Exercise sheet 9 - Introduction - Due date: June 29th

Submitted to:

Mohamed Aborageh : s0moabor@uni-bonn.de

Vinay Srinivas Bharadhwaj: s0vibhar@uni-bonn.de

Yasamin Salimi: yasisali@uni-bonn.de

Exercise 1 - Basics of NN (9 points)

From the MNIST database load the handwritten digits dataset.

1. Normalize your dataset before training your model. (1 point)

In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

In [5]:
```
mnist = tf.keras.datasets.mnist #28*28 image of handwritten of 0-9
(x_train, y_train),(x_test,y_test) = mnist.load_data()
```

In [ ]:
```
x_train = tf.keras.utils.normalize(x_train, axis = 1)
x_test = tf.keras.utils.normalize(x_test,axis = 1)

print("Training Data after normalizing is {}".format(x_train[0]))
print("Testing  Data after normalizing is {}".format(x_test[0]))
```

```
Training Data after normalizing is [[0.         0.         0.         0.         0.
0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
  0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
```

```
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
  0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
  0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
  0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
  0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.04500225 0.4219755  0.45852825 0.43408872 0.37314701
  0.33153488 0.32790981 0.28826244 0.26543758 0.34149427 0.31128482
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.1541463  0.28272888 0.18358693 0.37314701
  0.33153488 0.26569767 0.01601458 0.         0.05945042 0.19891229
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.0253731  0.00171577 0.22713296
  0.33153488 0.11664776 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.20500962
  0.33153488 0.24625638 0.00291174 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.01622378
  0.24897876 0.32790981 0.10191096 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.04586451 0.31235677 0.32757096 0.23335172 0.14931733 0.00129164
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.10498298 0.34940902 0.3689874  0.34978968 0.15370495
  0.04089933 0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.06551419 0.27127137 0.34978968 0.32678448
  0.245396   0.05882702 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.02333517 0.12857881 0.32549285
  0.41390126 0.40743158 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.32161793
  0.41390126 0.54251585 0.20001074 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.06697006 0.18959827 0.25300993 0.32678448
  0.41390126 0.45100715 0.00625034 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.05110617 0.19182076 0.33339444 0.3689874  0.34978968 0.32678448
```

```
 0.40899334 0.39653769 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.04117838 0.16813739
 0.28960162 0.32790981 0.36833534 0.3689874  0.34978968 0.25961929
 0.12760592 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.04431706 0.11961607 0.36545809 0.37314701
 0.33153488 0.32790981 0.36833534 0.28877275 0.111988   0.00258328
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.05298497 0.42752138 0.4219755  0.45852825 0.43408872 0.37314701
 0.33153488 0.25273681 0.11646967 0.01312603 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.37491383 0.56222061
 0.66525569 0.63253163 0.48748768 0.45852825 0.43408872 0.359873
 0.17428513 0.01425695 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.92705966 0.82698729
 0.74473314 0.63253163 0.4084877  0.24466922 0.22648107 0.02359823
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]]
Testing  Data after normalizing is [[0.         0.         0.         0.         0.
0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.34058377 0.55344342 0.51591571 0.47675838 0.16790986 0.06389561
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.90011425 0.75986285 0.82416724 0.80196443 0.71081842 0.42774558
 0.31460214 0.29919608 0.35451095 0.35818467 0.34876618 0.33626817
 0.34967436 0.335178   0.37058415 0.28257531 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.2716561  0.34104081 0.23362221 0.35993679 0.45615513 0.40289729
 0.40358052 0.33999555 0.45477668 0.45948942 0.44740712 0.42458103
 0.40442136 0.42997581 0.55369632 0.76077969 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.03017292
 0.10486738 0.02115528 0.11996078 0.1212039  0.11801684 0.10020112
 0.03708667 0.39950509 0.55369632 0.57601891 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.14658067 0.42828299 0.45560051 0.09781453 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.03736313
 0.41148549 0.43166863 0.18093226 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.21908381
 0.44857216 0.40289072 0.0959159  0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.10392528 0.4228827
 0.44857216 0.10495473 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.23427223 0.43137432
 0.33024801 0.00846409 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.01628112 0.3610963  0.42118438
 0.10242986 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.2279357  0.44740712 0.30909499
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.13428445 0.45406238 0.42274688 0.09680447
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

```
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.02871073 0.39569152 0.45948942 0.29239993 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.0047667  0.30675154 0.45477668 0.39617395 0.06165059 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.06037819 0.38381719 0.45477668 0.13929404 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.05502122
  0.35591353 0.38381719 0.20590283 0.00180901 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.23605877
  0.40358052 0.38381719 0.09310389 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.17070836 0.42952046
  0.40358052 0.38381719 0.09310389 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.33861822 0.450819
  0.40358052 0.330929   0.07161837 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.33861822 0.450819
  0.32890224 0.02719964 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```

1. Train a neural network once using Adam and once using AdaGrad optimizer. Hint: Set epochs = 20, neurons of hidden layer = 100, activation function = ReLU for reproducibility. (2 points)

In [ ]:

```
model_adam = tf.keras.models.Sequential()  # a basic feed-forward model
model_adam.add(tf.keras.layers.Flatten())  # takes our 28x28 and makes it 1x784
model_adam.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))  # a simple fully-
model_adam.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))  # a simple fully-
model_adam.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))  # our output la

model_adam.compile(optimizer='adam',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "
              metrics=['accuracy'])  # what to track
```

In [ ]:

```
adam=model_adam.fit(x_train, y_train, epochs=20)  # train the model
```

```
Epoch 1/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2838 - accuracy:
0.9168
Epoch 2/20
```

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1198 - accuracy: 
0.9633
Epoch 3/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0816 - accuracy: 
0.9746
Epoch 4/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0619 - accuracy: 
0.9798
Epoch 5/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0472 - accuracy: 
0.9850
Epoch 6/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0374 - accuracy: 
0.9878
Epoch 7/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0309 - accuracy: 
0.9894
Epoch 8/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0253 - accuracy: 
0.9914
Epoch 9/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0199 - accuracy: 
0.9936
Epoch 10/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0195 - accuracy: 
0.9930
Epoch 11/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0142 - accuracy: 
0.9954
Epoch 12/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0140 - accuracy: 
0.9953
Epoch 13/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0120 - accuracy: 
0.9960
Epoch 14/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0116 - accuracy: 
0.9960
Epoch 15/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0102 - accuracy: 
0.9965
Epoch 16/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0096 - accuracy: 
0.9966
Epoch 17/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0088 - accuracy: 
0.9970
Epoch 18/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0091 - accuracy: 
0.9966
Epoch 19/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0068 - accuracy: 
0.9977
Epoch 20/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0091 - accuracy: 
0.9970
```

In [ ]:

```
model_grad = tf.keras.models.Sequential()  # a basic feed-forward model
model_grad.add(tf.keras.layers.Flatten())  # takes our 28x28 and makes it 1x784
model_grad.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))  # a simple fully-
model_grad.add(tf.keras.layers.Dense(100, activation=tf.nn.relu))  # a simple fully-
model_grad.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))  # our output la

model_grad.compile(optimizer='adagrad',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "
              metrics=['accuracy'])  # what to track

adagrad=model_grad.fit(x_train, y_train, epochs=20)  # train the model
```

```
Epoch 1/20
1875/1875 [==============================] - 3s 2ms/step - loss: 1.9802 - accuracy:
0.4672
Epoch 2/20
1875/1875 [==============================] - 3s 2ms/step - loss: 1.1274 - accuracy:
0.7869
Epoch 3/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7154 - accuracy:
0.8376
Epoch 4/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5700 - accuracy:
0.8589
Epoch 5/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4979 - accuracy:
0.8721
Epoch 6/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4543 - accuracy:
0.8798
Epoch 7/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4248 - accuracy:
0.8855
Epoch 8/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4032 - accuracy:
0.8898
Epoch 9/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3865 - accuracy:
0.8938
Epoch 10/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3730 - accuracy:
0.8968
Epoch 11/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3620 - accuracy:
0.8993
Epoch 12/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3527 - accuracy:
0.9015
Epoch 13/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3446 - accuracy:
0.9037
Epoch 14/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3376 - accuracy:
0.9050
Epoch 15/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3312 - accuracy:
0.9064
Epoch 16/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3256 - accuracy:
0.9078
Epoch 17/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3204 - accuracy:
0.9093
Epoch 18/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3157 - accuracy:
0.9104
Epoch 19/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3114 - accuracy:
0.9116
Epoch 20/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3072 - accuracy:
0.9129
```

1. Plot the SparseCategoricalCrossentropy loss for both models. Plot the computed accuracy for both models. Which model performed better while training? (2 points)

In [ ]:

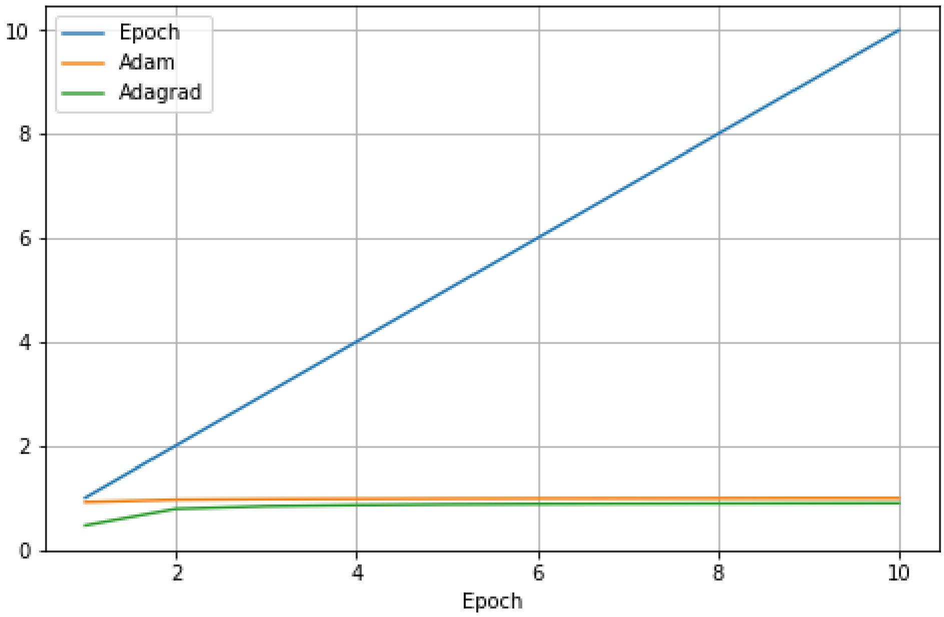
```
# plot accuracy

import pandas as pd
import matplotlib.pyplot as plt
```

```
ep=np.arange(1,11,1)
ada=adam.history['accuracy']
adagrd=adagrad.history['accuracy']
list_of_tuples = list(zip(ep,ada,adagrd ))

df = pd.DataFrame(list_of_tuples, columns = ['Epoch','Adam','Adagrad'])
df.index = df['Epoch']
df.plot(figsize=(8, 5))
plt.grid(True)

plt.show()
```



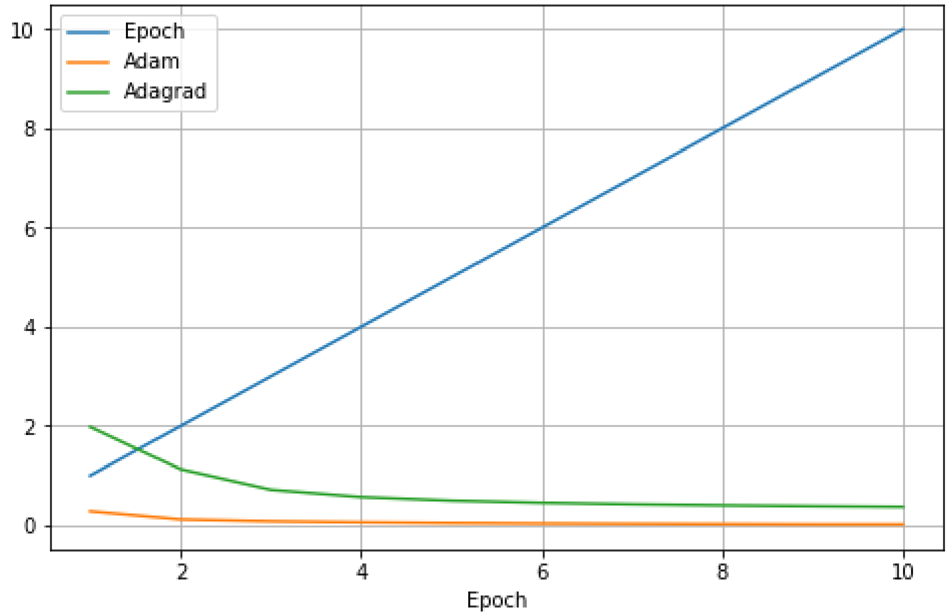
In [ ]:

```
# plot loss
ep=np.arange(1,11,1)
ada=adam.history['loss']
adagrd=adagrad.history['loss']
list_of_tuples = list(zip(ep,ada,adagrd ))

df = pd.DataFrame(list_of_tuples, columns = ['Epoch','Adam','Adagrad'])

df.index = df['Epoch']
df.plot(figsize=(8, 5))
plt.grid(True)

plt.show()
```



1. Compute the model accuracy on the test set for both optimizers. Which model performed better? (1 point)

In [ ]:
```
#adam
val_loss, val_acc = model_adam.evaluate(x_test, y_test)
print(val_loss)  # model's loss (error)
print(val_acc)  # model's accuracy
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.1429 - accuracy: 0.
9748
0.14294207096099854
0.9747999906539917
```

In [ ]:
```
#adagrad
val_loss, val_acc = model_grad.evaluate(x_test, y_test)
print(val_loss)  # model's loss (error)
print(val_acc)  # model's accuracy
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.2975 - accuracy: 0.
9179
0.29745998978614807
0.917900025844574
```

1. Familiarize yourself with Layer Normalization and explain how it works. (1 point)

- layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization
- layer normalization is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer normalization can substantially reduce the training time compared with previously published techniques.
- Layer Normalization directly estimates the normalization statistics from the summed inputs to the neurons within a hidden layer so the normalization does not introduce any new dependencies between training cases

In [ ]:
```
```

1. Using the same dataset to train a neural network with Layer Normalization. Hint: Set epochs

= 20, neurons of hidden layer = 100, activation function = ReLU for reproducibility.

a. Compute the SparseCategoricalCrossentropy loss and model accuracy. (1 point)

b. Evaluate the model performance using the test dataset. (1 point)

In [ ]:

In [ ]:

Exercise 2 - Hyper Parameter Optimization (9 points)

1. What are the main challenges with hyper-parameter optimization for neural networks? (1 point)

Challenges with hyper-parameter optimization for neural networks:

1.) Overfitting is the issue in which our model performs extremely well during training and optimization, and very poorly out of sample

2.)It is recommended that you optimize all hyperparameters of your model, including architecture parameters and model parameters, at the same time.

3.) One of the drawbacks of grid search is that when it comes to dimensionality, it suffers when evaluating the number of hyperparameters grows exponentially. However, there is no guarantee that the search will produce the perfect solution, as it usually finds one by aliasing around the right set.

In [ ]:

1. Inform yourself about variants of Bayesian-HPO and explain them in detail (2 points)

The aim of Bayesian reasoning is to become “less wrong” with more data which these approaches do by continually updating the surrogate probability model after each evaluation of the objective function. The basic idea is: spend a little more time selecting the next hyperparameters in order to make fewer calls to the objective function.

Sequential model-based optimization (SMBO) methods (SMBO) are a formalization of Bayesian optimization. The sequential refers to running trials one after another, each time trying better hyperparameters by applying Bayesian reasoning and updating a probability model (surrogate).

There are five aspects of model-based hyperparameter optimization:

A domain of hyperparameters over which to search An objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize) The surrogate model of the objective function A criteria, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model

A history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model

In [ ]:

1. Using the same MNIST dataset, optimize the activation function for the output layer and the number of dropout units in the NN model using the following methods. (6 points)

a. Grid search

In [6]:
```
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from skopt import BayesSearchCV
```

In [7]:
```
def build_model(activation,dropout_rate):
    model_adam = tf.keras.models.Sequential()
    model_adam.add(tf.keras.layers.Flatten())
    model_adam.add(tf.keras.layers.Dense(100, activation=activation))
    model_adam.add(tf.keras.layers.Dense(100, activation=activation))
    model_adam.add(tf.keras.layers.Dense(10, activation=activation))
    model_adam.add(tf.keras.layers.Dropout(dropout_rate))
    model_adam.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model_adam

model = KerasClassifier(build_fn = build_model, verbose=0)
parameters = {'dropout_rate' : [0.0, 0.1, 0.2, 0.3, 0.4, 0.5],'activation' : ['relu'
```

In [8]:
```
models = GridSearchCV(estimator = model, param_grid = parameters, n_jobs=1,scoring="
best_model = models.fit(x_train,y_train)
print("Best parameters by GridSearchCV:", best_model.best_params_)
```

```
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/keras/eng
ine/sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and will
be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=
-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax`
last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model
does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
Best parameters by GridSearchCV: {'activation': 'sigmoid', 'dropout_rate': 0.0}
```

b. Random search

In [9]:
```
random_search = RandomizedSearchCV(estimator=model, param_distributions=parameters,n
random_search.fit(x_train,y_train)
print("Best parameters by Random search:",random_search.best_params_)
```

```
Best parameters by Random search: {'dropout_rate': 0.1, 'activation': 'sigmoid'}
```

c. Bayesian Hyper-parameter optimization

In [10]:
```
search = BayesSearchCV(estimator=model, search_spaces=parameters, n_jobs=-1)
search.fit(x_train,y_train)
print("Best parameters by Bayesian Hyper-parameter optimization:", search.best_param
```

```
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
```

```
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
```

```
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
/Users/rohitha/opt/anaconda3/lib/python3.8/site-packages/skopt/optimizer/optimizer.p
y:449: UserWarning: The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Best parameters by Bayesian Hyper-parameter optimization: OrderedDict([('activatio
n', 'sigmoid'), ('dropout_rate', 0.0)])
```

In [ ]:

```

```

Exercise 3 - Transfer Learning & CNNs (7 points)

1. Load the VGG16 pre-trained model using Keras Applications API. Use the model to classify the dog images in canines.zip after pre-processing each image by doing the following: (2 points)

a. Load each image and set the size to 224 x 224 pixels

b. Convert the image pixels to a numpy array and reshape it according to the model's input requirements

c. Use the model to print out the predicted class and its probability for each image

In [17]:

```
import numpy as np
from keras.models import Model
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
import tensorflow.keras as keras
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGe

import os
```

In [18]:

```
model = VGG16()
print(model.summary())
```

```
Model: "vgg16"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544 
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312  
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000   
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
None
```

In [19]:
```
#Top 1% of the probability of classes
```

In [20]:
```
for file in os.listdir("Canines"):
    print(file)
    img_path = 'Canines/'+file
    print(img_path)
    image = load_img(img_path, target_size=(224,224,3))
    image = img_to_array(image)
    image= image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    y_pred = model.predict(image)
    label = decode_predictions(y_pred, top=1)
```

```
    print(label)
    print()
```

```
dog5.jpg
Canines/dog5.jpg
[[('n02109047', 'Great_Dane', 0.45382115)]]

dog4.jpg
Canines/dog4.jpg
[[('n02106166', 'Border_collie', 0.735858)]]

dog6.jpg
Canines/dog6.jpg
[[('n02109047', 'Great_Dane', 0.9088881)]]

dog7.jpg
Canines/dog7.jpg
[[('n02109961', 'Eskimo_dog', 0.49961603)]]

dog3.jpg
Canines/dog3.jpg
[[('n02099601', 'golden_retriever', 0.78778)]]

dog2.jpg
Canines/dog2.jpg
[[('n02113023', 'Pembroke', 0.7457447)]]

dog1.jpg
Canines/dog1.jpg
[[('n02107142', 'Doberman', 0.93426067)]]

dog8.jpg
Canines/dog8.jpg
[[('n02107142', 'Doberman', 0.35419115)]]
```

In [21]:

```
#Top 2% of the probability of classes
```

In [22]:

```
for file in os.listdir("Canines"):
    print(file)
    img_path = 'Canines/'+file
    print(img_path)
    image = load_img(img_path, target_size=(224,224,3))
    image = img_to_array(image)
    image= image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    y_pred = model.predict(image)
    label = decode_predictions(y_pred, top=2)
    print(label)
    print()
```

```
dog5.jpg
Canines/dog5.jpg
[[('n02109047', 'Great_Dane', 0.45382115), ('n02108422', 'bull_mastiff', 0.4368719
8)]]

dog4.jpg
Canines/dog4.jpg
[[('n02106166', 'Border_collie', 0.735858), ('n02106030', 'collie', 0.23651241)]]

dog6.jpg
Canines/dog6.jpg
[[('n02109047', 'Great_Dane', 0.9088881), ('n02087394', 'Rhodesian_ridgeback', 0.074
75051)]]

dog7.jpg
Canines/dog7.jpg
[[('n02109961', 'Eskimo_dog', 0.49961603), ('n02110185', 'Siberian_husky', 0.290228
```

```
9)]]

dog3.jpg
Canines/dog3.jpg
[[('n02099601', 'golden_retriever', 0.78778), ('n02104029', 'kuvasz', 0.022645768)]]

dog2.jpg
Canines/dog2.jpg
[[('n02113023', 'Pembroke', 0.7457447), ('n02113186', 'Cardigan', 0.2151999)]]

dog1.jpg
Canines/dog1.jpg
[[('n02107142', 'Doberman', 0.93426067), ('n02087046', 'toy_terrier', 0.048935466)]]

dog8.jpg
Canines/dog8.jpg
[[('n02107142', 'Doberman', 0.35419115), ('n02105412', 'kelpie', 0.19439045)]]
```

In [ ]:

1. Downscale the given matrix by applying the following pooling operations: a. Max Pool (1 point) b. Average Pool (1 point)

In [23]:

```
import tensorflow as tf


tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid')

Matrix = tf.constant([[1., 4., 1., 5.],
                     [4., 9., 4., 8.],
                     [4., 5., 4., 3.],
                     [6., 5., 7., 4.]])
Matrix = tf.reshape(Matrix, [1, 4, 4, 1])
max_pool_2d = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
   strides=(1, 1), padding='valid')
max_pool_2d(Matrix)
```

Out[23]:

```
<tf.Tensor: shape=(1, 3, 3, 1), dtype=float32, numpy=
array([[[[9.],
         [9.],
         [8.]],

        [[9.],
         [9.],
         [8.]],

        [[6.],
         [7.],
         [7.]]]], dtype=float32)>
```

In [24]:

```
tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid')

Matrix = tf.constant([[1., 4., 1., 5.],
                     [4., 9., 4., 8.],
                     [4., 5., 4., 3.],
                     [6., 5., 7., 4.]])
Matrix = tf.reshape(Matrix, [1, 4, 4, 1])
avg_pool_2d = tf.keras.layers.AveragePooling2D(pool_size=(2, 2),
   strides=(1, 1), padding='valid')
avg_pool_2d(Matrix)
```

Out[24]:

```
<tf.Tensor: shape=(1, 3, 3, 1), dtype=float32, numpy=
array([[[[4.5 ],
         [4.5 ],
```

```
          [4.5 ]],

         [[5.5 ],
          [5.5 ],
          [4.75]],

         [[5.  ],
          [5.25],
          [4.5 ]]]], dtype=float32)>
```

In [ ]:

```
```

In [ ]:

```
```

1. Load the CIFAR10 dataset using Keras datasets API and normalize the images' pixel values. Train a convolutional neural network to classify the dataset images with the following architecture: (3 points)

a. Convolutional Base:

i. An input convolution layer with 32 filters and a kernel size of (3,3).

Adjust your input shape to that of the CIFAR images' format

ii. 2 convolution layers, each with 64 filters and a kernel size of (3,3)

iii. 2 Max Pool layers, with a pool size of 2x2

b. 2 dense layers, with 64 and 10 units respectively. Adjust the output of the convolutional base such that it satisfies the input requirements of the dense layers.

c. Use the following parameters to train the network: i. Sparse categorical cross entropy as your loss function ii. Adam optimizer iii. 10 epochs iv. ReLU activation for your layers

Compile your model, then plot the accuracy across each epoch

In [ ]:

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
from keras.callbacks import Callback
from keras.utils import np_utils
import numpy as np
import matplotlib.pyplot as plt
from keras import backend as K
K.set_image_data_format('channels_last')
```

In [ ]:

```
data = tf.keras.datasets.cifar10.load_data()
(Xtrain, ytrain), (Xtest, ytest) = data
Xtrain = Xtrain.astype('float32')
Xtest = Xtest.astype('float32')
```

In [ ]:

```
#Normalize image pixels

mean = np.mean(Xtrain, axis = (0,1,2,3))
std = np.std(Xtrain, axis = (0,1,2,3))
Xtrain = (Xtrain - mean) / (std +1e-7)
Xtest = (Xtest - mean) / (std +1e-7)
```

In [ ]:

```
n_classes = 10
ytrain = np_utils.to_categorical(ytrain, n_classes)
```

```
ytest = np_utils.to_categorical(ytest, n_classes)

input_shape = (32, 32, 3)
```

In [ ]:
```
def CNN():
    '''Function to create model with given layers for CNN training'''
    model = Sequential([
    Conv2D(32, (3,3,), padding = 'same', activation = 'relu', input_shape = input_sh
    Conv2D(64, (3,3,), activation = 'relu'),
    Conv2D(64, (3,3,), activation = 'relu'),
    MaxPooling2D(pool_size = (2,2)),
    MaxPooling2D(pool_size = (2,2)),

    Flatten(),
    Dense(64, activation = 'relu'),
    Dense(10, activation = 'softmax')])

    return model
```

In [ ]:
```
K.clear_session()
model = CNN()

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 64)                200768
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 257,738
Trainable params: 257,738
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
from keras.callbacks import TensorBoard
tbCallBack = TensorBoard(log_dir='./log', histogram_freq=1, write_graph=True,
                         write_images=True)


epochs = 10
starttime = datetime.datetime.now()

history = model.fit(Xtrain, ytrain, batch_size=32,
                    epochs=epochs, verbose=1,
                    validation_data=(Xtest, ytest))

endtime = datetime.datetime.now()
print (endtime - starttime)
```

```
Epoch 1/10
1563/1563 [==============================] - 483s 309ms/step - loss: 0.3395 - accura
cy: 0.8792 - val_loss: 1.0522 - val_accuracy: 0.7217
Epoch 2/10
1563/1563 [==============================] - 443s 284ms/step - loss: 0.2933 - accura
cy: 0.8950 - val_loss: 1.2021 - val_accuracy: 0.7061
Epoch 3/10
1563/1563 [==============================] - 442s 283ms/step - loss: 0.2574 - accura
cy: 0.9087 - val_loss: 1.3103 - val_accuracy: 0.7057
Epoch 4/10
1563/1563 [==============================] - 441s 282ms/step - loss: 0.2317 - accura
cy: 0.9165 - val_loss: 1.3325 - val_accuracy: 0.7088
Epoch 5/10
1563/1563 [==============================] - 441s 282ms/step - loss: 0.2171 - accura
cy: 0.9215 - val_loss: 1.4119 - val_accuracy: 0.7057
Epoch 6/10
1563/1563 [==============================] - 442s 283ms/step - loss: 0.1978 - accura
cy: 0.9294 - val_loss: 1.4937 - val_accuracy: 0.7059
Epoch 7/10
1563/1563 [==============================] - 442s 283ms/step - loss: 0.1828 - accura
cy: 0.9342 - val_loss: 1.5788 - val_accuracy: 0.6950
Epoch 8/10
1563/1563 [==============================] - 441s 282ms/step - loss: 0.1735 - accura
cy: 0.9378 - val_loss: 1.6260 - val_accuracy: 0.7047
Epoch 9/10
1563/1563 [==============================] - 443s 284ms/step - loss: 0.1534 - accura
cy: 0.9441 - val_loss: 1.7186 - val_accuracy: 0.7062
Epoch 10/10
1563/1563 [==============================] - 441s 282ms/step - loss: 0.1539 - accura
cy: 0.9454 - val_loss: 1.8894 - val_accuracy: 0.6984
1:14:20.835258
```

In [ ]:

```
plt.plot(history.history['accuracy'], 'black', linewidth = 3.0)
plt.plot(history.history['val_accuracy'], 'black', ls = '--', linewidth = 3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc = 'center right')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.ylim(ymin=0)
plt.title('Accuracy Curves')
plt.show()
```