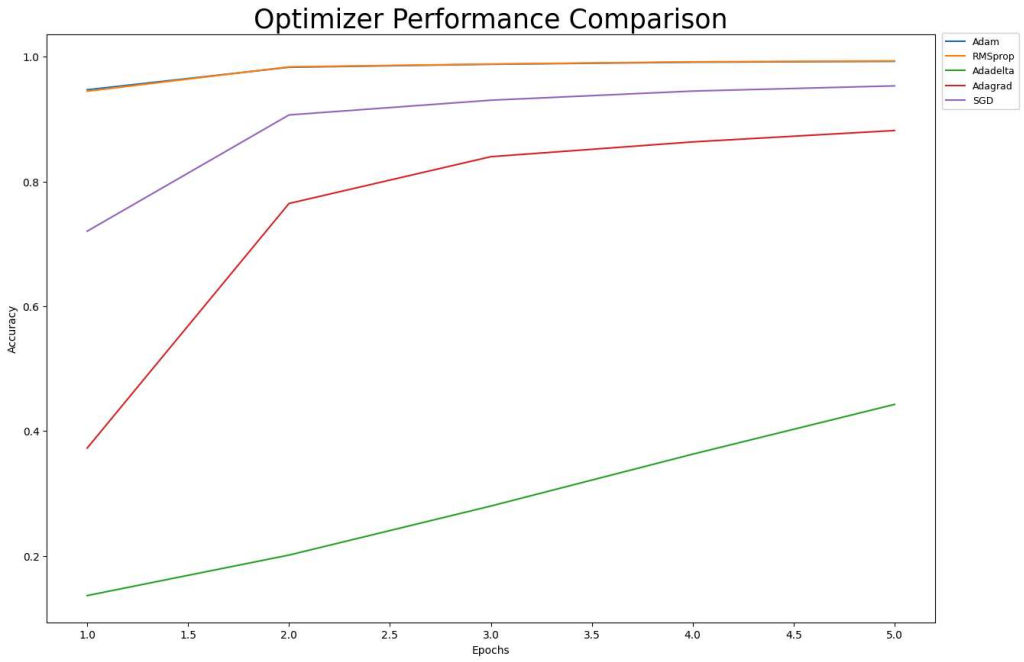
```
 #IMPORTING LIBRARIES
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dense, Dropout,Flatten


# Importing Data
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Data Reshape
x_train=x_train.reshape(x_train.shape[0],28,28,1)
x_test=x_test.reshape(x_test.shape[0],28,28,1)
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')
x_train/=255
x_test/=255
y_train=tf.keras.utils.to_categorical(y_train)
y_test=tf.keras.utils.to_categorical(y_test)


# Build Optimizer Call
def build_optimizer(op):
  model=tf.keras.Sequential()
  model.add(tf.keras.Input(shape=(28,28,1)))
  model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=1,activation='relu'))
  model.add(tf.keras.layers.MaxPool2D())
  model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), strides=1,activation='relu'))
  model.add(tf.keras.layers.Dropout(0.25))
  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(128, activation='relu'))
  model.add(tf.keras.layers.Dense(256, activation='relu'))
  model.add(tf.keras.layers.Dropout(0.5))
  model.add(tf.keras.layers.Dense(10, activation='softmax'))
  model.compile(optimizer=op, loss='categorical_crossentropy', metrics=['accuracy'])
  return model



# Comparing Each Optimizer Accuracy
import os, gc
optimizers=['Adam', 'RMSprop','Adadelta', 'Adagrad', 'SGD']
opt_res=[]
model_res=[]
for i in optimizers:
  model=build_optimizer(i)
  print("Accuracy for: ",i)
  print("\n")
  history=model.fit(x_train,y_train, epochs=5, batch_size=64,verbose=1,validation_data=(x_test, y_test))
  print("\n")
  gc.collect()
  model_res.append(history)
  opt_res.append(history.history['accuracy'])
```

```
Epoch 5/5
938/938 [==============================] - 60s 64ms/step - loss: 0.0224 - accuracy: 0.9935 - val_loss: 0.0285 -


Accuracy for:  Adadelta


Epoch 1/5
938/938 [==============================] - 65s 69ms/step - loss: 2.2880 - accuracy: 0.1363 - val_loss: 2.2661 -
Epoch 2/5
938/938 [==============================] - 63s 67ms/step - loss: 2.2520 - accuracy: 0.2012 - val_loss: 2.2178 -
Epoch 3/5
938/938 [==============================] - 64s 68ms/step - loss: 2.1976 - accuracy: 0.2798 - val_loss: 2.1418 -
Epoch 4/5
938/938 [==============================] - 64s 68ms/step - loss: 2.1145 - accuracy: 0.3630 - val_loss: 2.0273 -
Epoch 5/5
938/938 [==============================] - 63s 68ms/step - loss: 1.9900 - accuracy: 0.4428 - val_loss: 1.8594 -


Accuracy for:  Adagrad


Epoch 1/5
938/938 [==============================] - 60s 64ms/step - loss: 1.9542 - accuracy: 0.3727 - val_loss: 0.8790 -
Epoch 2/5
938/938 [==============================] - 60s 64ms/step - loss: 0.7414 - accuracy: 0.7649 - val_loss: 0.3936 -
Epoch 3/5
938/938 [==============================] - 62s 66ms/step - loss: 0.5143 - accuracy: 0.8399 - val_loss: 0.3176 -
Epoch 4/5
938/938 [==============================] - 58s 62ms/step - loss: 0.4381 - accuracy: 0.8636 - val_loss: 0.2800 -
Epoch 5/5
938/938 [==============================] - 59s 63ms/step - loss: 0.3900 - accuracy: 0.8818 - val_loss: 0.2529 -


Accuracy for:  SGD


Epoch 1/5
938/938 [==============================] - 60s 63ms/step - loss: 0.8777 - accuracy: 0.7204 - val_loss: 0.2642 -
Epoch 2/5
938/938 [==============================] - 56s 60ms/step - loss: 0.3083 - accuracy: 0.9068 - val_loss: 0.1798 -
Epoch 3/5
938/938 [==============================] - 57s 61ms/step - loss: 0.2303 - accuracy: 0.9304 - val_loss: 0.1375 -
Epoch 4/5
938/938 [==============================] - 56s 60ms/step - loss: 0.1846 - accuracy: 0.9451 - val_loss: 0.1152 -
Epoch 5/5
938/938 [==============================] - 57s 60ms/step - loss: 0.1538 - accuracy: 0.9534 - val_loss: 0.1035 -
```

```
# Plotting Optimizer Accuracy
import matplotlib.pyplot as plt
fully_nested = [list(zip(*[(ix+1,y) for ix,y in enumerate(x)])) for x in opt_res]
names = ['sublist%d'%(i+1) for i in range(len(fully_nested))]
fig = plt.figure(figsize=(15,10))
for l in fully_nested:
  plt.plot(*l)
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(optimizers, fontsize=9, loc = 'upper right', bbox_to_anchor=(1.1, 1.01))
plt.title("Optimizer Performance Comparison", fontsize=25)
plt.show()
```




Adam and RMSprop performed the best in terms of accuracy and loss reduction, with Adam being slightly better. Adadelta performed poorly, while Adagrad and SGD showed improvement over epochs but didn't reach the same level of performance as Adam and RMSprop within the given epochs.