Convolutional Neural Network on mnist dataset

Supervised Learning (Assignment 3):

Name: Fady Essam Fathy **ID:** 20190370 **Group:** s4

Case 1:

For this case we test for the best number of epochs with batch size=32

Model 1: we use 10 epochs

Final accuracy: **98.47%**

Number of parameters: **121.930**

Average time: **in the figure**

Layers and activations: **2 conv layers with relu activation function and softmax for output layer**

Learning rate: **0.01**

Optimizers: **SGD with 0.01 momentum**

```
# Model 1

model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=10, batch_size = 32)
```

```
Epoch 1/10
1875/1875 [==============================] - 20s 10ms/step - loss: 0.5035 - accuracy: 0.8618
Epoch 2/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.1368 - accuracy: 0.9591
Epoch 3/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0979 - accuracy: 0.9704
Epoch 4/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0784 - accuracy: 0.9765
Epoch 5/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0670 - accuracy: 0.9799
```

Convolutional Neural Network on mnist dataset

Model 2: we use 12 epochs

Final accuracy: **98.90%**

Number of parameters: **121.930**

Average time: **in the figure**

Layers and activations:

2 conv layers with relu activation function and softmax for output layer

Learning rate: **0.01**

Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=12, batch_size = 32)
```

```
Epoch 1/12
1875/1875 [==============================] - 28s 15ms/step - loss: 0.5318 - accuracy: 0.8415
Epoch 2/12
1875/1875 [==============================] - 28s 15ms/step - loss: 0.1479 - accuracy: 0.9554
Epoch 3/12
1875/1875 [==============================] - 29s 15ms/step - loss: 0.1014 - accuracy: 0.9686
Epoch 4/12
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0813 - accuracy: 0.9755
Epoch 5/12
1875/1875 [==============================] - 29s 16ms/step - loss: 0.0699 - accuracy: 0.9786
```

Model 3: we use 15 epochs

Final accuracy: **99.25%**

Number of parameters: **121.930**

Average time: **in the figure**

Layers and activations: **2 conv layers with relu activation function and softmax for output layer**

Learning rate: **0.01**

Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0342 - accuracy: 0.9894
Epoch 2/15
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0317 - accuracy: 0.9899
Epoch 3/15
1875/1875 [==============================] - 29s 16ms/step - loss: 0.0296 - accuracy: 0.9911
Epoch 4/15
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0280 - accuracy: 0.9912
Epoch 5/15
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0264 - accuracy: 0.9919
```

So, as we see the best number of epochs is 15 with 121.930 parameters because it has the highest accuracy (99.25) and worst accuracy came with 10 epochs.

Convolutional Neural Network on mnist dataset

Case 2:

For this case we test for the best learning rate with batch size=32

Model 4: we use 0.001 learning rate

Final accuracy: **96.53%**
Number of parameters: **121.930**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.001**
Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.001, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 18s 9ms/step - loss: 2.1727 - accuracy: 0.3708
Epoch 2/15
1875/1875 [==============================] - 20s 11ms/step - loss: 1.0226 - accuracy: 0.7666
Epoch 3/15
1875/1875 [==============================] - 21s 11ms/step - loss: 0.4476 - accuracy: 0.8743
Epoch 4/15
1875/1875 [==============================] - 22s 12ms/step - loss: 0.3481 - accuracy: 0.8997
Epoch 5/15
1875/1875 [==============================] - 24s 13ms/step - loss: 0.3000 - accuracy: 0.9131
```

Model 5: we use 0.1 learning rate

Final accuracy: **99.15%**
Number of parameters: **121.930**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.1**
Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.1, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 20s 11ms/step - loss: 0.2081 - accuracy: 0.9396
Epoch 2/15
1875/1875 [==============================] - 24s 13ms/step - loss: 0.0560 - accuracy: 0.9827
Epoch 3/15
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0409 - accuracy: 0.9876
Epoch 4/15
1875/1875 [==============================] - 25s 14ms/step - loss: 0.0307 - accuracy: 0.9905
Epoch 5/15
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0244 - accuracy: 0.9922
```

Convolutional Neural Network on mnist dataset

Model 6: we use 0.5 learning rate
Final accuracy: **11.35%**
Number of parameters: **121.930**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.5, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 20s 11ms/step - loss: 2.3254 - accuracy: 0.1121
Epoch 2/15
1875/1875 [==============================] - 25s 13ms/step - loss: 2.3050 - accuracy: 0.1055
Epoch 3/15
1875/1875 [==============================] - 30s 16ms/step - loss: 2.3050 - accuracy: 0.1049
Epoch 4/15
1875/1875 [==============================] - 26s 14ms/step - loss: 2.3047 - accuracy: 0.1072
Epoch 5/15
```

At case 1 when we use 0.01 learning rate the accuracy was 99.25% so it is the best learning rate because the max accuracy in case 2 is 99.15%

Convolutional Neural Network on mnist dataset

Case 3:

For this case we change number of cnn and parameters with batch size=32, epochs=15 and learning rate= 0.01

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(64,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

Model 7: we add another cnn layer with same size

Final accuracy: **98.22%**
Number of parameters: **60.554**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 17s 9ms/step - loss: 0.8240 - accuracy: 0.7440
Epoch 2/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1904 - accuracy: 0.9414
Epoch 3/15
1875/1875 [==============================] - 19s 10ms/step - loss: 0.1380 - accuracy: 0.9569
Epoch 4/15
1875/1875 [==============================] - 22s 12ms/step - loss: 0.1118 - accuracy: 0.9657
Epoch 5/15
1875/1875 [==============================] - 24s 13ms/step - loss: 0.0965 - accuracy: 0.9707
```

Model 8: we remove a cnn layer

Final accuracy: **98.12%**
Number of parameters: **347.146**
Average time: **in the figure**
Layers and activations: **1 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4793 - accuracy: 0.8689
Epoch 2/15
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2256 - accuracy: 0.9340
Epoch 3/15
1875/1875 [==============================] - 16s 8ms/step - loss: 0.1767 - accuracy: 0.9481
Epoch 4/15
1875/1875 [==============================] - 15s 8ms/step - loss: 0.1453 - accuracy: 0.9571
Epoch 5/15
1875/1875 [==============================] - 16s 8ms/step - loss: 0.1242 - accuracy: 0.9634
```

Convolutional Neural Network on mnist dataset

Model 9: we change size of cnn layers to (2,2)

Final accuracy: **98.21%**
Number of parameters: **156.586**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(32,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(64,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 15s 8ms/step - loss: 0.6064 - accuracy: 0.8148
Epoch 2/15
1875/1875 [==============================] - 15s 8ms/step - loss: 0.2064 - accuracy: 0.9390
Epoch 3/15
1875/1875 [==============================] - 16s 9ms/step - loss: 0.1350 - accuracy: 0.9602
Epoch 4/15
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1042 - accuracy: 0.9686
Epoch 5/15
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0876 - accuracy: 0.9733
```

Model 10: we increase number of filters

Final accuracy: **98.47%**
Number of parameters: **328.842**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 38s 20ms/step - loss: 0.6432 - accuracy: 0.8183
Epoch 2/15
1875/1875 [==============================] - 41s 22ms/step - loss: 0.2007 - accuracy: 0.9399
Epoch 3/15
1875/1875 [==============================] - 42s 22ms/step - loss: 0.1311 - accuracy: 0.9603
Epoch 4/15
1875/1875 [==============================] - 41s 22ms/step - loss: 0.1024 - accuracy: 0.9690
Epoch 5/15
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0857 - accuracy: 0.9741
```

Convolutional Neural Network on mnist dataset

Model 11: we add another FC layer

Final accuracy: **98.62%**
Number of parameters: **330.602**
Average time: **in the figure**
Layers and activations: **2 conv layers with relu activation function and softmax for output layer with another hidden layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(32,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 32)
```

```
Epoch 1/15
1875/1875 [==============================] - 41s 22ms/step - loss: 0.7145 - accuracy: 0.7766
Epoch 2/15
1875/1875 [==============================] - 46s 24ms/step - loss: 0.1962 - accuracy: 0.9393
Epoch 3/15
1875/1875 [==============================] - 43s 23ms/step - loss: 0.1280 - accuracy: 0.9601
Epoch 4/15
1875/1875 [==============================] - 43s 23ms/step - loss: 0.0970 - accuracy: 0.9704
Epoch 5/15
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0817 - accuracy: 0.9747
```

From case 3 we see that when we add another hidden layer the accuracy increased to 98.62 which is better than the other models

But number of parameters became worst, until we add another cnn layer

So most suitable for accuracy and parameters that we add another cnn layers (98.22)

Convolutional Neural Network on mnist dataset

Case 4:

For this case we test different batch sizes with epochs=15 and learning rate= 0.01

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 64)
```

```
Epoch 1/15
938/938 [==============================] - 39s 41ms/step - loss: 0.9038 - accuracy: 0.7345
Epoch 2/15
938/938 [==============================] - 42s 45ms/step - loss: 0.2401 - accuracy: 0.9274
Epoch 3/15
938/938 [==============================] - 54s 58ms/step - loss: 0.1580 - accuracy: 0.9522
Epoch 4/15
938/938 [==============================] - 53s 56ms/step - loss: 0.1194 - accuracy: 0.9631
Epoch 5/15
938/938 [==============================] - 54s 57ms/step - loss: 0.0992 - accuracy: 0.9686
```

Model 12: we try batch with size**= 64**

Final accuracy: **98.20%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

Model 13: we try batch with size**= 128**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 128)
```

```
Epoch 1/15
469/469 [==============================] - 35s 75ms/step - loss: 1.6341 - accuracy: 0.5698
Epoch 2/15
469/469 [==============================] - 40s 85ms/step - loss: 0.4241 - accuracy: 0.8693
Epoch 3/15
469/469 [==============================] - 47s 100ms/step - loss: 0.2966 - accuracy: 0.9100
Epoch 4/15
469/469 [==============================] - 52s 112ms/step - loss: 0.2260 - accuracy: 0.9316
Epoch 5/15
469/469 [==============================] - 48s 101ms/step - loss: 0.1799 - accuracy: 0.9458
```

Final accuracy: **97.87%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

Convolutional Neural Network on mnist dataset

Model 14: we try batch with size**= 20**

Final accuracy: **98.77%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 60s 20ms/step - loss: 0.4712 - accuracy: 0.8585
Epoch 2/15
3000/3000 [==============================] - 76s 25ms/step - loss: 0.1186 - accuracy: 0.9633
Epoch 3/15
3000/3000 [==============================] - 71s 24ms/step - loss: 0.0835 - accuracy: 0.9740
Epoch 4/15
3000/3000 [==============================] - 77s 26ms/step - loss: 0.0678 - accuracy: 0.9783
Epoch 5/15
3000/3000 [==============================] - 76s 25ms/step - loss: 0.0566 - accuracy: 0.9818
```

As we see the best accuracy came with the batch size=20 but a lot of time

while the worst accuracy with size= 128 but it was the fastest

Convolutional Neural Network on mnist dataset

Case 5:

For this case we test different activation functions (with softmax in output layer) with epochs=15, learning rate= 0.01 and batch size= 20

Model 15: we try sigmoid activation function

Final accuracy: **36.27%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with sigmoid activation function and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='sigmoid', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='sigmoid'))
model.add(layers.Conv2D(64,(2,2), activation='sigmoid'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'sigmoid'))
model.add(layers.Dense(10, activation= 'softmax'))


model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 53s 18ms/step - loss: 2.3075 - accuracy: 0.1034
Epoch 2/15
3000/3000 [==============================] - 64s 21ms/step - loss: 2.3053 - accuracy: 0.1052
Epoch 3/15
3000/3000 [==============================] - 65s 22ms/step - loss: 2.3043 - accuracy: 0.1061
Epoch 4/15
3000/3000 [==============================] - 66s 22ms/step - loss: 2.3035 - accuracy: 0.1073
Epoch 5/15
3000/3000 [==============================] - 72s 24ms/step - loss: 2.3030 - accuracy: 0.1085
```

Model 16: we try tanh activation function

Final accuracy: **98.77%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with tanh activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='tanh', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='tanh'))
model.add(layers.Conv2D(64,(2,2), activation='tanh'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'tanh'))
model.add(layers.Dense(10, activation= 'softmax'))


model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 58s 19ms/step - loss: 0.4306 - accuracy: 0.8823
Epoch 2/15
3000/3000 [==============================] - 65s 22ms/step - loss: 0.1373 - accuracy: 0.9618
Epoch 3/15
3000/3000 [==============================] - 65s 22ms/step - loss: 0.0943 - accuracy: 0.9735
Epoch 4/15
3000/3000 [==============================] - 65s 22ms/step - loss: 0.0743 - accuracy: 0.9790
Epoch 5/15
3000/3000 [==============================] - 66s 22ms/step - loss: 0.0624 - accuracy: 0.9822
```

Convolutional Neural Network on mnist dataset

Model 17: we try leaky_relu activation function

Final accuracy: **98.76%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with leacky_relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 momentum**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='LeakyReLU', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='LeakyReLU'))
model.add(layers.Conv2D(64,(2,2), activation='LeakyReLU'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'LeakyReLU'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01, momentum=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 65s 22ms/step - loss: 0.4841 - accuracy: 0.8496
Epoch 2/15
3000/3000 [==============================] - 69s 23ms/step - loss: 0.1137 - accuracy: 0.9647
Epoch 3/15
3000/3000 [==============================] - 70s 23ms/step - loss: 0.0818 - accuracy: 0.9745
Epoch 4/15
3000/3000 [==============================] - 68s 23ms/step - loss: 0.0676 - accuracy: 0.9788
Epoch 5/15
3000/3000 [==============================] - 68s 23ms/step - loss: 0.0571 - accuracy: 0.9819
```

So, we see that sigmoid activation function get the worst accuracy and it is not useful in the models, while the Relu activation function get the best accuracy (98.77%) but it take more time.

Convolutional Neural Network on mnist dataset

Case 6:

For this case we test different optimizers with epochs=15, learning rate= 0.01 and batch size= 20

Model 18: we use Adam optimizer

Final accuracy: **96.22%**
Number of parameters: **169.162**
Average time: **in the figure**

```
model.compile(optimizer = tf.optimizers.Adam(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.1853 - accuracy: 0.9433
Epoch 2/15
3000/3000 [==============================] - 61s 20ms/step - loss: 0.1154 - accuracy: 0.9661
Epoch 3/15
3000/3000 [==============================] - 62s 21ms/step - loss: 0.1056 - accuracy: 0.9690
Epoch 4/15
3000/3000 [==============================] - 60s 20ms/step - loss: 0.1016 - accuracy: 0.9702
Epoch 5/15
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0987 - accuracy: 0.9715
```

Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **Adam with 0.01 learning rate**

Model 19: we use Adadelta optimizer

Final accuracy: **97.65%**
Number of parameters: **169.162**
Average time: **in the figure**

```
model.compile(optimizer = tf.optimizers.Adadelta(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 49s 16ms/step - loss: 0.1653 - accuracy: 0.9650
Epoch 2/15
3000/3000 [==============================] - 62s 21ms/step - loss: 0.1504 - accuracy: 0.9676
Epoch 3/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.1387 - accuracy: 0.9700
Epoch 4/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.1294 - accuracy: 0.9717
Epoch 5/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.1222 - accuracy: 0.9731
```

Layers and activations:
3 conv layers with relu activation functions and softmax for output layer
Learning rate**: 0.01**
Optimizers: **Adadelta with 0.01 learning rate**

So, as we see the best optimizer is SGD because Adam optimizer get worst accuracy and take a lot of time and Adadleta optimizer get medium accuracy and medium time.

Case 7:

For this case we put a dropout layer with different rates and in different places, with epochs=15, learning rate= 0.01 and batch size= 20

(dropout layer helps prevent overfitting by sets inputs unit to 0 with a frequency of rate)

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.1))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.Conv2D(64,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

Model 20: we use dropout layer with 0.1 rate

Final accuracy: **98.74%**

Number of parameters: **169.162**

Average time: **in the figure**

Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**

Learning rate**: 0.01**

Optimizers: **SGD with 0.01 learning rate**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 67s 22ms/step - loss: 0.4955 - accuracy: 0.8438
Epoch 2/15
3000/3000 [==============================] - 74s 25ms/step - loss: 0.1270 - accuracy: 0.9607
Epoch 3/15
3000/3000 [==============================] - 71s 24ms/step - loss: 0.0893 - accuracy: 0.9716
Epoch 4/15
3000/3000 [==============================] - 70s 23ms/step - loss: 0.0736 - accuracy: 0.9776
Epoch 5/15
3000/3000 [==============================] - 73s 24ms/step - loss: 0.0638 - accuracy: 0.9802
```

Convolutional Neural Network on mnist dataset

Model 21: we use dropout layer with 0.5 rate

Final accuracy: **98.80%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 learning rate**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 60s 20ms/step - loss: 0.5690 - accuracy: 0.8178
Epoch 2/15
3000/3000 [==============================] - 65s 22ms/step - loss: 0.1589 - accuracy: 0.9500
Epoch 3/15
3000/3000 [==============================] - 74s 25ms/step - loss: 0.1118 - accuracy: 0.9650
Epoch 4/15
3000/3000 [==============================] - 74s 25ms/step - loss: 0.0893 - accuracy: 0.9716
Epoch 5/15
3000/3000 [==============================] - 70s 23ms/step - loss: 0.0770 - accuracy: 0.9764
```

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.9))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.Conv2D(64,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

Model 22: we use dropout layer with 0.9 rate

Final accuracy: **97.67%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 learning rate**

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 62s 21ms/step - loss: 0.6454 - accuracy: 0.7902
Epoch 2/15
3000/3000 [==============================] - 66s 22ms/step - loss: 0.2746 - accuracy: 0.9141
Epoch 3/15
3000/3000 [==============================] - 67s 22ms/step - loss: 0.1866 - accuracy: 0.9411
Epoch 4/15
3000/3000 [==============================] - 68s 23ms/step - loss: 0.1554 - accuracy: 0.9514
Epoch 5/15
3000/3000 [==============================] - 68s 23ms/step - loss: 0.1365 - accuracy: 0.9562
```

Convolutional Neural Network on mnist dataset

So, the best dropout rate is 0.5 as it get accuracy 98.80% and worst accuracy with rate 0.9

Model 23: we use dropout layer in different places, **first place**: after the first cnn layer

Final accuracy: **98.66%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01 learning rate**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.5))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.Conv2D(64,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 58s 19ms/step - loss: 0.5248 - accuracy: 0.8350
Epoch 2/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.1453 - accuracy: 0.9541
Epoch 3/15
3000/3000 [==============================] - 62s 21ms/step - loss: 0.1063 - accuracy: 0.9660
Epoch 4/15
3000/3000 [==============================] - 62s 21ms/step - loss: 0.0870 - accuracy: 0.9727
Epoch 5/15
3000/3000 [==============================] - 64s 21ms/step - loss: 0.0761 - accuracy: 0.9760
```

Model 24: we use dropout layer in different places, **second place**: after the fully conected layer
Final accuracy: **98.80%**
Number of parameters: **169.162**
Average time: **in the figure**
Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**
Learning rate**: 0.01**
Optimizers: **SGD with 0.01**

```
model = models.Sequential()
model.add(layers.Conv2D(64,(2,2), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(128,(2,2), activation='relu'))
model.add(layers.Conv2D(64,(2,2), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation = 'relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 54s 18ms/step - loss: 0.7046 - accuracy: 0.7731
Epoch 2/15
3000/3000 [==============================] - 57s 19ms/step - loss: 0.2423 - accuracy: 0.9283
Epoch 3/15
3000/3000 [==============================] - 58s 19ms/step - loss: 0.1835 - accuracy: 0.9452
Epoch 4/15
3000/3000 [==============================] - 61s 20ms/step - loss: 0.1564 - accuracy: 0.9539
Epoch 5/15
3000/3000 [==============================] - 58s 19ms/step - loss: 0.1359 - accuracy: 0.9587
```

Convolutional Neural Network on mnist dataset

So as we see, when we put dropout layer after the hidden layer (FC layer) the accuracy increased and became (98.80%)

Till now best accuracy is 99.13 % but with 121.994 parameters which is quite large and it can be reduced, so the last way is to reduce number of neurons (filters) of the layers and see what will happen..

Model 25:

we reduce numbers of filters to 16 with 30 neurons and size 3*3

Final accuracy: **98.46%**

Number of parameters: **12.820**

Average time: **in the figure**

Layers and activations: **3 conv layers with relu activation functions and softmax for output layer**

Learning rate**: 0.01**

Optimizers: **SGD with 0.01**

```
In [12]: model = models.Sequential()
         model.add(layers.Conv2D(16,(3,3), activation='relu', input_shape = (28,28,1)))
         model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
         model.add(layers.Conv2D(16,(3,3), activation='relu'))
         model.add(layers.Conv2D(16,(3,3), activation='relu'))
         model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
         model.add(layers.Flatten())
         model.add(layers.Dense(30,activation = 'relu'))
         model.add(Dropout(0.5))
         model.add(layers.Dense(10, activation= 'softmax'))

         model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 26, 26, 16)        160

 max_pooling2d_18 (MaxPoolin  (None, 13, 13, 16)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 11, 11, 16)        2320

 conv2d_29 (Conv2D)          (None, 9, 9, 16)          2320

 max_pooling2d_19 (MaxPoolin  (None, 4, 4, 16)         0
 g2D)

 flatten_9 (Flatten)         (None, 256)               0

 dense_18 (Dense)            (None, 30)                7710

 dropout_9 (Dropout)         (None, 30)                0

 dense_19 (Dense)            (None, 10)                310

=================================================================
Total params: 12,820
Trainable params: 12,820
Non-trainable params: 0
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print('loss=', score[0])
print('accuracy=', score[1] * 100, '%')
```

```
loss= 0.04785200580954552
accuracy= 98.46000075340271 %
```

Conclusion:

At the end after we test every parameter in the models we found that best accuracy with most suitable number of parameters and suitable time is **99.13%** and **121.994** parameters which is quite big, but when we reduce the number of filters (neurons), the number is greatly reduced and became **12.820** with accuracy= **98.65%** which is still suitable.

So the best final model came with:

Epochs= **15**

Learning rate= **0.01**

3 CNN layers with **1** FC Layer and output with number of filters= **16** and **30** for neurons in hidden layer

Batch size= **20**

Relu activation function

SGD optimizer

Dropout layer with rate= **0.5** and put it **after the FC layer**

<u>As the final (best) accuracy= 98.65%</u>

<u>With number of parameters= 12.820</u>

And this full Code for best model:

Convolutional Neural Network on mnist dataset

In [1]:
```
import numpy as np
import keras
import tensorflow as tf
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Input
from tensorflow.keras.utils import to_categorical
from keras.layers import Dropout
from keras import backend as k
from keras import models
from keras import layers
```

In [2]:
```
#Loading and Processing the Data

(x_train, y_train), (x_test, y_test) = mnist.load_data()

#Reshape Dataset to have a Single Channel
x_train = x_train.reshape((60000,28,28,1))
x_test = x_test.reshape((10000,28,28,1))

#Convert from Integers to Floats
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255

# Convert Class Vectors to Binary Class Matrices  OR one Hot Encode target values
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
y_train shape: (60000, 10)
60000 train samples
10000 test samples
```

Convolutional Neural Network on mnist dataset

In [3]:
```
model = models.Sequential()
model.add(layers.Conv2D(16,(3,3), activation='relu', input_shape = (28,28,1)))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Conv2D(16,(3,3), activation='relu'))
model.add(layers.Conv2D(16,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(30,activation = 'relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(10, activation= 'softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 16)        160

 max_pooling2d (MaxPooling2D  (None, 13, 13, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 16)        2320

 conv2d_2 (Conv2D)           (None, 9, 9, 16)          2320

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 30)                7710

 dropout (Dropout)           (None, 30)                0

 dense_1 (Dense)             (None, 10)                310

=================================================================
Total params: 12,820
Trainable params: 12,820
Non-trainable params: 0
_________________________________________________________________
```

Convolutional Neural Network on mnist dataset

In [4]:
```
model.compile(optimizer = tf.optimizers.SGD(learning_rate=0.01),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'] )

model.fit(x_train, y_train, epochs=15, batch_size = 20)
```

```
Epoch 1/15
3000/3000 [==============================] - 14s 5ms/step - loss: 1.0731 - accuracy: 0.6231
Epoch 2/15
3000/3000 [==============================] - 14s 5ms/step - loss: 0.4480 - accuracy: 0.8485
Epoch 3/15
3000/3000 [==============================] - 17s 6ms/step - loss: 0.3602 - accuracy: 0.8787
Epoch 4/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.3085 - accuracy: 0.8969
Epoch 5/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.2788 - accuracy: 0.9062
Epoch 6/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.2548 - accuracy: 0.9147
Epoch 7/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.2345 - accuracy: 0.9220
Epoch 8/15
3000/3000 [==============================] - 19s 6ms/step - loss: 0.2162 - accuracy: 0.9280
Epoch 9/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.2092 - accuracy: 0.9301
Epoch 10/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.1993 - accuracy: 0.9337
Epoch 11/15
3000/3000 [==============================] - 19s 6ms/step - loss: 0.1903 - accuracy: 0.9363
Epoch 12/15
3000/3000 [==============================] - 19s 6ms/step - loss: 0.1809 - accuracy: 0.9420
Epoch 13/15
3000/3000 [==============================] - 18s 6ms/step - loss: 0.1721 - accuracy: 0.9442
Epoch 14/15
3000/3000 [==============================] - 19s 6ms/step - loss: 0.1646 - accuracy: 0.9470
Epoch 15/15
3000/3000 [==============================] - 19s 6ms/step - loss: 0.1597 - accuracy: 0.9488
```

Out[4]: <keras.callbacks.History at 0x1825747a610>

In [5]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('loss=', score[0])
print('accuracy=', score[1] * 100, '%')
```

```
loss= 0.0426531545817852
accuracy= 98.65999817848206 %
```