**Final Project**

**Image Processing on Handwritten Alphabets and Numbers**

**Perat Damrongsiri**

University of Oregon

951805458

**Jay Shin**

University of Oregon

951563918

**Abstract**

Every person has a distinct handwriting style. Affected by the individual environment, handwriting can be nicely formed or poorly formed. Some of the math professors in the university occasionally have good numeric handwriting while maintaining bad alphabetic handwriting. In this project, the algorithm based on convolutional neural networks learns samples of numbers and characters from the EMNIST dataset. The model with up to 87.42% accuracy in train and validation set predicts the handwritten image (28 \* 28 \* 1) lecture sample from the math class.

1. **Introduction**

We first planned to build a Convolutional Recurrent Neural Network with Long Short Term Memory (a.k.a CRNN by LSTM), but then it was a huge challenge for us, we had to step down into just using Convolutional Neural Network. This project is mainly aimed at testing whether computers can easily distinguish what humans can barely read using probabilistic estimation. Currently Math 343 professor in the University of Oregon has incredibly distinct handwriting. Because he runs his lecture only by his handwriting, most of the students in the class have a hard time reading and understanding his lecture. Since his handwriting is uniquely indistinguishable, such letters and numbers as b and number 6, A and 4, w and a, and especially M, H and N are the hardest letters set in his lecture. Even taking his lecture for 10 weeks of the Spring term period, some of the words are still not understandable until students read the explanation from the textbook. We collected, not the most but, the most interesting samples of ten that the students questioned during his lecture. Due to the exceptional handwritten style, this project uses a massive dataset from EMNIST, so the algorithm learns a wide range of various handwritten characters and numbers to test out how far the computer can understand what humans cannot easily distinguish.

1. **Background**
2. Convolutional Neural Network

Convolutional Neural Network (CNN) is widely known as one of the best deep learning methods for image processing and computer vision. Distinct features of the CNN is that it uses kernel layers with several pooling mechanisms. CNN takes several kernel layers to catch relations and patterns for better efficiency, before CNN reaches fully connected neural network layers. Using several kernel layers, it is well known to have decreasing size of kernels, instead increasing the size, because the main reason for using the kernel layers is to reduce heavy calculation. Recent discovery of effectiveness of ReLU activation function which saves a lot of resources overall due to its simple structure applied on such complex architecture, this project only uses ReLU for the CNN, yet we are planning to use logistic activation or tangent activation functions for the further experiments.

1. Algorithms

Since our current working environment is on the Google Collab which has strict limitations on not only ram but also runtime, we need to save as much time as possible to run more experiments. Thus, we decide to use the Adaptive Movement (a.k.a Adam) optimizer which is known to be faster than Stochastic Gradient Descent. Overall code is focused on saving resources first because of limited usage of computation power. Therefore we use the Cross Entropy loss function due to its flexible behavior and our project is about probability estimation. The Cross Entropy function (In Keras, it is called as categorical cross entropy) provides bigger loss on poor prediction and smaller loss on good prediction, we could save a good amount of computational resources on the Google Collab.

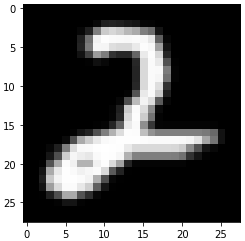
1. **Methods**

In this section we describe a general workflow to approach solving the problem with my key contributions.

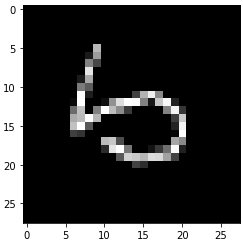
1. EMNIST dataset

Mathematic classes always use letters and numbers, so we obtain a large dataset of both numbers and letters from the EMNIST. Fortunately the Keras library already contains the EMNIST dataset. Thus it becomes much easier to import into our project with one line of code.

1. Math 343 Sample

Due to the limited time that I get provided, I have to pick the right testing sample from the class and preprocess into the EMNIST image format to increase the accuracy. The training and validation data set from the EMNIST looks like the picture provided below.

However, our sample from the lecture contains multiple letters with different sizes in randomly shaped. Thus I preprocess 10 samples from the math lecture turning the image into identical shape and reverse the color of the image to match with the EMNIST dataset (black letter in white background to white letter in black background).



1. Models

We prepare two models, one with two convolution layers that I created, and another with three convolutional layers that Poom created. We have tried to modify and increase complexity of the architecture, yet Google would not allow us unless we pay for the better version.

As it is mentioned in the background section, both of the models use ReLU function as activation function. To have the least calculation with the best accuracy and to have decreasing size of kernel layers, we use size of 5 for the first kernel layer and size of 3 for the second kernel layer. Since the max pooling method effectively increases accuracy, we add max pooling layers in between each kernel layer. Thus we could calculate the bigger size of hidden layers which results in higher accuracy in range of what Google provides us.

1. **Experiments**

In this section we describe how we use hill climbing search for tuning hyperparameters on both CNN models. Both models use The training dataset contains 558,346 handwritten alphabets and numbers while the validation dataset contains 139,586 handwritten alphabets and numbers. Lastly, the testing dataset contains 116,323 handwritten alphabets and numbers.

* 1. **2 Layers CNN**

First, I test out the dropout rate of 0.2 and 0.5, then the results show almost no difference (as what we have experimented on the project 4). Also, testing several batch sizes, higher the batch size inserted higher accuracy has been obtained. However, because the large batch size ended up taking too much time and RAM, to maximize my experiment, we decided to maintain a 0.5 dropout rate and reasonable batch size, 256, for the base of hyperparameter tuning.

Figure 1.a shows the difference between two different learning rates of 0.001 and 0.0001 with the first hidden layer size of 30, the second hidden layer size of 15, the first kernel size of 5, and the second kernel size of 3. The result shows that 0.001 learning rate outperformed 0.0001 learning rate in both accuracy and loss. Learning rate of 0.001 ended up with 86.48% accuracy, meanwhile that of 0.0001 resulted in 85.12% accuracy.

With the evidence from figure 1.a, I doubled up the first and second hidden layers and tested out 0.001 and 0.0001 learning rate to make sure the right learning rate to use. Overall graph of figure 1.b shapes similar to figure 1.a, yet doubling up the sizes of hidden layers performed slightly better than figure 1.a, an increased accuracy of 0.5%. The learning rate of 0.001 ended up with 86.95% accuracy, and that of 0.0001 resulted in 85.96% accuracy. Therefore I decided to maintain a learning rate of 0.001 for the further experiments.

Maintaining the learning rate of 0.001, I tested about doubled hidden size of figure 1.b and once more doubled hidden size for efficient comparison. The first hidden size of 128 with the second hidden size of 64 resulted in 87.22% accuracy which is about 0.3% increased accuracy than figure 1.b. Then, the first hidden size of 256 with the second hidden size of 128 resulted in 87.42% accuracy. As we can see on the figure 1.c, the gap between two graphs is tighter than previous experiments. Then, I tried a slightly bigger size of everything, yet the result had almost no difference. The first hidden layer size of 512 with the second hidden layer size of 256 enhances accuracy only by 0.2%. Therefore I decided to stop at the dropout rate of 0.5, the first hidden layer size of 256, the second hidden layer size of 128, the first kernel size of 5, and the second kernel size of 3 with 0.001 learning rate which gives me the highest accuracy of 87.42% due to save resources for more experiment.

The last experiment for model 1 is quite interesting. Figure 1.d shows that the orange line uses the categorical cross entropy which takes multi classes (in our project case, we already have numbers and alphabets), and the blue line uses the binary cross entropy which takes two classes (mainly for classifying 0 or 1). Since the loss graph contains a binary cross entropy function, blue loss shows a different loss result than any figures provided due to its incomplete calculation behavior. Although the binary cross entropy is not designed for cases like our project, the curve for the accuracy is smoother than and resulting in as high accuracy as categorical cross entropy. Testing the binary cross entropy model with our test sample, we obtained the result same as the result from the categorical cross entropy.

Using ten test samples from the math lecture, we obtained 1 correct prediction and failed others, which is about 10% accuracy from the test dataset.

* 1. **3 Layers CNN**

Having three kernel layers for the CNN architecture, Perat started with a fixed batch size of 128 due to the computing time limitation. Figure 2.a shows how he compared the hidden layer sizes ( [ 1: dropout : 0.2, hidden 1 - 3 : 75, kernel 1 : 5, kernel 2 - 3 : 3, learning rate : 0.0001 ] , [ 2: dropout : 0.2, hidden 1 - 3 : 90, kernel 1 : 5, kernel 2 - 3 : 3, learning rate : 0.0001 ] , [ 3: dropout : 0.2, hidden 1 - 3 : 100, kernel 1 : 5, kernel 2 - 3 : 3, learning rate : 0.0001 ] ). The graphs seem like a rainbow, such that each line never crosses over and directly shows that bigger is better. Perat picked the highest of 86% ( dropout : 0.2, hidden 1 - 3 : 100, kernel 1 : 5, kernel 2 - 3 : 3, learning rate : 0.000, loss : 'categorical\_crossentropy').

Before testing out the larger hidden layer size, Perat tested out different kernel sizes shown on figure 2.b containing ( [ 1: dropout : 0.2, hidden 1 - 3 : 75, kernel 1 - 3 : 3, learning rate : 0.0001 ] , [ 2: dropout : 0.2, hidden 1 - 3 : 90, kernel 1 : 5, kernel 2 - 3 : 3, learning rate : 0.0001 ] , [ 3: dropout : 0.2, hidden 1 - 3 : 100, kernel 1 - 2 : 5, kernel 3 : 3, learning rate : 0.0001 ] ). Interestingly, the rainbow shape of the graph is more distinct than figure 2.a. It is obvious that we must try a bigger kernel size later for further experimentation. Then, Perat picked the best result of 86.63% accuracy with 35.97% loss.

One thing that I learnt from project 4, the learning rate affects the most on the graph, and figure 2.c shows . Perat tested out ( [ 1: dropout : 0.2, hidden 1 - 3 : 100, kernel 1 - 2 : 5, kernel 3 : 3, learning rate : 0.001 ] , [ 2: dropout : 0.2, hidden 1 - 3 : 100, kernel 1 - 2 : 5, kernel 3 : 3, learning rate : 0.0001 ] ), then the graphs provides that learning rate of 0.001 shows better efficiency on finding highest accuracy of 86.71%. However, overall the shape of the graph is better with a learning rate of 0.0001 with highest accuracy of 86.78%, so we would like to implement a switching algorithm between 0.001 and 0.0001 learning rate expecting better results for the further experimentation. Using ( dropout : 0.2, hidden 1 - 3 : 100, kernel 1 : 5 - 2, Kernel 3 : 3, learning rate : 0.001, loss : 'categorical\_crossentropy' ), figure 2.d introduces us to overfitting.

The figure 3.d is the interesting one that overfitting occurs. Increasing the overall hidden sizes significant enough ( dropout : 0.2, hidden 1 : 2556, hidden 2 : 128, hidden 3 : 3 : 100, kernel 1 : 5 - 2, Kernel 3 : 3, learning rate : 0.001, loss : 'categorical\_crossentropy' ), we found out the proper point that the hidden sizes and the learning rate is large enough the graph is increasing. We added this fact to improve the learning scheduler that mentioned above (in between 0.001 and 0.0001) for further research and implementation.

1. **Conclusion**

The best score from model 1 is higher than the best score from model 2. However, when we are testing our test dataset with model 2, it predicts with accuracy of 20%.

Using 10 samples from a math lecture, we ended up having a maximum 2 out of 10 samples correctly classified. Model 1 with the highest accuracy of 87.42% from the validation dataset (dropout : 0.5, hidden 1 : 256, hidden 2 : 128, kernel 1 : 5, kernel 2 : 3, learning rate : 0.001, batch size : 256, loss : categorical cross entropy) missed test case “M” predicting “W” instead. Meanwhile, model 2 with the highest accuracy of 86.78% from the validation dataset (dropout : 0.2, hidden 1 : 100, hidden 2 : 100, hidden 3 : 100, kernel 1 - 2 : 5, kernel 3 : 3, learning rate : 0.0001, batch size : 128, loss : categorical cross entropy) correctly predict “M” on “M”.

The result supports that adding extra CNN kernel layers can detect more details and solve problems better. Model 1 missed the “M” test case predicting as “W”, and we understood that it was still a good guess. Then, model 2 predicts right for the same test data “M”, and we are assuming that the extra layer catches another relation so it understands the professor’s “M”. Based on the current result, we conclude that we can improve the program even better by adding more layers and increasing size, and the math professor's handwriting is exceptional data. The handwriting of the math professor is hard to read for both machines and humans, I guess even the computer confuses what humans hardly understand. Currently, our program has only three kernel layers and does not contain regularization. Also it has not been tested with various functions for activation, loss and et cetera. To improve the prediction, we are going to add more kernel layers testing out with different sizes as well. Since the result provides direct success, we are expecting significant improvement for the further experiments. Also, we decide to add L2 regularization, perhaps implement Leaky ReLU activation function and several other functions which we are expecting to have higher accuracy with. Especially implementing the learning scheduler would lead us to higher accuracy.

1. **References**
2. Brownlee, Jason. “Code Adam Optimization Algorithm From Scratch.” *Machine Learning Mastery*, 20 Feb. 2021, machinelearningmastery.com/adam-optimization-from-scratch/.
3. Brownlee, Jason. “Handwritten Digit Recognition Using Convolutional Neural Networks in Python with Keras.” *Machine Learning Mastery*, 26 Aug. 2020, machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/.
4. 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>.