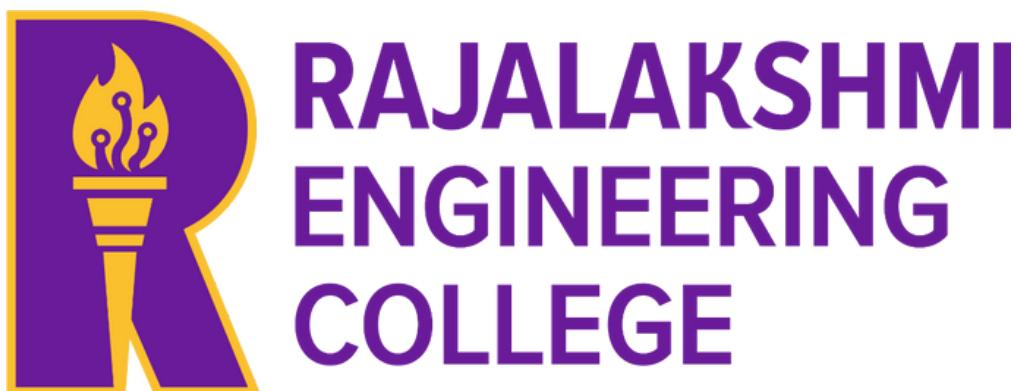


RAJALAKSHMI ENGINEERING COLLEGE
(An Autonomous Institution)

RAJALAKSHMINAGAR, THANDALAM-602105



CS19P18 - DEEP LEARNING CONCEPTS
LABORATORY RECORD NOTEBOOK

NAME: RAGUNANDAN B

YEAR/SEMESTER: IV/VII

BRANCH: CSE

REGISTER NO: 2116220701211

COLLEGE ROLL NO: 220701211

ACADEMIC YEAR: 2025 -2026



RAJALAKSHMI ENGINEERING COLLEGE
(An Autonomous Institution)
RAJALAKSHMINAGAR, THANDALAM-602 105

BONAFIDE CERTIFICATE

NAME: RAGUNANDAN B

BRANCH/SECTION: CSE

ACADEMIC YEAR: 2025 -2026

SEMESTER: VII

REGISTER NO:

2116220701211

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 THEEXPERIMENT	PAGENO	STAFF SIGN
1	20. 8.2025	Create a neural network to recognize handwritten digits using MNIST dataset	5	
2	27. 8.2025	Build a Convolutional Neural Network with Keras/TensorFlow	9	
3	3.9.2025	Image Classification on CIFAR-10 Dataset using CNN	14	
4	10.9.2025	Transfer learning with CNN and Visualization	18	
5	17.9.2025	Build a Recurrent Neural Network using Keras/Tensorflow	23	
6	24.9.2025	Sentiment Classification of Text using RNN	27	
7	24.9.2025	Build autoencoders with keras/tensorflow	30	
8	08.10.2025	Object detection with yolo3	34	
9	08.10.2025	Build GAN with Keras/TensorFlow	39	
10	08.10.2025	Mini Project	44	

INSTALLATION AND CONFIGURATION OF TENSORFLOW

Aim:

To install and configure TensorFlow in 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
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 matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

feature_vector_length = 784
num_classes = 10

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

input_shape = (feature_vector_length)
print(f'Feature shape: {input_shape}')

X_train = X_train.reshape(X_train.shape[0], feature_vector_length)
X_test = X_test.reshape(X_test.shape[0], feature_vector_length)
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
Y_train = to_categorical(Y_train, num_classes)
Y_test = to_categorical(Y_test, num_classes)

model = Sequential()
model.add(Dense(350, input_shape=input_shape, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, Y_train, epochs=10, batch_size=250, verbose=1, validation_split=0.2)
```

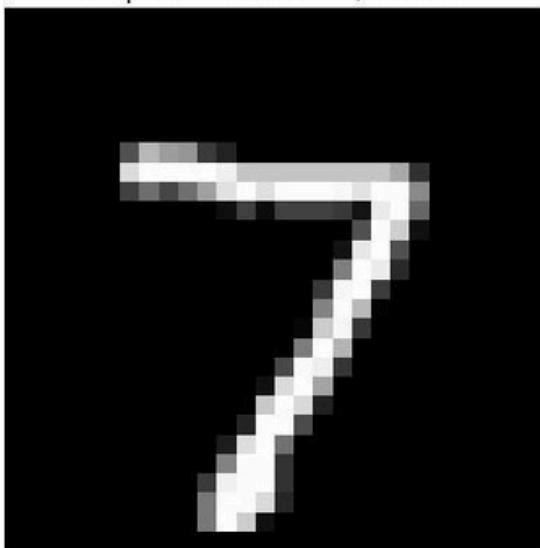
```
test_results = model.evaluate(X_test, Y_test, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}')

predictions = model.predict(X_test[:5])
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(Y_test[:5], axis=1)

for i in range(5):
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Sample {i+1} - Predicted: {predicted_classes[i]}, Actual: {true_classes[i]}")
    plt.axis('off')
    plt.show()
```

Output:

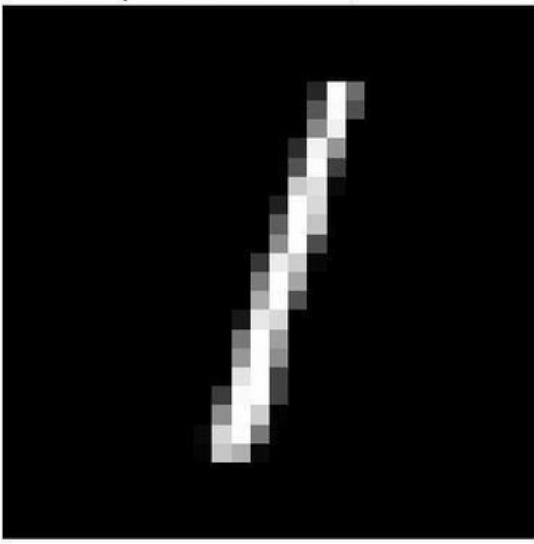
Sample 1 - Predicted: 7, Actual: 7



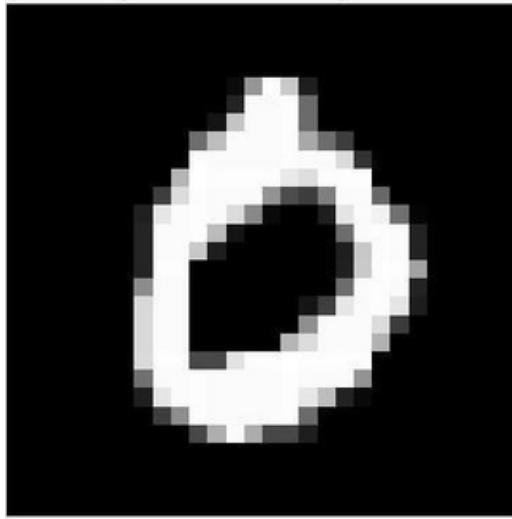
Sample 2 - Predicted: 2, Actual: 2



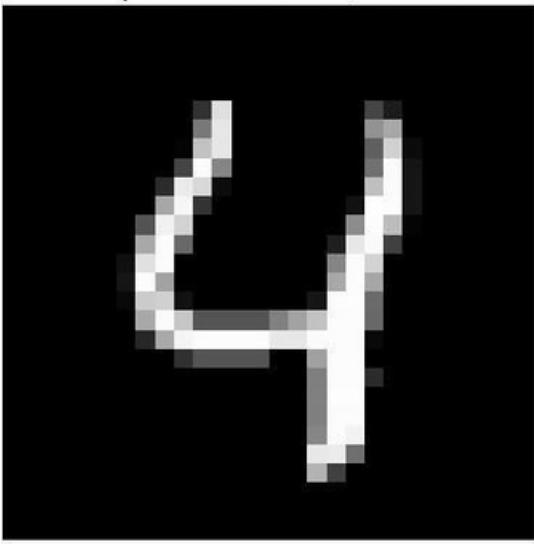
Sample 3 - Predicted: 1, Actual: 1



Sample 4 - Predicted: 0, Actual: 0



Sample 5 - Predicted: 4, Actual: 4



Result:

A handwritten digit's recognition with MNIST dataset is built and the output is verified.

EX NO:2 **BUILD A CONVOLUTIONAL NEURAL NETWORK**
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)

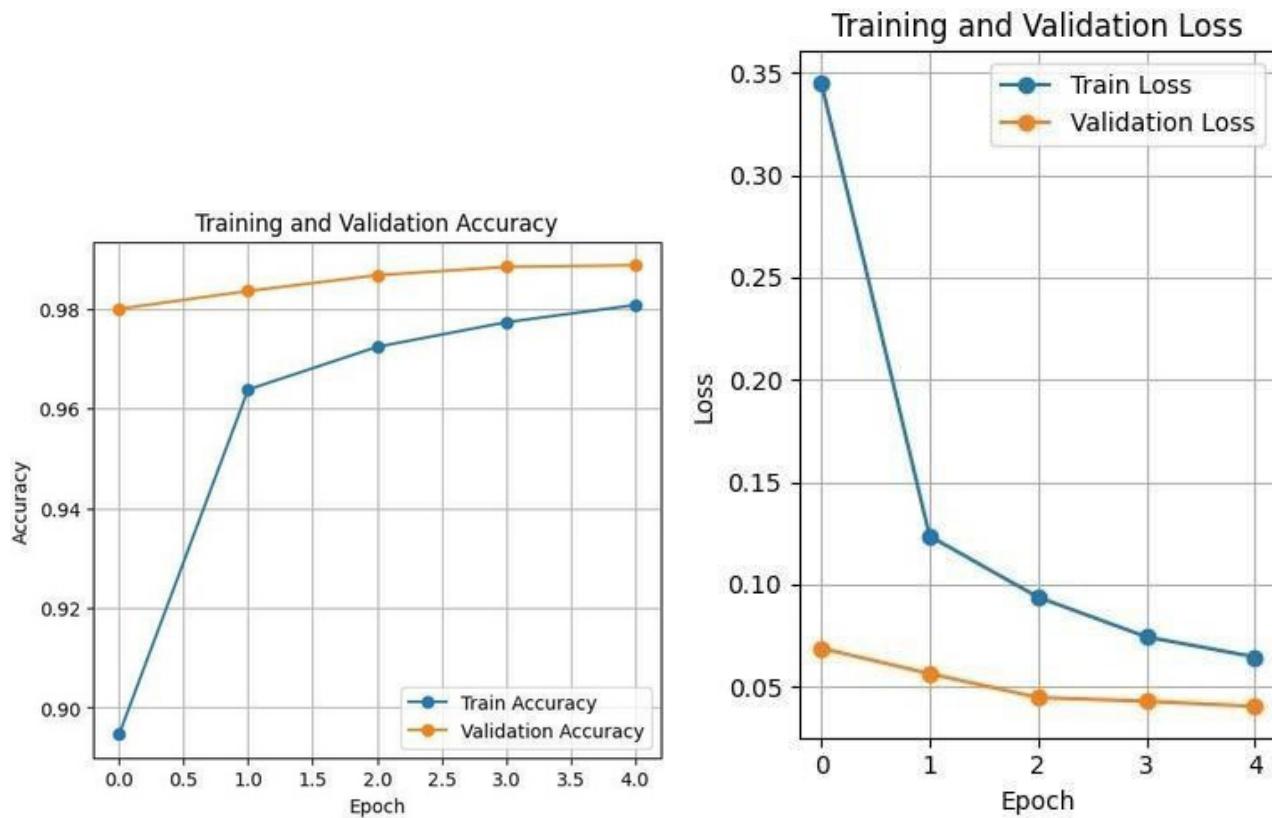
num_samples = 10

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:



Sample Predictions on Test Images

Pred: 7 Pred: 2 Pred: 1 Pred: 0 Pred: 4 Pred: 9
True: 7 True: 2 True: 1 True: 0 True: 4 True: 9 True: 5 True: 9



Result:

A CNN using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy is built and the output is verified.

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

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

4. Build a CNN model using Keras/TensorFlow:

 Include convolutional, pooling, flatten, and dense layers.

5. Compile the model with suitable loss function and optimizer.

6. Train the model using training data and validate using test data.

7. Evaluate the model using accuracy and loss on test dataset.

8. 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)

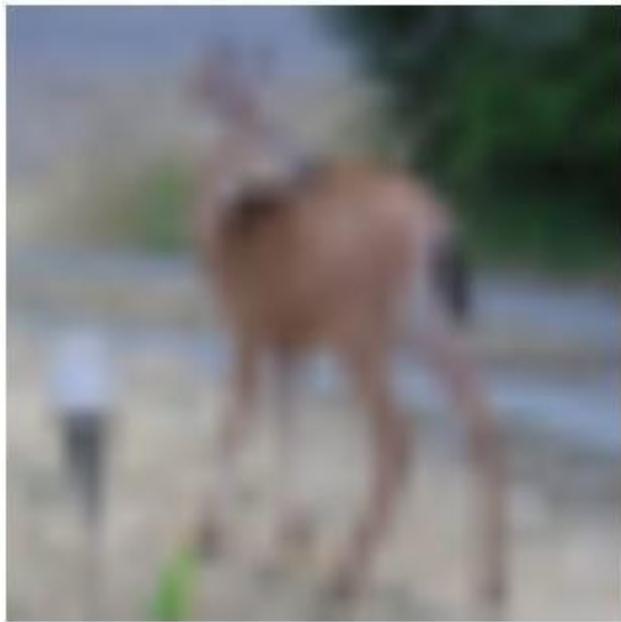
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

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:

Predicted: deer
Actual: deer



Result:

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. is successfully built and the output is verified.

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.

```

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()

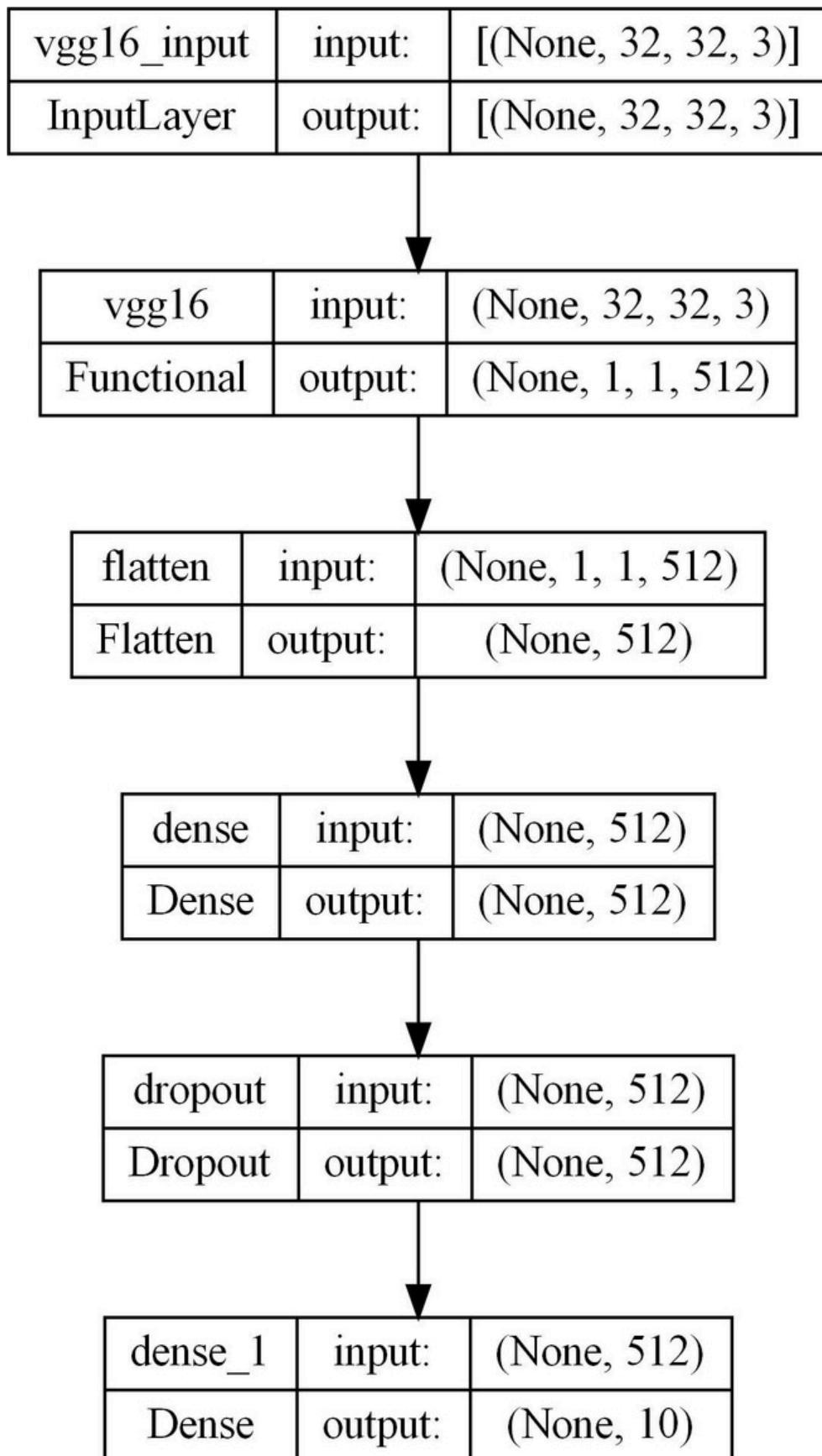
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

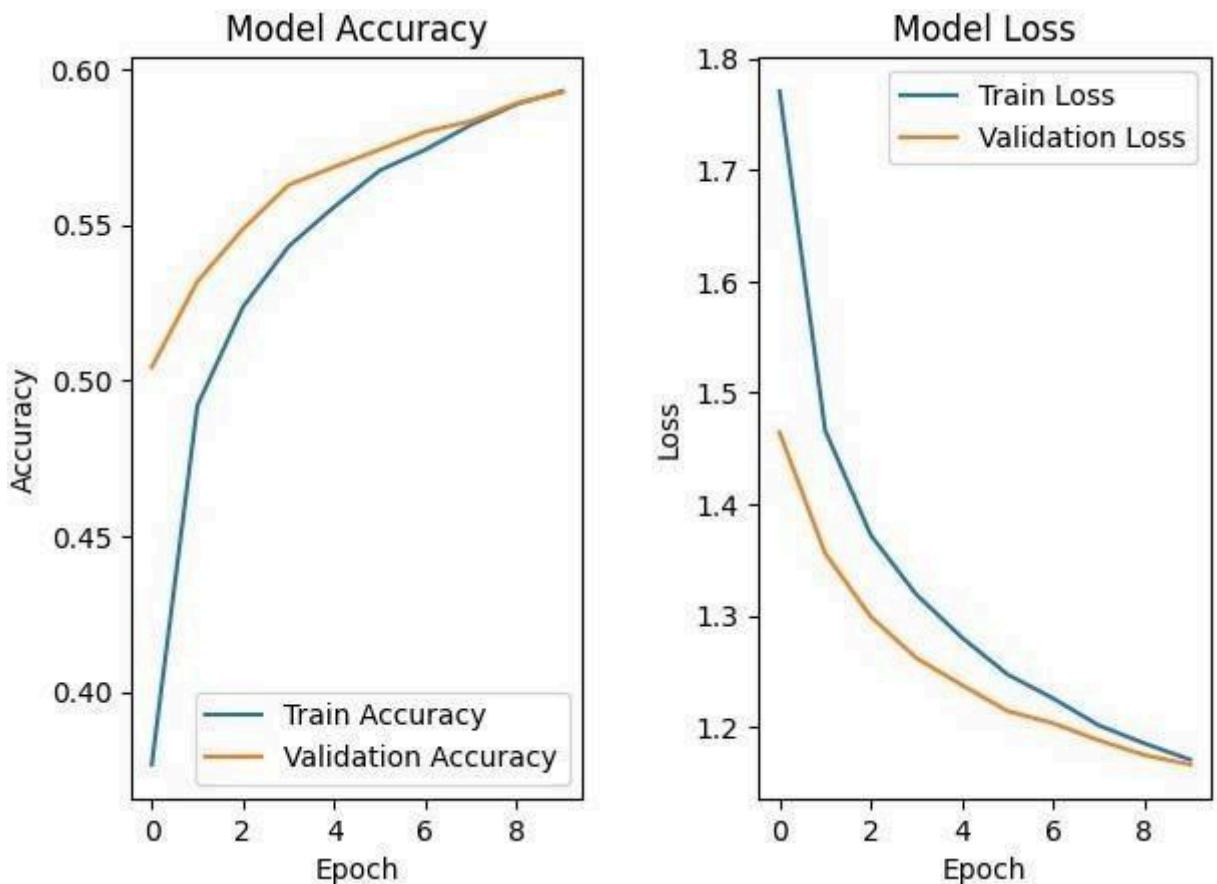
sample = x_test[0].reshape(1, 32, 32, 3)
prediction = model.predict(sample)
predicted_class = class_names[np.argmax(prediction)]


plt.imshow(x_test[0])
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()

```

Output:





Result:

A CNN with transfer learning and perform visualization is built and the output is verified.

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

```

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:

```
In [9]: ► test_loss = model.evaluate(x_test, y_test)
      print(f'Test Loss: {test_loss:.4f}')

      7/7 [=====] - 0s 5ms/step - loss: 0.0241
      Test Loss: 0.0241

In [10]: ► y_pred = model.predict(x_test)
      r2 = r2_score(y_test, y_pred)
      print(f'Test Accuracy (R^2): {r2:.4f}')

      7/7 [=====] - 0s 5ms/step
      Test Accuracy (R^2): 0.9974

In [11]: ► new_data = np.random.randn(5, seq_length, 1)
      predictions = model.predict(new_data)
      print("Predictions for new data:")
      print(predictions)

      1/1 [=====] - 0s 52ms/step
      Predictions for new data:
      [[ 1.7613161 ]
       [ 0.40740597]
       [ -2.23266   ]
       [ -0.6163975 ]
       [ -3.7167645 ]]
```

Result:

A recurrent neural network with Keras/TensorFlow is successfully built and the output is verified.

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

```

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:

```
Test Accuracy: 0.8502
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [=====] - 0s 0us/step

In [6]: M 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 ")

1/1 [=====] - 0s 244ms/step

Review text: ? please give this one a miss br br ? ? and the rest of the cast ? terrible performances the show is flat flat
flat br br i don't know how michael ? could have allowed this one on his ? he almost seemed to know this wasn't going to wor
k out and his performance was quite ? so all you ? fans give this a miss
Predicted Sentiment: Negative
```

Result: A Recurrent Neural Network (RNN) using Keras/TensorFlow for classifying the sentiment of text data (e.g., movie reviews) as positive or negative is successfully built and the output is verified.

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.

```

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_pixels
print("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:

Test Loss: 0.09151389449834824
Test Accuracy: 0.9713761479591837



Result:

A hautoencoders with Keras/TensorFlow is successfully built and the output is verified.

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.

```
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]])

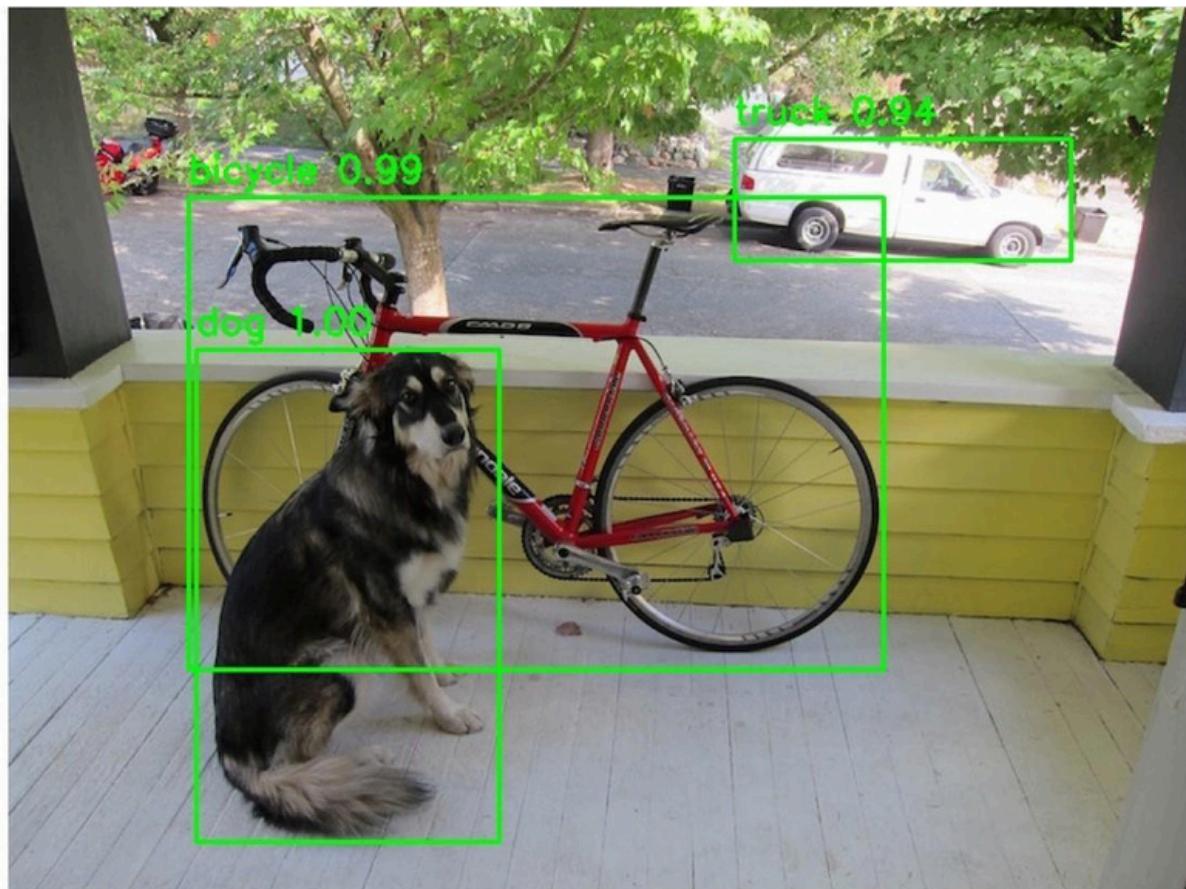
    confidence = confidences[i]

    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:

An object detection model with YOLO3 using Keras/TensorFlow is successfully built and the output is verified.

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

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.

```

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()

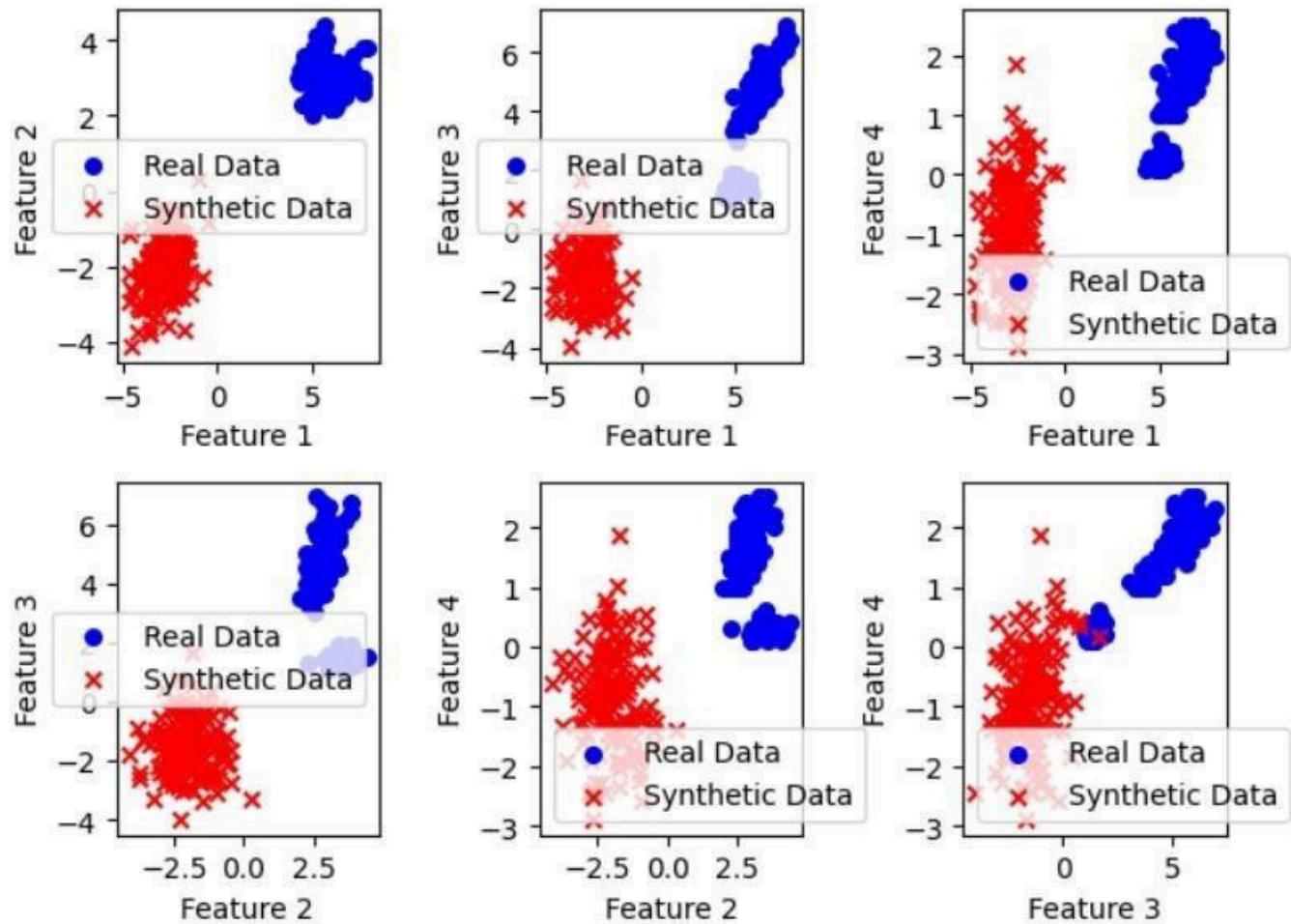
        plot_idx += 1

plt.tight_layout()

plt.show()

```

Output:



Result:

A generative adversarial neural network using Keras/TensorFlow is successfully built and the output is verified.

Aim:

To develop an application that is based on convolutional neural network or recurrent neural network in Keras/TensorFlow.

Code:

```
!pip install tensorflow keras captcha pillow numpy matplotlib opencv-python

import os

import string

import random

import numpy as np

from captcha.image import ImageCaptcha

from PIL import Image

import cv2

import matplotlib.pyplot as plt

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.models import Model

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

# Delete previous folders if they exist

if os.path.exists("captcha_train"):

    shutil.rmtree("captcha_train")

if os.path.exists("captcha_test"):

    shutil.rmtree("captcha_test")

# CAPTCHA parameters

characters = string.ascii_uppercase + string.digits

captcha_length = 2      # 2-character CAPTCHA

train_size = 2000      # large dataset
```

```

test_size = 400

width, height = 150, 50

# Create folders

os.makedirs("captcha_train", exist_ok=True)

os.makedirs("captcha_test", exist_ok=True)

# CAPTCHA generator

image = ImageCaptcha(width=width, height=height)

def generate_captchas(size, folder):

    for i in range(size):

        text = ''.join(random.choices(characters, k=captcha_length))

        image.write(text, f"{folder}/{text}.png")

# Generate datasets

generate_captchas(train_size, "captcha_train")

generate_captchas(test_size, "captcha_test")

def load_data(folder):

    X = []

    y = []

    for file in os.listdir(folder):

        label_text = file.split(".")[0]

        img = cv2.imread(os.path.join(folder, file), cv2.IMREAD_GRAYSCALE)

        img = img / 255.0      # normalize

        img = img.reshape(height, width, 1)

```

```

X.append(img)

y.append([characters.index(c) for c in label_text])

X = np.array(X)

y = np.array(y)

return X, y

X_train, y_train = load_data("captcha_train")

X_test, y_test = load_data("captcha_test")

# One-hot encode

y_train_oh = [to_categorical(y_train[:,i], num_classes=len(characters)) for i in range(captcha_length)]

y_train_oh = np.array(y_train_oh).transpose(1,0,2)

y_test_oh = [to_categorical(y_test[:,i], num_classes=len(characters)) for i in range(captcha_length)]

y_test_oh = np.array(y_test_oh).transpose(1,0,2)

print("Training data shape:", X_train.shape)

print("Training labels shape:", y_train_oh.shape)

input_img = Input(shape=(height, width, 1))

x = Conv2D(32, (3,3), activation='relu', padding='same')(input_img)

x = MaxPooling2D((2,2))(x)

x = Conv2D(64, (3,3), activation='relu', padding='same')(x)

x = MaxPooling2D((2,2))(x)

x = Conv2D(128, (3,3), activation='relu', padding='same')(x)

x = MaxPooling2D((2,2))(x)

```

```

x = Flatten()(x)

x = Dense(256, activation='relu')(x)

# 2 outputs (one per character)

outputs = [Dense(len(characters), activation='softmax', name=f'char{i+1}')(x) for i in
range(captcha_length)]


model = Model(inputs=input_img, outputs=outputs)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

history = model.fit(
    X_train, [y_train_oh[:,i,:] for i in range(captcha_length)],
    validation_data=(X_test, [y_test_oh[:,i,:] for i in range(captcha_length)]),
    epochs=50,    # can increase to 70 for even better accuracy
    batch_size=16
)

loss, *acc = model.evaluate(X_test, [y_test_oh[:,i,:] for i in range(captcha_length)])

print("Test Accuracy per character:", acc)

def predict_captcha(img_path):

    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    img = img / 255.0

    img = img.reshape(1, height, width, 1)

    pred = model.predict(img)

    pred_text = ''.join([characters[np.argmax(p)] for p in pred])

    return pred_text

# Display some predictions

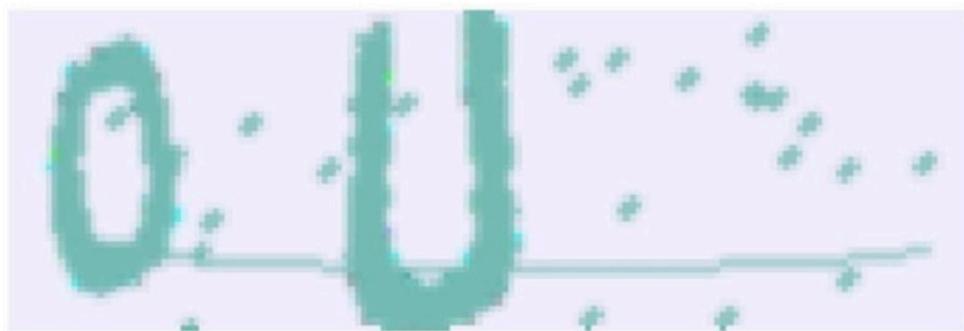
```

```
for file in random.sample(os.listdir("captcha_test"), 5):  
  
    path = os.path.join("captcha_test", file)  
  
    plt.imshow(cv2.imread(path))  
  
    plt.axis('off')  
  
    plt.show()  
  
    print("Actual:", file.split(".")[0], "| Predicted:", predict_captcha(path))
```

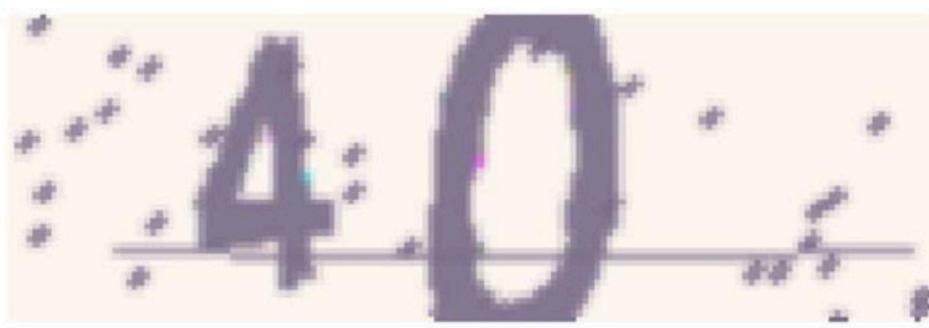
OUTPUT:



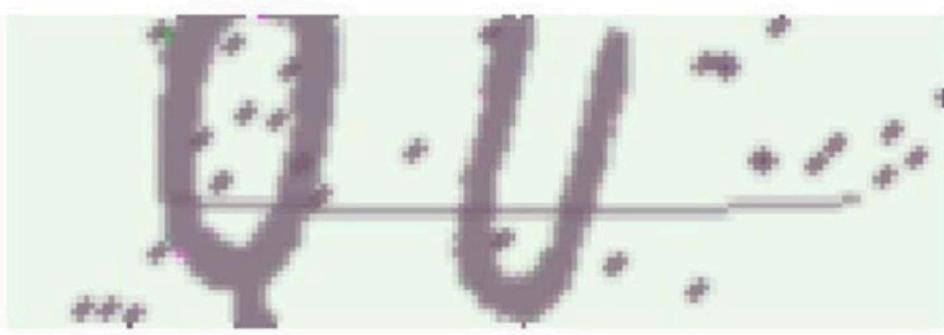
1/1 [=====] - 0s 130ms/step
Actual: OR | Predicted: OU



1/1 [=====] - 0s 25ms/step
Actual: OU | Predicted: OU



1/1 [=====] - 0s 23ms/step
Actual: 40 | Predicted: 0U



1/1 [=====] - 0s 22ms/step
Actual: QU | Predicted: 0U

```

Epoch 1/50
65/65 [=====] - 10s 128ms/step - loss: 7.1768 - char1_loss: 3.5871 - char2_loss: 3.5897 - char1_accuracy: 0.0244 - char2_accuracy: 0.0195 - val_loss: 7.1670 - val_char1_loss: 3.5835 - val_char2_loss: 3.5835 - val_char1_accuracy: 0.0317 - val_char2_accuracy: 0.0375
Epoch 2/50
65/65 [=====] - 10s 158ms/step - loss: 7.1674 - char1_loss: 3.5835 - char2_loss: 3.5839 - char1_accuracy: 0.0234 - char2_accuracy: 0.0293 - val_loss: 7.1675 - val_char1_loss: 3.5837 - val_char2_loss: 3.5838 - val_char1_accuracy: 0.0202 - val_char2_accuracy: 0.0259
Epoch 3/50
65/65 [=====] - 10s 150ms/step - loss: 7.1692 - char1_loss: 3.5849 - char2_loss: 3.5843 - char1_accuracy: 0.0273 - char2_accuracy: 0.0302 - val_loss: 7.1673 - val_char1_loss: 3.5837 - val_char2_loss: 3.5836 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0375
Epoch 4/50
65/65 [=====] - 10s 150ms/step - loss: 7.1663 - char1_loss: 3.5830 - char2_loss: 3.5833 - char1_accuracy: 0.0205 - char2_accuracy: 0.0263 - val_loss: 7.1673 - val_char1_loss: 3.5836 - val_char2_loss: 3.5836 - val_char1_accuracy: 0.0317 - val_char2_accuracy: 0.0231
Epoch 5/50
65/65 [=====] - 9s 145ms/step - loss: 7.1657 - char1_loss: 3.5827 - char2_loss: 3.5830 - char1_accuracy: 0.0283 - char2_accuracy: 0.0293 - val_loss: 7.1678 - val_char1_loss: 3.5838 - val_char2_loss: 3.5839 - val_char1_accuracy: 0.0202 - val_char2_accuracy: 0.0282
Epoch 6/50
65/65 [=====] - 9s 144ms/step - loss: 7.1654 - char1_loss: 3.5824 - char2_loss: 3.5830 - char1_accuracy: 0.0254 - char2_accuracy: 0.0234 - val_loss: 7.1684 - val_char1_loss: 3.5842 - val_char2_loss: 3.5842 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0202
Epoch 7/50
65/65 [=====] - 9s 144ms/step - loss: 7.1647 - char1_loss: 3.5821 - char2_loss: 3.5826 - char1_accuracy: 0.0293 - char2_accuracy: 0.0244 - val_loss: 7.1687 - val_char1_loss: 3.5842 - val_char2_loss: 3.5845 - val_char1_accuracy: 0.0317 - val_char2_accuracy: 0.0231
Epoch 8/50
Epoch 20/50
65/65 [=====] - 9s 145ms/step - loss: 7.1645 - char1_loss: 3.5815 - char2_loss: 3.5829 - char1_accuracy: 0.0244 - char2_accuracy: 0.0283 - val_loss: 7.1725 - val_char1_loss: 3.5871 - val_char2_loss: 3.5853 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0259
Epoch 21/50
65/65 [=====] - 9s 141ms/step - loss: 7.1623 - char1_loss: 3.5805 - char2_loss: 3.5817 - char1_accuracy: 0.0312 - char2_accuracy: 0.0293 - val_loss: 7.1732 - val_char1_loss: 3.5877 - val_char2_loss: 3.5856 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0375
Epoch 22/50
65/65 [=====] - 9s 140ms/step - loss: 7.1623 - char1_loss: 3.5806 - char2_loss: 3.5816 - char1_accuracy: 0.0312 - char2_accuracy: 0.0263 - val_loss: 7.1727 - val_char1_loss: 3.5871 - val_char2_loss: 3.5856 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0375
Epoch 23/50
65/65 [=====] - 9s 141ms/step - loss: 7.1621 - char1_loss: 3.5805 - char2_loss: 3.5816 - char1_accuracy: 0.0244 - char2_accuracy: 0.0302 - val_loss: 7.1732 - val_char1_loss: 3.5876 - val_char2_loss: 3.5856 - val_char1_accuracy: 0.0259 - val_char2_accuracy: 0.0259
Epoch 24/50
65/65 [=====] - 9s 140ms/step - loss: 7.1618 - char1_loss: 3.5805 - char2_loss: 3.5814 - char1_accuracy: 0.0302 - char2_accuracy: 0.0322 - val_loss: 7.1735 - val_char1_loss: 3.5876 - val_char2_loss: 3.5859 - val_char1_accuracy: 0.0259 - val_char2_accuracy: 0.0259
Epoch 25/50
65/65 [=====] - 9s 145ms/step - loss: 7.1620 - char1_loss: 3.5804 - char2_loss: 3.5817 - char1_accuracy: 0.0302 - char2_accuracy: 0.0322 - val_loss: 7.1741 - val_char1_loss: 3.5880 - val_char2_loss: 3.5862 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0259
Epoch 26/50
65/65 [=====] - 10s 161ms/step - loss: 7.1618 - char1_loss: 3.5805 - char2_loss: 3.5814 - char1_accuracy: 0.0312 - char2_accuracy: 0.0322 - val_loss: 7.1739 - val_char1_loss: 3.5878 - val_char2_loss: 3.5861 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0259
Epoch 27/50
65/65 [=====] - 10s 151ms/step - loss: 7.1616 - char1_loss: 3.5802 - char2_loss: 3.5814 - char1_accuracy: 0.0312 - char2_accuracy: 0.0322 - val_loss: 7.1744 - val_char1_loss: 3.5880 - val_char2_loss: 3.5864 - val_char1_accuracy: 0.0288 - val_char2_accuracy: 0.0259
Epoch 28/50
65/65 [=====] - 10s 157ms/step - loss: 7.1619 - char1_loss: 3.5805 - char2_loss: 3.5815 - char1_accuracy: 0.0263 - char2_accuracy: 0.0322 - val_loss: 7.1742 - val_char1_loss: 3.5878 - val_char2_loss: 3.5864 - val_char1_accuracy: 0.0202 - val_char2_accuracy: 0.0259

```

RESULT:

A captcha solver using convolutional neural network is developed and executed successfully.