dl-2-a

April 11, 2025

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
import matplotlib.pyplot as plt
```

[2]:
```
# Load the OCR letter recognition dataset
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪letter-recognition/letter-recognition.data'
dataset = pd.read_csv(url, header=None)
```

[3]:
```
# Split the dataset into features and labels
X = dataset.iloc[:, 1:].values  #selecting all rows and selecting all columns␣
 ↪from index 1
y = dataset.iloc[:, 0].values   #selecting all rows and selecting column with␣
 ↪index 0
```

[4]:
```
print(y[0])
```

```
T
```

[5]:
```
# Encode the labels into numeric value
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

[6]:
```
print(y[0])
```

```
19
```

[7]:
```
#splitting dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

[8]:
```
X_train = X_train / 15.0
X_test = X_test / 15.0
```

[9]:
```
#we are using sequential model where layers are stacked one after another,
#output of previous layer is given to as input to next layer

model = Sequential()
#1st layer is dense layer which consists on 128 neurons, since it is 1st layer␣
↪we need to define input_shape of our training data
model.add(Dense(128, activation='relu', input_shape=(16,)))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(26, activation='softmax'))  #softmax is used to predict␣
↪multiclass category outcome
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[10]:
```
#now we will compile the model

#sparse_categorical_crossentropy (scce) produces a category index of the most␣
↪likely matching category.
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics␣
↪=['accuracy'])
```

[11]:
```
#The batch size is a number of samples processed before the model is updated.
#verbose is the choice that how you want to see the output of your Nural␣
↪Network while it's training.
#If you set verbose = 0, It will show nothing
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=50, batch_size=12, verbose=1)
```

```
Epoch 1/50
1334/1334            6s 3ms/step -
accuracy: 0.1281 - loss: 3.0017 - val_accuracy: 0.5490 - val_loss: 1.8102
Epoch 2/50
1334/1334            4s 3ms/step -
accuracy: 0.3847 - loss: 1.9903 - val_accuracy: 0.6245 - val_loss: 1.3941
Epoch 3/50
1334/1334            5s 3ms/step -
accuracy: 0.4656 - loss: 1.6908 - val_accuracy: 0.6622 - val_loss: 1.2217
Epoch 4/50
1334/1334            5s 3ms/step -
accuracy: 0.5273 - loss: 1.5218 - val_accuracy: 0.6935 - val_loss: 1.1374
Epoch 5/50
1334/1334            4s 3ms/step -
```

```
accuracy: 0.5480 - loss: 1.4353 - val_accuracy: 0.7038 - val_loss: 1.0333
Epoch 6/50
1334/1334            4s 3ms/step -
accuracy: 0.5759 - loss: 1.3670 - val_accuracy: 0.7268 - val_loss: 0.9855
Epoch 7/50
1334/1334            4s 3ms/step -
accuracy: 0.5857 - loss: 1.2885 - val_accuracy: 0.7305 - val_loss: 0.9448
Epoch 8/50
1334/1334            4s 3ms/step -
accuracy: 0.5978 - loss: 1.2665 - val_accuracy: 0.7377 - val_loss: 0.9137
Epoch 9/50
1334/1334            5s 2ms/step -
accuracy: 0.6079 - loss: 1.2320 - val_accuracy: 0.7525 - val_loss: 0.8883
Epoch 10/50
1334/1334            5s 3ms/step -
accuracy: 0.6203 - loss: 1.2210 - val_accuracy: 0.7567 - val_loss: 0.8472
Epoch 11/50
1334/1334            6s 3ms/step -
accuracy: 0.6357 - loss: 1.1595 - val_accuracy: 0.7520 - val_loss: 0.8433
Epoch 12/50
1334/1334            3s 3ms/step -
accuracy: 0.6375 - loss: 1.1361 - val_accuracy: 0.7670 - val_loss: 0.8372
Epoch 13/50
1334/1334            3s 3ms/step -
accuracy: 0.6402 - loss: 1.1240 - val_accuracy: 0.7598 - val_loss: 0.8064
Epoch 14/50
1334/1334            4s 3ms/step -
accuracy: 0.6453 - loss: 1.1119 - val_accuracy: 0.7685 - val_loss: 0.7946
Epoch 15/50
1334/1334            4s 3ms/step -
accuracy: 0.6517 - loss: 1.0868 - val_accuracy: 0.7713 - val_loss: 0.7720
Epoch 16/50
1334/1334            4s 3ms/step -
accuracy: 0.6531 - loss: 1.0886 - val_accuracy: 0.7717 - val_loss: 0.7553
Epoch 17/50
1334/1334            6s 4ms/step -
accuracy: 0.6504 - loss: 1.0941 - val_accuracy: 0.7790 - val_loss: 0.7469
Epoch 18/50
1334/1334            4s 3ms/step -
accuracy: 0.6671 - loss: 1.0471 - val_accuracy: 0.7840 - val_loss: 0.7230
Epoch 19/50
1334/1334            4s 3ms/step -
accuracy: 0.6802 - loss: 1.0294 - val_accuracy: 0.7893 - val_loss: 0.7095
Epoch 20/50
1334/1334            5s 3ms/step -
accuracy: 0.6745 - loss: 1.0373 - val_accuracy: 0.7828 - val_loss: 0.7071
Epoch 21/50
1334/1334            5s 3ms/step -
```

```
accuracy: 0.6817 - loss: 1.0046 - val_accuracy: 0.7847 - val_loss: 0.6866
Epoch 22/50
1334/1334            5s 4ms/step -
accuracy: 0.6695 - loss: 1.0382 - val_accuracy: 0.7955 - val_loss: 0.6721
Epoch 23/50
1334/1334            4s 3ms/step -
accuracy: 0.6817 - loss: 1.0013 - val_accuracy: 0.7900 - val_loss: 0.6957
Epoch 24/50
1334/1334            5s 3ms/step -
accuracy: 0.6780 - loss: 0.9947 - val_accuracy: 0.7845 - val_loss: 0.6882
Epoch 25/50
1334/1334            5s 3ms/step -
accuracy: 0.6904 - loss: 0.9882 - val_accuracy: 0.7947 - val_loss: 0.6641
Epoch 26/50
1334/1334            4s 3ms/step -
accuracy: 0.6816 - loss: 0.9994 - val_accuracy: 0.8020 - val_loss: 0.6678
Epoch 27/50
1334/1334            5s 3ms/step -
accuracy: 0.6853 - loss: 0.9876 - val_accuracy: 0.7995 - val_loss: 0.6530
Epoch 28/50
1334/1334            5s 3ms/step -
accuracy: 0.6857 - loss: 0.9841 - val_accuracy: 0.8073 - val_loss: 0.6398
Epoch 29/50
1334/1334            5s 3ms/step -
accuracy: 0.6928 - loss: 0.9665 - val_accuracy: 0.8102 - val_loss: 0.6257
Epoch 30/50
1334/1334            6s 4ms/step -
accuracy: 0.6945 - loss: 0.9644 - val_accuracy: 0.8062 - val_loss: 0.6439
Epoch 31/50
1334/1334            4s 3ms/step -
accuracy: 0.6972 - loss: 0.9446 - val_accuracy: 0.8142 - val_loss: 0.6250
Epoch 32/50
1334/1334            6s 3ms/step -
accuracy: 0.6989 - loss: 0.9373 - val_accuracy: 0.8150 - val_loss: 0.6132
Epoch 33/50
1334/1334            4s 3ms/step -
accuracy: 0.6943 - loss: 0.9576 - val_accuracy: 0.8158 - val_loss: 0.6171
Epoch 34/50
1334/1334            4s 3ms/step -
accuracy: 0.7025 - loss: 0.9320 - val_accuracy: 0.8213 - val_loss: 0.6029
Epoch 35/50
1334/1334            5s 3ms/step -
accuracy: 0.7069 - loss: 0.9335 - val_accuracy: 0.8163 - val_loss: 0.6076
Epoch 36/50
1334/1334            4s 3ms/step -
accuracy: 0.7046 - loss: 0.9377 - val_accuracy: 0.8202 - val_loss: 0.6032
Epoch 37/50
1334/1334            5s 3ms/step -
```

```
accuracy: 0.7083 - loss: 0.9230 - val_accuracy: 0.8198 - val_loss: 0.5952
Epoch 38/50
1334/1334          6s 3ms/step -
accuracy: 0.7026 - loss: 0.9267 - val_accuracy: 0.8205 - val_loss: 0.5997
Epoch 39/50
1334/1334          4s 3ms/step -
accuracy: 0.7079 - loss: 0.9221 - val_accuracy: 0.8240 - val_loss: 0.5826
Epoch 40/50
1334/1334          6s 3ms/step -
accuracy: 0.7035 - loss: 0.9218 - val_accuracy: 0.8235 - val_loss: 0.5871
Epoch 41/50
1334/1334          5s 3ms/step -
accuracy: 0.7144 - loss: 0.9084 - val_accuracy: 0.8117 - val_loss: 0.5959
Epoch 42/50
1334/1334          5s 3ms/step -
accuracy: 0.7104 - loss: 0.9101 - val_accuracy: 0.8300 - val_loss: 0.5705
Epoch 43/50
1334/1334          5s 4ms/step -
accuracy: 0.7120 - loss: 0.8986 - val_accuracy: 0.8263 - val_loss: 0.5650
Epoch 44/50
1334/1334          4s 3ms/step -
accuracy: 0.7118 - loss: 0.9032 - val_accuracy: 0.8180 - val_loss: 0.5875
Epoch 45/50
1334/1334          5s 3ms/step -
accuracy: 0.7151 - loss: 0.9061 - val_accuracy: 0.8225 - val_loss: 0.5884
Epoch 46/50
1334/1334          5s 4ms/step -
accuracy: 0.7194 - loss: 0.8935 - val_accuracy: 0.8325 - val_loss: 0.5634
Epoch 47/50
1334/1334          4s 3ms/step -
accuracy: 0.7073 - loss: 0.9143 - val_accuracy: 0.8342 - val_loss: 0.5537
Epoch 48/50
1334/1334          6s 3ms/step -
accuracy: 0.7090 - loss: 0.9016 - val_accuracy: 0.8315 - val_loss: 0.5652
Epoch 49/50
1334/1334          4s 3ms/step -
accuracy: 0.7127 - loss: 0.9062 - val_accuracy: 0.8273 - val_loss: 0.5592
Epoch 50/50
1334/1334          5s 3ms/step -
accuracy: 0.7150 - loss: 0.8877 - val_accuracy: 0.8325 - val_loss: 0.5613
```

[12]:
```python
loss, accuracy = model.evaluate(X_test, y_test)
print("Test accuracy:", accuracy)
print("Test loss:", loss)
```

```
125/125          1s 10ms/step -
accuracy: 0.8274 - loss: 0.5700
Test accuracy: 0.8324999809265137
```

```
Test loss: 0.5613325834274292
```

[13]:
```python
model.save('ocr_model.h5')
# Save the trained model
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

[14]:
```python
from tensorflow.keras.models import load_model
model = load_model('ocr_model.h5')
# Load the trained model
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

[15]:
```python
sample_records = X_test[:1000]
# Select a few records for classification
```

[16]:
```python
# Perform classification
predictions = model.predict(sample_records)
```

```
32/32              0s 11ms/step
```

[17]:
```python
predicted_labels = np.argmax(predictions, axis=1)
predicted_letters = label_encoder.inverse_transform(predicted_labels)
actual_letters = label_encoder.inverse_transform(y_test)
```

[18]:
```python
# Calculate accuracy
accuracy = np.sum(predicted_labels == y[:1000]) / len(predicted_labels)
```

[19]:
```python
# Print the predicted labels and corresponding actual labels
print("Predicted Labels\tActual Labels")
for i in range(len(predicted_letters)):
    print(f"{predicted_letters[i]}\t\t\t{actual_letters[i]}")
```

```
Predicted Labels        Actual Labels
D                       D
D                       D
V                       V
B                       B
H                       H
N                       N
R                       E
Q                       Q
```

E	R
N	N
Q	Q
O	O
N	N
B	D
F	I
M	M
U	U
S	S
O	O
A	A
X	X
A	A
K	K
S	S
O	O
V	V
Y	Y
J	J
D	D
V	V
N	D
Y	V
K	K
T	F
N	N
I	I
G	G
T	T
H	H
K	K
J	J
O	O
M	M
T	T
B	B
Q	Q
Z	Z
C	C
D	D
X	X
E	C
G	G
Q	Q
F	F
N	N
Y	Y

Z	Z
K	K
S	C
K	T
M	M
W	V
Q	G
R	M
B	D
T	T
H	H
P	P
N	N
C	C
O	O
X	X
D	X
Q	Q
V	V
F	P
V	V
G	G
W	W
H	H
P	P
O	O
I	I
G	G
X	X
Y	X
Y	P
W	W
O	Q
D	D
B	N
Q	Q
T	T
T	T
V	V
Y	Y
T	T
V	V
J	J
M	U
D	D
L	L
M	M
Y	Y

E	E
M	M
A	A
Y	Y
K	K
Y	Y
U	U
U	U
U	U
E	C
Q	Q
I	I
A	A
E	E
B	D
T	T
B	P
Q	Q
Z	Z
G	G
X	X
G	G
F	F
M	M
S	S
E	E
E	E
K	K
Z	Z
X	X
O	O
E	E
U	U
I	I
R	R
O	O
M	M
J	J
F	F
K	K
M	M
N	N
G	T
D	D
H	H
R	R
L	L
Z	Z

F	F
V	V
W	W
E	E
T	T
W	W
Y	Y
Q	Q
L	L
F	F
O	O
D	D
K	K
S	I
T	X
P	P
H	H
E	Z
D	H
W	W
T	F
B	B
R	R
O	O
B	B
S	Z
N	N
W	W
Z	Z
W	W
V	V
B	S
V	Y
H	H
O	O
T	Y
V	V
S	S
W	W
B	B
P	P
D	D
Z	Z
O	O
X	X
F	P
B	B
S	S

T	T
X	X
J	J
B	B
Y	Y
S	I
Z	Z
Z	Z
P	P
E	E
R	H
Q	Q
P	I
A	A
H	H
P	P
E	E
U	U
K	K
P	P
M	M
C	C
X	Y
J	J
W	W
M	M
G	G
J	J
O	O
E	H
P	P
L	L
W	W
A	A
C	C
C	C
C	C
O	O
J	J
Y	Y
F	P
Y	Y
A	A
R	R
Z	Z
J	J
I	I
G	G

G	G
I	I
N	K
R	M
R	R
T	T
D	D
E	E
D	D
L	L
W	W
G	P
L	L
U	U
O	O
T	T
A	A
V	V
D	D
U	U
X	X
X	X
G	G
C	C
D	F
Z	Z
W	W
T	T
V	V
T	T
P	P
M	M
S	S
U	U
D	F
Q	Q
G	B
Q	G
K	K
L	L
F	F
E	E
R	R
P	P
R	R
W	W
I	I
Y	Y

Y	Y
B	B
C	C
T	T
M	M
M	M
D	N
B	B
R	R
Z	Z
G	G
D	D
R	R
I	I
Y	Y
F	P
M	M
A	A
B	B
M	M
R	F
U	G
Q	Q
Z	Z
P	P
O	O
Q	Q
P	P
S	G
N	N
O	O
T	T
G	G
S	S
V	V
Z	Z
A	A
P	P
J	J
O	O
N	N
Q	Q
K	K
X	X
S	E
F	P
J	J
N	N

X	X
X	X
V	V
Q	Q
Y	H
A	A
T	T
S	S
W	W
M	M
P	P
Y	Y
M	U
S	Z
R	R
B	B
A	A
Y	Y
F	F
F	F
B	D
A	A
N	N
A	A
H	H
Q	Q
P	P
Z	Z
R	R
D	D
W	W
R	O
Q	Q
K	K
N	N
W	W
P	P
W	W
S	S
K	K
B	S
I	J
E	E
S	S
V	V
S	S
O	O
V	V

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

B	V
G	G
U	U
Z	Z
H	H
O	O
H	N
S	J
G	T
O	O
H	H
V	V
F	F
L	L
W	W
M	M
T	Y
J	J
P	P
W	W
E	E
A	A
B	B
W	N
O	O
Q	Q
Y	Y
Q	Q
F	F
A	A
A	A
Q	Q
Z	Z
E	E
K	H
N	N
E	E
A	A
R	R
G	C
Z	Z
E	Q
V	V
O	O
B	T
K	K
M	M
R	R

Y	Y
X	X
X	X
J	J
X	X
Q	Q
T	T
S	S
A	A
X	X
G	G
T	T
N	N
B	S
Y	Y
P	P
A	A
S	I
T	T
O	X
X	X
C	C
O	O
B	B
G	G
R	W
D	D
K	K
U	U
S	S
W	O
Z	Z
S	L
N	N
R	R
V	V
A	A
K	L
B	D
W	W
I	I
B	O
K	C
W	W
A	A
Y	Y
Q	Q
L	L

Q	Q
K	K
J	J
E	E
B	B
P	P
J	J
R	R
X	X
N	N
A	A
G	G
C	C
D	T
L	L
S	S
A	A
N	N
B	K
V	V
E	G
A	A
D	D
H	H
Y	Y
Z	Z
Q	Q
C	C
U	U
Q	Q
Z	Z
M	M
D	D
B	B
L	L
B	R
R	Q
D	D
F	F
W	W
Q	Q
M	M
G	G
Z	Z
B	B
U	U
N	N
K	K

Q	P
A	A
U	U
F	F
T	T
X	X
L	L
U	U
Y	Y
P	P
Y	Y
B	R
S	S
K	K
Q	E
Q	Q
F	F
W	W
Q	P
P	P
M	M
N	N
Z	Z
G	G
B	K
O	O
W	W
I	I
F	C
E	E
U	U
R	R
F	I
Y	Y
J	S
V	V
R	R
C	C
Q	Q
C	C
S	J
Y	Y
S	S
K	K
Q	L
X	X
T	T
I	I

D	F
W	W
A	A
O	D
K	K
A	A
Z	Z
M	M
G	G
G	G
W	W
W	W
Z	Z
W	W
T	T
O	O
H	R
T	T
N	N
S	S
Q	Q
J	J
Q	Q
M	M
O	O
A	A
N	N
Z	Z
B	D
T	T
S	I
U	U
A	A
O	N
H	G
Q	P
L	L
I	F
R	R
C	C
P	P
Y	Y
Y	Y
T	T
A	A
S	Y
T	T
Q	E

T	T
O	O
S	S
N	N
O	O
I	I
O	O
U	U
N	N
Z	Z
J	J
C	C
N	H
S	S
G	L
H	H
E	C
Q	Q
F	F
B	B
W	O
I	I
K	K
O	O
B	R
E	E
L	L
M	M
L	L
X	T
E	E
U	U
W	W
E	F
D	D
N	N
V	V
M	M
I	I
K	K
L	L
T	Y
Q	Q
F	F
U	U
J	J
V	V
M	M

P	P
W	W
L	L
A	A
L	L
B	B
Y	Y
I	I
D	D
R	R
T	T
Q	Q
B	B
O	U
M	M
O	O
I	I
U	U
U	U
X	K
S	J
I	I
W	W
Z	Z
Z	Z
P	P
A	A
Z	Z
P	P
V	V
B	G
B	B
G	G
O	O
V	V
D	D
Z	Z
Y	Y
P	P
Q	Q
A	A
E	E
W	W
D	D
J	J
I	I
J	J
X	X

F	S
Z	Z
G	G
X	T
U	U
K	K
G	G
I	I
N	N
S	I
W	W
I	I
I	I
P	P
B	B
O	O
H	H
F	D
O	O
N	N
Z	S
D	D
S	S
M	M
O	O
H	N
Y	Y
S	X
Q	Q
M	M
R	R
G	G
G	G
A	A
B	K
U	U
O	O
B	P
C	C
V	V
B	D
B	B
A	A
E	E
B	R
T	T
I	I
S	Z

N	N
O	O
B	B
C	C
G	G
J	J
E	E
N	N
J	J
Q	H
B	P
O	O
O	O
B	B
E	X
D	D
S	S
O	O
O	O
G	G
B	B
R	H
C	C
A	A
W	W
P	P
I	I
N	N
M	M
K	G
H	N
A	A
Y	H
U	K
N	N
M	M
F	F
W	W
A	A
S	S
E	E
B	T
J	J
H	H
N	N
R	K
W	W
W	W

U	U
S	L
Z	Z
Z	Z
Z	Z
G	G
T	T
J	J
V	V
W	W
L	L
M	M
U	U
W	W
O	G
A	A
T	T
J	J
C	C
G	G
D	D
T	Y
Y	Y
X	X
N	N
Y	Y
Z	S
K	K
U	U
U	W
R	R
R	B
T	T
R	R
H	H
R	R
M	M
O	O
W	W
X	X
R	R
X	X
N	N
H	H
C	C
E	C
J	J
V	V

V	V
I	I
R	R
W	W
V	H
N	N
E	E
U	U
A	M
A	A
B	B
E	E
K	K
T	T
F	F
Y	Y
D	T
H	H
W	W
W	W
O	O
Q	G
W	W
S	L
Q	Q
B	R
K	K
B	D
P	P
T	T
B	B
E	L
V	V
Q	Q
G	G
B	S
O	W
V	V
L	L
D	D
R	R
L	L
C	C
N	N
V	V
I	I
P	P
K	C

```
F          F
W          W
A          A
P          P
F          T
H          H
B          B
K          K
I          I
W          W
R          R
J          J
R          R
K          K
A          A
Q          Q
Q          Q
A          A
M          M
V          V
U          U
G          G
D          O
U          U
H          H
V          V
G          G
Q          Q
I          I
T          T
C          G
F          I
```

```
[ ]:
```