COMP343 - Final Project - OCR

Carl Gross

July 20, 2022

1 The Problem

1.1 Background

Optical Character Recognizers (OCRs) are a common neural network problem. The pandemic created a demand for mobile document scanners. While these scanners can take and clarify a picture of the document, one still needs to manually translate actual text on the document. Apple has recently released an OCR as part of their camera app in the most recent iOS. With our world becoming more and more online, the ability to translate written documents to digital ones is becoming more and more important. Google has taken image classification even further and implemented a neural network that can classify almost any image through their search engine. Although an OCR only has to worry about classifying certain characters, it has its own obstacles. OCRs demand a much higher level of precision that general image classifiers because typos can drastically vary the tone and meaning of a message.

1.2 Goal

The goal of this project is to train a model that will successfully be able to identify a handwritten lowercase letter from a picture. With our developed convolutional neural network model, I can parse a scanned written document and identify each written letter, then I can rebuild the written document as a digital document.

2 Methods

2.1 Dataset

This dataset was created by Rob Kassel at MIT Spoken Language Systems Group. He collected a large sample of pictures of handwritten words. Took pictures of these pictures and divided them into pictures of individual handwritten letters. Some of the words were capitalized, so he excluded the uppercase letters. Then he selected a subset of the images that were more clear, and rasterized and normalized the selected images. The data included a feature for each pixel of the rasterized image as well as a letter classification for the image. (The dataset also includes a next_id feature which indicates that next letter in the sentence, but that is not used and is addressed in **Future Work** below)

2.2 Choosing an Architecture

At the start of this project, I considered multiple neural network structures. Initially, I decided to use an LSTM structure because I had worked with them before, and I was planning on utilizing the previously seen characters in my prediction. LSTM's are good for time series data, so it would be effective in remembering and utilizing previously classified seen letters. However, when I learned of convolutional neural networks, I found that that structure might be more effective, simply because they excel at image classification. I also noted that a standard multilayered perceptron could also be effective due to the fact that the data was already preprocessed into a single array of 0s and 1s (no rgb values). Because I had already explored them extensively in class and there are multiple OCR perceptrons available publicly online, I decided against using a perceptron model, so I implemented a convolutional neural network. As a result of this choice, I decided to drop the usage of the next_id feature, in favor or purely classifying the images based off of the pixel values. Because I am classifying pixel layouts of preprocessed images, this structure seemed like it would be effective.

2.3 Various Tests

I decided to vary the activation function to see if that would make an impact on the accuracy of my model. I tried relu, sigmoid, and tanh activation functions. I also decided to vary the loss function to see if that would make an impact on the accuracy of my model. I tried mean squared error and sparse categorical cross entropy loss functions. I also tried adding additional convolution layers as well as pooling layers.
** The results of these tests are in **Results** **

3 Convolutional Neural Networks

Convolutional neural networks are effective for image classifiers because they analyze nearby pixels as a group (kernel). Often the surrounding pixels will carry information that are useful to classifying an image. They usually consist of convolutional layers, pooling layers, and dense layers.

3.1 Convolutional Layers

References:
`https://www.deeplearningbook.org/contents/convnets.html`

`https://www.youtube.com/watch?v=Lakz2MoHy6o&ab_channel=TheIndependentCode`

Convolutional layers take a grid-like set of points or values, usually representing pixels. It then defines two matrices (of a chosen size) as the kernal and the bias matrix. Then a sub-matrix of the input with the same size as the kernal is selected. This sub-matrix is then convoluted with the kernel. This operation consists of first, rotating the kernal 180 degrees, then taking the product of each element in the kernal with its corresponding element in the sub-matrix, and finally summing these products and adding the corresponding value of the designated bias matrix. This is done with all sub-matrices of the input to calculate each value of the output matrix. The dimensions of the output matrix are inherent smaller than the input matrix because the convolution operation compresses multiple pixel values into one.

Once all the output as been calculated, the gradient of the output matrix is calculated (and it turns out to be equivalent to the gradient of the bias). Then the convolution operation is performed on the gradient matrix and the kernal to update the kernal values. The mathematical equations can be found in the reference link above.

3.2 Pooling Layers

Pooling is the idea of compressing the values of a feature map by grouping the pixels by their location, they assigning each group one total value based on the pixels in that group. Two of the most popular methods of assigning this value is by taking an average of all the pixel values (average pooling) or taking the maximum value of all the pixel values (max pooling). This method helps smooth out edges and points in a feature map, sometimes allowing for increased accuracy. Of course, this could also be detrimental to your accuracy if you lose significant data by combining the pixels. I decided to test models which include a max pooling filter as well as a model which does not include that filter. (described later in **Results**)

3.3 Dense Layer

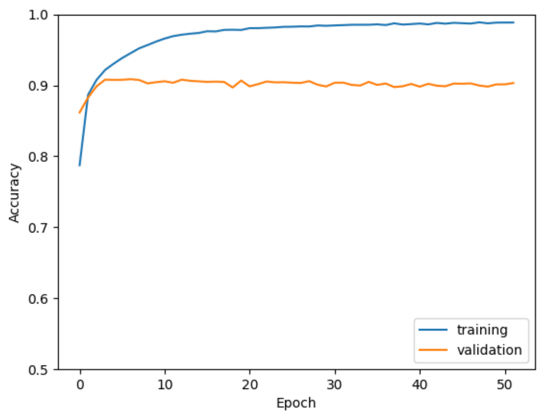
After filtering the data through multiple convolution layers as well as max pooling layers, a dense layer is used to compress the final output into a set of indices. With my dataset, these indices represented lowercase alphabet letters: 26 different indexes 0-25 (each representing a letter). This could then be used to compare with the actual letter classification.

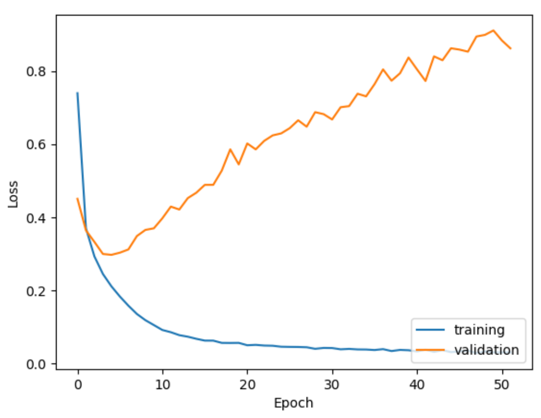
3.4 Callback Early Stopping

Initially the training of the neural network was taking an extremely long time, especially when run with for a large amount of iterations. Therefore, I took advantage of a Tensorflow function that tracks whether there is improvement in the loss and if not, it stops the training of the model early before finishing all the iterations. I decided to set it so that it would stop training if the loss had not improved for 4 consecutive iterations. I found that surprisingly, this early termination did not occur very often. Even with high number of epochs (50 or 100), the loss would often continue to decrease until the very last iteration.

4 Results

Pooling: None; Number of Convolution Layers: 2
Activation Function: Relu; Loss Function: sparse categorical cross entropy:



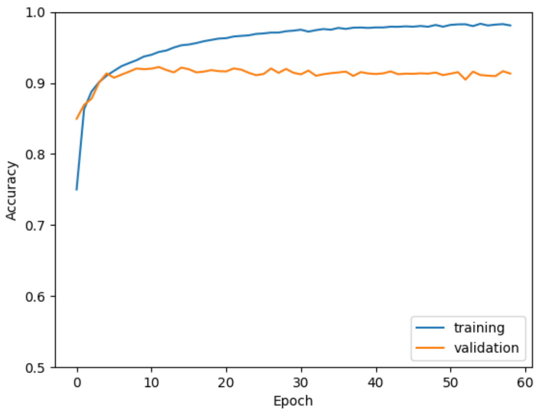


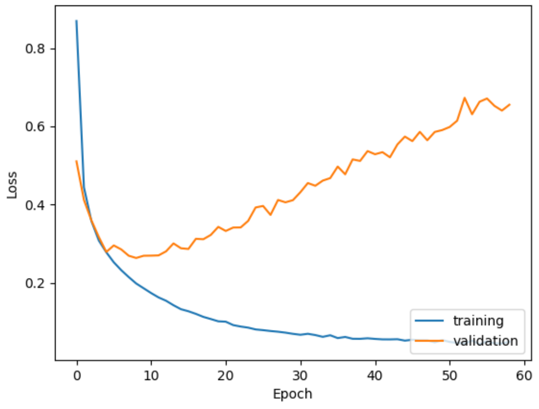
Test Loss: 0.8616726994514465 ; Test Accuracy: 0.9031639695167542

The accuracy seems to plateau rapidly around 15-20 epochs, while the loss continually declines until the 50th epoch, at which it triggers the early stoppage callback. The total test accuracy is 0.9033 which isn't too bad considering the simplistic convolution structure that I am using.

4.1 Pooling

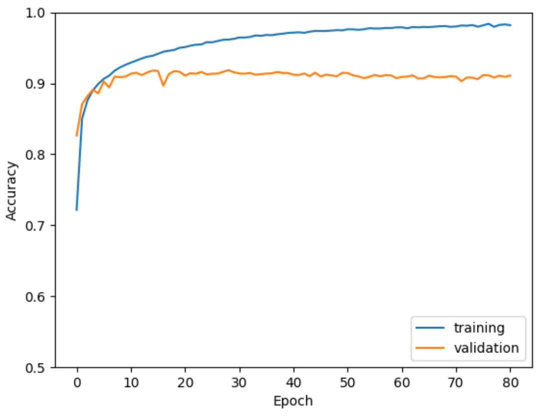
Pooling: Max; Number of Convolution Layers: 2
Activation Function: Relu; Loss Function: sparse categorical cross entropy:

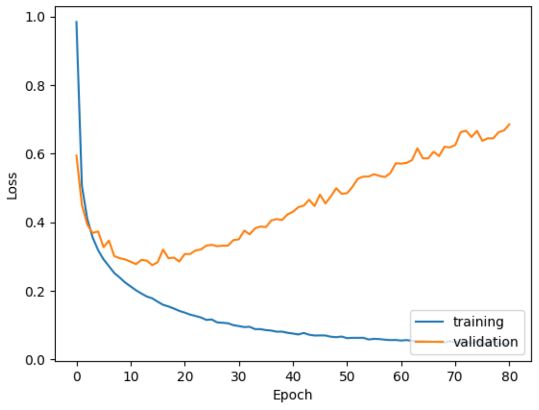




Test Loss: 0.6553019881248474 ; Test Accuracy: 0.9132310748100281

The training accuracy and training loss plots look similar to the model without max pooling. The total test accuracy is 0.913 which is slightly higher than the model without max pooling, indicating that max pooling helps slightly.
Pooling: Average; Number of Convolution Layers: 2
Activation Function: Relu; Loss Function: sparse categorical cross entropy:





Test Loss: 0.68584698843864441 ; Test Accuracy: 0.9109300374984741

The training accuracy and training loss plots also look similar. The total test accuracy is 0.911 which is slightly higher than the model without pooling, but lower than the accuracy obtained with max pooling, indicating that average pooling is less effective than max pooling, but still more effective than no pooling. (This very slight increase in accuracy may also be due to randomization)

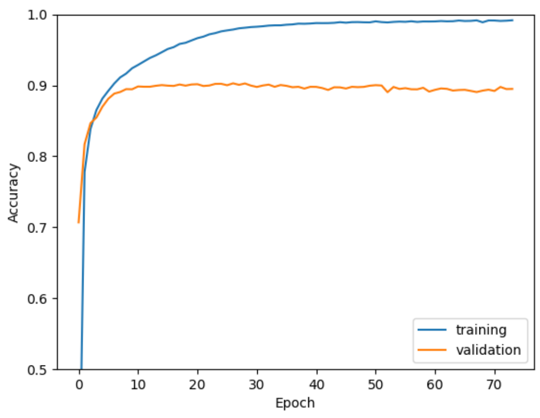
4.2 More Convolution Layers

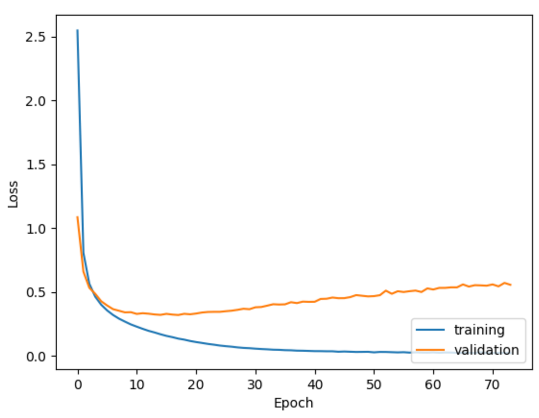
Pooling: None; Number of Convolution Layers: 3
Activation Function: Relu; Loss Function: sparse categorical cross entropy:
Test Loss: 0.756642758846283 ; Test Accuracy: 0.9055609107017517

The training accuracy and training loss plots look almost identical to the model with two convolutional layers. The total test accuracy is 0.905 which pretty much the same as the model with two layers, indicating that adding more convolution layers doesn't necessarily improve the model.

4.3 Activation Functions

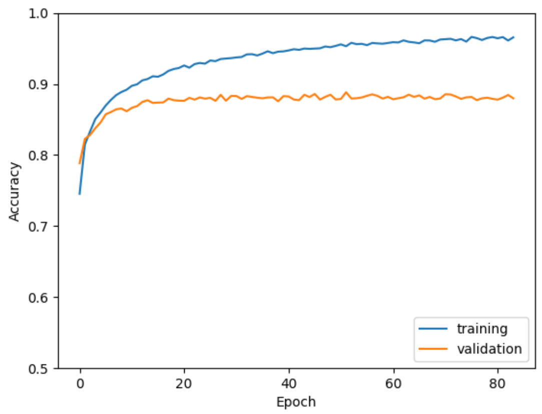
Pooling: None; Number of Convolution Layers: 2
Activation Function: Sigmoid; Loss Function: sparse categorical cross entropy:

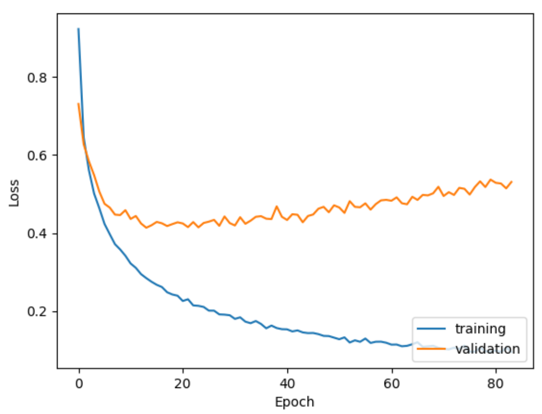




Test Loss: 0.5571948289871216 ; Test Accuracy: 0.8949185013771057

Pooling: None; Number of Convolution Layers: 2
Activation Function: Tanh; Loss Function: sparse categorical cross entropy:





Test Loss: 0.5308774709701538 ; Test Accuracy: 0.8797699213027954 The accuracy of the model using either a sigmoid activation function or a tanh activation function seems to have slightly less accurate predictions than the relu model.

4.4 Loss Functions

Pooling: None; Number of Convolution Layers: 2
Activation Function: Relu; Loss Function: mean squared error:
Test Loss: 9.423781394958496 ; Test Accuracy: 0.047171618789434443
Using mean squared error instead of sparse categorical cross entropy drastically decreases the accuracy.

4.5 Overall Effectiveness

Throughout our tests, it seems like a model that has 2 convolutional layers, utilizes max pooling, uses a relu activation function, and uses a sparse categorical cross entropy loss function, yields the highest accuracy of 0.91323. As stated previously, this accuracy is not bad considering the simplistic structure of our model. However, it would never be sufficient to release in a product like an Iphone. One typo or misclassification every 10 characters would make any message extremely frustrating to read, and possibly might entirely alter the meaning of the statement.

5 Future Work

One of the features of the dataset that I did not use is the next_id feature. This feature can be used in an LSTM model: a recurrent neural network that is effective for time series data. Having a ’memory’ of previously classified characters would give the model an advantage in predicting the current character and would probably increase the accuracy of the model. One might also consider simplifying the structure of the model by using a multilayered perceptron instead. There is a possibility that a convolutional neural network is not even necessary or beneficial because our data is already formatted into a linear array of 0s and 1s.