

**BANSILAL RAMNATH AGARWAL CHARITABLE TRUST’S**

**VISHWAKARMA INSTITUTE OF INFORMATION.TECHNOLOGY.**

## Sr.No.2/3/4,Kondhwa(BK),Pune-48

LAB MANUAL

Deep Learning Lab Manual

FOR

Subject Code: **CAUA31202**

Submitted by:

Name: Sai Sanjay Nangare

RollNo: 381049

PRN: 22210801

# Assignment 1

**1. Implementing Feedforward Neural Networks in Python Using Keras and TensorFlow**

**Title**

**Building and Training Feedforward Neural Networks with Keras and TensorFlow**

**Aim**

To develop a feedforward neural network model using Keras and TensorFlow for supervised learning tasks.

**Objective**

* To understand the architecture of feedforward neural networks.
* To implement a neural network from scratch using Keras.
* To evaluate the model's performance on a dataset.

**Theory**

Feedforward neural networks consist of layers of neurons where information moves in one direction—from input to output. These networks can learn to approximate complex functions through training.

**Advantages**

* **Simplicity**: Easy to understand and implement.
* **Flexibility**: Can be adapted for various tasks (e.g., regression, classification).
* **Strong Performance**: Effective for a wide range of applications.

**Disadvantages**

* **Overfitting**: Prone to overfitting with complex models.
* **Training Time**: Can require significant time for training on large datasets.
* **Local Minima**: May converge to local minima rather than the global minimum.

**Libraries Used**

* **Keras**: High-level neural networks API, running on TensorFlow.
* **TensorFlow**: Framework for building and training models.
* **NumPy**: For numerical computations.

**INPUT**:

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt

import numpy as np

# Load MNIST dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.mnist.load\_data()

# Normalize pixel values to between 0 and 1

train\_images = train\_images.astype('float32') / 255.0

test\_images = test\_images.astype('float32') / 255.0

# Flatten images from 28x28 to 784-dimensional vectors

train\_images = train\_images.reshape((train\_images.shape[0], 28 \* 28))

test\_images = test\_images.reshape((test\_images.shape[0], 28 \* 28))

# Plot the first 5 images and their labels

plt.figure(figsize=(10,2))

for i in range(5):

plt.subplot(1,5,i+1)

plt.imshow(train\_images[i].reshape(28,28), cmap='gray')

plt.title(f"Label: {train\_labels[i]}")

plt.axis('off')

plt.show()# Define the model

model = models.Sequential([

    layers.Dense(128, activation='relu', input\_shape=(784,)),

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

    layers.Dense(10, activation='softmax')

])

model.summary()

# compile the model

model.compile(

    optimizer='adam',

    loss='sparse\_categorical\_crossentropy',

    metrics=['accuracy']

)

# Train the model

history = model.fit(

    train\_images,

    train\_labels,

    epochs=10,

    batch\_size=32,

    validation\_split=0.1

)

test\_loss, test\_accuracy = model.evaluate(test\_images, test\_labels)

print(f"Test Accuracy: {test\_accuracy\*100:.2f}%")

# Plot training & validation accuracy values

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Val Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# Plot training & validation loss values

plt.subplot(1,2,2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Val Loss')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Predict the first 5 images in the test set

predictions = model.predict(test\_images[:15])

# Convert predictions to class labels

predicted\_labels = np.argmax(predictions, axis=1)

# Plot the results

plt.figure(figsize=(10,2))

for i in range(15):

  plt.subplot(1,15,i+1)

  plt.imshow(test\_images[i].reshape(28,28), cmap='gray')

  plt.title(f"Pred: {predicted\_labels[i]}\nTrue: {test\_labels[i]}")

  plt.axis('off')

plt.show()

**Conclusion**

Feedforward neural networks are foundational in deep learning. By implementing them using Keras and TensorFlow, we can effectively tackle various supervised learning problems while gaining insights into model training and evaluation.

# Assignment 2

**2. Facial Recognition Using OpenCV and Deep Learning for Binary Classification**

**Title**

**Facial Recognition for Binary Classification Using OpenCV and Deep Learning**

**Aim**

To develop a facial recognition system for binary classification (e.g., identifying whether a person is recognized or not).

**Objective**

* To implement a facial recognition system using OpenCV.
* To train a deep learning model for binary classification of facial images.

**Theory**

Facial recognition involves detecting and identifying faces in images. Deep learning models, especially Convolutional Neural Networks (CNNs), are effective in extracting features for recognition tasks.

**Advantages**

* **High Accuracy**: Deep learning models provide high accuracy for classification.
* **Real-Time Processing**: OpenCV allows for real-time face detection and recognition.
* **Robustness**: Handles variations in lighting, angles, and facial expressions.

**Disadvantages**

* **Data Requirements**: Requires a large dataset for effective training.
* **Complexity**: Implementation can be complex and resource-intensive.
* **Privacy Concerns**: Raises ethical and privacy issues.

**Libraries Used**

* **OpenCV**: For image processing and face detection.
* **TensorFlow/Keras**: For building and training the classification model.
* **NumPy**: For numerical operations.

**INPUT**:

import cv2

# Initialize face detector

faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# Start video capture

video\_capture = cv2.VideoCapture(0)

while True:

# Capture frame-by-frame

ret, frame = video\_capture.read()

# Detect faces and draw rectangles

faces = faceCascade.detectMultiScale(cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY), 1.1, 5)

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 2)

# Display the resulting frame

cv2.imshow('Face Detection', frame)

# Exit on pressing 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release resources

video\_capture.release()

cv2.destroyAllWindows()

**Conclusion**

Using OpenCV and deep learning, we can create an effective facial recognition system. The system demonstrates the power of combining computer vision and neural networks for real-world applications.

# Assignment 3

**3. Implement Image Classification Using Convolutional Neural Networks (CNNs) for Multiclass Classification**

**Title**

**Multiclass Image Classification Using Convolutional Neural Networks**

**Aim**

To implement a CNN for classifying images into multiple categories.

**Objective**

* To understand the structure and functionality of CNNs.
* To build a CNN model using Keras.
* To evaluate the model's performance on a multiclass dataset.

**Theory**

CNNs are specialized neural networks designed for processing grid-like data, such as images. They consist of convolutional layers, pooling layers, and fully connected layers to extract and learn features.

**Advantages**

* **Feature Extraction**: Automatically learns spatial hierarchies of features.
* **High Performance**: Achieves state-of-the-art results in image classification tasks.
* **Translation Invariance**: Robust to shifts and distortions in images.

**Disadvantages**

* **Computationally Intensive**: Requires substantial computational power, especially for large datasets.
* **Overfitting**: Can easily overfit if not properly regularized.
* **Data Requirement**: Requires large labeled datasets for training.

**Libraries Used**

* **Keras**: For building CNN models.
* **TensorFlow**: Backend for Keras.
* **NumPy**: For data manipulation.

**INPUT**:

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Normalize pixel values

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# One-hot encode the labels

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Data augmentation

datagen = ImageDataGenerator(

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    horizontal\_flip=True

)

datagen.fit(x\_train)

# Build the CNN model

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))  # Dropout layer to prevent overfitting

model.add(Dense(10, activation='softmax'))  # Output layer for 10 classes

# Compile the model

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

# Train the model with data augmentation

history = model.fit(datagen.flow(x\_train, y\_train, batch\_size=32),

                    epochs=20, validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_accuracy:.4f}')

# Visualize the training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend()

plt.show()

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend()

plt.show()

**Conclusion**

CNNs are powerful tools for multiclass image classification, leveraging their ability to learn complex features from images. This assignment highlights the importance of CNNs in the field of computer vision.

# Assignment 4

**4. Time Series Prediction Using RNN - Stock Market Analysis or Weather Forecasting**

**Title**

**Time Series Prediction with Recurrent Neural Networks for Stock Market Analysis/Weather Forecasting**

**Aim**

To implement an RNN for predicting time series data such as stock prices or weather conditions.

**Objective**

* To understand the workings of RNNs for sequential data.
* To develop a model for predicting future values based on historical data.
* To assess the model's predictive performance.

**Theory**

RNNs are designed for sequential data and maintain information in their hidden states across time steps. They are suitable for tasks like time series prediction, where the order of data matters.

**Advantages**

* **Temporal Dependencies**: Effectively captures temporal relationships in data.
* **Dynamic Input Length**: Can process sequences of varying lengths.
* **Versatile**: Applicable to various types of sequential data (text, audio, etc.).

**Disadvantages**

* **Vanishing Gradient Problem**: Can struggle with long sequences due to gradient decay.
* **Training Time**: Longer training times compared to feedforward networks.
* **Complexity**: More challenging to implement and tune.

**Libraries Used**

* **Keras**: For building RNN models.
* **TensorFlow**: Backend for Keras.
* **NumPy**: For data manipulation.

**INPUT**:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

# Load the stock price data (replace with your dataset)

data = pd.read\_csv('AAPL.csv')  # Example dataset: Apple stock prices

data['Date'] = pd.to\_datetime(data['Date'])

data.set\_index('Date', inplace=True)

data = data[['Close']]  # Use closing prices for prediction

# Data preprocessing

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(data)

# Create training and testing datasets

train\_size = int(len(scaled\_data) \* 0.8)

train\_data = scaled\_data[:train\_size]

test\_data = scaled\_data[train\_size:]

# Function to create dataset

def create\_dataset(data, time\_step=1):

    X, Y = [], []

    for i in range(len(data) - time\_step):

        X.append(data[i:(i + time\_step), 0])

        Y.append(data[i + time\_step, 0])

    return np.array(X), np.array(Y)

# Create datasets

time\_step = 1

X\_train, y\_train = create\_dataset(train\_data, time\_step)

X\_test, y\_test = create\_dataset(test\_data, time\_step)

# Reshape input to be [samples, time steps, features]

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

# Build the RNN model

model = Sequential()

model.add(LSTM(50, return\_sequences=False, input\_shape=(X\_train.shape[1], 1)))  # Single LSTM layer

model.add(Dense(1))  # Output layer

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=1, verbose=2)  # Reduced epochs for simplicity

# Predicting the stock prices

train\_predict = model.predict(X\_train)

test\_predict = model.predict(X\_test)

# Inverse transform to get actual prices

train\_predict = scaler.inverse\_transform(train\_predict)

test\_predict = scaler.inverse\_transform(test\_predict)

# Create a new array to hold the predictions

train\_predict\_plot = np.empty\_like(scaled\_data)

train\_predict\_plot[:, :] = np.nan

train\_predict\_plot[time\_step:len(train\_predict) + time\_step, :] = train\_predict

# Create a new array to hold the test predictions

test\_predict\_plot = np.empty\_like(scaled\_data)

test\_predict\_plot[:, :] = np.nan

test\_predict\_plot[len(train\_predict) + (time\_step \* 2):len(scaled\_data), :] = test\_predict

# Plotting the results

plt.figure(figsize=(14, 5))

plt.plot(data.index, data['Close'], label='Actual Price', color='blue')

plt.plot(data.index, train\_predict\_plot, label='Train Predict', color='orange')

plt.plot(data.index, test\_predict\_plot, label='Test Predict', color='red')

plt.title('Stock Price Prediction')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.legend()

plt.show()

**Conclusion**

RNNs provide a robust approach for time series prediction tasks. This implementation illustrates the application of deep learning in forecasting and analyzing temporal data.

# Assignment 5

**5. Text Identification Using OpenCV, Tesseract (OCR), and Deep Neural Network**

**Title**

**Text Identification Using OpenCV and Tesseract OCR with Deep Learning**

**Aim**

To develop a system for identifying and extracting text from images using OpenCV and Tesseract OCR.

**Objective**

* To implement text detection and recognition in images.
* To evaluate the effectiveness of Tesseract OCR for text identification.

**Theory**

Optical Character Recognition (OCR) involves converting images of text into machine-encoded text. Tesseract is an OCR engine that can recognize text in images, and it can be enhanced with deep learning techniques.

**Advantages**

* **Accuracy**: Tesseract provides high accuracy for text recognition.
* **Integration**: Can be easily integrated with OpenCV for pre-processing images.
* **Multilingual Support**: Supports recognition of multiple languages.

**Disadvantages**

* **Image Quality Dependency**: Performance can degrade with poor-quality images.
* **Preprocessing Requirement**: Images often require preprocessing for optimal results.
* **Limited Context Understanding**: Lacks understanding of the context of text.

**Libraries Used**

* **OpenCV**: For image processing.
* **Tesseract**: For OCR capabilities.
* **NumPy**: For numerical operations.

**INPUT**:

!apt-get install -y tesseract-ocr  # Install Tesseract OCR engine

!pip install pytesseract  # Install pytesseract (Python wrapper for Tesseract)

import cv2          # For image processing

import numpy as np   # Used for numerical operations on image arrays

import pytesseract   # For Optical Character Recognition (OCR)

from google.colab.patches import cv2\_imshow  # To display images in Colab

from google.colab import files  # To handle file uploads in Colab

import os           # For file path operations

from google.colab import drive

drive.mount('/content/drive') # Connect to the Drive

def preprocess\_image(image):

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  # Convert to grayscale

    denoised = cv2.fastNlMeansDenoising(gray)       # Remove noise

    thresh = cv2.threshold(denoised, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]  # Binary thresholding

    return thresh

def detect\_text\_regions(image):

    # Detecting words

    boxes = pytesseract.image\_to\_data(image, output\_type=pytesseract.Output.DICT)  # Get text region data

    return boxes

def draw\_bounding\_boxes(image, boxes):

output = image.copy() # Copy original image to draw boxes

n\_boxes = len(boxes['level']) # Total number of detected text regions

for i in range(n\_boxes):

if int(boxes['conf'][i]) > 60: # Only consider boxes with confidence > 60%

(x, y, w, h) = (boxes['left'][i], boxes['top'][i], boxes['width'][i], boxes['height'][i]) # Box coordinates

cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 2) # Draw rectangle on the image

return output def get\_detected\_text(boxes):

    detected\_text = []

    n\_boxes = len(boxes['level'])  # Total number of detected boxes

    for i in range(n\_boxes):

        if int(boxes['conf'][i]) > 60:  # Only extract text with confidence > 60%

            detected\_text.append(boxes['text'][i])  # Append detected text

    return ' '.join(detected\_text)  # Combine text into a single string

from google.colab import drive

drive.mount('/content/drive')

image\_path = "/content/drive/MyDrive/WhatsApp Image 2024-10-14 at 01.34.41\_9ea2ccfc.jpg"

image = cv2.imread(image\_path)  # Read the image

# Check if image was successfully read

if image is None:

    print(f"Error: Unable to read the image file: {image\_path}")

else:

    # Display original image

    print("\nOriginal Image:")

    cv2\_imshow(image)

    # Preprocess the image and detect text regions

    preprocessed = preprocess\_image(image)  # Preprocessing

    boxes = detect\_text\_regions(preprocessed)  # Detect text regions

    # Draw bounding boxes on original image

    image\_with\_boxes = draw\_bounding\_boxes(image, boxes)

    # Display the image with bounding boxes

    print("\nImage with Text Detection Regions:")

    cv2\_imshow(image\_with\_boxes)

    # Extract and print the detected text

    detected\_text = get\_detected\_text(boxes)

    print("\nDetected Text:")

    print(detected\_text)

**Conclusion**

Combining OpenCV and Tesseract allows for effective text identification in images. This project demonstrates the integration of computer vision and OCR technologies to solve practical problems.

# Assignment 6

**6. Sentiment Analysis Using LSTM Network or GRU**

**Title**

**Sentiment Analysis Using LSTM Networks**

**Aim**

To implement a deep learning model for sentiment analysis using Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRU).

**Objective**

* To understand the concepts of LSTM and GRU for sequence modeling.
* To develop a sentiment analysis model to classify text data.
* To evaluate the model's accuracy in predicting sentiments.

**Theory**

LSTMs and GRUs are specialized RNN architectures designed to address the vanishing gradient problem. They are effective in capturing long-range dependencies in sequence data, making them suitable for tasks like sentiment analysis.

**Advantages**

* **Handling Long Sequences**: Capable of learning from long input sequences.
* **Improved Accuracy**: Generally outperform traditional RNNs in text classification tasks.
* **Flexibility**: Can be adapted for various sequence prediction problems.

**Disadvantages**

* **Training Complexity**: More complex to train and require careful tuning.
* **Computationally Intensive**: Higher computational cost compared to simpler models.
* **Data Requirement**: Requires a substantial amount of labeled training data.

**Libraries Used**

* **Keras**: For building LSTM/GRU models.
* **TensorFlow**: Backend for Keras.
* **NumPy**: For data manipulation.

**INPUT**:

import pandas as pd    # to load dataset

import numpy as np     # for mathematic equation

from nltk.corpus import stopwords   # to get collection of stopwords

from sklearn.model\_selection import train\_test\_split       # for splitting dataset

from tensorflow.keras.preprocessing.text import Tokenizer  # to encode text to int

from tensorflow.keras.preprocessing.sequence import pad\_sequences   # to do padding or truncating

from tensorflow.keras.models import Sequential     # the model

from tensorflow.keras.layers import Embedding, LSTM, Dense # layers of the architecture

from tensorflow.keras.callbacks import ModelCheckpoint   # save model

from tensorflow.keras.models import load\_model   # load saved model

import re

data = pd.read\_csv('IMDB Dataset.csv')

print(data)

english\_stops = set(stopwords.words('english'))

def load\_dataset():

    df = pd.read\_csv('IMDB Dataset.csv')

    x\_data = df['review']       # Reviews/Input

    y\_data = df['sentiment']    # Sentiment/Output

    # PRE-PROCESS REVIEW

    x\_data = x\_data.replace({'<.\*?>': ''}, regex = True)          # remove html tag

    x\_data = x\_data.replace({'[^A-Za-z]': ' '}, regex = True)     # remove non alphabet

    x\_data = x\_data.apply(lambda review: [w for w in review.split() if w not in english\_stops])  # remove stop words

    x\_data = x\_data.apply(lambda review: [w.lower() for w in review])   # lower case

    # ENCODE SENTIMENT -> 0 & 1

    y\_data = y\_data.replace('positive', 1)

    y\_data = y\_data.replace('negative', 0)

    return x\_data, y\_data

x\_data, y\_data = load\_dataset()

print('Reviews')

print(x\_data, '\n')

print('Sentiment')

print(y\_data)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.2)

print('Train Set')

print(x\_train, '\n')

print(x\_test, '\n')

print('Test Set')

print(y\_train, '\n')

print(y\_test)

def get\_max\_length():

    review\_length = []

    for review in x\_train:

        review\_length.append(len(review))

return int(np.ceil(np.mean(review\_length)))

# ENCODE REVIEW

token = Tokenizer(lower=False)    # no need lower, because already lowered the data in load\_data()

token.fit\_on\_texts(x\_train)

x\_train = token.texts\_to\_sequences(x\_train)

x\_test = token.texts\_to\_sequences(x\_test)

max\_length = get\_max\_length()

x\_train = pad\_sequences(x\_train, maxlen=max\_length, padding='post', truncating='post')

x\_test = pad\_sequences(x\_test, maxlen=max\_length, padding='post', truncating='post')

total\_words = len(token.word\_index) + 1   # add 1 because of 0 padding

print('Encoded X Train\n', x\_train, '\n')

print('Encoded X Test\n', x\_test, '\n')

print('Maximum review length: ', max\_length)

# ARCHITECTURE

EMBED\_DIM = 32

LSTM\_OUT = 64

model = Sequential()

model.add(Embedding(total\_words, EMBED\_DIM, input\_length = max\_length))

model.add(LSTM(LSTM\_OUT))

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

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

print(model.summary())

checkpoint = ModelCheckpoint(

    'models/LSTM.h5',

    monitor='accuracy',

    save\_best\_only=True,

    verbose=1

)

model.fit(x\_train, y\_train, batch\_size = 128, epochs = 5, callbacks=[checkpoint])

y\_pred = model.predict\_classes(x\_test, batch\_size = 128)

true = 0

for i, y in enumerate(y\_test):

    if y == y\_pred[i]:

        true += 1

print('Correct Prediction: {}'.format(true))

print('Wrong Prediction: {}'.format(len(y\_pred) - true))

print('Accuracy: {}'.format(true/len(y\_pred)\*100))

loaded\_model = load\_model('models/LSTM.h5')

review = str(input('Movie Review: '))

# Pre-process input

regex = re.compile(r'[^a-zA-Z\s]')

review = regex.sub('', review)

print('Cleaned: ', review)

words = review.split(' ')

filtered = [w for w in words if w not in english\_stops]

filtered = ' '.join(filtered)

filtered = [filtered.lower()]

print('Filtered: ', filtered)

tokenize\_words = token.texts\_to\_sequences(filtered)

tokenize\_words = pad\_sequences(tokenize\_words, maxlen=max\_length, padding='post', truncating='post')

print(tokenize\_words)

result = loaded\_model.predict(tokenize\_words)

print(result)

if result >= 0.7:

    print('positive')

else:

    print('negative')

**Conclusion**

Using LSTM or GRU networks for sentiment analysis demonstrates the power of deep learning in understanding and classifying textual data. This assignment highlights the importance of sequence models in NLP tasks.

# Assignment 7

**7. Object Detection Using YOLO and Pretrained Model**

**Title**

**Object Detection Using YOLO and Pretrained Models**

**Aim:**

To implement an object detection system using the YOLO (You Only Look Once) algorithm with pretrained models.

**Objective**

* To understand the principles of object detection using deep learning.
* To develop an application that detects and classifies objects in real-time.
* To evaluate the performance of the YOLO model.

**Theory**

YOLO is a state-of-the-art object detection algorithm that uses a single neural network to predict bounding boxes and class probabilities directly from full images in real-time.

**Advantages**

* **Real-Time Performance**: Fast and efficient for real-time applications.
* **Single Network**: Processes images in a single

**INPUT**:

!pip install ultralytics

from IPython.display import display, Javascript

from google.colab.output import eval\_js

from google.colab.patches import cv2\_imshow

import cv2

from ultralytics import YOLO

import numpy as np

import base64

import io

from PIL import Image

# JavaScript code for capturing images

js\_code = '''

function initCamera() {

    return new Promise((resolve, reject) => {

        const video = document.createElement('video');

        video.style.display = 'none';

        document.body.appendChild(video);

        const streamPromise = navigator.mediaDevices.getUserMedia({video: true});

        streamPromise.then((stream) => {

            video.srcObject = stream;

            video.onloadedmetadata = () => {

                resolve(video);

            };

            video.play();

        }).catch((error) => {

            reject(error);

        });

    });

}

async function takePhoto() {

    const video = await initCamera();

    const canvas = document.createElement('canvas');

    canvas.width = video.videoWidth;

    canvas.height = video.videoHeight;

    const context = canvas.getContext('2d');

    context.drawImage(video, 0, 0, canvas.width, canvas.height);

    const img = canvas.toDataURL('image/jpeg');

    return img;

}

'''

# Execute JavaScript code

display(Javascript(js\_code))

# Function to convert base64 image to OpenCV format

def js\_to\_image(js\_reply):

    image\_bytes = base64.b64decode(js\_reply.split(',')[1])

    image\_PIL = Image.open(io.BytesIO(image\_bytes))

    image\_np = np.array(image\_PIL)

    frame = cv2.cvtColor(image\_np, cv2.COLOR\_RGB2BGR)

    return frame

# Load the YOLO model

yolo = YOLO('yolov8s.pt')

# Function to get class colors

def getColours(cls\_num):

    base\_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255)]

    color\_index = cls\_num % len(base\_colors)

    increments = [(1, -2, 1), (-2, 1, -1), (1, -1, 2)]

    color = [base\_colors[color\_index][i] + increments[color\_index][i] \*

    (cls\_num // len(base\_colors)) % 256 for i in range(3)]

    return tuple(color)

while True:

    # Capture image from the webcam

    js\_reply = eval\_js('takePhoto()')

    frame = js\_to\_image(js\_reply)

    if frame is None:

        continue

    results = yolo.track(frame, stream=True)

    for result in results:

        # get the classes names

        classes\_names = result.names

        # iterate over each box

        for box in result.boxes:

            # check if confidence is greater than 40 percent

            if box.conf[0] > 0.4:

                # get coordinates

                [x1, y1, x2, y2] = box.xyxy[0]

                # convert to int

                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                # get the class

                cls = int(box.cls[0])

                # get the class name

                class\_name = classes\_names[cls]

                # get the respective colour

                colour = getColours(cls)

                # draw the rectangle

                cv2.rectangle(frame, (x1, y1), (x2, y2), colour, 2)

                # put the class name and confidence on the image

                cv2.putText(frame, f'{classes\_names[int(box.cls[0])]} {box.conf[0]:.2f}', (x1, y1), cv2.FONT\_HERSHEY\_SIMPLEX, 1, colour, 2)

    # show the image

    cv2\_imshow(frame)

    # break the loop if 'q' is pressed

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

# release the video capture and destroy all windows

cv2.destroyAllWindows()

Top of Form

Bottom of Form