Handwritten Text Recognition using Convolutional Neural Networks in Deep Learning

Huan Ho

Department of Computer Science, North Dakota State University

University Honors Capstone Project

Dr. Simone Ludwig

May 12, 2021

Abstract

In this project, the handwritten text recognition machine is implemented by using the Convolutional Neural Network (CNN) model with 3 different datasets, MNIST, EMNIST, and CASIA-HWDB, and the Convolutional Neural Network (CNN) model is built by using TensorFlow and Keras in Python. The results show that the MNIST dataset reaches the highest accuracy and have the simplest model; EMNIST achieves an 87.53% accuracy, but it can be improved if the model is trained for more epochs; CASIA-HWDB only reaches 42.5% accuracy in testing images due to the limited number of samples for each character in the test image set. One major conclusion from the project is that complex models do not mean better accuracy. Complex models might diverge while training. Sometimes, a simpler model can reach a better accuracy. In the future, the model can be trained for more epochs to determine whether the models improve or implemented and trained with different types of neural and test if it will achieve better accuracy.

1. Introduction

Nowadays, there is a huge demand for storing handwritten paper documents into a computer-readable form in order to allow people to access them easier later. However, asking humans to enter paper documents' information manually is time-consuming and will increase the inaccuracy caused by humans. Moreover, today's technologies, such as mobile phones, tablets, and even smartwatches, allow users to handwrite the words and the words will be converted into text. Automatic recognition of medical forms, and processing of other types of files, such as postal mail sorting automation and bank checks identification, are all examples of applications for handwritten text recognition. This process will require a machine to recognize users' handwriting. In order to reduce the inaccuracy rate and identify the handwriting more efficiently, implement a handwritten recognition machine that can be trained is crucial. Handwritten Text Recognition is a popular field of research in Machine Learning, and there are various ways to implement the recognition. This project develops handwritten text recognition by using the Convolution Neural Network.

Convolutional Neural Network (CNN) is a Deep Learning algorithm and is designed for data with spatial structure, such as images. The CNN model architecture is presented in Fig. 1. The model takes in an input image, assigns importance to various aspects in the images, and differentiates one from the other. For Handwritten Character Recognition that uses CNN, characters are given as input. The model has multiple hidden layers and an output layer. Hidden layers are composed of several convolutional layers and pooling (or subsampling) layers optionally followed by fully connected layers, which

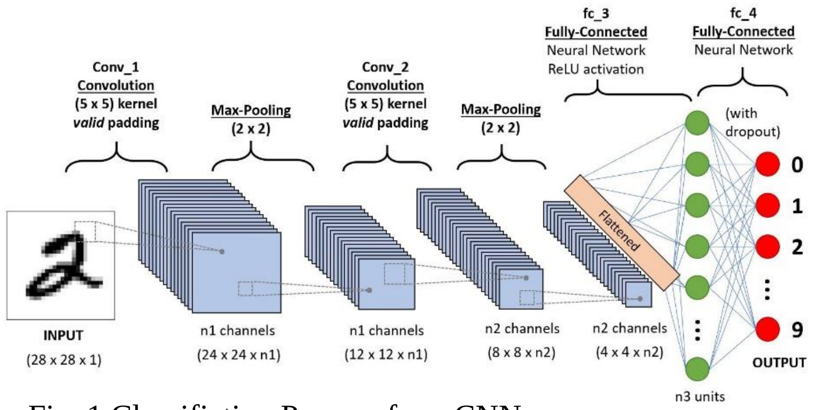

Fig. 1 Classifiation Process for a CNN

include but not limited to dropout layers, dense layers, and flatten layers. The convolutional layer is responsible for feature extraction from input data, and the pooling layer reduces the output information from the previous convolutional layer and reduces the computational complexity of the model (Siddique et al., 2019). Once the model is built and trained, the test set can be predicted and compared with the actual character to determine the accuracy.

2. Problem Statement

The field of handwritten recognition has achieved significant real-world success in targeted applications these days. However, handwritten recognition is an extensive research topic in Artificial Intelligent, and there is still room for improvement in this topic, such as performance accuracies. Characters written by humans vary a lot in multiple factors, such as curves and sizes. Thus, everyone's writing is not the same. It is challenging to develop a handwritten recognition with a stable result, especially for the characters that involve complex stroke, such as Chinese characters. Implementing a Chinese handwritten recognition is challenging due to the large size of the underlying character inventory, about 27,000 entries, not including those characters that are not commonly used ("Real-Time Recognition of Handwritten Chinese Characters").

The purpose of this capstone project is to investigate how do the depth of the network and the number of filters in the convolutional layers influence the classification accuracy, how do the different designs of models affect the accuracy of one dataset, and how to improve the accuracy of the handwritten in real-world.

3. Datasets Used

This project will use three different datasets: MNIST dataset, EMNIST dataset, and CASIA-HWDB dataset.

3.1 MNIST

The MNIST is the most famous handwritten digits dataset that is used in a lot of image processing projects. It is derived from the EMNIST dataset and contains 60,000 training images and 10,000 testing images (Lecun et al, 1998). MNIST dataset has the highest accuracy among all three datasets, around 99% accuracy (Lecun et al, 1998).

The MNIST dataset is downloaded from the Kaggle website. It has two csv files, mnist_train.csv and mnist_test.csv. Each row in both files consists of 785 values. The first column is the label that is a number from 0 to 9 and the remaining 784 values are the pixel values, each of them is a number from 0 to 255.

Before using the dataset to train the model, the data for both files need to be split into images (X) and labels (Y). The images need to be reshaped to have the same square size of 28x28 pixels and a single-color channel, which means the images are changed to grayscale. The pixel values for each image also need to be normalized to a range of 0 and 1. The labels, which are integers, are transformed into a 10 element binary vector with a 1 for the index of the class value and 0 values for all other classes by using one-hot coding.

3.2 EMNIST

The second dataset is the EMNIST dataset. EMNIST stands for extended MNIST, which is derived from the NIST Special Database 19 (Cohen et al., 2017). This is a handwritten digits and

English characters dataset. There are six different splits provided in the dataset: ByClass, By Merge, Balanced Letters, Digits, and MNIST (Cohen et al., 2017). This project uses the ByClass EMNIST dataset to train the model. The EMNIST contains a total of 240,000 training images, and 40,000 testing images (Cohen et al., 2017). The EMNIST dataset has an approximate 90% accuracy with CNN, and the accuracy value varies according to different split. This dataset's accuracy is a little bit lower than the MNIST dataset's, due to the fact that it involves more complex characters. A paper from Mor et al. used a CNN with two convolutional layers and one dense layer obtaining an accuracy of 87.1% over EMNIST ByClass (Mor et al., 2019).

EMNIST dataset is loaded directly in python by using emnist package. This emnist package provides functionality to download and cache the dataset, and to load it as NumPy arrays. After the dataset is loaded, the images also need to be reshaped to a square size of 28x28 pixels and a single-color channel, and the pixel values need to be normalized to a range of 0 and 1. The labels are transformed into a 62 element binary vector by using one-hot coding because there are 26 lower case English characters, 26 upper cases English characters, and 10 numbers.

3.3 CASIA-HWDB

The last one is the CASIA-HWDB dataset, which is a Chinese character dataset that is built by the Institute of Automation of Chinese Academy of Sciences (Liu et al., 2011). This dataset contains 940,800 training images and 235,200 testing images, which involves a total of 7,356 entries (7,185 Chinese characters and 171 symbols) (Liu et al., 2011). The CASIA-HWDB dataset requires a bigger and more complex model to train since it involves more entries and the stroke for Chinese characters is more complex. Fujitsu R&D Center implemented a CNN with 10 layers with the CASIA-HWDB and result in a 94.77% accuracy (Yin et al., 2013). Yang et al. report the accuracy for handwritten Chinese character recognition by using the CASIA-HWDB dataset as training data and HCL2000 dataset, another dataset for Chinese handwritten characters, for testing in their book. With AlexNet (the name for CNN), the accuracy reaches 85.93%, and the accuracy reaches 93.74% with ATC system (Yang et al., 2019).

The CASIA-HWDB dataset is downloaded from the "CASIA Online and Offline Chinese Handwriting Databases" website. This project uses the HWDB1.1 dataset, which includes 3,755 classes with 897,758 training samples and 223,991 testing samples. The dataset is downloaded as two zip files, one for training samples and one for testing samples, and .alz files once extract the zip files. Then they need to be unzipped again to get the folders containing multiple .gnt files. I use python to help me convert the .gnt files to .tfrecord files, which is a TensorFlow Record file that stores a sequence of binary records. The .tfrecord files are used to train the CASIA model.

4. Design and Method

The models used in this project are Convolutional Neural Network (CNN). The CNN models are built from scratch by using TensorFlow and Keras in Python. Multiple models with different architectures are trained for each dataset, and the model with the best accuracy for each dataset is used to predict the handwritten character images I collected outside of the dataset to test the character recognition accuracy.

4.1 Convolutional Neural Networks Model Architecture

For each dataset, a basic model with a convolutional layer and a max-pooling layer is implemented first. I improve the model by making one modification each time, and train the model to identify if there is any improvement to the model accuracy and loss. The modification includes but not limited to changing the filter size, kernel size, pool size, or stride parameter, add another set of convolutional and max-pooling layers, add a fully connected layer, and change the activation function used.

When compiling the model, all the models in every dataset used “categorical cross-entropy” as the loss function to calculate the cross-entropy loss between the labels and predictions by computing the following sum:

$$Loss = - \sum_{i=1}^{output\ size} y_i \cdot \log \hat{y_i}$$

where $\hat{y_i}$ is the i^{th} scalar value in the model output, y_i is the corresponding target value, and the output size is the number of scalar values in the model output. Since the handwritten text recognition is a multi-class classification, this loss function will be optimized.

I use “adam” as the optimization algorithm as the optimizer for majority of the models in every dataset. Adam is used to update network weights iterative based on the training data. I choose Adam because it is effective, and it can achieve good results fast from my previous experience in building the CNN model.

The activation function used for the last fully connected layer for every model in every dataset is the “softmax” function. Softmax assigns decimal probabilities to each class in a multi-class classification, which helps the training process converge more quickly. The softmax layer always has the same number of nodes as the output layer, which is the number of classes.

4.1.1 MNIST CNN Model

For the MNIST dataset, I designed and trained 10 CNN architectures, as is shown in Table 1 (the sequence of models is not in training order). MNIST dataset has a simpler CNN model design because it has fewer labels and it has a large number of samples for each label in both the training and testing dataset. I started by implementing a 4-layers model, and then made modifications to existing layers or added a layer to the previous model step by step. For the first 4 models, The filters and kernel size of convolutional layers are modified. Then I add additional layers to the previous model, up to 7 layers. The input data for the model is the 28 x 28 gray-scale image from the MNIST dataset, and the output is the vector with the class size that stores the probability of each class in it. The model is trained with the training samples, and the validation data is the testing samples. Each model trains for 5 epochs.

M1	M2	M3	M4	M5
4 Layers	4 Layers	4Layers	4 Layers	5 Layers
Input data (28 x 28 gray-scale image)				
Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5
MaxPool – 2				
Flatten Layer				
				FC – 128 ‘relu’
FC – 10: ‘softmax’				

M6	M7	M8	M9	M10
5 Layers	6 Layers	6 Layers	6 Layers	7 Layers
Input data (28 x 28 gray-scale image)				
Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5
	MaxPool – 2	MaxPool – 2		MaxPool – 2
Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 5	Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 3
MaxPool – 2				
Flatten Layer				
			FC – 128 ‘relu’	
FC – 10: ‘softmax’				

Table 1. MNIST CNN Configurations

4.1.2 EMNIST CNN Model

For the EMNIST dataset, I designed and trained 10 CNN architectures, as is shown in Table 2 (the sequence of models is not in training order). The EMNIST dataset has a total of 62 classes (capital case for English character, lower case for English character, and number from 0 to 9). I started by implementing a 6-layers model, which is more than the MNIST initial model layers, and then made modifications to existing layers or added layers to the previous model step by step. The input data for the model is the 28 x 28 gray-scale image from the EMNIST dataset, and the output is the vector with the length of the class size that stores the probability of each class in it. The model is trained with the training samples, and the testing samples is used as the validation data. The first three models are only trained for 5 epochs, M4 and M5 are trained for 8 epochs, M6, M9, and M10 are trained for 10 epochs, and M7 and M8 are trained for 15 epochs. I increase the epochs when the accuracy from the previous model is not as good. Thus, I increase the epoch of the model to see if the accuracy improves. M5, M6, and M7 have the same

M1	M2	M3	M4	M5
6 Layers	8 Layers	8 Layers	8 Layers	10 Layers
Input Data (28 x 28 gray-scale image)				
Conv2D – 64 ‘relu’ Kernel: 4	Conv2D – 64 ‘relu’ Kernel: 4	Conv2D – 64 ‘relu’ Kernel: 4	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5
MaxPool – 2				
Conv2D – 32 ‘relu’ Kernel: 4	Conv2D – 32 ‘relu’ Kernel: 4	Conv2D – 32 ‘relu’ Kernel: 4	Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 3
	MaxPool – 2			
Flatten layer				
FC – 128: ‘relu’		FC – 256: ‘relu’		
	Dropout Rate: 0.2			Dropout Rate: 0.2
		FC – 128: ‘relu’		
				Dropout Rate: 0.2
FC – 62: ‘softmax’				

M6*	M7*	M8	M9	M10
10 Layers	10 Layers	12 Layers	12 Layers	12 Layers
Input Data (28 x 28 gray-scale image)				
Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 32 ‘relu’ Kernel: 5	Conv2D – 1 ‘relu’ Kernel: 5
MaxPool – 2				
Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 3	Conv2D – 32 ‘relu’ Kernel: 5	Conv2D – 64 ‘relu’ Kernel: 5	Conv2D – 32 ‘relu’ Kernel: 5
MaxPool – 2				
		Conv2D – 1 ‘relu’ Kernel: 3	Conv2D – 128 ‘relu’ Kernel: 3	Conv2D – 64 ‘relu’ Kernel: 3
		MaxPool – 2		
Flatten layer				
FC – 256: ‘relu’				
Dropout – Rate: 0.2				
FC – 128: ‘relu’				
Dropout – Rate: 0.2				
FC – 62: ‘softmax’				

Table 2. EMNIST CNN Configurations

configurations. However, I use different optimizers while compiling the models. M5 uses Adam optimizer; M6 uses Adamax optimizer, and M7 uses SGD optimizer. For M10, I also implement a function that uses ImageDataGenerator from Keras.preprocessing.image package to augment the input data for training. Also, EarlyStopping is added to M6 to M10, which allows the model to strop training when validation loss value doesn't improve anymore for 3 epochs.

4.1.3 CASIA-HWDB CNN Model

For the CASIA-HWDB dataset, the CNN architectures are shown in Table 3 (the sequence of models is not in training order). CASIA-HWDB dataset has a total of 3755 classes, thus the number of neurons for the last fully connected layers is 3755. I started by implementing a 5 layers model, and then made modifications to existing layers or added layers to the previous model step by step. The input data for the model is the 64 x 64 gray-scale image from the CASIA-HWDB dataset, and the output is the vector with the class size that stores the probability of each class in it. The model is trained with the training samples, and testing samples are used as the validation data. Each model is trained for 150 epochs since this dataset has more classes and it is harder to train.

M1*	M2	M3	M4	M5
5 Layers	6 Layers	6 Layers	6 Layers	6 Layers
Input Data (64 x 64 gray-scale image)				
Conv2D – 64 'relu' Kernel: 3	Conv2D – 32 'relu' Kernel: 3 Strides: 1 Padding: same	Conv2D – 32 'relu' Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 'relu' Kernel: 5 Strides: 1 Padding: same	Conv2D – 32 'relu' Kernel: 5 Strides: 1 Padding: same
MaxPool – 2	MaxPool – 2 Padding: same	MaxPool – 2 Strides: 2	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same
	Conv2D – 64 'relu' Kernel: 3 Padding: same	Conv2D – 64 'relu' Kernel: 5 Strides: 1 Padding: same	Conv2D – 32 'relu' Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 'relu' Kernel: 5 Strides: 1 Padding: same
	MaxPool – 2 Padding: same	MaxPool – 2 Strides: 2	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same
Flatten layer				
FC – 32: 'relu'				
FC – 3755: 'softmax				

M6*	M7	M8	M9	M10
6 Layers	7 Layers	8 Layers	8 Layers	8 Layers
Input Data (64 x 64 gray-scale image)				
Conv2D – 32 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 32 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 32 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 32 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 ‘relu’ Kernel: 7 Strides: 1 Padding: same
MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2
Conv2D – 64 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 64 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 128 Kernel: 5 Strides: 1 Padding: same
MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2
			Conv2D – 128 ‘relu’ Kernel: 5 Strides: 1 Padding: same	Conv2D – 256 Kernel: 3 Strides: 1 Padding: same
			MaxPool – 2 Strides: 2 Padding: same	MaxPool – 2 Strides: 2
Flatten layer				
	FC – 64: ‘relu’	FC – 1024:‘relu’		FC – 1024:‘relu’
		Dropout – 0.25		
FC – 3755: ‘softmax				

Table 3. CASIA-HWDB CNN Configurations

4.2 Evaluation

The model is evaluated by generating two line plots that show the model performance on both train and test datasets. One plot presents the accuracy and val_accuracy for the training process, and another plot shows the loss and val_loss for the training process. These plots help me to diagnose the learning behavior of the model during training and estimate the model performance. With the line plots, I can clearly determine whether a model is overfitting, underfitting, or has a good fit for the dataset. The evaluation function from Keras is also used to help to evaluate the model. This function evaluates the trained model using the testing images (test_X) and its corresponding labels(test_Y), and returns the loss value and metrics values for the model.

For MNIST and EMNIST datasets, the classification_report function from sklearn.metrics is used to present the result for every class in the dataset. This function returns the summary of recall, precision, F1-score, and support for each class. Recall is the ratio of correctly predicted positive images to the total predicted positive images, precision is the number of correctly classified images among that class, f1-score is the harmonic mean between precision and recall, and support is the number of images of the true response for each class. Overall accuracy also presents in the report. Here are the equations used to calculate results.

	Predicted Class		
		Class = Yes	Class = No
Actual Class	Class = Yes	True Positive (TP)	False Negative (FN)
	Class = No	False Positive (FP)	True Negative (TN)

$$Accuracy = TP + \frac{TN}{TP} + FP + FN + TN$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ score = \frac{2(Precision * Recall)}{Recall + Precision}$$

4.3 Prediction

Test datasets are predicted after the model is trained. The predict function is used to predict the test dataset. Predict function returns the prediction scores as a Numpy array, and the predicted values will be the index that has the largest number from the returned array. The model's accuracy rate is calculated with the test dataset by using predict function.

The model is also used to predict other handwritten images. The model is saved to the h5 format after trained, which is a single HDF5 file containing the model's architecture, weights values, and compile information. In the prediction files, the model in HDF5 file format is loaded first. The model is used to predict the handwritten images. Images are stored in the folder name "handwritten_test". These images are the images that I collected from different websites, from friends, or by writing directly from paint. Every image in the folder is pre-processed first before predicted. The pre-process process resizes the images to the same size as the ones in datasets, normalizes the images, and then converts the image color to gray-scale images. The predict function is used to predict the class for the images. This function returns the NumPy array of probability for each class, then the index that has the maximum probability is the prediction result. For EMNIST and CASIA-HWDB datasets, the characters are stored in another NumPy array that has the same length as the prediction array. The prediction result is the item from the character array at the index that has the maximum probability in the prediction array. The prediction results are printed at the button of the images for the MNIST and EMNIST dataset, but CASIA-HWDB predicts in a little bit different way. Since CASIA-HWDB stores Chinese characters, the model is used to predict Chinese characters, and python will not recognize the Chinese character. Thus, the characters need to be encoded into the "utf-8" type first in order to print the Chinese character. The prediction result for the Chinese character is shown in the image file name instead of print at the button of each image.

5. Result

5.1 MNIST CNN

Each model performance is shown in Table 4. All the models from the MNIST CNN reach high accuracy after only trained for 5 epochs. However, we can still notice a slight difference between the models. When there is only one pair of convolutional layer and max-pooling layer, when the kernel size or filters size decrease, the accuracy slightly falls. The model with an extra fully connected layer with 128 units also has a higher accuracy rate than the ones that do not, such as M5 compares to M4, M9 compares to M6, and M10 compares to M7. From Table 4, it is reasonable to conclude that kernel size, filters size, and fully connected layer are the factors that will affect the model performance.

Model	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Accuracy	0.9927	0.9896	0.9890	0.9910	0.9961	0.9934	0.9921	0.993	0.9955	0.9938
Val_accu	0.9869	0.9842	0.9835	0.9852	0.9857	0.9904	0.9878	0.9871	0.9887	0.9884
Loss	0.0233	0.0349	0.0355	0.0221	0.0123	0.0206	0.0257	0.0221	0.0136	0.0198
Val_loss	0.0416	0.0507	0.0608	0.0312	0.0514	0.0347	0.0389	0.0463	0.0407	0.0402

Table 4. MNIST CNN Model Results

For MNIST, I have decided to use M6 as the final model. M6 reaches the highest validation accuracy rate while the accuracy rate is also relatively high, and the loss value and validation loss value are relatively low compared to other models. Figure 2 shows the line plots of the classification accuracy and cross-entropy loss for both train and test datasets for M6. The model reaches 99.34% accuracy for training images and 99.04 for testing images.

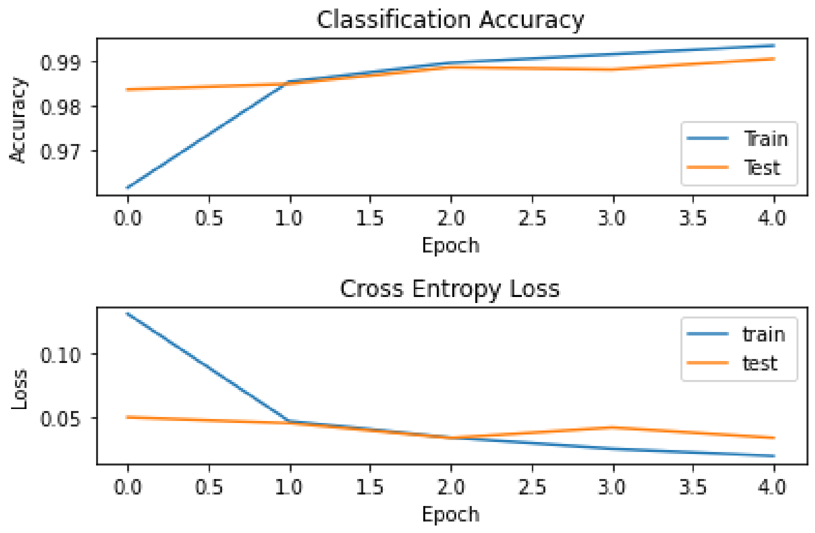

Figure 2. MNIST Final Result

After M6 is trained, the CNN error rate for the model in the test dataset is 1.18%, which means 98.82% of images from the test dataset have been successfully recognized. When predicting the test dataset with the trained model, 9882 images are correctly recognized out of a total of 10,000 images in the test dataset. From the classification report, every class has high precision and recall values, and the overall accuracy rate also reaches 99% in a 10,000 images dataset.

Twenty different handwritten digits are tested by using the trained M6 model. It turns out that all the digits are successfully predicted by the model. Figure 3 shows the prediction results for six different digits. The results are labeled as the text at the button of the images.

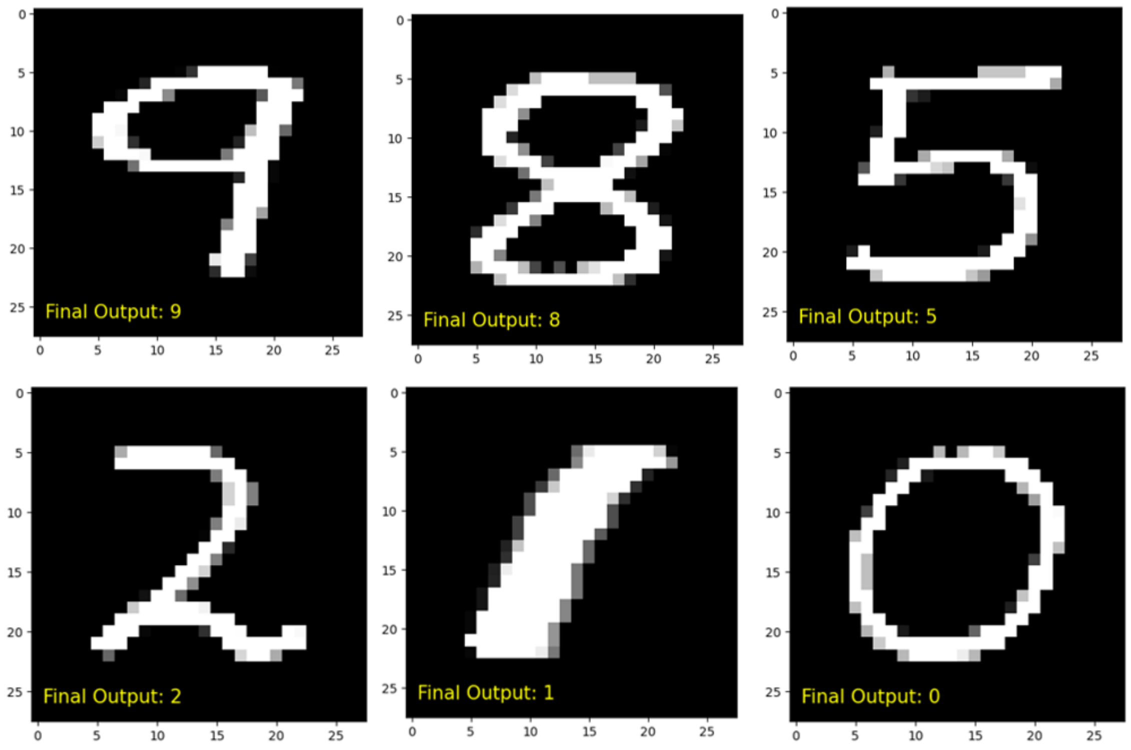

Figure 3. MNIST Prediction Result

5.1 EMNIST CNN

The following table, Table 5, shows the model performance for EMNIST CNN. Compared to MNIST, the overall model performance decreases because the EMNIST dataset is more complex than the MNIST dataset. However, it still reaches an average of 85% accuracy. Some of the models in EMNIST include the dropout layer. The purpose of the dropout layer is to prevent overfitting. However, adding a dropout layer in the model does not necessarily improve the model performance according to the result shown in Table 5. The number of fully connected layers in the model and unit for each fully connected layer affect the model performance. Three fully connected layers (regardless of the dropout layers) have a higher accuracy rate than two fully connected layers. However, if there are too many fully connected layers in the model, the performance starts to decrease. The model with different optimizers also results in different performance. M5, M6, and M7 have the same configuration but use different optimizers. M5, which uses Adam optimizers, reaches the highest accuracy rate and lowest loss value among the

Model	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Accuracy	0.8753	0.8582	0.868	0.8743	0.8607	0.819	0.8467	0.4859	0.8509	0.8409
Val_accu	0.8629	0.8626	0.8624	0.8619	0.8623	0.8519	0.8582	0.4946	0.8535	0.8459
Loss	0.3394	0.3914	0.3538	0.3311	0.3869	0.5224	0.4278	1.8356	0.4147	0.4541
Val_loss	0.379	0.3771	0.3822	0.3876	0.3791	0.4011	0.3771	1.7878	0.3919	0.432

Table 5. EMNIST CNN Model Results

three. Kernel size and filter size are the factors that affect the model performance in EMNIST CNN as well. From Table 5, it is reasonable to conclude that kernel size, filter size, fully connected layer, and optimizer functions are the factors that will affect the model performance.

For the EMNIST, M1 is used as the final model. M1 reaches the highest accuracy rate and the validation accuracy while the loss value and validation loss value are relatively low compared to other models. Figure 4 shows the line plots of the classification accuracy and cross-entropy loss for both train and test datasets for M1. The model reaches 87.53% accuracy for training images and 86.29% for testing images.

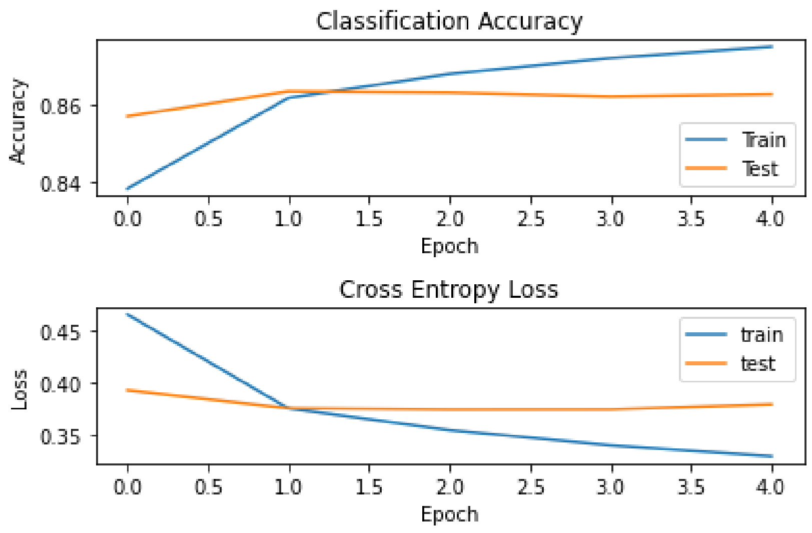

Figure 4. EMNIST Final Result

After M1 is trained, the CNN error rate for the model in the test dataset is 14.89%, which means 85.11% of images from the test dataset have been successfully recognized. When predicting the test dataset with the trained model, 99,003 images are correctly recognized out of a total of 116,323 images in the test dataset. From the classification report, some classes have high precision and recall values. However, some classes are confounding with other classes, which causes the precision and recall values to decline. For instance, handwritten characters for 0, o, and O look-alike a lot of time, which makes it harder for the model to classify between them. I and l, K and k, P and p, S and s, V and v, W and w, X and x, and Z and z are sets of examples that are easily be confounded while predicted as well. The overall accuracy rate for the model in the test dataset reaches 86%.

62 different English characters are tested by using the trained M1 model. Figure 5 shows the prediction results for 15 different characters. Some of the images are not successfully predicted, such as w, v, p, and s. The expected output for these images should be lower case characters, but the final output is upper case characters. Because the lower case and upper case for these characters have the same format, it is hard for the model to classify between them. Among 15 images, 11 of them are successfully predicted, which has a 73.3% accuracy. However, if the case is disregard, all 15 images can be considered correct prediction.

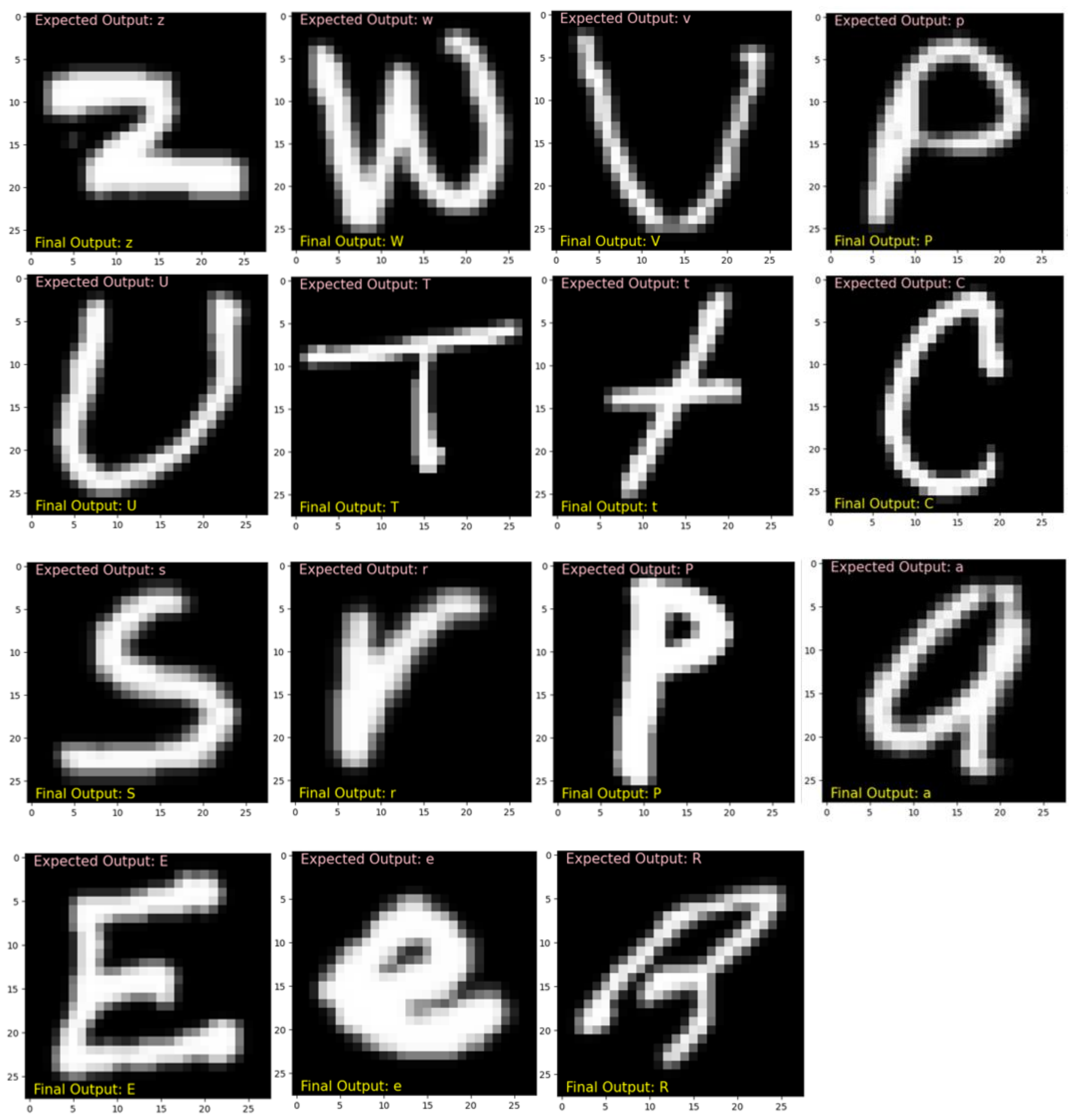

Figure 5. EMNIST Model Prediction Results

5.2 CASIA-HWDB CNN

CASIA-HWDB CNN is the hardest to train among all three different datasets. Table 6 presents the model performance for CASIA-HWDB CNN. Among the 10 models I trained, 6 of them diverge while training, as shown in Table 6 below. The models that were diverged were marked as "X" in the table below. As the result, it turns out that the model with the simplest architecture reaches the best performance after training for 150 epochs. The models with the max-pooling layer that has "same" padding have higher accuracy than the ones that do not. Padding in the max-pooling layer results in padding evenly to the left/right or up/down to the input such that output has the same height/width dimension as the input, which makes up for overlaps when the input size and kernel size do not perfectly fit. Besides padding in max-pooling layers, kernel size and filters size are also the factors that affect the model performance in CASIA-HWDB CNN. For this dataset, if we start the convolutional layer with a large filters size, the model diverges and cannot be trained successfully. M1, M4, and M10 all have a filters size of 64 in the first convolutional layer, and these models all diverge while training.

Model	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Accuracy	X	0.9573	0.868	X	0.9157	0.8824	X	X	X	X
Val_accu	X	0.4250	0.4143	X	0.5515	0.4812	X	X	X	X
Loss	X	0.1451	0.5838	X	0.3605	0.5186	X	X	X	X
Val_loss	X	5.491	4.7778	X	3.0109	3.9669	X	X	X	X

Table 6. CASIA-HWDB CNN Model Results

For CASIA-HWDB, M2 is used as the final model. M2 reaches the highest accuracy rate and validation accuracy rate with the lowest loss value compared to other models. Figure 6 shows the line plots of the classification accuracy and cross-entropy loss for both train and test datasets for M2. The model reaches 95.73% accuracy for training images, but it only reaches 42.5% for testing images. This happened due to the limited number of samples for each character in the test dataset. The test dataset has a total of 3755 characters in the dataset, and it does not have enough samples for each dataset.

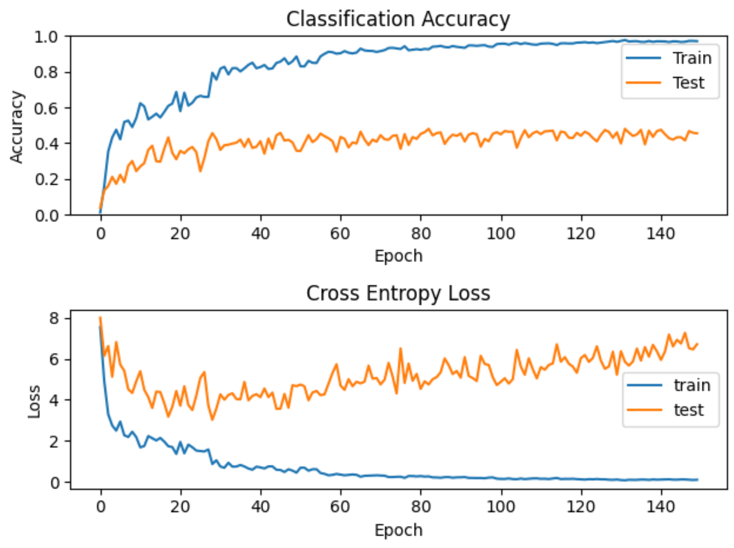

Figure 6. CASIA-HWDB Final Result

For the CASIA-HWDB dataset, since there are 3755 classes, I randomly select 100 characters to test by using the trained M2 model. Figure 7 presents the prediction results for 54 different characters. The prediction result reflects on the model performance. Compared to MNIST and EMNIST, there are more images that are not predicted correctly among the 100 predicted images. Among the 54 images shown in Figure 7, only 37 of them are predicted correctly, which only has a 68.5% accuracy. The file names that are highlighted in Figure 7 are the images that were not predicted correctly.

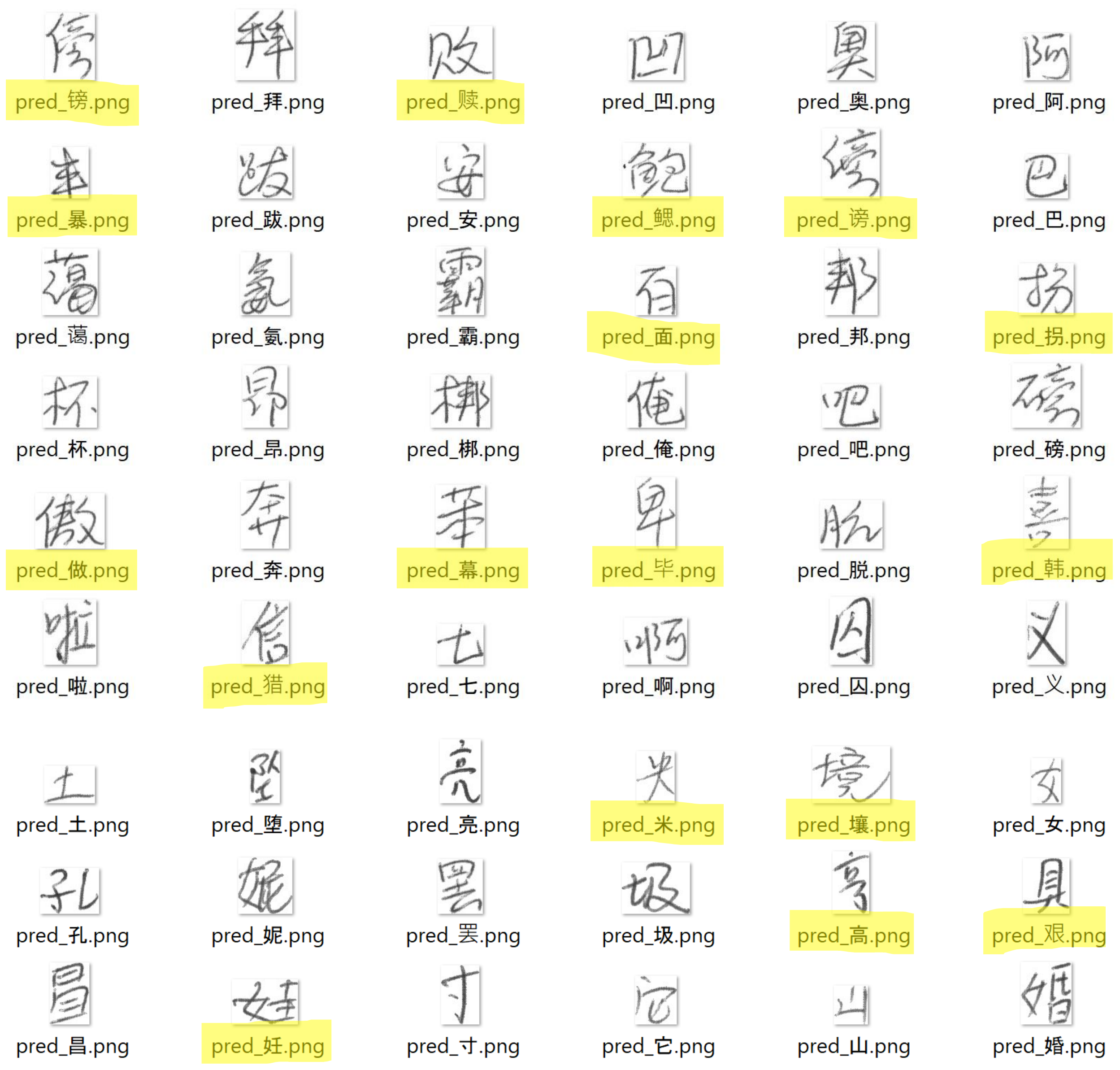

Figure 7. CASIA Model Prediction Results

6. Challenges

There are some challenges that I have faced when working on this project. The first challenge is when preparing the CASIA-HWDB dataset. This dataset is originally in a .alz file, and it needs to be converted into a .tfrecord file in order to use it to train the model. Since I have never heard of the .alz file before, I spend a lot of time doing research and trying to figure out how to convert the file to .tfrecord. Another challenge I have faced is when training the models. Some of the models diverge while training, which causes the accuracy rate to stay really low and not able to improve. For some models, the loss values are not successfully reduced while training. Models from CASIA-HWDB are a good example of this situation in that the loss values are high after the model is trained. Training the models is also time-consuming. For the CASIA-HWDB dataset, since the dataset includes a huge sample size and large classes, it takes longer to train the models. Every model from CASIA-HWDB takes over 12 hours to train.

7. Conclusion

To sum up everything that has been stated, the MNIST CNN model has the best performance and the highest prediction rate, while the CASIA-HWDB CNN model has the worst performance and the lowest prediction rate. Models' performance can be affected by a lot of factors, such as kernel size, number of fully connected layers, filters size, number of epochs trained, and optimization functions, etc. However, complex models do not mean better accuracy. Complex models might diverge while training as some examples shown in CASIA-HWDB CNN configuration. Sometimes, a simpler model can reach a better accuracy. In the future, I am planning to train the model for more epochs to determine whether the models' accuracy further improves, and I also want to use different types of neural networks to train the model and test to see if they will achieve a better accuracy.

8. Self-Reflection

Overall, I really enjoy working on this project. This is the first big project that I have done individually in college. It was very challenging not only because it is a personal project, but I was working on a topic that I do not have any experience in prior. However, I am glad that I have overcome all the challenges I faced and completed the project. I have always been interested in the field of Machine Learning. When I was thinking about what capstone project should I work on, I decided to do something related to Machine Learning right away even though I do not have any experience in it before, because I want to challenge myself and also want to gain more experience in Machine Learning field. The process of the project is not perfect, and I failed too many times while working on this project. In the beginning, I thought I would have plenty of time, so I did not start to work on this project soon enough. Gladly I was able to catch up on what I have left off and be back on track. There were also times that I could not figure out how to fix a bug that happened in my project, which I ended up asking people who have experience in the field. I also did not figure out how to implement a GUI for the recognized real-time handwritten characters, which is what I planned to do originally as an extra feature. However, I am still very satisfied with what I have done because I did take a huge step out of my comfort zone. I am planning to continue doing research in the field of Machine Learning in graduate school. In the end, I would like to thank Dr. Simone Ludwig for the guidance throughout my capstone project. Dr. Ludwig was always able to schedule a meeting with me to go through the process of my project and provide me the feedback on the project. I also want to thank the honors project for providing me the opportunity to work on the project with my interesting field.

References

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. doi: 10.1109/5.726791

Liu, C.-L., Yin, F., Wang, D.-H., & Wang, Q-F. (2011). CASIA Online and Offline Chinese Handwriting Databases. Retrieved from http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html

Mor, S.S., Solanki, S., Gupta, S., Dhingra, S., Jain, M., Saxena, R. (2019). Handwritten text recognition: With deep learning and Android. *International Journal of Engineering and Advanced Technology (IJEAT)*, 8(2S2), 172–178. Retrieved from https://www.ijeat.org/wp-content/uploads/papers/v8i2s2/ B10370182S219.pdf

Real-Time Recognition of Handwritten Chinese Characters Spanning a Large Inventory of 30,000 Characters - Apple. (2017, September). Retrieved from https://machinelearning.apple.com/2017/09/12/handwriting.html

Siddique, F., Sakib, S., & Siddique, M. A. B. (2019). Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers. *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*, 541–546. doi: 10.1109/icaee48663.2019.8975496

Yang, Q., Zhou, Z.-H., Gong, Z., Zhang, M.-L., & Huang, S.-J. (2019). *Advances in Knowledge Discovery and Data Mining: 23rd Pacific-Asia Conference, Pakdd 2019, Macau, China, April 14-17, 2019, Proceedings, Part I* (1st ed.), pp. 120

Yin, F., Wang, Q.-F., Zhang, X.-Y., & Liu, C.-L. (2013). ICDAR 2013 Chinese Handwriting Recognition Competition. *2013 12th International Conference on Document Analysis and Recognition, pp. 1464-1470*