Report of Clothing classification using CNN

Xuan ZHOU
Institut Polytechnique de Parise
xuan.zhou@telecom-sudparis.eu

Part I. Clothing classification using convolutional neural networks

This experiment is about studying the classification of clothing using convolutional neural networks (CNNs). The performance of the CNN is evaluated by building a CNN model and testing it. By improving the model and making predictions on new data. The system is evaluated by accuracy and loss.
This experiment was conducted by first loading Zalando's fashionable MNIST dataset, which consists of 60,000 training sets and 10,000 test sets. Each image is a 28x28 grayscale image and is associated with 10 labels. The results of the data visualization are shown in Figure 1.

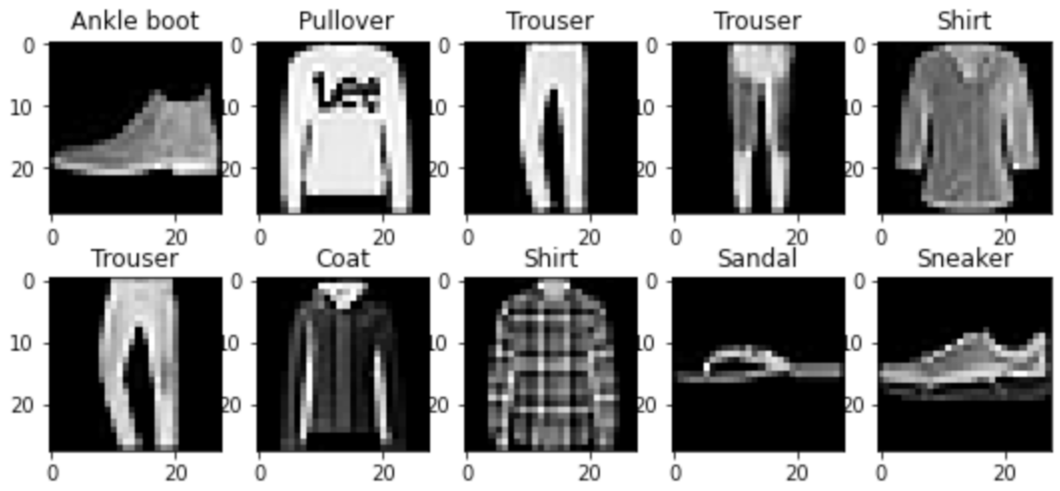
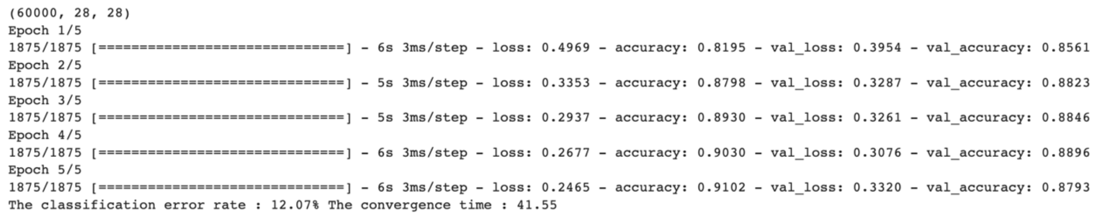


Figure 1. Visualize the shape of the loaded data

Next, the dataset and test set are loaded. and reshape the MNIST data, followed by normalizing the input values and transforming the category labels into binary matrices. Finally, the model is built and trained. the CNN model training results classification error rate is 12.07% and the convergence time is 41.55 s. The specific results are shown in Figure 2.



```
(60000, 28, 28)
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4969 - accuracy: 0.8195 - val_loss: 0.3954 - val_accuracy: 0.8561
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3353 - accuracy: 0.8798 - val_loss: 0.3287 - val_accuracy: 0.8823
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2937 - accuracy: 0.8930 - val_loss: 0.3261 - val_accuracy: 0.8846
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2677 - accuracy: 0.9030 - val_loss: 0.3076 - val_accuracy: 0.8896
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2465 - accuracy: 0.9102 - val_loss: 0.3320 - val_accuracy: 0.8793
The classification error rate : 12.07% The convergence time : 41.55
```

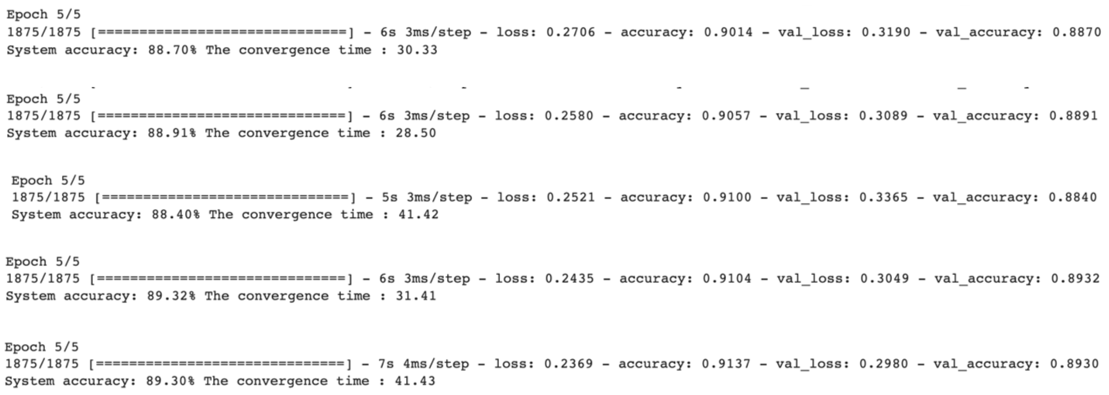
Figure 2. The result of classification error and the convergence time

Next, Next, the effect of the number of filters on the accuracy and convergence time of the system was observed by modifying the number of filters in the convolutional layer. The number of filters is 8, 16, 32, 64 and 128, and the results are shown in Table 1.

Number of filters	8	16	32	64	128
System accuracy (%)	88.70%	88.91%	88.40%	89.23%	89.30%
Convergence time(s)	30.33 s	28.50 s	41.42 s	31.41 s	41.43 s

Table 1. Results of modifying the number of filters in the convolutional layer

Figure 3 shows the results of the effect of a different number of filters on the accuracy and convergence time of the system after modifying the number of filters in the convolution layer.



```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2706 - accuracy: 0.9014 - val_loss: 0.3190 - val_accuracy: 0.8870
System accuracy: 88.70% The convergence time : 30.33

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2580 - accuracy: 0.9057 - val_loss: 0.3089 - val_accuracy: 0.8891
System accuracy: 88.91% The convergence time : 28.50

Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2521 - accuracy: 0.9100 - val_loss: 0.3365 - val_accuracy: 0.8840
System accuracy: 88.40% The convergence time : 41.42

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2435 - accuracy: 0.9104 - val_loss: 0.3049 - val_accuracy: 0.8932
System accuracy: 89.32% The convergence time : 31.41

Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2369 - accuracy: 0.9137 - val_loss: 0.2980 - val_accuracy: 0.8930
System accuracy: 89.30% The convergence time : 41.43
```

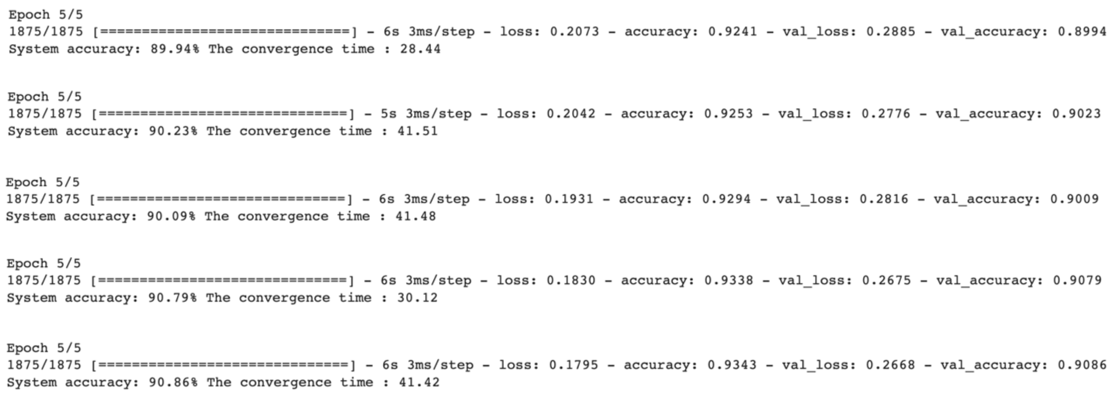
Figure 3. Results of modifying the number of filters in the convolutional layer

By modifying the number of filters in the convolutional layer, I found that the accuracy of the system increases with the number of filters in the convolutional layer, reaching a maximum accuracy of 89.30% at a filter number of 128. The convergence time also increases with the number of filters, with a maximum of 41.43 s.

Next, the change in accuracy and convergence time of the system is observed by modifying the number of neurons in the hidden layer of the density set. In this experiment, the number of filters in the convolutional layer is 32. The number of neurons in the modified density set hidden layer is 16, 64, 128, 256 and 512. the results are shown in Table 2 and Figure 4.

No of neurons	16	64	128	256	512
System accuracy (%)	89.94%	90.09%	90.52%	90.78%	90.86%
Convergence time(s)	28.44 s	41.48 s	41.42 s	30.12 s	41.42 s

Table 2. Results of modifying the number of neurons in the hidden layer of the density set



```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2073 - accuracy: 0.9241 - val_loss: 0.2885 - val_accuracy: 0.8994
System accuracy: 89.94% The convergence time : 28.44

Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2042 - accuracy: 0.9253 - val_loss: 0.2776 - val_accuracy: 0.9023
System accuracy: 90.23% The convergence time : 41.51

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1931 - accuracy: 0.9294 - val_loss: 0.2816 - val_accuracy: 0.9009
System accuracy: 90.09% The convergence time : 41.48

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1830 - accuracy: 0.9338 - val_loss: 0.2675 - val_accuracy: 0.9079
System accuracy: 90.79% The convergence time : 30.12

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1795 - accuracy: 0.9343 - val_loss: 0.2668 - val_accuracy: 0.9086
System accuracy: 90.86% The convergence time : 41.42
```

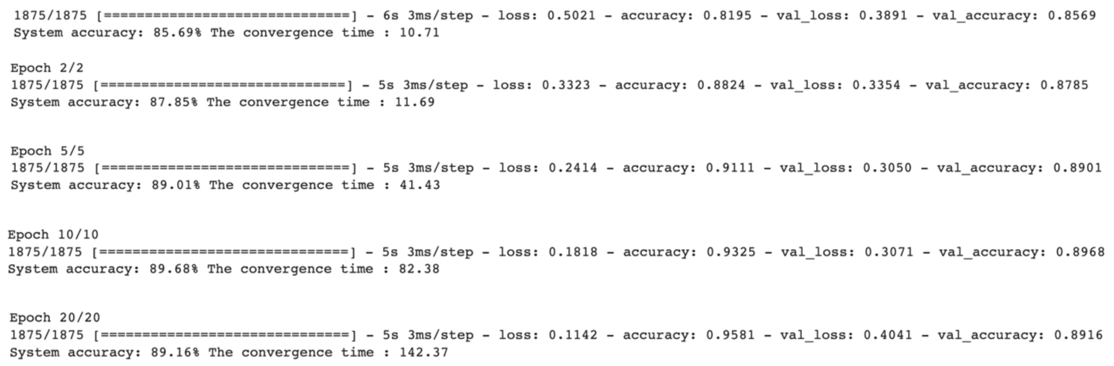
Figure 4. Results of modifying the number of neurons in the hidden layer of the density set

By modifying the number of neurons in the density set hidden layer, I found that the system accuracy increases as the number of neurons in the density set hidden layer increases, with a maximum of 90.86%. The convergence time also fluctuates with the increase of the number of neurons in the density set hidden layer, but it also shows an increasing trend, with the highest being 41.48 s.

Next, the changes in system accuracy and convergence time were observed by modifying the number of epochs. In this experiment, the number of neurons in the dense hidden layer is 16. The number of epochs modified in this experiment is 1, 2, 5, 10 and 20. the results are shown in Table 2 and Figure 5

No of neurons	1	2	5	10	20
System accuracy (%)	85.69%	87.85%	89.01%	89.68%	89.16%
Convergence time(s)	10.71 s	11.69 s	41.43 s	82.38 s	142.37 s

Table 2. The result of modifying the number of epochs



```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5021 - accuracy: 0.8195 - val_loss: 0.3891 - val_accuracy: 0.8569
System accuracy: 85.69% The convergence time : 10.71

Epoch 2/2
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3323 - accuracy: 0.8824 - val_loss: 0.3354 - val_accuracy: 0.8785
System accuracy: 87.85% The convergence time : 11.69

Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2414 - accuracy: 0.9111 - val_loss: 0.3050 - val_accuracy: 0.8901
System accuracy: 89.01% The convergence time : 41.43

Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1818 - accuracy: 0.9325 - val_loss: 0.3071 - val_accuracy: 0.8968
System accuracy: 89.68% The convergence time : 82.38

Epoch 20/20
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1142 - accuracy: 0.9581 - val_loss: 0.4041 - val_accuracy: 0.8916
System accuracy: 89.16% The convergence time : 142.37
```

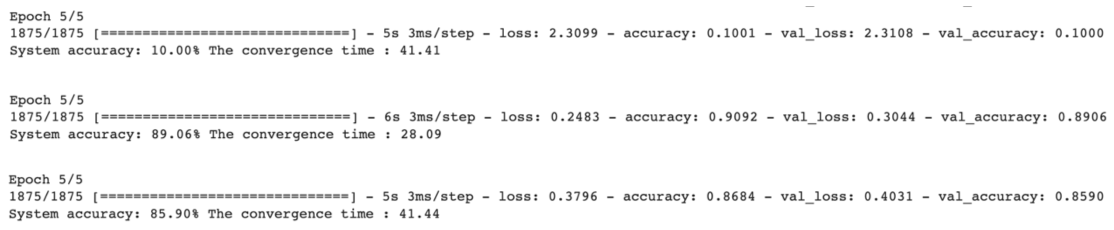
Figure 5. The result of modifying the number of epochs

By modifying the number of epochs, I found that the accuracy of the system increases as the number of epochs increases, up to a maximum of 89.68% in epoch 10, and the convergence time also increases gradually as the number of epochs increases, up to a maximum of 142.37 s in epoch 20. This shows that epochs also have a significant impact on the model, and in the process of training the model, we need to attention should be paid to the problem of overfitting caused by epochs.

The results of system accuracy and convergence time are observed by modifying the learning rate of stochastic gradient descent. For this experiment, the epoch is 5. The number of learning rates are 0.1, 0.01, 0.001, 0.0001 and 0.00001. The results are shown in Table 3 and Figure 6.

Learning rate	0.1	0.01	0.001	0.0001	0.00001
System accuracy (%)	10.00%	89.06%	85.90%	80.34%	56.85%
Convergence time(s)	41.41 s	28.09 s	41.44 s	28.16 s	41.44 s

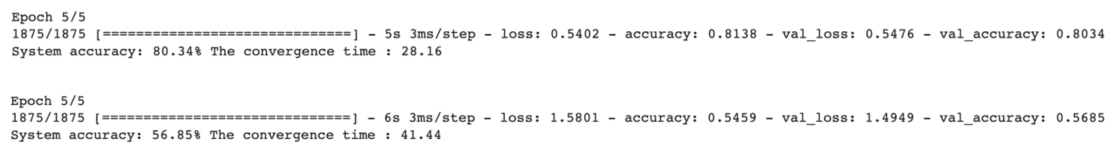
Table 3. Results of modifying the learning rate of the stochastic gradient descent method



```
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3099 - accuracy: 0.1001 - val_loss: 2.3108 - val_accuracy: 0.1000
System accuracy: 10.00% The convergence time : 41.41

Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2483 - accuracy: 0.9092 - val_loss: 0.3044 - val_accuracy: 0.8906
System accuracy: 89.06% The convergence time : 28.09

Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3796 - accuracy: 0.8684 - val_loss: 0.4031 - val_accuracy: 0.8590
System accuracy: 85.90% The convergence time : 41.44
```



```
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.5402 - accuracy: 0.8138 - val_loss: 0.5476 - val_accuracy: 0.8034
System accuracy: 80.34% The convergence time : 28.16
```

```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 1.5801 - accuracy: 0.5459 - val_loss: 1.4949 - val_accuracy: 0.5685
System accuracy: 56.85% The convergence time : 41.44
```

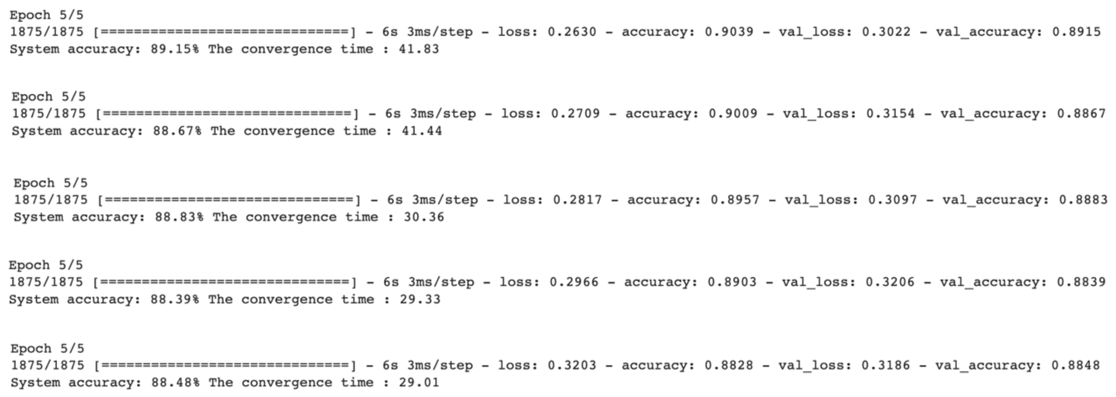
Figure 6. Results of modifying the learning rate of the stochastic gradient descent method

By modifying the learning rate of the stochastic gradient descent method, I found that the accuracy of the system fluctuates more as the learning rate decreases, showing a trend of increasing and then decreasing. The convergence time also fluctuates. The decrease in the learning rate of the stochastic gradient descent method leads to underfitting. The model does not accurately capture the relationship between the input and output variables. We can improve this by increasing the training time or inputting more features or less regularization.

Next, I will explore the effect on model performance by adding regularization. A Dropout layer was added to the CNN model where 20% of the neurons were randomly expelled and the learning efficiency was chosen to be 0.01. Based on this setup, the accuracy and convergence time of the system was observed. the Dropout Percentage was 0.1, 0.2, 0.3, 0.4 and 0.5. the results are shown in Table 4 and Figure 7.

Dropout Percentage	0.1	0.2	0.3	0.4	0.5
System accuracy (%)	89.15%	88.67%	88.83%	88.39%	88.48%
Convergence time(s)	41.83 s	41.44 s	30.36 s	29.33 s	29.01 s

Table 4. Results of the effect of adding regularization on model performance



```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2630 - accuracy: 0.9039 - val_loss: 0.3022 - val_accuracy: 0.8915
System accuracy: 89.15% The convergence time : 41.83
```

```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2709 - accuracy: 0.9009 - val_loss: 0.3154 - val_accuracy: 0.8867
System accuracy: 88.67% The convergence time : 41.44
```

```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2817 - accuracy: 0.8957 - val_loss: 0.3097 - val_accuracy: 0.8883
System accuracy: 88.83% The convergence time : 30.36
```

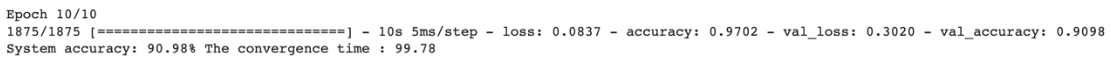
```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2966 - accuracy: 0.8903 - val_loss: 0.3206 - val_accuracy: 0.8839
System accuracy: 88.39% The convergence time : 29.33
```

```
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3203 - accuracy: 0.8828 - val_loss: 0.3186 - val_accuracy: 0.8848
System accuracy: 88.48% The convergence time : 29.01
```

Figure 7. Results of the effect of adding regularization on model performance

By adding regularization, we can find that the accuracy of the system changes relatively little, but the convergence time changes more significantly. As modify the percentage of excluded neurons in the dropout layer , the convergence time also decreases, and at a dropout percentage of 0.5, the convergence time is 29.01 s. This can also prevent overfitting.

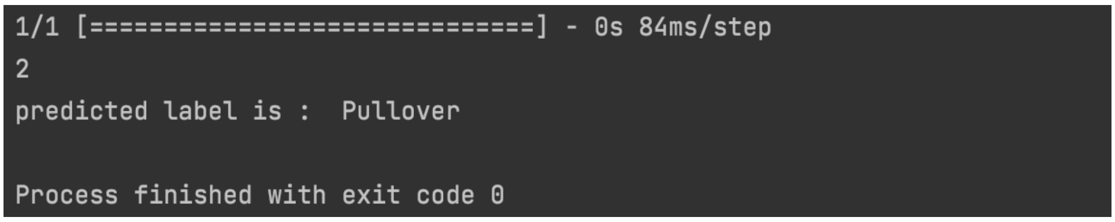
Next, the optimal values of the following parameters, such as the number of filters in the convolutional layer, size of the convolutional kernel, the number of neurons in the dense hidden layer, the number of epochs used for training, and learning rate, are specified in the CNN model to maximize the system accuracy. The results are shown in Figure 8.



```
Epoch 10/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0837 - accuracy: 0.9702 - val_loss: 0.3020 - val_accuracy: 0.9098
System accuracy: 90.98% The convergence time : 99.78
```

Figure 8. The result of specifying the optimal value of the parameter

After maximizing the accuracy of the system based on the specified optimal parameters, we save the weight file of the model and use our saved model to make predictions for the new image. Next, we load the weight file, and the predicted new image is a gray figure containing a centered dress on a black background, with a square size of 28x28. The predicted image results are shown in Figure 9.



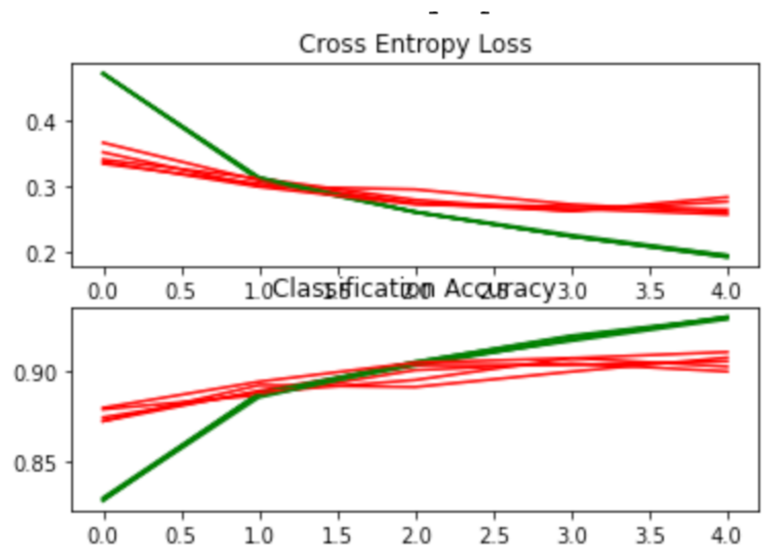
```
1/1 [==============================] - 0s 84ms/step
2
predicted label is :  Pullover

Process finished with exit code 0
```

Figure 9. New image prediction results

Part II. K-fold cross-validation algorithm

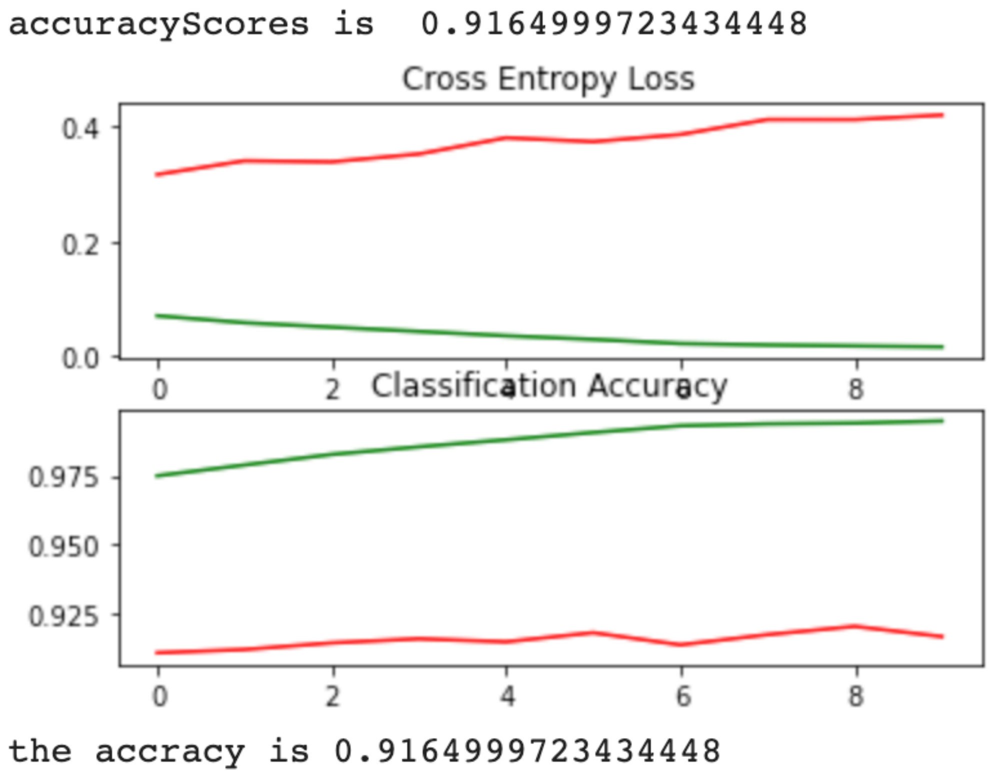
The second part, which is based on the first part, divides the dataset into a training set, a test set and a validation set. The K-fold cross-validation algorithm is a very good choice when training multiple models. This means that in the training set, we need to split the training dataset into two parts, one for the actual training of the model and the other for tuning the hyperparameters of the model. we can create an instance that splits a dataset into 5 folds and shuffles prior to the split for 5 folds, the validation dataset is 20% of the training dataset. The model is built and the results after running the CNN model are shown in Figure 10.



```
Accuracy: mean=89.890 std=0.406, n=5
```

Figure 10. Results of the system's performance

Adding padding to the convolution operation usually leads to better model performance, as more feature maps of the input image have the opportunity to participate or contribute to the output. The result after adding padding to the convolution operation is shown in Figure 11.



```
accuracyScores is  0.9164999723434448
```

```
the accracy is 0.9164999723434448
```

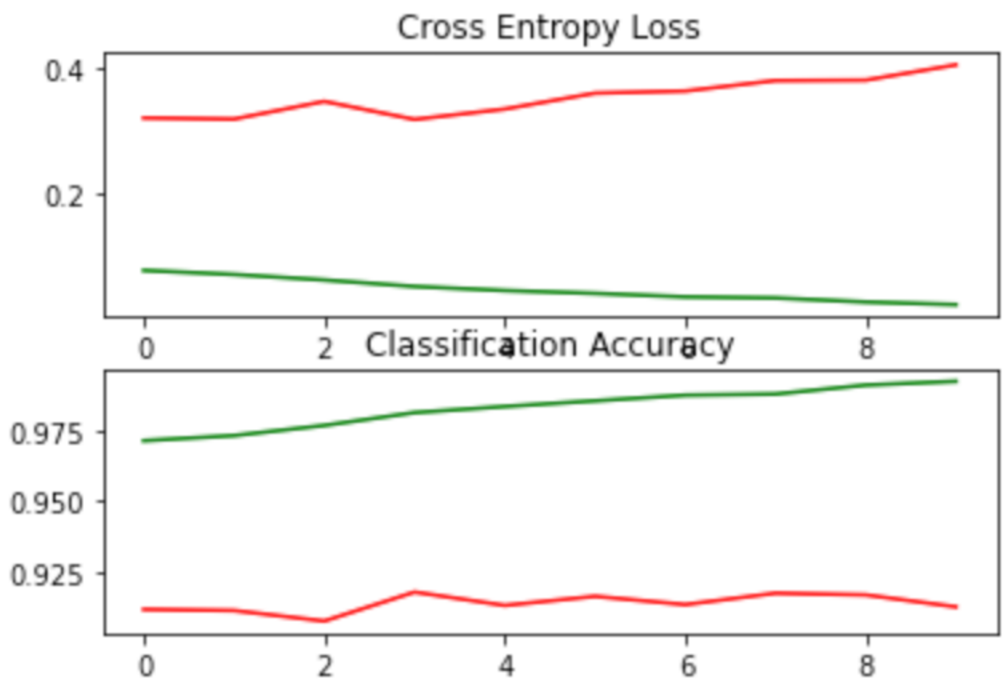
Figure 11. The result of adding padding

An increase in the number of filters used in the convolutional layer can often improve performance, as it can provide more opportunities for extracting simple features from the input images. Now we increase the number of filters from 32 to 64 when using very small filters, such as 3 × 3 pixels, by applying a fill operation during the convolution process. For application 1, the results of observing the effect of this parameter on the accuracy of the system are shown in Table 5 and Figure 12.

No of filters	32	64
Accuracy (%)	0.9129%	0.9171%

Table 5. Results of increasing the number of filters

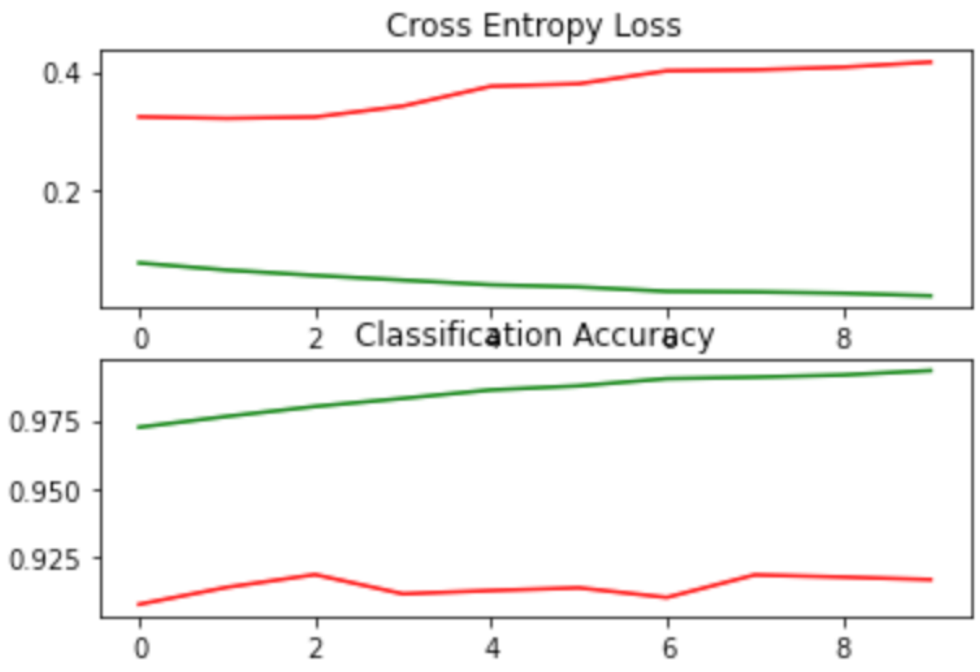
```
accuracyScores is  0.9128999710083008
```



```
the accracy is 0.9128999710083008
```

Figure 5. Results of increasing the number of filters

```
accuracyScores is  0.9171000123023987
```

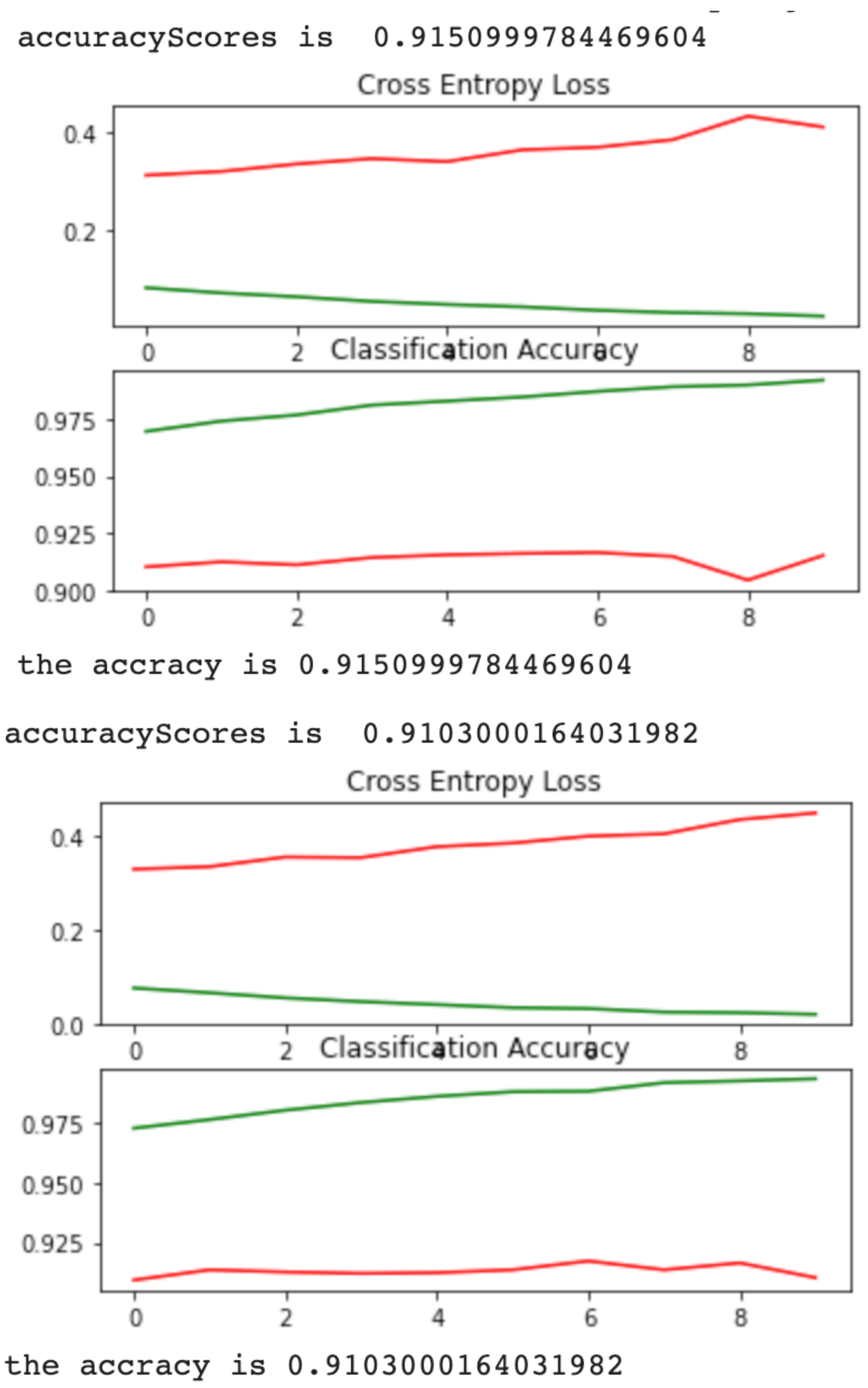


```
the accracy is 0.9171000123023987
```

The next case is without padding, where the number of filters is increased from 32 to 64. For application 1, the effect of this parameter on the accuracy of the system is observed and the results are shown in Table 6 and Figure 13.

No of filters	32	64
Accuracy (%)	0.9151%	0.9130%

Table 6. The result without padding



```
accuracyScores is  0.9150999784469604
```

```
the accracy is 0.9150999784469604
```

```
accuracyScores is  0.9103000164031982
```

```
the accracy is 0.9103000164031982
```

Figure 6. The result without padding

Comparing the results of using the number of filters in the convolutional layer as 32 and 64 with and without padding is not much different, but the accuracy with padding added is higher than without padding.

Appendix

Code for Application 1

```
import numpy as np
from sklearn.model_selection import KFold
import keras
from keras.optimizers import gradient_descent_v2
from keras.datasets import fashion_mnist
from keras.utils import np_utils
from keras import layers
from matplotlib import pyplot
import cv2
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout
import time
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
dict_classes = {0: "T-shirt/top", 1: "Trouser", 2: " Pullover", 3: "Dress", 4: "Coat", 5:
"Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle boot"}

def summarizeLearningCurvesPerformances(histories, accuracyScores):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(211)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], color='green', label='train')
        pyplot.plot(histories[i].history['val_loss'], color='red', label='test')
        # plot accuracy
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], color='green', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], color='red', label='test')
        #print accuracy for each split
        print("Accuracy for set {} = {}".format(i, accuracyScores[i]))
    pyplot.show()
    print('Accuracy:    mean    =    {:.3f}    std    =    {:.3f},    n    =
{}'.format(np.mean(accuracyScores)  *  100,  np.std(accuracyScores)  *  100,
len(accuracyScores)))

def prepareData(trainX, trainY, testX, testY):
    trainX = trainX.reshape(60000,28,28,1)
```

```
	testX = testX.reshape(10000,28,28,1)
	trainX = trainX / 255
	testX = testX / 255
	trainY = np_utils.to_categorical(trainY)
	testY = np_utils.to_categorical(testY)
	num_classes = testY.shape[1]
	return trainX, trainY, testX, testY

def defineModel(input_shape, num_classes):
	model = keras.models.Sequential()
	model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
	model.add(MaxPooling2D(pool_size=(2, 2)))
	model.add(Flatten())
	model.add(Dense(16,activation = 'relu'))
	kernel_initializer = 'he_uniform'
	model.add(Dense(10, activation='softmax'))
	model.compile(loss='categorical_crossentropy'                                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
				metrics = ['accuracy'])
	return model

def defineTrainAndEvaluateClassic(trainX, trainY, testX, testY):
	model=defineModel(trainX, 10)
	test_array = []
	time_1 = []
	time_begin = time.time()
	model.fit(trainX,trainY,   batch_size=128,   epochs=50,   validation_data=(testX,
testY))
	time_end = time.time()
	scores = model.evaluate(testX, testY, verbose=0)
	test_array.append(scores[1])
	time_1 = time_end - time_begin
	print("The classification error rate : {:.2f}%".format(100 - scores[1] * 100),"The
convergence time : {:.2f}".format(time_1))
	return

def main():
	#TODO - Application 1 - Step 2 - Load the Fashion MNIST dataset in Keras
	(trainX,   trainY), (testX, testY) = fashion_mnist.load_data()
	# Figure 1
	cols = 5
	rows = 2
	fig = plt.figure(figsize=(2 * cols - 1, 2.5 * rows - 1))
	idx = 0
```

```
    for i in range(cols):
        for j in range(rows):
            ax = fig.add_subplot(rows, cols, i * rows + j + 1)
            ax.imshow(testX[idx],"gray")
            label = testY[idx]
            ax.set_title(dict_classes[label])
            idx = idx + 1
    plt.show()
    print(trainX.shape)
    trainX, trainY, testX, testY = prepareData(trainX, trainY, testX, testY)
    defineTrainAndEvaluateClassic(trainX, trainY, testX, testY)


    return


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

Code for Exercise 2

```
def defineModel(input_shape, num_classes):
    model = keras.models.Sequential()
    # Exercise 2
    model.add(Conv2D(128, (3, 3), input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(16,activation = 'relu'))
    kernel_initializer = 'he_uniform'
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy'                          ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                    metrics = ['accuracy'])
    return model
```

Comment: Modify the number of filters in the convolutional layer and observe the effect on the accuracy of the system.

Code for Exercise 3

```
def defineModel(input_shape, num_classes):
    model = keras.models.Sequential()
```

```
        model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        # Exercise 3
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        model.compile(loss='categorical_crossentropy'                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                                    metrics = ['accuracy'])
        return model
```

Comment: The number of neurons in the dense hidden layer is modified to observe the effect on the accuracy of the system.

Code for Exercise 4

```
def defineTrainAndEvaluateClassic(trainX, trainY, testX, testY):
        model=defineModel(trainX, 10)
        test_array = []
        time_1 = []
        time_begin = time.time()
        # Exercise 4
        model.fit(trainX,trainY, batch_size=32, epochs=5, validation_data=(testX, testY))
        time_end = time.time()
        scores = model.evaluate(testX, testY, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        print("The classification error rate : {:.2f}%".format(100 - scores[1] * 100),"The
convergence time : {:.2f}".format(time_1))
        return
```

Comment: Modify the number of epochs and observe the impact on system accuracy.

Code for Exercise 5

```
def defineModel(input_shape, num_classes):
        model = keras.models.Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        # Exercise 5
        model.compile(loss='categorical_crossentropy'                                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.00001,momentum = 0.9),
                                    metrics = ['accuracy'])
        return model
```

Comment: Modify the size of the learning rate to see the effect on the accuracy of the system.

Code for Exercise 6

```
def defineModel(input_shape, num_classes):
        model = keras.models.Sequential()
        model.add(Conv2D(128, (3, 3), input_shape=(28, 28, 1)))
       model.add(MaxPooling2D(pool_size=(2, 2)))
       # Exercise 6
       model.add(Dropout(0.1))
        model.add(Flatten())
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        model.compile(loss='categorical_crossentropy'                                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                                    metrics = ['accuracy'])
        return model
```

Comment: Add regularization and observe the effect on the accuracy of the system.

Code for Exercise 7

```
def defineModel(input_shape, num_classes):
        model = keras.models.Sequential()
        # Exercise 7
        model.add(Conv2D(128, (3, 3), input_shape=(28, 28, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.1))
        model.add(Flatten())
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        model.compile(loss='categorical_crossentropy'                                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                                metrics = ['accuracy'])
        return model

def defineTrainAndEvaluateClassic(trainX, trainY, testX, testY):
        model=defineModel(trainX, 10)
        test_array = []
        time_1 = []
        time_begin = time.time()
        model.fit(trainX,trainY,   batch_size=32,   epochs=10,   validation_data=(testX,
testY))
        time_end = time.time()
        scores = model.evaluate(testX, testY, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        print("The classification error rate : {:.2f}%".format(100 - scores[1] * 100),"The
convergence time : {:.2f}".format(time_1))
        return
```

Comment: Specify the optimal value of the parameter and observe the effect on the system accuracy.

Code for Exercise 8

```
#   Exercise 8
def load_image(path):
    image = cv2.imread(path, 0)
    resized_image = cv2.resize(image, (28, 28))
    return resized_image

def defineTrainAndEvaluateClassic(trainX, trainY, testX, testY):
    # TODO - Application 1 - Step 6 - Call the defineModel function
    model = defineModel(trainX, 10)
    model = load_model('Fashion_MNIST_model.h5')
    image = load_image('sample_image.png')
    tensor_image = tf.convert_to_tensor(image)
    tensor_image = tf.expand_dims(tensor_image, 0)
    tensor_image = tf.expand_dims(tensor_image, -1)
    #print(tensor_image.shape)
    label = np.argmax(model.predict(tensor_image))
    print(label)
    print("predicted label is : {}".format(dict_classes[label]))
    return
```

Comment: Save the weights file and predict the result of the new image after reloading.

Code for Application 2 - - -Exercise 8

```
def defineModel(input_shape, num_classes):
        model = keras.models.Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        model.compile(loss='categorical_crossentropy'                                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                                    metrics = ['accuracy'])
        return model
def defineTrainAndEvaluateClassic(trainX, trainY, testX, testY):
        model=defineModel(trainX, 10)
        accuracyScores = []
        histories = []
        test_array = []
        time_1 = []
        time_begin = time.time()
        model.fit(trainX,trainY,    batch_size=32,    epochs=10,    validation_data=(testX,
testY))
        time_end = time.time()
        scores = model.evaluate(testX, testY, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        history        =        model.fit(trainX,trainY,        batch_size=32,        epochs=10,
validation_data=(testX, testY))
        histories.append(history)
        scores = model.evaluate(testX,testY)
        accuracyScores.append(scores[1])
        histories, accuracyScores
        return histories, accuracyScores

def summarizeLearningCurvesPerformances(histories, accuracyScores):
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(histories[i].history['loss'], color='green', label='train')
    pyplot.plot(histories[i].history['val_loss'], color='red',label='validation')
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(histories[i].history['accuracy'], color='green',label='train')
    pyplot.plot(histories[i].history['val_accuracy'], color='red',
    label='validation')
    pyplot.show()
    print("the accracy is {}".format(accuracyScores))
```

Comment: Add padding to the convolution layer and observe the effect on the accuracy of the system.

Code for Application 2 - - -Exercise 9

```
def defineModel(input_shape, num_classes):
        model = keras.models.Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(512,activation = 'relu'))
        kernel_initializer = 'he_uniform'
        model.add(Dense(10, activation='softmax'))
        model.compile(loss='categorical_crossentropy'                    ,
optimizer=gradient_descent_v2.SGD(learning_rate=0.01,momentum = 0.9),
                              metrics = ['accuracy'])
        return model
```

Comment: Now we increase the number of filters from 32 to 64 when using very small filters, such as 3×3 pixels, by applying a fill operation during the convolution process. For application 1, observe the effect of this parameter on the accuracy of the system. and without adding padding, increasing the number of filters from 32 to 64. For application 1, observe the effect of this parameter on the system accuracy.