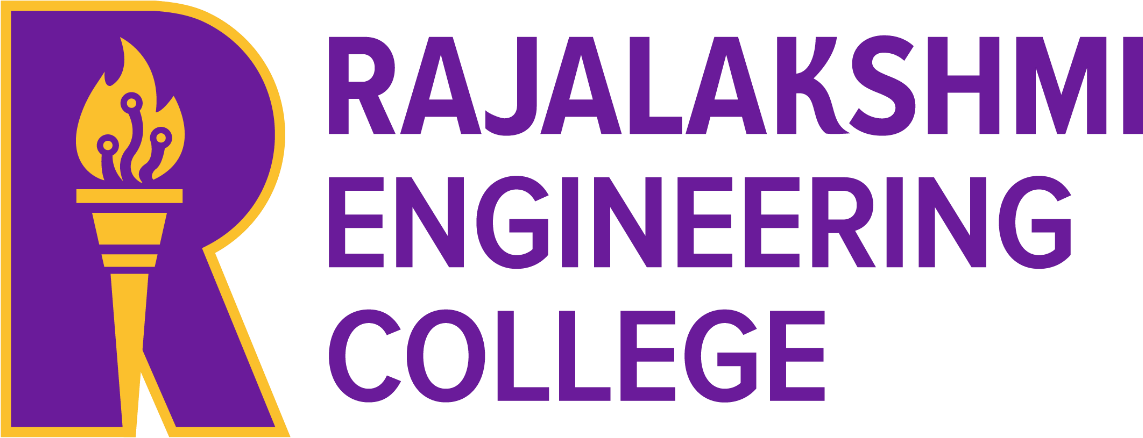
RAJALAKSHMI ENGINEERING COLLEGE

**(An Autonomous Institution)**

**RAJALAKSHMI NAGAR, THANDALAM- 602 105**



**CS19P18 - DEEP LEARNING CONCEPTS**

**LABORATORY RECORD NOTEBOOK**

**NAME:** SHAKTHI PRIYA V

**YEAR/SEMESTER:** IV / VII

**BRANCH:** COMPUTER SCIENCE AND ENGINEERING

**REGISTER NO:** 220701259

**COLLEGE ROLL NO:** 2116220701259  
 **ACADEMIC YEAR:** 2025 -2026

**RAJALAKSHMI ENGINEERING COLLEGE**

**(An Autonomous Institution)**

**RAJALAKSHMI NAGAR, THANDALAM- 602 105**

**BONAFIDE CERTIFICATE**

**NAME:** SHAKTHI PRIYA V

**BRANCH/SECTION:** COMPUTER SCIENCE AND ENGINEERING/D **ACADEMIC YEAR:** 2025 -2026

**SEMESTER:** SEVEN

**REGISTER NO:**

**220701259**

Certified that this is a Bonafide record of work done by the above student in the **CS19P18 - DEEP LEARNING CONCEPTS**

during the year 2025 - 2026

**Signature of Faculty In-charge**

**Submitted for the Practical Examination Held on: ………………………………………...**

**Internal Examiner External Examiner**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EX.NO** | **DATE** | **NAME OF THE EXPERIMENT** | **PAGE NO** | **STAFF SIGN** |
| 1 | 14/07/2025 | Create a neural network to recognize handwritten digits using MNIST dataset | 5 |  |
| 2 | 21/07/2025 | Build a Convolutional Neural Network with Keras/TensorFlow | 8 |  |
| 3 | 28/07/2025 | Image Classification on CIFAR-10 Dataset using CNN | 11 |  |
| 4 | 04/08/2025 | Transfer learning with CNN and Visualization | 13 |  |
| 5 | 25/08/2025 | Build a Recurrent Neural Network using Keras/Tensorflow | 17 |  |
| 6 | 15/09/2025 | Sentiment Classification of Text using RNN | 19 |  |
| 7 | 22/09/2025 | Build autoencoders with keras/tensorflow | 21 |  |
| 8 | 29/09/2025 | Object detection with yolo3 | 24 |  |
| 9 | 29/09/2025 | Build GAN with Keras/TensorFlow | 27 |  |
| 10 | 06/10/2025 | Mini Project | 31 |  |

# INSTALLATION AND CONFIGURATION OF TENSORFLOW

## Aim:

To install and configure TensorFlow in the anaconda environment in Windows 10.

## Procedure:

1. Download Anaconda Navigator and install.
2. Open Anaconda prompt
3. Create a new environment dlc with python 3.7 using the following command: conda create -n dlc python=3.7
4. Activate newly created environment dlc using the following command: conda activate dlc
5. In dlc prompt, install tensorflow using the following command: pip install tensorflow
6. Next install Tensorflow-datasets using the following command: pip install tensorflow-datasets
7. Install scikit-learn package using the following command: pip install scikit-learn
8. Install pandas package using the following command: pip install pandas
9. Lastly, install jupyter notebook pip install jupyter notebook
10. Open jupyter notebook by typing the following in dlc prompt: jupyter notebook
11. Click create new and then choose python 3 (ipykernel)
12. Give the name to the file
13. Type the code and click Run button to execute (eg. Type import tensorflow and then run)

# EX NO: 1 CREATE A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN

**DATE:14/07/2025**

# DIGITS USING MNIST DATASET

## Aim:

To build a handwritten digit’s recognition with MNIST dataset.

## Procedure:

1. Download and load the MNIST dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

import numpy as np import tensorflow as tf

from tensorflow import keras

from sklearn.datasets import make\_classification from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.metrics import accuracy\_score

# Generate a synthetic dataset

X, y = make\_classification(n\_samples=1000, n\_features=20, random\_state=42) # Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Standardize features (optional but often beneficial)

scaler = StandardScaler()

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

X\_test = scaler.transform(X\_test # Define the model

model = keras.Sequential([ keras.layers.Input(shape=(X\_train.shape[1],)), # Input layer

keras.layers.Dense(64, activation='relu'), # Hidden layer with 64 neurons and ReLU activation keras.layers.Dense(1, activation='sigmoid') # Output layer with 1 neuron and sigmoid activation

)

# Train the model

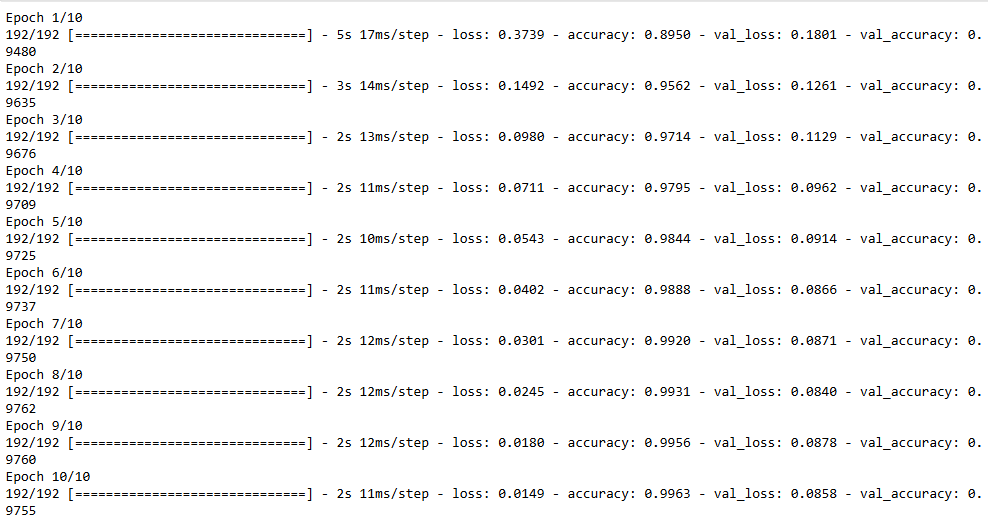
history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.1) # Evaluate the model on the test set

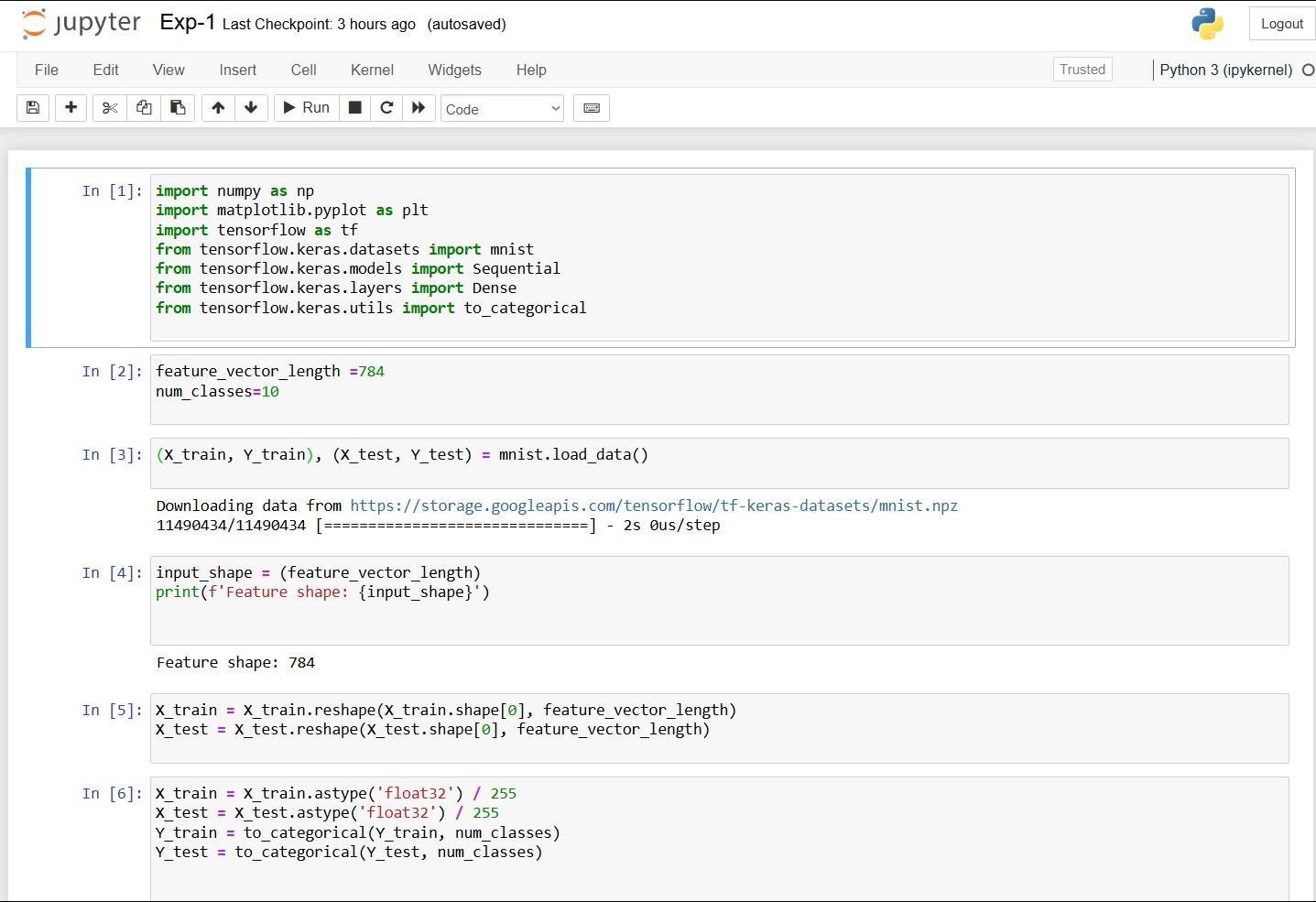
y\_pred = model.predict(X\_test) y\_pred\_classes = (y\_pred> 0.5).astype(int) # Calculate accuracy on the test set

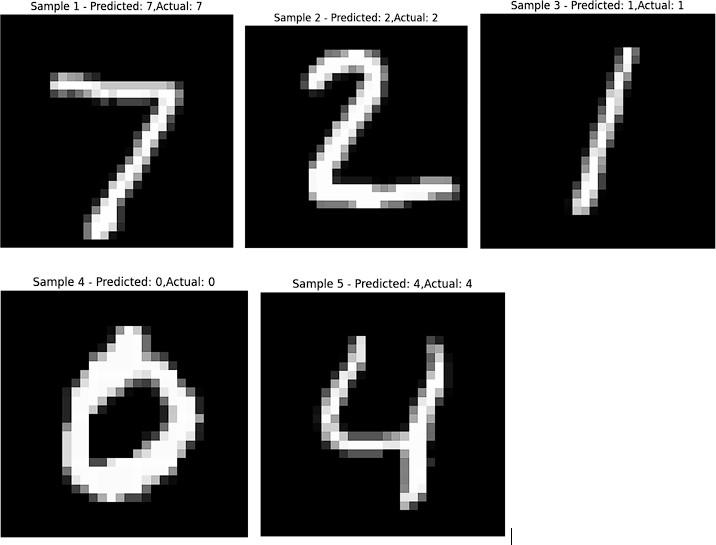
accuracy = accuracy\_score(y\_test, y\_pred\_classes) # Calculate test loss

test\_loss = model.evaluate(X\_test, y\_test) print(f"Test accuracy: {accuracy \* 100:.2f}%") print(f"Test loss: {test\_loss[0]:.4f}")

## Output:







**Result:**

Thus, the implementation to build a simple neural network using Keras/TensorFlow has been successfully executed.

# EX NO:2 BUILD A CONVOLUTIONAL NEURAL NETWORK

**DATE:21/07/2025 USING KERAS/TENSORFLOW**

## Aim:

To implement a Convolutional Neural Network (CNN) using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy.

## Procedure:

1. Import required libraries (TensorFlow/Keras, NumPy, etc.).
2. Load the MNIST dataset from Keras.
3. Normalize and reshape the image data.
4. Convert labels to one-hot encoded vectors.
5. Build a CNN model with Conv2D, MaxPooling, Flatten, and Dense layers.
6. Compile the model using categorical crossentropy and Adam optimizer.
7. Train the model on training data.
8. Evaluate the model on test data.
9. Display accuracy and predictions.

## Code:

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt import numpy as np

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data() train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

train\_images = train\_images.reshape(-1, 28, 28, 1)

test\_images = test\_images.reshape(-1, 28, 28, 1) model = Sequential([

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

MaxPooling2D((2, 2)),

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

MaxPooling2D((2, 2)), Flatten(),

Dense(64, activation='relu'), Dropout(0.5),

Dense(10, activation='softmax')

])

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

history = model.fit(train\_images, train\_labels, epochs=5,

batch\_size=64, validation\_split=0.2)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels) print(f"\n Test accuracy: {test\_acc:.4f}")

print(f" Test loss: {test\_loss:.4f}")

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

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o') plt.plot(history.history['val\_accuracy'], label='Validation Accuracy', marker='o') plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend() plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss', marker='o') plt.plot(history.history['val\_loss'], label='Validation Loss', marker='o') plt.title('Training and Validation Loss')

plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend() plt.grid(True) plt.tight\_layout() plt.show()

predictions = model.predict(test\_images) predicted\_labels = np.argmax(predictions, axis=1)

num\_samples = 10

plt.figure(figsize=(15, 4))

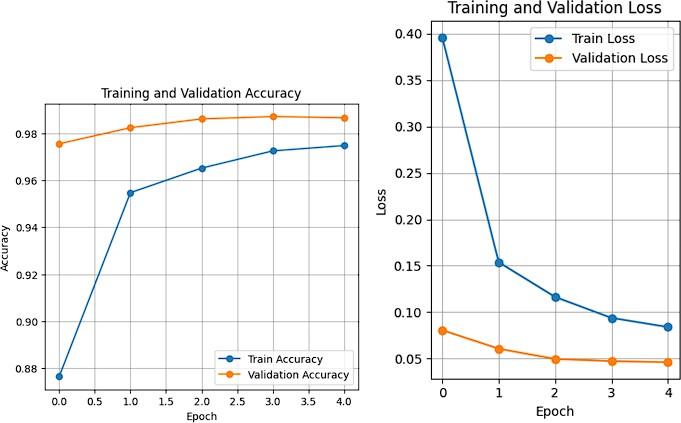
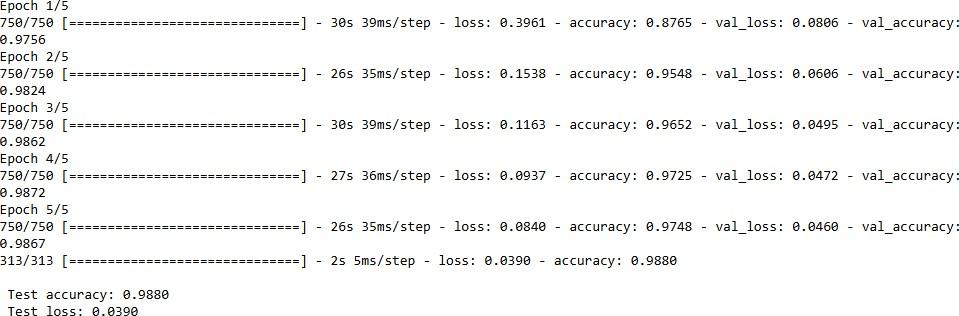
for i in range(num\_samples):

plt.subplot(1, num\_samples, 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.suptitle("Sample Predictions on Test Images", fontsize=16) plt.show()

## Output:



**Result:**

Thus, the Convolution Neural Network (CNN) using Keras / Tensorflow to recognize and classify handwritten digits from MNIST dataset has been implemented successfully.

# EX NO: 3 IMAGE CLASSIFICATION ON CIFAR-10 DATASET USING CNN

**DATE:28/07/2025**

## Aim:

To build a Convolutional Neural Network (CNN) model for classifying images from the CIFAR-10 dataset into one of the ten categories such as airplanes, cars, birds, cats, etc.

## Procedure:

1. Download and load the CIFAR-10 dataset using Keras/TensorFlow.
2. Visualize and analyze sample images from the dataset. 3, Preprocess the data:
   * Normalize the pixel values (divide by 255)
   * Convert class labels to one-hot encoded format
3. Build a CNN model using Keras/TensorFlow:
   * Include convolutional, pooling, flatten, and dense layers.
4. Compile the model with a suitable loss function and optimizer.
5. Train the model using training data and validate using test data.
6. Evaluate the model using accuracy and loss on the test dataset.
7. Perform predictions on new/unseen CIFAR-10 images.

9 Visualize prediction results with sample images and predicted labels.

## Code:

import tensorflow as tf import numpy as np

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data() x\_train = x\_train.astype('float32') / 255.0

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

y\_train = tf.keras.utils.to\_categorical(y\_train, 10) y\_test = tf.keras.utils.to\_categorical(y\_test, 10) model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(32,32,3))) model.add(tf.keras.layers.MaxPooling2D((2,2))) model.add(tf.keras.layers.Conv2D(64, (3,3), activation='relu')) model.add(tf.keras.layers.MaxPooling2D((2,2))) model.add(tf.keras.layers.Conv2D(64, (3,3), activation='relu')) model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(64, activation='relu')) model.add(tf.keras.layers.Dense(10, activation='softmax')) model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.2) class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

'dog', 'frog', 'horse', 'ship', 'truck']

index = int(input("Enter an index (0 to 9999) for test image: ")) if index < 0 or index >= len(x\_test):

print("Invalid index. Using index 0 by default.") index = 0

test\_image = x\_test[index]

true\_label = np.argmax(y\_test[index])

prediction = model.predict(np.expand\_dims(test\_image, axis=0)) predicted\_label = np.argmax(prediction)

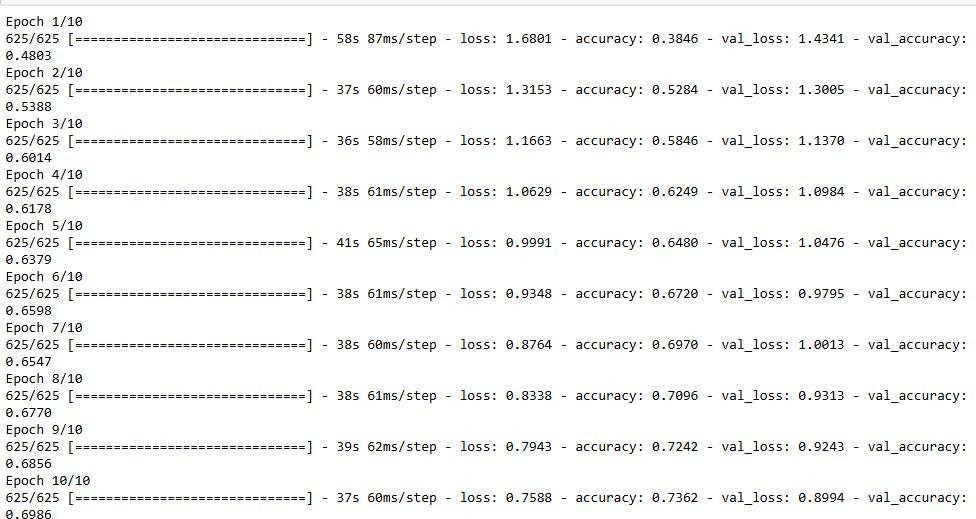
plt.figure(figsize=(4, 4))

resized\_image = tf.image.resize(test\_image, [128, 128]) plt.imshow(resized\_image)

plt.axis('off')

plt.title(f"Predicted: {class\_names[predicted\_label]}\nActual: {class\_names[true\_label]}") plt.show()

## Output:





**Result**

Thus, the Convolution Neural Network (CNN) model for classifying images from CIFAR-10 dataset is implemented successfully.

## Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION

**DATE:04/08/2025**

## Aim:

To build a convolutional neural network with transfer learning and perform visualization

## Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

conda install -c conda-forge python-graphviz -y import tensorflow as tf

from tensorflow.keras.applications import VGG16 from tensorflow.keras.models import Sequential

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

from tensorflow.keras.datasets import cifar10 from tensorflow.keras.utils import plot\_model import matplotlib.pyplot as plt

import numpy as np

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data() x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3)) for layer in vgg\_base.layers:

layer.trainable = False model = Sequential() model.add(vgg\_base) model.add(Flatten())

model.add(Dense(512, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(10, activation='softmax'))

model.compile(optimizer=Adam(learning\_rate=0.0001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

plot\_model(model, to\_file='cnn.png', show\_shapes=True,

show\_layer\_names=True, dpi=300) plt.figure(figsize=(20, 20))

img = plt.imread('cnn.png') plt.imshow(img) plt.axis('off')

plt.show()

history = model.fit(x\_train, y\_train, epochs=10,

batch\_size=32, validation\_split=0.2)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test) print(f'Test Loss: {test\_loss:.4f}')

print(f'Test Accuracy: {test\_acc \* 100:.2f}%') plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

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

plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.legend()

plt.subplot(1, 2, 2)

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

plt.xlabel('Epoch') plt.ylabel('Loss') plt.legend() plt.tight\_layout() plt.show()

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

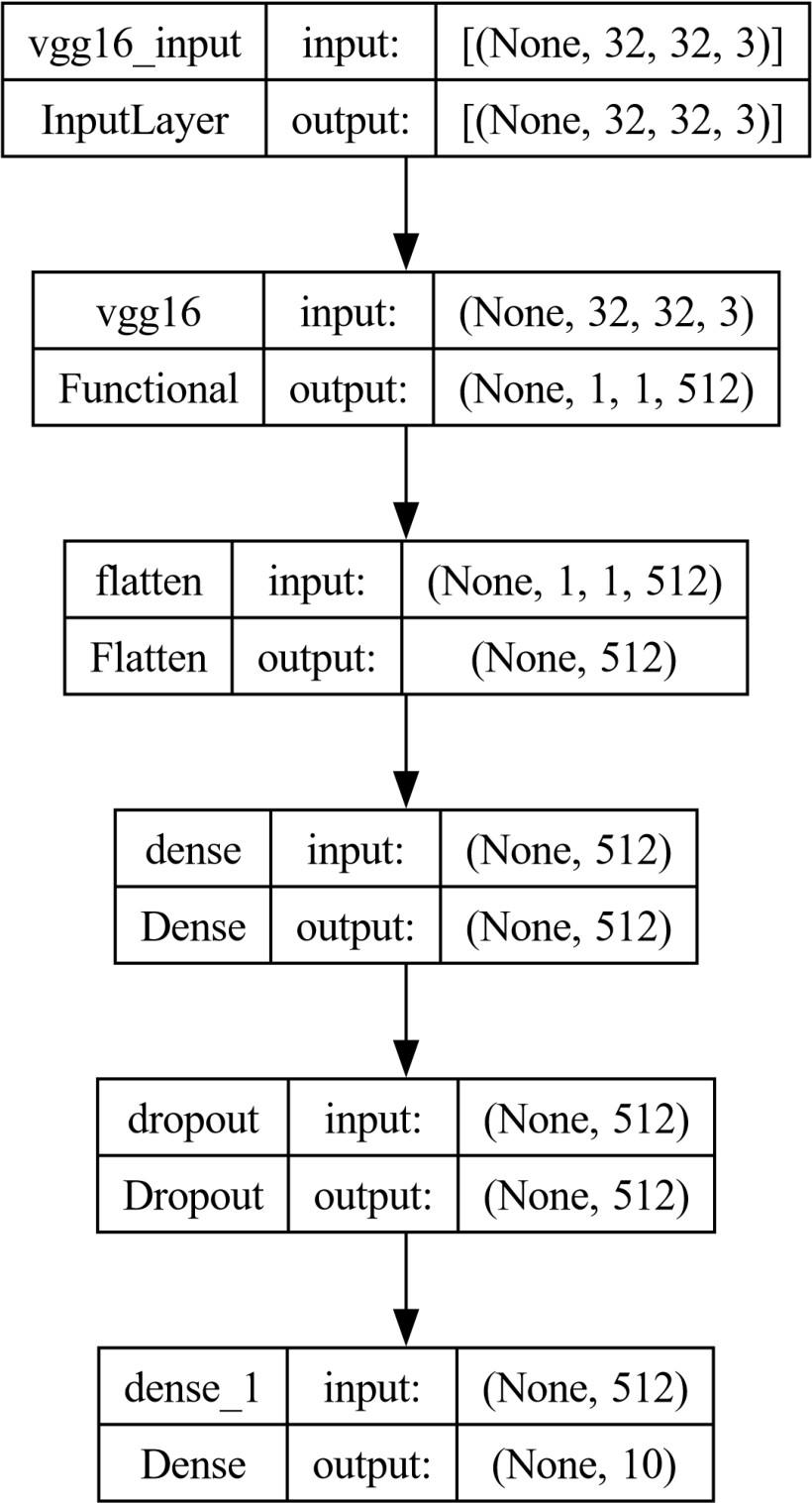
sample = x\_test[0].reshape(1, 32, 32, 3) prediction = model.predict(sample)

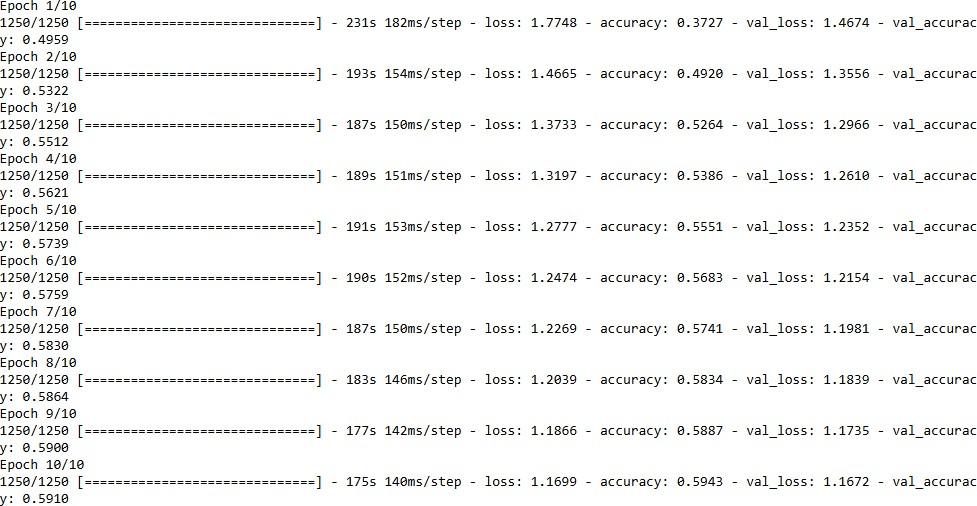
predicted\_class = class\_names[np.argmax(prediction)]

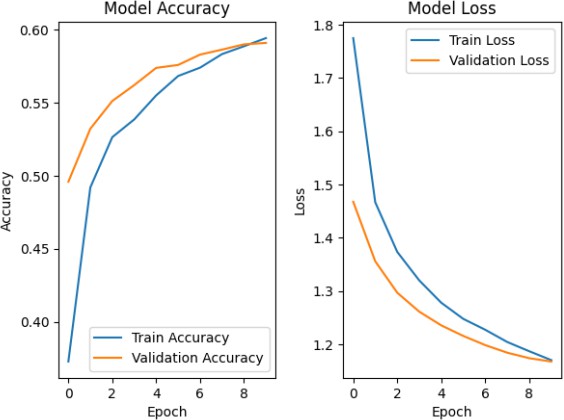
plt.imshow(x\_test[0]) plt.title(f"Predicted: {predicted\_class}") plt.axis('off')

plt.show()

## Output:







**Result**

Thus, the Convolution Neural Network (CNN) with transfer learning and perform visualization has been implemented successfully

# EX NO: 5 BUILD A RECURRENT NEURAL NETWORK (RNN) USING

**DATE:25/08/2025 KERAS/TENSORFLOW**

## Aim:

To build a recurrent neural network with Keras/TensorFlow.

## Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

import numpy as np import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense from sklearn.metrics import r2\_score np.random.seed(0)

seq\_length = 10

num\_samples = 1000

X = np.random.randn(num\_samples, seq\_length, 1)

y = X.sum(axis=1) + 0.1 \* np.random.randn(num\_samples, 1) split\_ratio = 0.8

split\_index = int(split\_ratio \* num\_samples) X\_train, X\_test = X[:split\_index], X[split\_index:] y\_train, y\_test = y[:split\_index], y[split\_index:] model = Sequential()

model.add(SimpleRNN(units=50, activation='relu', input\_shape=(seq\_length, 1))) model.add(Dense(units=1))

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

batch\_size = 30

epochs = 50 # Reduced epochs for quick demonstration history = model.fit(

X\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_split=0.2

)

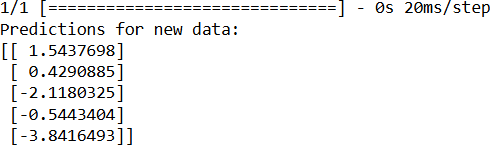
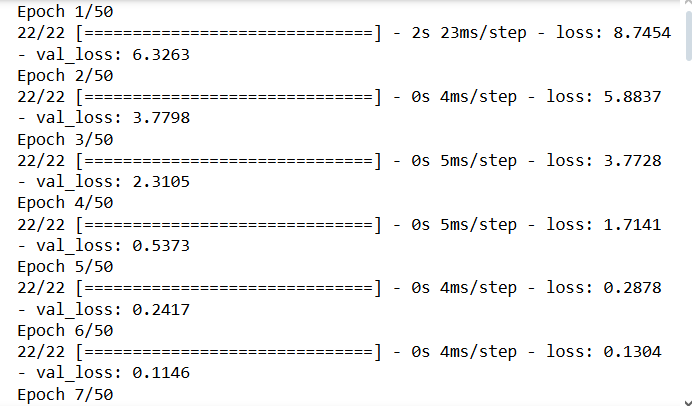
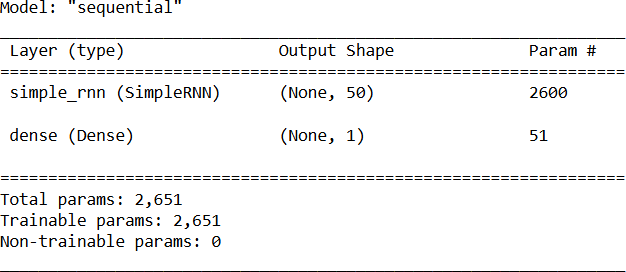
test\_loss = model.evaluate(X\_test, y\_test) print(f'Test Loss: {test\_loss:.4f}')

y\_pred = model.predict(X\_test) r2 = r2\_score(y\_test, y\_pred)

print(f'Test Accuracy (R^2): {r2:.4f}')

new\_data = np.random.randn(5, seq\_length, 1) predictions = model.predict(new\_data) print("Predictions for new data:") print(predictions)

## Output:



**Result:**

Thus, the Recurrent Neural Network (RNN) has been implemented using Tensorflow.

# EX NO: 6 SENTIMENT CLASSIFICATION OF TEXT USING RNN

**DATE:15/09/2025**

## Aim:

To implement a Recurrent Neural Network (RNN) using Keras/TensorFlow for classifying

the sentiment of text data (e.g., movie reviews) as positive or negative.

## Procedure:

1. Import necessary libraries.
2. Load and preprocess the text dataset (e.g., IMDb).
3. Pad sequences and prepare labels.
4. Build an RNN model with Embedding and SimpleRNN layers.
5. Compile the model with loss and optimizer.
6. Train the model on training data.
7. Evaluate the model on test data.
8. Predict sentiment for new inputs

## Code:

import numpy as np

from tensorflow.keras.datasets import imdb

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

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense max\_words = 5000

max\_len = 200

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_words) X\_train = pad\_sequences(x\_train, maxlen=max\_len)

X\_test = pad\_sequences(x\_test, maxlen=max\_len) model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=32, input\_length=max\_len)) model.add(SimpleRNN(32))

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

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

model.fit(X\_train, y\_train, epochs=2, batch\_size=64, validation\_split=0.2) loss, acc = model.evaluate(X\_test, y\_test)

print(f"\nTest Accuracy: {acc:.4f}") word\_index = imdb.get\_word\_index()

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

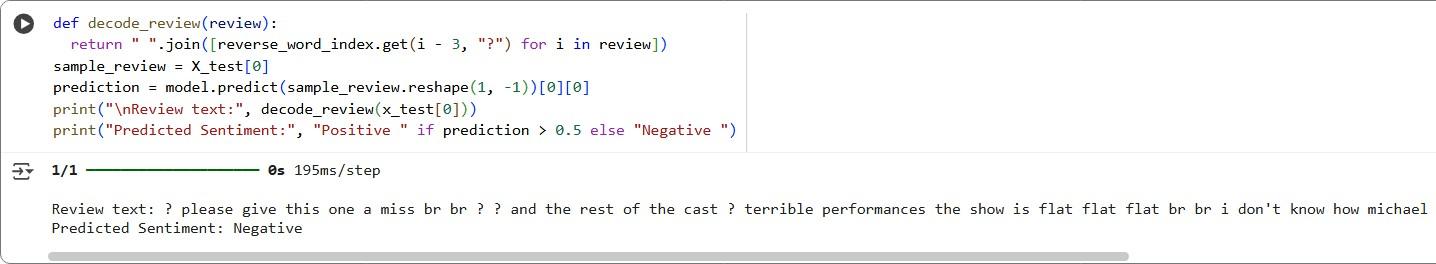
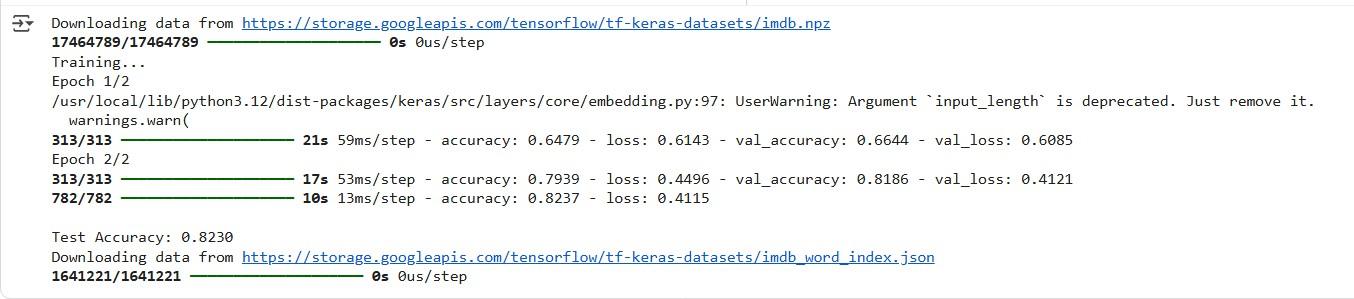
def decode\_review(review):

return " ".join([reverse\_word\_index.get(i - 3, "?") for i in review]) sample\_review = X\_test[0]

prediction = model.predict(sample\_review.reshape(1, -1))[0][0] print("\nReview text:", decode\_review(x\_test[0]))

print("Predicted Sentiment:", "Positive " if prediction > 0.5 else "Negative ")

## Output:



**Result**

Thus, the Recurrent Neural Network (RNN) using Keras has been implemented for classifying sentiment of text successfully.

## Ex No: 7 BUILD AUTOENCODERS WITH KERAS/TENSORFLOW

**DATE:22/09/2025**

## Aim:

To build autoencoders with Keras/TensorFlow.

## Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Input, Dense from keras.models import Model from keras.datasets import mnist

(x\_train, \_), (x\_test, \_) = mnist.load\_data() x\_train = x\_train.astype('float32') / 255. x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:]))) x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:]))) input\_img = Input(shape=(784,))

encoded = Dense(32, activation='relu')(input\_img) decoded = Dense(784, activation='sigmoid')(encoded) autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy') autoencoder.fit(x\_train, x\_train,

epochs=50, batch\_size=256, shuffle=True,

validation\_data=(x\_test, x\_test))

test\_loss = autoencoder.evaluate(x\_test, x\_test) decoded\_imgs = autoencoder.predict(x\_test) threshold = 0.5

correct\_predictions = np.sum( np.where(x\_test>= threshold, 1, 0) ==

np.where(decoded\_imgs>= threshold, 1, 0)) total\_pixels = x\_test.shape[0] \* x\_test.shape[1] test\_accuracy = correct\_predictions / total\_pixelsprint("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy) n = 10

plt.figure(figsize=(20, 4)) for i in range(n):

# Display original

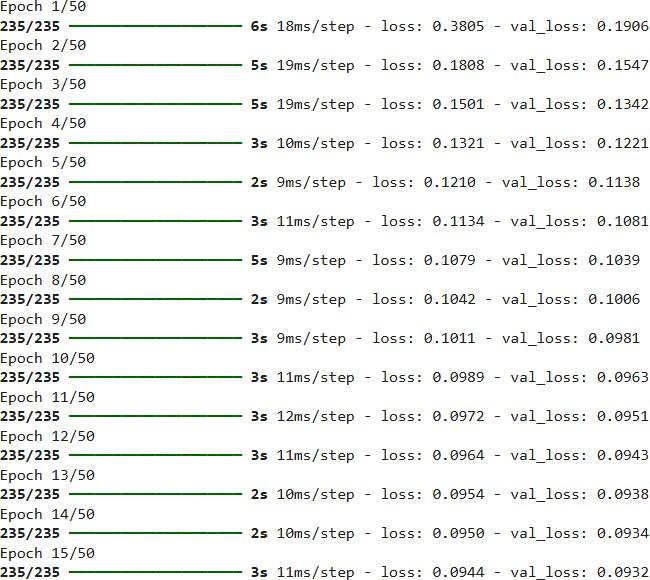
ax = plt.subplot(2, n, i + 1) plt.imshow(x\_test[i].reshape(28, 28)) plt.gray() ax.get\_xaxis().set\_visible(False) ax.get\_yaxis().set\_visible(False)

# Display reconstruction with threshold ax = plt.subplot(2, n, i + 1 + n)

reconstruction = decoded\_imgs[i].reshape(28, 28) plt.imshow(np.where(reconstruction >= threshold, 1.0, 0.0)) plt.gray()

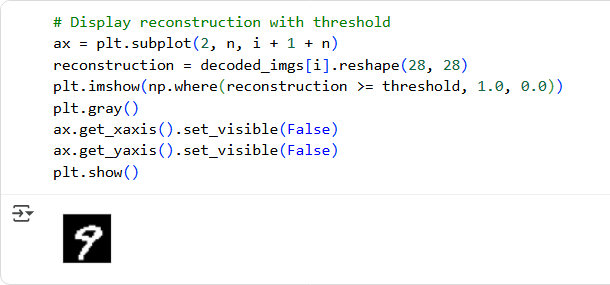
ax.get\_xaxis().set\_visible(False) ax.get\_yaxis().set\_visible(False) plt.show()

## Output:





A black numbers on a white background  AI-generated content may be incorrect.



**Result**

Thus, an Autoencoder has been implemented using Keras / Tensorflow.

## Ex No:8 OBJECT DETECTION WITH YOLO3

**DATE:29/09/2025**

## Aim:

To build an object detection model with YOLO3 using Keras/TensorFlow.

## Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

import cv2

import matplotlib.pyplot as plt import numpy as np

# Define the paths to the YOLOv3 configuration, weights, and class names files cfg\_file = '/content/yolov3.cfg'

weight\_file = '/content/yolov3.weights' namesfile = '/content/coco.names'

# Load the YOLOv3 model

net = cv2.dnn.readNet(weight\_file, cfg\_file)

# Load class names

with open(namesfile, 'r') as f:

classes = f.read().strip().split('\n')

# Load an image for object detection image\_path = '/content/hit.jpg' image = cv2.imread(image\_path)

# Get the height and width of the image height, width = image.shape[:2]

# Create a blob from the image

blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False) net.setInput(blob)

# Get the names of the output layers

layer\_names = net.getUnconnectedOutLayersNames()

# Run forward pass

outs = net.forward(layer\_names)

# Initialize lists to store detected objects' information class\_ids = []

confidences = [] boxes = []

# Define a confidence threshold for object detection conf\_threshold = 0.5

# Loop over the detections for out in outs:

for detection in out:

scores = detection[5:] class\_id = np.argmax(scores) confidence = scores[class\_id]

if confidence >conf\_threshold:

# Object detected

center\_x = int(detection[0] \* width) center\_y = int(detection[1] \* height) w = int(detection[2] \* width)

h = int(detection[3] \* height)

# Rectangle coordinates x = int(center\_x - w / 2) y = int(center\_y - h / 2)

class\_ids.append(class\_id) confidences.append(float(confidence)) boxes.append([x, y, w, h])

# Apply non-maximum suppression to eliminate overlapping boxes nms\_threshold = 0.4

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

# Draw bounding boxes and labels on the image for i in indices.flatten(): # flatten for compatibility x, y, w, h = boxes[i]

label = str(classes[class\_ids[i]]) confidence = confidences[i]

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

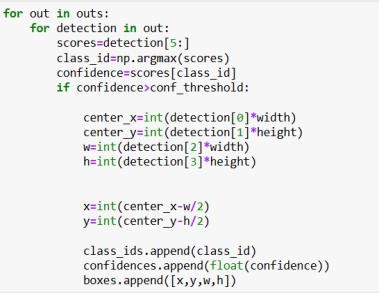
cv2.putText(image, f'{label} {confidence:.2f}', (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 255, 0), 2)

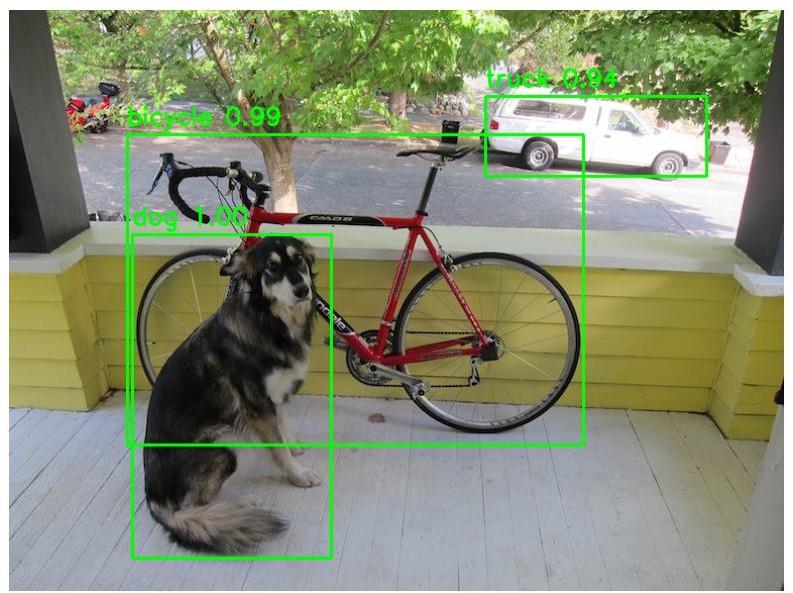
# Display the result in Jupyter Notebook plt.figure(figsize=(10, 8))

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)) plt.axis('off')

plt.show()

## Output:





**Result**

Thus, object detection using YOLOV5 has been implemented successfully.

## Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

**DATE:29/09/2025**

## Aim:

To build a generative adversarial neural network using Keras/TensorFlow.

## Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

## Code:

import numpy as np import tensorflow as tf

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential from tensorflow.keras.optimizers import Adam from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

# Load and Preprocess the Iris Dataset iris = load\_iris()

x\_train = iris.data

# Build the GAN model def build\_generator():

model = Sequential()

model.add(Dense(128, input\_shape=(100,), activation='relu')) model.add(Dense(4, activation='linear')) # Output 4 features return model

def build\_discriminator():

model = Sequential()

model.add(Dense(128, input\_shape=(4,), activation='relu')) model.add(Dense(1, activation='sigmoid'))

return model

def build\_gan(generator, discriminator): discriminator.trainable = False

model = Sequential() model.add(generator) model.add(discriminator)

return model

generator = build\_generator() discriminator = build\_discriminator()

gan = build\_gan(generator, discriminator)

# Compile the Models

generator.compile(loss='mean\_squared\_error', optimizer=Adam(0.0002, 0.5))

discriminator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])

gan.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

# Training Loop epochs = 200

batch\_size = 16

for epoch in range(epochs):

# Train discriminator

idx = np.random.randint(0, x\_train.shape[0], batch\_size) real\_samples = x\_train[idx]

fake\_samples = generator.predict(np.random.normal(0, 1, (batch\_size, 100)), verbose=0)

real\_labels = np.ones((batch\_size, 1)) fake\_labels = np.zeros((batch\_size, 1))

d\_loss\_real = discriminator.train\_on\_batch(real\_samples, real\_labels) d\_loss\_fake = discriminator.train\_on\_batch(fake\_samples, fake\_labels)

# Train generator

noise = np.random.normal(0, 1, (batch\_size, 100)) g\_loss = gan.train\_on\_batch(noise, real\_labels)

# Print progress

print(f"Epoch {epoch}/{epochs} | Discriminator Loss: {0.5 \* (d\_loss\_real[0] + d\_loss\_fake[0])} | Generator Loss: {g\_loss}")

# Generating Synthetic Data

synthetic\_data = generator.predict(np.random.normal(0, 1, (150, 100)), verbose=0)

# Create scatter plots for feature pairs plt.figure(figsize=(12, 8))

plot\_idx = 1

for i in range(4):

for j in range(i + 1, 4): plt.subplot(2, 3, plot\_idx)

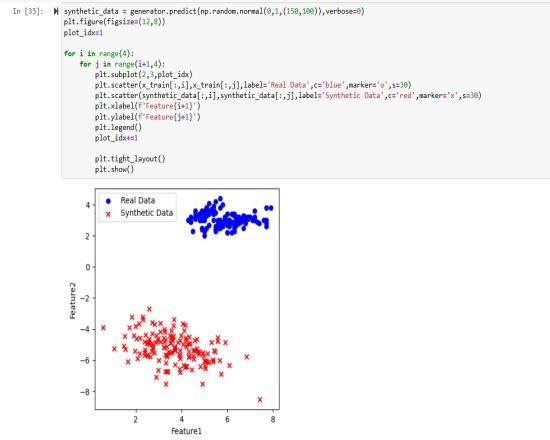
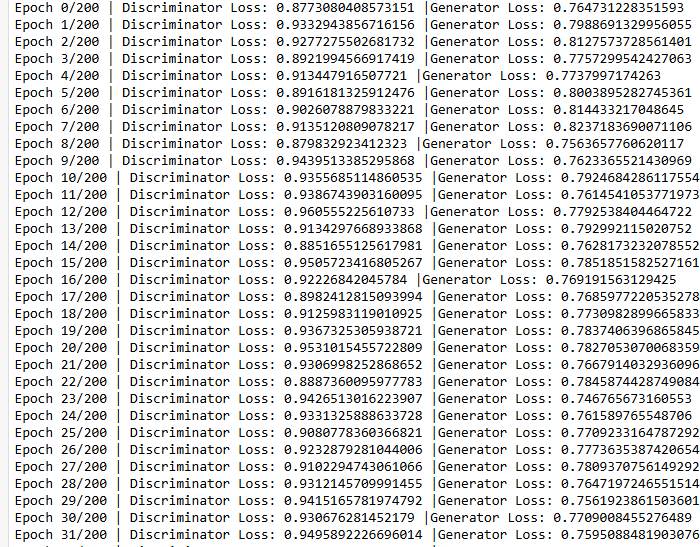
plt.scatter(x\_train[:,i], x\_train[:, j], label='Real Data', c='blue', marker='o', s=30) plt.scatter(synthetic\_data[:,i], synthetic\_data[:, j], label='Synthetic Data', c='red', marker='x', s=30)

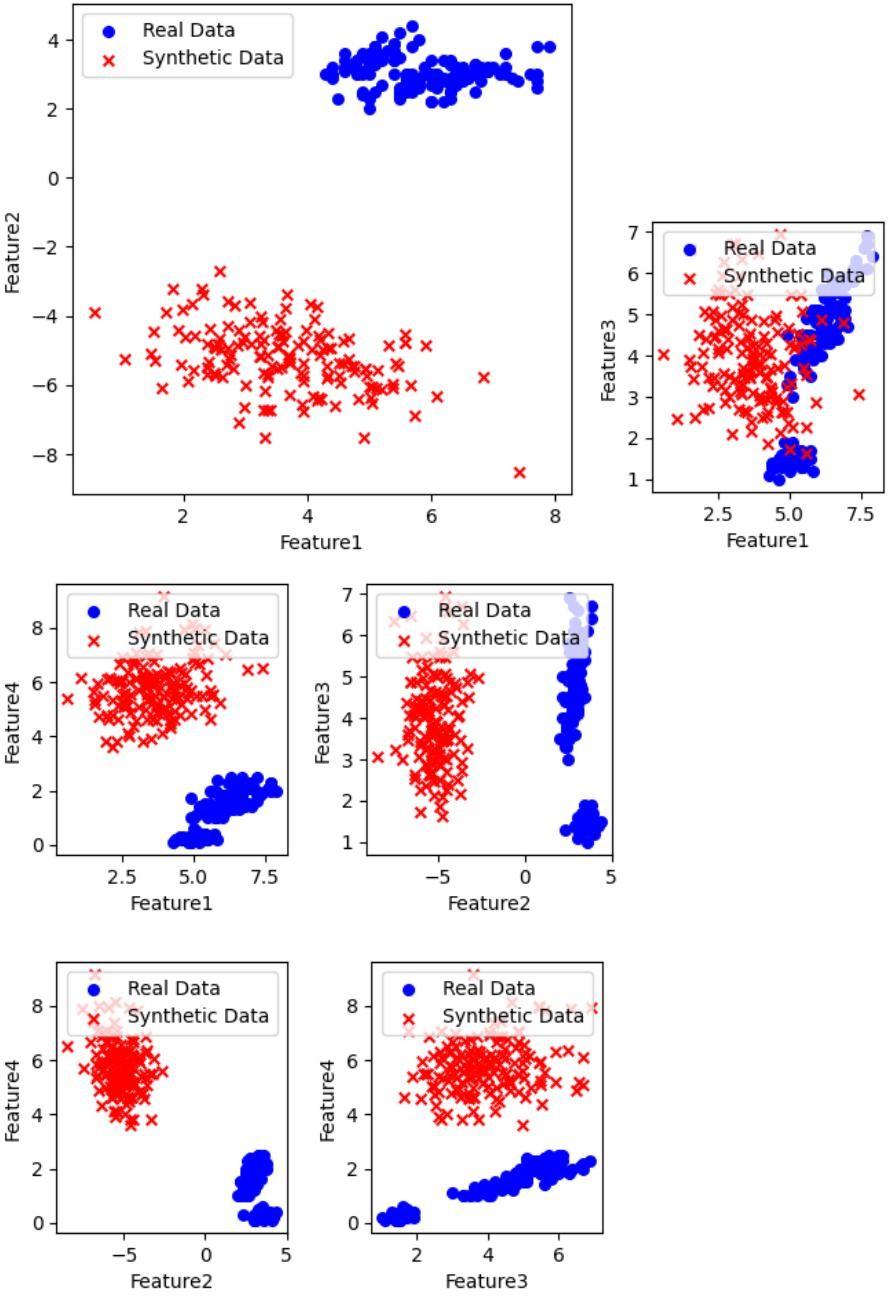
plt.xlabel(f'Feature {i + 1}') plt.ylabel(f'Feature {j + 1}') plt.legend()

plot\_idx += 1

plt.tight\_layout() plt.show()

## Output:





**Result**

Thus, a generative adversarial neural network using Keras / Tensorflow has been implemented successfully.

## Ex No:10 MINI PROJECT

**DATE:06/10/2025**

**CONVOLUTIONAL NEURAL NETWORKS**

**Aim:**

**To develop a deep learning-based system using Convolutional Neural Networks (CNNs)**

**For emotion recognition   
import** os

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator

**from** tensorflow.keras **import** layers, models

**from** tensorflow.keras.callbacks **import** EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

**from** sklearn.utils **import** class\_weight

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

**import** cv2

**from** PIL **import** Image

**import** tensorflow **as** tf

*# Paths*

BASE\_DIR **=** 'data'

TRAIN\_DIR **=** os**.**path**.**join(BASE\_DIR, 'train')

TEST\_DIR **=** os**.**path**.**join(BASE\_DIR, 'test')

SAVE\_DIR **=** 'saved\_models'

os**.**makedirs(SAVE\_DIR, exist\_ok**=True**)

*# Parameters*

IMG\_SIZE **=** (48, 48)

BATCH\_SIZE **=** 64

NUM\_CLASSES **=** 7

EPOCHS **=** 50

SEED **=** 42

labels **=** ['angry','disgust','fear','happy','sad','surprise','neutral']

BASE\_DIR **=** r'C:\Users\ssanj\emotion-recognition\data'

TRAIN\_DIR **=** os**.**path**.**join(BASE\_DIR, 'train')

TEST\_DIR **=** os**.**path**.**join(BASE\_DIR, 'test')

print("TRAIN\_DIR exists?", os**.**path**.**exists(TRAIN\_DIR))

print("TEST\_DIR exists?", os**.**path**.**exists(TEST\_DIR))

print("Train subfolders:", os**.**listdir(TRAIN\_DIR))

**import** os

print("Current working directory:", os**.**getcwd())  
train\_datagen **=** ImageDataGenerator(

rescale**=**1.**/**255,

rotation\_range**=**15,

width\_shift\_range**=**0.1,

height\_shift\_range**=**0.1,

zoom\_range**=**0.1,

horizontal\_flip**=True**,

validation\_split**=**0.2

)

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

TRAIN\_DIR,

target\_size**=**IMG\_SIZE,

color\_mode**=**'grayscale',

batch\_size**=**BATCH\_SIZE,

class\_mode**=**'categorical',

subset**=**'training',

shuffle**=True**,

seed**=**SEED

)

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

TRAIN\_DIR,

target\_size**=**IMG\_SIZE,

color\_mode**=**'grayscale',

batch\_size**=**BATCH\_SIZE,

class\_mode**=**'categorical',

subset**=**'validation',

shuffle**=False**,

seed**=**SEED

)

*# Class weights for imbalanced data*

classes **=** train\_generator**.**classes

*#class\_weights = class\_weight.compute\_class\_weight('balanced', np.unique(classes), classes)*

**from** sklearn.utils.class\_weight **import** compute\_class\_weight

**import** numpy **as** np

classes **=** train\_generator**.**classes

unique\_classes **=** np**.**unique(classes)

class\_weights **=** compute\_class\_weight(

class\_weight**=**'balanced', *# must be keyword*

classes**=**unique\_classes, *# must be keyword*

y**=**classes *# must be keyword*

)

class\_weights\_dict **=** {i: w **for** i, w **in** enumerate(class\_weights)}

print("Class indices:", train\_generator**.**class\_indices)

print("Class weights:", class\_weights\_dict)

class\_weights\_dict **=** {i: w **for** i, w **in** enumerate(class\_weights)}

print("Class indices:", train\_generator**.**class\_indices)

print("Class weights:", class\_weights\_dict)  
**def** create\_cnn(input\_shape**=**(48,48,1), num\_classes**=**7):

model **=** models**.**Sequential()

*# Block 1*

model**.**add(layers**.**Conv2D(64, (3,3), activation**=**'relu', padding**=**'same', input\_shape**=**input\_shape))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**Conv2D(64, (3,3), activation**=**'relu', padding**=**'same'))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**MaxPooling2D((2,2)))

model**.**add(layers**.**Dropout(0.25))

*# Block 2*

model**.**add(layers**.**Conv2D(128, (3,3), activation**=**'relu', padding**=**'same'))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**Conv2D(128, (3,3), activation**=**'relu', padding**=**'same'))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**MaxPooling2D((2,2)))

model**.**add(layers**.**Dropout(0.25))

*# Block 3*

model**.**add(layers**.**Conv2D(256, (3,3), activation**=**'relu', padding**=**'same'))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**Conv2D(256, (3,3), activation**=**'relu', padding**=**'same'))

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**MaxPooling2D((2,2)))

model**.**add(layers**.**Dropout(0.25))

*# Classifier*

model**.**add(layers**.**Flatten())

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

model**.**add(layers**.**BatchNormalization())

model**.**add(layers**.**Dropout(0.5))

model**.**add(layers**.**Dense(num\_classes, activation**=**'softmax'))

**return** model

*# Create model*

model **=** create\_cnn((IMG\_SIZE[0], IMG\_SIZE[1], 1), NUM\_CLASSES)

model**.**compile(optimizer**=**tf**.**keras**.**optimizers**.**Adam(1e-3),

loss**=**'categorical\_crossentropy',

metrics**=**['accuracy'])

model**.**summary()  
checkpoint\_path **=** os**.**path**.**join(SAVE\_DIR, 'Emotion\_Recognition\_CNN.h5')

callbacks **=** [

EarlyStopping(monitor**=**'val\_loss', patience**=**8, restore\_best\_weights**=True**, verbose**=**1),

ReduceLROnPlateau(monitor**=**'val\_loss', factor**=**0.5, patience**=**3, min\_lr**=**1e-6, verbose**=**1),

ModelCheckpoint(checkpoint\_path, monitor**=**'val\_loss', save\_best\_only**=True**, verbose**=**1)

]

history **=** model**.**fit(

train\_generator,

validation\_data**=**val\_generator,

epochs**=**EPOCHS,

class\_weight**=**class\_weights\_dict,

callbacks**=**callbacks

)

*# Save final model*

model**.**save(os**.**path**.**join(SAVE\_DIR, 'Emotion\_Recognition\_CNN.h5'))

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

plt**.**subplot(1,2,1)

plt**.**plot(history**.**history['accuracy'], label**=**'train\_acc')

plt**.**plot(history**.**history['val\_accuracy'], label**=**'val\_acc')

plt**.**xlabel('Epochs')

plt**.**ylabel('Accuracy')

plt**.**legend()

plt**.**title('Training vs Validation Accuracy')

plt**.**subplot(1,2,2)

plt**.**plot(history**.**history['loss'], label**=**'train\_loss')

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

plt**.**xlabel('Epochs')

plt**.**ylabel('Loss')

plt**.**legend()

plt**.**title('Training vs Validation Loss')

plt**.**show()  
**def** preprocess\_img(img\_path):

img **=** Image**.**open(img\_path)**.**convert('L')**.**resize(IMG\_SIZE)

arr **=** np**.**array(img)**.**astype('float32') **/** 255.0

arr **=** arr**.**reshape((1, IMG\_SIZE[0], IMG\_SIZE[1], 1))

**return** arr

**def** predict(img\_path):

arr **=** preprocess\_img(img\_path)

preds **=** model**.**predict(arr)

idx **=** int(np**.**argmax(preds))

**return** labels[idx], float(np**.**max(preds))

*# Example usage*

img\_path **=** 'data/test/angry/PrivateTest\_1623042.jpg'

label, conf **=** predict(img\_path)

print(f"Predicted: {label} (confidence: {conf:.3f})")

**OUTPUT**

