<Assigment_3_Supervised_learning>

<Hussein Hesham Hussein Badawy> <20190183>

<Nour El-Din Ahmed Ezzat> <20190593>

==

Handwritten digit recognition :

- **Firstly, we will train a CNN (Convolutional Neural Network) on MNIST dataset, which contains a total of 70,000 images of handwritten digits from 0-9 formatted as 28×28-pixel monochrome images.**

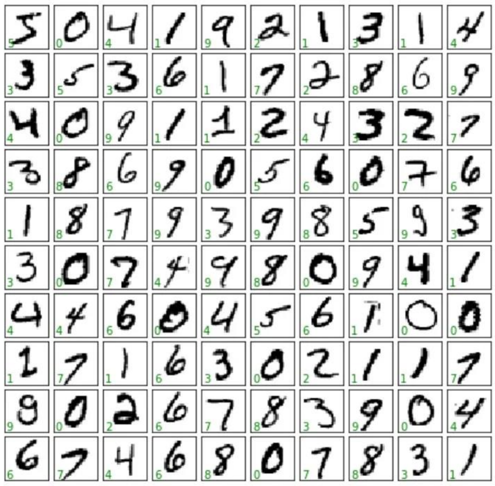
- **For this, we will first split the dataset into train and test data with size 60,000 and 10,000 respectively.**

- **Then, we will preprocess the input data by reshaping the image and scaling the pixel values between 0 and 1.**

- **After that, we will design the neural network and train the model.**

MINIST dataset

- **The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning.**



Convolutional Neural Network

- **CNN is one of the most important neural network models for computing tasks based on multi-layered perceptron. These models perform particularly well for the processing of images. For instance, recognition of handwriting. Handwriting Recognition is one of neural networks' most basic and excellent uses. CNN model is trained in multiple layers to make the correct predictions .**

Building CNN model

- **# 1. Import the necessary libraries and modules**

```
from tensorflow import keras
from keras.utils import np_utils
import numpy as np
import tensorflow as tf
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

- **# 2. Splitting the MNIST dataset into Train and Test**

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

- **# 3. Pre-processing the input data :**

```
num_of_trainImgs = x_train.shape[0] #60000 training images
num_of_testImgs = x_test.shape[0] #10000 testing images
img_width = 28
img_height = 28          # 28 x 28 pixels

 #  transform the 3-D data into a 4-D dataset
x_train = x_train.reshape(x_train.shape[0], img_height, img_width, 1)
x_test = x_test.reshape(x_test.shape[0], img_height, img_width, 1)
input_shape = (img_height, img_width, 1)
 # Normalizing the data, for which first the data is convered to float and then
it is divided by 255
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

*** The data size is (60000,28,28), which translates to 60000 photos of 28 x 28 pixels each.**

*** We require a 4-dimensional array dataset to apply Keras , thus we must transform the 3-D**
data into a 4-D dataset.

*** Normalizing the data, for which first the data is converted to float and t hen it is divided by 255.**

- # 4. Converting the class vectors to binary class :

```
#[0,1,2,3,4,5,6,7,8,9]

num_classes = 10

y_train = np_utils.to_categorical(y_train)

y_test = np_utils.to_categorical(y_test)

 # encode output which is a number in range [0,9] into a vector of size 10
    # ex 1 -> [1 0 0 0 0 0 0 0 0 0]
    #    2 -> [0 1 0 0 0 0 0 0 0 0]
```

*** encode output which is a number in range [0,9] into a vector of size 10**
*** ex 1 -> [1 0 0 0 0 0 0 0 0 0]**
***   2 -> [0 1 0 0 0 0 0 0 0 0]**

- # 5. Defining the model architecture :

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

*** In order to do that we will be importing the Sequential Model from Keras and adding multiple layers**

MODEL

- # 6. Compiling the model :

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

*** set an optimizer with a given loss function :**
*** The cross-entropy loss function is an optimization function that is used for training classification models which classify the data by predicting the probability (value between 0 and 1) of whether the data belong to one class or another.**

*** Adadelta optimizer continues learning even when many updates have been done.**

- **# 7. Fitting the model on training data :**

#1 is testing the learning rate

❖ lr = 0.001

```
model.fit(x_train, y_train,
          batch_size=128,
          epochs=50,
          verbose=1,
          validation_data=(x_test, y_test))

K.set_value(model.optimizer.learning_rate, 0.001)
print("Learning rate before second fit:", model.optimizer.learning_rate.numpy())
```

```
Epoch 35/50
469/469 [==============================] - 4s 9ms/step - loss: 0.5211 - accuracy: 0.8395 - val_loss: 0.3414 - val_accuracy: 0.9076
Epoch 36/50
469/469 [==============================] - 4s 9ms/step - loss: 0.5171 - accuracy: 0.8422 - val_loss: 0.3366 - val_accuracy: 0.9086
Epoch 37/50
469/469 [==============================] - 4s 9ms/step - loss: 0.5123 - accuracy: 0.8447 - val_loss: 0.3324 - val_accuracy: 0.9094
Epoch 38/50
469/469 [==============================] - 4s 9ms/step - loss: 0.5049 - accuracy: 0.8461 - val_loss: 0.3280 - val_accuracy: 0.9103
Epoch 39/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4971 - accuracy: 0.8474 - val_loss: 0.3239 - val_accuracy: 0.9106
Epoch 40/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4965 - accuracy: 0.8484 - val_loss: 0.3202 - val_accuracy: 0.9108
Epoch 41/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4886 - accuracy: 0.8509 - val_loss: 0.3165 - val_accuracy: 0.9117
Epoch 42/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4834 - accuracy: 0.8526 - val_loss: 0.3124 - val_accuracy: 0.9131
Epoch 43/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4753 - accuracy: 0.8529 - val_loss: 0.3091 - val_accuracy: 0.9134
Epoch 44/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4724 - accuracy: 0.8551 - val_loss: 0.3057 - val_accuracy: 0.9145
Epoch 45/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4712 - accuracy: 0.8562 - val_loss: 0.3026 - val_accuracy: 0.9147
Epoch 46/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4630 - accuracy: 0.8588 - val_loss: 0.2997 - val_accuracy: 0.9153
Epoch 47/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4611 - accuracy: 0.8605 - val_loss: 0.2966 - val_accuracy: 0.9164
Epoch 48/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4571 - accuracy: 0.8612 - val_loss: 0.2940 - val_accuracy: 0.9165
Epoch 49/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4540 - accuracy: 0.8632 - val_loss: 0.2912 - val_accuracy: 0.9168
Epoch 50/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4477 - accuracy: 0.8643 - val_loss: 0.2884 - val_accuracy: 0.9180
Learning rate before second fit: 0.001
```

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2884 - accuracy: 0.9180
Test loss: 0.2883935272693634
Test accuracy: 0.9179999828338623
```

- lr = 0.01

```
469/469 [==============================] - 4s 9ms/step - loss: 0.3651 - accuracy: 0.8909 - val_loss: 0.2282 - val_accuracy: 0.9306
Epoch 35/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3618 - accuracy: 0.8906 - val_loss: 0.2265 - val_accuracy: 0.9315
Epoch 36/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3589 - accuracy: 0.8923 - val_loss: 0.2251 - val_accuracy: 0.9325
Epoch 37/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3584 - accuracy: 0.8912 - val_loss: 0.2244 - val_accuracy: 0.9319
Epoch 38/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3579 - accuracy: 0.8919 - val_loss: 0.2227 - val_accuracy: 0.9335
Epoch 39/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3603 - accuracy: 0.8923 - val_loss: 0.2219 - val_accuracy: 0.9332
Epoch 40/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3543 - accuracy: 0.8931 - val_loss: 0.2207 - val_accuracy: 0.9332
Epoch 41/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3554 - accuracy: 0.8929 - val_loss: 0.2196 - val_accuracy: 0.9336
Epoch 42/50
469/469 [==============================] - 5s 10ms/step - loss: 0.3509 - accuracy: 0.8943 - val_loss: 0.2182 - val_accuracy: 0.9341
Epoch 43/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3493 - accuracy: 0.8959 - val_loss: 0.2172 - val_accuracy: 0.9348
Epoch 44/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3477 - accuracy: 0.8959 - val_loss: 0.2160 - val_accuracy: 0.9348
Epoch 45/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3485 - accuracy: 0.8952 - val_loss: 0.2151 - val_accuracy: 0.9350
Epoch 46/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3454 - accuracy: 0.8962 - val_loss: 0.2138 - val_accuracy: 0.9355
Epoch 47/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3421 - accuracy: 0.8969 - val_loss: 0.2128 - val_accuracy: 0.9360
Epoch 48/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3466 - accuracy: 0.8972 - val_loss: 0.2118 - val_accuracy: 0.9362
Epoch 49/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3424 - accuracy: 0.8970 - val_loss: 0.2107 - val_accuracy: 0.9367
Epoch 50/50
469/469 [==============================] - 4s 9ms/step - loss: 0.3402 - accuracy: 0.8980 - val_loss: 0.2096 - val_accuracy: 0.9366
Learning rate : 0.01
```

- lr = 0.1

```
Epoch 35/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1394 - accuracy: 0.9581 - val_loss: 0.0767 - val_accuracy: 0.9766
Epoch 36/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1393 - accuracy: 0.9588 - val_loss: 0.0754 - val_accuracy: 0.9763
Epoch 37/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1365 - accuracy: 0.9601 - val_loss: 0.0741 - val_accuracy: 0.9772
Epoch 38/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1328 - accuracy: 0.9617 - val_loss: 0.0729 - val_accuracy: 0.9773
Epoch 39/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1353 - accuracy: 0.9600 - val_loss: 0.0719 - val_accuracy: 0.9776
Epoch 40/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1297 - accuracy: 0.9613 - val_loss: 0.0704 - val_accuracy: 0.9777
Epoch 41/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1281 - accuracy: 0.9621 - val_loss: 0.0695 - val_accuracy: 0.9781
Epoch 42/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1269 - accuracy: 0.9628 - val_loss: 0.0684 - val_accuracy: 0.9787
Epoch 43/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1237 - accuracy: 0.9633 - val_loss: 0.0676 - val_accuracy: 0.9783
Epoch 44/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1221 - accuracy: 0.9637 - val_loss: 0.0664 - val_accuracy: 0.9787
Epoch 45/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1205 - accuracy: 0.9646 - val_loss: 0.0654 - val_accuracy: 0.9789
Epoch 46/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1210 - accuracy: 0.9639 - val_loss: 0.0647 - val_accuracy: 0.9787
Epoch 47/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1181 - accuracy: 0.9651 - val_loss: 0.0640 - val_accuracy: 0.9792
Epoch 48/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1179 - accuracy: 0.9642 - val_loss: 0.0637 - val_accuracy: 0.9798
Epoch 49/50
469/469 [==============================] - 4s 10ms/step - loss: 0.1115 - accuracy: 0.9670 - val_loss: 0.0622 - val_accuracy: 0.9803
Epoch 50/50
469/469 [==============================] - 4s 9ms/step - loss: 0.1155 - accuracy: 0.9656 - val_loss: 0.0614 - val_accuracy: 0.9805
Learning rate : 0.1
```

✓ 3m 43s completed at 20:06

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0614 - accuracy: 0.9805
Test loss: 0.06137840077280998
Test accuracy: 0.9804999828338623
```

Conclusion

As observed that model 1by increasing the learning rate value our test loss decreased significantly as we ran our model for 50 epochs and accuracy improved to over 98 % which is the best result we got overall .

So the best result we go by changing lr parameter and make other parameters unchanged is that setting lr = 0.1 which is the highest value we used

#2 is testing the number of epoch

- Epoch = 50

```
469/469 [==============================] - 4s 9ms/step - loss: 0.4903 - accuracy: 0.8501 - val_loss: 0.3235 - val_accuracy: 0.9074
Epoch 41/50
469/469 [==============================] - 4s 10ms/step - loss: 0.4829 - accuracy: 0.8542 - val_loss: 0.3195 - val_accuracy: 0.9084
Epoch 42/50
469/469 [==============================] - 4s 10ms/step - loss: 0.4815 - accuracy: 0.8546 - val_loss: 0.3161 - val_accuracy: 0.9092
Epoch 43/50
469/469 [==============================] - 5s 10ms/step - loss: 0.4741 - accuracy: 0.8573 - val_loss: 0.3123 - val_accuracy: 0.9116
Epoch 44/50
469/469 [==============================] - 5s 10ms/step - loss: 0.4715 - accuracy: 0.8579 - val_loss: 0.3090 - val_accuracy: 0.9119
Epoch 45/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4658 - accuracy: 0.8587 - val_loss: 0.3061 - val_accuracy: 0.9128
Epoch 46/50
469/469 [==============================] - 4s 10ms/step - loss: 0.4630 - accuracy: 0.8605 - val_loss: 0.3031 - val_accuracy: 0.9139
Epoch 47/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4536 - accuracy: 0.8631 - val_loss: 0.2997 - val_accuracy: 0.9138
Epoch 48/50
469/469 [==============================] - 4s 9ms/step - loss: 0.4512 - accuracy: 0.8654 - val_loss: 0.2968 - val_accuracy: 0.9144
Epoch 49/50
469/469 [==============================] - 4s 10ms/step - loss: 0.4500 - accuracy: 0.8645 - val_loss: 0.2940 - val_accuracy: 0.9153
Epoch 50/50
469/469 [==============================] - 5s 10ms/step - loss: 0.4452 - accuracy: 0.8656 - val_loss: 0.2915 - val_accuracy: 0.9159
Learning rate : 0.1
```

```python
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2915 - accuracy: 0.9159
Test loss: 0.29149067401885986
Test accuracy: 0.9158999919891357
```

❖ Epoch = 70

```
Epoch 56/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4406 - accuracy: 0.8666 - val_loss: 0.2880 - val_accuracy: 0.9169
Epoch 57/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4334 - accuracy: 0.8686 - val_loss: 0.2859 - val_accuracy: 0.9173
Epoch 58/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4327 - accuracy: 0.8685 - val_loss: 0.2837 - val_accuracy: 0.9177
Epoch 59/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4278 - accuracy: 0.8718 - val_loss: 0.2812 - val_accuracy: 0.9182
Epoch 60/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4273 - accuracy: 0.8705 - val_loss: 0.2790 - val_accuracy: 0.9186
Epoch 61/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4212 - accuracy: 0.8740 - val_loss: 0.2767 - val_accuracy: 0.9191
Epoch 62/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4181 - accuracy: 0.8736 - val_loss: 0.2747 - val_accuracy: 0.9202
Epoch 63/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4184 - accuracy: 0.8730 - val_loss: 0.2726 - val_accuracy: 0.9207
Epoch 64/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4157 - accuracy: 0.8750 - val_loss: 0.2702 - val_accuracy: 0.9214
Epoch 65/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4136 - accuracy: 0.8740 - val_loss: 0.2687 - val_accuracy: 0.9223
Epoch 66/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4080 - accuracy: 0.8778 - val_loss: 0.2668 - val_accuracy: 0.9223
Epoch 67/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4086 - accuracy: 0.8750 - val_loss: 0.2644 - val_accuracy: 0.9228
Epoch 68/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4084 - accuracy: 0.8757 - val_loss: 0.2627 - val_accuracy: 0.9234
Epoch 69/70
469/469 [==============================] - 4s 9ms/step - loss: 0.3990 - accuracy: 0.8785 - val_loss: 0.2610 - val_accuracy: 0.9238
Epoch 70/70
469/469 [==============================] - 4s 9ms/step - loss: 0.4011 - accuracy: 0.8788 - val_loss: 0.2592 - val_accuracy: 0.9242
Learning rate : 0.1
```

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2592 - accuracy: 0.9242
Test loss: 0.2592056691646576
Test accuracy: 0.9241999983787537
```

❖ Epoch = 100

```
Epoch 87/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3543 - accuracy: 0.8928 - val_loss: 0.2225 - val_accuracy: 0.9348
Epoch 88/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3518 - accuracy: 0.8925 - val_loss: 0.2214 - val_accuracy: 0.9350
Epoch 89/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3524 - accuracy: 0.8945 - val_loss: 0.2203 - val_accuracy: 0.9352
Epoch 90/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3535 - accuracy: 0.8939 - val_loss: 0.2195 - val_accuracy: 0.9351
Epoch 91/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3516 - accuracy: 0.8942 - val_loss: 0.2185 - val_accuracy: 0.9355
Epoch 92/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3504 - accuracy: 0.8966 - val_loss: 0.2174 - val_accuracy: 0.9354
Epoch 93/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3487 - accuracy: 0.8958 - val_loss: 0.2166 - val_accuracy: 0.9361
Epoch 94/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3478 - accuracy: 0.8965 - val_loss: 0.2157 - val_accuracy: 0.9363
Epoch 95/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3424 - accuracy: 0.8978 - val_loss: 0.2145 - val_accuracy: 0.9370
Epoch 96/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3467 - accuracy: 0.8958 - val_loss: 0.2136 - val_accuracy: 0.9370
Epoch 97/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3441 - accuracy: 0.8976 - val_loss: 0.2129 - val_accuracy: 0.9367
Epoch 98/100
469/469 [==============================] - 4s 10ms/step - loss: 0.3426 - accuracy: 0.8962 - val_loss: 0.2119 - val_accuracy: 0.9376
Epoch 99/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3385 - accuracy: 0.8988 - val_loss: 0.2109 - val_accuracy: 0.9373
Epoch 100/100
469/469 [==============================] - 4s 10ms/step - loss: 0.3375 - accuracy: 0.8990 - val_loss: 0.2099 - val_accuracy: 0.9373
Learning rate : 0.1
```

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2099 - accuracy: 0.9373
Test loss: 0.20986227691173553
Test accuracy: 0.9373000264167786
```

Conclusion

Lr = 0.1 epochs=100 test-accuracy = 93.73%

As observed that model 1 by increasing the ephoch our test loss decreased as we ran our model for 100 epochs and accuracy improved to over 93 %

But from 70 epochs to 100 epochs that's not a big improvement so that there is no need to to increase number of epochs more than 100 Looking to execution time also .

#3 is testing the batch_size

- Batch_size= 128

```
Epoch 87/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3543 - accuracy: 0.8928 - val_loss: 0.2225 - val_accuracy: 0.9348
Epoch 88/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3518 - accuracy: 0.8925 - val_loss: 0.2214 - val_accuracy: 0.9350
Epoch 89/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3524 - accuracy: 0.8945 - val_loss: 0.2203 - val_accuracy: 0.9352
Epoch 90/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3535 - accuracy: 0.8939 - val_loss: 0.2195 - val_accuracy: 0.9351
Epoch 91/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3516 - accuracy: 0.8942 - val_loss: 0.2185 - val_accuracy: 0.9355
Epoch 92/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3504 - accuracy: 0.8966 - val_loss: 0.2174 - val_accuracy: 0.9354
Epoch 93/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3487 - accuracy: 0.8958 - val_loss: 0.2166 - val_accuracy: 0.9361
Epoch 94/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3478 - accuracy: 0.8965 - val_loss: 0.2157 - val_accuracy: 0.9363
Epoch 95/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3424 - accuracy: 0.8978 - val_loss: 0.2145 - val_accuracy: 0.9370
Epoch 96/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3467 - accuracy: 0.8958 - val_loss: 0.2136 - val_accuracy: 0.9370
Epoch 97/100
469/469 [==============================] - 4s 9ms/step - loss: 0.3441 - accuracy: 0.8976 - val_loss: 0.2129 - val_accuracy: 0.9367
Epoch 98/100
469/469 [==============================] - 4s 10ms/step - loss: 0.3426 - accuracy: 0.8962 - val_loss: 0.2119 - val_accuracy: 0.9376
Epoch 99/100
469/469 [==============================] - 5s 10ms/step - loss: 0.3385 - accuracy: 0.8988 - val_loss: 0.2109 - val_accuracy: 0.9373
Epoch 100/100
469/469 [==============================] - 4s 10ms/step - loss: 0.3375 - accuracy: 0.8990 - val_loss: 0.2099 - val_accuracy: 0.9373
Learning rate : 0.1
```

```python
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2099 - accuracy: 0.9373
Test loss: 0.20986227691173553
Test accuracy: 0.9373000264167786
```

❖ Batch_size= 64

```
Epoch 85/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3186 - accuracy: 0.9043 - val_loss: 0.1943 - val_accuracy: 0.9429
Epoch 86/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3151 - accuracy: 0.9055 - val_loss: 0.1934 - val_accuracy: 0.9431
Epoch 87/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3112 - accuracy: 0.9068 - val_loss: 0.1924 - val_accuracy: 0.9435
Epoch 88/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3156 - accuracy: 0.9053 - val_loss: 0.1912 - val_accuracy: 0.9439
Epoch 89/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3098 - accuracy: 0.9074 - val_loss: 0.1901 - val_accuracy: 0.9439
Epoch 90/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3097 - accuracy: 0.9084 - val_loss: 0.1889 - val_accuracy: 0.9447
Epoch 91/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3119 - accuracy: 0.9064 - val_loss: 0.1882 - val_accuracy: 0.9451
Epoch 92/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3049 - accuracy: 0.9078 - val_loss: 0.1870 - val_accuracy: 0.9447
Epoch 93/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3027 - accuracy: 0.9092 - val_loss: 0.1864 - val_accuracy: 0.9450
Epoch 94/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3053 - accuracy: 0.9086 - val_loss: 0.1855 - val_accuracy: 0.9462
Epoch 95/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3017 - accuracy: 0.9097 - val_loss: 0.1846 - val_accuracy: 0.9460
Epoch 96/100
938/938 [==============================] - 6s 6ms/step - loss: 0.2988 - accuracy: 0.9105 - val_loss: 0.1838 - val_accuracy: 0.9461
Epoch 97/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3011 - accuracy: 0.9109 - val_loss: 0.1827 - val_accuracy: 0.9469
Epoch 98/100
938/938 [==============================] - 6s 6ms/step - loss: 0.3001 - accuracy: 0.9102 - val_loss: 0.1816 - val_accuracy: 0.9472
Epoch 99/100
938/938 [==============================] - 6s 6ms/step - loss: 0.2934 - accuracy: 0.9136 - val_loss: 0.1805 - val_accuracy: 0.9472
Epoch 100/100
938/938 [==============================] - 6s 6ms/step - loss: 0.2972 - accuracy: 0.9108 - val_loss: 0.1800 - val_accuracy: 0.9481
Learning rate : 0.1
```

```python
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1800 - accuracy: 0.9481
Test loss: 0.1799737811088562
Test accuracy: 0.9480999708175659
```

Conclusion

Lr = 0.1 epochs=100 batch_size = 64 test-accuracy = 94.81%

As observed that model 1 by setting batch_size to 64 test loss decreased as we ran our model for 100 epochs and accuracy improved to over 94 %

But also execution time significantly increased

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_12 (Conv2D)          (None, 26, 26, 32)        320       
                                                                 
 conv2d_13 (Conv2D)          (None, 24, 24, 64)        18496     
                                                                 
 max_pooling2d_6 (MaxPooling  (None, 12, 12, 64)       0         
 2D)                                                             
                                                                 
 dropout_12 (Dropout)        (None, 12, 12, 64)        0         
                                                                 
 flatten_6 (Flatten)         (None, 9216)              0         
                                                                 
 dense_12 (Dense)            (None, 128)               1179776   
                                                                 
 dropout_13 (Dropout)        (None, 128)               0         
                                                                 
 dense_13 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

==

MODEL

With 'adam' optimizer

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
#################################### Creating a neural networknow #################
##
model = Sequential()

model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation=tf.nn.relu))

model.add(Dropout(0.2))

model.add(Dense(10,activation=tf.nn.softmax))
```

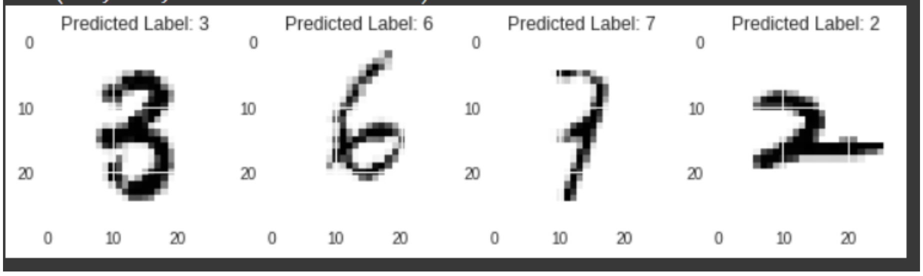
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a
ccuracy'])
model.fit(x=x_train,y=y_train, epochs=10 , batch_size=64)

K.set_value(model.optimizer.learning_rate, 0.1)
print("Learning rate :", model.optimizer.learning_rate.numpy())
```

```
Epoch 1/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2288 - accuracy: 0.9321
Epoch 2/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0934 - accuracy: 0.9712
Epoch 3/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0628 - accuracy: 0.9808
Epoch 4/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0479 - accuracy: 0.9851
Epoch 5/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0405 - accuracy: 0.9872
Epoch 6/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0314 - accuracy: 0.9899
Epoch 7/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0266 - accuracy: 0.9908
Epoch 8/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0235 - accuracy: 0.9922
Epoch 9/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0192 - accuracy: 0.9934
Epoch 10/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0184 - accuracy: 0.9940
Learning rate : 0.1
```

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

```
Test loss: 0.05378325283527374
Test accuracy: 0.9857000112533569
```

```
Text(0.5, 1.0, 'Predicted Label: 2')
```




```
model.summary()
model.save("Assigment_3_20190183_20190593_model2.h5")
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 28)        280

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 28)       0
 2D)

 flatten_3 (Flatten)         (None, 4732)              0

 dense_6 (Dense)             (None, 128)               605824

 dropout_3 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 607,394
Trainable params: 607,394
Non-trainable params: 0
_________________________________________________________________
```

Best Model we got

```
model = Sequential()

model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation=tf.nn.relu))

model.add(Dropout(0.2))

model.add(Dense(10,activation=tf.nn.softmax))


model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a
ccuracy'])
model.fit(x=x_train,y=y_train, epochs=10 , batch_size=64)

K.set_value(model.optimizer.learning_rate, 0.1)
print("Learning rate :", model.optimizer.learning_rate.numpy())
```

- observed that model 2 with the training optimizer 'adam ', our test loss decreased significantly as we ran our model for 10 epochs and accuracy improved to over 98 % which is the best result we got overall .

It's the best model as it's the most accurate , least memory consuming and the fastest in execution

It only needs 10 epoch to reach this aacuracy , batch_size= 64 , learning rate = 0.1 , optimizer = 'adam'

- Building upon the strength of previous model, Adam optimizer gives much higher performance than the previously used and outperforms them by a big margin into giving an optimized gradient descent.

Adam Optimizer

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the ‘gradient descent with momentum’ algorithm and the ‘RMSP’ algorithm.

How Adam works?

Adam optimizer involves a combination of two gradient descent methodologies:

Momentum:

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.