In [1]:
```
pip install --upgrade numpy

```

```
Requirement already satisfied: numpy in c:\users\anusha v\anaconda3\lib\si
te-packages (1.26.1)
Collecting numpy
  Using cached numpy-1.26.2-cp39-cp39-win_amd64.whl (15.8 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.1
    Can't uninstall 'numpy'. No files were found to uninstall.
Successfully installed numpy-1.26.2
Note: you may need to restart the kernel to use updated packages.

WARNING: Error parsing requirements for numpy: [Errno 2] No such file or d
irectory: 'c:\\users\\anusha v\\anaconda3\\lib\\site-packages\\numpy-1.26.
1.dist-info\\METADATA'
    WARNING: No metadata found in c:\users\anusha v\anaconda3\lib\site-pac
kages
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the follo
wing dependency conflicts.
daal4py 2021.6.0 requires daal==2021.4.0, which is not installed.
scipy 1.9.1 requires numpy<1.25.0,>=1.18.5, but you have numpy 1.26.2 whic
h is incompatible.
numba 0.55.1 requires numpy<1.22,>=1.18, but you have numpy 1.26.2 which i
s incompatible.
```

In [1]:
```
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
from tensorflow.keras import datasets, layers, models

```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [3]:
```
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
x_train.shape
```

```
A local file was found, but it seems to be incomplete or outdated because
the auto file hash does not match the original value of 6d958be074577803d1
2ecdefd02955f39262c83c16fe9348329d7fe0b5c001ce so we will re-download the
data.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.ta
r.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 [==============================] - 91s 1us/step
```

Out[3]:
```
(50000, 32, 32, 3)
```

In [98]:
```
x_test.shape
```

Out[98]:
```
(10000, 32, 32, 3)
```

In [99]:
```
1 x_train[0]
```

Out[99]:
```
array([[[0.00090734, 0.00095348, 0.00096886],
        [0.00066128, 0.00070742, 0.00069204],
        [0.00076894, 0.00073818, 0.00066128],
        ...,
        [0.00242983, 0.00202999, 0.0016609 ],
        [0.00233756, 0.00192234, 0.00156863],
        [0.00227605, 0.00190696, 0.00158401]],

       [[0.00024606, 0.00030757, 0.00030757],
        [0.        , 0.        , 0.        ],
        [0.00027682, 0.00012303, 0.        ],
        ...,
        [0.00189158, 0.00135333, 0.00084583],
        [0.00183007, 0.00127643, 0.00076894],
        [0.0018762 , 0.00133795, 0.00087659]],

       [[0.00038447, 0.00036909, 0.00032295],
        [0.00024606, 0.00010765, 0.        ],
        [0.00075356, 0.00041522, 0.00012303],
        ...,
        [0.00181469, 0.00129181, 0.00076894],
        [0.00184544, 0.00129181, 0.00076894],
        [0.00167628, 0.00112265, 0.00064591]],

       ...,

       [[0.00319877, 0.00261438, 0.00147636],
        [0.00309112, 0.00235294, 0.00052288],
        [0.00304498, 0.00247597, 0.00039985],
        ...,
        [0.00246059, 0.00204537, 0.00107651],
        [0.00086121, 0.00047674, 0.00010765],
        [0.00081507, 0.00052288, 0.00030757]],

       [[0.00276817, 0.00213764, 0.00147636],
        [0.00266052, 0.00189158, 0.00064591],
        [0.00286044, 0.00221453, 0.00046136],
        ...,
        [0.00282968, 0.00227605, 0.0014456 ],
        [0.00149173, 0.00095348, 0.00052288],
        [0.00127643, 0.00081507, 0.00052288]],

       [[0.00272203, 0.00221453, 0.00178393],
        [0.00258362, 0.00198385, 0.0014456 ],
        [0.00275279, 0.00218378, 0.00133795],
        ...,
        [0.0033218 , 0.00282968, 0.00215302],
        [0.00232218, 0.00181469, 0.00129181],
        [0.00189158, 0.00141484, 0.00110727]]])
```

In [10]:
```
1 y_train.shape
```

Out[10]:
```
(50000, 1)
```

In [11]:
```
1 y_train[15]
```

Out[11]:
```
array([9], dtype=uint8)
```

In [14]:
```
1 y_train = y_train.reshape(-1,)
2 y_train[:5]
```

Out[14]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

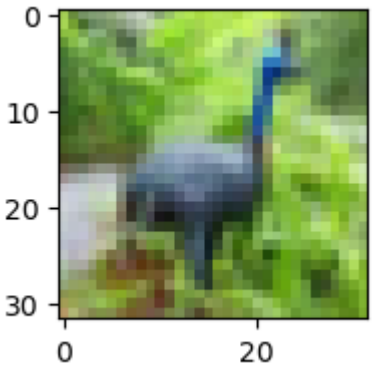
In [96]:
```
1 classes = ["airplane","autompbile","cat","dog","deer","frog","hourse","
```

In [19]:
```
1 classes[6]
```

Out[19]:
```
'hourse'
```

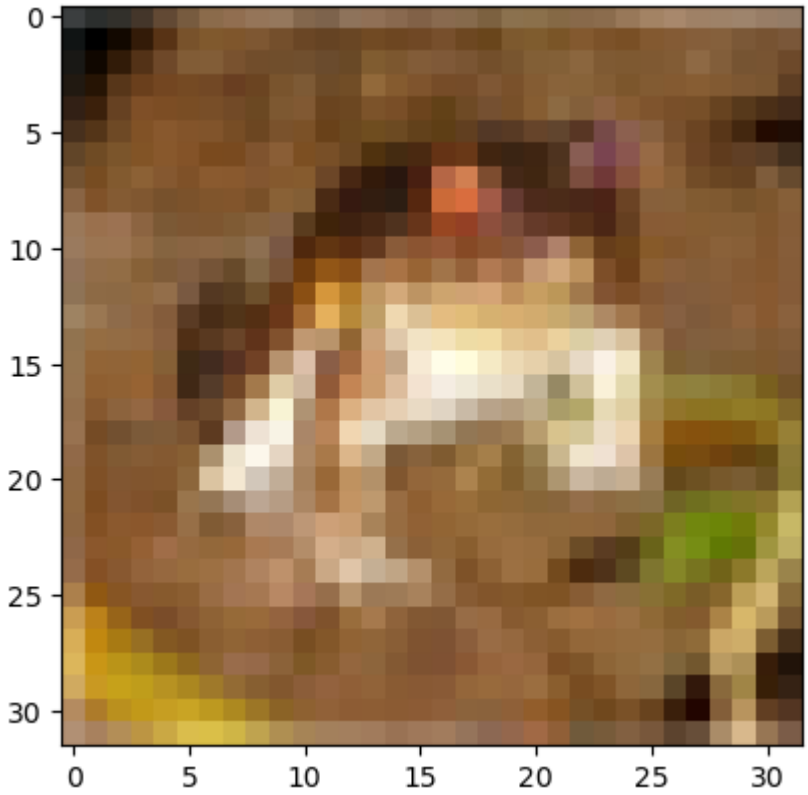
In [20]:
```
1 plt.figure(figsize = (15, 2))
2 plt.imshow(x_train[6])
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x17ae7fc5f10>
```



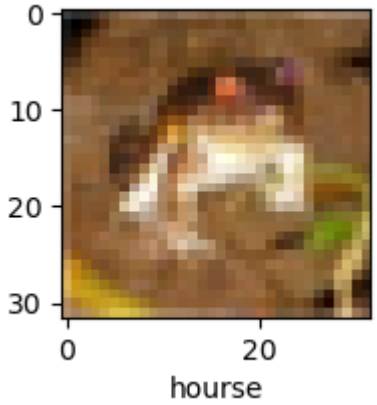
In [6]:
```
import matplotlib.pyplot as plt
plt.imshow(x_train[0])
```

Out[6]: <matplotlib.image.AxesImage at 0x17ae7bd2730>



In [21]:
```
def polt_sample(x, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(x[index])
    plt.xlabel(classes[y[index]])
```

In [23]:
```
polt_sample(x_train, y_train, 0)
```



In [25]:
```
1 x_train[0]
```

Out[25]:
```
array([[[ 59,  62,  63],
        [ 43,  46,  45],
        [ 50,  48,  43],
        ...,
        [158, 132, 108],
        [152, 125, 102],
        [148, 124, 103]],

       [[ 16,  20,  20],
        [  0,   0,   0],
        [ 18,   8,   0],
        ...,
        [123,  88,  55],
        [119,  83,  50],
        [122,  87,  57]],

       [[ 25,  24,  21],
        [ 16,   7,   0],
        [ 49,  27,   8],
        ...,
        [118,  84,  50],
        [120,  84,  50],
        [109,  73,  42]],

       ...,

       [[208, 170,  96],
        [201, 153,  34],
        [198, 161,  26],
        ...,
        [160, 133,  70],
        [ 56,  31,   7],
        [ 53,  34,  20]],

       [[180, 139,  96],
        [173, 123,  42],
        [186, 144,  30],
        ...,
        [184, 148,  94],
        [ 97,  62,  34],
        [ 83,  53,  34]],

       [[177, 144, 116],
        [168, 129,  94],
        [179, 142,  87],
        ...,
        [216, 184, 140],
        [151, 118,  84],
        [123,  92,  72]]], dtype=uint8)
```

In [26]:
```
1 x_train[0]/255
```

Out[26]:
```
array([[[0.23137255, 0.24313725, 0.24705882],
        [0.16862745, 0.18039216, 0.17647059],
        [0.19607843, 0.18823529, 0.16862745],
        ...,
        [0.61960784, 0.51764706, 0.42352941],
        [0.59607843, 0.49019608, 0.4       ],
        [0.58039216, 0.48627451, 0.40392157]],

       [[0.0627451 , 0.07843137, 0.07843137],
        [0.        , 0.        , 0.        ],
        [0.07058824, 0.03137255, 0.        ],
        ...,
        [0.48235294, 0.34509804, 0.21568627],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117647, 0.22352941]],

       [[0.09803922, 0.09411765, 0.08235294],
        [0.0627451 , 0.02745098, 0.        ],
        [0.19215686, 0.10588235, 0.03137255],
        ...,
        [0.4627451 , 0.32941176, 0.19607843],
        [0.47058824, 0.32941176, 0.19607843],
        [0.42745098, 0.28627451, 0.16470588]],

       ...,

       [[0.81568627, 0.66666667, 0.37647059],
        [0.78823529, 0.6       , 0.13333333],
        [0.77647059, 0.63137255, 0.10196078],
        ...,
        [0.62745098, 0.52156863, 0.2745098 ],
        [0.21960784, 0.12156863, 0.02745098],
        [0.20784314, 0.13333333, 0.07843137]],

       [[0.70588235, 0.54509804, 0.37647059],
        [0.67843137, 0.48235294, 0.16470588],
        [0.72941176, 0.56470588, 0.11764706],
        ...,
        [0.72156863, 0.58039216, 0.36862745],
        [0.38039216, 0.24313725, 0.13333333],
        [0.3254902 , 0.20784314, 0.13333333]],

       [[0.69411765, 0.56470588, 0.45490196],
        [0.65882353, 0.50588235, 0.36862745],
        [0.70196078, 0.55686275, 0.34117647],
        ...,
        [0.84705882, 0.72156863, 0.54901961],
        [0.59215686, 0.4627451 , 0.32941176],
        [0.48235294, 0.36078431, 0.28235294]]])
```

In [100]:
```
1 x_train = x_train / 255
2 x_test = x_test / 255
```

In [101]:
```
ann = models.Sequential([
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(3000, activation='relu'),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='sigmoid')
])
```

In [31]:
```
ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])
ann.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 186s 118ms/step - loss: 1.810
5 - accuracy: 0.3577
Epoch 2/5
1563/1563 [==============================] - 172s 110ms/step - loss: 1.622
1 - accuracy: 0.4289
Epoch 3/5
1563/1563 [==============================] - 172s 110ms/step - loss: 1.541
2 - accuracy: 0.4542
Epoch 4/5
1563/1563 [==============================] - 97027s 62s/step - loss: 1.480
6 - accuracy: 0.4763
Epoch 5/5
1563/1563 [==============================] - 214s 137ms/step - loss: 1.433
5 - accuracy: 0.4956
```

Out[31]:
```
<keras.src.callbacks.History at 0x17ae4fc53a0>
```

In [33]:
```
ann.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 13s 39ms/step - loss: 1.4941 -
accuracy: 0.4664
```

Out[33]:
```
[1.4940873384475708, 0.46639999747276306]
```

In [40]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
y_pred = ann.predict(x_test)
y_pred_classes =[np.argmax(element) for element in y_pred]
y_pred_classes

```

```
313/313 [==============================] - 13s 42ms/step
```

Out[40]:
```
[3,
 9,
 9,
 0,
 4,
 6,
 3,
 4,
 4,
 1,
 0,
 9,
 5,
 7,
 9,
 8,
 7,
 4
```

In [103]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(x_test)
y_pred_classes = [np.argmax(element) for element in y_pred]
print("Classification Report: \n", classification_report(y_test, y_pred
```

```
313/313 [==============================] - 14s 42ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00      1000
           1       0.00      0.00      0.00      1000
           2       0.14      0.00      0.00      1000
           3       0.08      0.01      0.01      1000
           4       0.00      0.00      0.00      1000
           5       0.00      0.00      0.00      1000
           6       0.08      0.35      0.13      1000
           7       0.12      0.07      0.08      1000
           8       0.15      0.02      0.03      1000
           9       0.05      0.25      0.09      1000

    accuracy                           0.07     10000
   macro avg       0.06      0.07      0.03     10000
weighted avg       0.06      0.07      0.03     10000


C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [108]:
```
ann.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
```

In [ ]:
```
ann.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
1549/1563 [============================>.] - ETA: 4s - loss: 2.3028 - accu
racy: 0.1001
```

In [41]:
```
ann = models.Sequential([
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

```

In [42]:
```
(0.45) / (0.45+0.67)
```

Out[42]:
```
0.40178571428571425
```

In [43]:
```
(0.67) / (0.45+0.67)
```

Out[43]:
```
0.5982142857142857
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

text_Representation_Using_Bag_Of_n-grams

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer
v = CountVectorizer()
v.fit(["HELLO!, THIS IS CRISTIANO RONALDO"])
v.vocabulary_
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[3]:
```
{'hello': 1, 'this': 4, 'is': 2, 'cristiano': 0, 'ronaldo': 3}
```

In [4]:
```
v = CountVectorizer(ngram_range=(1,2))
v.fit(["HELLO!, THIS IS CRISTIANO RONALDO"])
v.vocabulary_

```

Out[4]:
```
{'hello': 2,
 'this': 7,
 'is': 4,
 'cristiano': 0,
 'ronaldo': 6,
 'hello this': 3,
 'this is': 8,
 'is cristiano': 5,
 'cristiano ronaldo': 1}
```

In [5]:
```
v = CountVectorizer(ngram_range=(1,3))
v.fit(["HELLO!, THIS IS CRISTIANO RONALDO"])
v.vocabulary_
```

Out[5]:
```
{'hello': 2,
 'this': 9,
 'is': 5,
 'cristiano': 0,
 'ronaldo': 8,
 'hello this': 3,
 'this is': 10,
 'is cristiano': 6,
 'cristiano ronaldo': 1,
 'hello this is': 4,
 'this is cristiano': 11,
 'is cristiano ronaldo': 7}
```

In [6]:
```
corpus = [
 " Ronaldo loves water",
 "Messi is Pessi",
 "Messi likes Pepsi"
]
```

In [7]:
```
preprocess = ("Ronaldo likes water")
preprocess
```

Out[7]:
```
'Ronaldo likes water'
```

In [8]:
```
preprocess = ("Messi likes Pepsi")
preprocess
```

Out[8]:
```
'Messi likes Pepsi'
```

In [9]:
```
import spacy

nlp = spacy.load("en_core_web_sm")

def preprocess(text):
    doc = nlp(text)
    filtered_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
        filtered_tokens.append(token.lemma_)
    return " ".join(filtered_tokens)

# Assuming 'corpus' is a list of strings
corpus_processed = [preprocess(text) for text in corpus]
corpus_processed
```

Out[9]:
```
['  Ronaldo love water', 'Messi Pessi', 'Messi like Pepsi']
```

In [10]:
```
import spacy
nlp = spacy.load("en_core_web_sm")
def preprocess(text):
    doc = nlp(text)
    filtered_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
            filtered_tokens.append(token.lemma_)
            return " ".join(filtered_tokens)
```

In [11]:
```
preprocess("Ronaldo likes water")

```

In [12]:
```
preprocess("Messi likes Pepsi")

```

In [13]:
```
corpus_processed = [
 preprocess(text) for text in corpus
]
corpus_processed
```

Out[13]:
```
[None, None, None]
```

In [ ]:
```
v.transform(["Ronaldo likes water"]).toarray()
```

In [ ]:
```

```

In [ ]:
```

```

tf/idf

In [2]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
sents = ["coronavirus is a highly infectious disease",
        "coronavirus affects older people the most",
        "older people are at high risk due to this diease"]
```

In [5]:
```
tfidf = TfidfVectorizer()
transformed = tfidf.fit_transform(sents)

import pandas as pd
df = pd.DataFrame(transformed[0].T.todense(),
                  index = tfidf.get_feature_names_out(),
                  columns=["TF-IDF"])
df = df.sort_values('TF-IDF',ascending = False)
df
```

Out[5]:

	TF-IDF
infectious	0.467351
disease	0.467351
highly	0.467351
is	0.467351
coronavirus	0.355432
most	0.000000
this	0.000000
the	0.000000
risk	0.000000
people	0.000000
older	0.000000
affects	0.000000
are	0.000000
high	0.000000
due	0.000000
diease	0.000000
at	0.000000
to	0.000000

text preprocess

In [9]:
```
pip install python-aiml

```

```
Collecting python-aimlNote: you may need to restart the kernel to use upda
ted packages.

  Downloading python-aiml-0.9.3.zip (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 252.7 kB/s eta 0:
00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: setuptools in c:\users\anusha v\anaconda3\l
ib\site-packages (from python-aiml) (63.4.1)
Building wheels for collected packages: python-aiml
  Building wheel for python-aiml (setup.py): started
  Building wheel for python-aiml (setup.py): finished with status 'done'
  Created wheel for python-aiml: filename=python_aiml-0.9.3-py3-none-any.w
hl size=2122485 sha256=4aca5c28ac9dd05ac41c5cc552300e15c1f570b29748f0c1fce
873ece75f1091
  Stored in directory: c:\users\anusha v\appdata\local\pip\cache\wheels\fe
\2a\54\7317c5c3ace9ca4f12aca947fc4fcedb945a0dfe32281756bc
Successfully built python-aiml
Installing collected packages: python-aiml
Successfully installed python-aiml-0.9.3
```

In [10]:

```
import aiml

# Create an AIML kernel
kernel = aiml.Kernel()

# Load AIML files into the kernel
kernel.learn("path/to/your/aiml/files/*.aiml")

# Function for text preprocessing
def preprocess_input(user_input):
    # Convert to lowercase
    user_input = user_input.lower()

    # Perform additional preprocessing if needed (e.g., removing specia
    # Add your preprocessing steps here

    return user_input

# Get user input
user_input = input("User: ")

# Preprocess the input
preprocessed_input = preprocess_input(user_input)

# Get the bot's response using AIML
bot_response = kernel.respond(preprocessed_input)

# Print the response
print("Bot:", bot_response)


# Create an AIML kernel
kernel = aiml.Kernel()

# Load AIML files into the kernel
kernel.learn("path/to/your/aiml/files/*.aiml")

# Function for text preprocessing
def preprocess_input(user_input):
    # Convert to lowercase
    user_input = user_input.lower()

    # Perform additional preprocessing if needed (e.g., removing specia
    # Add your preprocessing steps here

    return user_input

# Get user input
user_input = input("User: ")

# Preprocess the input
preprocessed_input = preprocess_input(user_input)

# Get the bot's response using AIML
bot_response = kernel.respond(preprocessed_input)

# Print the response
print("Bot:", bot_response)
```

```
59
```

```
User: anusha v

WARNING: No match found for input: anusha v

Bot:
User: Anushav
Bot:

WARNING: No match found for input: anushav
```

In [ ]:

```
1
```