Subject Code: **CSE5ML** Student Name: **Ashish Rawat**
Student Number: **20343966**

ANALYSIS OF DIFFERENT NEURAL NETWORK MODELS

Introduction

This report aims to present the various neural networks (NNs) for supervised learning. The MNIST dataset of handwritten numbers from 0 to 9, has a training set of 60,000 examples images, and a test set of 10,000 examples images. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 1. The task is formulated as multi-class supervised classification problem for hand-written images using three different Neural Network, and the goal is to model the relationship between an image's content and label approaches applied to a given dataset to find the most optimal and efficient trained model/approach to predict digits in images with high accuracy and reduced loss values using different convolutional layers, activation and loss functions.

Methodology

The data is divided into two sets; train data (to train our models) and test data (to evaluate and test against trained models). The data was loaded on Jupiter Notebook, operations performed importantly using library TensorFlow, Keras, Matplotlib, and NumPy. The data for modeling first reshaped to 1 channel (monochrome image) and normalized between values 0 and 1 to reduce processing size and increase performance. Labels are encoded to one-hot vectors to do a better job in prediction.

Modeling

Three Models Are Trained For Digits (0-9) Images Supervised Classification Using Different Layers, Complexity, And Other Parameters For Predictions -

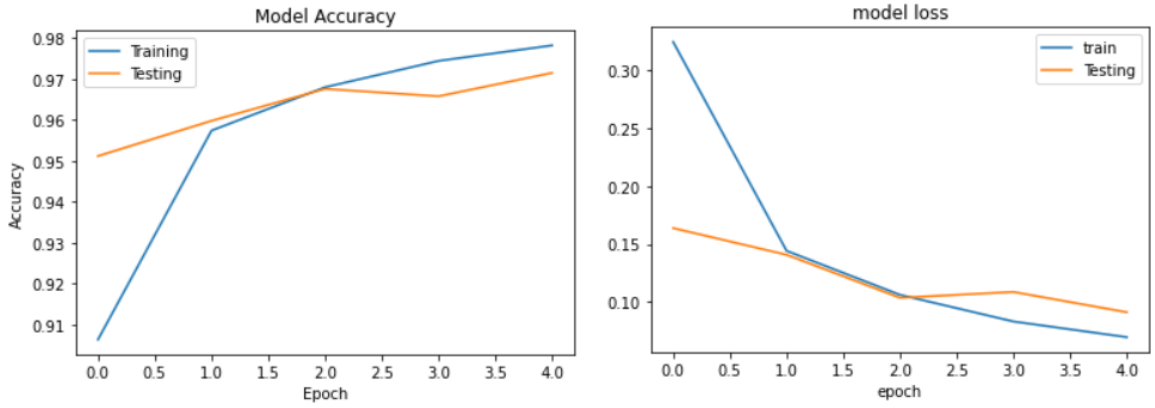
A) MODEL 1 (Simple Sequential Model)

It is a simple sequential model with two fully connected dense layers of 64-neurons in each layer with activation function 'Relu' (rectified linear unit) which is the most used non-linear activation function in neural networks, especially in CNNs. Adding 'SoftMax' function in the output layer for a multi-class classification problem and categorical_crossentropy loss function as for one-hot encoded for labels. 'Adam' optimizer is used to make model computationally efficient and deal with problems of noisy and sparse gradients in an efficient manner. Flatten layer added at the top to flatten input before adding hidden layers to speed the process.

Experiments (Improvement / Deteriorations):

1. Activation Function sigmoid drastically improves the accuracy and reduces the risk of loss rate but inconsistency, could have result in overfitting for new data.

2. Epochs more than 5 times did not add any weight to model improvement on this NN architecture.
3. The Flatten Layer at the top of this model drastically improved the execution speed of model execution.
4. Adding more Dense Layer (Fully connected) with 64, 128, 256 neurons in architecture did not add significant improvement but a processing time.
5. This model is suitable for building a model on real-time generated inputs with less time to process.
6. This model will have higher chances on model over-fitting on new data.




```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_25 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_73 (Dense)             (None, 64)                50240
_________________________________________________________________
dense_74 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_75 (Dense)             (None, 10)                650
=================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
```

```
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 3s 53us/sample - loss: 0.3246 - acc: 0.9064 - val_loss: 0.1637 - val_acc: 0.95
12
Epoch 2/5
60000/60000 [==============================] - 2s 39us/sample - loss: 0.1441 - acc: 0.9574 - val_loss: 0.1407 - val_acc: 0.95
98
Epoch 3/5
60000/60000 [==============================] - 2s 39us/sample - loss: 0.1060 - acc: 0.9680 - val_loss: 0.1035 - val_acc: 0.96
76
Epoch 4/5
60000/60000 [==============================] - 2s 40us/sample - loss: 0.0830 - acc: 0.9744 - val_loss: 0.1086 - val_acc: 0.96
58
Epoch 5/5
60000/60000 [==============================] - 2s 39us/sample - loss: 0.0695 - acc: 0.9782 - val_loss: 0.0910 - val_acc: 0.97
15
10000/10000 [==============================] - 1s 60us/sample - loss: 0.0910 - acc: 0.9715
Loss: 9.10%, Accuracy: 97.15%
```

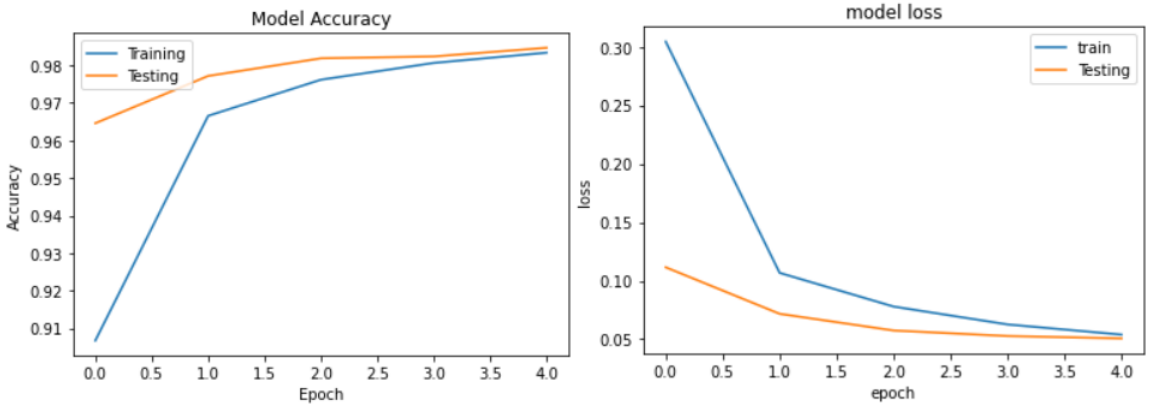
B) MODEL 2 (Convolutional layer with optimizer - RMSPROP)

It is a sequential classification model with convolutional neural network learning filters optimized to perform well (high accuracy and less loss) while considering computation cost and time to build the model. This model is built with '**rmsprop**' a full-batch optimizer to resolve the problem that gradients may vary widely in magnitudes and find a single global learning rate for the algorithm. Maximum epochs (iterations) used is 5 with a batch size of 64 samples in each epoch to process and learn in this model.
First layer, a Conv2D filter with the 'Relu' activation function, a model learns a total of 16 filters with size 3x3 on input values (images) with a default value of strides (1,1). Maxpooling Layer is then used to reduce the spatial dimensions of the output volume for the next layer. In a second layer, we repeated the process with 32 image detection filters of 3x3 size and further reduce the dimensions with the Maxpooling Layer of (2, 2) of the default value to reduce it half size. Finally, after flattening the Input data received from the Conv2D layers model, two fully connected dense layers of 128 neurons in each layer to produce the predicted values.

This model performs better than Model 1 but computational cost issues.

Experiments (Improvement / Deteriorations):

1. Filter size of 1x1 on layer 1 of Con2D with 'Relu' decrease performance.
2. Epochs more than 5 did not add any significance model improvement to this architecture.
3. Optimizer 'rmsprop' produces more significant performance with this NN architecture compare to optimizer 'Adam' or 'SGD'.
4. The dropout layer decreased accuracy in the model but could reduce overfitting issues.
5. Increased computation cost by adding more layers without any significant improvement in this architecture.
6. The overfitting error rate is low.



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 7s 120us/sample - loss: 0.3044 - acc: 0.9067 - val_loss: 0.1115 - val_acc: 0.964
6
Epoch 2/5
60000/60000 [==============================] - 5s 88us/sample - loss: 0.1067 - acc: 0.9666 - val_loss: 0.0717 - val_acc: 0.9772
Epoch 3/5
60000/60000 [==============================] - 5s 84us/sample - loss: 0.0779 - acc: 0.9762 - val_loss: 0.0574 - val_acc: 0.9819
Epoch 4/5
60000/60000 [==============================] - 5s 86us/sample - loss: 0.0626 - acc: 0.9807 - val_loss: 0.0527 - val_acc: 0.9824
Epoch 5/5
60000/60000 [==============================] - 5s 88us/sample - loss: 0.0539 - acc: 0.9834 - val_loss: 0.0507 - val_acc: 0.9847
Loss: 5.07%, Accuracy: 98.47%
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 13, 13, 16)        160
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 6, 6, 16)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 4, 4, 32)          4640
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 2, 2, 32)          0
_________________________________________________________________
dense_31 (Dense)             (None, 2, 2, 128)         4224
_________________________________________________________________
dense_32 (Dense)             (None, 2, 2, 128)         16512
_________________________________________________________________
flatten_10 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_33 (Dense)             (None, 10)                5130
=================================================================
Total params: 30,666
Trainable params: 30,666
Non-trainable params: 0
_________________________________________________________________
```

C) MODEL 3 (Dense Convolutional Neural Network Model)

It is a dense convolutional neural network, learning filters optimized to perform well reducing the overfitting by dropout layer of (0.5) in it. This model is built with an ‘SGD’ (Stochastic Gradient Descent) optimizer to find parameters that reduce the loss while optimizing the model in Neural Networks by taking the average gradient on all pairs of samples in the training set. The parameter set to learning rate of 0.002 and momentum of 0.9 to helps accelerate gradients vectors in the right directions, thus leading to faster converging. Maximum epochs (iterations) used are 5 with a batch size of 64 samples in each epoch to process.

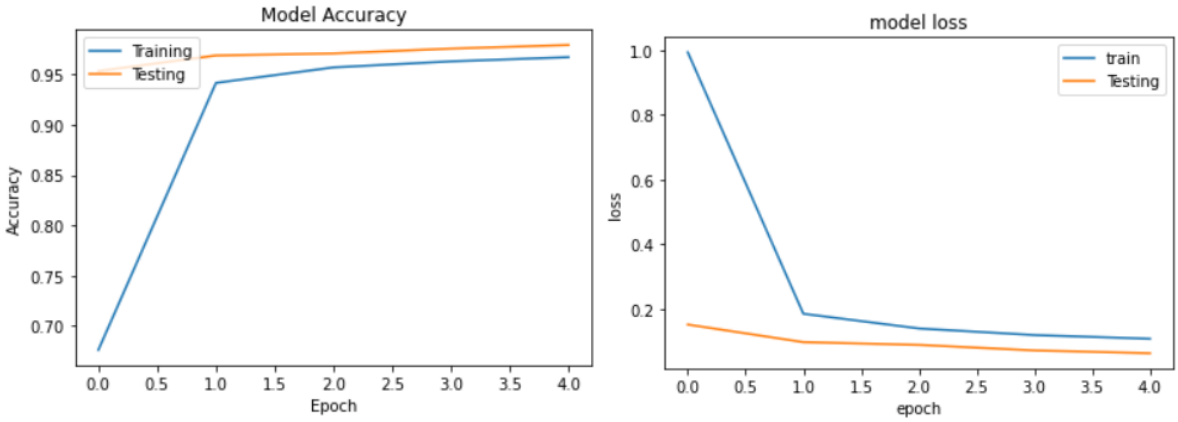
This model has the addition of extra Conv2D with ‘Relu’ activation model of 64 filters with size 3x3 on input values in Model 2 to reduce dimensions.

Experiments (Improvement / Deteriorations):

1. Using different activation function (‘adam’ or ‘rmsprop’) increase variations in accuracy of training and testing data, may result in overfitting in future data
2. Epochs more than 5 did not increase accuracy or reduce loss to model to that extend as much as computational cost and increased time processes.
3. Learning rate 0.10 or any parameter higher than 0.002 increased loss in model evaluation report.
4. Additional Conv2D layer improved accuracy.
5. Setting momentum of 0.5 in the 'SGD' optimizer parameter did not change accuracy or loss but the performance graph showed variance. So, 0.9 momentum value is optimum for this model.

```
Model: "sequential_38"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_52 (Conv2D)           (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 14, 14, 32)        4640
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 7, 7, 64)          18496
_________________________________________________________________
max_pooling2d_48 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
dense_106 (Dense)            (None, 3, 3, 128)         8320
_________________________________________________________________
dense_107 (Dense)            (None, 3, 3, 128)         16512
_________________________________________________________________
flatten_36 (Flatten)         (None, 1152)              0
_________________________________________________________________
dropout_10 (Dropout)         (None, 1152)              0
_________________________________________________________________
dense_108 (Dense)            (None, 10)                11530
=================================================================
Total params: 59,658
Trainable params: 59,658
Non-trainable params: 0
```

```
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 64s 1ms/sample - loss: 0.9928 - acc: 0.6762 - val_loss: 0.1527 - val_acc: 0.9533
Epoch 2/5
60000/60000 [==============================] - 63s 1ms/sample - loss: 0.1860 - acc: 0.9415 - val_loss: 0.0984 - val_acc: 0.9688
Epoch 3/5
60000/60000 [==============================] - 62s 1ms/sample - loss: 0.1406 - acc: 0.9570 - val_loss: 0.0900 - val_acc: 0.9708
Epoch 4/5
60000/60000 [==============================] - 63s 1ms/sample - loss: 0.1202 - acc: 0.9629 - val_loss: 0.0728 - val_acc: 0.9756
Epoch 5/5
60000/60000 [==============================] - 62s 1ms/sample - loss: 0.1089 - acc: 0.9671 - val_loss: 0.0642 - val_acc: 0.9792
0s - loss: 0.1088 - acc: 0.
10000/10000 [==============================] - 7s 723us/sample - loss: 0.0642 - acc: 0.9792
Loss: 6.42%, Accuracy: 97.92%
```



RESULTS

Although all models dispense good accuracy of over 95% and loss rate less than 10% when fitted the training and test data in the model.

The analysis and evaluation of neural network models suggests –

Model 2 is the best model because of the evaluation report (Loss: 5.07%, Accuracy: 98.47%) and the graph shows loss reduction and accuracy improved with consistent rate which depicts the chances of less overfitting or underfitting issue while keeping manageable computation cost and model evaluation time under 120us/sample.

Fully connected model 1 (Loss: 9.10%, Accuracy: 97.15%), despite its good accuracy and model evaluation speed (39us/sample), because evaluations report reveals the inconsistency of loss and accuracy rate which might result in overfitting in new data.

Dense NN Model 3 (Loss: 6.42%, Accuracy: 97.92%) have good accuracy and reduced loss rate similar to Model 2 but computationally expensive and slow model building and processing time (up to 1 ms/sec per sample).

REFERENCES

Medium. 2020. *Five Video Classification Methods Implemented In Keras And Tensorflow*. [online] Available at: <https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5> [Accessed 17 May 2020].

LI, A., LI, Y. and LI, X., 2017. TensorFlow and Keras-based Convolutional Neural Network in CAT Image Recognition. *DEStech Transactions on Computer Science and Engineering*, (cmsam).

Medium. 2020. *Convolutional Neural Networks From The Ground Up*. [online] Available at: <https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1> [Accessed 17 May 2020].