ಗ್ಲೋಬಲ್ ಅಕಾಡೆಮಿ ಆಫ್ ಟೆಕ್ನಾಲಜಿ, ಬೆಂಗಳೂರು
GLOBAL ACADEMY OF TECHNOLOGY
An Autonomous Institute Affiliated to VTU-Belagavi, Approved by AICTE and Accredited by NAAC, "A" grade
Aditya Layout, Ideal Homes Township, RR Nagar, Bengaluru, Karnataka 560098
Department of Computer Science and Engineering
Accredited by NBA 2022 - 2025
+91 80 28603158(Ext - 300) hodcse@gat.ac.in www.gat.ac.in

LAB MANUAL

Deep Learning Laboratory with Mini Project Manual

VII Semester

Course Code: 20CSEL76

Version 1.0

w.e.f 15.11.2023

Editorial Committee

DL Faculty
Dept. of CSE

Approved by

Dr. Kumaraswamy S
Professor & Head,
Department of CSE

ಗ್ಲೋಬಲ್ ಅಕಾಡೆಮಿ ಆಫ್ ಟೆಕ್ನಾಲಜಿ, ಬೆಂಗಳೂರು
GLOBAL ACADEMY OF TECHNOLOGY
An Autonomous Institute Affiliated to VTU-Belagavi, Approved by AICTE and Accredited by NAAC, "A" grade
Aditya Layout, Ideal Homes Township, RR Nagar, Bengaluru, Karnataka 560098
Department of Computer Science and Engineering
Accredited by NBA 2022 - 2025
+91 80 28603158(Ext - 300) hodcse@gat.ac.in www.gat.ac.in

LABORATORY CERTIFICATE

This is to certify that Mr./ Ms .....

bearing USN..... of Department of Computer Science and Engineering has satisfactorily completed the course of Deep Learning Laboratory with Mini Project(20CSEL76) prescribed by Global Academy of Technology (Autonomous Institute, under VTU) for the Academic Year 2023-2024

Marks	
Maximum	**Obtained**
50	

Signature of the Faculty

Signature of the HOD

Date:

DOCUMENT LOG

Name of the document	Deep Learning Laboratory with Mini Project Manual
Syllabus Scheme	2020
Current version number and date	Version 1.0 / 15.11.2023
Subject code	20CSEL76
Editorial Committee	DL Faculty
Approved by	HOD, Dept. of CSE
Lab Faculty	Prof. Kamleshwar Kumar Yadav Prof. Ravindranath R C Prof. Snigdha Sen
Computer Programmer	Mr. Parashivamurthy C

TABLE OF CONTENTS

Vision of the Department

To achieve academic excellence and strengthen the skills to meet emerging challenges of Computer Science and Engineering.

Mission of the Department

M1: To impart a strong theoretical foundation in the field of Computer Science and Engineering accompanied by extensive practical skills.

M2: To inculcate research and innovation spirit through interaction with industry and carry out projects that address societal needs.

M3: Instill professional ethics and values with concern for the environment.

Program Educational Objectives (PEOs) of the Department

After the course completion, CSE graduates will be able to:

PEO1: Succeed in engineering/management positions with professional ethics.

PEO2: Engage in improving professional knowledge through certificate/post-graduate programs in engineering or management.

PEO3: Establish themselves as entrepreneurs and contribute to the Society.

Program Specific Outcomes (PSOs)

PSO1: Design, implement, and test System Software and Application Software to meet the desired needs.

PSO2: Develop solutions in the area of Communication, Networks, database systems, and computing systems

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Details

Course Name: Deep Learning Laboratory with Mini Project

Course Code: 20CSEL76

Course prerequisite: Linear Algebra, Probability, Python, ML

Course Learning Objectives

1. Implement the various deep learning algorithms in Python.
2. Demonstrate different deep learning frameworks like Keras, Tensorflow, PyTorch, Anaconda, etc.
3. Design and develop an application using specific deep learning models.
4. Demonstrate practical knowledge in handling and analyzing real-world applications.

SYLLABUS

Semester:	**VII**	**CIE Marks:**	**50**
Course Code:	**20CSEL76**	**SEE Marks:**	**50**
Hours/Week (L: T:P):	**0:0:2**	**Duration of SEE (Hours):**	**03**
Credits: 01			

Prerequisites: NIL

Course Learning Objectives:

CLO1	Implement the various deep learning algorithms in Python.
CLO2	Demonstrate different deep learning frameworks like Keras, Tensorflow, PyTorch, Anaconda, etc.
CLO3	Design and develop an application using specific deep learning models.
CLO4	Demonstrate practical knowledge in handling and analyzing real-world applications.

1. Features of this course include the use of NVidia Al/ Deep learning Software/ Libraries, Tensor Flow, Caffe, Caffe2, PyTorch, Misc, Numpy, Scikit, pandas, other relevant py libs, Inference of trained model on embedded GPU board using CUDA/ cuDNN/ TensorRT.
2. Installation procedure of the required software must be demonstrated, carried out in groups, and documented in the report.

Lab Programs:

PART A

1. Implement the following for the CIFAR-10 (Canadian Institute for Advanced Research) dataset:
    a. using KNN,
    b. using 3-layer neural network
2. Train a Deep learning model to classify a given image using the pre-trained model.
3. Train CNN using Tensorflow and Keras.
4. Improve the Deep learning model (ex. CNN model of Question 3) by tuning hyperparameters.
5. Implement an Object detection using a Convolution Neural Network variant. (ex. Faster RCNN)
6. Perform Sentiment Analysis using RNN.
7. Design a Chatbot using bi-directional LSTMs.

PART B

Mini Project:

For the Selected Problem statement, demonstrate the following:

a. Image Classification
b. Neural Network Development

Course Outcomes

Upon completion of this course, students will be able to:

20CSEL76.1	Use the characteristics of deep learning models that are needed to solve real-world problems.
20CSEL76.2	Apply appropriate deep learning algorithms for analyzing the data for a variety of problems.
20CSEL76.3	Implement different methodologies for applications using deep nets.
20CSEL76.4	Carry out the test procedures to assess the efficiency of the developed model.

CO-PO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	2	2	3	-	2	-	-	1	2	2	1	2	-	2
CO2	2	2	3	-	2	-	-	1	2	2	1	2	-	2
CO3	2	2	3	-	2	-	-	1	2	2	1	2	-	2
CO4	2	2	3	-	2	-	-	1	2	2	1	2	-	2
CO5	2	2	3	-	2	-	-	1	2	2	1	2	-	2
AVG	**2**	**2**	**3**	**-**	**2**	**-**	**-**	**1**	**2**	**2**	**1**	**2**	**-**	**2**

Assessment Details (both CIE and SEE):

The weightage of Continuous Internal Evaluation (CIE) and Semester End Examination (SEE) are 50% each. The student must obtain a minimum of 40% marks both in CIE and SEE to pass. Practical Semester End Exam (SEE) is conducted for 100 marks (3 hours duration). Based on this, grading will be awarded.

Continuous Internal Evaluation (CIE):

60% CIE marks awarded in case of practical shall be based on the weekly evaluation of laboratory reports after the conduction of every experiment and 40% marks for one practice test for practical-based learning with mini project.

Conduct of Practical Examination (SEE):

The Laboratories having **PART A** and **PART B**: Students are allowed to pick one experiment from **PART A** and demonstrate the mini project as **PART B**.
Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.

MARKS DISTRIBUTION

1. Continuous Internal Evaluation (CIE)	
A	**Regular Lab (Max. Marks 30)** Observation + Record + Viva Voce = 5 + 20 + 5 = 30 M
B	**Internal Assessment (Max. Marks 20)** (Writeup + Execution + Viva Voce) + Project = (10+20+10=40: reduced 10) + 10 = 20 M
CIE Total Marks = A + B = 50 Marks	
2. Semester End Examination (SEE):	
A	**Part A:** Writeup + Execution + Viva Voce = 10 + 40 + 10 = 60 M
B	**Part B:** Demonstration + Report + Viva Voce = 25 + 10 + 5 = 40 M
Total Marks = A + B = 100 Marks (scaled down to 50) **SEE Total Marks = 50 Marks**	

LAB EVALUATION PROCESS

Continuous Internal Evaluation (CIE)

Evaluation of CIE		
S. No	**Activity**	**Marks**
1	Average of Weekly Entries	30
2	Internal Assessment reduced to	20
	TOTAL	**50**

Semester End Examination (SEE)

Evaluation of CIE		
S. No	**Activity**	**Marks**
1	**Part A:** Writeup + Execution + Viva Voce	60
2	**Part B:** Demonstration + Report + Viva Voce	40
	TOTAL	**100**

LAB RUBRICS

Rubrics for Evaluation of Record and Observation

Attribute	Max. Marks	Good	Satisfactory	Poor
		4-5	2-3	0-1
Writeup	**05**	Writes the programwithout errors.	Writes the program withfew mistakes.	Unable to write theprogram.
	Max. Marks	**10-20**	**5-9**	**0-4**
Execution of program	**20**	• Debugs the program independently. • Executed the program for all possible inputs.	• Works with little help from faculty. • All possible inputcases are not covered.	• Unable to complete the execution of the program within the labsession.
	Max. Marks	**3-5**	**1-2**	**0**
Viva Voce	**05**	• Able to explain the logic of the program. • Answered all questions.	• Partially understood thelogic of the program. • Answered few questions.	• Not understood the logic of the program. • Not answering any questions.

Rubrics for Evaluation of Internal

Test Part A

Attribute	Max Marks	Good	Satisfactory	Poor
		4-5	2-3	0-1
Writeup	10	• Completes source code. • No syntax and logical errors. • All possible inputs are listed with the expected output.	• Completes source code. • Syntax and logical errors exist. • All possible inputs are listed with the expected output.	• Incomplete source code. • Change of program.
	Max Marks	**21-30**	**6-20**	**1-5**
Execution	20	• Able to debug the program and get the right set of outputs.	• The program is executed for only a few input cases.	Not executed.
	Max Marks	**3-5**	**1-2**	**0**
Viva Voce	10	Answering all questions.	Answering a few questions.	Not Able to answer basic questions.

Part B

Sl. No.	Criteria	Marks
1	Problem statement, Collection of data	02
2	Implementation	04
3	Project Demo (Demonstration & Questionnaire Session)	02
4	Report	02
TOTAL		**10**

Introduction

In the ever-evolving landscape of artificial intelligence, deep learning has emerged as a transformative force, revolutionizing the way machines learn and make decisions. Deep learning is a subset of machine learning that utilizes neural networks with multiple layers (deep neural networks) to model and process complex patterns in data. This sophisticated approach has found applications across various domains, from image and speech recognition to natural language processing and autonomous systems.

At its core, deep learning mimics the human brain's neural architecture, with interconnected layers of artificial neurons, also known as nodes. These nodes process information and pass it through the layers, allowing the system to recognize intricate patterns and make informed predictions. The depth of these networks enables the model to learn hierarchical representations of data, capturing both simple and abstract features.

Neural Networks and Training:

The training process in deep learning involves feeding the neural network vast amounts of labeled data, allowing it to adjust its parameters through iterative optimization. This is achieved through a process called backpropagation, where the model learns from its mistakes and refines its predictions. The depth of the network is crucial, as it enables the model to automatically extract features from the data, eliminating the need for manual feature engineering.

Applications Across Industries:

Deep learning has made significant strides in computer vision, powering facial recognition systems, autonomous vehicles, and medical image analysis. In natural language processing, it facilitates language translation, sentiment analysis, and chatbot interactions. Technology has also found its way into finance, predicting market trends, fraud detection, and personalized recommendations.

Challenges and Future Directions:

Despite its remarkable achievements, deep learning faces challenges such as the need for large, labeled datasets, interpretability of complex models, and computational resources. Researchers are actively exploring ways to make deep learning more transparent, efficient, and applicable to a broader range of tasks. Hybrid models, combining deep learning with other machine learning techniques, are emerging to address specific limitations.

As deep learning continues to push the boundaries of artificial intelligence, its impact on various industries is undeniable. The ability of deep neural networks to automatically learn and adapt from data has opened new possibilities and fueled innovation. With ongoing research and advancements, the future promises even more breakthroughs, bringing us closer to realizing the full potential of this transformative technology.

Origin and short history of the different Deep Learning Techniques/ Algorithms:

The origins of deep learning can be traced back to the 1940s and 1950s, with the conceptualization of artificial neural networks inspired by the structure and function of the human brain. Early pioneers like Warren McCulloch and Walter Pitts laid the groundwork for these networks, theorizing that interconnected artificial neurons could perform complex computations.

The late 1970s and early 1980s saw a resurgence of interest in neural networks, thanks in part to the development of the backpropagation algorithm. This breakthrough allowed for efficient training of multi-layered neural networks, overcoming some of the challenges that had hindered progress during the AI winter.

The true ascent of deep learning began in the 21st century, fueled by the confluence of abundant data, powerful computational resources, and innovative algorithms. In 2006, Geoffrey Hinton and his team demonstrated the effectiveness of deep neural networks in the field of speech recognition, sparking a renewed interest in deep learning.

The pivotal moment came in 2012 when Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton achieved ground breaking success in the ImageNet Large Scale Visual Recognition Challenge. Their deep convolutional neural network (CNN) surpassed traditional methods, showcasing the potential of deep learning in image classification.

Since the ImageNet breakthrough, deep learning has permeated various domains, achieving remarkable success in computer vision, natural language processing, healthcare, finance, and beyond. The advent of deep learning frameworks, such as TensorFlow and PyTorch, has democratized the development of sophisticated models, enabling researchers and practitioners worldwide.

Program 1

Implement the following for the CIFAR-10 (Canadian Institute for Advanced Research) dataset:

a. using KNN,

b. using 3-layer neural network

Program: # Using KNN

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# Load the CIFAR-10 dataset (You may need to download it separately)
# Load and preprocess the data
cifar10 = datasets.fetch_openml(name="CIFAR_10_small")
X = cifar10.data.astype('float32')
y = cifar10.target.astype('int')
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create and train a K-Nearest Neighbors (KNN) classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
# Evaluate the classifier
accuracy = knn.score(X_test, y_test)
print(f"KNN Accuracy: {accuracy:.2f}")
```

Output: KNN Accuracy: 0.30

Code: # 3-layer neural network

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from torch.utils.data import DataLoader
# Set random seed for reproducibility
torch.manual_seed(42)
# Define a simple 3-layer neural network for Softmax classification
class SimpleNet(nn.Module):
    def __init__(self, input_size, num_classes):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)
    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
# Load and preprocess the CIFAR-10 dataset using torchvision
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
cifar10_train = datasets.CIFAR10(root='./data', train=True, transform=transform, download=True)
cifar10_test = datasets.CIFAR10(root='./data', train=False, transform=transform, download=True)
# Set data loaders
train_loader = DataLoader(cifar10_train, batch_size=64, shuffle=True)
test_loader = DataLoader(cifar10_test, batch_size=64)
# Initialize the model and optimizer
model = SimpleNet(input_size=32*32*3, num_classes=10)
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
criterion = nn.CrossEntropyLoss()
# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
# Evaluate the model on the test set
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
accuracy = 100 * correct / total
print(f"3-Layer Neural Network Accuracy: {accuracy:.2f}%")
```

Output:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 42993818.54it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
3-Layer Neural Network Accuracy: 53.74%
```

Applications:

Text mining, Agriculture, Finance, Medical and Facial recognition

Program 2

Train a Deep learning model to classify a given image using the pre-trained model.

Code:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
import numpy as np
from PIL import Image
# Define data transformations for preprocessing
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
# Load a pre-trained ResNet-50 model
model = models.resnet50(pretrained=True)
num_features = model.fc.in_features
# Replace the final fully connected layer for binary classification
model.fc = nn.Linear(num_features, 2)
# Define a loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
# Load an image to classify
image_path = '/...../Path to Cat1.jpg'
image = Image.open(image_path)
image = data_transforms['train'](image)
```

```
image = image.unsqueeze(0)  # Add batch dimension
# Load the pre-trained model's weights
#model.load_state_dict(torch.load('/content/drive/MyDrive/20CSEL76-LAB Programs/resnet50-0676ba61.pth'))
# Save the model state dictionary to a file
torch.save(model.state_dict(), 'resnet50_pretrained_weights.pth')
# Set the model to evaluation mode
model.eval()
# Make predictions
with torch.no_grad():
    outputs = model(image)
    _, predicted = torch.max(outputs, 1)
# Map class indices to class labels
class_labels = ['cat', 'dog']
predicted_class = class_labels[predicted.item()]
print(f'The image is classified as: {predicted_class}')
```

Output: The image is classified as: cat

Applications:

Translation, chatbots and other natural language processing applications

Program 3

Train a CNN using Tensorflow and Keras

Code:

```
import tensorflow as tf
from tensorflow import keras
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
# Preprocess the data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Define a simple CNN model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(32, 32, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
# Compile the model
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
Epoch 1/10
625/625 [==============================] - 42s 63ms/step - loss: 1.5269 - accuracy: 0.4552 - val_loss: 1.3100 - val_accuracy: 0.5384
Epoch 2/10
625/625 [==============================] - 36s 58ms/step - loss: 1.2210 - accuracy: 0.5699 - val_loss: 1.2296 - val_accuracy: 0.5697
Epoch 3/10
625/625 [==============================] - 37s 59ms/step - loss: 1.1088 - accuracy: 0.6115 - val_loss: 1.1191 - val_accuracy: 0.6138
Epoch 4/10
625/625 [==============================] - 35s 57ms/step - loss: 1.0218 - accuracy: 0.6432 - val_loss: 1.1003 - val_accuracy: 0.6196
Epoch 5/10
625/625 [==============================] - 36s 57ms/step - loss: 0.9418 - accuracy: 0.6746 - val_loss: 1.0600 - val_accuracy: 0.6335
Epoch 6/10
625/625 [==============================] - 39s 62ms/step - loss: 0.8775 - accuracy: 0.6961 - val_loss: 1.0440 - val_accuracy: 0.6437
Epoch 7/10
625/625 [==============================] - 39s 62ms/step - loss: 0.8259 - accuracy: 0.7135 - val_loss: 1.0736 - val_accuracy: 0.6319
Epoch 8/10
625/625 [==============================] - 39s 62ms/step - loss: 0.7702 - accuracy: 0.7320 - val_loss: 1.0551 - val_accuracy: 0.6459
Epoch 9/10
625/625 [==============================] - 36s 58ms/step - loss: 0.7217 - accuracy: 0.7499 - val_loss: 1.0430 - val_accuracy: 0.6486
Epoch 10/10
625/625 [==============================] - 35s 56ms/step - loss: 0.6729 - accuracy: 0.7691 - val_loss: 1.0607 - val_accuracy: 0.6454
313/313 [==============================] - 4s 12ms/step - loss: 1.0715 - accuracy: 0.6375
Test accuracy: 63.75%
```

Applications:

Image Classification, Object Recognition, facial recognition, medical image analysis and self-driving cars.

Program 4

Improve the Deep learning model (ex. CNN model of Question 3) by tuning hyperparameters.

Code:

```
import numpy as np
import pandas as pd

# visuals
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,classification_report

# keras
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from keras.models import load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import models,layers
from tensorflow.keras.layers import Dense,Dropout,BatchNormalization,Conv2D,MaxPool2D,Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers.experimental import SGD,Adam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers, optimizers
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
```

```
# Normalizing
X_train = X_train/255
X_test = X_test/255
# One-Hot-Encoding
Y_train_en = to_categorical(Y_train,10)
Y_test_en = to_categorical(Y_test,10)

# Data Augementation
datagen = ImageDataGenerator(rotation_range=10,
                             horizontal_flip=True,
                             width_shift_range=0.1,
                             height_shift_range=0.1
                               )
train_gen = datagen.flow(X_train,Y_train_en,batch_size=64)

# Base Model
model = Sequential()
model.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(128, activation ='relu'))
model.add(Dense(10, activation ='softmax'))
model.compile(loss ='categorical_crossentropy', optimizer ='adam', metrics =['accuracy'])

model.summary()
history = model.fit(train_gen, epochs = 10, verbose=1,validation_data=(X_test,Y_test_en))

test_loss, test_accuracy = model.evaluate(X_test, Y_test_en)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 29, 32)        1568

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        16416

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 128)               102528

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 121,802
Trainable params: 121,802
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/10
782/782 [==============================] - 85s 106ms/step - loss: 1.7021 - ac
curacy: 0.3776 - val_loss: 1.4883 - val_accuracy: 0.4668
Epoch 2/10
782/782 [==============================] - 78s 99ms/step - loss: 1.4259 - acc
uracy: 0.4836 - val_loss: 1.2738 - val_accuracy: 0.5434
Epoch 3/10
782/782 [==============================] - 81s 104ms/step - loss: 1.3215 - ac
curacy: 0.5288 - val_loss: 1.1958 - val_accuracy: 0.5712
Epoch 4/10
782/782 [==============================] - 76s 97ms/step - loss: 1.2607 - acc
uracy: 0.5531 - val_loss: 1.1577 - val_accuracy: 0.5868
Epoch 5/10
782/782 [==============================] - 81s 103ms/step - loss: 1.2155 - ac
curacy: 0.5687 - val_loss: 1.1262 - val_accuracy: 0.6050
Epoch 6/10
782/782 [==============================] - 76s 97ms/step - loss: 1.1669 - acc
uracy: 0.5859 - val_loss: 1.0600 - val_accuracy: 0.6236
Epoch 7/10
782/782 [==============================] - 76s 98ms/step - loss: 1.1291 - acc
uracy: 0.6013 - val_loss: 1.0621 - val_accuracy: 0.6254
Epoch 8/10
782/782 [==============================] - 101s 130ms/step - loss: 1.1017 - a
ccuracy: 0.6120 - val_loss: 1.0035 - val_accuracy: 0.6530
Epoch 9/10
782/782 [==============================] - 107s 136ms/step - loss: 1.0680 - a
ccuracy: 0.6231 - val_loss: 0.9930 - val_accuracy: 0.6553
Epoch 10/10
782/782 [==============================] - 82s 105ms/step - loss: 1.0473 - ac
curacy: 0.6318 - val_loss: 0.9843 - val_accuracy: 0.6580
313/313 [==============================] - 3s 8ms/step - loss: 0.9843 - accur

acy: 0.6580
Test accuracy: 65.80%
```

Applications:

Increasing model efficiency, robustness

Program 5

Implement an Object detection using a Convolution Neural Network variant. (ex. Faster RCNN)

Code:

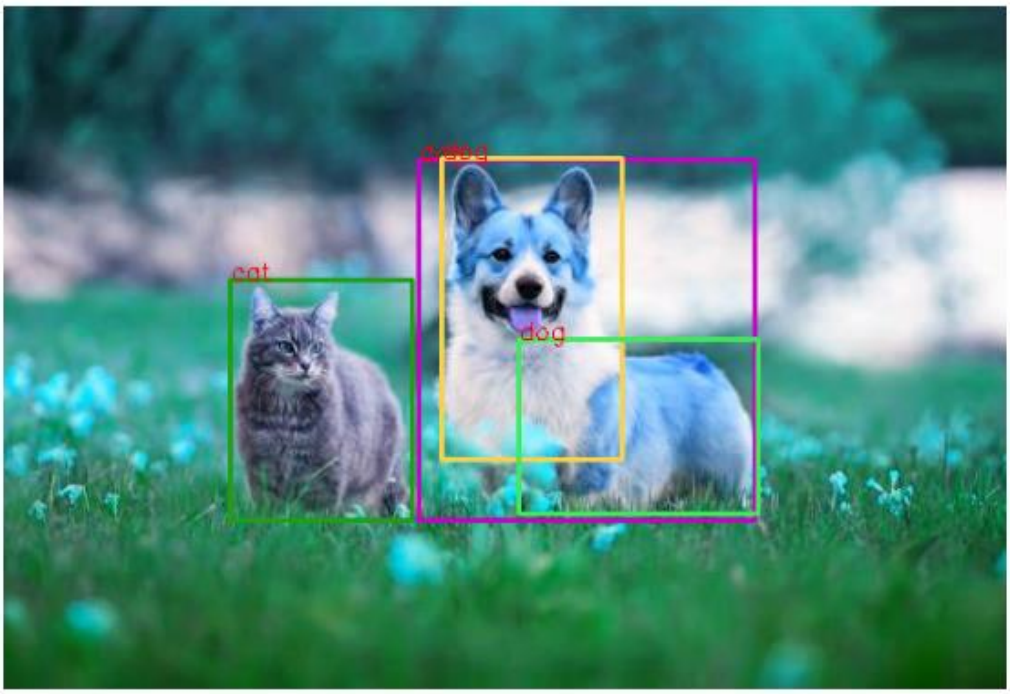
```
from PIL import Image
import cv2
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision.transforms as T
import torchvision

# Download the pre-trained model from Pytorch repository
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
print("type(model): \t", type(model), "\n")
print("model: \n", model, "\n")
# Set the model to evaluation mode; this step is IMPORTANT
model.eval()

# Getting the list of all categories used to train the Faster R-CNN network
# Predictions from this list are supposed to be returned
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
```

```
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]
print("Faster R-CNN network train on ", len(COCO_INSTANCE_CATEGORY_NAMES), " object categories")
THRESHOLD = 0.8
test_image_path ='/path to /object detection.jpg'  # Replace with your image path
test_image = Image.open(test_image_path)
print("type(test_image): \t", test_image, "\n")
# transforming the test image to a Pytorch tensor
transform = T.Compose([T.ToTensor()])
test_image_tensor = transform(test_image)
print("type(test_image_tensor): \t", type(test_image_tensor), "\n")
print("test_image_tensor.size(): \t", test_image_tensor.size(), "\n")
print("test_image_tensor: \n", test_image_tensor, "\n")
# Make predictions on the test image using pre-trained model
with torch.no_grad():
    predictions = model([test_image_tensor])

print("type(predictions): \t", type(predictions), "\n")
print("len(predictions): \t", len(predictions), "\n")
print("predictions: \n", predictions, "\n")
# Getting the actual predictions
prediction = predictions[0]
print("type(prediction): \t", type(prediction), "\n")

for key in prediction.keys():
    print(key, "\t", prediction[key])

# Extract the individual prediction components
bounding_boxes = prediction['boxes']
print("bounding_boxes: \n", bounding_boxes, "\n")
# Detach the bounding boxes from the computation graph
bounding_boxes = bounding_boxes.cpu().detach().numpy()
```

```
print("bounding_boxes: \n", bounding_boxes, "\n")
print("Number of bounding boxes (objects) detected: \t", bounding_boxes.shape[0], "\n")
# Extract the individual prediction components
class_labels = prediction['labels']
print("class_labels: \t\t\t\t", class_labels, "\n")
print("Number of bounding boxes (objects) detected: \t", class_labels.shape[0], "\n")
# Now visualizing the detection results
# Swapping the axes to convert the dimension in the format (H, W, Channels)
test_image = cv2.imread(test_image_path)
test_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)
print("Unannotated test image")
plt.imshow(test_image)
plt.show()

font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1.0
font_color = (255, 255, 255)  # White text color
font_thickness = 2

# Define the classes you want to keep (dog and cat)
classes_to_keep = ["dog", "cat"]

for box_index in range(bounding_boxes.shape[0]):
    class_index = int(class_labels[box_index])
    class_name = COCO_INSTANCE_CATEGORY_NAMES[class_index]

    if class_name in classes_to_keep:
        print("Showing box: ", box_index+1, "\n")
        x1, y1, x2, y2 = bounding_boxes[box_index]
        random_color = list(np.random.choice(range(256), size=3))
        x1 = int(x1)
        x2 = int(x2)
        y1 = int(y1)
```

```
        y2 = int(y2)
        cv2.rectangle(test_image, (x1, y1), (x2, y2), (int(random_color[0]), int(random_color[1]),
int(random_color[2])), 2)
        cv2.putText(test_image, class_name, (x1, y1), font, font_scale, font_color, font_thickness)

plt.imshow(test_image)
plt.show()
```

Output:



Applications:

Object detection and Region Proposal Network (RPN)

Program 6

Perform Sentiment Analysis using RNN.

Code:

```
from keras.datasets import imdb
from keras import Sequential
from keras.layers import Embedding, Dense, Dropout
from keras.layers import Activation, SimpleRNN
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = vocabulary_size)
print('Loaded dataset with {} training samples, {} test samples'.format(len(X_train), len(X_test)))
print('---review---')
print(X_train[6])
print('---label---')
print(y_train[6])
word2id = imdb.get_word_index()
id2word = {i: word for word, i in word2id.items()}
print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[6]])
print('---label---')
print(y_train[6])
print('Maximum review length: {}'.format(
len(max((X_train + X_test), key=len))))
print('Minimum review length: {}'.format(
len(min((X_test + X_test), key=len))))
from keras.preprocessing import sequence
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
embedding_size=32
model=Sequential()
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(SimpleRNN(100))
```

```
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
batch_size = 64
num_epochs = 1
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train2, y_train2 = X_train[batch_size:], y_train[batch_size:]
model.fit(X_train2, y_train2, validation_data=(X_valid, y_valid), batch_size=batch_size,
epochs=num_epochs)
scores = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', scores[1])
```

Output: Test accuracy: 0.6672000288963318

Application:

Speech recognition, voice recognition, time series prediction, and natural language processing

Program 7

Design a Chatbot using bi-directional LSTMs

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Sample training data
training_data = [
    "Hello",
    "Hi",
    "How are you?",
    "I'm doing well, thanks!",
    "What's your name?",
    "I'm a chatbot.",
    "Tell me a joke",
    "Why don't scientists trust atoms?",
    "Because they make up everything!",
    "exit"
]
# Tokenize the training data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(training_data)
total_words = len(tokenizer.word_index) + 1
# Create input sequences and labels
input_sequences = []
for line in training_data:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
```

```
        input_sequences.append(n_gram_sequence)
# Pad sequences
max_sequence_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')
# Create inputs and labels
input_sequences = np.array(input_sequences)
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
# Build the model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Embedding(total_words, 64, input_length=max_sequence_length - 1))
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)))
model.add(tf.keras.layers.Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train the model
model.fit(X, y, epochs=100, verbose=1)
# Function to generate a response
def generate_response(input_text):
    input_seq = tokenizer.texts_to_sequences([input_text])[0]
    input_seq = pad_sequences([input_seq], maxlen=max_sequence_length - 1, padding='pre')
    predicted_probs = model.predict(input_seq)[0]
    predicted_id = np.argmax(predicted_probs)
    # Check if predicted_id is within a valid range
    if predicted_id >= 1 and predicted_id <= len(tokenizer.word_index):
        predicted_word = list(tokenizer.word_index.keys())[predicted_id - 1]
    else:
        predicted_word = "UNKNOWN_WORD"
    return predicted_word
# Chat with the bot
user_input = "Hello"
```

```
while user_input != "exit":
    response = generate_response(user_input)
    print(f"User: {user_input}")
    print(f"Chatbot: {response}")
    user_input = input("You: ")
```

Output:

```
1/1 [==============================] - 0s 16ms/step - loss: 1.4513 - accuracy: 0.8500
Epoch 100/100
1/1 [==============================] - 0s 19ms/step - loss: 1.4191 - accuracy: 0.8500
1/1 [==============================] - 1s 877ms/step
User: Hello
Chatbot: a
You: how are you
1/1 [==============================] - 0s 22ms/step
User: how are you
Chatbot: you
You: im fine
1/1 [==============================] - 0s 21ms/step
User: im fine
Chatbot: a
You: what is a
1/1 [==============================] - 0s 26ms/step
User: what is a
Chatbot: chatbot
You: what's your name?
1/1 [==============================] - 0s 24ms/step
User: what's your name?
Chatbot: name
You: exit
```

Applications:

Sentiment analysis, language modeling, speech recognition, and video analysis.

ADDITIONAL PROGRAMS

1. Machine Learning with Keras: https://www.tensorflow.org/tutorials/keras/classification
2. Load and preprocess images: https://www.tensorflow.org/tutorials/load_data/images
3. TensorFlow Tutorials: https://www.tensorflow.org/tutorials/quickstart/beginner
4. CNN: https://www.tensorflow.org/tutorials/images/cnn
5. Object Detection: https://www.tensorflow.org/hub/tutorials/tf2_object_detection
6. Multilingual Question and Answering: https://www.tensorflow.org/hub/tutorials/retrieval_with_tf_hub_universal_encoder_qa

VIVA QUESTIONS

1. What's the difference between bias and variance?

Bias is error due to erroneous or overly simplistic assumptions in the learning algorithm. This can lead to the model underfitting your data, making it hard for it to have high predictive accuracy

Variance is error due to too much complexity in the learning algorithm. This leads to the algorithm being highly sensitive to high degrees of variation in your training data, which can lead your model to overfit the data.

2. What is the difference between supervised and unsupervised machine learning?

Supervised learning requires training labeled data. For example, in order to do classification (a supervised learning task), you'll need to first label the data you'll use to train the model to classify data into your labeled groups. Unsupervised learning, in contrast, does not require labeling data explicitly.

3. How is KNN different from k-means clustering?

K-Nearest Neighbors is a supervised classification algorithm, while k-means clustering is an unsupervised clustering algorithm. While the mechanisms may seem similar at first, what this really means is that in order for K-Nearest Neighbors to work, you need labeled data you want to classify an unlabeled point into (thus the nearest neighbor part). K-means clustering requires only a set of unlabeled points and a threshold: the algorithm will take unlabeled points and gradually learn how to cluster them into groups by computing the mean of the distance between different points.

The critical difference here is that KNN needs labeled points and is thus supervised learning, while k-means doesn't — and is thus unsupervised learning.

4. Explain how a ROC curve works.

The ROC curve is a graphical representation of the contrast between true positive rates and false positive rate at various thresholds. It's often used as a proxy for the trade-off between the sensitivity of the model (true positives) vs the fall-out or the probability it will trigger a false alarm (false positives).

5. Define precision and recall.

Recall is also known as the true positive rate: the amount of positives your model claims compared to the actual number of positives there are throughout the data. Precision is also known as the positive predictive value, and it is a measure of the amount of accurate positives your model claims compared to the number of positives it actually claims.

6. What is Bayes' Theorem? How is it useful in a machine learning context?

Bayes' Theorem gives you the posterior probability of an event given what is known as prior knowledge.

7. Why is "Naive" Bayes naive?

Despite its practical applications, especially in text mining, Naive Bayes is considered "Naive" because it makes an assumption that is virtually impossible to see in real-life data: the conditional probability is calculated as the pure product of the individual probabilities of components. This implies the absolute independence of features — a condition probably never met in real life.

8. Explain the difference between L1 and L2 regularization.

L2 regularization tends to spread error among all the terms, while L1 is more binary/sparse, with many variables either being assigned a 1 or 0 in weighting. L1 corresponds to setting a Laplacian prior on the terms, while L2 corresponds to a Gaussian prior.

9. What's your favorite algorithm, and can you explain it to me in less than a minute?

This type of question tests your understanding of how to communicate complex and technical nuances with poise and the ability to summarize quickly and efficiently. Make sure you have a choice and make sure you can explain different algorithms so simply and effectively that a five-year-old could grasp the basics.

10. What's the difference between a generative and discriminative model?

A generative model will learn categories of data while a discriminative model will simply learn the distinction between different categories of data. Discriminative models will generally outperform generative models on classification tasks.

11. What cross-validation technique would you use on a time series dataset?

Instead of using standard k-fold cross-validation, you have to pay attention to the fact that a time series is not randomly distributed data — it is inherently ordered by chronological order.

12. How is a decision tree pruned?

Pruning is what happens in decision trees when branches that have weak predictive power are removed in order to reduce the complexity of the model and increase the predictive accuracy of a decision tree model. Pruning can happen bottom-up and top-down, with approaches such as reduced error pruning and cost complexity pruning.

13. What's the F1 score? How would you use it?

The F1 score is a measure of a model's performance. It is a weighted average of the precision and recall of a model, with results tending to 1 being the best, and those tending to 0 being the worst. You would use it in classification tests where true negatives don't matter much.

14. How would you handle an imbalanced dataset?

An imbalanced dataset is when you have, for example, a classification test and 90% of the data is in one class. That leads to problems: an accuracy of 90% can be skewed if you have no predictive power on the other category of data! Here are a few tactics to get over the hump:

a. Collect more data to even the imbalances in the dataset.
b. Resample the dataset to correct for imbalances
c. Try a different algorithm altogether on your dataset.

15. When should you use classification over regression?

Classification produces discrete values and dataset to strict categories, while regression gives you continuous results that allow you to better distinguish differences between individual points.

16. How do you ensure you're not overfitting with a model?

This is a simple restatement of a fundamental problem in machine learning: the possibility of overfitting training data and carrying the noise of that data through to the test set, thereby providing inaccurate generalizations.

There are three main methods to avoid overfitting:

a. Keep the model simpler: reduce variance by taking into account fewer variables and parameters, thereby removing some of the noise in the training data.
b. Use cross-validation techniques such as k-fold cross-validation.
c. Use regularization techniques such as LASSO that penalize certain model parameters if they're likely to cause overfitting.

17. What is the purpose of activation functions in neural networks?

Activation functions introduce non-linearities to the neural network, enabling it to learn complex relationships in the data. Common activation functions include sigmoid, tanh, and ReLU.

18. Describe the architecture of a convolutional neural network.

A CNN consists of convolutional layers, pooling layers, and fully connected layers. Convolutional layers use filters to capture spatial hierarchies in the data while pooling layers reduce spatial dimensions.

19. How does a recurrent neural network differ from a feedforward network?

RNNs have connections that form directed cycles, allowing them to capture sequential information. This makes them suitable for tasks where the order of input data matters.

20. What is the role of a loss function in deep learning?

The loss function measures the difference between the predicted output and the actual target. During training, the goal is to minimize this loss to improve the model's performance.

21. Name some popular deep learning frameworks.

TensorFlow, PyTorch, and Keras are widely used deep learning frameworks. They provide high-level abstractions for building and training neural networks.

22. What are word embeddings, and why are they used in NLP?

Word embeddings are dense vector representations of words in a continuous vector space. They capture semantic relationships between words and are used in NLP to represent words in a more meaningful way than traditional one-hot encoding.

23. What are word embeddings, and why are they used in NLP?

Word embeddings are dense vector representations of words in a continuous vector space. They capture semantic relationships between words and are used in NLP to represent words in a more meaningful way than traditional one-hot encoding.

24. Explain the Word2Vec model.

Word2Vec is a shallow neural network model that learns word embeddings by predicting the surrounding words in a given context. It has two architectures: Continuous Bag of Words (CBOW) and Skip-gram.

25. How are Recurrent Neural Networks (RNNs) utilized in NLP tasks?

RNNs are used in NLP for sequential data processing. They can capture context dependencies, making them suitable for tasks like text generation, sentiment analysis, and machine translation.

26. Why are LSTMs preferred over traditional RNNs in NLP?

LSTMs address the vanishing gradient problem associated with traditional RNNs, enabling them to capture long-term dependencies in sequences. This is crucial for NLP tasks where context over longer spans is important.