COMP 562 Final Project

Andrew Lauer, Srikar Pasumarthy, Thomas Drake

1. Introduction

The objective of this project is to train a model that can translate handwritten words into recognizable typed text. To structure this, we divided our model into three different phases: character segmentation, character recognition, and word recognition.

2. Preprocessing

a. Character Segmentation

The first step was to determine how to segment an image into individual characters so that they can be processed by our trained neural network. While this wasn't the primary focus of our project, it turned out to be very difficult to successfully accomplish. Because our main focus was the training of the neural network, we decided to implement a simple, naive solution that can separate non-cursive characters in an image with relative accuracy. This solution was developed by researching different methods that other people have used for segmentation, and testing out different strategies with the type of data we wanted to use. The ultimate solution essentially uses the OpenCV python library to find contours in the image and put bounding boxes around these contours which will become our character segments (Figure 2.1).

While this solution works well on most images, its performance seems to vary greatly based on the resolution of the image; for example, higher resolution images that were taken on an iPhone did not perform well. Resizing the images helped this issue, but we were never able to achieve great accuracy on large, variable samples of data which led to error when testing our final implementation on words. In fact, we believe most of the errors we saw when testing our full project on words could be attributed to errors in the segmentation step. So, if we wanted to improve our project in some future iteration, the first place we would look to modify would be this preprocessing step. Perhaps a separate machine learning model could be trained to predict where lines-words-characters are on a page of writing, as opposed to simply using the contours on the image. However, since the main focus of our project was on the neural network to predict letters, we felt that this simple solution was adequate as a proof of concept that our preprocessing, prediction, post-processing pipeline can be used to decipher handwritten text.

b. Efforts for Line and Word Segmentation

Once we had a working solution for character segmentation, our next idea was to see if this could be applied to a full paragraph of text. When we ran our character segmentation code on a paragraph, we saw that it was able to segment the letters relatively well (Figure 2.2). However, this could not be directly applied to our system without first solving two problems. First, simply doing character segmentation doesn't allow us to know the order of the letters that we segment; if all characters are in one line, we can simply sort by x-coordinate, so this problem can be solved if we use segment by line first. Second, with

just character segmentation, we can only generate a stream of letters without word breaks; so we would need to address this by either adding word breaks in post-processing, or by also segmenting by words.

Both of these problems could be solved by segmenting by line and word in addition to character. We tried to do this, but our simple solutions using the OpenCV library had widely varying success based on image size and resolution. So we decided to focus on testing our scheme on single words, so only character segmentation was necessary. But this is another area where future iterations could improve on this and expand the utility of our project.

Figure 2.1

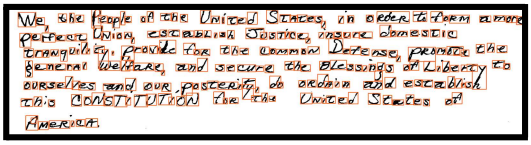


Figure 2.2

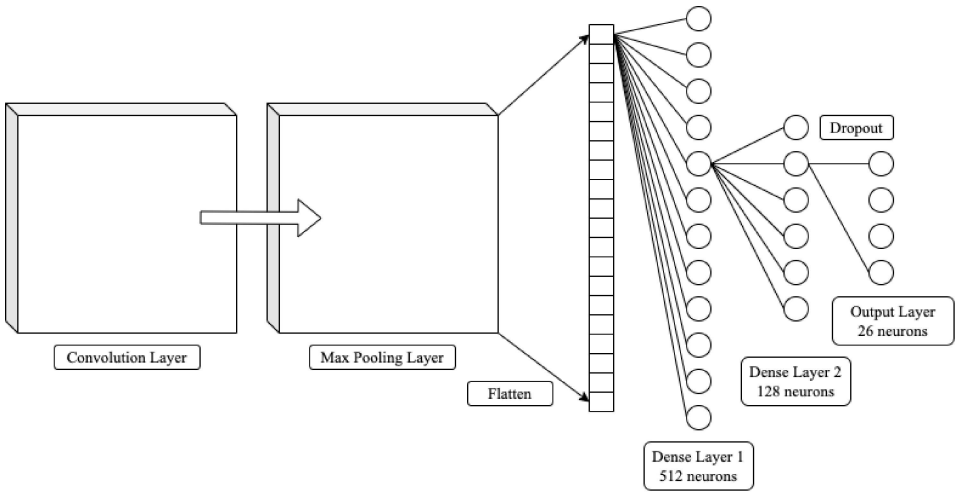
c. Data Generation

Since we couldn't find a dataset of handwritten words that was consistently non-cursive, we decided to generate our own by combining letters from the MNIST dataset. We randomly chose words from a large dictionary and concatenated random MNIST letters together to form a reasonable dataset of handwritten words. This was used to test our entire preprocessing, prediction, and post-processing pipeline. An example of one of these words is in Figure 2.1.

3. Neural Network

a. Structure

To classify the digits, we trained a convolutional neural network diagrammed below. The neural network was implemented using TensorFlow Keras.



The first layer in the network is a convolutional layer. It takes in an input image with an input shape (28, 28, 1) representing one character from the full word image. Since the preprocessing step scales

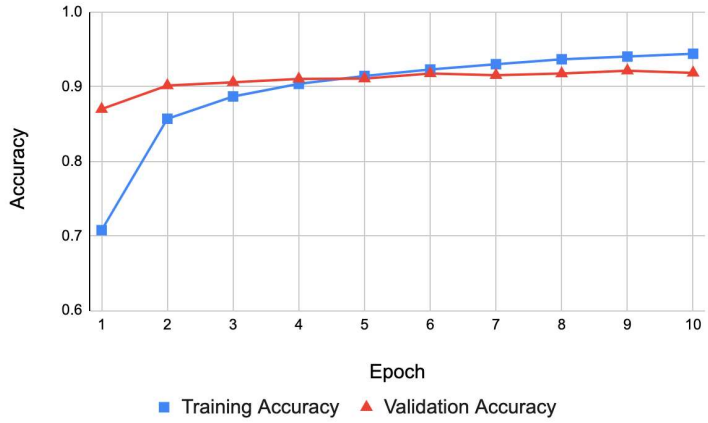
the characters to 28 by 28 pixels, we can use a constant shape for the input. The convolutional layer utilizes 32 3 by 3 filters.

After a max pooling layer is applied, the 28 by 28 2D image is reshaped into a 784 1D vector with a flattening layer. This allows the 1D vector to be passed through two fully connected layers with 512 and 128 neurons respectively. Each of the layers uses a ReLU activation function. The next hidden layer is a dropout layer with a frequency of 0.5. By dropping some neurons, we were able to avoid significant amounts of overfitting in the model.

The final layer of the network is an additional dense layer with 26 neurons, representing the 26 output classes specified by the 'EMNIST letters only' dataset. This final output layer uses a ReLU activation function as well.

b. Training

The neural net was trained on the EMNIST letters dataset, containing 103,598 28 by 28 pixel images of handwritten upper and lower case letters. The test set of the data contains 88,799 samples while the test set contains 14,799 samples. The test data and labels were fed into the neural network and used a validation split of 0.2. Below is a chart of the model's accuracy throughout the training process.



When tested on the test data set, our neural network achieved accuracy of 0.906.

4. Post Processing

a. Recognition of Words

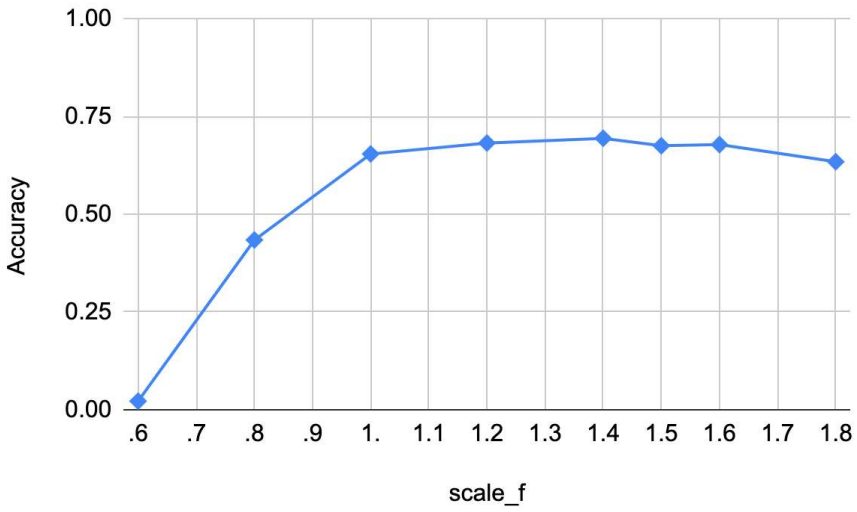
The objective of the post-processing is to identify the most appropriate word within two edit distances of the sequence of characters. Here, an edit distance of one is defined by inserting, deleting, or replacing a character or swapping two characters in a string. Similarly an edit distance of two is defined as the combination of two edit distances of one. In order to ensure that strings produced by these functions are valid words, we define a set to contain strings that are contained within our predefined dictionary–which was created through identifying all unique words in the Sherlock Holmes books. Using this information, we finally characterize the string by checking if it can be replaced by a word with an edit distance of zero, one, or two.

b. Drawbacks

The use of edit distance does not take into account context. In other words, the string “three, too, one, go!” would not be changed given our current post-processing model; when in reality, the string should be redefined to be “three, two, one, go!”. This problem can be solved using a form of natural language processing that incorporates Bayesian updating.

5. Results

With our completed pipeline of character segmentation, character classification, and word recognition, we could not test the pipeline on the words we generated from the MNIST dataset. As part of the preprocessing steps, we multiply the images by a scale factor before running our character segmentation process. This had a major impact on the character segmentation, and in turn an impact on the accuracy of the total output. Below are some values of the scale factor and its impact on the pipeline's accuracy.



The scale factor that resulted in the highest accuracy was 1.4. With this factor, our entire pipeline was able to correctly identify a hand written word with accuracy of 0.694.

6. Conclusion

In this project, we constructed a full pipeline to classify handwritten words with an accuracy of 0.694. The pipeline identifies the location of each character, classifies each character, then corrects any minor spelling errors. The character classification is done with a convolutional neural network trained on the EMNIST letters dataset and achieves accuracy of 0.906 for all 26 classes.

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

Norvig, P. (2016). *How to Write a Spelling Corrector* . Natural language corpus data: Beautiful data. Retrieved from http://norvig.com/ngrams/

Rosebrock, Adrian. (2020). OCR: Handwriting recognition with OpenCV, Keras, and TensorFlow. Retrieved from https://pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/