

#Phapale Apeksha Roll NO: 4231 Div:B

DL Practical NO.2

```
import pandas as pd
import numpy as np

df = pd.read_csv(r'C:\Users\apeksha\Downloads\DL\pract_2\letter-recognition.csv')
print(df)
columns = ["letter", "x-box", "y-box", "width", "height", "onpix", "x-bar",
"y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-ege", "xegvy",
"y-ege", "yegvx"]
x = df.drop("letter", axis=1).values
y = df["letter"].values
print("shape of x")
print(x.shape)
print("shape of y")
print(y.shape)
print(np.unique(y))

# Split the data into training and testing sets
test_size = 0.2 # Percentage of data to be used for testing
num_test_samples = int(test_size * len(df))

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[:num_test_samples]
train_indices = shuffled_indices[num_test_samples:]

# Split the features and target variable based on the indices
x_train = x[train_indices]
x_test = x[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

print("Shape of x_train:", x_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

print(x_train[0])
```

```
print(y_train[0])
```

```
class_names=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

```
print(x_test[10])
```

```
print(y_test[10])
```

```
#preprocessing step
```

```
x_train = x_train/255
```

```
x_test = x_test/255
```

```
from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
```

```
y_train = encoder.fit_transform(y_train)
```

```
y_test = encoder.fit_transform(y_test)
```

```
#model building
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
```

```
model=Sequential()
```

```
model.add(Dense(512, activation='relu', input_shape=(16,)))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(256, activation='relu'))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(26, activation='softmax'))
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
```

```
metrics=['accuracy'])
```

```
model.summary()
```

```
#model training
```

```
model.fit(x_train, y_train, epochs=50, batch_size=128, verbose=1, validation_data=(x_test, y_test))
```

```
#testing model
```

```
predictions = model.predict(x_test)
```

```
index=10
```

```
print(predictions[index])
```

```
final_value=np.argmax(predictions[index])
```

```
print("Actual label :",y_test[index])
```

```
print("Predicted label :",final_value)
```

```
print("Class (A-Z) :",class_names[final_value])
```

```
#model Evaluation
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print("Loss :",loss)
print("Accuracy (Test Data) :",accuracy*100)
```

Output:

```
letter xbox ybox width ... xedge xedgey yedge yedgex
0    T   2   8   3 ...   0   8   0   8
1    I   5  12   3 ...   2   8   4  10
2    D   4  11   6 ...   3   7   3   9
3    N   7  11   6 ...   6  10   2   8
4    G   2   1   3 ...   1   7   5  10
...   ...   ...   ...   ...   ...   ...   ...
19995 D   2   2   3 ...   2   8   3   7
19996 C   7  10   8 ...   2   9   3   7
19997 T   6   9   6 ...   2  12   2   4
19998 S   2   3   4 ...   1   9   5   8
19999 A   4   9   6 ...   2   7   2   8
```

```
[20000 rows x 17 columns]
```

```
shape of x
```

```
(20000, 16)
```

```
shape of y
```

```
(20000,)
```

```
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R'
 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z']
```

```
Shape of x_train: (16000, 16)
```

```
Shape of x_test: (4000, 16)
```

```
Shape of y_train: (16000,)
```

```
Shape of y_test: (4000,)
```

```
[4 6 6 5 5 7 7 4 4 6 5 8 6 8 4 6]
```

```
N
```

```
[1 3 1 2 0 7 7 1 8 7 6 8 0 8 3 8]
```

```
I
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	8704

dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 26)	6682

=====

Total params: 146,714

Trainable params: 146,714

Non-trainable params: 0

Epoch 1/50

125/125 [=====] - 1s 7ms/step - loss: 3.1287 - accuracy: 0.1683 - val_loss: 2.7543 - val_accuracy: 0.3203

Epoch 2/50

125/125 [=====] - 1s 6ms/step - loss: 2.3519 - accuracy: 0.3257 - val_loss: 2.0490 - val_accuracy: 0.4112

Epoch 3/50

125/125 [=====] - 1s 6ms/step - loss: 1.9355 - accuracy: 0.4156 - val_loss: 1.7457 - val_accuracy: 0.4885

Epoch 4/50

125/125 [=====] - 1s 6ms/step - loss: 1.7040 - accuracy: 0.4884 - val_loss: 1.5754 - val_accuracy: 0.5428

Epoch 5/50

125/125 [=====] - 1s 5ms/step - loss: 1.5696 - accuracy: 0.5303 - val_loss: 1.4665 - val_accuracy: 0.5705

Epoch 6/50

125/125 [=====] - 1s 6ms/step - loss: 1.4787 - accuracy: 0.5598 - val_loss: 1.3805 - val_accuracy: 0.6037

Epoch 7/50

125/125 [=====] - 1s 7ms/step - loss: 1.4022 - accuracy: 0.5866 - val_loss: 1.3334 - val_accuracy: 0.6125

Epoch 8/50

125/125 [=====] - 1s 6ms/step - loss: 1.3565 - accuracy: 0.5974 - val_loss: 1.2783 - val_accuracy: 0.6315

Epoch 9/50

125/125 [=====] - 1s 5ms/step - loss: 1.2979 - accuracy: 0.6168 - val_loss: 1.2290 - val_accuracy: 0.6345

Epoch 10/50

125/125 [=====] - 1s 6ms/step - loss: 1.2422 - accuracy: 0.6357 - val_loss: 1.1690 - val_accuracy: 0.6718
Epoch 11/50
125/125 [=====] - 1s 6ms/step - loss: 1.1950 - accuracy: 0.6498 - val_loss: 1.1155 - val_accuracy: 0.6842
Epoch 12/50
125/125 [=====] - 1s 5ms/step - loss: 1.1487 - accuracy: 0.6609 - val_loss: 1.0799 - val_accuracy: 0.6938
Epoch 13/50
125/125 [=====] - 1s 6ms/step - loss: 1.1130 - accuracy: 0.6729 - val_loss: 1.0414 - val_accuracy: 0.6995
Epoch 14/50
125/125 [=====] - 1s 6ms/step - loss: 1.0716 - accuracy: 0.6819 - val_loss: 1.0016 - val_accuracy: 0.7140
Epoch 15/50
125/125 [=====] - 1s 6ms/step - loss: 1.0416 - accuracy: 0.6911 - val_loss: 0.9684 - val_accuracy: 0.7172
Epoch 16/50
125/125 [=====] - 1s 6ms/step - loss: 1.0067 - accuracy: 0.6962 - val_loss: 0.9760 - val_accuracy: 0.7140
Epoch 17/50
125/125 [=====] - 1s 6ms/step - loss: 0.9810 - accuracy: 0.7078 - val_loss: 0.9111 - val_accuracy: 0.7390
Epoch 18/50
125/125 [=====] - 1s 6ms/step - loss: 0.9423 - accuracy: 0.7169 - val_loss: 0.8839 - val_accuracy: 0.7427
Epoch 19/50
125/125 [=====] - 1s 6ms/step - loss: 0.9161 - accuracy: 0.7271 - val_loss: 0.8605 - val_accuracy: 0.7480
Epoch 20/50
125/125 [=====] - 1s 6ms/step - loss: 0.8934 - accuracy: 0.7290 - val_loss: 0.8480 - val_accuracy: 0.7465
Epoch 21/50
125/125 [=====] - 1s 6ms/step - loss: 0.8738 - accuracy: 0.7359 - val_loss: 0.8183 - val_accuracy: 0.7632
Epoch 22/50
125/125 [=====] - 1s 7ms/step - loss: 0.8545 - accuracy: 0.7408 - val_loss: 0.7997 - val_accuracy: 0.7592
Epoch 23/50
125/125 [=====] - 1s 6ms/step - loss: 0.8356 - accuracy: 0.7479 - val_loss: 0.7805 - val_accuracy: 0.7628
Epoch 24/50

125/125 [=====] - 1s 7ms/step - loss: 0.8167 - accuracy: 0.7516 - val_loss: 0.7541 - val_accuracy: 0.7747
Epoch 25/50
125/125 [=====] - 1s 7ms/step - loss: 0.7982 - accuracy: 0.7580 - val_loss: 0.7389 - val_accuracy: 0.7812
Epoch 26/50
125/125 [=====] - 1s 7ms/step - loss: 0.7831 - accuracy: 0.7637 - val_loss: 0.7354 - val_accuracy: 0.7720
Epoch 27/50
125/125 [=====] - 1s 6ms/step - loss: 0.7728 - accuracy: 0.7638 - val_loss: 0.7095 - val_accuracy: 0.7912
Epoch 28/50
125/125 [=====] - 1s 6ms/step - loss: 0.7552 - accuracy: 0.7686 - val_loss: 0.7051 - val_accuracy: 0.7862
Epoch 29/50
125/125 [=====] - 1s 7ms/step - loss: 0.7402 - accuracy: 0.7725 - val_loss: 0.6894 - val_accuracy: 0.7920
Epoch 30/50
125/125 [=====] - 1s 6ms/step - loss: 0.7271 - accuracy: 0.7766 - val_loss: 0.6751 - val_accuracy: 0.7918
Epoch 31/50
125/125 [=====] - 1s 6ms/step - loss: 0.7144 - accuracy: 0.7790 - val_loss: 0.6678 - val_accuracy: 0.7977
Epoch 32/50
125/125 [=====] - 1s 6ms/step - loss: 0.6950 - accuracy: 0.7866 - val_loss: 0.6394 - val_accuracy: 0.8077
Epoch 33/50
125/125 [=====] - 1s 6ms/step - loss: 0.6847 - accuracy: 0.7910 - val_loss: 0.6516 - val_accuracy: 0.8010
Epoch 34/50
125/125 [=====] - 1s 6ms/step - loss: 0.6696 - accuracy: 0.7931 - val_loss: 0.6345 - val_accuracy: 0.8060
Epoch 35/50
125/125 [=====] - 1s 6ms/step - loss: 0.6600 - accuracy: 0.7968 - val_loss: 0.6001 - val_accuracy: 0.8185
Epoch 36/50
125/125 [=====] - 1s 5ms/step - loss: 0.6448 - accuracy: 0.8016 - val_loss: 0.5999 - val_accuracy: 0.8170
Epoch 37/50
125/125 [=====] - 0s 3ms/step - loss: 0.6355 - accuracy: 0.8054 - val_loss: 0.5788 - val_accuracy: 0.8195
Epoch 38/50

125/125 [=====] - 0s 4ms/step - loss: 0.6240 - accuracy: 0.8071 - val_loss:
0.5701 - val_accuracy: 0.8255
Epoch 39/50
125/125 [=====] - 1s 4ms/step - loss: 0.6099 - accuracy: 0.8106 - val_loss:
0.5735 - val_accuracy: 0.8192
Epoch 40/50
125/125 [=====] - 0s 4ms/step - loss: 0.6054 - accuracy: 0.8142 - val_loss:
0.5661 - val_accuracy: 0.8275
Epoch 41/50
125/125 [=====] - 0s 4ms/step - loss: 0.5954 - accuracy: 0.8156 - val_loss:
0.5394 - val_accuracy: 0.8322
Epoch 42/50
125/125 [=====] - 1s 4ms/step - loss: 0.5823 - accuracy: 0.8202 - val_loss:
0.5310 - val_accuracy: 0.8372
Epoch 43/50
125/125 [=====] - 1s 5ms/step - loss: 0.5741 - accuracy: 0.8206 - val_loss:
0.5314 - val_accuracy: 0.8335
Epoch 44/50
125/125 [=====] - 1s 4ms/step - loss: 0.5711 - accuracy: 0.8209 - val_loss:
0.5194 - val_accuracy: 0.8382
Epoch 45/50
125/125 [=====] - 1s 5ms/step - loss: 0.5549 - accuracy: 0.8297 - val_loss:
0.5096 - val_accuracy: 0.8425
Epoch 46/50
125/125 [=====] - 1s 5ms/step - loss: 0.5517 - accuracy: 0.8314 - val_loss:
0.4934 - val_accuracy: 0.8435
Epoch 47/50
125/125 [=====] - 1s 5ms/step - loss: 0.5375 - accuracy: 0.8311 - val_loss:
0.4827 - val_accuracy: 0.8475
Epoch 48/50
125/125 [=====] - 1s 5ms/step - loss: 0.5306 - accuracy: 0.8357 - val_loss:
0.4778 - val_accuracy: 0.8515
Epoch 49/50
125/125 [=====] - 1s 5ms/step - loss: 0.5211 - accuracy: 0.8364 - val_loss:
0.4640 - val_accuracy: 0.8605
Epoch 50/50
125/125 [=====] - 1s 5ms/step - loss: 0.5127 - accuracy: 0.8398 - val_loss:
0.4571 - val_accuracy: 0.8622
125/125 [=====] - 0s 1ms/step

[1.3898159e-10 9.1300986e-12 6.5993061e-07 4.6125048e-11 2.4729352e-05
2.9295850e-06 1.3052782e-09 2.5103008e-12 9.9594790e-01 8.2694432e-06

5.1369742e-10 1.2201001e-04 1.7026043e-25 1.8152464e-21 1.9431759e-16
6.4693090e-12 9.0628713e-11 9.0060873e-16 8.4172538e-04 1.5709724e-04
1.9780768e-12 7.9353318e-16 2.3770898e-31 2.8693927e-03 8.3942140e-08
2.5305171e-05]

Actual label : 8

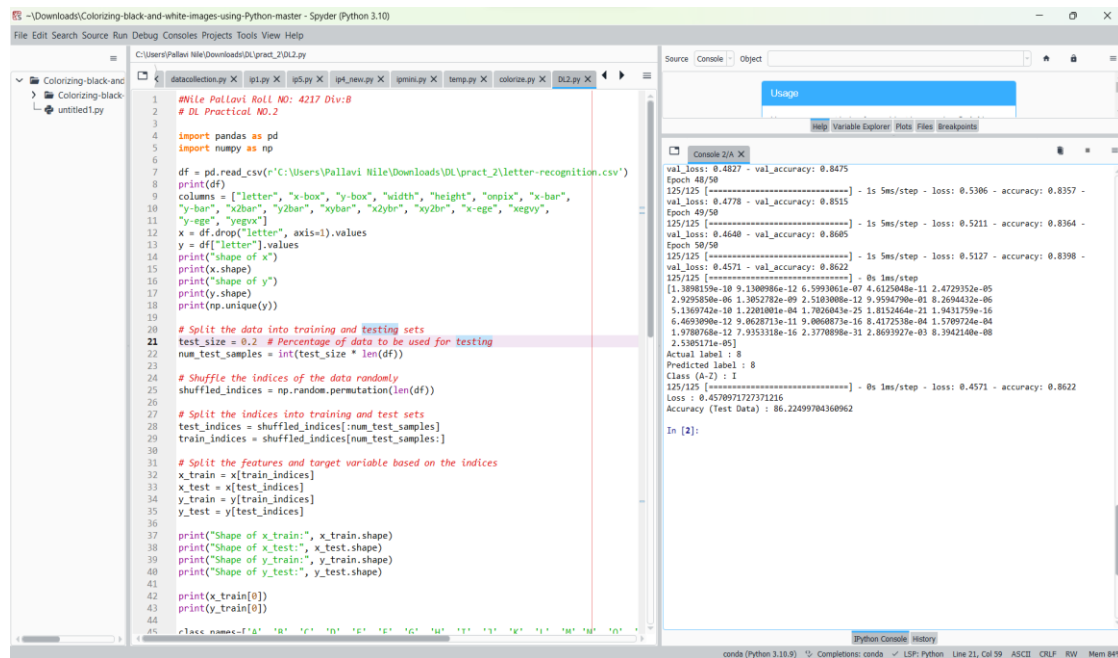
Predicted label : 8

Class (A-Z) : I

125/125 [=====] - 0s 1ms/step - loss: 0.4571 - accuracy: 0.8622

Loss : 0.4570971727371216

Accuracy (Test Data) : 86.22499704360962



```
#Nile Pallavi Roll NO: 4217 Div:8
#DL Practical NO.2

import pandas as pd
import numpy as np

df = pd.read_csv(r'C:\Users\Pallavi Nile\Downloads\DL\pract_2\letter-recognition.csv')
print(df)
columns = ["letter", "x-box", "y-box", "width", "height", "ompix", "x-bar",
            "y-bar", "x2bar", "y2bar", "xybar", "xy2br", "x-eg", "xegy",
            "y-eg", "yegx"]
x = df.drop("letter", axis=1).values
y = df["letter"].values
print("shape of x")
print(x.shape)
print("shape of y")
print(y.shape)
print(np.unique(y))

# Split the data into training and testing sets
test_size = 0.2 # Percentage of data to be used for testing
num_test_samples = int(test_size * len(df))

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[num_test_samples:]
train_indices = shuffled_indices[:num_test_samples]

# Split the features and target variable based on the indices
x_train = x[train_indices]
x_test = x[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

print("Shape of x_train:", x_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

print(x_train[0])
print(y_train[0])

class namac="A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

val_loss: 0.4827 - val_accuracy: 0.8475
Epoch 48/50
125/125 [=====] - 1s 5ms/step - loss: 0.5306 - accuracy: 0.8357 -
val_loss: 0.4778 - val_accuracy: 0.8515
Epoch 49/50
125/125 [=====] - 1s 5ms/step - loss: 0.5211 - accuracy: 0.8364 -
val_loss: 0.4640 - val_accuracy: 0.8605
Epoch 50/50
125/125 [=====] - 1s 5ms/step - loss: 0.5127 - accuracy: 0.8398 -
val_loss: 0.4571 - val_accuracy: 0.8622
125/125 [=====] - 0s 1ms/step
[1.3898159e-10 9.1308986e-12 6.5993061e-07 4.6125048e-11 2.472952e-05
2.925850e-06 1.3852702e-09 2.5181008e-12 9.9584790e-01 8.2684432e-06
5.1369742e-10 1.2201001e-04 1.7026043e-25 1.8152464e-21 1.9431759e-16
6.4693090e-12 9.0628713e-11 9.0060873e-16 8.4172538e-04 1.5709724e-04
1.9780768e-12 7.9353318e-16 2.3770898e-31 2.8693927e-03 8.3942140e-08
2.5305171e-05]
Actual label : 8
Predicted label : 8
Class (A-Z) : I
125/125 [=====] - 0s 1ms/step - loss: 0.4571 - accuracy: 0.8622
Loss : 0.4570971727371216
Accuracy (Test Data) : 86.22499704360962
In [2]:


```
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Pallavi Nile\Downloads\DL\pract_2\DL2.py

#Nile Pallavi Roll NO: 4217 Div:B
# DL Practical NO.2
import pandas as pd
import numpy as np

df = pd.read_csv('C:\Users\Pallavi Nile\Downloads\DL\pract_2\letter-recognition.csv')
print(df)
columns = ["letter", "x-box", "y-box", "width", "height", "onpix", "x-bar",
           "y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-egs", "xegvy",
           "y-egs", "yegvy"]
x = df.drop("letter", axis=1).values
y = df["letter"].values
print("shape of x")
print(x.shape)
print("shape of y")
print(y.shape)
print(np.unique(y))

# Split the data into training and testing sets
test_size = 0.2 # Percentage of data to be used for testing
num_test_samples = int(test_size * len(df))

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[num_test_samples:]
train_indices = shuffled_indices[:num_test_samples]

# Split the features and target variable based on the indices
x_train = x[train_indices]
x_test = x[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

print("Shape of x_train:", x_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

print(x_train[0])
print(y_train[0])

# Save the data
np.save('x_train.npy', x_train)
np.save('y_train.npy', y_train)
np.save('x_test.npy', x_test)
np.save('y_test.npy', y_test)
```

Source Console Object

Console 2/A X

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 26)	6682

Total params: 146,714
Trainable params: 146,714
Non-trainable params: 0

Epoch 1/50
125/125 [=====] - 1s 7ms/step - loss: 3.1287 - accuracy: 0.1683 - val_loss: 2.7543 - val_accuracy: 0.3283
Epoch 2/50
125/125 [=====] - 1s 6ms/step - loss: 2.3519 - accuracy: 0.3257 - val_loss: 2.0490 - val_accuracy: 0.4112
Epoch 3/50
125/125 [=====] - 1s 6ms/step - loss: 1.9355 - accuracy: 0.4156 - val_loss: 1.7457 - val_accuracy: 0.4885
Epoch 4/50
125/125 [=====] - 1s 6ms/step - loss: 1.7040 - accuracy: 0.4884 - val_loss: 1.5754 - val_accuracy: 0.5428
Epoch 5/50
125/125 [=====] - 1s 5ms/step - loss: 1.5696 - accuracy: 0.5303 - val_loss: 1.4665 - val_accuracy: 0.5705
Epoch 6/50
125/125 [=====] - 1s 6ms/step - loss: 1.4787 - accuracy: 0.5598 - val_loss: 1.3805 - val_accuracy: 0.6037
Epoch 7/50
125/125 [=====] - 1s 7ms/step - loss: 1.4022 - accuracy: 0.5866 - val_loss: 1.3334 - val_accuracy: 0.6125
Epoch 8/50
125/125 [=====] - 1s 6ms/step - loss: 1.3565 - accuracy: 0.5974 - val_loss: 1.2783 - val_accuracy: 0.6315
Epoch 9/50
125/125 [=====] - 1s 5ms/step - loss: 1.2979 - accuracy: 0.6168 - val_loss: 1.2290 - val_accuracy: 0.6345
Epoch 10/50
125/125 [=====] - 1s 6ms/step - loss: 1.2422 - accuracy: 0.6357 - val_loss: 1.1890 - val_accuracy: 0.6718
Epoch 11/50

conda (Python 3.10.9) Completions: conda LSP: Python Line 21, Col 59 ASCII CRUF RW Mem 89%

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Pallavi Nile\Downloads\DL\pract_2\DL2.py

#Nile Pallavi Roll NO: 4217 Div:B
# DL Practical NO.2
import pandas as pd
import numpy as np

df = pd.read_csv('C:\Users\Pallavi Nile\Downloads\DL\pract_2\letter-recognition.csv')
print(df)
columns = ["letter", "x-box", "y-box", "width", "height", "onpix", "x-bar",
           "y-bar", "x2bar", "y2bar", "xybar", "x2ybr", "xy2br", "x-egs", "xegvy",
           "y-egs", "yegvy"]
x = df.drop("letter", axis=1).values
y = df["letter"].values
print("shape of x")
print(x.shape)
print("shape of y")
print(y.shape)
print(np.unique(y))

# Split the data into training and testing sets
test_size = 0.2 # Percentage of data to be used for testing
num_test_samples = int(test_size * len(df))

# Shuffle the indices of the data randomly
shuffled_indices = np.random.permutation(len(df))

# Split the indices into training and test sets
test_indices = shuffled_indices[num_test_samples:]
train_indices = shuffled_indices[:num_test_samples]

# Split the features and target variable based on the indices
x_train = x[train_indices]
x_test = x[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

print("Shape of x_train:", x_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

print(x_train[0])
print(y_train[0])

# Save the data
np.save('x_train.npy', x_train)
np.save('y_train.npy', y_train)
np.save('x_test.npy', x_test)
np.save('y_test.npy', y_test)
```

Source Console Object

Console 2/A X

In [1]: runfile('C:/Users/Pallavi Nile/Downloads/DL/pract_2/DL2.py', wdir='C:/Users/Pallavi Nile/Downloads/DL/pract_2')

letter	xbox	ybox	width	xedge	xedgey	yedge	yedgex
0	1	5	12	3	2	8	4
2	0	4	11	6	3	7	3
3	0	7	11	6	6	10	2
4	6	2	1	3	1	7	5

...
19995 0 2 2 3 ... 2 8 3 7
19996 C 7 10 8 ... 2 9 3 7
19997 T 6 9 6 ... 2 12 2 4
19998 S 2 3 4 ... 1 9 5 8
19999 A 4 9 6 ... 2 7 2 8

[20000 rows x 17 columns]
shape of x
(20000, 16)
shape of y
(20000,)
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z']
Shape of x_train: (16000, 16)
Shape of x_test: (4000, 16)
Shape of y_train: (16000,)
Shape of y_test: (4000,)
[4 6 5 5 7 7 4 6 5 8 6 8 4 6]
N
[1 3 1 2 0 7 7 1 8 7 6 8 0 8 3 8]
I
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	8704
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0

conda (Python 3.10.9) Completions: conda LSP: Python Line 21, Col 59 ASCII CRUF RW Mem 89%