1. import numpy as np

def one\_hot\_encode\_words(text):

words = text.lower().split()

unique\_words = sorted(list(set(words)))

word\_to\_int = {word: i for i, word in enumerate(unique\_words)}

vocab\_size = len(unique\_words)

one\_hot\_encoded\_vectors = {}

all\_vectors = []

for word in words:

vector = np.zeros(vocab\_size, dtype=int)

vector[word\_to\_int[word]] = 1

one\_hot\_encoded\_vectors[word] = vector

all\_vectors.append(vector)

return one\_hot\_encoded\_vectors, word\_to\_int, np.array(all\_vectors)

print("--- Word One-Hot Encoding Example ---")

sentence = "The cat sat on the mat"

word\_vectors, word\_vocab, word\_array = one\_hot\_encode\_words(sentence)

print("Original Sentence:", sentence)

for word, vector in word\_vectors.items():

print(f"'{word}': {vector}")

print("\nEncoded sequence of vectors for the sentence:")

print(word\_array)

print("-" \* 40 + "\n")

--- Word One-Hot Encoding Example ---

Original Sentence: The cat sat on the mat

'the': [0 0 0 0 1]

'cat': [1 0 0 0 0]

'sat': [0 0 0 1 0]

'on': [0 0 1 0 0]

'mat': [0 1 0 0 0]

Encoded sequence of vectors for the sentence:

[[0 0 0 0 1] [1 0 0 0 0] [0 0 0 1 0] [0 0 1 0 0] [0 0 0 0 1] [0 1 0 0 0]]

**2.**

import tensorflow as tf

from tensorflow import keras

from keras.datasets import imdb

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Embedding, GlobalAveragePooling1D, Dense, Dropout

VOCAB\_SIZE = 10000

MAX\_LEN = 256

EMBEDDING\_DIM = 16

BATCH\_SIZE = 512

EPOCHS = 10

print("Loading IMDB dataset...")

(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=VOCAB\_SIZE)

print(f"Number of training sequences: {len(train\_data)}")

print(f"Number of testing sequences: {len(test\_data)}")

print("Padding sequences...")

train\_data = pad\_sequences(train\_data, maxlen=MAX\_LEN, padding='post', truncating='post')

test\_data = pad\_sequences(test\_data, maxlen=MAX\_LEN, padding='post', truncating='post')

print(f"Shape of training data after padding: {train\_data.shape}")

print(f"Shape of testing data after padding: {test\_data.shape}")

print("Building the model...")

model = Sequential([

Embedding(input\_dim=VOCAB\_SIZE, output\_dim=EMBEDDING\_DIM, input\_shape=[MAX\_LEN]),

GlobalAveragePooling1D(),

Dense(16, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

print("Compiling the model...")

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.summary()

print("\n--- Training the model ---")

x\_val = train\_data[:10000]

partial\_x\_train = train\_data[10000:]

y\_val = train\_labels[:10000]

partial\_y\_train = train\_labels[10000:]

history = model.fit(

partial\_x\_train,

partial\_y\_train,

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

validation\_data=(x\_val, y\_val),

verbose=1

)

print("\n--- Evaluating the model ---")

results = model.evaluate(test\_data, test\_labels, verbose=2)

print(f"\nTest Loss: {results[0]:.4f}")

print(f"Test Accuracy: {results[1]:.4f}")

prediction = model.predict(test\_data[0:1])

print(f"\nPrediction for first test review: {prediction[0][0]:.4f}")

print(f"Actual label for first test review: {test\_labels[0]}")

print("A prediction > 0.5 is considered positive, and <= 0.5 is negative.")  
(25000, 256)  
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ embedding (Embedding) │ (None, 256, 16) │ 160,000 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ global\_average\_pooling1d │ (None, 16) │ 0 │

│ (GlobalAveragePooling1D) │ │ │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_2 (Dense) │ (None, 16) │ 272 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout (Dropout) │ (None, 16) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_3 (Dense) │ (None, 1) │ 17 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 160,289 (626.13 KB) **Trainable params:** 160,289 (626.13 KB) **Non-trainable params:** 0 (0.00 B)

--- Evaluating the model --- 782/782 - 2s - 2ms/step - accuracy: 0.8389 - loss: 0.4250 Test Loss: 0.4250 Test Accuracy: 0.8389 **1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 215ms/step Prediction for first test review: 0.411

**4.**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

feature\_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',

'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']

X = pd.DataFrame(data, columns=feature\_names)

y = pd.Series(target, name='MEDV')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

model = keras.Sequential([

layers.Dense(64, activation='relu', input\_shape=[X\_train.shape[1]]),

layers.Dense(64, activation='relu'),

layers.Dense(1)

])

model.compile(optimizer='adam',

loss='mean\_squared\_error',

metrics=['mean\_absolute\_error'])

model.summary()

history = model.fit(

X\_train\_scaled, y\_train,

epochs=100,

validation\_split=0.2,

verbose=1

)

loss, mae = model.evaluate(X\_test\_scaled, y\_test, verbose=0)

print("\n--- Model Evaluation ---")

print(f"Mean Absolute Error on Test Data: {mae:.2f}")

test\_predictions = model.predict(X\_test\_scaled).flatten()

print("\n--- Example Predictions ---")

for i in range(5):

print(f"Predicted Price: {test\_predictions[i]:.2f}, Actual Price: {y\_test.iloc[i]:.2f}")

**Model: "sequential\_2"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ dense\_4 (Dense) │ (None, 64) │ 896 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_5 (Dense) │ (None, 64) │ 4,160 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_6 (Dense) │ (None, 1) │ 65 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 5,121 (20.00 KB)

**Trainable params:** 5,121 (20.00 KB)

**Non-trainable params:** 0 (0.00 B)

Epoch 1/100**11/11** ━━━━━━━━━━━━━━━━━━━━ **3s** 130ms/step - loss: 602.7761 - mean\_absolute\_error: 22.5738 - val\_loss: 525.9553 - val\_mean\_absolute\_error: 21.3836

Epoch 100/100 **11/11** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 7.1968 - mean\_absolute\_error: 1.9888 - val\_loss: 13.6237 - val\_mean\_absolute\_error: 2.6591 --- Model Evaluation --- Mean Absolute Error on Test Data: 2.32 **4/4** ━━━━━━━━━━━━━━━━━━━━ **0s** 62ms/step --- Example Predictions --- Predicted Price: 27.37, Actual Price: 23.60 Predicted Price: 33.55, Actual Price: 32.40 Predicted Price: 16.75, Actual Price: 13.60 Predicted Price: 26.21, Actual Price: 22.80 Predicted Price: 16.09, Actual Price: 16.10

**6.**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import os

import shutil

import matplotlib.pyplot as plt

from PIL import Image

base\_dir = 'cats\_and\_dogs\_small'

if os.path.exists(base\_dir):

shutil.rmtree(base\_dir)

os.makedirs(base\_dir)

train\_dir = os.path.join(base\_dir, 'train')

os.makedirs(train\_dir)

validation\_dir = os.path.join(base\_dir, 'validation')

os.makedirs(validation\_dir)

train\_cats\_dir = os.path.join(train\_dir, 'cats')

os.makedirs(train\_cats\_dir)

train\_dogs\_dir = os.path.join(train\_dir, 'dogs')

os.makedirs(train\_dogs\_dir)

validation\_cats\_dir = os.path.join(validation\_dir, 'cats')

os.makedirs(validation\_cats\_dir)

validation\_dogs\_dir = os.path.join(validation\_dir, 'dogs')

os.makedirs(validation\_dogs\_dir)

def create\_dummy\_files(directory, prefix, num\_files):

for i in range(num\_files):

with open(os.path.join(directory, f'{prefix}.{i}.jpg'), 'w') as f:

f.write(f'Dummy: {prefix} {i}')

create\_dummy\_files(train\_cats\_dir, 'cat', 100)

create\_dummy\_files(train\_dogs\_dir, 'dog', 100)

create\_dummy\_files(validation\_cats\_dir, 'cat', 50)

create\_dummy\_files(validation\_dogs\_dir, 'dog', 50)

def create\_pixel\_image(directory):

for filename in os.listdir(directory):

if filename.endswith(".jpg"):

img = Image.new('RGB', (150, 150), color='black')

img.save(os.path.join(directory, filename))

create\_pixel\_image(train\_cats\_dir)

create\_pixel\_image(train\_dogs\_dir)

create\_pixel\_image(validation\_cats\_dir)

create\_pixel\_image(validation\_dogs\_dir)

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest'

)

validation\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(150, 150),

batch\_size=20,

class\_mode='binary'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

validation\_dir,

target\_size=(150, 150),

batch\_size=20,

class\_mode='binary'

)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dropout(0.5),

Dense(512, activation='relu'),

Dense(1, activation='sigmoid')

])

model.summary()

model.compile(

loss='binary\_crossentropy',

optimizer=tf.keras.optimizers.RMSprop(learning\_rate=1e-4),

metrics=['accuracy']

)

history = model.fit(

train\_generator,

steps\_per\_epoch=5,

epochs=15,

validation\_data=validation\_generator,

validation\_steps=2,

verbose=2

)

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, loss, 'ro', label='Training loss')

plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.suptitle('CNN Training Metrics')

plt.show()

shutil.rmtree(base\_dir)

util.rmtree(base\_dir)

**10.**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

max\_features = 10000

maxlen = 500

batch\_size = 32

print("Loading data...")

(input\_train, y\_train), (input\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(input\_train), 'train sequences')

print(len(input\_test), 'test sequences')

print("Pad sequences (samples x time)")

input\_train = sequence.pad\_sequences(input\_train, maxlen=maxlen)

input\_test = sequence.pad\_sequences(input\_test, maxlen=maxlen)

print('input\_train shape:', input\_train.shape)

print('input\_test shape:', input\_test.shape)

print("\nBuilding the RNN model...")

model = Sequential()

model.add(Embedding(max\_features, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

model.build(input\_shape=(None, maxlen))

model.summary()

print("\nCompiling the model...")

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

print("\nTraining the model...")

history = model.fit(input\_train, y\_train, epochs=10, batch\_size=batch\_size, validation\_split=0.2)

print("\nEvaluating the model on the test set...")

loss, accuracy = model.evaluate(input\_test, y\_test, batch\_size=batch\_size)

print(f"Test Loss: {loss:.4f}")

print(f"Test Accuracy: {accuracy:.4f}")

**1.**

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import numpy as np

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Using device: {device}")

input\_size = 784

hidden\_size1 = 512

hidden\_size2 = 256

num\_classes = 10

num\_epochs = 15

batch\_size = 128

learning\_rate = 0.001

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

train\_dataset = torchvision.datasets.MNIST(root='./data', train=True, transform=transform, download=True)

test\_dataset = torchvision.datasets.MNIST(root='./data', train=False, transform=transform)

train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset, batch\_size=batch\_size, shuffle=True)

test\_loader = torch.utils.data.DataLoader(dataset=test\_dataset, batch\_size=batch\_size, shuffle=False)

def imshow(img):

img = img / 2 + 0.5

npimg = img.numpy()

plt.imshow(np.transpose(npimg, (1, 2, 0)))

plt.show()

dataiter = iter(train\_loader)

images, labels = next(dataiter)

print("Visualizing a batch of training data:")

imshow(torchvision.utils.make\_grid(images[:8]))

print('Labels: ', ' '.join(f'{labels[j]}' for j in range(8)))

class NeuralNet(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size1, hidden\_size2, num\_classes):

super(NeuralNet, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, hidden\_size1)

self.relu1 = nn.ReLU()

self.dropout1 = nn.Dropout(0.2)

self.fc2 = nn.Linear(hidden\_size1, hidden\_size2)

self.relu2 = nn.ReLU()

self.dropout2 = nn.Dropout(0.2)

self.fc3 = nn.Linear(hidden\_size2, num\_classes)

def forward(self, x):

out = self.fc1(x)

out = self.relu1(out)

out = self.dropout1(out)

out = self.fc2(out)

out = self.relu2(out)

out = self.dropout2(out)

out = self.fc3(out)

return out

model = NeuralNet(input\_size, hidden\_size1, hidden\_size2, num\_classes).to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=learning\_rate)

print("\nStarting training...")

n\_total\_steps = len(train\_loader)

for epoch in range(num\_epochs):

model.train()

for i, (images, labels) in enumerate(train\_loader):

images = images.reshape(-1, 28\*28).to(device)

labels = labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if (i+1) % 200 == 0:

print(f'Epoch [{epoch+1}/{num\_epochs}], Step [{i+1}/{n\_total\_steps}], Loss: {loss.item():.4f}')

print("Finished training.")

print("\nEvaluating model on test data...")

with torch.no\_grad():

model.eval()

n\_correct = 0

n\_samples = 0

for images, labels in test\_loader:

images = images.reshape(-1, 28\*28).to(device)

labels = labels.to(device)

outputs = model(images)

\_, predicted = torch.max(outputs.data, 1)

n\_samples += labels.size(0)

n\_correct += (predicted == labels).sum().item()

acc = 100.0 \* n\_correct / n\_samples

print(f'Accuracy of the network on the 10,000 test images: {acc:.2f} %')

**3.**

import numpy as np

from tensorflow.keras.datasets import reuters

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

print("Loading Reuters dataset...")

(train\_data, train\_labels), (test\_data, test\_labels) = reuters.load\_data(num\_words=10000)

print(f"Number of training samples: {len(train\_data)}")

print(f"Number of testing samples: {len(test\_data)}")

print(f"Number of categories: {len(np.unique(train\_labels))}")

print("--- Sample training data (integer sequence) ---")

print(train\_data[0])

word\_index = reuters.get\_word\_index()

reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])

decoded\_newswire = ' '.join([reverse\_word\_index.get(i - 3, '?') for i in train\_data[0]])

print("\n--- Decoded sample newswire ---")

print(decoded\_newswire)

def vectorize\_sequences(sequences, dimension=10000):

results = np.zeros((len(sequences), dimension))

for i, sequence in enumerate(sequences):

results[i, sequence] = 1.

return results

print("\nVectorizing data...")

x\_train = vectorize\_sequences(train\_data)

x\_test = vectorize\_sequences(test\_data)

one\_hot\_train\_labels = to\_categorical(train\_labels)

one\_hot\_test\_labels = to\_categorical(test\_labels)

print("Building the neural network model...")

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(10000,)))

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

model.add(Dense(46, activation='softmax'))

print("Compiling the model...")

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

x\_val = x\_train[:1000]

partial\_x\_train = x\_train[1000:]

y\_val = one\_hot\_train\_labels[:1000]

partial\_y\_train = one\_hot\_train\_labels[1000:]

print("\nTraining the model...")

history = model.fit(partial\_x\_train, partial\_y\_train, epochs=20, batch\_size=512, validation\_data=(x\_val, y\_val), verbose=1)

print("\nEvaluating the model on the test set...")

results = model.evaluate(x\_test, one\_hot\_test\_labels, batch\_size=512)

print("--- Test Results ---")

print(f"Test Loss: {results[0]:.4f}")

print(f"Test Accuracy: {results[1]:.4f}")

history\_dict = history.history

loss\_values = history\_dict['loss']

val\_loss\_values = history\_dict['val\_loss']

acc\_values = history\_dict['accuracy']

val\_acc\_values = history\_dict['val\_accuracy']

epochs = range(1, len(loss\_values) + 1)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs, loss\_values, 'bo', label='Training loss')

plt.plot(epochs, val\_loss\_values, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, acc\_values, 'bo', label='Training acc')

plt.plot(epochs, val\_acc\_values, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

print("\nMaking predictions on the test set...")

predictions = model.predict(x\_test)

predicted\_class = np.argmax(predictions[0])

print(f"\nSample Prediction for the first test newswire:")

print(f"Predicted class index: {predicted\_class}")

print(f"Actual class index: {test\_labels[0]}")

print(f"Confidence: {predictions[0][predicted\_class]:.2%}")

**9.**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense

import numpy as np

vocab\_size = 10000

(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=vocab\_size)

word\_index = imdb.get\_word\_index()

word\_index = {k: (v + 3) for k, v in word\_index.items()}

word\_index["<PAD>"] = 0

word\_index["<START>"] = 1

word\_index["<UNK>"] = 2

word\_index["<UNUSED>"] = 3

reverse\_word\_index = {value: key for key, value in word\_index.items()}

def decode\_review(text):

return ' '.join([reverse\_word\_index.get(i, '?') for i in text])

print("--- Example Decoded Review ---")

print(decode\_review(train\_data[0]))

print("Label:", train\_labels[0])

print("-" \* 30)

max\_length = 256

train\_data = pad\_sequences(train\_data, value=word\_index["<PAD>"], padding='post', maxlen=max\_length)

test\_data = pad\_sequences(test\_data, value=word\_index["<PAD>"], padding='post', maxlen=max\_length)

embedding\_dim = 16

model = Sequential([

Embedding(vocab\_size, embedding\_dim),

GlobalAveragePooling1D(),

Dense(16, activation='relu'),

Dense(1, activation='sigmoid')

])

model.build(input\_shape=(None, max\_length))

model.summary()

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(

train\_data,

train\_labels,

epochs=30,

batch\_size=512,

validation\_split=0.2,

verbose=1

)

results = model.evaluate(test\_data, test\_labels, verbose=2)

print("\n--- Model Evaluation ---")

print(f"Test Loss: {results[0]}")

print(f"Test Accuracy: {results[1]}")

print("-" \* 30)

def preprocess\_text(text):

words = text.lower().split()

encoded\_review = [word\_index.get(word, 2) for word in words]

padded\_review = pad\_sequences([encoded\_review], value=word\_index["<PAD>"], padding='post', maxlen=max\_length)

return padded\_review

positive\_review = "this movie was fantastic I really enjoyed it and would recommend it to everyone"

preprocessed\_positive = preprocess\_text(positive\_review)

prediction\_positive = model.predict(preprocessed\_positive)

print(f"Prediction for positive review: {prediction\_positive[0][0]}")

negative\_review = "it was a terrible movie I would not recommend it to anyone"

preprocessed\_negative = preprocess\_text(negative\_review)

prediction\_negative = model.predict(preprocessed\_negative)

print(f"Prediction for negative review: {prediction\_negative[0][0]}")

**7.**

import numpy as np

from tensorflow.keras.applications.vgg16 import VGG16, preprocess\_input, decode\_predictions

from tensorflow.keras.preprocessing import image

from PIL import Image

import requests

from io import BytesIO

import matplotlib.pyplot as plt

print("Loading VGG16 model...")

model = VGG16(weights='imagenet')

print("Model loaded successfully.")

image\_url = 'https://upload.wikimedia.org/wikipedia/commons/4/45/A\_small\_cup\_of\_coffee.JPG'

print(f"Loading image from URL: {image\_url}")

headers = {'User-Agent': 'Mozilla/5.0'}

response = requests.get(image\_url, headers=headers)

response.raise\_for\_status()

img = Image.open(BytesIO(response.content))

img\_resized = img.resize((224, 224))

img\_array = image.img\_to\_array(img\_resized)

img\_batch = np.expand\_dims(img\_array, axis=0)

img\_preprocessed = preprocess\_input(img\_batch)

print("Classifying the image...")

predictions = model.predict(img\_preprocessed)

decoded\_predictions = decode\_predictions(predictions, top=3)[0]

print("Classification complete.")

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(img)

plt.title("Original Image")

plt.axis('off')

plt.subplot(1, 2, 2)

y\_pos = np.arange(len(decoded\_predictions))

performance = [pred[2] for pred in decoded\_predictions]

class\_names = [pred[1] for pred in decoded\_predictions]

plt.barh(y\_pos, performance, align='center', color='skyblue')

plt.yticks(y\_pos, class\_names)

plt.gca().invert\_yaxis()

plt.xlabel('Probability')

plt.title('Top 3 Predictions')

for index, value in enumerate(performance):

plt.text(value, index, f"{value:.2f}")

plt.tight\_layout()

plt.show()

print("\n--- Top 3 Predictions ---")

for i, (imagenet\_id, label, score) in enumerate(decoded\_predictions):

print(f"{i+1}: {label} ({score:.2%})")

**5.**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

print("x\_train shape:", x\_train.shape, "y\_train shape:", y\_train.shape, "x\_test shape:", x\_test.shape, "y\_test shape:", y\_test.shape)

np.random.seed(0)

plt.xticks([])

plt.yticks([])

plt.xlabel([y\_train[1]])

plt.imshow(x\_train[1], cmap=plt.cm.binary)

plt.show()

x\_train = x\_train.astype('float32') / 255

x\_test = x\_test.astype('float32') / 255

x\_train = x\_train.reshape(-1, 28, 28, 1)

x\_test = x\_test.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(64, (2, 2), strides=(1, 1), padding='same', activation='relu', input\_shape=(28, 28, 1)))

model.add(tf.keras.layers.MaxPooling2D((2, 2)))

model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Conv2D(32, (2, 2), strides=(1, 1), padding='same', activation='relu'))

model.add(tf.keras.layers.MaxPooling2D((2, 2)))

model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(256, activation='relu'))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model\_log = model.fit(x\_train, y\_train, batch\_size=60, epochs=10, verbose=1, validation\_split=0.3)

score = model.evaluate(x\_test, y\_test, verbose=0)

print('\nTest accuracy:', score[1])

predictions = model.predict(x\_test)

x\_test = x\_test.reshape(-1, 28, 28)

plt.xticks([])

plt.yticks([])

plt.xlabel([y\_train[0]])

plt.imshow(x\_test[0], cmap=plt.cm.binary)

plt.show()

plt.figure()

plt.subplot(2, 1, 1)

plt.plot(model\_log.history['accuracy'])

plt.plot(model\_log.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.subplot(2, 1, 2)

plt.plot(model\_log.history['loss'])

plt.plot(model\_log.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.tight\_layout()

plt.show()

def plot\_image(i, predictions\_array, true\_label, img):

predictions\_array, true\_label, img = predictions\_array[i], true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

color = 'blue' if predicted\_label == true\_label else 'red'

plt.xlabel(f"{[predicted\_label]} {100\*np.max(predictions\_array):.0f}% ({[true\_label]})", color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

predictions\_array, true\_label = predictions\_array[i], true\_label[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

bars = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

bars[predicted\_label].set\_color('red')

bars[true\_label].set\_color('blue')

num\_rows, num\_cols = 5, 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i + 1)

plot\_image(i, predictions, y\_test, x\_test)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i + 2)

plot\_value\_array(i, predictions, y\_test)

plt.show()