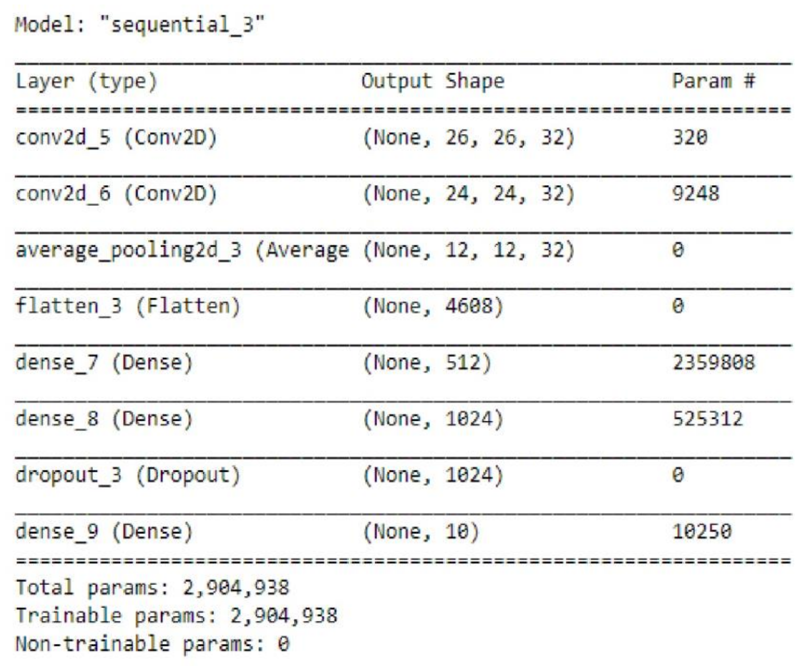
MNIST dataset classification using Neural Networks:

Architectures of the Neural Networks along with Figures

Neural Network 1:

All the Neural networks are using a sequential model which is best for layers which one input and output tensor. I have used 8 layers in the Neural Networks to train and test the model which are listed in figure 1 below.



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
average_pooling2d_3 (Average (None, 12, 12, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               2359808
_________________________________________________________________
dense_8 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                10250
=================================================================
Total params: 2,904,938
Trainable params: 2,904,938
Non-trainable params: 0
```

Figure 1 Neural Network-1

# load libraries

```
import tensorflow as tf

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import AveragePooling2D
from keras.utils import np_utils
```

Above code is to Import the tensorflow/keras libraries.

```
# load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

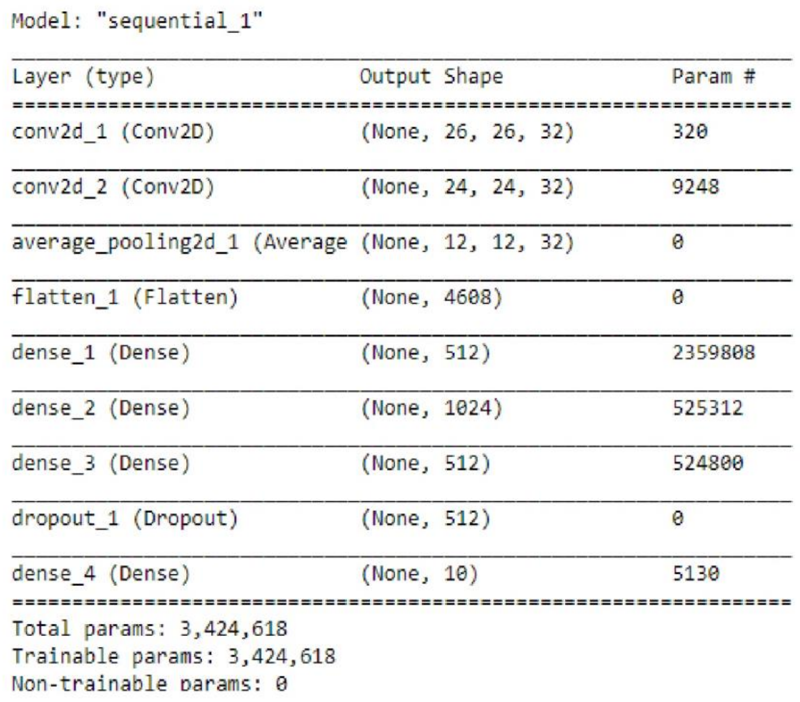
This is to load the database **mnist.** x_train and y_train are variables for training data and label of the training data. Similarly, x_test and y_test are for test data and label of testing data.

```
# normalize inputs from 0-255 to 0.0-1.0
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32') # This step is the change the dimension of the data from 3 dimensional to 4 dimensional.
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32') # This step is the change the dimension of the data from 3 dimensional to 4 dimensional.
x_train = x_train / 255.0 # This is to normalize the data points
x_test = x_test / 255.0 # This is to normalize the data points
# Encode the outputs
y_train = np_utils.to_categorical(y_train) #Converts a class vector (integers) to binary class matrix.
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# Build the model
model = Sequential()
```

```
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='relu')) # First layer – Convolutional layer using 32 filters of size 3 by 3

model.add(Conv2D(32, (3, 3), activation='relu')) )) # Second layer – Convolutional layer with 'relu' activation function using 32 filters of size 3 by 3

model.add(AveragePooling2D()) # Third layer – Average Pooling layer

model.add(Flatten()) # Fourth layer – Flatten layer is to add an extra dimension

model.add(Dense(512, activation='tanh')) # Fifth layer – Fully connected layer with 512 neurons and activation function 'tanh'

model.add(Dense(1024, activation='sigmoid')) # Sixth layer – Fully connected layer with 1024 neurons and activation function 'sigmoid'

model.add(Dropout(0.2)) # Seventh layer – to reduce the effect of overfitting

model.add(Dense(num_classes, activation='softmax')) # Eighth layer – defines the matching label with data

# Compile model

epochs = 5 # Number of iterations

lrate = 0.0012

decay = lrate/epochs


sgd = SGD(lr=lrate, momentum=0.7, decay=decay) #Stochastic gradient descent optimizer

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy']) # compiling the model

model.summary() # Print the summary of the model

# Fit the model

model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=epochs, batch_size=60, verbose=1)

# Final evaluation of the model

scores = model.evaluate(x_test, y_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]*100))
```

Neural Network 2:

The whole code is same as Neural Network 1 but the number of layers are 9 in this Neural Network.



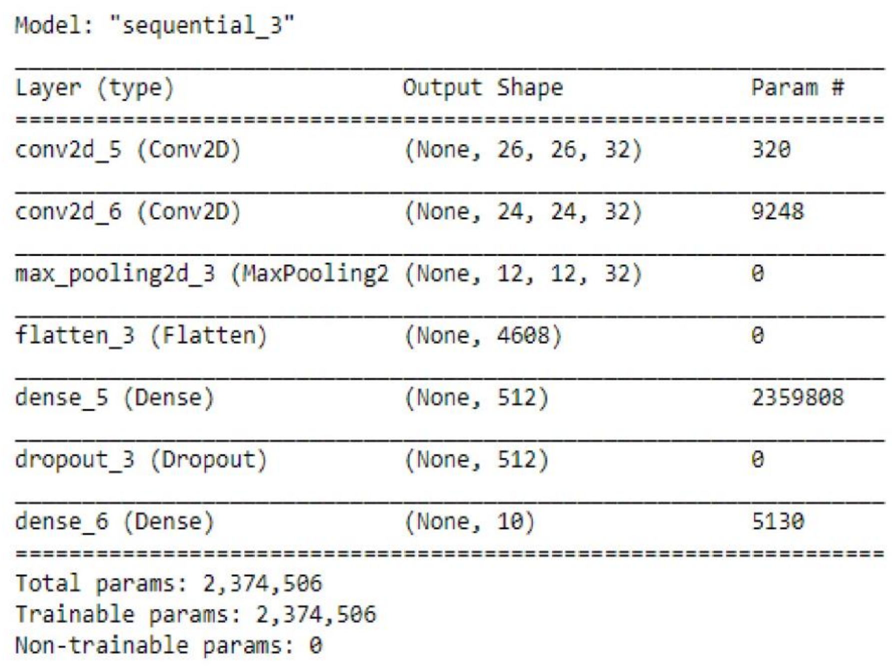
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
average_pooling2d_1 (Average (None, 12, 12, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2359808
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
dense_3 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,424,618
Trainable params: 3,424,618
Non-trainable params: 0
```

Figure 2 Neural Network-2

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='relu'))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(AveragePooling2D())

model.add(Flatten())

model.add(Dense(512, activation='relu')) # Additional layer with ‘relu’ activation function.

model.add(Dense(1024, activation='tanh'))# change from 512 to 1024 neurons

model.add(Dense(512, activation='sigmoid')) # change from 1024 to 512 neurons

model.add(Dropout(0.2))

model.add(Dense(num_classes, activation='softmax'))
```

Neural Network 3:

The whole code is same as Neural Network 1 but the number of layers are 7 in this Neural Network.



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 32)        9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 32)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 4608)              0         
_________________________________________________________________
dense_5 (Dense)              (None, 512)               2359808   
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130      
=================================================================
Total params: 2,374,506
Trainable params: 2,374,506
Non-trainable params: 0
```

Figure 3 Neural Network-3

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='relu'))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(num_classes, activation='softmax'))
```

Experiments and performances, with parameter settings

All the three neural networks are trained at 60000 samples and tested at 10000 samples. I changed two parameters to test the NN's: epoch and learning rate, to see the effects on the performance of the neural networks.

NN1:

With Epoch as 5 and learning rate as 0.0012, the accuracy of NN 1 is 0.7627 or 76.27% in first iteration, 0.8779 or 87.79% in second, 0.9020 or 90.20% in third, 0.9122 or 91.22% in forth, and 0.9179 or 91.79% in fifth iteration.

With the increase in the Epoch value, the accuracy of the model increased. For example, when I increase the Epoch of NN1 to 10 the accuracy increases to 0.9414 or 94.14% in the tenth iteration.

I changed the learning rate of the NN1 from 0.0012 to 0.00012 keeping Epoch as 5, to see the effect. It is evident that the change in learning rate caused the accuracy to drop to 0.1241 or 12.41% in the first iteration, followed by 0.1329 or 13.29% in second, 0.1593 or 15.93% in third, 0.3609 or 36.09% in fourth, 0.4815 or 48.15% in the fifth iteration.

I tried changing only the learning rate to 0.012 to see its effect and it turned out that the accuracy improved. The first iteration showed 0.9321 or 93.21% accuracy, 0.9427 or 94.27 in second, 0.9498 or 94.98% in third, 0.9512 or 95.12% in fourth, 0.9533 or 95.33% in fifth iteration.

NN2:

With Epoch as 5 and learning rate as 0.0012, the accuracy of NN 2 is 0.4828 or 48.28% in first iteration, 0.8546 or 85.46% in second, 0.8988 89.88% in third, 0.9116 or 91.16% in forth, and 0.9186 or 91.86% in fifth iteration.

With the increase in the Epoch value, the accuracy of the model increased. For example, when I increase the Epoch of NN1 to 7 the accuracy increases to 0.9404 or 94.04% in the seventh iteration.

I changed the learning rate of the NN1 from 0.0012 to 0.00012 keeping Epoch as 5, to see the effect. It is evident that the change in learning rate caused the accuracy to drop to 0.1135 or 11.35% in the first iteration, followed by 0.1135 or 11.35%, 0.1274 or 12.74% in third, 0.1142 or 11.42% in fourth, 0.1174 or 11.74% in fifth.

I tried changing only the learning rate to 0.012 to see its effect and it turned out that the accuracy improved. The first iteration showed 0.9381 or 93.81% accuracy, 0.9520 or 95.20 in second, 0.9567 or 95.67% in third, 0.9606 or 96.06% in fourth, 0.9617 or 96.17% in fifth iteration.

NN3:

With Epoch as 5 and learning rate as 0.0012, the accuracy of NN 3 is 0.9193 or 91.93% in first iteration, 0.8546 or 85.46% in second, 0.8988 89.88% in third, 0.9116 or 91.16% in forth, and 0.9186 or 91.86% in fifth iteration.

With the increase in the Epoch value, the accuracy of the model increased. For example, when I increase the Epoch of NN1 to 7 the accuracy increases to 0.9671 or 96.71% in the seventh iteration.

I changed the learning rate of the NN1 from 0.0012 to 0.00012 keeping Epoch as 5, to see the effect. It is evident that the change in learning rate caused the accuracy to drop to 0.2689 or 26.89% in the first iteration, followed by 0.5510 or 55.10% in second, 0.7209 or 72.09% in third, 0.8229 or 82.29% in fourth, 0.8669 or 86.69% in the fifth iteration.

I tried changing only the learning rate to 0.012 to see its effect and it turned out that the accuracy improved. The first iteration showed 0.9589 or 95.89% accuracy, 0.9659 or 96.59 in second, 0.9712 or 97.12 in third, 0.9728 or 97.28% in fourth, 0.9755 or 97.55% in fifth iteration.

Observations when changing parameters

With experiment, it is clear that the number of Epochs can increase the accuracy of the NN. In first one or two iteration, the accuracy can double but as the accuracy reaches its peak the number of iterations put a little or no effect on the accuracy of the NNs.

In the case of learning rate, the smaller learning rate can reduce the accuracy of the NNS. Whereas a large learning rate can increase the accuracy of the NNs which is evident in all three NNs I used.

Also adding fewer layer improved the performance of the NNs for this experiment. As in NN1 I used 9 layers and showed less accuracy as compared to NN3 which has 7 layers and showed more accuracy. Also, different activation functions work in different ways so, it depends on the problem of how many layers should be used.

Best performance of NNs

According to the experiments conducted by changing the parameters (Epoch and Learning rate), the NN3 performed the best followed by NN2 and NN1.