The output of the first layer is then fed into the second layer, which extracts more complex features. The deeper into CNN, the layers then start detecting higher level features. This is significant for not only being good in learning features but in a scalable to massive datasets. The group decided to process the MNIST database through CNN from our research into various Python packages that

ran handwritten digit recognition that primarily use CNN for its easy to implement and high accuracy results.

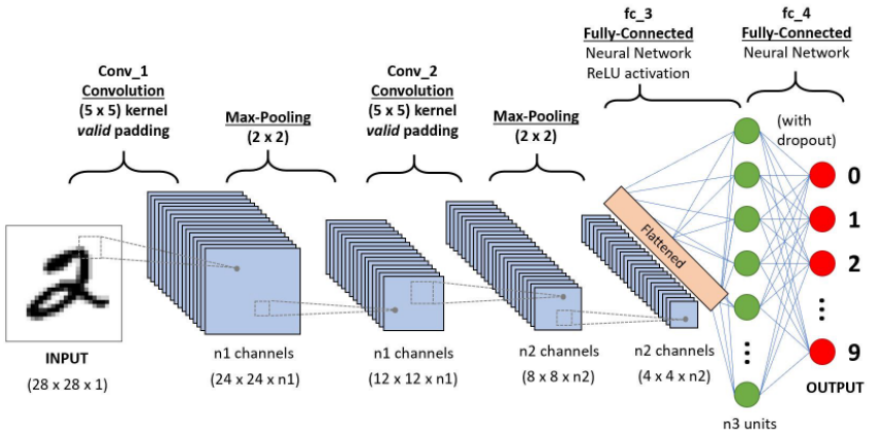

Figure 4: CNN Basic Process of Classifying Handwritten Digits

In starting the project, the group first decided on utilizing Python as the main computer language. This was launched on Google Colab in order to save on GPU and make the process of running the code a more user friendly situation. With the immediate response to any trouble shooting, the foremost arrangement was to download the necessary python libraries.The group has the libraries of numpy to work on the arrays, keras for the filters and layers of CNN, pandas for anayaization of the data, and matplotlib and seaborn for graphics and visualization.

The group was then able to input the data set using the command "`(Xtrain, Ytrain), (Xtest, Ytest) = mnist.load_data()`". This both acted to load the data but to also separate the data to a training set and a testing set. To allow for a faster process, the group made a decision to not use the entire MNIST data set. Instead, only the 60 thousand was taken in a split of 48 thousand toward training, 6 thousand was taken for validation and 6 thousand was taken for testing. This made it to a 80%, 10%, 10% split. Next, while doing the preprocess of the data, one of the most significant commands used was "`np_utils.to_categorical`", which

converted the class vector integers to a binary class matrix. This opened the way for the group to begin the layer process.

The group used the sequential command to group a linear stack of layers into a tf.keras.Model in order to provide training and inference features. Then a spiral convolution layer with max pooling to downsample the input representation using the maximum value window defined by the pool_size for each dimension along the features axis was added. To go through the method, the group adds a spiral convolutional layer between the input image and a filter of a particular size MxM[8]. Then activation by ReLu, f(x) = max(0,x), is to overcome the vanishing gradient problem, allowing models to learn faster and perform better to the input shape[8]. Max pooling is then used to take the spatial data and downsample it. The groups then convolutes again, activation ReLu, and max pooling. Then the data is flattened, which does not affect the batch size, and a dense layer with activation Relu is made. Dense layers add an interesting non-linearity property, thus they can model any mathematical function. However, they are still limited in the sense that for the same input vector obtained is the same output vector. Finally, the data is run through a dense layer again with an activation of softmax[2]. Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector[2].

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 64)                51264
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Model Summary of layers

The group now takes the data to epoch. Using 5 epoch, a hyperparameter to define the number times that the learning algorithm will work through the entire training dataset[2]. Where One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters[2]. In the final epoch, there was an accuracy of 0.994 and a loss of 0.0043. Given that, when tested, it was able to predict which array to use to print out the right result.

```
[205] model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
      history = model.fit(Xtrain,Ytrain, verbose=1,  epochs=5, batch_size=32, validation_data=(x_validation, y_validation))

      Epoch 1/5
      1500/1500 [==============================] - 35s 23ms/step - loss: 0.1101 - accuracy: 0.8112 - val_loss: 0.0143 - val_accuracy: 0.9833
      Epoch 2/5
      1500/1500 [==============================] - 34s 23ms/step - loss: 0.0158 - accuracy: 0.9799 - val_loss: 0.0096 - val_accuracy: 0.9873
      Epoch 3/5
      1500/1500 [==============================] - 34s 23ms/step - loss: 0.0102 - accuracy: 0.9868 - val_loss: 0.0059 - val_accuracy: 0.9930
      Epoch 4/5
      1500/1500 [==============================] - 34s 23ms/step - loss: 0.0073 - accuracy: 0.9908 - val_loss: 0.0060 - val_accuracy: 0.9915
      Epoch 5/5
      1500/1500 [==============================] - 34s 23ms/step - loss: 0.0058 - accuracy: 0.9928 - val_loss: 0.0043 - val_accuracy: 0.9943
```

Figure 6: Training Model over 5 Epochs

In the end, the group was able to graph the results of the epoch. With the yellow line meaning training data and the red line representing the validation data it was noticed that the

more epoch run, the lower the loss goes and the viceversa happens to the accuracy rises. It was most notably observed that the training loss drops drastically within the first two epoch, while angina the viceversa happens to the accuracy.

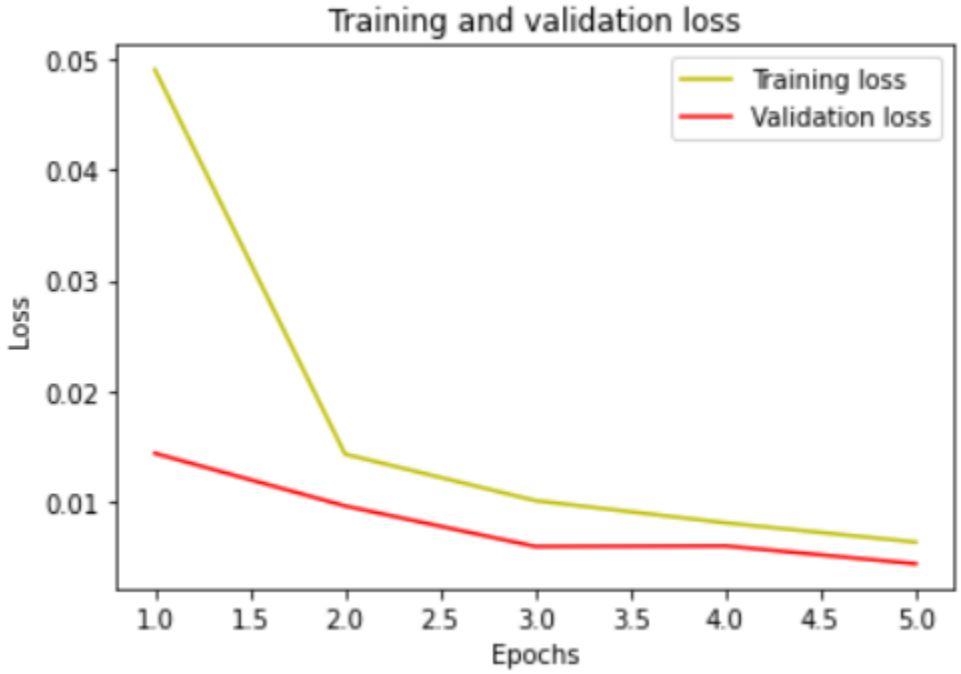

Figure 7: Training and Validation Loss vs Epochs

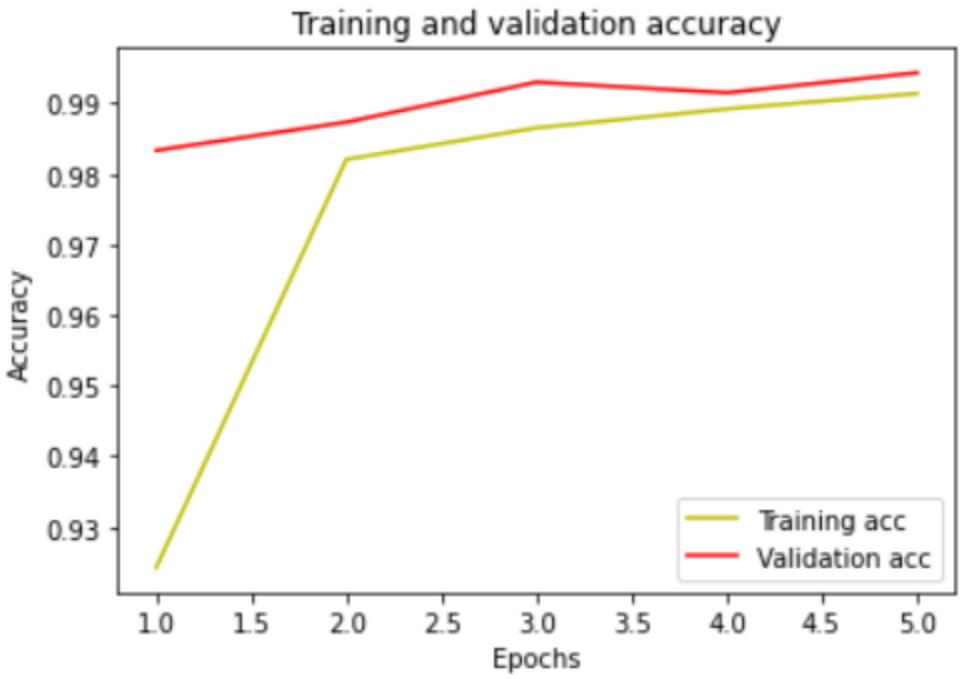

Figure 8: Training and Validation Accuracy vs Epochs

Linear SVM

Support Vector Machines use classification algorithms to classify data into categories making it great for image classification. Support Vector Machines create hyperplanes to linearly separate data and depending which side of the hyperplane are the data classified accordingly. Depending on the hyperplane positioning can larger or small margins be made against data. The larger the margins between the hyperplane and the data the better fitted. To improve computational efficiency, a linear kernel is used to transform the data to a more suitable format for classification. Because when inputting a lot of data into SVM models it can be redundant for a majority of the data except around the decision variables, the number of data split between training, validation, and test was reduced by over 90% of the original mnist data. The data was then split into 60% training (3000), 20% validation (1000), and 20% test (1000). The main purpose of this was to reduce redundancy, improve accuracy, and significantly reduce training

time. Before inputting data into the linear SVM model, it's important to feed it data that it can classify properly, which means the data needs to be within a certain range of either 0 to 1 or -1 to 1, therefore data was normalized to be within the range of -1 to 1 by dividing by 255, multiplying by 2 and subtracting by 1. Then scaling the data to fit the model into a single array for example 3000, 784. Now that the data has been reduced significantly and transformed, it can be fed into the linear SVM model which took a few minutes vs hours to train. After successfully training the model, the model can be fed the testing data to predict the handwritten digits. Those predictions are compared to the testing data labels and all put into a confusion matrix seen in figure 8 below.

```
array([[100,   0,   0,   0,   0,   1,   0,   0,   1,   0],
       [  0, 119,   0,   0,   0,   0,   0,   0,   0,   0],
       [  3,   0,  84,   3,   2,   0,   0,   0,   7,   0],
       [  2,   0,   5,  87,   0,   4,   0,   1,   2,   1],
       [  1,   0,   0,   0,  86,   0,   0,   2,   1,   2],
       [  3,   0,   0,   3,   2,  74,   1,   0,   1,   1],
       [  1,   0,   0,   0,   0,   6,  95,   0,   0,   0],
       [  0,   0,   7,   0,   0,   0,   0, 106,   1,   1],
       [  0,   4,   1,   5,   0,   4,   0,   0,  79,   1],
       [  0,   0,   0,   0,   3,   0,   0,   4,   1,  82]])
```

Figure 9: Linear SVM Confusion Matrix Xtest predictions vs Ytest true labels

The confusion matrix helps visualize where the data was classified for each handwritten digit from 0 to 9. The rows are based on the number of digits populated in the data, for example, for digit 9 there were 90 times it showed up in the test set and out of those 90 times, it was correctly classified 82 times out of 90. Based on all the digit classifications, it was found that the linear model had an accuracy of 91.2%. The full report can be found in figure 9 featuring a f1 score of 91.2%, as the f1-score gives you the harmonic mean of precision, and helps give a better metric when there are unbalanced amounts of data in each classification.

Non-Linear SVM

Non-Linear Support Vector Machines are used for when the data is not linearly separable. The data can still be separable, however not by a linear hyperplane. One kernel that can be used is the radial basis function (rbf); the mathematical model shown in figure 10 below.

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Figure 10: Gaussian Radial Basis Function Kernel

The radial basis function helps with classifying multivariable functions by linear combinations, this way the data can be mapped in more than one dimension so the model can better fit the data. The rbf kernel uses gamma, and depending on the size of gamma is whether the data is overfitting when large, or underfitting when too small. By fine tuning the hyperparameters, the gamma value can be optimized to result in the greatest accuracy possible. For the non-linear model the data was split in the same quantities as the linear SVM model for an accurate comparison; 3000 training, 1000 validation, 1000 test sets. After feeding the data into the non-linear svm rbf kernel model, a predicted accuracy of 92.1% was generated and the confusion matrix shown in figure 11.

```
[[100   0   1   1   0   0   0   0   0   0]
 [  0 119   0   0   0   0   0   0   0   0]
 [  4   0  83   2   1   0   0   2   7   0]
 [  1   1   5  88   0   1   0   2   3   1]
 [  0   0   2   0  87   0   0   3   0   0]
 [  2   0   0   2   0  73   3   2   2   1]
 [  1   0   0   0   0   2  95   4   0   0]
 [  0   0   6   0   0   0   0 107   1   1]
 [  0   0   0   2   0   3   0   0  88   1]
 [  0   0   0   0   3   0   0   6   0  81]]
```

Figure 11: Confusion matrix of rbf kernel

To begin fine tuning the hyperparameters, it's essential to lay out the parameters such as C: 1, 10, 100, and gamma: 1e-2, 1e-3, 1e-4. When gamma is high, such as 0.01, the training accuracy is high, but the test scores are low in comparison. When gamma is low, such as 0.0001 the training accuracy is low because the decision region is much broader. Therefore, the best region is somewhere in between for good training accuracy and test score. The figure 12 below highlights

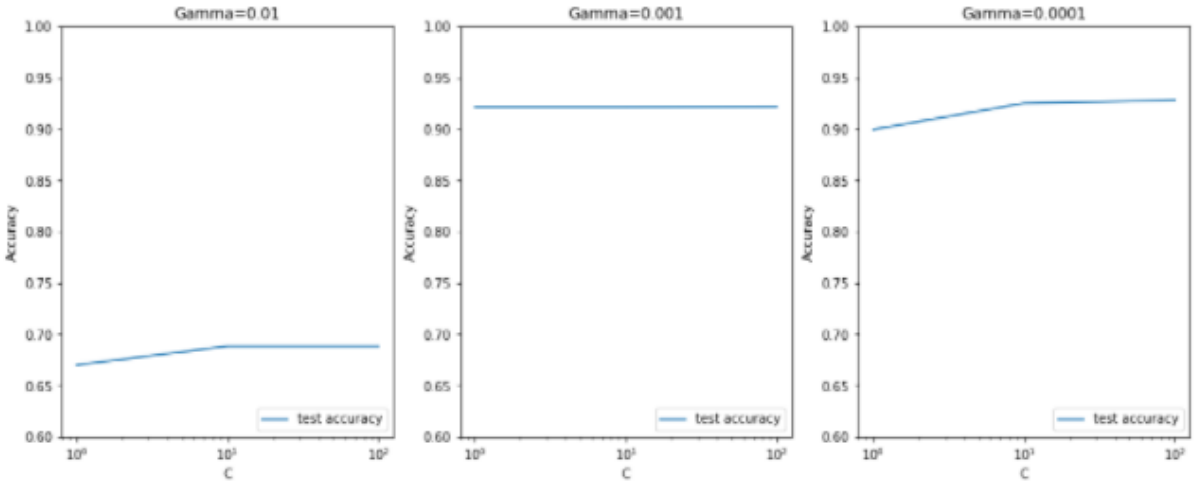

Figure 12: Parameter C vs Accuracy for each Gamma value

The training accuracy for the C parameters vs accuracy, however when creating the models the validation set would not populate. After finding the optimal values for C and Gamma 10 and 0.001 respectively, the optimized model can then predict the handwritten digits in the test set. The final accuracy for the non-linear svm rbf kernel after fine tuning the hyperparameters is 93.3%.

Final thoughts

In conclusion, it can be taken that the convolutional neural network was the fastest model, only taking a few minutes, and the most accurate. The final results between the classifications were 99.4% for CNN, 91.2 % for Linear SVM, and Non-linear SVM was 93.3%. For a final thought, the CNN will be the go to model when processing handwritten digit recognition.

If anything could have been done any different within this project, some improvement would definitely be made. Towards CNN, training in more randomness with a shuffled data would be taken into account. It was also noted that binary cross entropy was a better fit to compare each of the predicted probabilities. Towards the Non-linear SVM, it was the better preformed SVM as was per the circumstances stated above and the Linear SVM were much faster to train. If given more time, more work would be given to optimise the data for better performance.

On a personal note for this project, this was the first project that the group used Python in. While learning the new language was a challenge, it was a decision made due to the vast resources that could be found on the internet with ready examples of handwritten classifications. With a new language also came many problems. A struggle with time came, and worked on a mismanagement on work progress. Discovering pakest and Google Colab was a miracle in the making, that helped lighten the struggles but also came with a drawback. Getting stuck on formulas and code that wouldn’t process was a major factor. There was also the issue with utilizing tensorflow. It was ultimately made that tensorflow was not the way to go for the group and the method was scrapped. Yet, with all the struggles, all the hardships, this project also tremendously nurtured the group's skills in machine learning. By going through this trial, the

group now has the confidence and knowledge to take their machine learning education to the next level.

References

[1]S. Saha, “A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way,” *Towards Data Science*, Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[2]Lesson LogiCS, “Train a CNN on Google colab | MNIST Digit classifier - Deep learning codes,” *www.youtube.com*, Jan. 29, 2021. https://www.youtube.com/watch?v=TMw9dTBNiRs.

[3]D. Horn, “Handwritten Digit Recognition Using Artificial Intelligence on a Low-Cost FPGA Board – Digilent Blog,” *Digilent Blog*. https://blog.digilentinc.com/handwritten-digit-recognition-using-artificial-intelligence-on-a-low-cost-fpga-board/#:~:text=In%20the%20world%20of%20artificial%20intelligence%20%28AI%29%2C%20the (accessed May 16, 2021).

[4]Google Colab, “tflite_c02_transfer_learning,” 2018. https://colab.research.google.com/github/tensorflow/examples/blob/master/courses/udacity_intro_to_tensorflow_lite/tflite_c02_transfer_learning.ipynb#scrollTo=BDlmpjC6VnFZ.

[5]N. Patel, “MNIST Digit recognition using SVM,” *kaggle.com*, 2019. https://www.kaggle.com/nishan192/mnist-digit-recognition-using-svm.

[6]HandWiki, “MNIST database - HandWiki,” *handwiki.org*, Nov. 18, 2020. https://handwiki.org/wiki/MNIST_database (accessed May 16, 2021).

[7]G. Jain and J. Ko, “Handwritten Digits Recognition ECE462 -Multimedia Systems | Project Report University of Toronto,” 2008. Accessed: May 16, 2021. [Online]. Available: http://individual.utoronto.ca/gauravjain/ECE462-HandwritingRecognition.pdf.

[8]B. Dickson, “What are convolutional neural networks (CNN)?,” *TechTalks*, Jan. 06, 2020. https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/.

[9]M. Gurucharan, “Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network,” *upGrad blog*, Dec. 07, 2020. https://www.upgrad.com/blog/basic-cnn-architecture/.