VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Name: G. Bhuvan Sai
Roll No: 22071A6621

WEEK 1: Implement single layer feed forward network for binary classification.

A Single-Layer Feedforward Network (SLFN) for binary classification is a neural network with one input layer and one output layer (without hidden layers). It directly maps input features to output predictions using an activation function like sigmoid to classify data into two categories (0 or 1).

Program

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import
train_test_split from sklearn.preprocessing
import StandardScaler
from tensorflow.keras.utils import to_categorical

# Generate synthetic dataset
np.random.seed(42)
X = np.random.rand(300, 3) # 300 samples, 3 features
y = np.random.randint(0, 3, 300) # 3 classes (0, 1, 2)

# One-hot encode the labels
y = to_categorical(y, num_classes=3)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)

# Build multilayer feedforward neural
network model = keras.Sequential([
    keras.layers.Dense(16, input_shape=(3,), activation='relu'), #
    Hidden Layer 1 keras.layers.Dense(8, activation='relu'),      #
```

```
    Hidden Layer 2
    keras.layers.Dense(3, activation='softmax')          # Output Layer
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model with fewer epochs
model.fit(X_train, y_train, epochs=20, batch_size=4, verbose=1) # Only 20 epochs

# Evaluate the model
loss, accuracy = model.evaluate(X_test,
y_test) print(f"Test Accuracy:
{accuracy:.2f}")
```

Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
  `input_shape`/`input_dim` argumen super().__init__(activity_regularizer=activity_regularizer, **kwargs)


60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3257 - loss: 1.1148
Epoch 2/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
4ms/step - accuracy: 0.3122 - loss: 1.1036 Epoch 3/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
5ms/step - accuracy: 0.3650 - loss: 1.0939 Epoch 4/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.4338 - loss: 1.0891
Epoch 5/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s
6ms/step - accuracy: 0.4360 - loss: 1.0747 Epoch 6/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s
6ms/step - accuracy: 0.3779 - loss: 1.0843 Epoch 7/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
4ms/step - accuracy: 0.3612 - loss: 1.0889 Epoch 8/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4295 - loss: 1.0737
Epoch 9/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s
13ms/step - accuracy: 0.3901 - loss: 1.0843 Epoch 10/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s
11ms/step - accuracy: 0.4151 - loss: 1.0774 Epoch 11/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s
12ms/step - accuracy: 0.4387 - loss: 1.0682 Epoch 12/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4258 - loss: 1.0526
Epoch 13/20
```

60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s**

3ms/step - accuracy: 0.4287 - loss: 1.0496 Epoch 14/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **1s** 9ms/step - accuracy: 0.3913 - loss: 1.0921
Epoch 15/20

60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **1s**
6ms/step - accuracy: 0.4017 - loss: 1.0739 Epoch 16/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **1s**
7ms/step - accuracy: 0.4193 - loss: 1.0456 Epoch 17/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **1s**
6ms/step - accuracy: 0.4731 - loss: 1.0393 Epoch 18/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 4ms/step - accuracy: 0.4265 - loss: 1.0669
Epoch 19/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s**
6ms/step - accuracy: 0.3656 - loss: 1.0664 Epoch 20/20
60/60 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - accuracy: 0.4600 - loss: 1.0410 **2/2**
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s**
25ms/step - accuracy: 0.4125 - loss: 1.1091 Test Accuracy:
0.40

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Name: G. Bhuvan Sai
Roll No: 22071A6621

WEEK 2: Implement a multilayer feed-forward network for binary classification

A Multilayer Feedforward Network (MLFN) for binary classification is a neural network with an input layer, one or more hidden layers, and an output layer. It uses activation functions (e.g., ReLU, Sigmoid) to learn patterns from data and classify it into two categories (0 or 1)

Program

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import
train_test_split from sklearn.preprocessing
import StandardScaler

# Generate synthetic dataset
np.random.seed(42)
X = np.random.rand(200, 2) # 200 samples, 2 features
y = (X[:, 0] + X[:, 1] > 1).astype(int) # Label 1 if sum > 1, else 0

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)

# Build multilayer feedforward neural
network model =keras.Sequential([
    keras.layers.Dense(8, input_shape=(2,), activation='relu'), # Hidden Layer 1 (8 neurons, ReLU
    activation) keras.layers.Dense(4, activation='relu'), # Hidden Layer 2 (4 neurons, ReLU activation)
    keras.layers.Dense(1, activation='sigmoid') # Output Layer (1 neuron, Sigmoid activation)
])
```

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model

    model.fit(X_train, y_train, epochs=50, batch_size=4,
verbose=1) # Evaluate the model
loss, accuracy = model.evaluate(X_test,
y_test) print(f"Test Accuracy:
{accuracy:.2f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
      super(). init (activity_regularizer=activity_regularizer, **kwargs)
    Epoch 1/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s
    4ms/step - accuracy: 0.5316 - loss: 0.7070 Epoch 2/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    4ms/step - accuracy: 0.5462 - loss: 0.6006 Epoch 3/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    3ms/step - accuracy: 0.5449 - loss: 0.5731 Epoch 4/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5573 - loss: 0.5543
    Epoch 5/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    4ms/step - accuracy: 0.8477 - loss: 0.4737 Epoch 6/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9204 - loss: 0.4513
    Epoch 7/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    3ms/step - accuracy: 0.9174 - loss: 0.4255 Epoch 8/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.8645 - loss: 0.4410 Epoch 9/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9175 - loss: 0.4214
    Epoch 10/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.9260 - loss: 0.4153 Epoch 11/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.9426 - loss: 0.3924 Epoch 12/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.9197 - loss: 0.3800 Epoch 13/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9016 - loss: 0.3764
    Epoch 14/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.9318 - loss: 0.3991 Epoch 15/50
    40/40 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
    2ms/step - accuracy: 0.9611 - loss: 0.3316 Epoch 16/50
```

40/40 ——— **0s** 3ms/step - accuracy: 0.9622 - loss: 0.3617
Epoch 17/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9894 - loss: 0.3266 Epoch 18/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9502 - loss: 0.3469 Epoch 19/50
40/40 ——— **0s**

2ms/step - accuracy: 0.9569 - loss: 0.3313 Epoch 20/50
40/40 ——— **0s** 3ms/step - accuracy: 0.9779 - loss: 0.2818
Epoch 21/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9720 - loss: 0.2740 Epoch 22/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9651 - loss: 0.2984 Epoch 23/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9934 - loss: 0.2917 Epoch 24/50
40/40 ——— **0s** 2ms/step - accuracy: 0.9915 - loss: 0.2740
Epoch 25/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9935 - loss: 0.3088 Epoch 26/50
40/40 ——— **0s**
2ms/step - accuracy: 0.9882 - loss: 0.2684 Epoch 27/50
40/40 ——— **0s** 2ms/step - accuracy: 0.9901 - loss: 0.2678

Name: G. Bhuvan Sai
Roll No: 22071A6621

WEEK 3: Implement a multilayer feed-forward network for multi-class classification

A Multilayer Feedforward Network (MLFN) for multi-class classification is a neural network with multiple layers that classifies input data into more than two categories. It uses softmax activation in the output layer to predict probabilities for multiple classes.

Program

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import
train_test_split from sklearn.preprocessing
import StandardScaler
from tensorflow.keras.utils import
to_categorical # Generate synthetic
dataset
np.random.seed(42)
X = np.random.rand(300, 3) # 300 samples, 3 features
y = np.random.randint(0, 3, 300) # 3
classes (0, 1, 2) # One-hot encode the
labels
y = to_categorical(y,
num_classes=3) # Split data
into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=42) # Standardize the data
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test =
scaler.transform(X_test)
# Build multilayer feedforward neural
network model = keras.Sequential([
keras.layers.Dense(16, input_shape=(3,),
activation='relu'), # Hidden Layer 1
keras.layers.Dense(8,
activation='relu'), # Hidden
```

```
Layer 2
keras.layers.Dense(3, activation='softmax')

# Output ⎵ Layer (Softmax for multi-

class)])

# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',metrics=['accuracy']) # Train the
model with fewer epochs
model.fit(X_train, y_train, epochs=20,
batch_size=4, verbose=1) # Only 20 epochs
# Evaluate the model
loss, accuracy = model.evaluate(X_test,
y_test) print(f"Test Accuracy:
{accuracy:.2f}")
```

Output

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
  `input_shape`/`input_dim` argumen super().__init__(activity_regularizer=activity_regularizer, **kwargs)
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3233 - loss: 1.1458
Epoch 2/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
2ms/step - accuracy: 0.3188 - loss: 1.1098 Epoch 3/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
2ms/step - accuracy: 0.3516 - loss: 1.0981 Epoch 4/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
2ms/step - accuracy: 0.3527 - loss: 1.0945 Epoch 5/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3037 - loss: 1.0894
Epoch 6/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
2ms/step - accuracy: 0.4000 - loss: 1.0925 Epoch 7/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
2ms/step - accuracy: 0.3850 - loss: 1.0913 Epoch 8/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3359 - loss: 1.0954
Epoch 9/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
Epoch 16/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
3ms/step - accuracy: 0.4123 - loss: 1.0644 Epoch 17/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s
3ms/step - accuracy: 0.4428 - loss: 1.0651 Epoch 18/20
```

60/60 ——————————————————— **0s**
3ms/step - accuracy: 0.4944 - loss: 1.0593 Epoch 19/20
60/60 ——————————————————— **0s** 3ms/step - accuracy: 0.4587 - loss: 1.0526
Epoch 20/20
60/60 ——————————————————— **0s** 3ms/step - accuracy: 0.4635 - loss: 1.0587 **2/2**

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Azadi Ka Amrit Mahotsav

Name: G. Bhuvan Sai
Roll No: 22071A6621

Week-4 : Implement backpropagation algorithm

Backpropagation is an algorithm used to train neural networks by adjusting weights based on the error. It works in two main steps:

Forward Propagation – Computes the predicted output.

Backward Propagation – Calculates the error and updates weights using gradi- ent descent to minimize the error

Program

```
import numpy as np

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize dataset (X as input, y as expected output)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])      # XOR problem

# Set random seed for reproducibility
np.random.seed(42)

# Initialize weights and biases
input_layer_neurons = 2
hidden_layer_neurons = 4
output_layer_neurons = 1

weights_input_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_layer_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))
bias_output = np.random.uniform(size=(1, output_layer_neurons))
lr = 0.5, epochs = 10000
for epoch in range(epochs):
 # Forward propagation
 hidden_input = np.dot(X, weights_input_hidden) + bias_hidden hidden_output
 = sigmoid(hidden_input)
```

```
final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
final_output = sigmoid(final_input)
# Compute error
error = y - final_output

# Backpropagation

d_output = error * sigmoid_derivative(final_output) error_hidden =
d_output.dot(weights_hidden_output.T) d_hidden = error_hidden *
sigmoid_derivative(hidden_output)

# Update weights and biases
weights_hidden_output += hidden_output.T.dot(d_output) * lr weights_input_hidden +=
X.T.dot(d_hidden) * lr
bias_output += np.sum(d_output, axis=0, keepdims=True) * lr
bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * lr


# Print error every 1000 epochs
        if epoch % 1000 == 0:
            print(f'Epoch {epoch}, Error: {np.mean(np.abs(error))}')

  # Output final predictions
  y_pred = sigmoid(np.dot(sigmoid(np.dot(X, weights_input_hidden) + bias_hidden),␣
    sweights_hidden_output) + bias_output)
print("Final Predictions:") print(y_pred)
```

OUTPUT

```
Epoch 0, Error: 0.49721892454240146
Epoch 1000, Error: 0.12916864787356297
Epoch 2000, Error: 0.04947848250016216
Epoch 3000, Error: 0.03434239725474404
Epoch 4000, Error: 0.027490045054573684
Epoch 5000, Error: 0.023443831433154382
Epoch 6000, Error: 0.020717994486819137
Epoch 7000, Error: 0.018730483840957563
Epoch 8000, Error: 0.017202734017324094
Epoch 9000, Error: 0.01598320849044888
Final Predictions:
[[0.016172]
 [0.98334387]
 [0.98758925]
 [0.01468812]]
```

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Name: G. Bhuvan Sai
Roll No: 22071A6621

WEEK5–6: Implementationofreal-world case study- Amazon Food Review (or) Personalized Cancer Diagnosis

1.1 1. Amazon Food Review– Sentiment Analysis

[3]: pip install nltk

```
Collecting nltk
Downloading nltk-3.9.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: click in
c:\users\ramum\appdata\local\programs\python\python312\lib\site-packages (from nltk)
(8.1.7) Requirement already satisfied: joblib in
c:\users\ramum\appdata\local\programs\python\python312\lib\site-packages (from nltk)
(1.4.2) Collecting regex>=2021.8.3 (from nltk)
Downloading regex-2024.11.6-cp312-cp312-win_amd64.whl.metadata (41 kB)
---------------------------- 0.0/41.5 kB ? eta -:--:--
---------------- - 41.5/41.5 kB 978.3 kB/s eta 0:00:00
Collecting tqdm (from nltk)
Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
---------------------------- 0.0/57.7 kB ? eta -:--:--
---------------------------- - 57.7/57.7 kB 3.0 MB/s eta 0:00:00
Requirement already satisfied: colorama in
c:\users\ramum\appdata\local\programs\python\python312\lib\site-packages (from click->nltk)
(0.4.6) Downloading nltk-3.9.1-py3-none-any.whl (1.5 MB)
---------------------------- 0.0/1.5 MB ? eta -:--:--
---------------------------- - 0.5/1.5 MB 14.2 MB/s eta 0:00:01
---------------------------------------- 0.5/1.5 MB 5.9 MB/s eta 0:00:01
---------------------------------------- 0.8/1.5 MB 6.4 MB/s eta 0:00:01
---------------------------------------- 1.1/1.5 MB 6.3 MB/s eta 0:00:01
---------------------------------------- 1.1/1.5 MB 5.1 MB/s eta 0:00:01
---------------------------------------- 1.3/1.5 MB 5.3 MB/s eta 0:00:01
---------------------------------------- 1.5/1.5 MB 5.0 MB/s eta 0:00:00
Downloading regex-2024.11.6-cp312-cp312-win_amd64.whl (273 kB)
---------------------------- 0.0/273.6 kB ? eta -:--:--
---------------------------- - 273.6/273.6 kB 5.6 MB/s eta 0:00:00
Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
---------------------------- 0.0/78.5 kB ? eta -:--:--
---------------------------- 78.5/78.5 kB ? eta 0:00:00
Installing collected packages: tqdm, regex, nltk
Successfully installed nltk-3.9.1 regex-2024.11.6 tqdm-
4.67.1 Note: you may need to restart the kernel to use
```

updated packages. [notice] A new release of pip is
available: 24.0-> 25.0.1
[notice] To update, run: python.exe-m pip install--upgrade pip [4]:

```
import nltk
nltk.download('stopwo
rds')
```

[nltk_data] Downloading package stopwords to
[nltk_data]
C:\Users\ramum\AppData\Roaming\nltk_data…
[nltk_data] Unzipping corpora\stopwords.zip.

[4]: True

```
[6]: import pandas
as pd import
numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
TfidfVectorizer from sklearn.linear_model import
LogisticRegression  from sklearn.metrics import
accuracy_score
# Load dataset (change path accordingly)
df = pd.read_csv("C:\\Users\\ramum\\Downloads\\amazonFood.csv") # Ensure you ␣ ↪ have
the dataset # Select required columns
df = df[['Text', 'Score']]
# Preprocessing function # Convert ratings to binary labels (positive: 4,5; negative:
1,2,3) df['Sentiment'] = df['Score'].apply(lambda x: 1 if x > 3 else 0)
nltk.download('stopwords')
stop_words =
set(stopwords.words('english')) def
preprocess_text(text):
        text = re.sub(r'[^a-zA-Z]', ' ', text) # Remove special
        characters text = text.lower() # Convert to lowercase
        text = text.split() # Tokenization 2
        text = [word for word in text if word not in stop_words] # Remove
        stopwords return " ".join(text)
df['Cleaned_Text'] =
df['Text'].apply(preprocess_text) # Split
dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(df['Cleaned_Text'], df['Sentiment'], test_size=0.2,
random_state=42)
# Convert text to TF-IDF features
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)

  X_test_tfidf = vectorizer.transform(X_test)
# Train Logistic Regression
model model =
LogisticRegression()
model.fit(X_train_tfidf,
y_train)

  # Predictions y_pred =
model.predict(X_test_tfidf) # Accuracy
accuracy = accuracy_score(y_test,
y_pred) print(f"Model Accuracy:
{accuracy:.2f}")

[nltk_data] Downloading package stopwords to
[nltk_data]
C:\Users\ramum\AppData\Roaming\nltk_data…
[nltk_data] Package stopwords is already up-to-
date!

Model Accuracy: 0.87
```

1.2 2. Personalized Cancer Diagnosis– Classification

```
[7]: import numpy as
np import pandas as
pd
from
sklearn.model_selecti
on import
train_test_split
from sklearn.ensemble
import RandomForestClassifier
from sklearn.metrics import
accuracy_score from sklearn.datasets
import load_breast_cancer # Load
dataset data = load_breast_cancer()
```

```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target # 0: Malignant, 1: Benign
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(df. drop(columns=['Target']), df['Target'], test_size=0.2,
random_state=42)


# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predictions
y_pred =
model.predict(X_test) #
Accuracy
accuracy = accuracy_score(y_test,
y_pred) print(f"Model Accuracy:
{accuracy:.2f}")
```

Model Accuracy: 0.96

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Name: G. Bhuvan Sai
Roll No: 22071A6621

Week - 7 Write a program to implement an Artificial Neural network to classify MNIST dataset

Program

```
import tensorflow as tf
from tensorflow import keras import
 numpy as np
import matplotlib.pyplot as plt
# Load the MNIST dataset
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from HYPERLINK "https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz" \h
https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
    11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Normalize the images to a range of 0 to 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Build the neural network model model
 = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)), # Flatten the 28x28 images to a 1D array
keras.layers.Dense(128, activation='relu'), # Fully connected layer with 128 neurons and ReLU activation
    keras.layers.Dropout(0.2), # Dropout layer to reduce overfitting
keras.layers.Dense(10, activation='softmax') # Output layer with 10 neurons for classification
 ])
  /usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
      `input_shape`/`input_ super().init(**kwargs)
# Compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
 val_accuracy: 0.9789 - val_loss: 0.0743
    <keras.src.callbacks.history.History at 0x79ce5ff52010>
```

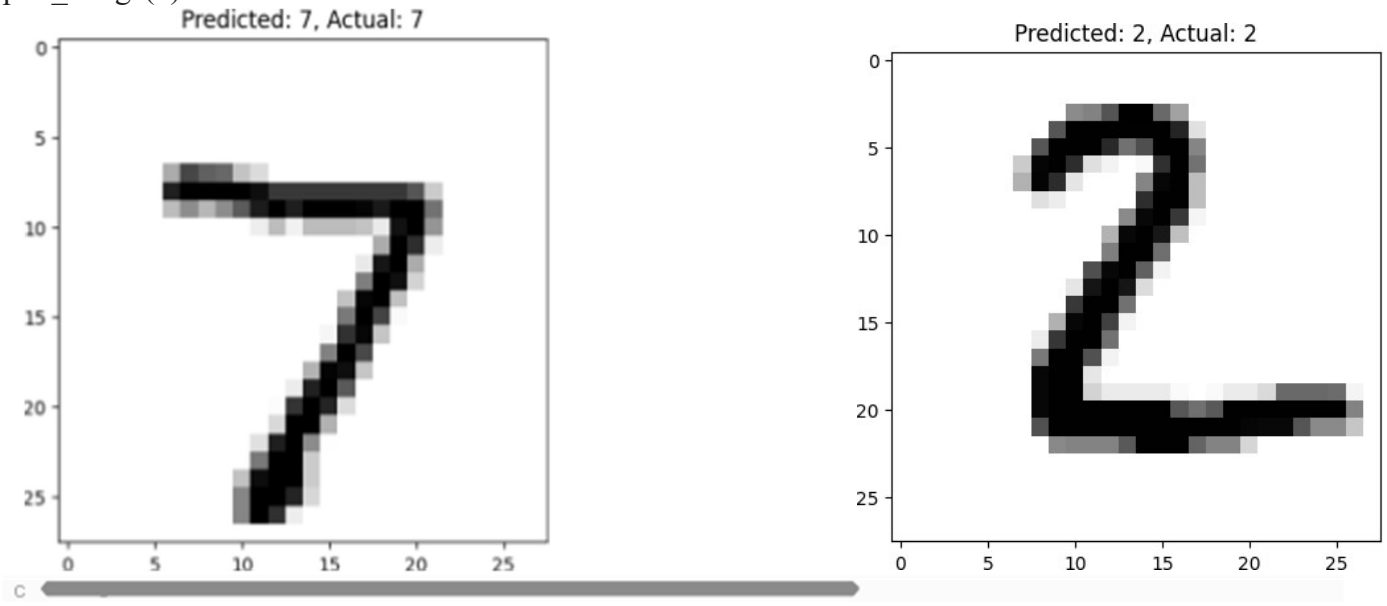
```
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test,
y_test) print(f'\nTest accuracy: {test_acc:.4f}')
```

```
313/313 ———————————————————— 1s
    2ms/step - accuracy: 0.9753 - loss: 0.0896 Test accuracy: 0.978
```

```
# Predict on test images
predictions = model.predict(x_test)
```

```
313/313 ———————————————————— 1s 1ms/step
```

```
# Function to display a test image and predicted
label def plot_image(index):
    plt.imshow(x_test[index], cmap=plt.cm.binary)
    plt.title(f'Predicted: {np.argmax(predictions[index])}, Actual:
    {y_test[index]}') plt.show()
# Display a sample image and prediction
plot_image(0)
plot_image(1)
```



VALLURUPALLI NAGESWARA RAO VIGNANA JYOTH
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" G
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programs
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,C
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Name; G. Bhuvan Sai
Roll No: 22071A6621

Week-8 Demonstration and installation of Python, PyTorch, and other libraries in Google Colaboratory

Cloning GitHub repository, Upload Data, Importing Kaggle's dataset, Basic File operations

Step 1: Install Required Libraries

```
!pip install torch torchvision torchaudio # Install PyTorch
!pip install opencv-python pandas kaggle # Install OpenCV, Pandas, and Kaggle
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages
(0.21.0+cu124) Requirement already satisfied: torchaudio in
/usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages
(from torch) (4.12.2) Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages
(from torch) (12.4.127) Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127 Requirement already satisfied:
nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from
torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from
torch) (12.4.5.8) Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
/usr/local/lib/python3.11/dist-packages (from torch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages
(from torch) (10.3.5.147) Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages
(from torch) (12.3.1.170
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from
torch) (0.6.2) Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from
torch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages
```

```
/usr/local/lib/python3.11/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from
torchvision) (11.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from
jinja2->torch) (3.0.2) Requirement already satisfied: opencv-python in
/usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from
opencv-python) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages
(from pandas) (2.8.2) Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas)
(2025.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from
kaggle) (2025.1.31) Requirement already satisfied: charset-normalizer in
/usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages
(from kaggle) (8.0.4) Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle)
(75.1.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages
(from kaggle) (1.3) Requirement already satisfied: tqdm in
/usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from
kaggle) (2.3.0) Requirement already satisfied: webencodings in
/usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
```

Step 2: Verify Installation

```
import torch
import cv2
import pandas as pd

print("PyTorch Version:", torch.__version__)
print("OpenCV Version:", cv2.__version__)
print("Pandas Version:", pd.__version__)
```

```
PyTorch Version:
2.6.0+cu124 OpenCV
Version: 4.11.0
Pandas Version: 2.2.2
```

Step 3: Clone a GitHub Repository

```
!git clone https://github.com/pytorch/examples.git
```

```
Cloning into 'examples'...
remote: Enumerating objects: 4353, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 4353 (delta 15), reused 5 (delta 5), pack-reused
4321 (from 3) Receiving objects: 100% (4353/4353), 41.39
MiB | 18.22 MiB/s, done.
Resolving deltas: 100% (2163/2163), done.
```

Step 4: Change Directory to Cloned Repo

```
%cd examples
```

```
/content/examples/examples
```

Step 5: Upload a File Manually

```
from google.colab import files
uploaded = files.upload() # Opens file upload dialog
```

```
Choose Files Week 7.pdf
   Week 7.pdf(application/pdf) - 119906 bytes, last modified: 27/3/2025 - 100% done
Saving Week 7.pdf to Week 7.pdf
```

Step 6: Mount Google Drive for File Access

```
from google.colab import
drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Step 7: Import a Kaggle Dataset - Upload kaggle.json (API Key)

```
# Configure Kaggle API
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
mv: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
# Download a Sample Dataset
!kaggle datasets download -d ankurzingy/cifar10

# Extract the Dataset
import zipfile
with zipfile.ZipFile(&amp;quot;cifar10.zip&amp;quot;, &amp;#39;r&amp;#39;) as
zip_ref: zip_ref.extractall(&amp;quot;cifar10&amp;quot;)
```

Step 8: Basic File Operations

```
# List Files in Directory
import os
print(os.listdir('/content'))
```

```
['.config', 'examples', 'drive', 'sample_data']
```

```
# Create and Write to a File
with open("sample.txt", "w") as f:
    f.write("Hello, this is a sample file.")


# Read File Content
with open("sample.txt", "r") as f:
    print(f.read())
# Delete a File
import os
os.remove("sample.txt")
```

```
Hello, this is a sample file.
```

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTH
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with "A++" G
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE,PE,AMS,SWE Programi
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 2022 :113th Rank - Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,C
Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
☎+91-40-2304 2758/59/60, +91-40-23042761 postbox@vnrvjiet.ac.in www.vnrvjiet.ac.in

Azadi Ka Amrit Mahotsav

Week – 9- Convolution Neural Network application using PyTorch - Classification CIFAR10 Dataset using CNN

Name: G. Bhuvan Sai
Roll No: 22071A6621

Step 1: Install & Import Dependencies

```
# Install PyTorch if not already installed
!pip install torch torchvision
```

```
                                        24.6/24.6 MB 42.3 MB/s eta 0:00:00
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
                                        883.7/883.7 kB 30.7 MB/s eta 0:00:00
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
                                        664.8/664.8 MB 2.5 MB/s eta 0:00:00
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
                                        211.5/211.5 MB 5.9 MB/s eta 0:00:00
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
                                        56.3/56.3 MB 12.0 MB/s eta 0:00:00
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
                                        127.9/127.9 MB 7.8 MB/s eta 0:00:00
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
                                        207.5/207.5 MB 5.6 MB/s eta 0:00:00
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
```

21.1/21.1 MB 38.7 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cud Attempting uninstall: nvidia-nvjitlink-cu12
Found existing installation: nvidia-nvjitlink-cu12
12.5.82 Uninstalling nvidia-nvjitlink-cu12-12.5.82:
Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
Attempting uninstall: nvidia-curand-cu12
Found existing installation: nvidia-curand-cu12
10.3.6.82 Uninstalling
nvidia-curand-cu12-10.3.6.82:
Successfully uninstalled
nvidia-curand-cu12-10.3.6.82 Attempting uninstall:
nvidia-cufft-cu12
Found existing installation: nvidia-cufft-cu12
11.2.3.61 Uninstalling nvidia-cufft-cu12-11.2.3.61:
Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
Attempting uninstall: nvidia-cuda-runtime-cu12
Found existing installation: nvidia-cuda-runtime-cu12
12.5.82 Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
Successfully uninstalled
nvidia-cuda-runtime-cu12-12.5.82 Attempting
uninstall: nvidia-cuda-nvrtc-cu12
Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
Successfully uninstalled
nvidia-cuda-nvrtc-cu12-12.5.82 Attempting uninstall:
nvidia-cuda-cupti-cu12
Found existing installation: nvidia-cuda-cupti-cu12
12.5.82 Uninstalling
nvidia-cuda-cupti-cu12-12.5.82:
Successfully uninstalled
nvidia-cuda-cupti-cu12-12.5.82 Attempting uninstall:
nvidia-cublas-cu12
Found existing installation: nvidia-cublas-cu12 12.5.3.2
Uninstalling nvidia-cublas-cu12-12.5.3.2:
Successfully uninstalled
nvidia-cublas-cu12-12.5.3.2 Attempting uninstall:
nvidia-cusparse-cu12
Found existing installation: nvidia-cusparse-cu12
12.5.1.3 Uninstalling nvidia-cusparse-cu12-12.5.1.3:
Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
Attempting uninstall: nvidia-cudnn-cu12
Found existing installation: nvidia-cudnn-cu12
9.3.0.75 Uninstalling nvidia-cudnn-cu12-9.3.0.75:

```
    Successfully uninstalled
  nvidia-cudnn-cu12-9.3.0.75 Attempting uninstall:
  nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12
    11.6.3.83 Uninstalling
    nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127
nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-ru
```

```
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
```

Step 2: Load & Preprocess CIFAR-10

```
# Define transformations (convert to tensor &
 normalize) transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # Normalize images
 ])
# Download CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
 transform=transform) trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
 shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
 transform=transform) testloader = torch.utils.data.DataLoader(testset, batch_size=64,
 shuffle=False)
# Define class labels
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
100%|██████████| 170M/170M [00:02<00:00, 61.5MB/s]
```

Step 3: Define the CNN Model

```
# Define CNN
architecture class
CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3,
        padding=1) self.conv2 = nn.Conv2d(32, 64,
        kernel_size=3, padding=1) self.conv3 =
        nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2) # Max Pooling
        self.fc1 = nn.Linear(128 * 4 * 4, 256)
        self.fc2 = nn.Linear(256, 10) # Output 10 classes
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5) # Prevent overfitting

    def forward(self, x):
        x =
        self.pool(self.relu(self.conv1(x)))
        x =
        self.pool(self.relu(self.conv2(x)))
        x =
        self.pool(self.relu(self.conv3(x)))
        x = x.view(-1, 128 * 4 * 4) #
        Flatten x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Instantiate model
model = CNN()
```

Step 4: Define Loss Function & Optimizer

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Step 5: Train the CNN Model

```
num_epochs = 10
for epoch in
    range(num_epochs):
    running_loss = 0.0
```

```
    for images, labels in trainloader:
        optimizer.zero_grad() # Clear gradients
        outputs = model(images)
        loss = criterion(outputs, labels) # Compute loss
        loss.backward() # Backpropagation
        optimizer.step() # Update weights

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss:

{running_loss/len(trainloader):.4f}") print("Training Complete!")
```

Epoch [1/10], Loss: 1.4879 Epoch [2/10], Loss: 1.0885 Epoch [3/10],

Loss: 0.9180 Epoch [4/10], Loss: 0.8020 Epoch [5/10], Loss: 0.7179 Epoch

[6/10], Loss: 0.6506 Epoch [7/10], Loss: 0.5925 Epoch [8/10], Loss: 0.5407

Epoch [9/10], Loss: 0.4986 Epoch [10/10], Loss: 0.4533 Training Complete

Step 6: Evaluate the Model

```
correct = 0
total = 0

with torch.no_grad(): # No gradient needed for
    testing for images, labels in testloader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1) # Get predicted
        class total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy:

    {accuracy:.2f}%") Test
```

Accuracy: 74.61

Step 7: Make Predictions

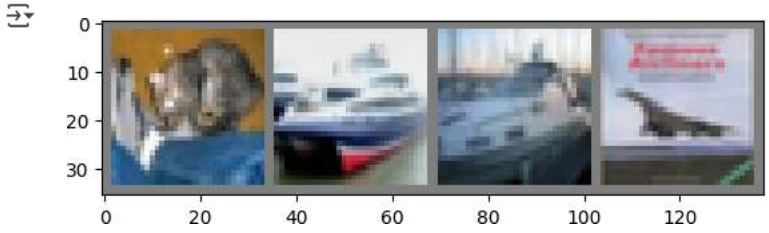
```
import matplotlib.pyplot as
plt import numpy as np

# Function to show
images def
imshow(img):
    img = img / 2 + 0.5 #
    Unnormalize npimg =
    img.numpy()
    plt.imshow(np.transpose(npimg, (1,
    2, 0))) plt.show()

# Get some test images
dataiter = iter(testloader)
images, labels = next(dataiter)

# Predict
outputs = model(images)
_, predicted = torch.max(outputs, 1)

# Show images with predictions
imshow(torchvision.utils.make_grid(images[:4]))
print("Predicted:", [classes[predicted[j]] for j in range(4)])
print("Actual:", [classes[labels[j]] for j in range(4)])
```




```
Predicted: ['dog', 'ship', 'ship', 'ship']
Actual: ['cat', 'ship', 'ship', 'plane']
```