MNIST Digit classification

622 – Final Project

Introduction:
This document explains about the neural network approach and the choices/tradeoffs made to create a MNIST handwritten digits classification model.

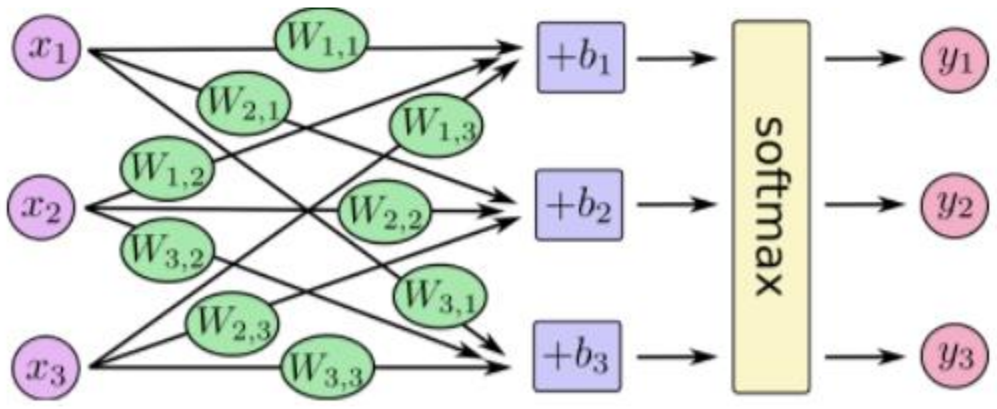
About MNIST dataset:
The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.

For our convenience the dataset is part of most of the neural network libraries dataset. One gray scale image is of size 28x28.

Preprocessing:
In some datasets (in tensorflow library) the images are stored in 784 pixels and the pixel is between the range 0 to 255. We need to reshape the image size to 28x28 and normalize the pixel values by dividing it by 255 and convert it from 0 to 1.

Start Simple - Neural Network:
As we are not aware where to start, we will start with a simple neural network. A simple neural network is a network which is less than three fully connected layers. It will be of below type

- Input Layer
- Hidden Layers
- Output Layer



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

Output Activation function: Softmax

$$z_i = \sum_j W_{i,j} x_j + b_i$$

$$y = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

W – Weights, x – input pixel, b – bias, y -output of softmax

Explanation on model:

1. Weights and bias are randomly initialized. It can be zeros or normalized values.
2. As the softmax outputs various outputs, we need to calculate the cross-entropy loss function.
3. Optimizer can be chosen as GradientDecent or RMSProp, etc.
4. Epoch can be in the range of 100 to 1000.
5. Batch can be in the range of 10 to 128.
6. Accuracy is calculated on the test dataset.

Pros:

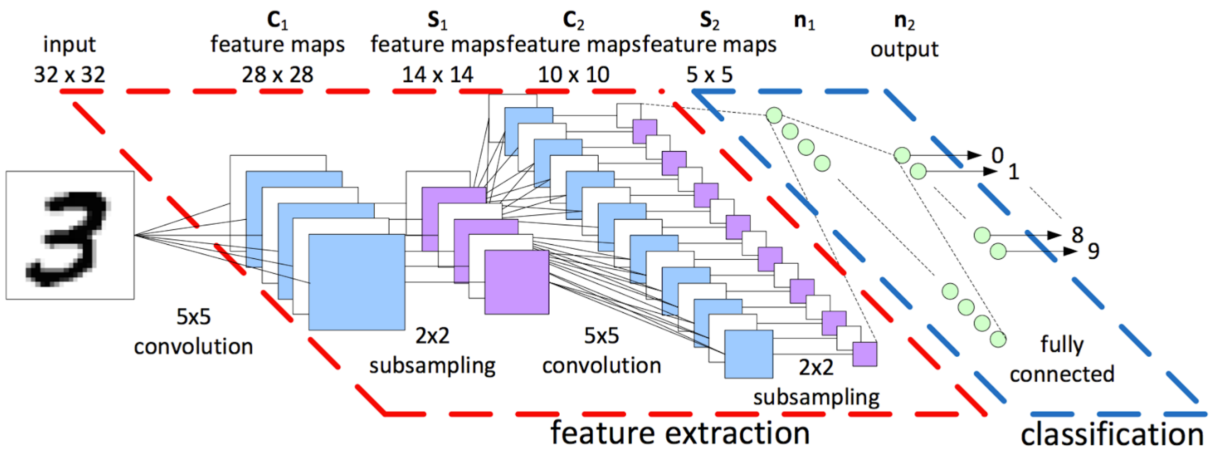
1. Neural network is simple and easy to create and understand as the pixels are flattened and inputed to the neural network.
2. Weights and backpropagation are easy to calculate.

Cons:

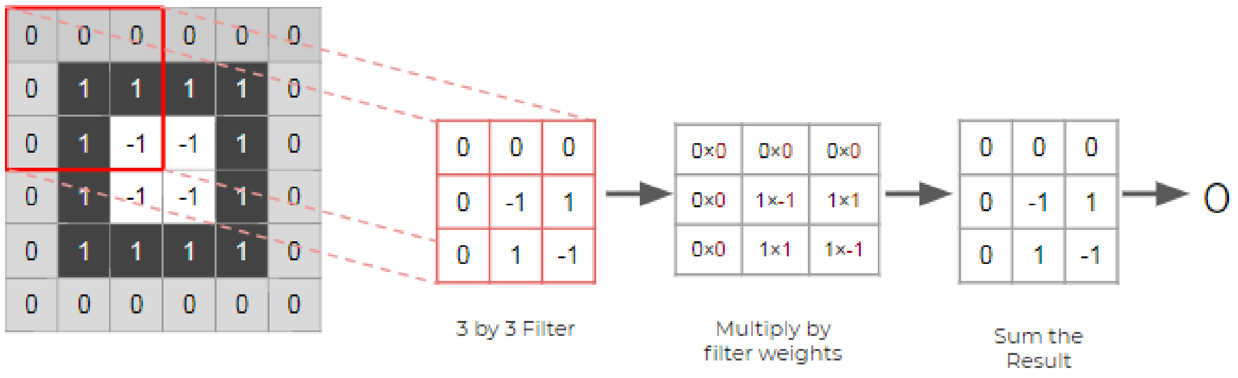
1. As the array is flattened it almost lose the relation between nearby pixels.
2. Input array is sparse and most of the neurons are assigned to 0's in this dataset.
3. Accuracy of the model is around 92%. Which is bad for MNIST dataset.

Convolutional Neural Networks(CNN):

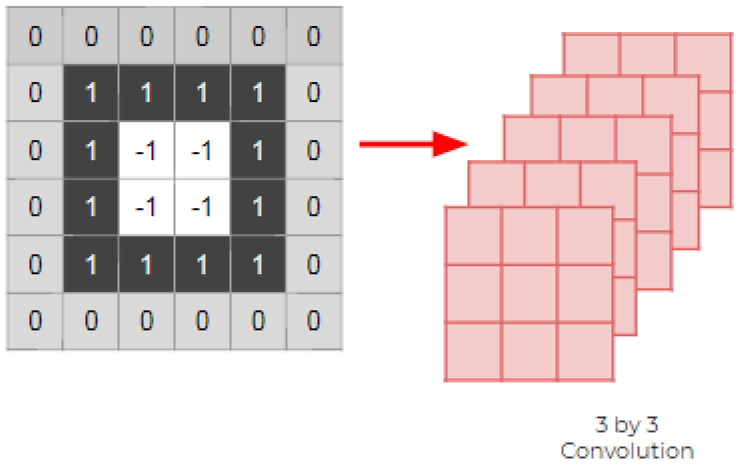
Convolutional Neural networks are specifically used for image classification. It acts like the neurons in brain by classifying the images with small portion of the image. Below is the complete picture of CNN.



1. Convolution:
   Each image (In above example 3 is the image) is looked for the local receptive fields. Image is kept in original shape of nxn dimensions. Feature map or filter which is of smaller dimension (ex. 5x5) is applied over each pixel of the original image.



Feature map applied over the set of pixels means applying the dot product and adding each pixel. These convolution is applied over the entire part of the image to create a new pixel image. Convenient stride size is chosen to navigate around all the pixels.

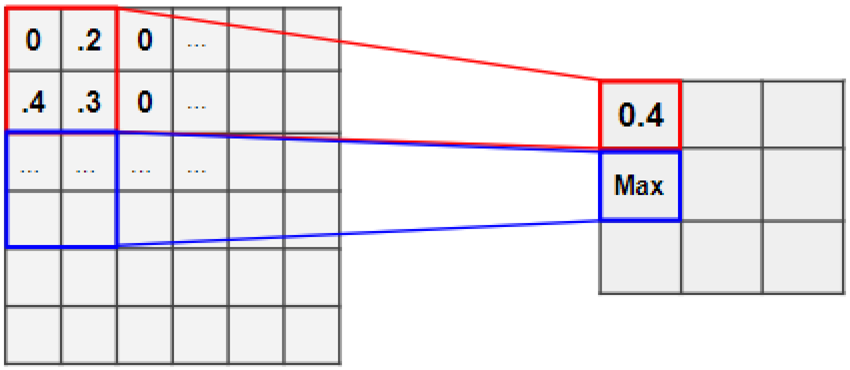


Many different feature maps are selected and different convolution images are created as output.

Parameters chosen in model:
Feature map size: 3x3
Stride: 1x1
Initial weights: Normalized weights
Regularizer: L2

2. Pooling:

   Once the convolution is applied, the images are traversed using pooling layer. Here the a pooling layer (ex. 2x2) is applied. It reduces the memory use and number of parameters. It also fetches the important input parameters.



Parameters chosen in model:
Pooling type: Maxpooling
kernel size: 2x2
Stride: 1x1

3. Dropout:

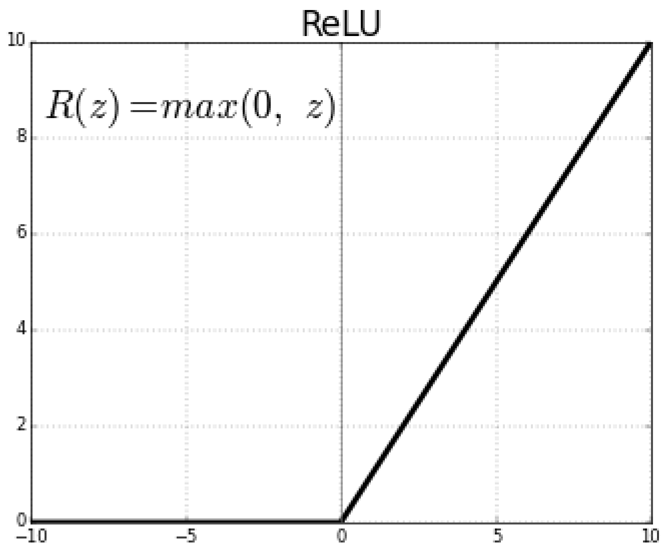
   Due to backpropagation, the model uses certain neurons heavily. To reduce the dependency on certain neurons, we will drop neurons randomly, so the model will learn from new neurons.

   Parameters chosen in model:
   Dropout Percentage: 10%

4. BatchNormalization:

   As there are many layers with different weights, the output of previous layers soon passes more than 1. By applying batch normalization the output is maintained close to 0 with the standard deviation of 1.

5. Steps 1, 2,3 and 4 are repeated to reduce the size and get the important features from input image.
6. After reducing the image size and extracting important features, we will apply RELU activation function. It will activate if the neurons is above the particular threshold.



7. Fully connected Neural network:

   Now the output of the step 6 is flattened into one single big array and passed to the fully connected neural network with Relu activation function. This is like the simple neural network which was mentioned first section. The number of output neurons is mentioned in this layer. Many layers can be added to it for finer control and accuracy.

8. Output layer(Softmax):

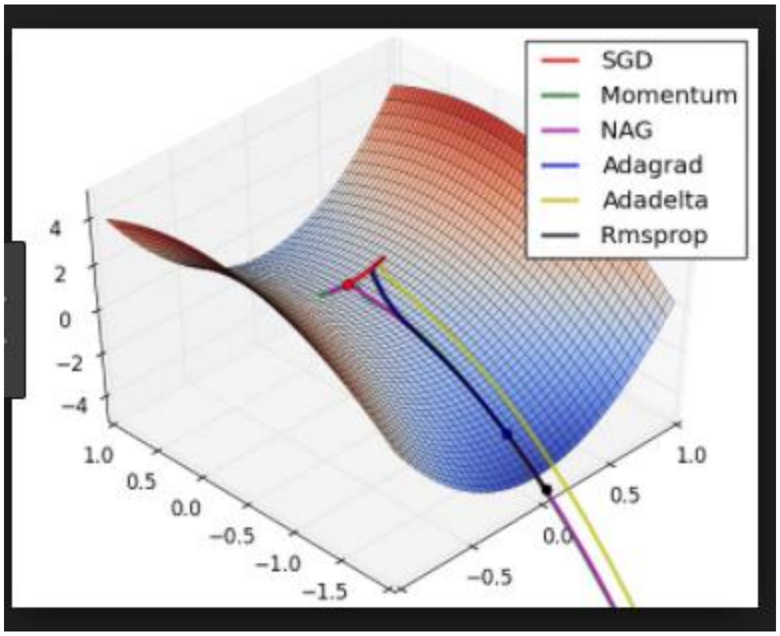
Finally, the softmax output layer is added to the layer with the output neurons of 10. It outputs the probabilities of each image with 10 different values. Highest probability index is the number of the image.

9. Loss Function:

   Loss function plays a critical role in accuracy of the model. It can be quadratic or cross-entropy. As this model is using softmax layer which outputs multiple values, we will use cross-entropy loss.

10. Optimizer:

    The optimizer function can be used to adjust the weights by calculating the loss. It also allows to converge to the value quickly by reducing the losses using backpropagation.



Parameters chosen in model:
Optimizer: RMSPROP
Learning Rate: 0.01

Model creation using Keras and Tensorflow:

Model has been created using two neural network libraries.

1. Keras
2. Tensorflow

Keras:

It is a rich higher-level API which runs tensorflow on backend (In this example).

1. Mode is developed using Sequential and Functional API.
2. It has detailed comments on all the inputs. Training of the model and evaluate/predict are performed in different files.
3. Model is exported to hdf5 format for future usage.
4. New images were generated and predicted.

Tensorflow:

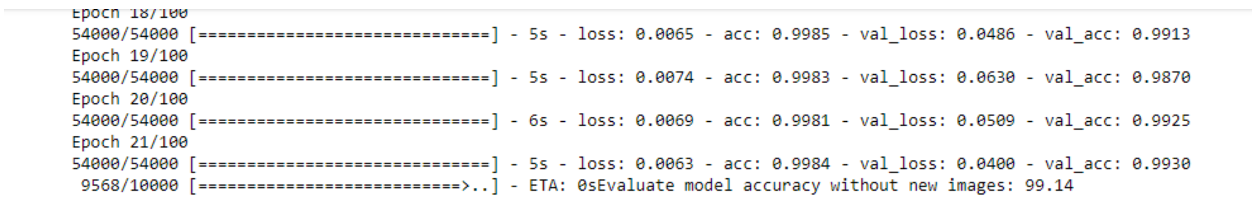
Low level neural network library which is used for MNIST classification.

1. Tried train and prediction with ANN. Also performed accuracy calculation.
2. Performed CNN separately and displayed evaluation metrics.

Running the model in GPU Cloud:

1. To achieve higher accuracy, we need to run with the complete model for higher epoch and batch size.
2. For this model we have chosen a epoc around 100 to 1000 and the batch size of 128.
3. We will use Floydhub(floydhub.com) project space which is GPU enabled.
4. Below are the final metrics from all the libraries.

Library	Type	Accuracy Score
Keras	Without new generated images	99.14
Keras	With new generated images	93.30
Tensorflow	ANN Fully connected	91.28
Tensorflow	CNN without new generated images	93.28



```
Epoch 18/100
54000/54000 [==============================] - 5s - loss: 0.0065 - acc: 0.9985 - val_loss: 0.0486 - val_acc: 0.9913
Epoch 19/100
54000/54000 [==============================] - 5s - loss: 0.0074 - acc: 0.9983 - val_loss: 0.0630 - val_acc: 0.9870
Epoch 20/100
54000/54000 [==============================] - 6s - loss: 0.0069 - acc: 0.9981 - val_loss: 0.0509 - val_acc: 0.9925
Epoch 21/100
54000/54000 [==============================] - 5s - loss: 0.0063 - acc: 0.9984 - val_loss: 0.0400 - val_acc: 0.9930
 9568/10000 [===========================>..] - ETA: 0sEvaluate model accuracy without new images: 99.14
```

Conclusion:

Keras has provided an accuracy score of 99.14% without generating new images. We can use this model to make future predictions.