Digit Recognizer using CNNs

Jaswant Naidu

EECS
Washington State University
Pullman, WA-99164

Tejas Ghanwat

EECS
Washington State University
Pullman, WA-99164

Abstract

Handwritten digit recognition plays an important role in many applications in today's world. As the handwritten digits are not of the same size, thickness, style, and orientation, therefore, these challenges are to be faced to resolve this problem. We have used the MNIST dataset which is a classic dataset of handwritten images. We are given with only training and testing data, so we have split the training data into two parts and used the other part as validation data which is for our reference. We have approached this problem with the concept of Convolutional Neural Networks(CNN). We have used multi-layers to unravel this complication and also KERAS is the API that is used for implementing the neural networks. We trained a classifier that is able to identify a digit after being trained on tens of thousands of such handwritten images. The multilayers are connected to visualize the output. The result shows the confusion matrix, accuracy curve and the loss curve.

1 Introduction

Handwritten digit recognition has been a huge research in the recent times where various algorithms are being used to perform recognition. Today, numerous touch screen based gadgets allow users to write notes, draw, etc. This very application needs automated digit recognition. There are numerous old records in hardcopy form in areas like banking, hospitals, politics, etc. Scanning these as images won't be enough until the data from those documents could be converted in computerized data so it could be used in building datasets and generate predictions using machine learning algorithms. CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output.

1.1 Motivation

As the handwritten digits are not of the same size, thickness, style, and orientation, therefore, these challenges are to be faced to resolve this problem. Based on the input to the system, handwritten digit recognition can be categorized into online and offline recognition. This project will lead us deep into the field of neural networks and understanding the concepts with a much better picture. Also working on classification problem that involves images would not only allow us to tackle a real world problem but also help us gain a decent insight of Computer Vision with Machine Learning.

2 Problem Setup

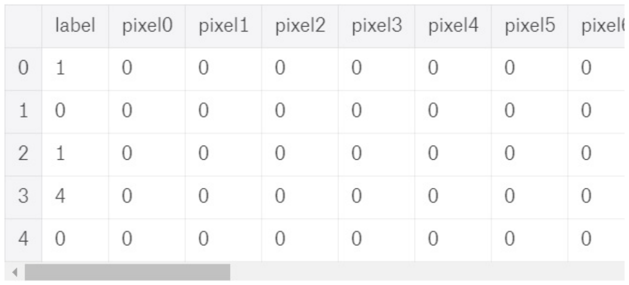
2.1 Data

The data is set of handwritten images borrowed from the Kaggle challenge. The dataset is built by MNIST which is considered to be a classic dataset and is widely used to implement digit recognition algorithms. The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. Each pixel column in the training set has a name like pixel x, where x is an

Figure 1: Sample MNIST handwritten digits.

integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixel x is located on row i and column j of a 28 x 28 matrix, (indexing by zero).[3] pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column. For each of the 10000 images in the test set, output a single line containing the ImageId and the digit you predict. The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that you have correctly classified all but 3% of the images are misclassified.



	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6
0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0

Figure 2: Dataframe with class label and pixels.

2.2 Base Learner

The base learner in our case is neural networks which we use in the convoluted form. As we know neural networks or artificial neural networks is a subfield of deep learning.

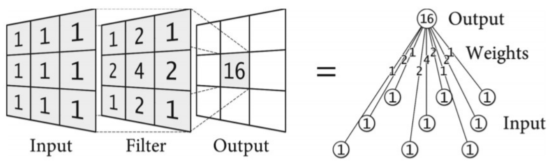


Figure 3: CNN filters [1].

A standard neural network (NN) consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons [2]

3 Experiments

3.1 Conceptual Understanding

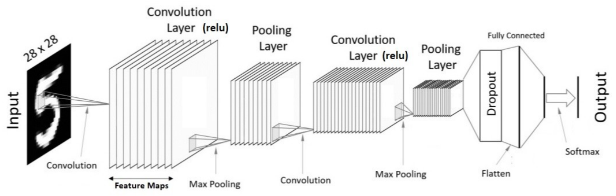


Figure 4: Convolutional Neural Network working.

The image above briefly explains how we tackle the problem and build a system that recognizes handwritten digits. Initially we take is put, the 28*28 images, and pass it to the convolutional layer. Further, the output from the that is passed to the pooling layer. The max-pooling process is explained later in the paper. Then the output from max-pooling is again dropped in the convolutional layers and this is repeated until flattened data is received which is preferably one dimensional. After that, softmax is performed to output the final data on which the model is trained and accuracy can be shown. We used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. We choose to set 32 filters for the two firsts conv2D layers and 64 filters for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their wieghts to zero) for each training sample. This drops randomly a propotion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

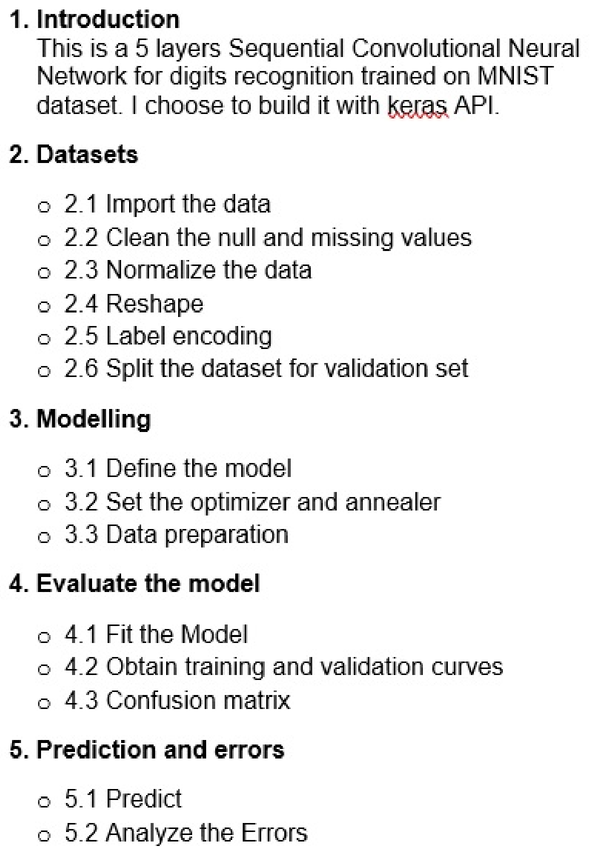
'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

In the end i used the features in two fully-connected (Dense) layers which is just artificial an neural networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

3.2 Programming and Approach

The training data is being split into training and validation set as the test data does not have class labels. This way we get to calculate the accuracy based on the given class labels of the validation set.So, the training set is 80% and validation set is 20%



1. Introduction
This is a 5 layers Sequential Convolutional Neural Network for digits recognition trained on MNIST dataset. I choose to build it with keras API.

2. Datasets

- 2.1 Import the data
- 2.2 Clean the null and missing values
- 2.3 Normalize the data
- 2.4 Reshape
- 2.5 Label encoding
- 2.6 Split the dataset for validation set

3. Modelling

- 3.1 Define the model
- 3.2 Set the optimizer and annealer
- 3.3 Data preparation

4. Evaluate the model

- 4.1 Fit the Model
- 4.2 Obtain training and validation curves
- 4.3 Confusion matrix

5. Prediction and errors

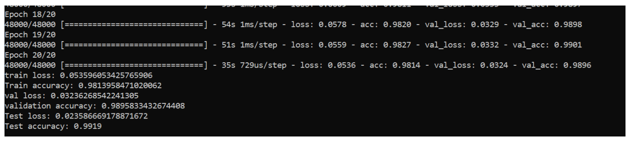
- 5.1 Predict
- 5.2 Analyze the Errors

Figure 5: Approach.

4 Results

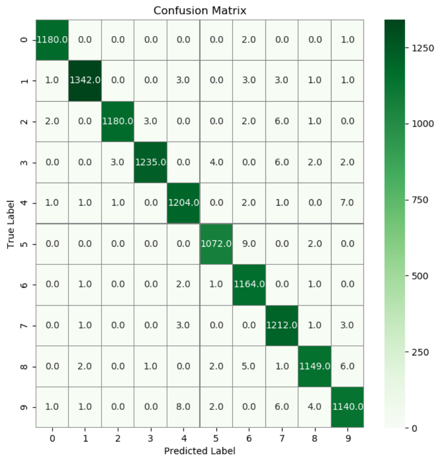
The figure 6 gives frequency of each digit in the data set. We have performed data augmentation which means that the images are put into proper order and position.These include the activities like rotation, shifting, This calibration helps in vision of predicting the examples appropriately. On a topic of epoch and batch size parameters, let's talk about an example and make this clear. If there are 100 examples and the batch size is set to 20 and epochs value is set to 5. That means in each epoch we have 5 batches(100/5). So we will be having 5 iterations per epoch. Increasing the epochs will lead to a better accuracy figure on the model. We have used Adam, It is one of the best gradient descent algorithm. We use it to minimize the cost function to approach the minimum point. The training data is being split into training and validation. This way we get to calculate the accuracy based on the given class labels of the validation set as well.

As we can see above in the figure for confusion matrix, CNN fails to classify certain numbers such as '6' is



```
Epoch 18/20
48000/48000 [==============================] - 54s 1ms/step - loss: 0.0578 - acc: 0.9820 - val_loss: 0.0329 - val_acc: 0.9898
Epoch 19/20
48000/48000 [==============================] - 51s 1ms/step - loss: 0.0559 - acc: 0.9827 - val_loss: 0.0332 - val_acc: 0.9901
Epoch 20/20
48000/48000 [==============================] - 35s 729us/step - loss: 0.0536 - acc: 0.9814 - val_loss: 0.0324 - val_acc: 0.9896
train loss: 0.053596053425765906
Train accuracy: 0.9813958471020062
val loss: 0.03236268542241305
validation accuracy: 0.9895833432674408
Test loss: 0.023586669178871672
Test accuracy: 0.9919
```

Figure 6: Accuracy.



	0	1	2	3	4	5	6	7	8	9
0	1180.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	1.0
1	1.0	1342.0	0.0	0.0	3.0	0.0	3.0	3.0	1.0	1.0
2	2.0	0.0	1180.0	3.0	0.0	0.0	2.0	6.0	1.0	0.0
3	0.0	0.0	3.0	1235.0	0.0	4.0	0.0	6.0	2.0	2.0
4	1.0	1.0	1.0	0.0	1204.0	0.0	2.0	1.0	0.0	7.0
5	0.0	0.0	0.0	0.0	0.0	1072.0	9.0	0.0	2.0	0.0
6	0.0	1.0	0.0	0.0	2.0	1.0	1164.0	0.0	1.0	0.0
7	0.0	1.0	0.0	0.0	3.0	0.0	0.0	1212.0	1.0	3.0
8	0.0	2.0	0.0	1.0	0.0	2.0	5.0	1.0	1149.0	6.0
9	1.0	1.0	0.0	0.0	8.0	2.0	0.0	6.0	4.0	1140.0

Figure 7: Confusion Matrix.

being misclassified as '1' one time and same is the case with '3' is being misclassified as '7' six times. However, the model performs well for most of the digits by gaining an accuracy of more than 98%.

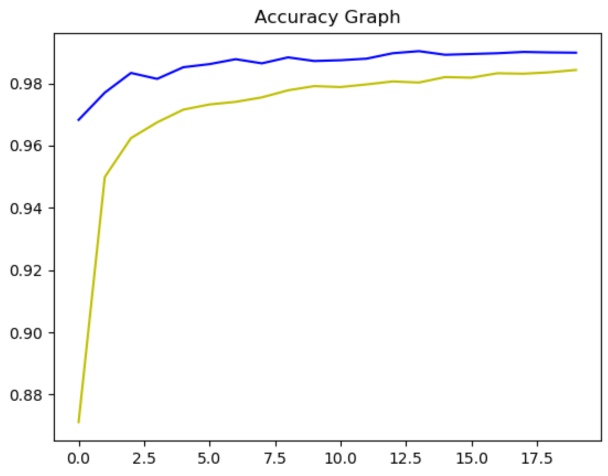


Figure 8: Accuracy Curve.

The accuracy plot shows the training accuracy which is represented by the yellow curve and the validation accuracy with the blue curve. As we can see, the accuracy shoots from 1st epoch and then tends to asymptote after 2.5 epochs.

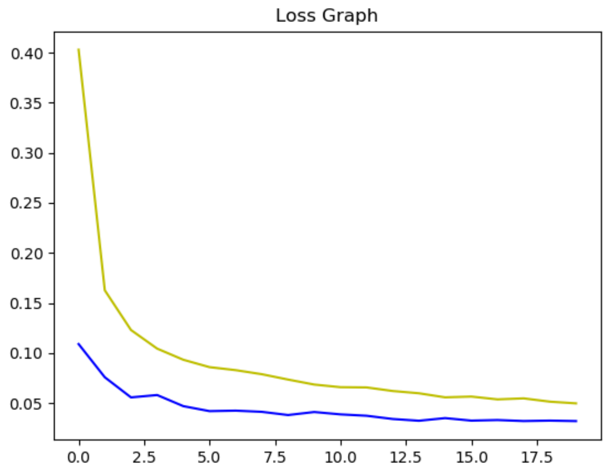


Figure 9: Loss Curve.

The program is run with 20 ephocs and batch size of 64. The figure above shows the working for each epoch and the respective accuracies gained and the drop in loss.

5 Conclusions

As you can see in the result's we have gained an accuracy of around 98% on the unseen examples also referred as validation set. The CNN model works efficient in this scenario. Also, the model failed to differentiate some numbers such as '8', '0'. After going through deep into this project, there exist an extension scope of recognizing the handwritten words. So, we are planning to implement the same on Alphabets. After 20 Epochs with a batch size 64, the accuracy obtained on the testing data is 98.19%, validation data is 98.95% and the testing accuracy is 99.19%.

6 Future Scope

Now that we got a better insight on how the Convolutional Neural networks work with the MNIST dataset, we may look forward to do a alphabet recognizer. As we know handwriting recognition is a major application in today's tech-age where many gadgets have introduced notebook features where users can just write on the screen and the context is converted in actual text. Also, huge number of historic manuscripts that are used for research purpose can be brought to digital format.

Acknowledgement

I'd like to thank Prof. Janardhan Rao Doppa for encouraging to do this project and teaching us Machine Learning concepts which were highly useful in the making of this project.

References

[1] *An Interactive Node-Link Visualization of Convolutional Neural Networks Adam W. Harley(B,* Department of Computer Science, Ryerson University, Toronto, ON M5B 2K3, Canada.

[2] Jürgen Schmidhuber,

Deep learning in neural networks.

[3] Digit Recognizer, *MNIST DATASET,*