Supervised learning

Assignment 3

Fatima Salih Ibrahim (20190744)

Mohamed Faysal (20190711)

The code link on Google colab :

Assignment3.ipynb - Colaboratory (google.com)

#Load the libraries

```
import numpy as np
from sklearn.model_selection import KFold
from keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D , AveragePooling2D
from keras.layers import Dense , Dropout
from keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam ,RMSprop
```

#Building models with different optimizers

```
def Build_model_1(model , train_norm , trainY):
  scores , losses = list() , list()
  kfold = KFold(5, shuffle=True, random_state=1)

  for train_ix, test_ix in kfold.split(train_norm):
    opt =SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accuracy'])

    trainXTemp, trainYTemp, testXTemp, testYTemp = train_norm[train_ix], trainY[train_ix], train_norm
[test_ix], trainY[test_ix]

    model.fit(trainXTemp, trainYTemp, epochs=10, batch_size=128, validation_data=(testXTemp, testYTe
mp), verbose=0)

    loss, acc = model.evaluate(testXTemp, testYTemp, verbose=0)
    print('accuracy: %.3f' % (acc * 100.0))
    print('loss: %.7f' % (loss))

    scores.append(acc)
    losses.append(loss)
  return scores , losses
```

```
def Build_Model_2(model , train_norm , trainY):
  scores , losses = list() , list()
  kfold = KFold(5, shuffle=True, random_state=1)
  for train_ix, test_ix in kfold.split(train_norm):
    opt=Adam(lr=0.01, decay=0.00001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    trainXTemp, trainYTemp, testXTemp, testYTemp = train_norm[train_ix], trainY[train_ix], train_norm
[test_ix], trainY[test_ix]

    model.fit(trainXTemp, trainYTemp, epochs=10, batch_size=128, validation_data=(testXTemp, testYTe
mp), verbose=0)

    loss, acc = model.evaluate(testXTemp, testYTemp, verbose=0)
    print('accuracy: %.3f' % (acc * 100.0))
    print('loss: %.7f' % (loss))

    scores.append(acc)
    losses.append(loss)
  return scores , losses,
```

```
def Build_Model_3(model , train_norm , trainY):
  scores , losses = list() , list()
  kfold = KFold(5, shuffle=True, random_state=1)
  for train_ix, test_ix in kfold.split(train_norm):
    opt=RMSprop(lr=0.001, decay=0.00001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    trainXTemp, trainYTemp, testXTemp, testYTemp = train_norm[train_ix], trainY[train_ix], train_norm
[test_ix], trainY[test_ix]

    model.fit(trainXTemp, trainYTemp, epochs=10, batch_size=128, validation_data=(testXTemp, testYTe
mp), verbose=0)

    loss, acc = model.evaluate(testXTemp, testYTemp, verbose=0)

    print('accuracy: %.3f' % (acc * 100.0))
    print('loss: %.7f' % (loss))
    scores.append(acc)
    losses.append(loss)
  return scores , losses
```

#**Load the data**

```
(trainX, trainY), (testX, testY) = mnist.load_data()

trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

testX = testX.reshape((testX.shape[0], 28, 28, 1))

trainY = to_categorical(trainY)

testY = to_categorical(testY)

train_norm = trainX.astype('float32')

test_norm = testX.astype('float32')

train_norm = train_norm / 255.0

test_norm = test_norm / 255.0
```

```
#Model 1 test 1

model_1 = Sequential()

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 1

number of parameters	**59,926**
The execution time	15 minutes
The learning rate	0.01 (momentum=0.5)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	20
The epochs	12
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.917 loss: 0.0654842

accuracy: 99.150 loss: 0.0269630

accuracy: 99.567 loss: 0.0126277

accuracy: 99.900 loss: 0.0027203

accuracy: 99.975 loss: 0.0010514

Model 1 test 2

number of parameters	**59,926**
The execution time	6 minutes
The learning rate	0.01 (momentum=0.5)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	32
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.258 loss: 0.0745163

accuracy: 99.308 loss: 0.0218641

accuracy: 99.883 loss: 0.0034714

accuracy: 99.975 loss: 0.0009744

accuracy: 99.958 loss: 0.0009536

Model 1 test 3

number of parameters	**59,926**
The execution time	5 minutes
The learning rate	0.01 (momentum=0.5)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	60
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.100 loss: 0.0631513

accuracy: 98.775 loss: 0.0380671

accuracy: 99.242 loss: 0.0240674

accuracy: 99.442 loss: 0.0162485

accuracy: 99.600 loss: 0.0110394

Observation:

Increasing the batch size from 20 to 60 and decrease the epochs from 12 to 10 reduce the model accuracy and it's execution time and increase the loss

Model 1 test 4

number of parameters	**59,926**
The execution time	2 minutes
The learning rate	0.01 (momentum=0.1)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.100 loss: 0.0940685

accuracy: 97.542 loss: 0.0726603

accuracy: 98.358 loss: 0.0519563

accuracy: 98.950 loss: 0.0343243

accuracy: 98.917 loss: 0.0341533

Observation:

Reducing the momentum to (0.1) reduces the model accuracy.

```
model_1 = Sequential()

model_1.add(Conv2D(32, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(32, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(32, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 5

number of parameters	**22,326**
The execution time	2 minutes
The learning rate	0.01
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 95.742 loss: 0.1369589

accuracy: 96.867 loss: 0.0985484

accuracy: 97.792 loss: 0.0732227

accuracy: 98.350 loss: 0.0517636

accuracy: 98.292 loss: 0.0530156

Observation:

Using 32 channels in the CNN's reduce the model accuracy.

```
model_1 = Sequential()

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1
)))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1
)))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1
)))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(50, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 6

number of parameters	**46,576**	
The execution time	3 minutes	
The learning rate	0.01	
The layers	3 CNN's – 2 FC's	
The optimizer	SGD	
The batch size	128	
The epochs	10	
The activation function	ReLU	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 96.883 loss: 0.0983839

accuracy: 97.967 loss: 0.0681203

accuracy: 98.458 loss: 0.0517966

accuracy: 98.817 loss: 0.0359416

accuracy: 98.900 loss: 0.0343441

Observation:

Using 50 channels in the FC layer reduce the model accuracy and increase the execution time and the number of parameters.

```
model_1 = Sequential()

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 7

number of parameters	**59,926**
The execution time	2 minutes
The learning rate	0.1 (momentum=0.5)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.150 loss: 0.0944306

accuracy: 98.150 loss: 0.0636358

accuracy: 98.583 loss: 0.0491753

accuracy: 98.875 loss: 0.0327443

accuracy: 99.000 loss: 0.0314264

Model 1 test 8

number of parameters	**59,926**
The execution time	2 minutes
The learning rate	0.1 (momentum=0.5)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.900 loss: 0.0813774

accuracy: 98.400 loss: 0.0744248

accuracy: 98.692 loss: 0.0603818

accuracy: 98.858 loss: 0.0495050

accuracy: 98.983 loss: 0.0583437

Observation:

Using large learning rate (0.1) instead of (0.01) decreases the model accuracy and also the execution time.

Model 1 test 9

number of parameters	**59,926**
The execution time	35 sec
The learning rate	0.01 (momentum=0.9)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	1
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 96.050 loss: 0.1303196

accuracy: 96.867 loss: 0.0969248

accuracy: 97.650 loss: 0.0725034

accuracy: 97.783 loss: 0.0646960

accuracy: 98.575 loss: 0.0476408

Observation:

Decreasing the epochs number with the same batch size leads to decreasing the model accuracy and increase the loss.

```
model_1 = Sequential()

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 10

number of parameters	**59,926**
The execution time	5 mins
The learning rate	0.01 (momentum=0.9)
The layers	3 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.417 loss: 0.0606659

accuracy: 99.267 loss: 0.0247502

accuracy: 99.475 loss: 0.0154012

accuracy: 99.925 loss: 0.0025019

accuracy: 99.992 loss: 0.0007493

Model 1 test 11

number of parameters	**248,278**
The execution time	5 mins
The learning rate	0.01 (momentum=0.9)
The layers	2 CNN's – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.658 loss: 0.0470705

accuracy: 99.333 loss: 0.0181307

accuracy: 99.817 loss: 0.0057304

accuracy: 99.983 loss: 0.0012515

accuracy: 100.000 loss: 0.0005472

```
model_1 = Sequential()

model_1.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_1.add(MaxPooling2D((2, 2)))

model_1.add(Flatten())

model_1.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_1.add(Dense(10, activation='softmax'))

scores , losses = Build_model_1(model_1,train_norm ,trainY)
```

Model 1 test 12

number of parameters	**1,083,030**
The execution time	3 mins
The learning rate	0.01 (momentum=0.9)
The layers	1 CNN – 2 FC's
The optimizer	SGD
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.02 loss: 0.0663515

accuracy: 99.208 loss: 0.0264444

accuracy: 99.675 loss: 0.0099402

accuracy: 99.950 loss: 0.0031479

accuracy: 99.975 loss: 0.0018670

Observation:

Decreasing the number of convolutional layers increase the parameters number.

When removing some convolutional layers it means that the image size before being passed to the fully connected layers is larger and contains more pixels. Since each neuron in the fully connected layer is connected to all pixels in the layer before it the increase in the number of parameters caused by the larger image size is larger than the decrease in the number of parameters caused by the removal of the convolutional layers

```
model_2 = Sequential()

model_2.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Flatten())

model_2.add(Dropout(0.3))

model_2.add(Dense(100, activation='tanh', kernel_initializer='he_uniform'))

model_2.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_2(model_2,train_norm ,trainY)
```

Model 2 test 1

number of parameters	**59,926**
The execution time	5 mins
The learning rate	0.01 (decay=0.00001)
The layers	3 CNN's – 1 Dropout – 2 FC 's
The optimizer	Adam
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 96.358 loss: 0.1212646

accuracy: 95.750 loss: 0.1318936

accuracy: 96.017 loss: 0.1309534

accuracy: 96.150 loss: 0.1220197

accuracy: 96.283 loss: 0.1249149

Observation:

Using ‘Adam’ optimizer and adding dropout layer with rate 0.3 reduce the accuracy of the model and increase the loss.

```
model_2 = Sequential()

model_2.add(Conv2D(64, (2, 2), activation='sigmoid', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='sigmoid', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='sigmoid', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Flatten())

model_2.add(Dropout(0.1))

model_2.add(Dense(100, activation='sigmoid', kernel_initializer='he_uniform'))

model_2.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_2(model_2,train_norm ,trainY)
```

Model 2 test 2

number of parameters	**59,926**
The execution time	5 mins
The learning rate	0.01 (decay=0.00001)
The layers	3 CNN's – 1 Dropout – 2 FC 's
The optimizer	Adam
The batch size	128
The epochs	10
The activation function	sigmoid

Final accuracy of the model in the first 5 epoch:

accuracy: 94.067 loss: 0.1907499

accuracy: 96.300 loss: 0.1228421

accuracy: 97.358 loss: 0.0865857

accuracy: 97.625 loss: 0.0806135

accuracy: 97.883 loss: 0.0695184

Observation:

using ‘sigmoid’ as an activation function and adding dropout layer with rate 0.1 **increase** the accuracy of the model more than ‘tanh’ and also **decrease** the loss.

```
model_2 = Sequential()

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Flatten())

model_2.add(Dropout(0.3))

model_2.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_2.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_2(model_2,train_norm ,trainY)
```

Model 2 test 3

number of parameters	**59,926**
The execution time	5 mins
The learning rate	0.01 (decay=0.00001)
The layers	3 CNN's – 1 Dropout – 2 FC 's
The optimizer	Adam
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.867 loss: 0.0697569

accuracy: 98.367 loss: 0.0554662

accuracy: 98.442 loss: 0.0596680

accuracy: 98.542 loss: 0.0524064

accuracy: 98.158 loss: 0.0648085

Observation:

Using ‘ReLU’as an activation function **increases** the accuracy of the model more than ‘tanh’& ‘sigmoid’.

```
model_2 = Sequential()

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Flatten())

model_2.add(Dropout(0.3))

model_2.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_2.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_2(model_2,train_norm ,trainY)
```

Model 2 test 4

number of parameters	**248,278**
The execution time	4 mins
The learning rate	0.01 (decay=0.00001)
The layers	2 CNN's – 1 Dropout – 2 FC 's
The optimizer	Adam
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 98.275 loss: 0.0561873

accuracy: 98.875 loss: 0.0386083

accuracy: 99.083 loss: 0.0329114

accuracy: 99.158 loss: 0.0274989

accuracy: 99.175 loss: 0.0283319

Observation:

Decreasing the number of convolutional layers and add dropout layer with rate 0.3 **increase** the model accuracy.

```
model_2 = Sequential()

model_2.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1 )))

model_2.add(MaxPooling2D((2, 2)))

model_2.add(Flatten())

model_2.add(Dropout(0.3))

model_2.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))

model_2.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_2(model_2,train_norm ,trainY)
```

Model 2 test 5

number of parameters	**1,083,030**
The execution time	3 mins
The learning rate	0.01 (decay=0.00001)
The layers	1 CNN – 1 Dropout – 2 FC ‘s
The optimizer	Adam
The batch size	128
The epochs	10
The activation function	ReLU

Final accuracy of the model in the first 5 epoch:

accuracy: 97.317 loss: 0.1215623

accuracy: 98.500 loss: 0.0658872

accuracy: 99.183 loss: 0.0359311

accuracy: 99.500 loss: 0.0149898

accuracy: 99.267 loss: 0.0326141

```
model_3 = Sequential()

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Flatten())

model_3.add(Dropout(0.3))

model_3.add(Dense(100, activation='tanh', kernel_initializer='he_uniform'))

model_3.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_3(model_3,train_norm ,trainY)
```

Model 3 test 1

number of parameters	**248,278**
The execution time	3 mins
The learning rate	0.001 (decay=0.00001)
The layers	2 CNN – 1 Dropout – 2 FC ‘s
The optimizer	RMSprop
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 98.600 loss: 0.0538694

accuracy: 99.592 loss: 0.0131574

accuracy: 99.842 loss: 0.0045087

accuracy: 99.925 loss: 0.0015018

accuracy: 99.967 loss: 0.0012394

Observation:

Decreasing learning rate to 0.001 and add use RMSprop optimizer **increase** the model accuracy and **decrease** the loss.

```
model_3 = Sequential()

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Flatten())

model_3.add(Dropout(0.3))

model_3.add(Dense(100, activation='tanh', kernel_initializer='he_uniform'))

model_3.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_3(model_3,train_norm ,trainY)
```

Model3 test 2

number of parameters	**1,083,030**
The execution time	3 mins
The learning rate	0.001 (decay=0.00001)
The layers	1 CNN – 1 Dropout – 2 FC ‘s
The optimizer	RMSprop
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 97.908 loss: 0.0737287

accuracy: 99.292 loss: 0.0227477

accuracy: 99.675 loss: 0.0094607

accuracy: 99.917 loss: 0.0023665

accuracy: 99.942 loss: 0.0026654

```
model_3 = Sequential()

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Conv2D(64, (2, 2), activation='tanh', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Flatten())

model_3.add(Dropout(0.3))

model_3.add(Dense(100, activation='tanh', kernel_initializer='he_uniform'))

model_3.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_3(model_3,train_norm ,trainY)
```

model 3 test 3

number of parameters	**59,926**
The execution time	6 mins
The learning rate	0.001 (decay=0.00001)
The layers	3 CNN – 1 Dropout – 2 FC ‘s
The optimizer	RMSprop
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 98.608 loss: 0.0499678

accuracy: 99.200 loss: 0.0239296

accuracy: 99.483 loss: 0.0160271

accuracy: 99.725 loss: 0.0087766

accuracy: 99.858 loss: 0.0052837

```
model_3 = Sequential()

model_3.add(Conv2D(64, (2, 2), activation='sigmoid', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Conv2D(64, (2, 2), activation='sigmoid', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))

model_3.add(MaxPooling2D((2, 2)))

model_3.add(Flatten())

model_3.add(Dropout(0.3))

model_3.add(Dense(100, activation='sigmoid', kernel_initializer='he_uniform'))

model_3.add(Dense(10, activation='softmax'))

scores , losses = Build_Model_3(model_3,train_norm ,trainY)
```

model 3 test 4

number of parameters	**248,278**
The execution time	5 mins
The learning rate	0.001 (decay=0.00001)
The layers	2 CNN – 1 Dropout – 2 FC ‘s
The optimizer	RMSprop
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 98.183 loss: 0.0633895

accuracy: 98.883 loss: 0.0372791

accuracy: 99.125 loss: 0.0264763

accuracy: 99.500 loss: 0.0144612

accuracy: 99.617 loss: 0.0129995

```
model_3 = Sequential()
model_3.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model_3.add(MaxPooling2D((2, 2)))
model_3.add(Conv2D(64, (2, 2), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model_3.add(MaxPooling2D((2, 2)))
model_3.add(Flatten())
model_3.add(Dropout(0.3))
model_3.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
model_3.add(Dense(10, activation='softmax'))
scores , losses = Build_Model_3(model_3,train_norm ,trainY)
```

Model 3 test 5

number of parameters	**248,278**
The execution time	6 mins
The learning rate	0.001 (decay=0.00001)
The layers	1 CNN – 1 Dropout – 2 FC ‘s
The optimizer	RMSprop
The batch size	128
The epochs	10
The activation function	tanh

Final accuracy of the model in the first 5 epoch:

accuracy: 98.867 loss: 0.0502256

accuracy: 99.483 loss: 0.0187512

accuracy: 99.825 loss: 0.0074756

accuracy: 99.917 loss: 0.0022337

accuracy: 99.908 loss: 0.0038617

The best model we've achieved is model 3 (test 1), with accuracy 99.96% and loss 0.001

The model has a dropout layer to avoid over fitting like in model 1 (test 5) which has accuracy 100% and the same number of parameters (248,278) but with less execution time (3 minutes).

It performed better than the other models because of the small learning rate and the optimizer used (RMSprop).