Name: Osama Khaled

ID: 20190086

Group: S3

Assignment #3 CNN

Starting with a model consists of 2 convolutional layers with 32 filters and 2x2 kernel sizes with ReLU activation function and Max-Pool of size 2x2 and stride 2x2 and 2 fully connected layers, one hidden layer of 128 neuron and sigmoid activation function and the other is an output layer of 10 neurons as the number of classes in mnist data-set is 10 with soft-max activation function

Model #1:

0-

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=15,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
```

```
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.02% for test , 99.86% for training

98.80% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 42.6 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- with just 15 epochs the result was very high according to other hyper parameters used randomly than the number of epoch.

11- a very good results as a start

Model #2:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)
```

```
c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=17,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.11% for test , 99.93% for training

98.32% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 44.8 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- with just 17 epochs the result was very high according to other hyper parameters used randomly than the number of epoch more than the epoch when it was only 15.

11- the epoch value when it was 17 gave a better result more than it was only 15,but with average time higher than of 15.

Model #3:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.21% for test , 99.99% for training

99.27% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 41 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- with just 20 epochs the result was very high according to other hyper parameters used randomly than the number of epoch more than the epoch when it was only 15 and 17.

11- the epoch value when it was 20 gave a better result more than it was only 15 or 17, it seems that when increases the number of the epochs it gives a better results so, epochs = 20 will continue to the rest.

Model #4:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.1, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 10.28% for test, 10.30% for training

10.41% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 40.75 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.1, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.1 and momentum is = 0.9

9- Dropout not used yet

10- Learning rate was increased up to 0.1 instead of being just 0.01 which made a horrible decrease in the accuracy from 99% to 10%.

11- The learning with value 0.1 overshoots in the loss function gives a very bad accuracy on test set as with also training set which is 10% after it was nearly 100%.

Model #5:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)
```

```
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.05, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.29% for test, 98.99% for training

98.83% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 40.05 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.05, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.05 and momentum is = 0.9

9- Dropout not used yet

10- Learning rate was decreased down to 0.05 instead of being just 0.1 which made a horrible increase in the accuracy from 10% to 98% but not as high as when it was 0.01.

11- it decreases the error to nearly 99% gives a very good accuracy on test set as with also training set which is 98% after it was nearly 10% when it was with value 0.1

Model #6:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.005, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.05% for test, 99.93% for training

98.94% for training in the first 5 epoch

2- number of parameters in the model: 113,386

3- average time to train in each epoch: 45.25 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.005, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.005 and momentum is = 0.9

9- Dropout not used yet

10- Learning rate was decreased down to 0.005 instead of being just 0.05 as it was last model above which made an increase in the accuracy from 98.29% to 99.05% but not as high as when it was 0.01 also which gives almost perfect results on both training and test sets, but not as perfect as when it was 0.01

11- it decreases the error to nearly 99% gives a very good accuracy on test set as with also training set which is 99% after it was nearly 98.9% when it was with value 0.05 so we will continue with learning rate 0.01.

Model #7:

```
#model 1
inputs = KL.Input(shape=(28,28,1))
```

```
c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

#c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
#m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.91% for test, 100% for training

99.14% for training in the first 5 epoch

2- number of parameters in the model: 693,962

3- average time to train in each epoch: 32.4 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 1 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with the same model above except it is with 1 Conv layer instead of two of 32 number of filters and size 3x3 with stride = 1, which decreases the number of layers and increases the number of trainable parameters with a reasonable decrease in the accuracy on test set.

11- it decreases the number of layers but increased the number of trainable parameters to nearly the double which decreases the computation time and training error is decreased to 100% with accuracy on test set decreases from 99% to 98.9%.

Model #8:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

#c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
```

```
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.90% for test, 99.73% for training

98.82% for training in the first 5 epoch

2- number of parameters in the model: 17,578

3- average time to train in each epoch: 41.2 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 1 FC layers which is the output layer all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model with 2 Conv layer of 32 number of filters and size 3x3 with stride = 1, and only 1 FC layer which is the ouput layer of 10 perceptron as the number of classes in the mnist dataset which increases the computational time and decreases not much the accuracy on just the training set.

11- it decreases the number of FC layers and also decreases the number of trainable parameters very much increases the computation time from the last model and decreases the accuracy on training set to 99.73% after it was 100% with accuracy on test set same as last model from 99.9%.

Model #9:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(64,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(c)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.72% for test, 99.83% for training

98.70% for training in the first 5 epoch

2- number of parameters in the model: 37,674

3- average time to train in each epoch: 45 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 3 Conv layers and 2 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model with 3 Conv layers 2 of them of them of 32 number of filters and 1 of 64 number of filters, all of size 3x3 with stride = 1, and 2 FC layer which are one hidden layer and the other is the output layer of 10 perceptron as the number of classes in the mnist dataset which gives accuracy on unseen data reasonably less than the last above model

11- it kind of starting to overfit because it gives accuracy of training set about 99.83% and it gives accuracy on test set 98.72 which is less than the last model above as it was 98.90% and with parameters more than the above model.

Model #10:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(64,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)
```

```
f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.83% for test, 99.79% for training

98.80% for training in the first 5 epoch

2- number of parameters in the model: 71,978

3- average time to train in each epoch: 45 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 3 Conv layers and 3 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model with 3 Conv layers 2 of them of them of 32 number of filters and 1 of 64 number of filters, all of size 3x3 with stride = 1, and 3 FC layer which are two hidden layer and the other is the output layer of 10 perceptron as the number of classes in the mnist dataset, which gives accuracy on unseen data reasonably less than the last above model

11- it gives accuracy of training set about 99.79% and it gives accuracy on test set 98.83% which is less than the last model above as it was 98.90% and with parameters more than the above model.

Model #11:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

#c = KL.Conv2D(64,(3,3),padding='valid',activation=tf.nn.relu)(m)
#m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.19% for test, 99.98% for training

99.26% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 45 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model with 2 Conv layers 2 both of 32 filters, all of size 3x3 with stride = 1, and 3 FC layer which are two hidden layer and the other is the output layer of 10 perceptron as the number of classes in the mnist dataset better accuracy than before

11- it gives accuracy of training set about 99.998% and it gives accuracy on test set 99.19% which is greater than the last model above as it was 98.83% and with parameters more than the above model.

This tells us that the last architecture is the best, as it has less parameters and very high accuracy approximately 100% correct on both training and test sets.

Model #12:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=64,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.08% for test, 99.992% for training

99.08% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 38 sec

4- average test time in each epoch: 5 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model of batch size greater than the last implemented model which is 64 instead of 32 but gives a less good results compares when it was 32

11- it gives accuracy of training set about 99.992% and it gives accuracy on test set 99.08% which is less than the last model above with batch size 32 as it was 99.19% and with parameters same as the above model .

Model #13:

```
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)
```

```
model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=16,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.19% for test, 99.98% for training

99.24% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 49.35 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are ReLU activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with a model of batch size less than the last implemented model which is 16 instead of 32 or 64 gives results better than it was 64 but less when it was 32

11- it gives accuracy of training set about 99.81% and it gives accuracy on test set 99.13% which is greater than when batch size was 64 as it was 99.08%but less than the model with batch size 32 as it was 99.19% and with parameters same as the above model.

This tells us that this model with batch size = 32 is the best among equal to 64 or 16.

Model #14:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.selu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.selu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.selu)(f)

f = KL.Dense(256,activation=tf.nn.selu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.21% for test, 100% for training

99.30% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 47.3 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activation Selu activation function with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working with selu activation function at all layers which gives a very high accuracy on both training and test data sets.

11- it gives accuracy on the training set greater than that of the last tested model as it gives accuracy 100% on the training set and gives on the training set of the last model 99.81%, and it gives also higher accuracy on unseen data test set than that of the last model that used ReLU activation function at all layers as it gives accuracy on test set of that model 99.21% but gives on test set of the last tested model 99.13%.

Model #15:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.sigmoid)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.sigmoid)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)
```

```
f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.sigmoid)(f)

f = KL.Dense(256,activation=tf.nn.sigmoid)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 98.269% for test, 99.29% for training

10.85% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 41.4 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are sigmoid activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working sigmoid activation function at layers of both conv layers and FC layers but sigmoid has sort of saturation which make it less likely to be used, so it generates results less accurately than ReLU activation function.

11- it gives accuracy on the training set less than that of the last tested model as it gives accuracy 98.29% on the training set and gives on the training set of the last model 99.96%, and it gives less accuracy on unseen data test set than that of the last model that used ReLU activation function at all layers as it gives accuracy on test set of that model 98.269% but gives on test set of the last tested model 99.08%.

Model #16:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.tanh)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.tanh)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.tanh)(f)

f = KL.Dense(256,activation=tf.nn.tanh)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
```

```
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.05% for test, 100% for training

99.51% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 43.2 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are tanh activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working on tanh activation function at layers of both conv layers and FC layers which not very good compares to ReLU

11- it gives accuracy on the training set much greater than that of the last tested model as it gives accuracy 100% on the training set and gives on the training set of the last model 99.29%, and it gives less accuracy on unseen data test set than that of the last model that used ReLU activation function at all layers as it gives accuracy on test set of that model 98.05% but gives on test set of the last tested model 99.269%.

This tells us that the Selu activation function gives us better results on both training and unseen data even any other activation function gives more better results on training data even it gives accuracy on the training dataset 100%, but we are going to continue with ReLU activation function as some popular and top optimizers such as Adam or Adagrade will generate a very very bad results with Selu activation function and generate a very very good results using ReLU.

Model #17:

```
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.RMSprop(
    learning_rate=0.01, momentum=0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 96.35% for test, 97.90% for training

96.96% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 46.5 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- RMS optimizer with learning rate = 0.01 and momentum is = 0.9

9- Dropout not used yet

10- working on relu activation function at layers of both conv layers and FC layers using RMSprop optimizer.

11- A good accuracy on both training and test datasets and take more time but with less accuracy when using SGD as an optimizer.

Model #18:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)
```

```
model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.Adam(
    learning_rate=0.01
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 96.35% for test, 97.70% for training

96.96% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 45.4 sec

4- average test time in each epoch: 3 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- Adam optimizer with learning rate = 0.01 and default values of beta_1 is = 0.9, beta_2 is = 0.999 and epsilon is = 1e-07

9- Dropout not used yet

10- working on relu activation function at layers of both conv layers and FC layers using a very popular optimizer Adam.

11- a good accuracy on both training and test datasets and take more time but with less good results compares to SGD

So SGD optimizer will continue to the rest of the assignment.

Model #19:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

d = KL.Dropout(0.5)(m)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(d)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

d = KL.Dropout(0.5)(c)

f = KL.Dense(256,activation=tf.nn.relu)(d)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(f)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum = 0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.08% for test, 98.77% for training

97.56% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 44.1 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- using a dropout layer of rate 50% after the first conv layer, and another one of the same rate after the first hidden FC layer because I want to test what the result would be after using a big rate on both locations together

10- using dropout layer of rate 50% after the first conv layer, and another one of the same rate after the first hidden FC layer, gives an accuracy on test set higher than the accuracy on training set by about 0.23%, which less than both when the model has not a dropout layer at all.

11- A less good accuracy on both training and test datasets and take more time compares to when no dropout layer exist.

Model #20:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

d = KL.Dropout(0.3)(m)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(d)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)
```

```
f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

d = KL.Dropout(0.6)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(d)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum = 0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.26% for test, 99.56% for training

98.63% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 46.7 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- using a dropout layer of rate of less rate after the first conv layer of rate 30% compares to the corresponding location at the last model tested of rate 50%0, and another dropout layer after the second hidden FC layer of greater rate of 60%

10- using 2 dropout layers at different locations with different rates than the last tested model with greater rate after the second hidden FC layer and smaller after the first Conv layer.

11- A greater accuracy on both training and test datasets and take a reasonably more time compares to when no dropout layer exist and compares to the last tested model before it.

Model #21:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)


c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(m)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

d = KL.Dropout(0.3)(m)


f = KL.Flatten()(f)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

d = KL.Dropout(0.65)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(d)

model = KM.Model(inputs,ouputs)
```

```
model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum = 0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.30% for test, 99.31% for training

98.33% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 45.2 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- using a dropout layer of rate of less rate after the second conv layer of rate 30% compares to the corresponding location at the last model tested of rate 30% after the first conv layer in the model, and another dropout layer after the second hidden FC layer of greater rate of 65%

10- using 2 dropout layers at different locations with different rates than the last tested model with greater rate after the second hidden FC layer and same as the last tested model after the second Conv layer.

11- A greater accuracy on test set but less at training dataset and take a reasonably less time compares to the last tested model before it.

Model #22:

```
#model 1
inputs = KL.Input(shape=(28,28,1))

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(inputs)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

d = KL.Dropout(0.2)(m)

c = KL.Conv2D(32,(3,3),padding='valid',activation=tf.nn.relu)(d)
m = KL.MaxPool2D((2,2),strides=(2,2))(c)

f = KL.Flatten()(m)

c = KL.Dense(128,activation=tf.nn.relu)(f)

f = KL.Dense(256,activation=tf.nn.relu)(c)

d = KL.Dropout(0.75)(f)

ouputs = KL.Dense(10,activation=tf.nn.softmax)(d)

model = KM.Model(inputs,ouputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum = 0.9
),loss = "sparse_categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=20,batch_size=32,verbose = 2)
print("\nabove training below testing\n")
test_loss, test_acc = model.evaluate(x_test,y_test,verbose = 2)
print("Test loss: {0} - Test accuracry: {1}".format(test_loss,test_acc))
```

1- 99.28% for test, 99.50% for training

98.68% for training in the first 5 epoch

2- number of parameters in the model: 147,690

3- average time to train in each epoch: 49.25 sec

4- average test time in each epoch: 2 sec

5- The layers of each model (including activations): 2 Conv layers and 3 FC layers all activations are relu activation functions with the softmax activation function at the output layer ofcourse

6- The learning rate used: 0.01, momentum is: 0.9

7- not used

8- SGD optimizer with learning rate = 0.01 and momentum is = 0.9

9- using a dropout layer of rate of less rate after the first conv layer of rate 20% compares to the corresponding location at the last model tested of rate 30% after the first conv layer in the model, and another dropout layer after the second hidden FC layer of greater rate than the corresponding dropout layer rate in the last tested model to be 75% instead of 65%

10- using 2 dropout layers at different locations with different rates than the last tested model with greater rate after the second hidden FC layer and less rate of the last tested model after the first Conv layer.

11- A greater accuracy on training set but less at test dataset and take a great more time compared to the last tested model before it.

So with Dropout layers gives a reasonably less accuracy than when no dropout layer used. So, with using dropout 30% after the 2nd conv layer, and 65% rate after the 2nd hidden FC layer is the best with regarding to accuracy on both test and training sets with also less computation time.