DEEP LEARNING PRACTICALS

EXPERIMENT-6

AIM: To build a Convolutional Neural Network and evaluate its performance on MNIST dataset.

CODE and OUTPUT:

```
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix
```

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

```
trainX, testX = prep_pixels(trainX, testX)
```

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
model = define_model()
model.summary()
```

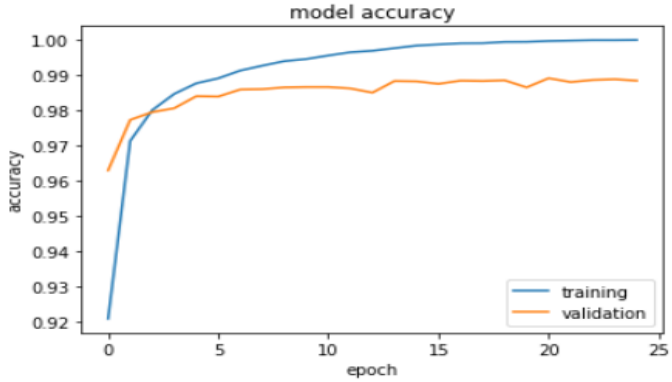
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 100)               540900
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(trainX, trainY, epochs=25, batch_size=128, validation_data=(testX, testY), verbose=1)

loss, accuracy  = model.evaluate(testX, testY, verbose=True)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

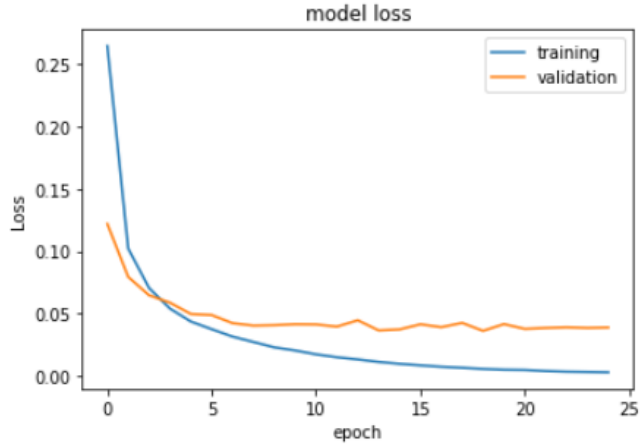
```
313/313 [==============================] - 1s 2ms/step - loss: 0.0388 - accuracy: 0.9883
```



```
loss, accuracy  = model.evaluate(testX, testY, verbose=True)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0388 - accuracy: 0.9883
```




```
y_pred = model.predict(testX)

Y_pred = np.argmax(y_pred, 1) # Decode Predicted labels
Y_test = np.argmax(testY, 1) # Decode labels

mat = confusion_matrix(Y_test, Y_pred) # Confusion matrix
print(mat)
```

```
[[ 973    0    1    0    1    1    2    1    1    0]
 [   0 1131    2    0    0    0    2    0    0    0]
 [   1    1 1019    1    3    0    2    4    1    0]
 [   0    0    1 1003    0    2    0    2    2    0]
 [   0    0    3    0  971    0    2    0    1    5]
 [   2    0    0    4    0  882    2    1    0    1]
 [   6    2    1    0    1    3  944    0    1    0]
 [   0    1    8    0    2    0    0 1013    2    2]
 [   5    0    2    1    0    2    0    2  959    3]
 [   1    1    0    3   12    1    0    3    0  988]]
```