ML Lab Assignment 5

1. Implementing a Deep Convolutional Neural Network for Handwritten Digit Recognition(0-9)

2. Implementing a Deep Convolutional Neural Network for Handwritten Character Recognition(0-9)

REPORT BY : KAPIL RATHOD
177227

Handwritten Digit Classification using Convolutional Neural Network(CNN)

Importing the Required Libraries

In [127]:
```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

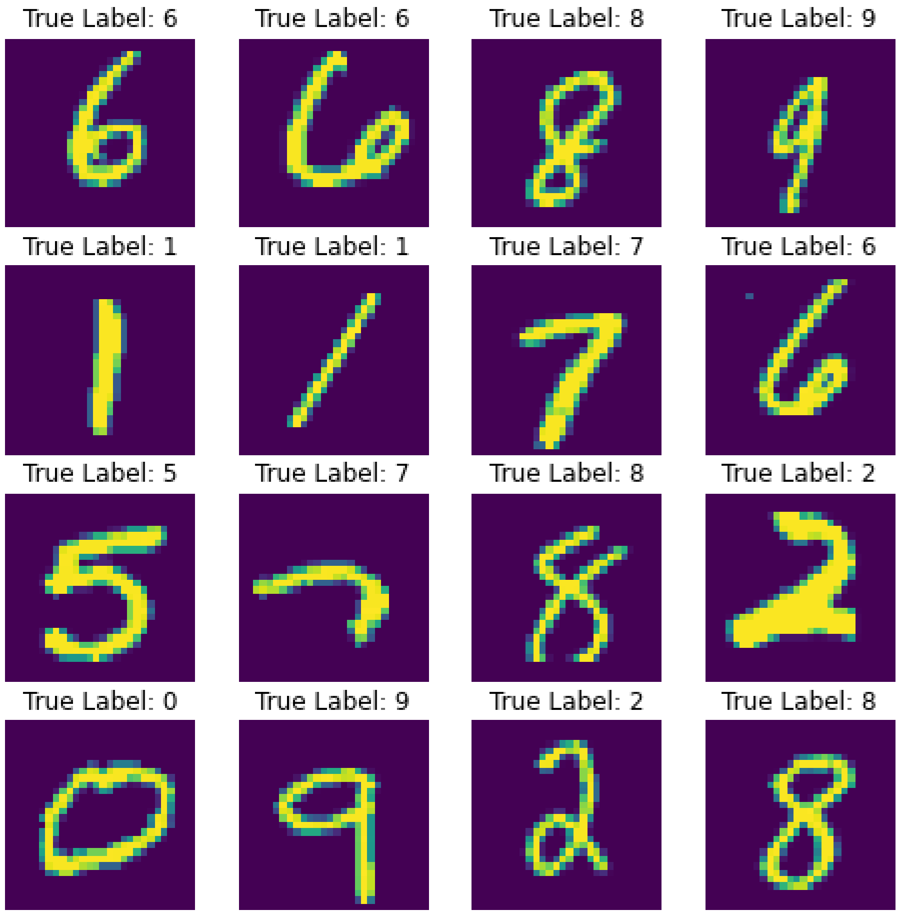
Loading the Data

In [128]:
```
# Loading data from CSV file
X = pd.read_csv('mnist_train.csv')
y = X['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Preprocessing and Visualizing the Training Data

In [129]:
```
# Storing the labels in a separate vector y_train
y_train = np.array(X_train['label'])

# Dropping the labels column
X_train = X_train.drop(['label'], axis=1)

# Normalizing the values of the pixels
X_train = X_train / 255

# Converting X_train dataframe to numpy array
X_train = np.array(X_train)

print("-------------------------")
print("SAMPLE OF TRAINING DATA")
print("-------------------------")

# Plotting the data
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    r = np.random.randint(X_train.shape[0])   # Get a random image to show
    plt.title('True Label: '+str(y_train[r])) # Show its label as title
    plt.imshow(X_train[r].reshape(28,28))    # Plotting the image
plt.show()

# Converting the labels vector into one hot encoded form
oh = np.zeros((y_train.size, y_train.max()+1))
oh[np.arange(y_train.size), y_train] = 1
y_train = np.array(oh)

# Reshaping X_train to the format of the network
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
```

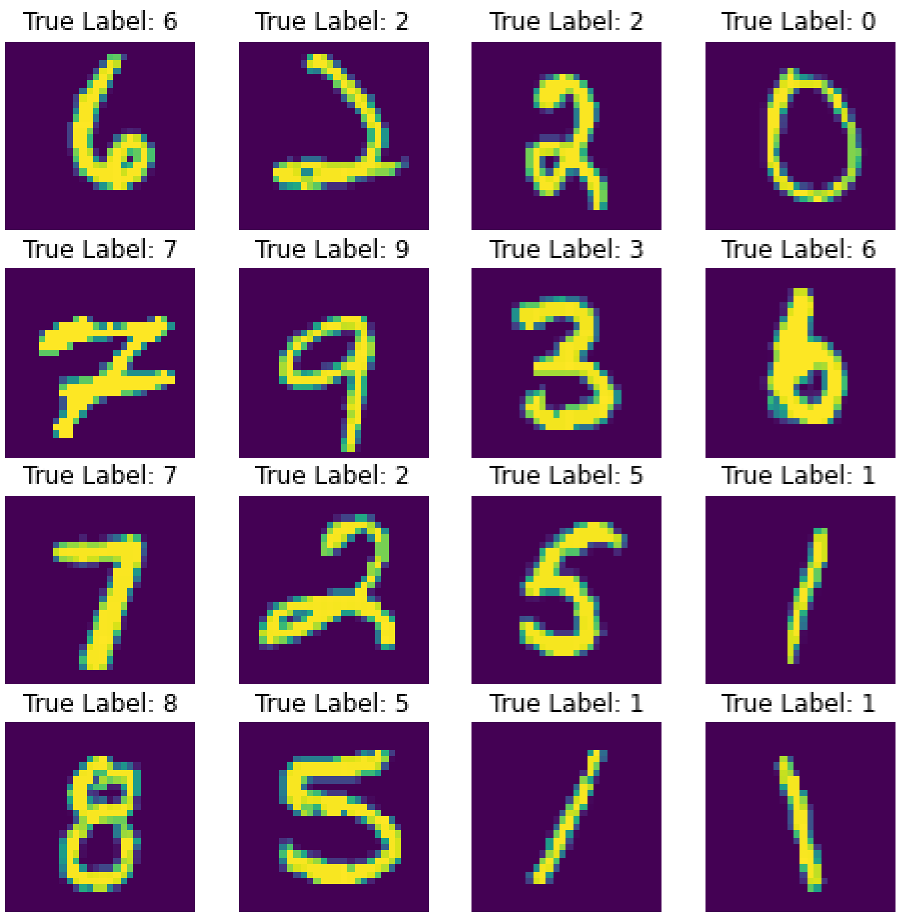
```
-------------------------
SAMPLE OF TRAINING DATA
-------------------------
```



Preprocessing and Visualizing the Testing Data

In [130]:
```
# Storing the labels in a separate vector y_train
y_test = np.array(X_test['label'])

# Dropping the labels column
X_test = X_test.drop(['label'], axis=1)

# Normalizing the values of the pixels
X_test = X_test/ 255

# Converting X_train dataframe to numpy array
X_test = np.array(X_test)

print("-------------------------")
print("SAMPLE OF TESTING DATA")
print("-------------------------")

# Plotting the data
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    r = np.random.randint(X_test.shape[0])   # Get a random image to show
    plt.title('True Label: '+str(y_test[r])) # Show its label as title
    plt.imshow(X_test[r].reshape(28,28))    # Plotting the image
plt.show()

# Converting the labels vector into one hot encoded form
oh = np.zeros((y_test.size, y_test.max()+1))
oh[np.arange(y_test.size), y_test] = 1
y_test = np.array(oh)

# Reshaping X_train to the format of the network
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

```
-------------------------
SAMPLE OF TESTING DATA
-------------------------
```



Defining the Number of Classes

In [131]:
```
no_of_label_classes = y_test.shape[1]
```

Designing the Convolutional Neural Network

In [132]:
```
def create_model():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 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(no_of_label_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
    return model
```

Initializing the Hyperparameters

In [133]:
```
iterations = 100
b_size = 20
```

Training the Model

In [134]:
```
# Creating the model
model = create_model()
# Fitting the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=iterations,
```

```
Epoch 1/100
1407/1407 - 22s - loss: 0.1973 - accuracy: 0.9383 - val_loss: 0.0812 - val_ac
curacy: 0.9745
Epoch 2/100
1407/1407 - 21s - loss: 0.0681 - accuracy: 0.9794 - val_loss: 0.0608 - val_ac
curacy: 0.9815
Epoch 3/100
1407/1407 - 21s - loss: 0.0427 - accuracy: 0.9861 - val_loss: 0.0507 - val_ac
curacy: 0.9838
Epoch 4/100
1407/1407 - 21s - loss: 0.0335 - accuracy: 0.9891 - val_loss: 0.0567 - val_ac
curacy: 0.9829
Epoch 5/100
1407/1407 - 22s - loss: 0.0235 - accuracy: 0.9918 - val_loss: 0.0467 - val_ac
curacy: 0.9855
Epoch 6/100
1407/1407 - 21s - loss: 0.0186 - accuracy: 0.9935 - val_loss: 0.0604 - val_ac
curacy: 0.9832
Epoch 7/100
1407/1407 - 21s - loss: 0.0150 - accuracy: 0.9953 - val_loss: 0.0646 - val_ac
curacy: 0.9836
Epoch 8/100
1407/1407 - 22s - loss: 0.0133 - accuracy: 0.9958 - val_loss: 0.0535 - val_ac
curacy: 0.9863
Epoch 9/100
1407/1407 - 21s - loss: 0.0099 - accuracy: 0.9967 - val_loss: 0.0590 - val_ac
curacy: 0.9864
Epoch 10/100
1407/1407 - 21s - loss: 0.0091 - accuracy: 0.9966 - val_loss: 0.0611 - val_ac
curacy: 0.9859
Epoch 11/100
1407/1407 - 21s - loss: 0.0100 - accuracy: 0.9967 - val_loss: 0.0515 - val_ac
curacy: 0.9879
Epoch 12/100
1407/1407 - 21s - loss: 0.0067 - accuracy: 0.9976 - val_loss: 0.0627 - val_ac
curacy: 0.9863
Epoch 13/100
1407/1407 - 21s - loss: 0.0064 - accuracy: 0.9977 - val_loss: 0.0581 - val_ac
curacy: 0.9874
Epoch 14/100
1407/1407 - 21s - loss: 0.0078 - accuracy: 0.9977 - val_loss: 0.0623 - val_ac
curacy: 0.9868
Epoch 15/100
1407/1407 - 21s - loss: 0.0073 - accuracy: 0.9977 - val_loss: 0.0812 - val_ac
curacy: 0.9842
Epoch 16/100
1407/1407 - 21s - loss: 0.0057 - accuracy: 0.9978 - val_loss: 0.0632 - val_ac
curacy: 0.9873
Epoch 17/100
1407/1407 - 21s - loss: 0.0070 - accuracy: 0.9978 - val_loss: 0.0771 - val_ac
curacy: 0.9853
```

```
Epoch 18/100
1407/1407 - 21s - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.0653 - val_ac
curacy: 0.9861
Epoch 19/100
1407/1407 - 21s - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.0719 - val_ac
curacy: 0.9866
Epoch 20/100
1407/1407 - 21s - loss: 0.0042 - accuracy: 0.9985 - val_loss: 0.0694 - val_ac
curacy: 0.9872
Epoch 21/100
1407/1407 - 21s - loss: 0.0057 - accuracy: 0.9984 - val_loss: 0.0821 - val_ac
curacy: 0.9863
Epoch 22/100
1407/1407 - 21s - loss: 0.0037 - accuracy: 0.9989 - val_loss: 0.0943 - val_ac
curacy: 0.9850
Epoch 23/100
1407/1407 - 21s - loss: 0.0049 - accuracy: 0.9984 - val_loss: 0.0645 - val_ac
curacy: 0.9878
Epoch 24/100
1407/1407 - 21s - loss: 0.0038 - accuracy: 0.9989 - val_loss: 0.0846 - val_ac
curacy: 0.9873
Epoch 25/100
1407/1407 - 21s - loss: 0.0043 - accuracy: 0.9988 - val_loss: 0.0781 - val_ac
curacy: 0.9877
Epoch 26/100
1407/1407 - 21s - loss: 0.0011 - accuracy: 0.9996 - val_loss: 0.0840 - val_ac
curacy: 0.9888
Epoch 27/100
1407/1407 - 21s - loss: 0.0041 - accuracy: 0.9988 - val_loss: 0.0902 - val_ac
curacy: 0.9859
Epoch 28/100
1407/1407 - 21s - loss: 0.0037 - accuracy: 0.9985 - val_loss: 0.0976 - val_ac
curacy: 0.9866
Epoch 29/100
1407/1407 - 21s - loss: 0.0039 - accuracy: 0.9990 - val_loss: 0.0894 - val_ac
curacy: 0.9869
Epoch 30/100
1407/1407 - 21s - loss: 0.0023 - accuracy: 0.9991 - val_loss: 0.0900 - val_ac
curacy: 0.9879
Epoch 31/100
1407/1407 - 21s - loss: 0.0039 - accuracy: 0.9988 - val_loss: 0.0927 - val_ac
curacy: 0.9868
Epoch 32/100
1407/1407 - 21s - loss: 0.0045 - accuracy: 0.9985 - val_loss: 0.0907 - val_ac
curacy: 0.9874
Epoch 33/100
1407/1407 - 21s - loss: 0.0039 - accuracy: 0.9991 - val_loss: 0.1035 - val_ac
curacy: 0.9856
Epoch 34/100
1407/1407 - 21s - loss: 0.0037 - accuracy: 0.9989 - val_loss: 0.0919 - val_ac
curacy: 0.9867
Epoch 35/100
1407/1407 - 21s - loss: 0.0035 - accuracy: 0.9991 - val_loss: 0.1281 - val_ac
curacy: 0.9843
Epoch 36/100
1407/1407 - 21s - loss: 0.0047 - accuracy: 0.9990 - val_loss: 0.1047 - val_ac
curacy: 0.9874
```

```
Epoch 37/100
1407/1407 - 22s - loss: 0.0022 - accuracy: 0.9993 - val_loss: 0.0993 - val_ac
curacy: 0.9877
Epoch 38/100
1407/1407 - 22s - loss: 0.0041 - accuracy: 0.9988 - val_loss: 0.1001 - val_ac
curacy: 0.9861
Epoch 39/100
1407/1407 - 21s - loss: 0.0020 - accuracy: 0.9995 - val_loss: 0.1104 - val_ac
curacy: 0.9857
Epoch 40/100
1407/1407 - 21s - loss: 0.0037 - accuracy: 0.9991 - val_loss: 0.1146 - val_ac
curacy: 0.9861
Epoch 41/100
1407/1407 - 21s - loss: 0.0034 - accuracy: 0.9990 - val_loss: 0.1088 - val_ac
curacy: 0.9879
Epoch 42/100
1407/1407 - 21s - loss: 0.0044 - accuracy: 0.9989 - val_loss: 0.1151 - val_ac
curacy: 0.9864
Epoch 43/100
1407/1407 - 21s - loss: 0.0047 - accuracy: 0.9988 - val_loss: 0.0961 - val_ac
curacy: 0.9877
Epoch 44/100
1407/1407 - 21s - loss: 0.0016 - accuracy: 0.9994 - val_loss: 0.1175 - val_ac
curacy: 0.9868
Epoch 45/100
1407/1407 - 21s - loss: 0.0059 - accuracy: 0.9986 - val_loss: 0.1042 - val_ac
curacy: 0.9875
Epoch 46/100
1407/1407 - 21s - loss: 0.0035 - accuracy: 0.9988 - val_loss: 0.1014 - val_ac
curacy: 0.9882
Epoch 47/100
1407/1407 - 21s - loss: 7.4977e-04 - accuracy: 0.9998 - val_loss: 0.1097 - va
l_accuracy: 0.9877
Epoch 48/100
1407/1407 - 21s - loss: 0.0052 - accuracy: 0.9986 - val_loss: 0.1395 - val_ac
curacy: 0.9863
Epoch 49/100
1407/1407 - 21s - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.1227 - val_ac
curacy: 0.9877
Epoch 50/100
1407/1407 - 21s - loss: 0.0032 - accuracy: 0.9992 - val_loss: 0.1425 - val_ac
curacy: 0.9862
Epoch 51/100
1407/1407 - 21s - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.1150 - val_ac
curacy: 0.9882
Epoch 52/100
1407/1407 - 21s - loss: 0.0017 - accuracy: 0.9995 - val_loss: 0.1183 - val_ac
curacy: 0.9880
Epoch 53/100
1407/1407 - 22s - loss: 0.0029 - accuracy: 0.9991 - val_loss: 0.1360 - val_ac
curacy: 0.9869
Epoch 54/100
1407/1407 - 21s - loss: 0.0027 - accuracy: 0.9993 - val_loss: 0.1509 - val_ac
curacy: 0.9856
Epoch 55/100
1407/1407 - 21s - loss: 0.0047 - accuracy: 0.9989 - val_loss: 0.1386 - val_ac
curacy: 0.9859
```

```
Epoch 56/100
1407/1407 - 21s - loss: 0.0022 - accuracy: 0.9994 - val_loss: 0.1398 - val_ac
curacy: 0.9865
Epoch 57/100
1407/1407 - 21s - loss: 0.0042 - accuracy: 0.9990 - val_loss: 0.1339 - val_ac
curacy: 0.9863
Epoch 58/100
1407/1407 - 21s - loss: 0.0025 - accuracy: 0.9994 - val_loss: 0.1454 - val_ac
curacy: 0.9874
Epoch 59/100
1407/1407 - 21s - loss: 0.0038 - accuracy: 0.9991 - val_loss: 0.1473 - val_ac
curacy: 0.9866
Epoch 60/100
1407/1407 - 21s - loss: 0.0021 - accuracy: 0.9992 - val_loss: 0.1307 - val_ac
curacy: 0.9880
Epoch 61/100
1407/1407 - 21s - loss: 8.0019e-04 - accuracy: 0.9997 - val_loss: 0.1412 - va
l_accuracy: 0.9879
Epoch 62/100
1407/1407 - 21s - loss: 0.0026 - accuracy: 0.9995 - val_loss: 0.1611 - val_ac
curacy: 0.9867
Epoch 63/100
1407/1407 - 21s - loss: 0.0027 - accuracy: 0.9994 - val_loss: 0.1517 - val_ac
curacy: 0.9872
Epoch 64/100
1407/1407 - 21s - loss: 0.0033 - accuracy: 0.9993 - val_loss: 0.1409 - val_ac
curacy: 0.9870
Epoch 65/100
1407/1407 - 21s - loss: 0.0018 - accuracy: 0.9995 - val_loss: 0.1562 - val_ac
curacy: 0.9869
Epoch 66/100
1407/1407 - 21s - loss: 0.0028 - accuracy: 0.9994 - val_loss: 0.1533 - val_ac
curacy: 0.9861
Epoch 67/100
1407/1407 - 21s - loss: 0.0022 - accuracy: 0.9993 - val_loss: 0.1576 - val_ac
curacy: 0.9872
Epoch 68/100
1407/1407 - 21s - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.1492 - val_ac
curacy: 0.9876
Epoch 69/100
1407/1407 - 21s - loss: 3.6072e-04 - accuracy: 0.9999 - val_loss: 0.1599 - va
l_accuracy: 0.9874
Epoch 70/100
1407/1407 - 21s - loss: 0.0043 - accuracy: 0.9991 - val_loss: 0.1400 - val_ac
curacy: 0.9878
Epoch 71/100
1407/1407 - 21s - loss: 0.0028 - accuracy: 0.9993 - val_loss: 0.1623 - val_ac
curacy: 0.9869
Epoch 72/100
1407/1407 - 21s - loss: 0.0019 - accuracy: 0.9996 - val_loss: 0.1750 - val_ac
curacy: 0.9866
Epoch 73/100
1407/1407 - 21s - loss: 0.0045 - accuracy: 0.9990 - val_loss: 0.1651 - val_ac
curacy: 0.9882
Epoch 74/100
1407/1407 - 21s - loss: 0.0028 - accuracy: 0.9994 - val_loss: 0.1448 - val_ac
curacy: 0.9881
```

```
Epoch 75/100
1407/1407 - 21s - loss: 0.0030 - accuracy: 0.9995 - val_loss: 0.1658 - val_ac
curacy: 0.9875
Epoch 76/100
1407/1407 - 21s - loss: 0.0025 - accuracy: 0.9993 - val_loss: 0.1882 - val_ac
curacy: 0.9859
Epoch 77/100
1407/1407 - 21s - loss: 0.0026 - accuracy: 0.9994 - val_loss: 0.1613 - val_ac
curacy: 0.9879
Epoch 78/100
1407/1407 - 21s - loss: 0.0027 - accuracy: 0.9993 - val_loss: 0.1693 - val_ac
curacy: 0.9878
Epoch 79/100
1407/1407 - 21s - loss: 0.0021 - accuracy: 0.9994 - val_loss: 0.1863 - val_ac
curacy: 0.9859
Epoch 80/100
1407/1407 - 21s - loss: 0.0028 - accuracy: 0.9996 - val_loss: 0.1809 - val_ac
curacy: 0.9858
Epoch 81/100
1407/1407 - 21s - loss: 0.0028 - accuracy: 0.9995 - val_loss: 0.1909 - val_ac
curacy: 0.9874
Epoch 82/100
1407/1407 - 21s - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.1718 - val_ac
curacy: 0.9873
Epoch 83/100
1407/1407 - 21s - loss: 0.0048 - accuracy: 0.9988 - val_loss: 0.2144 - val_ac
curacy: 0.9864
Epoch 84/100
1407/1407 - 21s - loss: 0.0044 - accuracy: 0.9991 - val_loss: 0.1836 - val_ac
curacy: 0.9865
Epoch 85/100
1407/1407 - 21s - loss: 0.0026 - accuracy: 0.9994 - val_loss: 0.2137 - val_ac
curacy: 0.9851
Epoch 86/100
1407/1407 - 21s - loss: 0.0020 - accuracy: 0.9995 - val_loss: 0.1678 - val_ac
curacy: 0.9882
Epoch 87/100
1407/1407 - 21s - loss: 0.0023 - accuracy: 0.9997 - val_loss: 0.2113 - val_ac
curacy: 0.9872
Epoch 88/100
1407/1407 - 21s - loss: 0.0038 - accuracy: 0.9992 - val_loss: 0.2044 - val_ac
curacy: 0.9869
Epoch 89/100
1407/1407 - 21s - loss: 0.0043 - accuracy: 0.9993 - val_loss: 0.1879 - val_ac
curacy: 0.9866
Epoch 90/100
1407/1407 - 21s - loss: 0.0024 - accuracy: 0.9995 - val_loss: 0.1852 - val_ac
curacy: 0.9874
Epoch 91/100
1407/1407 - 21s - loss: 0.0042 - accuracy: 0.9992 - val_loss: 0.1818 - val_ac
curacy: 0.9872
Epoch 92/100
1407/1407 - 21s - loss: 0.0031 - accuracy: 0.9995 - val_loss: 0.1623 - val_ac
curacy: 0.9877
Epoch 93/100
1407/1407 - 21s - loss: 3.1192e-04 - accuracy: 0.9999 - val_loss: 0.1649 - va
l_accuracy: 0.9885
```

```
Epoch 94/100
1407/1407 - 21s - loss: 0.0018 - accuracy: 0.9996 - val_loss: 0.2199 - val_ac
curacy: 0.9861
Epoch 95/100
1407/1407 - 21s - loss: 0.0023 - accuracy: 0.9996 - val_loss: 0.2103 - val_ac
curacy: 0.9869
Epoch 96/100
1407/1407 - 21s - loss: 0.0034 - accuracy: 0.9996 - val_loss: 0.1989 - val_ac
curacy: 0.9874
Epoch 97/100
1407/1407 - 22s - loss: 0.0044 - accuracy: 0.9994 - val_loss: 0.1951 - val_ac
curacy: 0.9885
Epoch 98/100
1407/1407 - 22s - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.1841 - val_ac
curacy: 0.9873
Epoch 99/100
1407/1407 - 21s - loss: 0.0017 - accuracy: 0.9995 - val_loss: 0.1867 - val_ac
curacy: 0.9874
Epoch 100/100
1407/1407 - 21s - loss: 0.0027 - accuracy: 0.9996 - val_loss: 0.2268 - val_ac
curacy: 0.9878
```

Out[134]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd5512dc5c0>
```

FINAL VALIDATION ACCURACY : 98.78

In [ ]:

Handwritten Character Classification using Convolutional Neural Network(CNN)

Importing the Required Libraries

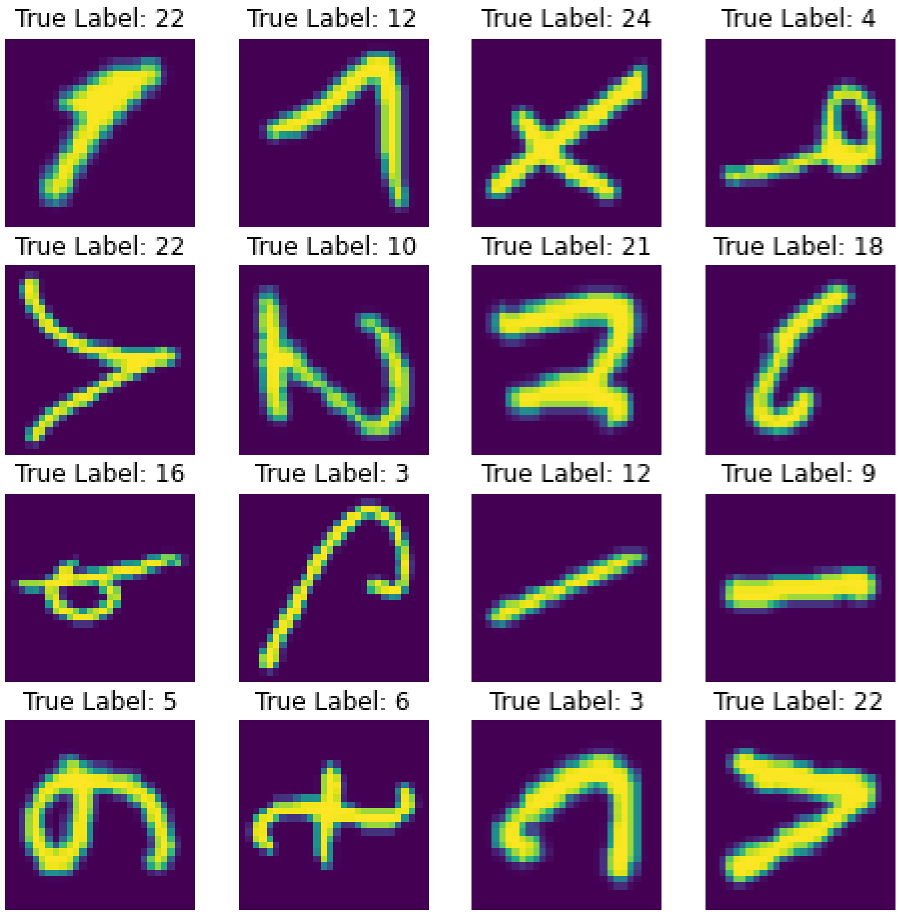
In [19]:
```
import pandas as pd
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

In [20]:
```
# Loading data from CSV file
X = pd.read_csv('emnist-letters.csv')
y = X['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
```

Preprocessing and Visualizing the Training Data

In [21]:
```
# Storing the labels in a separate vector y_train
y_train = np.array(X_train['label'])

# Dropping the labels column
X_train = X_train.drop(['label'], axis=1)

# Normalizing the values of the pixels
X_train = X_train / 255

# Converting X_train dataframe to numpy array
X_train = np.array(X_train)

print("-------------------------")
print("SAMPLE OF TRAINING DATA")
print("-------------------------")

# Plotting the data
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    r = np.random.randint(X_train.shape[0])   # Get a random image to show
    plt.title('True Label: '+str(y_train[r])) # Show its label as title
    plt.imshow(X_train[r].reshape(28,28))    # Plotting the image
plt.show()

# Converting the labels vector into one hot encoded form
oh = np.zeros((y_train.size, y_train.max()+1))
oh[np.arange(y_train.size), y_train] = 1
y_train = np.array(oh)

# Reshaping X_train to the format of the network
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
```

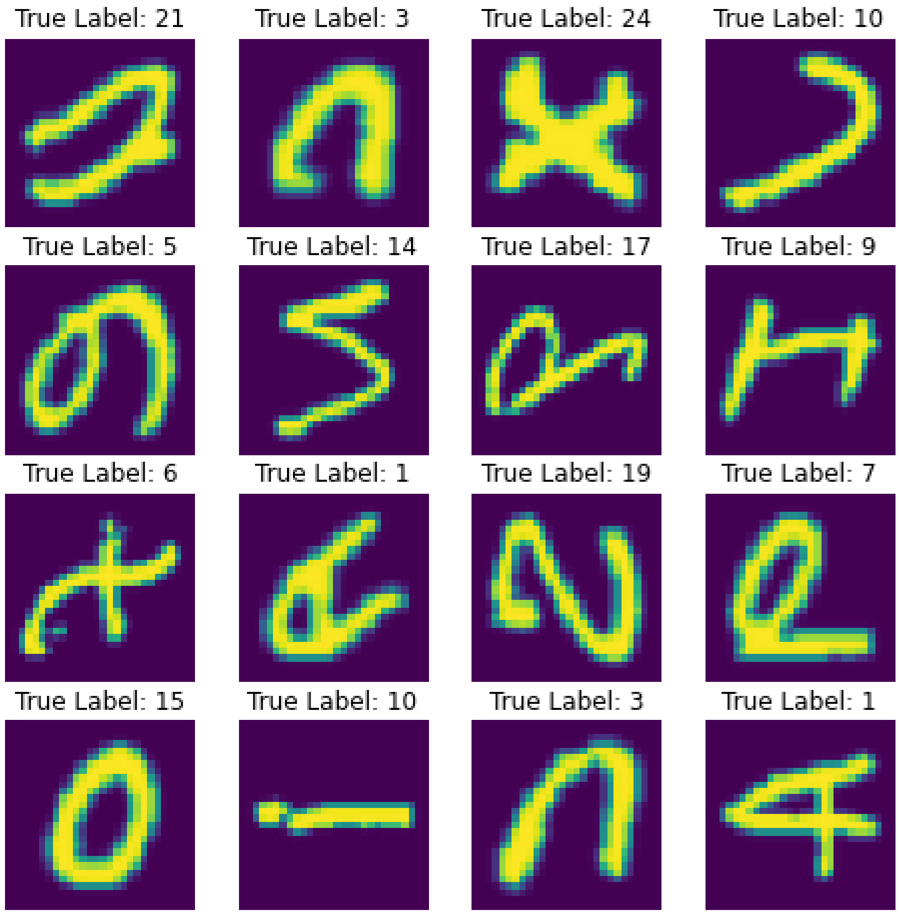
```
-------------------------
SAMPLE OF TRAINING DATA
-------------------------
```



Preprocessing and Visualizing the Testing Data

In [22]:
```
# Storing the labels in a separate vector y_train
y_test = np.array(X_test['label'])

# Dropping the labels column
X_test = X_test.drop(['label'], axis=1)

# Normalizing the values of the pixels
X_test = X_test/ 255

# Converting X_train dataframe to numpy array
X_test = np.array(X_test)

print("-------------------------")
print("SAMPLE OF TESTING DATA")
print("-------------------------")

# Plotting the data
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    r = np.random.randint(X_test.shape[0])   # Get a random image to show
    plt.title('True Label: '+str(y_test[r])) # Show its label as title
    plt.imshow(X_test[r].reshape(28,28))    # Plotting the image
plt.show()

# Converting the labels vector into one hot encoded form
oh = np.zeros((y_test.size, y_test.max()+1))
oh[np.arange(y_test.size), y_test] = 1
y_test = np.array(oh)

# Reshaping X_train to the format of the network
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

```
-------------------------
SAMPLE OF TESTING DATA
-------------------------
```



Defining the Number of Classes

In [23]:
```
no_of_label_classes = y_train.shape[1]
```

Designing the Convolutional Neural Network

In [24]:
```
def create_model():
    # create model
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 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(no_of_label_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
    return model
```

Initializing the Hyperparameters

In [27]:
```
iterations = 30
b_size = 20
```

Training the Model

In [28]:
```
# Creating the model
model = create_model()
# Fitting the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=iterations,
```

```
Epoch 1/30
235/235 - 4s - loss: 1.5554 - accuracy: 0.5529 - val_loss: 1.0177 - val_accurac
y: 0.6900
Epoch 2/30
235/235 - 4s - loss: 0.7240 - accuracy: 0.7797 - val_loss: 0.7632 - val_accurac
y: 0.7550
Epoch 3/30
235/235 - 4s - loss: 0.4874 - accuracy: 0.8477 - val_loss: 0.6690 - val_accurac
y: 0.7840
Epoch 4/30
235/235 - 4s - loss: 0.3619 - accuracy: 0.8865 - val_loss: 0.6364 - val_accurac
y: 0.8004
Epoch 5/30
235/235 - 4s - loss: 0.2834 - accuracy: 0.9115 - val_loss: 0.5860 - val_accurac
y: 0.8117
Epoch 6/30
235/235 - 4s - loss: 0.2176 - accuracy: 0.9307 - val_loss: 0.5702 - val_accurac
y: 0.8281
Epoch 7/30
235/235 - 4s - loss: 0.1745 - accuracy: 0.9411 - val_loss: 0.6160 - val_accurac
y: 0.8208
Epoch 8/30
235/235 - 4s - loss: 0.1456 - accuracy: 0.9477 - val_loss: 0.5865 - val_accurac
y: 0.8290
Epoch 9/30
235/235 - 4s - loss: 0.1214 - accuracy: 0.9573 - val_loss: 0.6009 - val_accurac
y: 0.8368
Epoch 10/30
235/235 - 4s - loss: 0.1013 - accuracy: 0.9627 - val_loss: 0.6184 - val_accurac
y: 0.8394
Epoch 11/30
235/235 - 4s - loss: 0.0854 - accuracy: 0.9701 - val_loss: 0.6264 - val_accurac
y: 0.8299
Epoch 12/30
235/235 - 4s - loss: 0.0772 - accuracy: 0.9710 - val_loss: 0.6719 - val_accurac
y: 0.8346
Epoch 13/30
235/235 - 4s - loss: 0.0783 - accuracy: 0.9735 - val_loss: 0.6496 - val_accurac
y: 0.8368
Epoch 14/30
235/235 - 4s - loss: 0.0743 - accuracy: 0.9725 - val_loss: 0.6799 - val_accurac
y: 0.8303
Epoch 15/30
235/235 - 4s - loss: 0.0590 - accuracy: 0.9782 - val_loss: 0.7436 - val_accurac
y: 0.8290
Epoch 16/30
235/235 - 4s - loss: 0.0718 - accuracy: 0.9748 - val_loss: 0.7282 - val_accurac
y: 0.8325
Epoch 17/30
235/235 - 4s - loss: 0.0706 - accuracy: 0.9757 - val_loss: 0.7571 - val_accurac
y: 0.8351
```

```
Epoch 18/30
235/235 - 4s - loss: 0.0582 - accuracy: 0.9795 - val_loss: 0.7775 - val_accurac
y: 0.8312
Epoch 19/30
235/235 - 4s - loss: 0.0469 - accuracy: 0.9817 - val_loss: 0.7261 - val_accurac
y: 0.8433
Epoch 20/30
235/235 - 4s - loss: 0.0540 - accuracy: 0.9806 - val_loss: 0.7766 - val_accurac
y: 0.8368
Epoch 21/30
235/235 - 4s - loss: 0.0404 - accuracy: 0.9851 - val_loss: 0.7807 - val_accurac
y: 0.8437
Epoch 22/30
235/235 - 4s - loss: 0.0473 - accuracy: 0.9834 - val_loss: 0.8312 - val_accurac
y: 0.8255
Epoch 23/30
235/235 - 4s - loss: 0.0595 - accuracy: 0.9767 - val_loss: 0.8422 - val_accurac
y: 0.8260
Epoch 24/30
235/235 - 4s - loss: 0.0574 - accuracy: 0.9774 - val_loss: 0.7451 - val_accurac
y: 0.8407
Epoch 25/30
235/235 - 4s - loss: 0.0424 - accuracy: 0.9825 - val_loss: 0.8520 - val_accurac
y: 0.8294
Epoch 26/30
235/235 - 4s - loss: 0.0453 - accuracy: 0.9831 - val_loss: 0.8499 - val_accurac
y: 0.8390
Epoch 27/30
235/235 - 4s - loss: 0.0366 - accuracy: 0.9885 - val_loss: 0.7616 - val_accurac
y: 0.8511
Epoch 28/30
235/235 - 4s - loss: 0.0307 - accuracy: 0.9898 - val_loss: 0.7989 - val_accurac
y: 0.8381
Epoch 29/30
235/235 - 4s - loss: 0.0287 - accuracy: 0.9898 - val_loss: 0.9088 - val_accurac
y: 0.8303
Epoch 30/30
235/235 - 4s - loss: 0.0477 - accuracy: 0.9804 - val_loss: 0.8951 - val_accurac
y: 0.8351
```

Out[28]:
```
<tensorflow.python.keras.callbacks.History at 0x7f0903592ef0>
```

FINAL VALIDATION ACCURACY : 83.51

In [ ]:
```

```