Handwriting Recognition using Optical Character Recognition(OCR)

Keller Lawson
Computer Science
Python for Machine Learning
CS 477 – Spring 2021

Handwriting Recognition using Optical Character Recognition(OCR)

Keller Lawson
Computer Science
University of Idaho
Couer d'Alene, ID
laws1689@vandals.uidaho.edu

***Abstract*— This project aimed to create and train a deep learning model that could classify individual characters of the English Alphabet as well as the digits 0 through 9. To achieve this, two datasets were combined and used to train the model for all characters simultaneously.**

Keywords*— *handwriting recognition, deep learning, neural network, Keras, TensorFlow

I. Introduction

Even with the advancements of technology related to note taking, there are many people who prefer to use pen and paper to take notes. However, when it comes to needing access to these notes, physical notes can be harder to access due to multiple factors. Firstly, physical notes tend to get damaged after being handled many times. Second, you will not always have access to the physical copies of these notes when they are needed. Lastly, sharing physical notes with others is difficult without lending the notes out to be copied.

This project seeks to fix all these issues by beginning the process towards digitalizing all your handwritten notes by classifying the characters on paper. This step in the process was aimed at creating a machine learning model that would be able to recognize individual characters with high accuracy and precision. After the model has been trained and tested to verify that we have achieved the goal of identifying individual characters, the model will be used to identify multiple words in a picture fed to it.

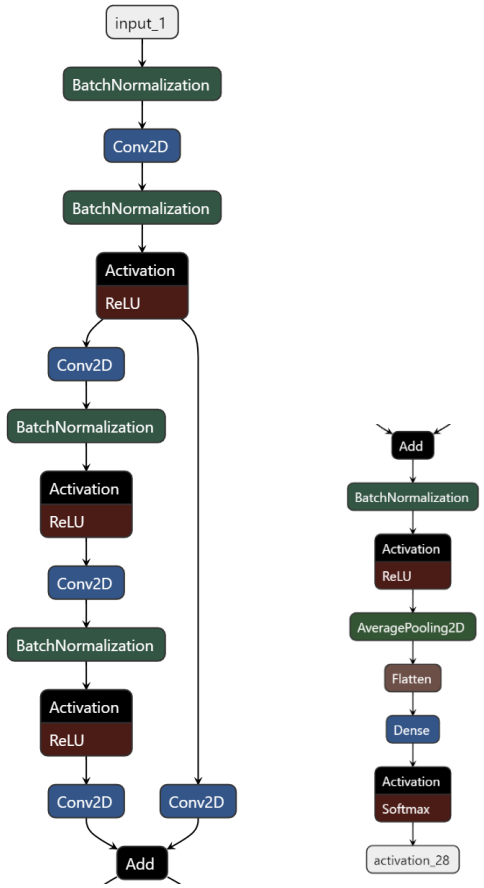
The data sources used can be found at the following links:

- https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format
- https://www.kaggle.com/oddrationale/mnist-in-csv/

II. Method

A. Model Architecture

To create the model, I chose to use TensorFlow and Keras to create a ResNet deep learning neural network model. This model design was sourced from another project on GitHub [1] and had achieved excellent results after training and testing. Using Netron.app, I loaded my model (saved in h5 format) to visualize the architecture. In (1), we see the general composition of our layers. There are 8 layers with the same architecture, but I have only shown the beginning and end of the architecture due to size constraints.



(1)

B. Parameter Settings

To find the final parameter settings for training the model, there was a lot of trail and error that occurred. To train the model, I ultimately settled on a batch size of 128, 50 epochs, and an initial learning rate of 0.1. For the first training attempt, a batch size of 128, 20 epochs and initial learning rate of 0.0001. These parameters were based off of examples we had used in class and yielded successful results. However, the training results were very poor results with my accuracy values never rising above 43%. This was due mainly to the initial learning rate being too small, so the model was not learning quick enough for the number of epochs we were using. Without letting the model train for a significant amount of time, we would never receive good results from these parameters. The

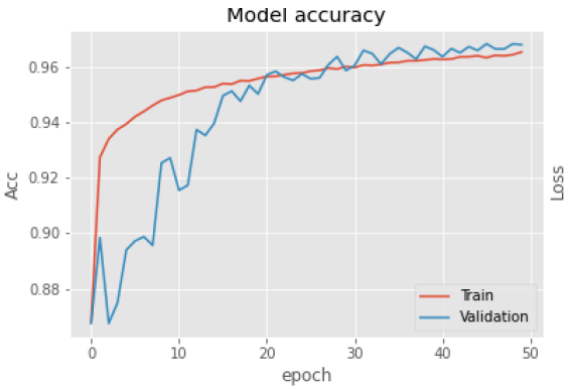
parameters we ended up selecting to train the final model resulted in excellent testing results seen below.

III. Experimental Results

A. Model Training

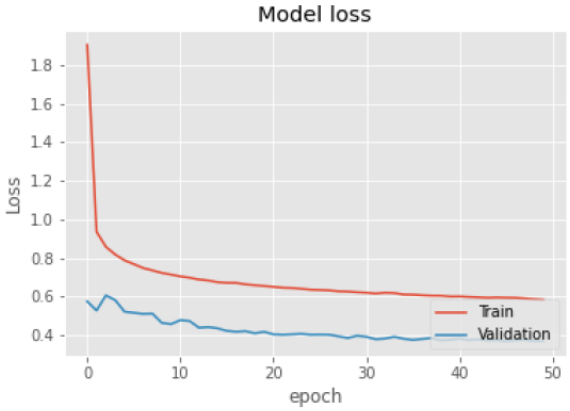
Training the model, after the many test runs, yielded excellent performance results. Figure (2) displays the macro and weighted average of the accuracy for the model after training. These results can be furthered explored in (3) and (4) where the graphs show the results achieved per epoch during training.

accuracy			
macro avg	0.96	0.96	(2)
weighted avg	0.97	0.97	



(3)

In the first few epochs of training, the model accuracy quickly increases as it begins to learn about the characters. Around the 9th epoch, the rate of increase for the model accuracy begins to move towards a plateau as the learning rate decreases due to the model learning each character better. The relational inverse of this graph and increase in model accuracy can be seen in (4) which plots the model loss over time.



(4)

These plots being near inverses of each other is a good result that shows as we are training the model more and achieved a higher accuracy, there less loss each epoch.

B. Model Testing

After the model was trained, saved and the training results were calculated and plotted, it was time to test the model. To test the model, we split the original datasets in 2 so that the original dataset was broken into 80% training data and 20% testing data. This would allow us to compare prediction results against their labels for verification.

To illustrate our test results, I imported a package named imutils that allowed me to create a montage of many images. Combining this with the package cv2, we were able to label each image during the for loop with the correct label for the character being predicted. If the model predicted it right, the letter would be green in color, if not then it would be red. For demonstration purposes, I chose a square montage that was 7 images tall and 7 images wide for a total of 49 images per test. Figure(s) (5) and (6) show the output of the montages after the predictions.

In (6), it is unable to correctly identify 2 characters. The top left character that was incorrect it matched the character to an ‘S’ which from the character’s label is incorrect. However, looking at the picture, it is hard for me to determine the character was well. As a guess, I would think it might be a ‘5’. The lower character it missed seemed to be the most common miss. This is the difference between the digit ‘0’ and the letter ‘O’. These two characters are very similar to each other depending on a person’s handwriting making it difficult for the model to predict the difference between them. Running the tests and building montages, the application of the model seemed to generally follow the results of our training accuracy.

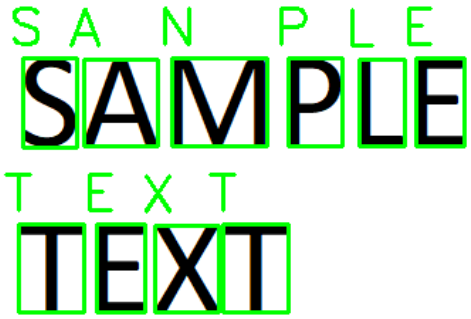
(5)

(6)

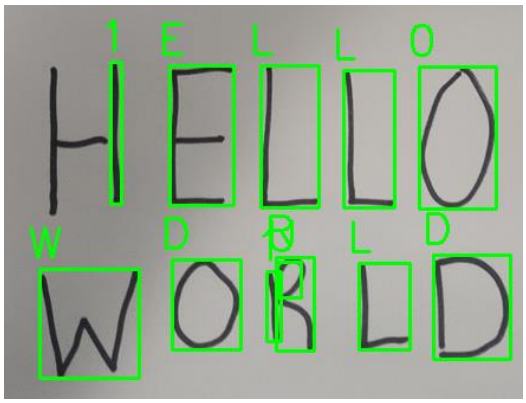
C. *Image Recognition*

After I was satisfied with the training and testing of the model, I moved on to testing the model on actual images of handwriting. I used 2 different images during the tests. The first, (7), was a simple image containing the words “SAMPLE TEXT” typed out. The second image, (8), is my handwriting sample of the tradition programming phrase “HELLO WORLD”.

In (7), the model was able to read and correctly predict 9 of the 10 characters. It mistook the letter ‘M’ for the letter ‘N’. In (8), there were a lot more mistakes with the predictions. Due to my poor handwriting, combined with error possibility in the model, only 7 out of the 11 characters were correctly predicted. The first character it missed was the ‘H’. This was likely due to my handwriting leaving a break in the connecting bar so there was a vertical line by itself. The other obvious misprediction is the letter ‘R’. It determined that there was 3 characters in the spot where ‘R’ was written. The predictions show ‘1’, ‘U’, and ‘R’ were all present.



(7)



(8)

IV. Conclusion

The final outcome of this project turned out much better than I expected. Being able to feed the model an image containing handwritten text and display the predicted characters can lead to many simple programs to help automate daily tasks. One of the biggest issues I ran into while working on this project was the time requirement. When I first started to train my model, the predicted time was around 50 hours or over 2 days. I did not have that amount of time to wait while the training ran so I research alternate training methods and was able to reinstall TensorFlow to use my graphical processing unit (GPU) for training. This cut my training time down to about 2 hours which was manageable. To increase accuracy of the model, I would want to change the training parameters for the model to allow for a slower learn rate and larger number of samples.

Additionally, after refining the model for uppercase English letters and digits, I would want to make the model even more complex by adding an additional training set for lowercase English letters since most people do not write in uppercase block letters all the time.

V. References

[1] A. Rosebrock, "jrosebr1 - Overview", GitHub, 2021. [Online]. Available: https://github.com/jrosebr1. [Accessed: 28- Apr- 2021].

[2] "Deep Learning Project - Handwritten Digit Recognition using Python - DataFlair", DataFlair, 2021. [Online]. Available: https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/. [Accessed: 27- Apr- 2021].

[3] "Build a Handwritten Text Recognition System using TensorFlow", Medium, 2021. [Online]. Available: https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5. [Accessed: 27- Apr- 2021].

[4] "OpenCV: Image Processing", Docs.opencv.org, 2021. [Online]. Available: https://docs.opencv.org/master/d7/dbd/group__imgproc.html. [Accessed: 31- Apr- 2021].