Classify MNIST data with an MLP neural network. Allocate 20% of the data to the validation data.

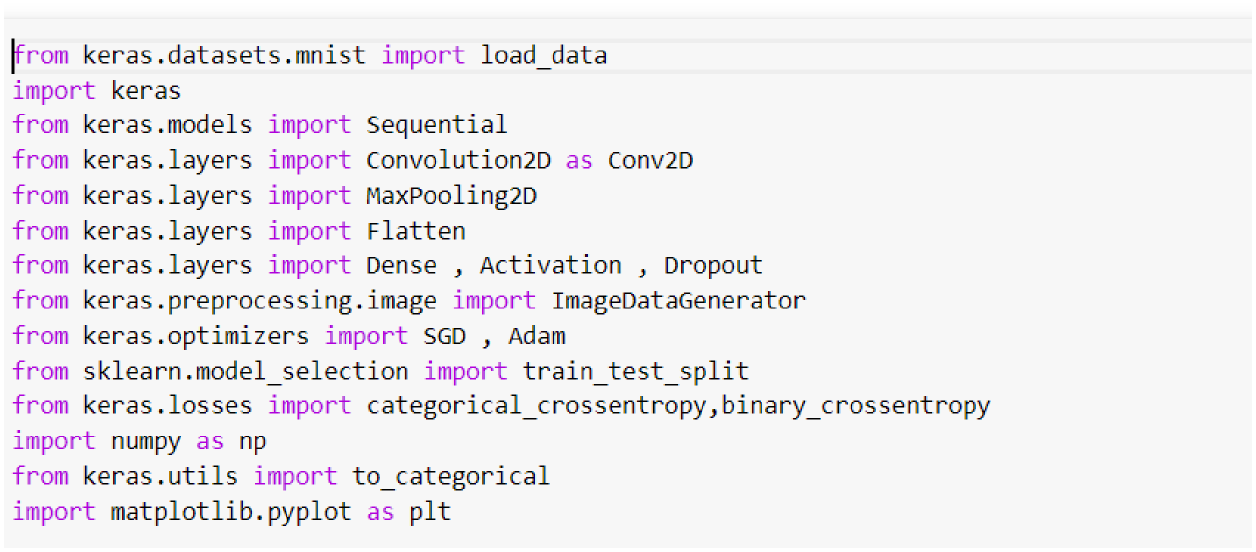
These data include 60,000 data for training and 10,000 data for testing in 100x100 pixel and black and white photos, all of which are handwritten figures.

We have used two cnn and nn networks in this project, which are not just convolutional layers in the neural network and are similar in the rest of the code.

All weights, charts and code are attached

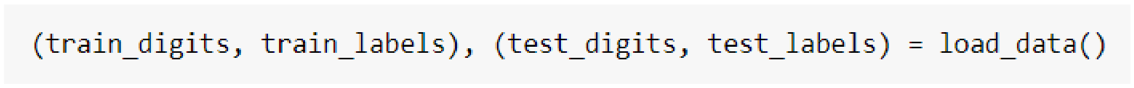
code :

First we import the required libraries



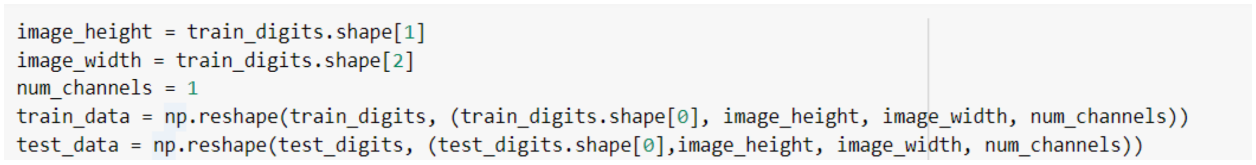
```
from keras.datasets.mnist import load_data
import keras
from keras.models import Sequential
from keras.layers import Convolution2D as Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense , Activation , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD , Adam
from sklearn.model_selection import train_test_split
from keras.losses import categorical_crossentropy,binary_crossentropy
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

Then we load the mnist test and training data



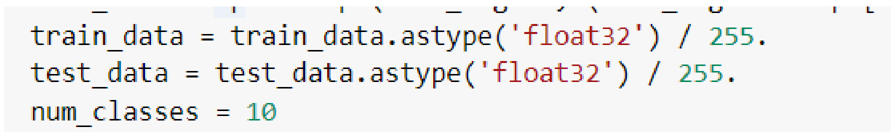
```
(train_digits, train_labels), (test_digits, test_labels) = load_data()
```

Now we prepare the data to give to the network



```
image_height = train_digits.shape[1]
image_width = train_digits.shape[2]
num_channels = 1
train_data = np.reshape(train_digits, (train_digits.shape[0], image_height, image_width, num_channels))
test_data = np.reshape(test_digits, (test_digits.shape[0],image_height, image_width, num_channels))
```

We give the size of the photo and also since the photos are black and white, it has only one channel



```
train_data = train_data.astype('float32') / 255.
test_data = test_data.astype('float32') / 255.
num_classes = 10
```

And we also normalize the number of pixels in each image to make calculations easier and also to prevent overflow of the network, and we also have ten digits from 0-9, which is ten classes.

```
train_labels_cat = to_categorical(train_labels,num_classes)
test_labels_cat = to_categorical(test_labels,num_classes)
```

Since the output of the network is a tens matrix of numbers between zero and one, which gives the membership rate in each class, we have to divide the training labels into a tens matrix, which is the number of class one and the rest of the matrix cells are zero.

```
model = Sequential()
```

The model is sequential, meaning that all the layers are placed one after the other

```
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same',input_shape=(image_height, image_width, num_channels)))
```

Now we add the convolutional layer, which includes 32 filters of the 3 in 3 matrix, and the activation of the function, after passing the convolution, we pass this function and do the same padding so that the image size (input matrix does not change). For the first convolutional layer, the input image size must be This layer brings features of the photo

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

We use the pulling layer to reduce the calculations by placing the largest amount of that block in the input matrix of two by two blocks.

```
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

Apply another layer of convolution and polishing

```
model.add(Flatten())
```

To enter the neural network, the input matrix of the neural network must be in the form of a vector, ie it must make the multidimensional matrix one-dimensional.

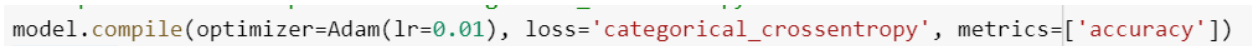
```
model.add(Dense(128, activation='relu'))
```

We add the first layer of the neural network with 128 neurons and activation function

And the second layer is 10 neurons, which is the number of classes

```
model.add(Dense(num_classes, activation='softmax'))
```

The activation of this layer is softmax, which makes the outputs between zero and one so that the sum of the neurons becomes one, and the output class is the number of the largest cell of the output vector.



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

We compile the model with optimizers that we use two types of adam and sgd

$$\Delta \overline{W}_j = -\eta \nabla \overline{E} = \eta (d_j - y_j) f'(\overline{x}\overline{w}_j)\overline{x}^T$$

In SGD, the change in weights is inversely high. We move in the opposite direction of the vector perpendicular to the error function. One of its problems is that it may fall to the local minimum and can no longer come out of it, so the grid can not weigh. Update your skills correctly to reach the desired value. We may be able to get out of this valley by increasing the learning factor.

Choosing the right learning rate can be difficult. If the learning rate is too small, it can cause the convergence to be very slow, and on the other hand, if it is too large, it can interfere with the convergence and cause the loss function to fluctuate or even diverge around the minimum value.

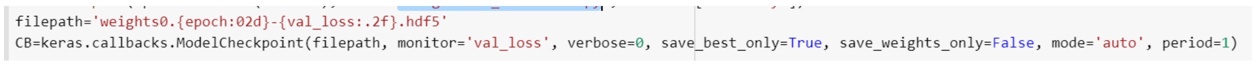
In SGD the learning rate is constant during learning and does not change. In adam it acts as if the learning rate was large at first to be able to see the entire error space. Then it reduces this rate according to the error to reach the minimum error .

Also for categorical_crossentropy is one of the loss functions that shows how different the network output is from the desired output. In mse this difference is equal to the power of 2 distances between these numbers. In this function this difference is shown by the fact that y ^ represents the output of the network and y represents the desired output, which is either zero or one.

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

And in the matrix we say print accuracy

Now we want to use the callback to find the best weights



```
filepath='weights0.{epoch:02d}-{val_loss:.2f}.hdf5'
CB=keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=True, save_weights_only=False, mode='auto', period=1)
```

In the filepath section, we define the name and method of storing weights

And we define a callback in such a way that reducing the loss of loss is important to us (monitor). This is to show that the change in the value of the monitor is a better increase or decrease that the model itself recognizes here.

Now we will teach the model

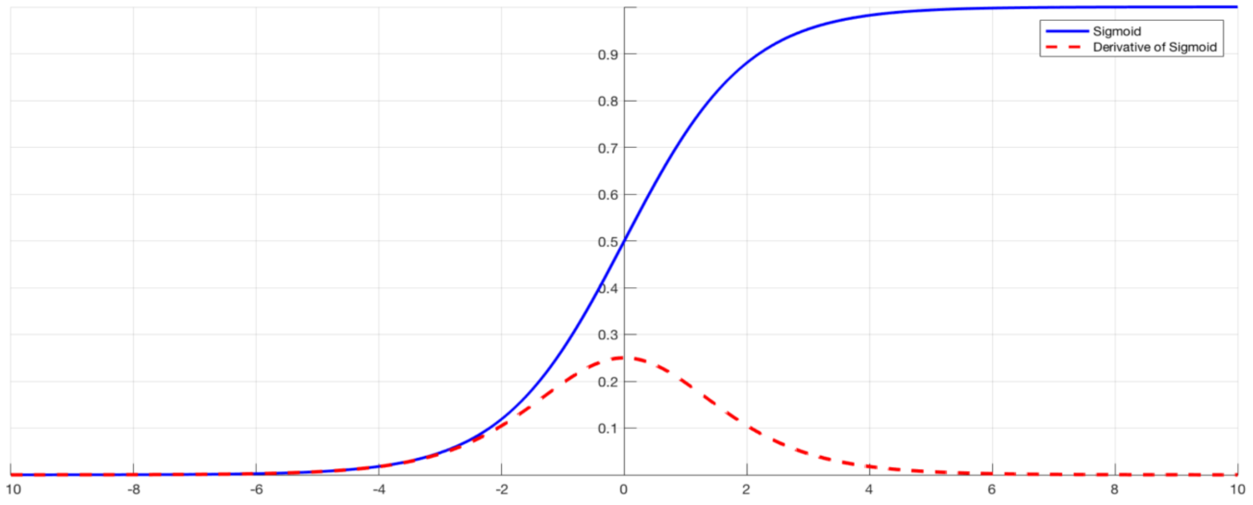
```
history = model.fit(train_data, train_labels_cat, epochs=100, batch_size=64,validation_split=0.2 , callbacks=[CB])
model.summary()
```

We took 20% of the training data for volition and defined it 100 times with 64 batch and using callback.

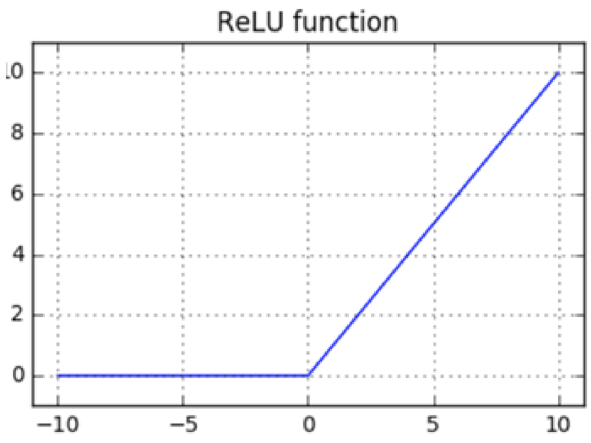
And we summarize the number of weights and biases of the system

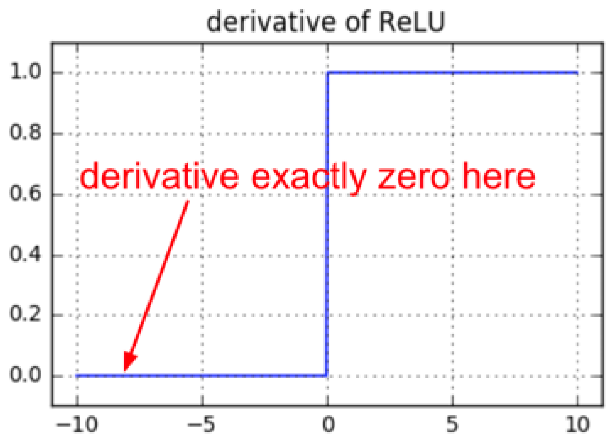
The difference between the sigmoid and relu functions:

Sigmoid is a function that converts input to a range of zero to one



To update the parameters, the back propagation method requires a function derivative, which is multiplied by a chain law, and if these functions and derivatives are too small, they approach zero and prevent the propagation of errors, and other parameters can not be updated. And we call this problem vanishing gradient, now we do not have this problem using the relu function



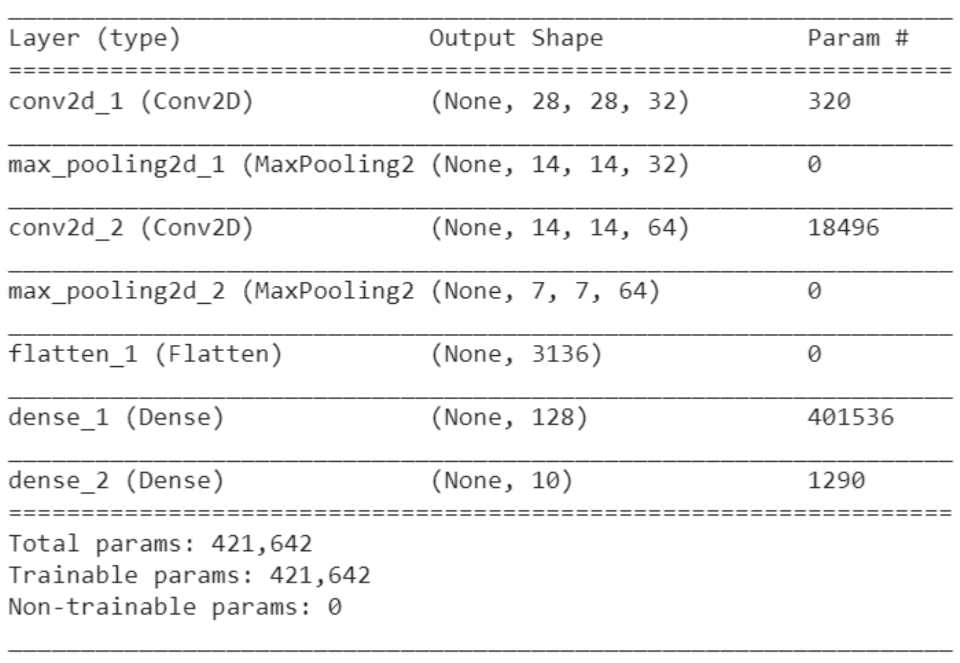


Difference in learning rate:

In the case of the sigmoid activation function, we have the same small derivative problem. We see that in the case where the learning rate is small and the sigmoid activation function is, the network can not be fitted to the data at all and the accuracy is very low. And it can not update the weights in such a way that it reaches the minimum global error and fluctuates, and we no longer have good accuracy, even though it is a reluctuation function, and we no longer have the problem of the derivative being small.

As we can see from the error and accuracy diagrams when we use SGD, the network takes longer for the network to be correct, and this is due to the constant learning rate in this algorithm. There is an algorithm that speeds up the learning process.

The higher the learning rate, the shorter the learning time and the weights increase or decrease with a larger coefficient, and therefore the system may become unstable and unable to get stuck at the minimum.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
_________________________________________________________________
```

Results are attached