Practical 12: Write a program to demonstrate use of Convolutional Classifier.

17CP011

CNN performs functions known as convolutions and apply feature detecting filters for higher image features learning.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from keras.datasets.mnist import load_data
from keras import layers
from keras import Model
from keras.utils import to_categorical
```

In [2]:

```
(X_train, y_train), (X_test, y_test) = load_data()
X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
print("y_train: ", y_train.shape)
```

```
X_train:  (60000, 28, 28, 1)
X_test:  (10000, 28, 28, 1)
y_train:  (60000, 10)
```

In [3]:

```
input_layer = layers.Input(shape=(28, 28, 1))
conv1_layer = layers.Conv2D(64, 2, strides=2,activation='relu', padding='valid')(input_l
ayer)
conv2_layer = layers.Conv2D(64, 2, strides=2,activation='relu', padding='valid')(conv1_l
ayer)
conv3_layer = layers.Conv2D(64, 2, strides=2,activation='relu', padding='valid')(conv2_l
ayer)
conv3_layer = layers.Conv2D(64, 2, strides=2,activation='relu', padding='valid')(conv2_l
ayer)
flatten1_layer = layers.Flatten()(conv3_layer)
output_layer = layers.Dense(10, activation='softmax')(flatten1_layer)

model = Model(inputs=input_layer, outputs=output_layer)
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 64)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          16448
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          16448
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 10)                5770
```

```
dense (Dense)                (None, 10)                5770
=================================================================
Total params: 38,986
Trainable params: 38,986
Non-trainable params: 0
_________________________________________________________________
```

In [4]:

```
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.5709 - accuracy: 0.8
588 - val_loss: 0.1269 - val_accuracy: 0.9601
Epoch 2/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.1021 - accuracy: 0.9
689 - val_loss: 0.0982 - val_accuracy: 0.9721
Epoch 3/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0726 - accuracy: 0.9
779 - val_loss: 0.0995 - val_accuracy: 0.9731
Epoch 4/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0632 - accuracy: 0.9
792 - val_loss: 0.0899 - val_accuracy: 0.9755
Epoch 5/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.0536 - accuracy: 0.9
828 - val_loss: 0.0914 - val_accuracy: 0.9758
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0411 - accuracy: 0.9
863 - val_loss: 0.0912 - val_accuracy: 0.9756
Epoch 7/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0373 - accuracy: 0.9
872 - val_loss: 0.1065 - val_accuracy: 0.9743
Epoch 8/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0344 - accuracy: 0.9
889 - val_loss: 0.1088 - val_accuracy: 0.9739
Epoch 9/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0273 - accuracy: 0.9
902 - val_loss: 0.0995 - val_accuracy: 0.9757
Epoch 10/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0221 - accuracy: 0.9
926 - val_loss: 0.1040 - val_accuracy: 0.9744
```

Out[4]:

```
<tensorflow.python.keras.callbacks.History at 0x7f7ff4048650>
```

In [5]:

```
# Training set Accuracy
loss, accuracy = model.evaluate(X_train, y_train)
print("Training Set Loss: %.4f Accuracy: %.2f"%(loss, accuracy*100))

# Testing set Accuracy
loss, accuracy = model.evaluate(X_test, y_test)
print("Training Set Loss: %.4f Accuracy: %.2f"%(loss, accuracy*100))
```

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0374 - accuracy: 0.988
7
Training Set Loss: 0.0374 Accuracy: 98.87
313/313 [==============================] - 1s 4ms/step - loss: 0.0893 - accuracy: 0.9784
Training Set Loss: 0.0893 Accuracy: 97.84
```

In [6]:

```
ix = np.random.randint(0, X_test.shape[0], 9)
X = X_test[ix]
y = y_test[ix]

y_pred = model.predict(X)


for i in range(9):
```

```
    # define subplot
    plt.subplot(330 + 1 + i)
    plt.text(2, 2, str(np.argmax(y_pred[i])), bbox=dict(fill=True, edgecolor='red', line
width=2))
    # plot raw pixel data
    plt.imshow(X[i, :, :,0], cmap=plt.get_cmap('gray'))
# show the figure
plt.show()
```