Artificial intelligence for VR/AR

Report for Lab2 Session

Xuefeng Wei
Institut Polytechnique de Paris
xuefeng.wei@ip-paris.fr

Part I. Artificial Neural Networks (ANN)

This section focuses on the classification of Handwritten digit using ANN, and the dataset used is MNIST dataset. In the MNIST dataset, the size of each sample is 28x28x1, which is 784-pixel points in total, then 60,000 images are used for training and 10,000 images are used to test the trained model. First, we need to evaluate the effect of the number of neurons in the ANN on the performance of the model. The other hyperparameters are set as follows: batch_size=200, epochs=10, loss='categorical_crossentropy', optimizer= 'adam', metrics='accuracy'. Because of the stochastic nature of the algorithm, the authors ran each experiment five times and averaged the results to better evaluate the performance of the model. The effect of the number of neurons in the hidden layer on the model is shown in Table1 and Figure 1.

No of Neurons	8	16	32	64	128
System Accuracy (%)	92.79	95.18	96.58	97.39	97.79
Time Costing (s)	3.92	3.94	4.12	4.77	5.23

Table1. System performance evaluation for various numbers of neurons on the hidden layer

```
When number of neurons = 8,Batch_size= 200 Average errors in 5 times 7.21 Average testing accuracy in 5 times 92.79% The time costing is 3.92s

Process finished with exit code 0
```

```
When number of neurons = 16,Batch_size= 200 Average errors in 5 times 4.82 Average testing accuracy in 5 times 95.18% The time costing is 3.94s

Process finished with exit code 0
```

```
When number of neurons = 32,Batch_size= 200 Average errors in 5 times 3.42 Average testing accuracy in 5 times 96.58% The time costing is 4.12s

Process finished with exit code 0
```

```
When number of neurons = 64,Batch_size= 200 Average errors in 5 times 2.61 Average testing accuracy in 5 times 97.39% The time costing is 4.77s

Process finished with exit code 0
```

```
When number of neurons = 128,Batch_size= 200 Average errors in 5 times 2.21 Average testing accuracy in 5 times 97.79% The time costing is 5.23s

Process finished with exit code 0
```

Figure1. Results shown in PyCharm console for exercise 1

As can be seen from the results in Table 1, the effect of the model on the test set increase as the number of hidden layer neurons increases, while the running time increases, with no change in other hyperparameters. This indicates that more neurons in the hidden layer the better performance model can achieve, and more neurons means that the neural network needs to learn more parameters, so it takes longer to train.

In the second experiment, the effect of Batch size on the model was investigated by keeping the same hyperparameters as in Experiment 1 and fixing the number of neurons to 8. The values of Batch Size were 32, 64, 128, 256 and 512, and the results are shown in Table 2 and Figure 2.

Batch Size	32	64	128	256	512
System accuracy (%)	92.71	92.57	93.06	92.55	92.27
Time Costing (s)	19.31	10.37	5.85	3.49	2.17

Table2. System performance evaluation for different values of the batch size

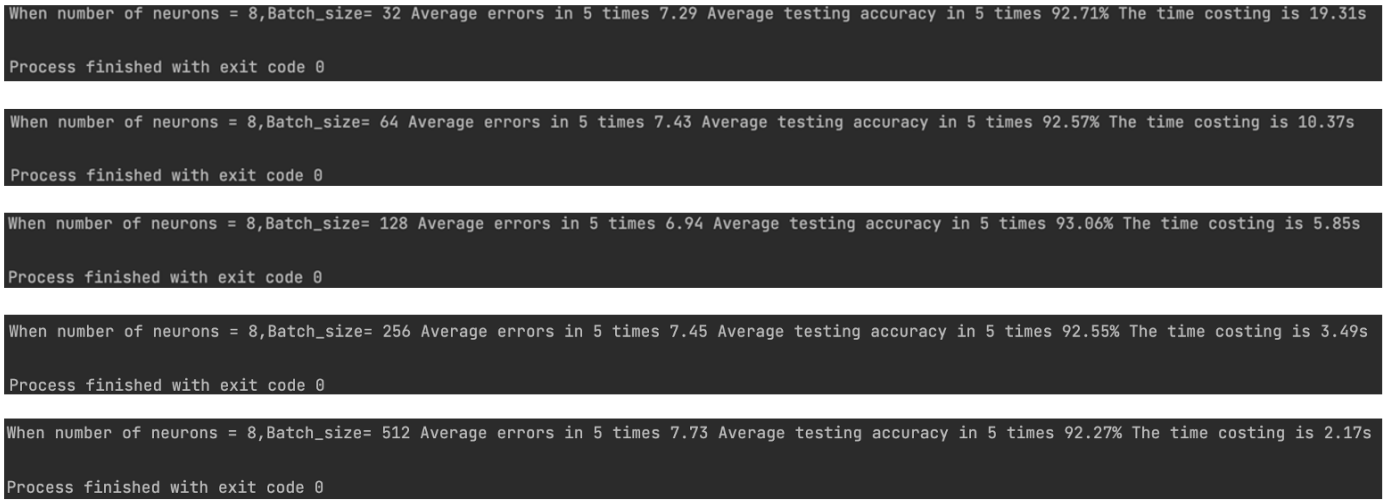

Figure2. Results shown in PyCharm console for exercise 2

From the results in Table 2, we know that the value of batch size also affects the testing accuracy of the model, and the model performs best when the batchwise is 128, but a larger batchwise implies a faster training time and vice versa.

The effect of different metrics on the performance of the model was similarly explored. In the third experiment, the authors set the metrics to Mean Square Error (MSE) and set the number of neurons at 8, 16, 32, 64 and 128, while keeping the other hyperparameters constant. Table 3 and Figure 3 show the results of the experiments.

No of Neurons	8	16	32	64	128
System accuracy (%)	92.43	94.69	96.86	97.46	97.75
Mean Square Error	0.01165	0.00817	0.00496	0.00396	0.00353
Time Costing (s)	4.03	4.02	4.21	4.61	5.27

Table3. System performance evaluation for different metric

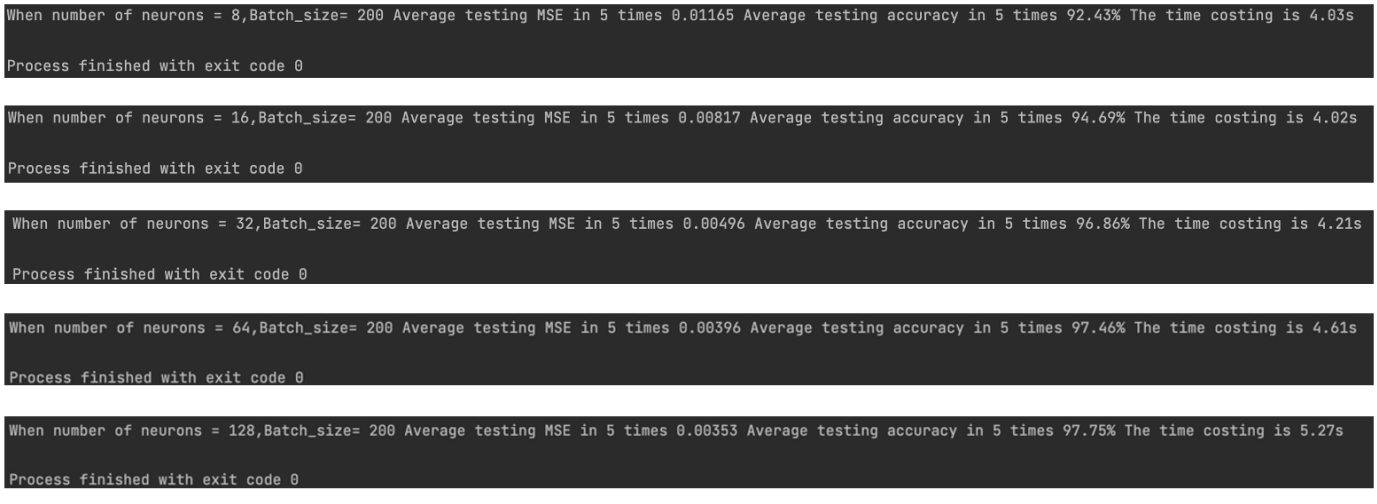

Figure3. Results shown in PyCharm console for exercise 3

From the results in Table 3, changing the metric does not affect the training of the system, as it yields accuracy values and trends consistent with Experiment 1.
Then the authors performed experiments to save the model weights. The (model.save_weights) function allows us to save the weights of the trained model, which allows us not to repeat the training in future tasks. After saving the weight file, the authors loaded the weight file in another python file and made predictions for the first 5 samples in the test, and the results are shown in Figure 4, with a test accuracy of 100%.

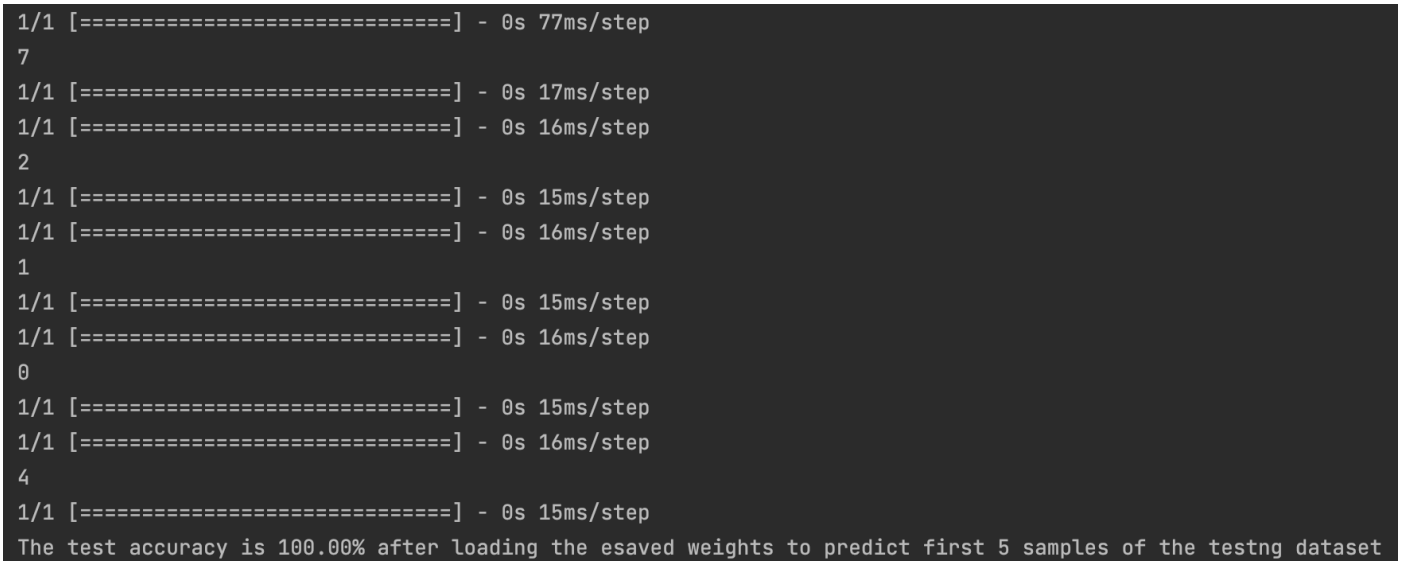

Figure4. Results shown in PyCharm console for exercise 5

Part II. Convolutional Neural Networks (CNN)

In the first part the authors used ANN to train and test MNIST, while in the second part the authors focused mainly on processing this dataset using CNN. The first experiment was to test its effect on the accuracy of the system by varying the size of the feature maps of the convolutional layers, and the results are shown in Table 4 and Figure 5.

Size of the feature map	1 x 1	3 x 3	5 x 5	7 x 7	9 x 9
System accuracy (%)	90.99	93.78	95.06	95.42	95.87
Time Costing (s)	15.02	20.15	27.36	38.13	42.88

Table4. System performance evaluation for various sizes of the convolutional filters

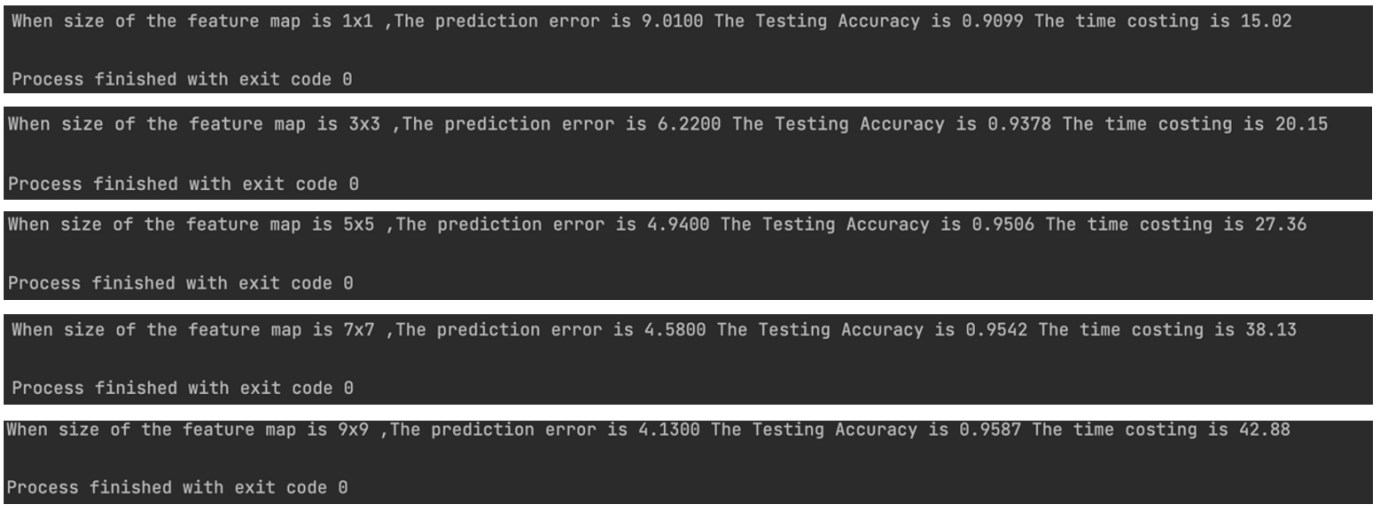

Figure5. Results shown in PyCharm console for exercise 6

Next, the authors discuss the effect of the number of neurons in the dense layer of the CNN on the model, and the results are shown in Table 5 and Figure 6.

No of neurons	16	64	128	256	512
System accuracy (%)	91.08	93.76	94.38	94.10	96.40
Time Costing (s)	18.11	17.67	19.23	20.41	24.96

Table5. System performance evaluation for various numbers of neurons on the dense hidden layer

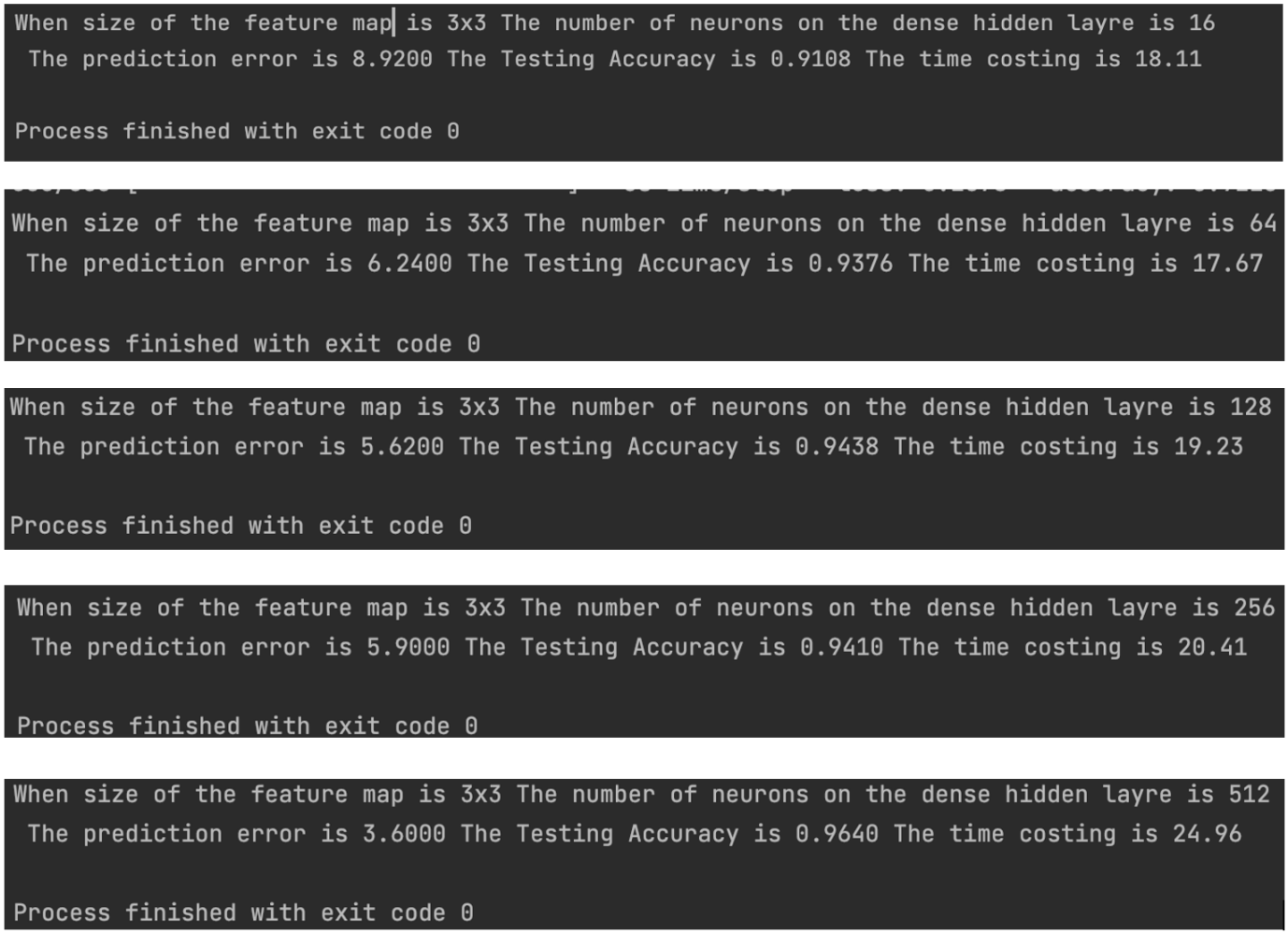

Figure6. Results shown in PyCharm console for exercise 7

From the results in Figure 5, as the number of neurons in the dense layer increases, the accuracy of the model on the test set improves, while the running time for training also increases.

The authors next explore the effect of changing the number of epochs on the model. Set the number of neurons in the dense layer to 128, and set the number of epochs to 1, 2, 5, 10 and 20. The results are shown in Table 6 and Figure 7.

No of epochs	1	2	5	10	20
System accuracy (%)	88.66	91.77	94.11	96.03	97.06
Time Costing (s)	4.56	7.74	18.09	37.58	72.95

Table6. System performance evaluation for different values of thee number of epochs

```
When size of the feature map is 3x3 The number of neurons on the dense hidden layre is 128 and the number of erpochs is 1
 The prediction error is 11.3400 The Testing Accuracy is 0.8866 The time costing is 4.56

Process finished with exit code 0
```

```
When size of the feature map is 3x3 The number of neurons on the dense hidden layre is 128 and the number of erpochs is 2
 The prediction error is 8.2300 The Testing Accuracy is 0.9177 The time costing is 7.74

Process finished with exit code 0
```

```
When size of the feature map is 3x3 The number of neurons on the dense hidden layre is 128 and the number of erpochs is 5
 The prediction error is 5.8900 The Testing Accuracy is 0.9411 The time costing is 18.09

Process finished with exit code 0
```

```
When size of the feature map is 3x3 The number of neurons on the dense hidden layre is 128 and the number of erpochs is 10
 The prediction error is 3.9700 The Testing Accuracy is 0.9603 The time costing is 37.58

Process finished with exit code 0
```

```
When size of the feature map is 3x3 The number of neurons on the dense hidden layre is 128 and the number of erpochs is 20
 The prediction error is 2.9400 The Testing Accuracy is 0.9706 The time costing is 72.95

Process finished with exit code 0
```

Figure7. Results shown in PyCharm console for exercise 8

From the results in Table 6, more epochs can give more time for the model to converge to achieve better results, but more epochs may make the model overfit and also require more training time.

Next, the authors change the CNN network to be more complex (adding more convolutional layers and pooling layers) by novel CNN structures in the hope of achieving better results. The structure of the model can be seen in Figure 8. The authors trained the network with number of epochs = 20 and batch size = 200, and the results can be seen in Figure 9.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 32)        832

 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 10, 10, 15)        4335

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 15)         0
 2D)

 dropout (Dropout)           (None, 5, 5, 15)          0

 flatten (Flatten)           (None, 375)               0

 dense (Dense)               (None, 128)               48128

 dense_1 (Dense)             (None, 50)                6450

 dense_2 (Dense)             (None, 10)                510

=================================================================
```

Figure8. The architecture of the novel CNN

```
For the CNN with new architecture  The prediction error is 1.2600 The Testing Accuracy is 0.9874 The time costing is 260.95

Process finished with exit code 0
```

Figure9. Results shown in PyCharm console for exercise 9

From the results in Figure 9, the new model achieves a test accuracy of 98.69%, which is better than any previous model, indicating that more complex and subtle models can achieve better performance.

Appendix

Code for Exercise 1

```
def baseline_model(num_pixels, num_classes):

    model = keras.models.Sequential()
    model.add(layers.Dense(8,      input_dim=num_pixels,kernel_initializer='normal',      activation='relu')).
    #Change the numbers of neurons here above
    model.add(layers.Dense(num_classes, kernel_initializer='normal', activation = 'softmax'))
    model.compile(loss='categorical_crossentropy' , optimizer='adam',
                  metrics = ['accuracy'])
    return model


def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
    num_pixels = X_train.shape[1] * X_train.shape[2]
    X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255

    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]

    errors = []
    score = []
    time_all = []
    for i in range(5):
        model = baseline_model(num_pixels, num_classes)
        time_start = time.time()
        model.fit(X_train,  Y_train,  validation_data=(X_test,  Y_test),  epochs=10,  batch_size=200,
        verbose=2)
        time_end = time.time()
        time_cost = time_end - time_start
        scores = model.evaluate(X_test, Y_test, verbose=0)
        err = 100 - scores[1] * 100
        score.append(scores[1])
        errors.append(err)
        time_all.append(time_cost)

    return
```

```
def main():
    (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
    trainAndPredictMLP(X_train, Y_train, X_test, Y_test)
    return

if __name__ == '__main__':
    main()
```

Comment: by model.add(layers.Dense(*,....)) The function changes the value of * to change the number of neurons in the hidden layer of ANN, set it to 8, 16, 32, 64 and 128 respectively and then test it, then the time function can record the running time, then save the corresponding accuracy results, and finally the average

---

Code for Exercise 2

```
def baseline_model(num_pixels, num_classes):

    model = keras.models.Sequential()
    model.add(layers.Dense(8, input_dim=num_pixels,kernel_initializer='normal', activation='relu'))
    model.add(layers.Dense(num_classes, kernel_initializer='normal', activation = 'softmax'))
    model.compile(loss='categorical_crossentropy' , optimizer='adam',
                  metrics = ['accuracy'])
    return model

def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
    num_pixels = X_train.shape[1] * X_train.shape[2]
    X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255

    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]

    errors = []
    score = []
    time_all = []
    for i in range(5):
        model = baseline_model(num_pixels, num_classes)
        time_start = time.time()
        model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=32, verbose=2)
        #change the batch_size here in 32, 64, 128, 256, 512
        time_end = time.time()
        time_cost = time_end - time_start
        scores = model.evaluate(X_test, Y_test, verbose=0)
        err = 100 - scores[1] * 100
```

```
        score.append(scores[1])
        errors.append(err)
        time_all.append(time_cost)

    return
```

Comment: The results are the same as those of the experimental generation, except that the number of neurons is fixed to 8 in the baseline_model, and the value of the batch_size hyperparameter in the trainAndPredictMLP function is changed.

Code for Exercise 3

```
def baseline_model(num_pixels, num_classes):

    model = keras.models.Sequential()
    model.add(layers.Dense(8, input_dim=num_pixels,kernel_initializer='normal', activation='relu')).
    model.add(layers.Dense(num_classes, kernel_initializer='normal', activation = 'softmax'))
    model.compile(loss='categorical_crossentropy' , optimizer='adam',
                  metrics = [‘accuracy’,'mse'])
    return model
```

Comment: The requirements of experiment 3 can be achieved by changing from metrics=[‘accuracy’] to metrics=['mse'] in the model.compile function.

Code for Exercise 4

```
def baseline_model(num_pixels, num_classes):

    model = keras.models.Sequential()
    model.add(layers.Dense(8, input_dim=num_pixels,kernel_initializer='normal', activation='relu'))
    model.add(layers.Dense(num_classes, kernel_initializer='normal', activation = 'softmax'))
    model.compile(loss='categorical_crossentropy' , optimizer='adam',
                  metrics = ['accuracy'])
    return model

def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
```

```
    num_pixels = X_train.shape[1] * X_train.shape[2]
    X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255

    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]

    errors = []
    score = []
    time_all = []
    for i in range(5):
        model = baseline_model(num_pixels, num_classes)
        time_start = time.time()
        model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=32, verbose=2)
        #change the batch_size here in 32, 64, 128, 256, 512
        time_end = time.time()
        time_cost = time_end - time_start
        scores = model.evaluate(X_test, Y_test, verbose=0)
        err = 100 - scores[1] * 100
        score.append(scores[1])
        errors.append(err)
        time_all.append(time_cost)
        model.save_weights('./lab2/weights_MLP.model')

    return
```

Comment: The trained weights file can be saved via model.save_weights

---

Code for Exercise 5

```
def baseline_model(num_pixels, num_classes):

        model = keras.models.Sequential()
        model.add(layers.Dense(8, input_dim=num_pixels,
                                    kernel_initializer='normal', activation='relu'))
        model.add(layers.Dense(num_classes, kernel_initializer='normal', activation = 'softmax'))
        model.compile(loss='categorical_crossentropy' , optimizer='adam',
                            metrics = ['accuracy'])
        return model
```

```
def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
        num_pixels = X_train.shape[1] * X_train.shape[2]
        X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
        X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')
        X_train = X_train / 255
        X_test = X_test / 255

        Y_train = np_utils.to_categorical(Y_train)
        Y_test = np_utils.to_categorical(Y_test)
        num_classes = Y_test.shape[1]

        model = baseline_model(num_pixels, num_classes)
        model.load_weights('weights_MLP.model')
        count= 0
        for idx,element in enumerate(X_test[0:5]):
            new = np.array(element).reshape(1,784)
            if np.argmax(model.predict(new)) == np.argmax(Y_test[idx]):
                count = count+1
        print("The test accuracy is {:.2f}% after loading the esaved weights to predict first 5 samples of the
testng dataset".format(count/5*100))
        return

def main():
        (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
        trainAndPredictMLP(X_train, Y_train, X_test, Y_test)

if __name__ == '__main__':
        main()
```

Comment: The model.load_weights function can load the weights file saved from exercise4, and then the model.predict function can be used to predict the first 5 samples of the test set.

Code for Exercise 6

```
def CNN_model(input_shape, num_classes):

    model = keras.models.Sequential()    #Modify this
    model.add(Conv2D(8, (9, 9), input_shape=(input_shape, input_shape, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
```

```
    model.compile(loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  metrics=['accuracy'])
    return model

def trainAndPredictCNN(X_train, Y_train, X_test, Y_test):

    X_train = X_train.reshape((X_train.shape[0], 28,28, 1)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0], 28,28, 1)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    model = CNN_model(28, num_classes)

    time_start = time.time()
    model.fit(X_train,Y_train,
              batch_size=200,
              epochs=5,
              validation_data=(X_test, Y_test))
    time_end = time.time()
    time_cost = time_end - time_start
    scores = model.evaluate(X_test, Y_test, verbose=0)
    err = 100 - scores[1] * 100

    print("When size of the feature map is 9x9",",The prediction error is {:.4f}".format(err),"The Testing
Accuracy is {:.4f}".format(scores[1]),"The time costing is {:.2f}".format(time_cost))


    return

def main():

    (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
    trainAndPredictCNN(X_train, Y_train, X_test, Y_test)


    return
```

Comment: Unlike ANN, CNN needs to use Conv2D, Maxpooliing and other layers to extract features, then at the end there is a sense layer similar to ANN, the blue code is the corresponding CNN implementation part.

Code for Exercise 7

```
def CNN_model(input_shape, num_classes):

    model = keras.models.Sequential()    #Modify this
    model.add(Conv2D(8, (3, 3), input_shape=(input_shape, input_shape, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(512, activation='relu')) //Change the number of neurons here from 16,64,128,256,512
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  metrics=['accuracy'])
    return model
```

Comment: The number of neurons can be changed by changing the value of the number of neurons at the blue code.

Code for Exercise 8

```
def trainAndPredictCNN(X_train, Y_train, X_test, Y_test):

    X_train = X_train.reshape((X_train.shape[0], 28,28, 1)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0], 28,28, 1)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    model = CNN_model(28, num_classes)

    time_start = time.time()
    model.fit(X_train,Y_train,
              batch_size=200,
              epochs=20, // change the number of here to determine the number of epochs
              validation_data=(X_test, Y_test))
    time_end = time.time()
    time_cost = time_end - time_start
    scores = model.evaluate(X_test, Y_test, verbose=0)
    err = 100 - scores[1] * 100
```

Comment: The number of epochs can be changed by changing the value of the epochs at the blue code.

==========

Code for Exercise 9

```
def CNN_model(input_shape, num_classes):

    model = keras.models.Sequential()    #Modify this
    model.add(Conv2D(32, (5, 5), input_shape=(input_shape, input_shape, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  metrics=['accuracy'])
    return model
def trainAndPredictCNN(X_train, Y_train, X_test, Y_test):

    X_train = X_train.reshape((X_train.shape[0], 28,28, 1)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0], 28,28, 1)).astype('float32')

    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    model = CNN_model(28, num_classes)
    model.summary()

    time_start = time.time()
    model.fit(X_train,Y_train,
              batch_size=200,
              epochs=20,
              validation_data=(X_test, Y_test))
    time_end = time.time()
    time_cost = time_end - time_start
    scores = model.evaluate(X_test, Y_test, verbose=0)
    err = 100 - scores[1] * 100
    return
```

Comment: The blue code shows the results of the new CNN, and better results can be achieved with this structure