Deep Learning Lab Assignment 02

Name of Student: Aniket Yashvant Sandbhor
Branch: Computer Engineering
Exam number: B190424404

0.1 Implement Classification using Artificial Neural Network

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
df=pd.read_csv("IMDB Dataset.csv")
df1=df.head(10)
df1
```

[3]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
```

```
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

[5]:
```
df.isnull().sum()
```

[5]:
```
review       0
sentiment    0
dtype: int64
```

[6]:
```
df.sentiment.value_counts()
```

[6]:
```
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

[7]:
```
df.review.value_counts().head(2)
```

[7]:
```
review
Loved today's show!!! It was a variety and not solely cooking (which
 ↪would have
been great too). Very stimulating and captivating, always keeping the
 ↪viewer
peeking around the corner to see what was coming up next. She is as
 ↪down to
earth and as personable as you get, like one of us which made the
 ↪show all the
more enjoyable. Special guests, who are friends as well made for a
 ↪nice surprise
too. Loved the 'first' theme and that the audience was invited to
 ↪play along
too. I must admit I was shocked to see her come in under her time
 ↪limits on a
few things, but she did it and by golly I'll be writing those recipes
 ↪down.
Saving time in the kitchen means more time with family. Those who
 ↪haven't tuned
in yet, find out what channel and the time, I assure you that you
 ↪won't be
disappointed.
5
```

```
Hilarious, clean, light-hearted, and quote-worthy. What else can you
 ↪ask for in
a film? This is my all-time, number one favorite movie. Ever since I
 ↪was a
little girl, I've dreamed of owning a blue van with flames and an
 ↪observation
bubble.<br /><br />The cliché characters in ridiculous situations are
 ↪what make
this film such great fun. The wonderful comedic chemistry between
 ↪Stephen Furst
(Harold) and Andy Tennant (Melio) make up most of my favorite parts
 ↪of the
movie. And who didn't love the hopeless awkwardness of Flynch? Don't
 ↪forget the
airport antics of Leon's cronies, dressed up as Hari Krishnas:
 ↪dancing, chanting
and playing the tambourine--unbeatable! The clues are genius, the
 ↪locations are
classic, and the plot is timeless.<br /><br />A word to the wise, if
 ↪you didn't
watch this film when you were little, it probably won't win a place
 ↪in your
heart today. But nevertheless give it a chance, you may find that "It
 ↪doesn't
matter what you say, it doesn't matter what you do, you've gotta play.
 ↪"    4
Name: count, dtype: int64
```

[8]:
```
# checking how many duplicate valu there are?
df.duplicated().value_counts()
```

[8]:
```
False    49582
True       418
Name: count, dtype: int64
```

[9]:
```
data=df.sample(10000)
data
```

[9]:

	review	sentiment
23564	*Warning! Some spoilers!* Matt, a r...	positive
16640	Here's another Antonioni that will be rediscov...	positive
592	As is often the case when you attempt to take ...	positive
33590	I'm a collector of films starring Ms. Weaver, ...	negative
23132	This film is definetly Fonda's best film. The ...	positive
...	...	...
20223	Average viewers looking for any sense of inter...	negative
37206	This move reminded my of Tales from the Crypt ...	negative

```
13248  Thinking that it could only get better was the...  negative
27156  This is awful, you just could't believe it. Th...  negative
3199   Dramatic license - some hate it, though it is ...  positive

[10000 rows x 2 columns]
```

[10]:
```
data.drop_duplicates(inplace=True)
```

[11]:
```
data.duplicated().value_counts()
```

[11]:
```
False    9983
Name: count, dtype: int64
```

[12]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in /opt/anaconda3/lib/python3.11/
 ↪site-
packages (3.8.1)
Requirement already satisfied: click in /opt/anaconda3/lib/python3.11/
 ↪site-
packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.11/
 ↪site-
packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in
/opt/anaconda3/lib/python3.11/site-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.11/
 ↪site-
packages (from nltk) (4.65.0)
Note: you may need to restart the kernel to use updated packages.
```

[13]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from bs4 import BeautifulSoup
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/tejasmote/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

[13]:
```
True
```

[14]:
```
# function to clean whole text
def clean_review(review, stemmer = PorterStemmer(), stop_words =
 ↪set(stopwords.words("english"))):
    #removing html tags from reviews
```

```
    soup = BeautifulSoup(review, "html.parser")
    no_html_review = soup.get_text().lower()

    # empty list for adding clean words
    clean_text = []
    # cleaning stopwords and not alpha characters
    for word in review.split():
        if word not in stop_words and word.isalpha():
            clean_text.append(stemmer.stem(word))

    return " ".join(clean_text)
```

[15]:
```
data.review = data.review.apply(clean_review)
```

[16]:
```
#checking the clean review in specific locaion
data.review.iloc[3537]
```

[16]:
```
'i gave receiv low definit realli kind shallow realli hot babe i like
 ↪eclect
kill peopl way sometim get'
```

[17]:
```
#how the data looks like now
data
```

[17]:

	review	sentiment
23564	some rich fact still boy he care need care les...	positive
16640	anoth antonioni rediscov soon come tape i saw ...	positive
592	as often case attempt take plu page book cram ...	positive
33590	collector film star i bought i find realli odd...	negative
23132	thi film definetli best the plot act direct an...	positive
...	...	...
20223	averag viewer look sens intern coher film prob...	negative
37206	thi move remind tale crypt it sort idea peopl ...	negative
13248	think could get better worst assumpt i ever de...	negative
27156	thi believ the score film sometim see shadow c...	negative
3199	dramat licens hate though necessari retel life...	positive

[9983 rows x 2 columns]

[18]:
```
# verctorizing reviews
#import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# setting max_features to 5000 to get most repeated 5000 words in
 ↪reviews
cv = CountVectorizer(max_features=300,ngram_range=(1,4))
```

[19]:
```
# Fitting countvectorizer in data.review and getting X for ML
X = cv.fit_transform(data.review).toarray()

x1 = pd.DataFrame(X, columns = cv.get_feature_names_out())
x1
```

[19]:

	absolut	act	action	actor	actual	all	almost	along	also	although
0	2	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	1	0	2	0
3	0	0	0	0	0	0	0	0	1	0
4	0	1	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...
9978	0	0	0	0	0	0	0	0	0	0
9979	0	0	0	0	0	0	0	0	0	0
9980	0	1	0	1	0	0	0	0	0	0
9981	0	0	0	1	0	0	0	0	0	0
9982	0	0	0	3	0	0	0	0	2	0

	...	work	world	worst	worth	would	write	year	yet	you	young
0	...	0	0	0	0	2	0	0	0	1	0
1	...	0	0	0	0	1	0	2	0	0	0
2	...	1	0	0	2	0	0	0	0	0	0
3	...	1	0	0	2	0	0	1	0	0	0
4	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...

```
9978  ...     0      0      0      0      0      0      0      0      0
 ↪  0
9979  ...     0      0      0      0      0      0      0      0      0
 ↪  0
9980  ...     1      0      1      0      1      0      0      0      0
 ↪  0
9981  ...     0      0      1      0      1      0      0      0      0
 ↪  0
9982  ...     2      0      0      0      2      0      1      1      0
 ↪  0

[9983 rows x 300 columns]
```

[20]:
```
x1.shape
```

[20]:
```
(9983, 300)
```

[21]:
```
#Importing label encoder
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()

# positive = 1, negative = 0
data.sentiment = lb.fit_transform(data.sentiment)
```

[22]:
```
data.sentiment
```

[22]:
```
23564    1
16640    1
592      1
33590    0
23132    1
        ..
20223    0
37206    0
13248    0
27156    0
3199     1
Name: sentiment, Length: 9983, dtype: int64
```

[23]:
```
y=data.sentiment
```

[24]:
```
y.shape
```

[24]:
```
(9983,)
```

[25]:
```
from sklearn.model_selection import train_test_split

# converting X, y into train test split
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x1, y, test_size=0.2,
 ↪random_state=42)
```

[26]:
```
ytrain.shape
```

[26]:
```
(7986,)
```

[27]:
```
import tensorflow.keras as tk
```

[28]:
```
model = tk.Sequential()
model.add(tk.layers.Input(shape=(300,)))
model.add(tk.layers.Dense(50,
 ↪activation='relu',kernel_initializer="he_uniform"))
model.add(tk.layers.Dense(1,
 ↪activation='sigmoid',kernel_initializer="he_uniform"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                15050

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 15,101
Trainable params: 15,101
Non-trainable params: 0
_________________________________________________________________
```

[29]:
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy',
 ↪metrics = ['accuracy'])
```

[30]:
```
obj1=model.
 ↪fit(x=xtrain,y=ytrain,epochs=80,batch_size=64,validation_data=(xtest,ytest))
```

```
Epoch 1/80
  1/125 [...] - ETA: 18s - loss: 0.7049 - accuracy:
0.5938

2024-04-11 02:50:22.095715: W
tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get
 ↪CPU
frequency: 0 Hz

125/125 [==============================] - 0s 1ms/step - loss: 0.5537 -
accuracy: 0.7238 - val_loss: 0.4710 - val_accuracy: 0.7832
Epoch 2/80
```

```
125/125 [==============================] - 0s 624us/step - loss: 0.
 ↪4390 -
accuracy: 0.8070 - val_loss: 0.4533 - val_accuracy: 0.7812
Epoch 3/80
125/125 [==============================] - 0s 602us/step - loss: 0.
 ↪4120 -
accuracy: 0.8188 - val_loss: 0.4485 - val_accuracy: 0.7962
Epoch 4/80
125/125 [==============================] - 0s 608us/step - loss: 0.
 ↪3949 -
accuracy: 0.8253 - val_loss: 0.4562 - val_accuracy: 0.7922
Epoch 5/80
125/125 [==============================] - 0s 620us/step - loss: 0.
 ↪3787 -
accuracy: 0.8337 - val_loss: 0.4512 - val_accuracy: 0.7932
Epoch 6/80
125/125 [==============================] - 0s 618us/step - loss: 0.
 ↪3649 -
accuracy: 0.8403 - val_loss: 0.4523 - val_accuracy: 0.7957
Epoch 7/80
125/125 [==============================] - 0s 622us/step - loss: 0.
 ↪3489 -
accuracy: 0.8476 - val_loss: 0.4627 - val_accuracy: 0.7927
Epoch 8/80
125/125 [==============================] - 0s 626us/step - loss: 0.
 ↪3335 -
accuracy: 0.8586 - val_loss: 0.4667 - val_accuracy: 0.7882
Epoch 9/80
125/125 [==============================] - 0s 622us/step - loss: 0.
 ↪3172 -
accuracy: 0.8690 - val_loss: 0.4665 - val_accuracy: 0.7882
Epoch 10/80
125/125 [==============================] - 0s 614us/step - loss: 0.
 ↪3005 -
accuracy: 0.8779 - val_loss: 0.4734 - val_accuracy: 0.7892
Epoch 11/80
125/125 [==============================] - 0s 616us/step - loss: 0.
 ↪2839 -
accuracy: 0.8866 - val_loss: 0.4842 - val_accuracy: 0.7842
Epoch 12/80
125/125 [==============================] - 0s 619us/step - loss: 0.
 ↪2668 -
accuracy: 0.8968 - val_loss: 0.4862 - val_accuracy: 0.7817
Epoch 13/80
125/125 [==============================] - 0s 620us/step - loss: 0.
 ↪2501 -
accuracy: 0.9071 - val_loss: 0.4978 - val_accuracy: 0.7812
```

```
Epoch 14/80
125/125 [==============================] - 0s 615us/step - loss: 0.
 ↪2349 -
accuracy: 0.9135 - val_loss: 0.5072 - val_accuracy: 0.7782
Epoch 15/80
125/125 [==============================] - 0s 621us/step - loss: 0.
 ↪2192 -
accuracy: 0.9234 - val_loss: 0.5145 - val_accuracy: 0.7732
Epoch 16/80
125/125 [==============================] - 0s 776us/step - loss: 0.
 ↪2037 -
accuracy: 0.9294 - val_loss: 0.5288 - val_accuracy: 0.7727
Epoch 17/80
125/125 [==============================] - 0s 633us/step - loss: 0.
 ↪1894 -
accuracy: 0.9391 - val_loss: 0.5368 - val_accuracy: 0.7687
Epoch 18/80
125/125 [==============================] - 0s 628us/step - loss: 0.
 ↪1760 -
accuracy: 0.9448 - val_loss: 0.5480 - val_accuracy: 0.7621
Epoch 19/80
125/125 [==============================] - 0s 621us/step - loss: 0.
 ↪1643 -
accuracy: 0.9512 - val_loss: 0.5587 - val_accuracy: 0.7626
Epoch 20/80
125/125 [==============================] - 0s 619us/step - loss: 0.
 ↪1519 -
accuracy: 0.9591 - val_loss: 0.5712 - val_accuracy: 0.7566
Epoch 21/80
125/125 [==============================] - 0s 632us/step - loss: 0.
 ↪1403 -
accuracy: 0.9641 - val_loss: 0.5872 - val_accuracy: 0.7636
Epoch 22/80
125/125 [==============================] - 0s 632us/step - loss: 0.
 ↪1298 -
accuracy: 0.9688 - val_loss: 0.6019 - val_accuracy: 0.7621
Epoch 23/80
125/125 [==============================] - 0s 627us/step - loss: 0.
 ↪1199 -
accuracy: 0.9728 - val_loss: 0.6217 - val_accuracy: 0.7576
Epoch 24/80
125/125 [==============================] - 0s 645us/step - loss: 0.
 ↪1102 -
accuracy: 0.9762 - val_loss: 0.6260 - val_accuracy: 0.7621
Epoch 25/80
125/125 [==============================] - 0s 622us/step - loss: 0.
 ↪1021 -
```

```
accuracy: 0.9802 - val_loss: 0.6413 - val_accuracy: 0.7646
Epoch 26/80
125/125 [==============================] - 0s 626us/step - loss: 0.
 ↪0941 -
accuracy: 0.9831 - val_loss: 0.6612 - val_accuracy: 0.7591
Epoch 27/80
125/125 [==============================] - 0s 630us/step - loss: 0.
 ↪0872 -
accuracy: 0.9856 - val_loss: 0.6856 - val_accuracy: 0.7576
Epoch 28/80
125/125 [==============================] - 0s 623us/step - loss: 0.
 ↪0798 -
accuracy: 0.9887 - val_loss: 0.6939 - val_accuracy: 0.7571
Epoch 29/80
125/125 [==============================] - 0s 632us/step - loss: 0.
 ↪0731 -
accuracy: 0.9902 - val_loss: 0.7122 - val_accuracy: 0.7581
Epoch 30/80
125/125 [==============================] - 0s 625us/step - loss: 0.
 ↪0671 -
accuracy: 0.9916 - val_loss: 0.7344 - val_accuracy: 0.7546
Epoch 31/80
125/125 [==============================] - 0s 629us/step - loss: 0.
 ↪0616 -
accuracy: 0.9932 - val_loss: 0.7442 - val_accuracy: 0.7546
Epoch 32/80
125/125 [==============================] - 0s 627us/step - loss: 0.
 ↪0566 -
accuracy: 0.9942 - val_loss: 0.7611 - val_accuracy: 0.7566
Epoch 33/80
125/125 [==============================] - 0s 628us/step - loss: 0.
 ↪0518 -
accuracy: 0.9956 - val_loss: 0.7879 - val_accuracy: 0.7546
Epoch 34/80
125/125 [==============================] - 0s 768us/step - loss: 0.
 ↪0475 -
accuracy: 0.9961 - val_loss: 0.7963 - val_accuracy: 0.7571
Epoch 35/80
125/125 [==============================] - 0s 643us/step - loss: 0.
 ↪0439 -
accuracy: 0.9970 - val_loss: 0.8238 - val_accuracy: 0.7556
Epoch 36/80
125/125 [==============================] - 0s 633us/step - loss: 0.
 ↪0404 -
accuracy: 0.9972 - val_loss: 0.8411 - val_accuracy: 0.7471
Epoch 37/80
```

```
125/125 [==============================] - 0s 632us/step - loss: 0.
 ↪0370 -
accuracy: 0.9981 - val_loss: 0.8586 - val_accuracy: 0.7481
Epoch 38/80
125/125 [==============================] - 0s 627us/step - loss: 0.
 ↪0340 -
accuracy: 0.9979 - val_loss: 0.8649 - val_accuracy: 0.7586
Epoch 39/80
125/125 [==============================] - 0s 635us/step - loss: 0.
 ↪0311 -
accuracy: 0.9981 - val_loss: 0.8830 - val_accuracy: 0.7541
Epoch 40/80
125/125 [==============================] - 0s 646us/step - loss: 0.
 ↪0285 -
accuracy: 0.9986 - val_loss: 0.9181 - val_accuracy: 0.7531
Epoch 41/80
125/125 [==============================] - 0s 644us/step - loss: 0.
 ↪0262 -
accuracy: 0.9987 - val_loss: 0.9222 - val_accuracy: 0.7561
Epoch 42/80
125/125 [==============================] - 0s 639us/step - loss: 0.
 ↪0239 -
accuracy: 0.9990 - val_loss: 0.9388 - val_accuracy: 0.7516
Epoch 43/80
125/125 [==============================] - 0s 628us/step - loss: 0.
 ↪0221 -
accuracy: 0.9991 - val_loss: 0.9559 - val_accuracy: 0.7506
Epoch 44/80
125/125 [==============================] - 0s 637us/step - loss: 0.
 ↪0204 -
accuracy: 0.9994 - val_loss: 0.9785 - val_accuracy: 0.7526
Epoch 45/80
125/125 [==============================] - 0s 630us/step - loss: 0.
 ↪0185 -
accuracy: 0.9994 - val_loss: 0.9959 - val_accuracy: 0.7516
Epoch 46/80
125/125 [==============================] - 0s 641us/step - loss: 0.
 ↪0171 -
accuracy: 0.9997 - val_loss: 1.0145 - val_accuracy: 0.7496
Epoch 47/80
125/125 [==============================] - 0s 619us/step - loss: 0.
 ↪0157 -
accuracy: 0.9997 - val_loss: 1.0255 - val_accuracy: 0.7516
Epoch 48/80
125/125 [==============================] - 0s 633us/step - loss: 0.
 ↪0144 -
accuracy: 1.0000 - val_loss: 1.0466 - val_accuracy: 0.7491
```

```
Epoch 49/80
125/125 [==============================] - 0s 618us/step - loss: 0.
 ↪0133 -
accuracy: 1.0000 - val_loss: 1.0716 - val_accuracy: 0.7526
Epoch 50/80
125/125 [==============================] - 0s 629us/step - loss: 0.
 ↪0123 -
accuracy: 1.0000 - val_loss: 1.0863 - val_accuracy: 0.7446
Epoch 51/80
125/125 [==============================] - 0s 636us/step - loss: 0.
 ↪0112 -
accuracy: 1.0000 - val_loss: 1.1036 - val_accuracy: 0.7506
Epoch 52/80
125/125 [==============================] - 0s 632us/step - loss: 0.
 ↪0104 -
accuracy: 1.0000 - val_loss: 1.1210 - val_accuracy: 0.7486
Epoch 53/80
125/125 [==============================] - 0s 764us/step - loss: 0.
 ↪0095 -
accuracy: 1.0000 - val_loss: 1.1351 - val_accuracy: 0.7491
Epoch 54/80
125/125 [==============================] - 0s 631us/step - loss: 0.
 ↪0088 -
accuracy: 1.0000 - val_loss: 1.1487 - val_accuracy: 0.7496
Epoch 55/80
125/125 [==============================] - 0s 623us/step - loss: 0.
 ↪0081 -
accuracy: 1.0000 - val_loss: 1.1708 - val_accuracy: 0.7481
Epoch 56/80
125/125 [==============================] - 0s 618us/step - loss: 0.
 ↪0075 -
accuracy: 1.0000 - val_loss: 1.1882 - val_accuracy: 0.7456
Epoch 57/80
125/125 [==============================] - 0s 619us/step - loss: 0.
 ↪0069 -
accuracy: 1.0000 - val_loss: 1.2025 - val_accuracy: 0.7501
Epoch 58/80
125/125 [==============================] - 0s 614us/step - loss: 0.
 ↪0064 -
accuracy: 1.0000 - val_loss: 1.2257 - val_accuracy: 0.7511
Epoch 59/80
125/125 [==============================] - 0s 615us/step - loss: 0.
 ↪0059 -
accuracy: 1.0000 - val_loss: 1.2398 - val_accuracy: 0.7451
Epoch 60/80
125/125 [==============================] - 0s 606us/step - loss: 0.
 ↪0055 -
```

```
accuracy: 1.0000 - val_loss: 1.2546 - val_accuracy: 0.7471
Epoch 61/80
125/125 [==============================] - 0s 617us/step - loss: 0.
 ↪0050 -
accuracy: 1.0000 - val_loss: 1.2702 - val_accuracy: 0.7476
Epoch 62/80
125/125 [==============================] - 0s 609us/step - loss: 0.
 ↪0046 -
accuracy: 1.0000 - val_loss: 1.2875 - val_accuracy: 0.7481
Epoch 63/80
125/125 [==============================] - 0s 616us/step - loss: 0.
 ↪0043 -
accuracy: 1.0000 - val_loss: 1.3004 - val_accuracy: 0.7491
Epoch 64/80
125/125 [==============================] - 0s 619us/step - loss: 0.
 ↪0040 -
accuracy: 1.0000 - val_loss: 1.3178 - val_accuracy: 0.7491
Epoch 65/80
125/125 [==============================] - 0s 623us/step - loss: 0.
 ↪0037 -
accuracy: 1.0000 - val_loss: 1.3339 - val_accuracy: 0.7456
Epoch 66/80
125/125 [==============================] - 0s 651us/step - loss: 0.
 ↪0034 -
accuracy: 1.0000 - val_loss: 1.3604 - val_accuracy: 0.7456
Epoch 67/80
125/125 [==============================] - 0s 621us/step - loss: 0.
 ↪0032 -
accuracy: 1.0000 - val_loss: 1.3638 - val_accuracy: 0.7496
Epoch 68/80
125/125 [==============================] - 0s 622us/step - loss: 0.
 ↪0029 -
accuracy: 1.0000 - val_loss: 1.3835 - val_accuracy: 0.7501
Epoch 69/80
125/125 [==============================] - 0s 613us/step - loss: 0.
 ↪0027 -
accuracy: 1.0000 - val_loss: 1.4041 - val_accuracy: 0.7476
Epoch 70/80
125/125 [==============================] - 0s 624us/step - loss: 0.
 ↪0025 -
accuracy: 1.0000 - val_loss: 1.4194 - val_accuracy: 0.7481
Epoch 71/80
125/125 [==============================] - 0s 614us/step - loss: 0.
 ↪0023 -
accuracy: 1.0000 - val_loss: 1.4323 - val_accuracy: 0.7481
Epoch 72/80
```

```
125/125 [==============================] - 0s 616us/step - loss: 0.
 ↪0022 -
accuracy: 1.0000 - val_loss: 1.4550 - val_accuracy: 0.7491
Epoch 73/80
125/125 [==============================] - 0s 613us/step - loss: 0.
 ↪0020 -
accuracy: 1.0000 - val_loss: 1.4708 - val_accuracy: 0.7491
Epoch 74/80
125/125 [==============================] - 0s 624us/step - loss: 0.
 ↪0019 -
accuracy: 1.0000 - val_loss: 1.4798 - val_accuracy: 0.7491
Epoch 75/80
125/125 [==============================] - 0s 770us/step - loss: 0.
 ↪0017 -
accuracy: 1.0000 - val_loss: 1.5012 - val_accuracy: 0.7476
Epoch 76/80
125/125 [==============================] - 0s 610us/step - loss: 0.
 ↪0016 -
accuracy: 1.0000 - val_loss: 1.5078 - val_accuracy: 0.7456
Epoch 77/80
125/125 [==============================] - 0s 615us/step - loss: 0.
 ↪0015 -
accuracy: 1.0000 - val_loss: 1.5279 - val_accuracy: 0.7476
Epoch 78/80
125/125 [==============================] - 0s 620us/step - loss: 0.
 ↪0014 -
accuracy: 1.0000 - val_loss: 1.5494 - val_accuracy: 0.7471
Epoch 79/80
125/125 [==============================] - 0s 616us/step - loss: 0.
 ↪0013 -
accuracy: 1.0000 - val_loss: 1.5608 - val_accuracy: 0.7471
Epoch 80/80
125/125 [==============================] - 0s 629us/step - loss: 0.
 ↪0012 -
accuracy: 1.0000 - val_loss: 1.5837 - val_accuracy: 0.7461
```

[31]:
```
y_pred=model.predict(xtest)
```

```
63/63 [==============================] - 0s 303us/step
```

[32]:
```
y_pred
```

[32]:
```
array([[1.0000000e+00],
       [1.1438821e-08],
       [4.6890029e-01],
       ...,
       [4.7211042e-06],
       [2.5063330e-01],
```

```
       [3.8178583e-05]], dtype=float32)
```

[33]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(ytest,y_pred.round())
```

[34]:
```
accuracy
```

[34]:
```
0.7461191787681523
```