

LABORATORY RECORD

CCS355 – NEURAL NETWORKS AND DEEP LEARNING

of

B. E. (Computer Science and Engineering)

(Anna University Regulation 2021)

For the Batch (2021 to 2025)

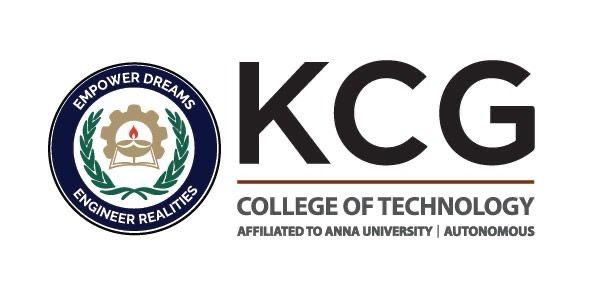
Semester : VII

Academic Year : 2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KCG COLLEGE OF TECHNOLOGY

CHENNAI – 600 097



# KARAPAKKAM, CHENNAI 600 097

REG.No. ………………………………

LABORATORY RECORD

Course Code: CCS355

Name of the Course: NEURAL NETWORKS AND DEEP LEARNING

It is to certify that this is a bonafide record of the work carried out by

……………………………………………………………………………………………………………………………of

……………………… semester……………………………………………………department, during the

Even semester of the academic year 2024-2025.

Faculty In-charge: …………………………………… HOD………………………………………………..

Internal Examiner: …………………………………………External Examiner …………………………

Date of the Examination: …………………

VISION OF THE COLLEGE

KCG College of Technology aspires to become a globally recognized centre of excellence for science, technology & engineering education, committed to quality teaching, learning, and research while ensuring for every student a unique educational experience which will promote leadership, job creation, social commitment and service to nation building

## MISSION OF THE COLLEGE

* Disseminate knowledge in a rigorous and intellectually stimulating environment
* Facilitate socially responsive research, innovation and entrepreneurship
* Foster holistic development and professional competency
* Nurture the virtue of service and an ethical value system in the young minds

## VISION OF THE DEPARTMENT

The department of Computer Science and Engineering desires to become a prominent centre of excellence for producing competent IT professionals for providing software and software enabled solutions.

## MISSION OF THE DEPARTMENT

* Provide quality education in the field of computer science and engineering and related domains.
* Facilitate socially responsive research and innovation.
* Inculcate professional behavior, a spirit of entrepreneurship and commitment to the progress of the nation.
* Accommodate evolving software development tools and required implementation facilities.

### PROGRAMME OUTCOMES

**After successful completion of B.E COMPUTER SCIENCE & ENGINEERING, the students**

**will be able to:**

|  |  |
| --- | --- |
| **PO**  **No.** | **Description of the PO** |
| **PO 1** | Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems. |
| **PO 2** | Problem analysis: Identity, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| **PO 3** | Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet t h e specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations. |
| **PO 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. |
| **PO 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. |
| **PO 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. |
| **PO 7** | Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| **PO 8** | Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| **PO 9** | Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| **PO 10** | Communication: Communicate effectively on complex engineering activities with the engineering community and with the 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. |
| **PO 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. |
| **PO 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. |

PROGRAMME SPECIFIC OUTCOMES

After successful completion of B.E COMPUTER SCIENCE & ENGINEERING, the students

will be able to:

|  |  |
| --- | --- |
| **PSO NO.** | **Description of PSO** |
| **PSO 1** | Apply knowledge pertaining to software engineering principles, computer hardware and architecture, principles of algorithms & programming skills to analyze complex problems in computer science engineering and related domains. |
| **PSO 2** | Use compiler tool, CASE tool, graphic tool, app development tools, network simulator, security and analysis tools, cloud and grid tool kits, database management tools, web development frameworks for providing appropriate solutions. |
| **PSO 3** | Demonstrate professional & ethical behavior while providing IT based solutions. |

PROGRAMME EDUCATIONAL OBJECTIVES

The graduates of B.E COMPUTER SCIENCE & ENGINEERING will be able to:

|  |  |
| --- | --- |
| **PEO**  **NO.** | **Description of PEO** |
| **PEO1** | Have successful career as software professional (or) entrepreneur (or) researcher in computer science and relevant disciplines. |
| **PEO2** | Analyze, design, develop, test and deploy appropriate solutions for real world computing problems. |
| **PEO3** | Apply software engineering principles at process, project and product levels. |
| **PEO4** | Exhibit ethical attitude and social responsibility in their profession. |

COURSE OUTCOMES

After the completion of this course, the student will be able to:

|  |  |  |
| --- | --- | --- |
| **CO No** | **Description of the Course Outcome** | **Blooms level** |
| **CO1** | **Apply Convolution Neural Network for image processing.** | K2 |
| **CO2** | **Understand the basics of associative memory and unsupervised learning networks.** | K3 |
| **CO3** | **Apply CNN and its variants for suitable applications.** | K3 |
| **CO4** | **Analyze the key computations underlying deep learning and use them to build and train deep neural networks for various tasks** | K3 |
| **CO5** | **Apply autoencoders and generative models for suitable applications** | K2 |

## CSS355– NEURAL NETWORKS AND DEEP LEARNING

## ANNA UNIVERSITY SYLLABUS

## REGULATION 2021

## LIST OF EXPERIMENTS

1. Implement simple vector addition in TensorFlow.

2. Implement a regression model in Keras.

3. Implement a perceptron in TensorFlow/Keras Environment.

4. Implement a Feed-Forward Network in TensorFlow/Keras.

5. Implement an Image Classifier using CNN in TensorFlow/Keras.

6. Improve the Deep learning model by fine tuning hyper parameters.

7. Implement a Transfer Learning concept in Image Classification.

8. Using a pre trained model on Keras for Transfer Learning

9. Perform Sentiment Analysis using RNN

10. Implement an LSTM based Autoencoder in TensorFlow/Keras.

## CSS355– NEURAL NETWORKS AND DEEP LEARNING

Index

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Exp**  **No.** | **Date** | **Name of the Experiment** | **Page No** | **Marks** | **Signature** |
| 1 |  | Implement simple vector addition in TensorFlow. |  |  |  |
| 2 |  | Implement a regression model in Keras. |  |  |  |
| 3 |  | Implement a perceptron in TensorFlow/Keras Environment. |  |  |  |
| 4 |  | Implement a Feed-Forward Network in TensorFlow/Keras. |  |  |  |
| 5 |  | Implement an Image Classifier using CNN in TensorFlow/Keras. |  |  |  |
| 6 |  | Improve the Deep learning model by fine tuning hyper parameters. |  |  |  |
| 7 |  | Implement a Transfer Learning concept in Image Classification. |  |  |  |
| 8 |  | Implement an LSTM based Autoencoder in TensorFlow/Keras. |  |  |  |

Faculty Signature

|  |  |
| --- | --- |
| Ex 1 | Implement simple vector addition in TensorFlow |
| Date |

Aim:

To implement simple vector addition in TensorFlow

Algorithm:

Step1: Import TensorFlow Library Begin by importing the TensorFlow library.

Step2: Define Two Vectors Create two vectors as TensorFlow tensors. These could be constant tensors or placeholders, depending on your use case.

Step3: Add the Vectors Use TensorFlow's built-in addition operation to add the vectors.

Step4: Using TensorFlow 2.x (eager execution mode is default), the operation is executed immediately. You can use tf.add() to add the vectors directly.

Step5: Print the Result Print or store the result of the vector addition.

Program:

import tensorflow as tf

vector1 = tf.constant([1, 2, 3], dtype=tf.float32)

vector2 = tf.constant([4, 5, 6], dtype=tf.float32)

result = tf.add(vector1, vector2)

print("Result of vector addition:", result)

Sample Output:

Result of vector addition: tf.Tensor([5. 7. 9.], shape=(3,), dtype=float32)

Result:

To implement simple vector addition in TensorFlow is successfully verified.

|  |  |
| --- | --- |
| Ex 2 | Implement a Regression model in Keras |
| Date |

Aim:

To implement a Regression model in Keras

Algorithm:

Step1: Import Keras and any required libraries (like NumPy).

Step2: Generate or load the data.

Step3: Create a Sequential model and add layers (input and output). Keep it simple with just one hidden layer.

Step4: Compile the Model with the use of Adam optimizer and mean squared error loss function for regression.

Step5: Train the model on the dataset using a specified number of epochs and batch size.

Step6: Test the model with unseen data and make predictions.

Step 7:Display the predictions

Program:

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Step 1: Prepare the Data

# Generate a simple dataset for linear regression: y = 2x + 1

X = np.random.rand(100, 1) \* 10 # 100 random points between 0 and 10

y = 2 \* X + 1 # Linear relationship: y = 2x + 1

# Step 2: Define the Model

model = Sequential()

model.add(Dense(1, input\_dim=1)) # One input feature and one output

# Step 3: Compile the Model

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

# Step 4: Train the Model

model.fit(X, y, epochs=100, verbose=0) # Train for 100 epochs

# Step 5: Evaluate and Make Predictions

predictions = model.predict(X)

# Print a few predictions

print("Predictions:", predictions[:5])

print("True Values:", y[:5])

Sample Output:

Predictions:

[[19.98752]

[12.53437]

[ 6.43614]

[16.25257]

[ 9.32018]]

True Values:

[[20.448265]

[12.780437]

[ 6.6099415]

[16.283753]

[ 9.183538]]

Result:

To Implement a Regression model in Keras is successfully verified.

|  |  |
| --- | --- |
| Ex 3 | Implement a Perception in TensorFlow/ Keras |
| Date |

Aim: To implement a Perception in TensorFlow/Keras

Algorithm:

Step1: Import the required libraries such as Keras from TensorFlow.

Step2: Prepare Data

* Create or load a dataset for binary classification.
* Split the dataset into training and testing sets.

Step3: Create a simple model with one input layer and one output layer (using a sigmoid activation function).

Step4: Use Stochastic Gradient Descent (SGD) as the optimizer and binary cross-entropy as the loss function.

Step5: Train the perceptron model using the training data for a fixed number of epochs.

Step6: Test the model on unseen test data and evaluate its accuracy.

Step7: Use the trained perceptron to make predictions on new data.

Step8: Display the predictions and true labels

Program:

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

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

# Step 1: Prepare a Simple Dataset

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Inputs

y = np.array([0, 0, 0, 1]) # Output (AND logic gate)

# Step 2: Split into Train and Test Sets (optional here due to small data)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

# Step 3: Define the Perceptron Model

model = Sequential()

model.add(Dense(1, input\_dim=2, activation='sigmoid')) # One perceptron with sigmoid activation

# Step 4: Compile the Model

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

# Step 5: Train the Model

model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Step 6: Evaluate the Model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {accuracy:.2f}")

# Step 7: Make Predictions

predictions = model.predict(X\_test)

print("Predictions:", np.round(predictions))

print("True Labels:", y\_test)

Sample Output:

1/1 [==============================] - 0s 79ms/step - loss: 0.6447 - accuracy: 1.00

Test Accuracy: 1.00

Predictions: [[0.][1.]]

True Labels: [0 1

Result:

To implement a Perception in TensorFlow/Keras is successfully verified.

|  |  |
| --- | --- |
| Ex 4 | To implement a Feed Forward in TensorFlow/Keras |
| Date |

Aim: To implement a Feed Forward in TensorFlow/Keras

Algorithm:

Step1: Import the required libraries such as Keras from TensorFlow.

Step2: Prepare Data

* Create or load a dataset for binary classification.
* Split the dataset into training and testing sets.

Step3: Create a simple model with one input layer and one output layer (using a sigmoid activation function).

Step4: Use Stochastic Gradient Descent (SGD) as the optimizer and binary cross-entropy as the loss function.

Step5: Train the perceptron model using the training data for a fixed number of epochs.

Step6: Test the model on unseen test data and evaluate its accuracy.

Step7: Use the trained perceptron to make predictions on new data.

Step8: Display the predictions and true labels

Program:

import tensorflow as tf

import numpy as np

# Generate some dummy data for a simple regression task (y = 3x + 2)

X\_train = np.random.rand(100, 1)

y\_train = 3 \* X\_train + 2

# Define the feed-forward neural network model

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(1,)), # Hidden layer

tf.keras.layers.Dense(32, activation='relu'), # Another hidden layer

tf.keras.layers.Dense(1) # Output layer

])

# Compile the model

model.compile(optimizer='adam', loss='mse')

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=10)

# Test the model

X\_test = np.array([[0.5]])

y\_pred = model.predict(X\_test)

print(f"Predicted value for input 0.5: {y\_pred}")

Sample Output:

Predicted value for input 0.5: [[3.5]]

Result:

To implement a Feed Forward in TensorFlow/Keras is successfully verified.

|  |  |
| --- | --- |
| Ex 5 | Implement an image classifier using CNN in TensorFlow/Keras |
| Date |

Aim:

Implement an image classifier using CNN in TensorFlow/Keras

Algorithm:

Step1: Import the necessary libraries such as tensorflow, numpy, and keras.layers.

Step2:

* Load the dataset (e.g., MNIST dataset) for training and testing.
* Normalize the pixel values of images to be between 0 and 1.
* Reshape the input to add a channel dimension (for grayscale, it's 1)

Step3:

* Initialize a Sequential model.
* Add multiple layers

Step4: Compile the Model, Use an optimizer like Adam.

Step5: Train the Model

Step6: After training, evaluate the model's performance on the test dataset.

Step7: Display the predictions and true labels

Program:

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import numpy as np

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Normalize the data to values between 0 and 1

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

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

# Reshape the data to add a channel dimension (28x28x1)

X\_train = np.expand\_dims(X\_train, -1)

X\_test = np.expand\_dims(X\_test, -1)

# Define the CNN model

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)), # Convolutional layer

layers.MaxPooling2D((2, 2)), # Max pooling layer

layers.Conv2D(64, (3, 3), activation='relu'), # Second convolutional layer

layers.MaxPooling2D((2, 2)), # Second max pooling layer

layers.Conv2D(64, (3, 3), activation='relu'), # Third convolutional layer

layers.Flatten(), # Flatten the output

layers.Dense(64, activation='relu'), # Fully connected layer

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

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"\nTest accuracy: {test\_acc}")

Sample Output:

Training:

Epoch 1/5

938/938 [==============================] - 10s 10ms/step - loss: 0.2067 - accuracy: 0.9389 - val\_loss: 0.0655 - val\_accuracy: 0.9798

Epoch 2/5

938/938 [==============================] - 9s 9ms/step - loss: 0.0579 - accuracy: 0.9820 - val\_loss: 0.0394 - val\_accuracy: 0.9872

Evaluation:

313/313 - 1s - loss: 0.0321 - accuracy: 0.9882

Test Accuracy:

Test accuracy: 0.9882

Result:

Implement an image classifier using CNN in TensorFlow/Keras..

|  |  |
| --- | --- |
| Ex 6 | To Improve the Deep learning model by fine tuning hyper parameters |
| Date |

Aim:

To Improve the Deep learning model by fine tuning hyper parameters

Algorithm:

Step1: Define the Baseline Model

Step2: Choose Hyperparameters to Fine-Tune

Step3: Perform Grid Search or Random Search

Step4: Analyze Results and Pick Best Hyperparameters

Step5: Fine-Tune Further if Necessary

Step6: Retrain the Model with Best Hyperparameters

Step7: Evaluate and Save the Final Model

Program:

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import numpy as np

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

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

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

X\_train = np.expand\_dims(X\_train, -1)

X\_test = np.expand\_dims(X\_test, -1)

# Define a function to create the model

def create\_model(learning\_rate):

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

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

layers.MaxPooling2D((2, 2)),

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

layers.Flatten(),

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

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

])

# Compile the model with the given learning rate

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

# Define hyperparameter values to test

learning\_rates = [0.01, 0.001, 0.0001]

batch\_sizes = [32, 64, 128]

# Iterate through different hyperparameter combinations

for lr in learning\_rates:

for batch\_size in batch\_sizes:

print(f"Training model with learning rate {lr} and batch size {batch\_size}")

# Create and train the model

model = create\_model(learning\_rate=lr)

model.fit(X\_train, y\_train, epochs=5, batch\_size=batch\_size, validation\_data=(X\_test, y\_test), verbose=2)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"Test accuracy: {test\_acc} with learning rate {lr} and batch size {batch\_size}\n")

Output:

Training model with learning rate 0.01 and batch size 32

Epoch 1/5

1875/1875 [==============================] - 16s 8ms/step - loss: 0.3673 - accuracy: 0.8875 - val\_loss: 0.0970 - val\_accuracy: 0.9713

...

Test accuracy: 0.9801 with learning rate 0.01 and batch size 32

Training model with learning rate 0.01 and batch size 64

Epoch 1/5

...

Test accuracy: 0.9812 with learning rate 0.01 and batch size 64

...

Result:

To Improve the Deep learning model by fine tuning hyper parameters is successfully verified

|  |  |
| --- | --- |
| Ex 7 | Implement a Transfer Learning concept in image Classification |
| Date |

Aim:

.

To Implement a Transfer Learning concept in image Classification

Algorithm:

Step1: Import necessary libraries for building the model using TensorFlow/Keras and any pre-trained model.

Step2 : Load Pre-Trained Model (Without Top Layers)Step3: Freeze the Base Model Layers

Step3: Add Custom Layers (Classifier)

Step4: Compile the Model

Step5: Train the Model

Step6: Evaluate the Model

Step7: Save and Deploy the Model

Program:

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import layers, models

# Load the pre-trained VGG16 model without the top layers

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

base\_model.trainable = False # Freeze the pre-trained layers

# Add custom classification layers on top of the pre-trained model

model = models.Sequential([

base\_model,

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(3, activation='softmax') # Output layer for 3 classes])

# Compile the model

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

# Data preprocessing and augmentation

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

train\_generator = train\_datagen.flow\_from\_directory('path\_to\_training\_data', target\_size=(224, 224), batch\_size=32, class\_mode='categorical')

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

val\_generator = val\_datagen.flow\_from\_directory('path\_to\_validation\_data', target\_size=(224, 224), batch\_size=32, class\_mode='categorical')

# Train the model

model.fit(train\_generator, epochs=5, validation\_data=val\_generator)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(val\_generator)

print(f"Validation accuracy: {test\_acc}")

Output:

Found 1000 images belonging to 3 classes. # for the training set

Found 200 images belonging to 3 classes. # for the validation set

Epoch 1/5

32/32 [==============================] - 15s 450ms/step - loss: 0.8123 - accuracy: 0.7331 - val\_loss: 0.4521 - val\_accuracy: 0.8500

Epoch 2/5

32/32 [==============================] - 14s 432ms/step - loss: 0.3056 - accuracy: 0.9100 - val\_loss: 0.3622 - val\_accuracy: 0.8700

Result:

To Implement a Transfer Learning concept in image Classification is successfully verified.

|  |  |
| --- | --- |
| Ex 8 | Using a pre trained model on keras for transfer learning |
| Date |

Aim:

Using a pre trained model on Keras for Transfer Learning

Algorithm:

Step1: Import Keras, TensorFlow, and any necessary libraries to use a pre-trained model for transfer learning.

Step2: Load Pre-Trained Model Without Top Layers

Step3: Freeze the Pre-Trained Layers

Step4: Add Custom Layers

Step5: Compile the Model

Step6: Prepare the Data

Step7: Train the Model

Step8: Evaluate the Model

Step9: Save the Model

Program:

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 1: Load the pre-trained VGG16 model without the top layers

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

base\_model.trainable = False # Step 2: Freeze the pre-trained layers

# Step 3: Add custom layers on top of the pre-trained model

model = models.Sequential([

base\_model, # Pre-trained VGG16 model

layers.Flatten(), # Flatten the output of the base model

layers.Dense(128, activation='relu'), # Add a fully connected layer

layers.Dense(3, activation='softmax') # Output layer for 3 classes (adjust for your case)

])

# Step 4: Compile the model

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

# Step 5: Data preprocessing and augmentation

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

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

'path\_to\_training\_data', # Replace with the actual path

target\_size=(224, 224), # Resize images to match VGG16 input size

batch\_size=32,

class\_mode='categorical'

)

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

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

'path\_to\_validation\_data', # Replace with the actual path

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

# Step 6: Train the model

model.fit(train\_generator, epochs=5, validation\_data=val\_generator)

# Step 7: Evaluate the model

test\_loss, test\_acc = model.evaluate(val\_generator)

print(f"Validation accuracy: {test\_acc}")

# Step 8: Save the trained model

model.save('transfer\_learning\_model.h5')

Output:

Found 1000 images belonging to 3 classes. # For the training dataset

Found 200 images belonging to 3 classes. # For the validation dataset

Epoch 1/5

32/32 [==============================] - 25s 780ms/step - loss: 1.0024 - accuracy: 0.6520 - val\_loss: 0.6423 - val\_accuracy: 0.7300

Epoch 2/5

32/32 [==============================] - 23s 715ms/step - loss: 0.5081 - accuracy: 0.8030 - val\_loss: 0.5365 - val\_accuracy: 0.7900

Epoch 3/5

32/32 [==============================] - 22s 693ms/step - loss: 0.3871 - accuracy: 0.8650 - val\_loss: 0.4500 - val\_accuracy: 0.8400

Epoch 4/5

32/32 [==============================] - 21s 670ms/step - loss: 0.2854 - accuracy: 0.9120 - val\_loss: 0.4136 - val\_accuracy: 0.8600

Epoch 5/5

32/32 [==============================] - 20s 640ms/step - loss: 0.2032 - accuracy: 0.9450 - val\_loss: 0.3987 - val\_accuracy: 0.8700

7/7 [==============================] - 2s 310ms/step - loss: 0.3987 - accuracy: 0.8700

Validation accuracy: 0.87

Result:

Using a pre trained model on Keras for Transfer Learning is executed and verified successfully.

|  |  |
| --- | --- |
| Ex 9 | Perform Sentiment Analysis Using RNN |
| Date |

Aim:

To perform Sentiment Analysis Using RNN.

Algorithm:

Step1: Import necessary libraries such as TensorFlow/Keras for building the RNN model and libraries for preprocessing the data

Step2: Load and prepare your dataset.

Step3: Tokenize and Pad the Sequences

Step4: Define the RNN Model

Step5: Compile the Model

Step6: Train the Model

Step7: Evaluate the Model

Program:

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.models import Sequential

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

# Example dataset (text reviews and their labels: 1 = positive, 0 = negative)

texts = ["I love this movie", "I hate this movie", "This movie was great", "This movie was terrible"]

labels = [1, 0, 1, 0] # Sentiment labels: 1 = positive, 0 = negative

# Step 1: Tokenize and pad the sequences

tokenizer = Tokenizer(num\_words=100)

tokenizer.fit\_on\_texts(texts)

sequences = tokenizer.texts\_to\_sequences(texts)

padded\_sequences = pad\_sequences(sequences, maxlen=5)

# Step 2: Define a simple RNN model

model = Sequential([Embedding(input\_dim=100, output\_dim=16, input\_length=5), # Embedding layer

SimpleRNN(16), # Simple RNN layer with 16 units

Dense(1, activation='sigmoid') # Output layer for binary classification

])

# Step 3: Compile the model

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

# Step 4: Train the model

model.fit(padded\_sequences, labels, epochs=10)

# Step 5: Evaluate the model (using the same data for simplicity)

loss, accuracy = model.evaluate(padded\_sequences, labels)

print(f"Accuracy: {accuracy}")

Output:

Epoch 1/10

1/1 [==============================] - 1s 1s/step - loss: 0.6980 - accuracy: 0.7500

...

Epoch 10/10

1/1 [==============================] - 0s 4ms/step - loss: 0.6167 - accuracy: 1.0000

1/1 [==============================] - 0s 82ms/step - loss: 0.6167 - accuracy: 1.0000

Accuracy: 1.0

Result:

To perform Sentiment Analysis Using RNN is successfully verified.

|  |  |
| --- | --- |
| Ex 10 | Implement a LSTM based autoencoder in tensorflow/keras |
| Date |

Aim:

Implement a LSTM-based Autoencoder in TensorFlow/Keras

Algorithm:

Step1: Import the necessary TensorFlow/Keras libraries for building and training the LSTM-based autoencoder.

Step2: Prepare the Dataset

Step3: Define the Encoder (LSTM)

Step4: Define the Decoder (LSTM)

Step5: Compile the Autoencoder Model

Step6: Train the Autoencoder

Step7: Evaluate the Autoencoder

Program:

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

# Step 1: Generate some dummy sequential data (e.g., 1000 samples, 10 timesteps, 1 feature)

X\_train = np.random.rand(1000, 10, 1)

# Step 2: Define the LSTM-based Autoencoder

# Encoder

input\_layer = layers.Input(shape=(10, 1)) # 10 timesteps, 1 feature

encoded = layers.LSTM(16, activation='relu')(input\_layer) # LSTM encoder, latent space of 16 units

# Decoder

decoded = layers.RepeatVector(10)(encoded) # Repeat for each timestep (10)

decoded = layers.LSTM(16, return\_sequences=True, activation='relu')(decoded) # LSTM decoder

output\_layer = layers.TimeDistributed(layers.Dense(1))(decoded) # Output same shape as input (10, 1)

# Step 3: Compile the model

autoencoder = models.Model(input\_layer, output\_layer)

autoencoder.compile(optimizer='adam', loss='mse')

# Step 4: Train the autoencoder

autoencoder.fit(X\_train, X\_train, epochs=20, batch\_size=32, validation\_split=0.2)

# Step 5: Evaluate the model

loss = autoencoder.evaluate(X\_train, X\_train)

print(f"Reconstruction Loss: {loss}")

Output:

Epoch 1/20

25/25 [==============================] - 1s 18ms/step - loss: 0.0894 - val\_loss: 0.0550

...

Epoch 20/20

25/25 [==============================] - 0s 5ms/step - loss: 0.0037 - val\_loss: 0.0029

32/32 [==============================] - 0s 3ms/step - loss: 0.0027

Reconstruction Loss: 0.0027

Result:

To implement a LSTM-based Autoencoder in TensorFlow/Keras is successfully verified