Using a Convolutional Neural Network (CNN) for Handwritten Digit Recognition

Matthew Bouchard  
1. Undergraduate Student, University of North Georgia, Dahlonega, Georgia, U.S.A  
Email Address: mabouc4699@ung.edu

**Abstract – The task of recognizing handwritten characters has proven itself at times to be a difficult one to solve for both humans and computers; however, the aforementioned task appears to be one that can be resolved using current machine learning tools and practices. In this project, I approach the problem by using a convolutional neural network which contains varying structures and hyperparameters to solve the problem. In training and configuring the network, I hoped to develop a working solution for identifying handwritten numbers, and I report on the problems and discoveries associated with achieving the solution along the way. The significance of this project lies in the fact that it delves into the basics of deep learning. With more complex neural networks, the possibilities are endless with what can be done given tailored parameters. Having solved this problem previously with an artificial neural network, the goal of this project is to develop a better solution for the problem of recognizing handwritten numbers by using the power of convolutional neural networks.**

*Keywords – MNIST, Neural Network, Label, Convolutional, Solver, Neuron*

1. INTRODUCTION

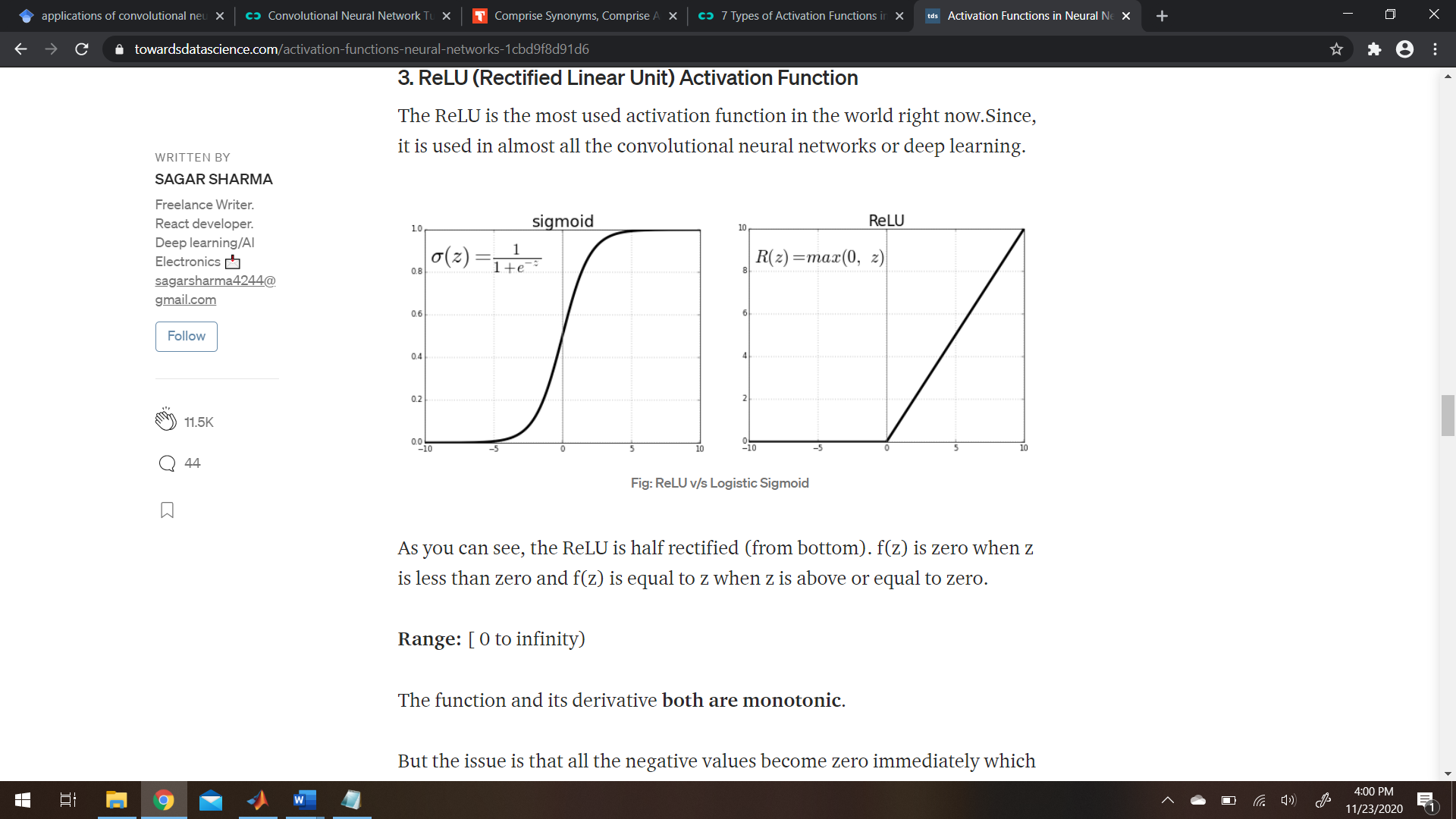
The task of recognizing handwritten characters can be a difficult one for humans and computers to solve; however, seeing as this problem can be solved using an artificial neural network (ANN) like in Loren Shures (contributor and staff member of Mathworks) tutorial [1], one can only assume that using a more complex neural network (NN) like that of a convolutional neural network (CNN) can perform at comparable levels and even go further to result in a NN that more accurately identifies handwritten digits. This can be attributed to the fact that CNNs utilize some properties of ANNs which will be elaborated on in the next section.

1. INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS

CNNs can be composed of any combination of layers a developer may wish to have; however, the basic components every developer includes is a convolution, activation, pooling, and fully connected layers.

Convolutions can be seen as the process of sliding a filter (matrix) over the input data creating an array of dot products in the process. Each filter results in a single feature being extracted from the input data. The result of convolving each filter with the input data is a feature map. The purpose of extracting features is to gather information on the input data for example: given a 28 x 28 x 1 image of the handwritten digit two, the convolutional layer results in the detection of edges, corners, and etc. which are useful data to use later in identifying what digit that number represents. The process of convolving input data using user-defined settings for filters and padding comprises the convolutional layer of a CNN.

The purpose of the activation layer is different depending on the case usage; however, for many modern neural network models, non-linear activation functions are used as many modern problems are non-linear including image classification. For this project, the rectified linear unit (ReLU) activation function is used because of its computational efficiency and convergence performance. This can be attributed to the function graph of ReLU seen below [3]:



ReLU calculates the activation value of the given node based on the weighted sum input setting the activation value equal to zero if the input to the function is negative and leaves the input value unchanged if it is greater than zero.

The pooling layer is responsible for reducing the number of parameters that are used in training the network. Average and max pooling are the most common types and calculate respectively: the average of the pooling region or the maximum element in the pooling region. This is useful because it can save computational resources.

The fully connected layer of a CNN can perform various tasks; however, for this project, that is taking the result of the previous layer and gives probabilities to the proceeding layer for use in classifying the input. This is where the ANN and CNN relate on the architectural level as, in an ANN, the neurons are all connected to each other meaning fully connected.

1. IMPLEMENTATION DETAILS

The requirements needed to satisfy the project were to: read in training and testing data sets and labels, write those images into subfolders spanning 0 – 9 within their respective parent folders (used later to create an imagedatastores), create / train a handwritten image detection network, test the accuracy of the network using the testing data set, and choose the hyperparameters that yielded the most accurate network possible.

To satisfy the initial two requirements for the project, the training and testing images were loaded into the initializing script using the loadMNISTImages and loadMNISTLabels. This gave a total of four data sets: two for training / testing images and two for training / testing labels. Afterwards, both label sets were set to 10 where they equaled 0 to offset Matlab’s 1 based system. This resulted in formatted data usable in the proceeding lines of code where the images were written to numbered subfolders within their respective training and testing folders. This was done through the use of two for loops and a series of conditional statements. This can be seen in the example code below for writing the training images where the label = 0:

for i = 1:60000

txt\_img = 'img';

i\_txt = num2str(i);

if tr\_labels(1,i) == 10

a = strcat('Path\_to\_Training/0/ Folder', txt\_img, '\_', i\_txt, '.jpg');

a\_image = reshape(tr\_set(:,i), 28, 28, 1);

imwrite(a\_image, a);

The variables txt\_img and i\_txt are used for the naming convention for each image (Ex: img\_200.jpg) when concatenated with the string that includes the path to the training folder. After, the image for that given index in the training set is reshaped from its 784 x 1 format to 28 x 28 x 1. The image is then written to the concatenated string a which holds the path to the correct folder (in addition to the name of the image) and the image at that given index of the training set. This is conducted for all the training and testing data, 60000 and 10000 images respectively, and concludes the formatting / initialization of data for use in future operations.

In the next script, the paths for the training and testing data sets were saved to variable matrices. These variables were used to create the imagedatastores (imd) using the following code:

trainingPath = fullfile('Path\_to\_Training\_Set);

testingPath = fullfile('Path\_to\_Testing\_Set');

trainImds = imageDatastore(trainingPath, 'IncludeSubfolders', true, 'LabelSource', 'foldernames', 'FileExtensions', '.jpg');

testImds = imageDatastore(testingPath, 'IncludeSubFolders', true, 'LabelSource', 'foldernames', 'FileExtensions', '.jpg');

The training imd was portioned out to 80% training and 20% validation in preparation for training the CNN.

[trainImdsTraining, trainImdsValidation] = splitEachLabel(trainImds,0.8);

To train the CNN, the parameters for inputSize, numClasses, layers, and options were defined. inputSize for our given data set was [28 x 28 x 1] corresponding to the dimensions of the images used in training. numClasses was set to 10 for the 10 possible outcomes of the network spanning 0 – 9. The layers for the network correspond to the architecture of the CNN and was created using the following code:

layers = [ imageInputLayer(inputSize)

convolution2dLayer(5,15, 'Padding', 'same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(2,'Stride',2)

convolution2dLayer(5,30,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(2,'Stride',2)

convolution2dLayer(5,45,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(2,'Stride',2)

fullyConnectedLayer(500)

fullyConnectedLayer(numClasses)

softmaxLayer

classificationLayer];

The layer structure represents the third and most complex architecture used to obtain the most accurate CNN for digit recognition. The first architecture created and used for training represents the least complex NN and only used one convolutional, batchnormalization, and relu layers while the second architecture used two of each mentioned layer.

To elaborate, the imageInputLayer was given the inputSize representing the input layer. Afterwards, a series of repeating layers of convolutional, batchnormalization, relu, and maxpooling layers were used to create a deep learning network; however, Arch. 1 is likely considered simply CNNs for its lack of multiple repeating layers of convolutional, among other, layers. The convolutional2dLayers all use 5x5x1 filters; though, the number of filters used per repeated convolutional layer increments by 15. Meaning, the first conv. layer uses 15 filters, followed by 30, and lastly 45. Padding for the convolutional layer was set to ‘same’ to ensure the size of the output was equal to the size of the input. A maxPooling2dLayer was used after every set of convolutions, batchnormalization, and relu layers to aid in reduction of parameters used for calculating proceeding operations. Post maxPooling, all the network data was fed into a fullyConnectedLayer composed of 500 neurons. This FC layer is responsible for learning possible non-linear combinations of features and aids in the classification process. This FC layer was then fed into another FC layer with only 10 neurons equal to the number of classes established before the training of the network (equal to the number of labels to classify an image as 0-9). Then, the data is fed into a softmax layer for use in creating a probability distribution for all the input nodes. This aids in further classifying the images to their correct label value. Used after the softmax layer, the classification layer calculates the cross-entropy loss for the given inputs. This finalizes the structure of the CNN.

An options matrix was created that held the settings for parameters like solver name (SGDM or Adam), ValidationFrequency, MiniBatchSize, executionEnvironment, and more.

options = trainingOptions('sgdm','MaxEpochs',25,'ValidationData',trainImdsValidation,'ValidationFrequency',30,'Verbose',false,'Plots','training-progress','MiniBatchSize',128, 'ExecutionEnvironment','gpu');

The code that established the settings chosen for the most accurate network possible (third architecture using SGDM solver) are depicted above and effectively complete preceding operations required to train the CNN with a MiniBatchSize=128, ValidationFrequency=30, and MaxEpochs=25.

The network was trained via the use of the predefined function trainNetwork and utilized the 80% allocated training data (trainImdsTraining) as well as the layers and options matrices. The result was saved to matrix variable net and is used later in calculating the accuracy of the NN given the testing Imds.

The accuracy of the network on the testing imds is calculated using the following snippet of code:

testPredictions = classify(net, testImds);

testLabels = testImds.Labels;

testing\_accuracy = sum(testPredictions == testLabels)/numel(testLabels);

Firstly, the testPredictions (predicted labels of the given images) are calculated using the classify() function with the parameters net and testImds. After, the testLabels are set to the actual label value for the given image. Finally, the result of summing every instance where the testPredictions equal the testLabels and dividing that value by the number of elements in the testLabels array is saved to the variable testing\_accuracy. The validation accuracy and testing\_accuracy of the CNNs created using various parameters are discussed in the next section.

1. EXPERIMENTAL RESULTS

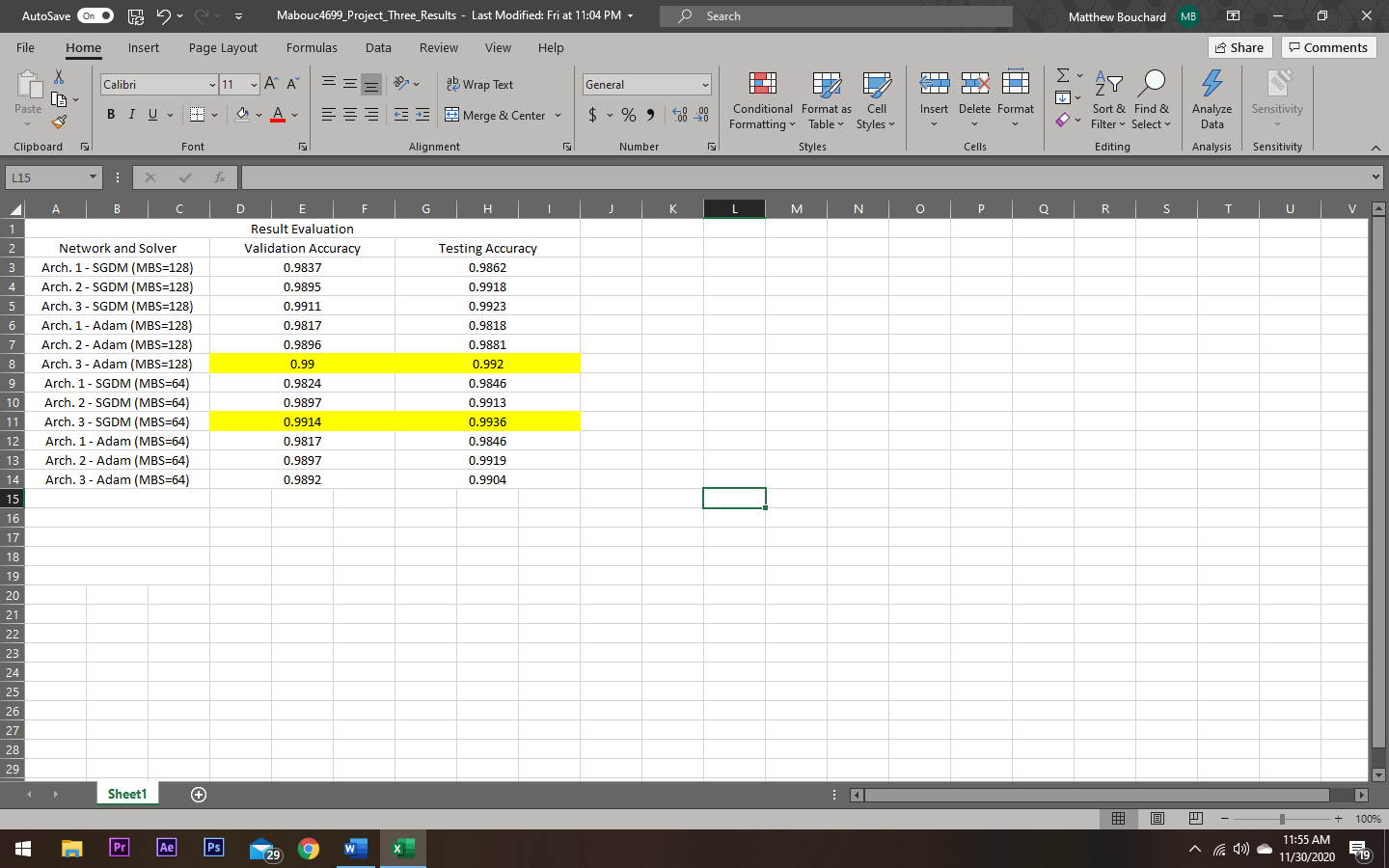
The CNN created using Arch. 3 (first and simplest architecture with only one convolutional, batchnormalization, relu, and maxpooling layers) with the following parameters: Solver = SGDM, MiniBatchSize = 128, MaxEpochs = 25 and ValidationFrequency = 30, yielded a validation accuracy of 0.9911 and a testing accuracy of 0.9923.

The CNN created using Arch. 3 (first and simplest architecture with only one convolutional, batchnormalization, relu, and maxpooling layers) with the following parameters: Solver = Adam, MiniBatchSize = 128, MaxEpochs = 25, and ValidationFrequency = 30, yielded a validation accuracy of 0.9900 and a testing accuracy of 0.9920.

The CNN created using Arch. 3 (first and simplest architecture with only one convolutional, batchnormalization, relu, and maxpooling layers) with the following parameters: Solver = SGDM, MiniBatchSize = 64, MaxEpochs = 25, and ValidationFrequency = 30, yielded a validation accuracy of 0.9914 and a testing accuracy of 0.9936.

The CNN created using Arch. 3 (first and simplest architecture with only one convolutional, batchnormalization, relu, and maxpooling layers) with the following parameters: Solver = Adam, MiniBatchSize = 64, MaxEpochs = 25, and ValidationFrequency = 30, yielded a validation accuracy of 0.9914 and a testing accuracy of 0.9936.

The rest of the results are posted in the excel screenshot below and provide a visual for how each architecture with various parameters performed when tested against the validation and testing data sets.



The conclusion reached from the results of testing the trained CNN is that using Arch. 3 (most complex network structure I created) with a MiniBatchSize of 64, Max Epoch of 25, and a solver of SGDM gave a 99.14% accuracy on the validation training data. Additionally, an accuracy of 99.36% was reached with classifying images from the testing imd. Also, the second most accurate network created used the parameters for solver equal to adam, MiniBatchSize of 128, and Max Epoch equal to 25 and resulted in a validation accuracy of 99%. The result of classifying the testing imd using the network was a 99.20% accuracy.

1. CONCLUSION

One can conclude that creating, training, and utilizing a deep learning algorithm like that of a CNN to recognize handwritten digits is feasible and can result in a script that can, within a small margin of error, return a correctly predicted target / label given a set of image data. This same process can be conducted to solve an array of problems within the world around us, and many references have done so with great success. The continuation of research within the field of machine learning signifies a great step towards advancing society and creating automated solutions for otherwise manual tasks.

1. REFERENCES

[1] L. Shure, “Artificial Neural Networks for Beginners,” Loren on the Art of MATLAB, 04-Aug-2015. [Online]. Available: https://blogs.mathworks.com/loren/2015/08/04/artificial-neural-networks-for-beginners/. [Accessed: 23-Nov-2020].

[2] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way,” Medium, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 23-Nov-2020].

[3] S. Sharma, “Activation Functions in Neural Networks,” Medium, 14-Feb-2019. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 23-Nov-2020].