# assignment-4

February 7, 2024

Name : Naveen Kumar Khuntey (121CS0176)

**Question - 01)** Implement deep Multi Layer Perceptron(MLP) models with the following specifications using Tensorflow and Keras for classifying the MNIST dataset of handwritten digits. Train the model on the MNIST training set and evaluate its performance on the test set. Write modularized code and call it 10 times and compute the mean of test accuracy for each of the following 4 Sequential Models:

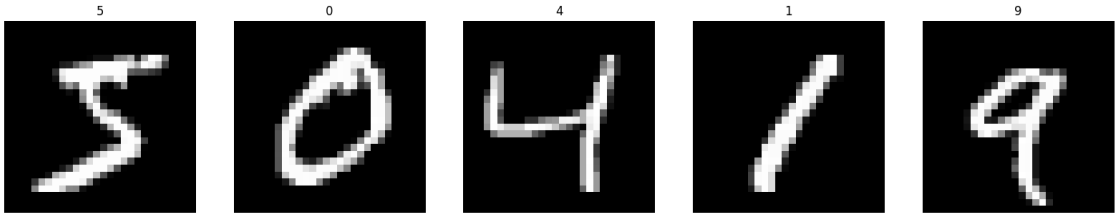
[2]:
```
# importing the libraries needed
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.activations import sigmoid, tanh, softmax, relu, selu
from tensorflow.keras.losses import MeanSquaredError,␣
 ↪SparseCategoricalCrossentropy
from tensorflow.keras.metrics import Accuracy
from tensorflow.keras.optimizers import SGD, Adam, AdamW, Nadam

from keras_tuner import RandomSearch, GridSearch, BayesianOptimization,␣
 ↪Hyperband
```

[3]:
```
# load the data from tensorflow.keras.datasets.mnist (handwritten digits datat)
(training_images, training_labels), (testing_images, testing_labels)= mnist.
 ↪load_data()
```

[4]:
```
# display a few images from the dataset
fig, axes = plt.subplots(1, 5, figsize=(20, 20))
for i in range(5):
    axes[i].imshow(training_images[i], cmap='gray')
    axes[i].set_title(training_labels[i])
    axes[i].axis('off')
plt.show()
```




[5]:
```
print("Training Image Shape: ", training_images.shape, "Training Label Shape:␣
 ↪", training_labels.shape)
print("Testing Image Shape: ", testing_images.shape, "Testing Label Shape: ",␣
 ↪testing_labels.shape)
```

```
Training Image Shape:  (60000, 28, 28) Training Label Shape:  (60000,)
Testing Image Shape:  (10000, 28, 28) Testing Label Shape:  (10000,)
```

[6]:
```
# flattening the images in the dataset from 28 * 28 to 784 * 1 for intuitive␣
 ↪computation
training_images = training_images.reshape(training_images.shape[0], 28*28).
 ↪astype('float32') / 255
testing_images = testing_images.reshape(testing_images.shape[0], 28*28).
 ↪astype('float32') / 255

training_images.shape, testing_images.shape
```

[6]:
```
((60000, 784), (10000, 784))
```

[7]:
```
# basic ANN model that takes in an optimizer argument and returns a compiled␣
 ↪model
def build_model(optimizer):
    model = keras.Sequential()
    model.add(Dense(128, activation='sigmoid', input_shape=(784, )))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='tanh'))
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(16, activation='selu'))
    model.add(Dropout(0.1))
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

[8]:
```
# using a subset of the training and testing samples to reduce training/␣
 ↪testing time
```

```
x_trn = training_images[:2000]
y_trn = training_labels[:2000]

x_tst = testing_images[:1000]
y_tst = testing_labels[:1000]
```

Model-1)
4 hidden layers having 128, 64, 32, 16 number of neurons respectively with activation function sigmoid, tanh, relu and selu respectively and dropout rate set to 0.5, 0.4, 0.3, 0.1 respectively. Use optimizer as SGD with batch size set to 32.

[10]:
```
num_calls = 10
accuracy_list = []

for i in range(num_calls):
    print("Call: ", i + 1)
    sgd = SGD()
    model = build_model(sgd)
    model.fit(x_trn, y_trn, epochs=5, batch_size=32)
    _, test_accuracy = model.evaluate(x_tst, y_tst, verbose=0)
    accuracy_list.append(test_accuracy)

mean_accuracy_sgd = np.mean(accuracy_list)
print("Mean Accuracy: ", mean_accuracy_sgd)
```

```
Call:  1
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 6.9580 - accuracy:
0.1200
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.4648 - accuracy:
0.1115
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.0428 - accuracy:
0.1175
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 3.7849 - accuracy:
0.1015
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.8274 - accuracy:
0.1100
Call:  2
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 6.6818 - accuracy:
0.0905
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.4854 - accuracy:
0.1080
```

```
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4575 - accuracy:
0.0985
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 3.8949 - accuracy:
0.1085
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.9663 - accuracy:
0.1030
Call:  3
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 8.0651 - accuracy:
0.0975
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 5.4683 - accuracy:
0.0890
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.2050 - accuracy:
0.1015
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 3.9517 - accuracy:
0.1115
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.8269 - accuracy:
0.1035
Call:  4
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 6.4253 - accuracy:
0.1025
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 5.4360 - accuracy:
0.1065
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.1930 - accuracy:
0.1050
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 4.1268 - accuracy:
0.1125
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.6685 - accuracy:
0.1080
Call:  5
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 7.3261 - accuracy:
0.0840
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 5.8022 - accuracy:
0.1025
```

```
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 5.2811 - accuracy:
0.0970
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 5.1848 - accuracy:
0.0950
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.9413 - accuracy:
0.0965
Call:  6
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 7.9530 - accuracy:
0.0900
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.9460 - accuracy:
0.0860
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.1675 - accuracy:
0.0910
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 4.0345 - accuracy:
0.0870
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 3.6458 - accuracy:
0.0975
Call:  7
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 10.7040 - accuracy:
0.0860
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 8.6208 - accuracy:
0.0905
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 7.7147 - accuracy:
0.0865
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 7.2063 - accuracy:
0.0920
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 6.7289 - accuracy:
0.0910
Call:  8
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 6.8017 - accuracy:
0.1025
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.4047 - accuracy:
0.1040
```

```
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.0064 - accuracy:
0.1165
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 4.0856 - accuracy:
0.1020
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.9721 - accuracy:
0.1075
Call:  9
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 7.5628 - accuracy:
0.0820
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.6748 - accuracy:
0.1025
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.0219 - accuracy:
0.0960
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 4.2366 - accuracy:
0.0900
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.9098 - accuracy:
0.0945
Call:  10
Epoch 1/5
63/63 [==============================] - 1s 2ms/step - loss: 7.9457 - accuracy:
0.0900
Epoch 2/5
63/63 [==============================] - 0s 2ms/step - loss: 4.9675 - accuracy:
0.1070
Epoch 3/5
63/63 [==============================] - 0s 2ms/step - loss: 4.2217 - accuracy:
0.1125
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 4.1597 - accuracy:
0.1145
Epoch 5/5
63/63 [==============================] - 0s 2ms/step - loss: 3.9113 - accuracy:
0.1060
Mean Accuracy:  0.10089999958872795
```

Model-2)
4 hidden layers having 128, 64, 32, 16 number of neurons respectively with activation function sigmoid, tanh, relu and selu respectively and dropout rate set to 0.5, 0.4, 0.3, 0.1 respectively. Use optimizer as Adam with batch size set to 32.

[11]:
```
num_calls = 10
accuracy_list = []

for i in range(num_calls):
    print("Call: ", i + 1)
    adam = Adam()
    model = build_model(adam)
    model.fit(x_trn, y_trn, epochs=5, batch_size=32)
    _, test_accuracy = model.evaluate(x_tst, y_tst, verbose=0)
    accuracy_list.append(test_accuracy)

mean_accuracy_adam = np.mean(accuracy_list)
print("Mean Accuracy: ", mean_accuracy_adam)
```

```
Call:  1
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 6.7303 - accuracy:
0.1010
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 5.1611 - accuracy:
0.1260
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4051 - accuracy:
0.1345
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 4.2771 - accuracy:
0.1545
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 3.8288 - accuracy:
0.1520
Call:  2
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 11.7946 - accuracy:
0.0700
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 9.6262 - accuracy:
0.1200
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 9.1017 - accuracy:
0.1300
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 9.1786 - accuracy:
0.1425
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 7.9791 - accuracy:
0.1400
Call:  3
```

```
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 9.8930 - accuracy:
0.1045
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 7.5519 - accuracy:
0.1300
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 5.9934 - accuracy:
0.1455
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 4.8160 - accuracy:
0.1365
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4403 - accuracy:
0.1315
Call:  4
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 9.1525 - accuracy:
0.1005
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 6.8680 - accuracy:
0.1185
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 6.1767 - accuracy:
0.1175
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 5.9363 - accuracy:
0.1170
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 5.4261 - accuracy:
0.1295
Call:  5
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 8.9967 - accuracy:
0.1195
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 7.2433 - accuracy:
0.1325
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 5.7100 - accuracy:
0.1575
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 5.2913 - accuracy:
0.1495
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.6845 - accuracy:
0.1460
Call:  6
```

```
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 8.4171 - accuracy:
0.0890
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 6.1175 - accuracy:
0.1180
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 4.9642 - accuracy:
0.1140
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 5.0171 - accuracy:
0.1115
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.5389 - accuracy:
0.1250
Call:  7
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 9.0586 - accuracy:
0.0950
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 7.2870 - accuracy:
0.0965
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 6.2759 - accuracy:
0.1115
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 5.6666 - accuracy:
0.1230
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.7109 - accuracy:
0.1245
Call:  8
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 11.1742 - accuracy:
0.0940
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 6.2276 - accuracy:
0.1045
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4567 - accuracy:
0.0925
Epoch 4/5
63/63 [==============================] - 0s 2ms/step - loss: 3.8363 - accuracy:
0.0675
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 3.5988 - accuracy:
0.0830
Call:  9
```

```
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 8.6804 - accuracy:
0.0965
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 6.4999 - accuracy:
0.1195
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 5.4419 - accuracy:
0.1250
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 5.0422 - accuracy:
0.1290
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4898 - accuracy:
0.1175
Call:  10
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 8.0415 - accuracy:
0.0955
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 5.9525 - accuracy:
0.1040
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 5.1284 - accuracy:
0.1255
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 4.3904 - accuracy:
0.1200
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 4.4721 - accuracy:
0.1185
Mean Accuracy:  0.16389999836683272
```

Model-3)
4 hidden layers having 128, 64, 32, 16 number of neurons respectively with activation function sigmoid, tanh, relu and selu respectively and dropout rate set to 0.5, 0.4, 0.3, 0.1 respectively. Use optimizer as AdamW with learning rate 0.1 with batch size set to 32.

[12]:
```python
num_calls = 10
accuracy_list = []

for i in range(num_calls):
    print("Call: ", i + 1)
    adamw = AdamW(learning_rate=0.1)
    model = build_model(adamw)
    model.fit(x_trn, y_trn, epochs=5, batch_size=32)
    _, test_accuracy = model.evaluate(x_tst, y_tst, verbose=0)
    accuracy_list.append(test_accuracy)
```

```
mean_accuracy_adamw = np.mean(accuracy_list)
print("Mean Accuracy: ", mean_accuracy_adamw)
```

```
Call:  1
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.9598 - accuracy:
0.0885
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7859 - accuracy:
0.0905
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7859 - accuracy:
0.0925
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0780
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0850
Call:  2
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.4980 - accuracy:
0.0820
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8756 - accuracy:
0.0795
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8766 - accuracy:
0.0880
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8366 - accuracy:
0.0830
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8991 - accuracy:
0.0805
Call:  3
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 4.2030 - accuracy:
0.0860
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8432 - accuracy:
0.0890
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8167 - accuracy:
0.0925
Epoch 4/5
```

```
63/63 [==============================] - 0s 3ms/step - loss: 2.8250 - accuracy:
0.0800
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7906 - accuracy:
0.0850
Call:  4
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.2184 - accuracy:
0.0870
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7957 - accuracy:
0.0875
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7993 - accuracy:
0.0785
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8158 - accuracy:
0.0850
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7993 - accuracy:
0.0775
Call:  5
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 12.6295 - accuracy:
0.1070
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 13.3819 - accuracy:
0.1080
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 13.2549 - accuracy:
0.1195
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 13.4570 - accuracy:
0.1000
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 13.2458 - accuracy:
0.1075
Call:  6
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 9.7769 - accuracy:
0.0920
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 3.3365 - accuracy:
0.0875
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 3.1301 - accuracy:
0.0780
Epoch 4/5
```

```
63/63 [==============================] - 0s 3ms/step - loss: 2.7923 - accuracy:
0.0830
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7918 - accuracy:
0.0905
Call:  7
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.8606 - accuracy:
0.0815
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8622 - accuracy:
0.0840
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7857 - accuracy:
0.0790
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0840
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0895
Call:  8
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 6.0865 - accuracy:
0.1010
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 3.0752 - accuracy:
0.0815
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8172 - accuracy:
0.0795
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8060 - accuracy:
0.0765
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8112 - accuracy:
0.0835
Call:  9
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.2267 - accuracy:
0.0790
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8130 - accuracy:
0.0875
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.9941 - accuracy:
0.0845
Epoch 4/5
```

```
63/63 [==============================] - 0s 3ms/step - loss: 2.8825 - accuracy:
0.0830
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8066 - accuracy:
0.0935
Call:  10
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 4.3780 - accuracy:
0.0840
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8056 - accuracy:
0.0785
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8056 - accuracy:
0.0820
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7859 - accuracy:
0.0740
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7786 - accuracy:
0.0760
Mean Accuracy:  0.08750000074505807
```

Model-4)
4 hidden layers having 128, 64, 32, 16 number of neurons respectively with activation function sigmoid, tanh, relu and selu respectively and dropout rate set to 0.5, 0.4, 0.3, 0.1 respectively. Use optimizer as Nadam with learning rate 0.1 with batch size set to 32.

[13]:
```
num_calls = 10
accuracy_list = []

for i in range(num_calls):
    print("Call: ", i + 1)
    nadam = Nadam(learning_rate=0.1)
    model = build_model(nadam)
    model.fit(x_trn, y_trn, epochs=5, batch_size=32)
    _, test_accuracy = model.evaluate(x_tst, y_tst, verbose=0)
    accuracy_list.append(test_accuracy)

mean_accuracy_nadam = np.mean(accuracy_list)
print("Mean Accuracy: ", mean_accuracy_nadam)
```

```
Call:  1
Epoch 1/5
63/63 [==============================] - 2s 3ms/step - loss: 4.7437 - accuracy:
0.0905
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8258 - accuracy:
```

```
0.0880
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0765
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0795
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7788 - accuracy:
0.0935
Call:  2
Epoch 1/5
63/63 [==============================] - 2s 3ms/step - loss: 3.9074 - accuracy:
0.0720
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7923 - accuracy:
0.0790
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0885
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0760
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0835
Call:  3
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.5671 - accuracy:
0.0885
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7986 - accuracy:
0.0905
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7790 - accuracy:
0.0840
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0855
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0825
Call:  4
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.9215 - accuracy:
0.0805
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
```

```
0.0865
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7911 - accuracy:
0.0885
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7924 - accuracy:
0.0870
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7848 - accuracy:
0.0865
Call:  5
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.3749 - accuracy:
0.0805
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0805
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0825
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0785
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0785
Call:  6
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.6649 - accuracy:
0.0850
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8121 - accuracy:
0.0840
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7924 - accuracy:
0.0865
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0800
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0875
Call:  7
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.7036 - accuracy:
0.0800
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7789 - accuracy:
```

```
0.0865
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0750
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0685
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0840
Call:  8
Epoch 1/5
63/63 [==============================] - 1s 3ms/step - loss: 3.9619 - accuracy:
0.0820
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7793 - accuracy:
0.0860
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7849 - accuracy:
0.0875
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7796 - accuracy:
0.0800
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7933 - accuracy:
0.0825
Call:  9
Epoch 1/5
63/63 [==============================] - 2s 3ms/step - loss: 3.4367 - accuracy:
0.0950
Epoch 2/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0870
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0900
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0880
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0750
Call:  10
Epoch 1/5
63/63 [==============================] - 2s 3ms/step - loss: 3.7611 - accuracy:
0.0925
Epoch 2/5
63/63 [==============================] - 0s 4ms/step - loss: 2.8986 - accuracy:
```

```
0.0840
Epoch 3/5
63/63 [==============================] - 0s 3ms/step - loss: 2.8191 - accuracy:
0.0750
Epoch 4/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7859 - accuracy:
0.0905
Epoch 5/5
63/63 [==============================] - 0s 3ms/step - loss: 2.7726 - accuracy:
0.0780
Mean Accuracy:  0.08500000089406967
```

**Question - 02)** Tune the hyperparameters using kerastuner to select the best learning rate among the set {0.1, 0.01, 0.15} with batch size varying between {4,8,16} and first hidden layer neurons varying between 250 to 260 with a step value of 2. 2nd, 3rd and 4th hidden layer contains 16, 8, 4 numbers of neurons respectively. The four layers haveactivation function sigmoid, tanh, relu and selu respectively. Use optimizer as SGD and find the best hyperparameters to predict the MNIST test data.

[14]:
```
def build_model(hp):
    hp_neurons = hp.Int('neurons', min_value=250, max_value=260, step=2)
    # structure of the ANN
    model = keras.Sequential()
    model.add(Dense(hp_neurons, activation='sigmoid', input_shape=(784, )))
    model.add(Dropout(0.5))
    model.add(Dense(16, activation='tanh'))
    model.add(Dropout(0.4))
    model.add(Dense(8, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(4, activation='selu'))
    model.add(Dropout(0.1))
    model.add(Dense(10, activation='softmax'))
    # parameters for tuning learning_rate and batch_sizes
    hp_learning_rate = hp.Choice('learning_rate', values=[0.1, 0.01, 0.15])
    hp_batch_size = hp.Choice('batch_size', values=[4, 8, 16])
    sgd = SGD(learning_rate=hp_learning_rate)
    model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

[17]:
```
# configure the tuner
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=1,
    directory='project',
```

```
    project_name='mnist'
)
```

[18]:
```
hp_batch_size = tuner.oracle.get_space()['batch_size']
tuner.search(x=x_trn, y=y_trn, epochs=5, batch_size=hp_batch_size,␣
 ↪validation_data=(x_tst, y_tst))
```

```
Trial 10 Complete [00h 00m 08s]
val_accuracy: 0.12600000202655792

Best val_accuracy So Far: 0.23800000548362732
Total elapsed time: 00h 01m 20s
```

[20]:
```
# get the best set of hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_neuron = best_hps.get('neurons')
best_learning_rate = best_hps.get('learning_rate')
best_batch_size = best_hps.get('batch_size')

print(f"Best Neurons: {best_neuron}")
print(f"Best Learning Rate: {best_learning_rate}")
print(f"Best Batch Size: {best_batch_size}")

# get the best model
best_model = tuner.get_best_models(num_models=1)[0]

# Evaluate the best model
loss, accuracy = best_model.evaluate(x_tst, y_tst)
print(f"Accuracy of best model: {accuracy}")
```

```
Best Neurons: 260
Best Learning Rate: 0.01
Best Batch Size: 8
32/32 [==============================] - 0s 2ms/step - loss: 2.2294 - accuracy:
0.2380
Accuracy of best model: 0.23800000548362732
```

[21]:
```
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 260)               204100

 dropout (Dropout)           (None, 260)               0

 dense_1 (Dense)             (None, 16)                4176
```

```
 dropout_1 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 8)                 136

 dropout_2 (Dropout)         (None, 8)                 0

 dense_3 (Dense)             (None, 4)                 36

 dropout_3 (Dropout)         (None, 4)                 0

 dense_4 (Dense)             (None, 10)                50

=================================================================
Total params: 208498 (814.45 KB)
Trainable params: 208498 (814.45 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```