RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

CS19P18 - DEEP LEARNING CONCEPTS

LABORATORY RECORD NOTEBOOK

NAME: DIVYASHREE S

YEAR/SEMESTER: IV/VII

BRANCH: CSE

REGISTER NO: 2116220701070

COLLEGE ROLL NO: 220701070

ACADEMIC YEAR: 2025 -2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: DIVYASHREE S **BRANCH/SECTION:** CSE

ACADEMIC YEAR: 2025 -2026 **SEMESTER:** VII

REGISTER NO: **2116220701070**

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

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.

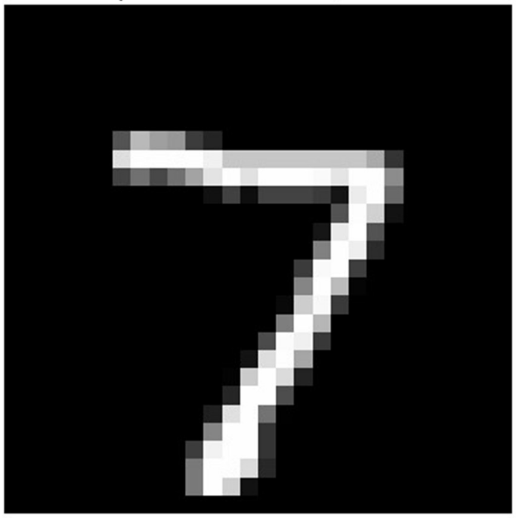
Code:

```
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)
```

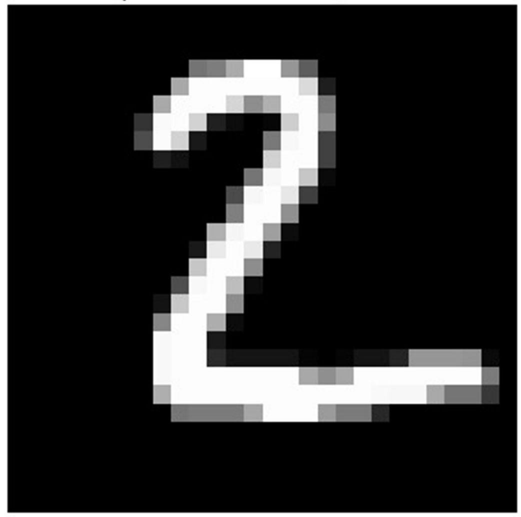
```python
test_results = model.evaluate(X_test, Y_test, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}')

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:

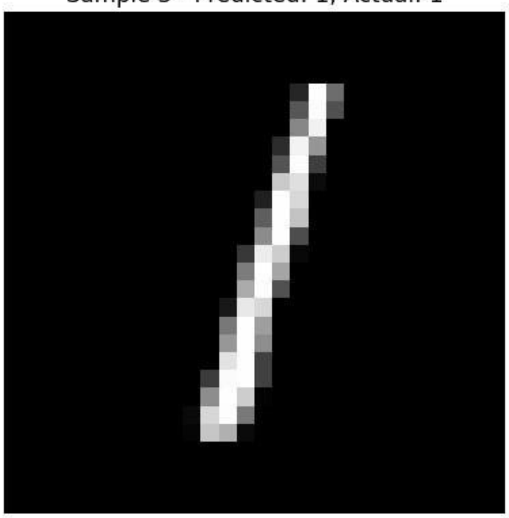
Sample 1 - Predicted: 7, Actual: 7



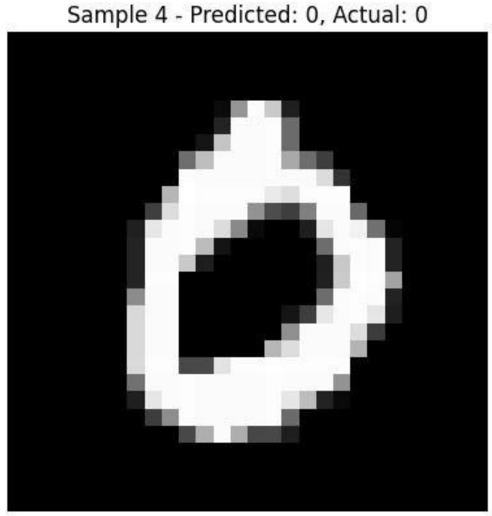
Sample 2 - Predicted: 2, Actual: 2



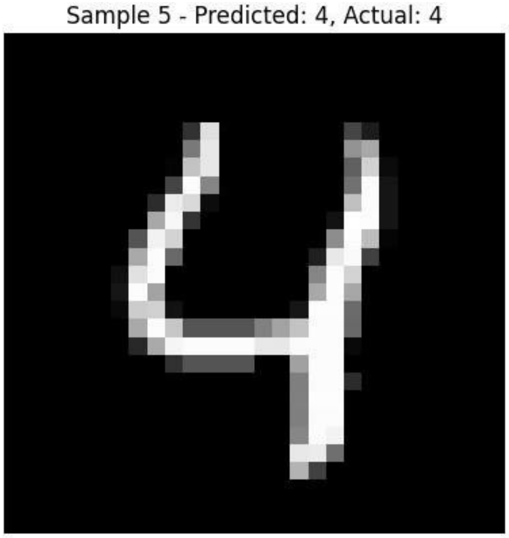
Sample 3 - Predicted: 1, Actual: 1



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.

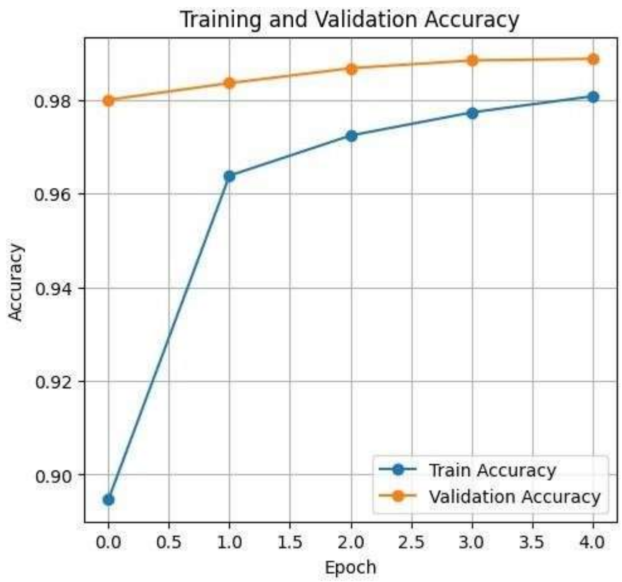
Code:

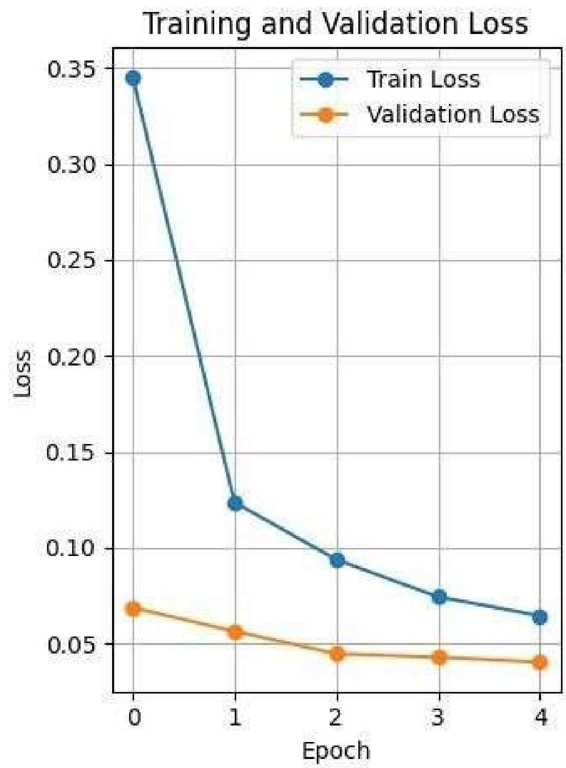
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np

(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)
```

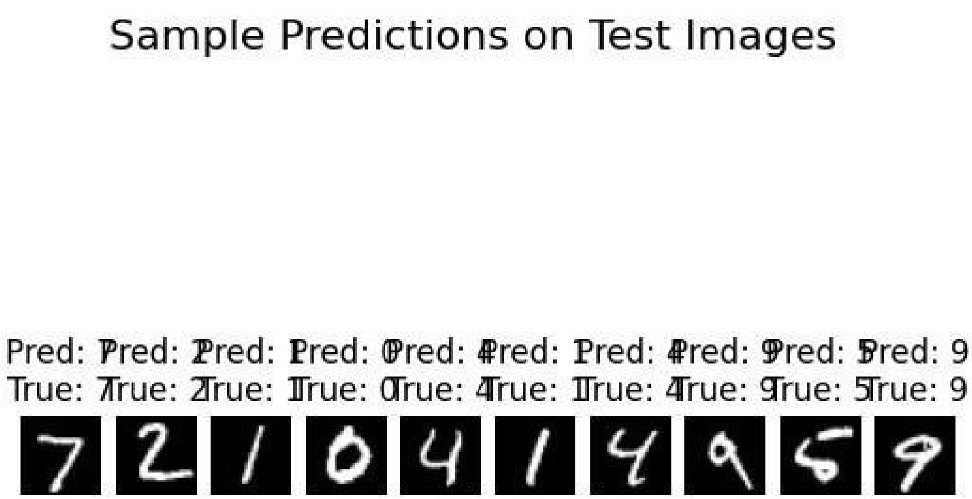
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"\n Test accuracy: {test_acc:.4f}")
print(f" Test loss: {test_loss:.4f}")

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))
```

```python
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:
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:

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.

Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION

Aim:

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

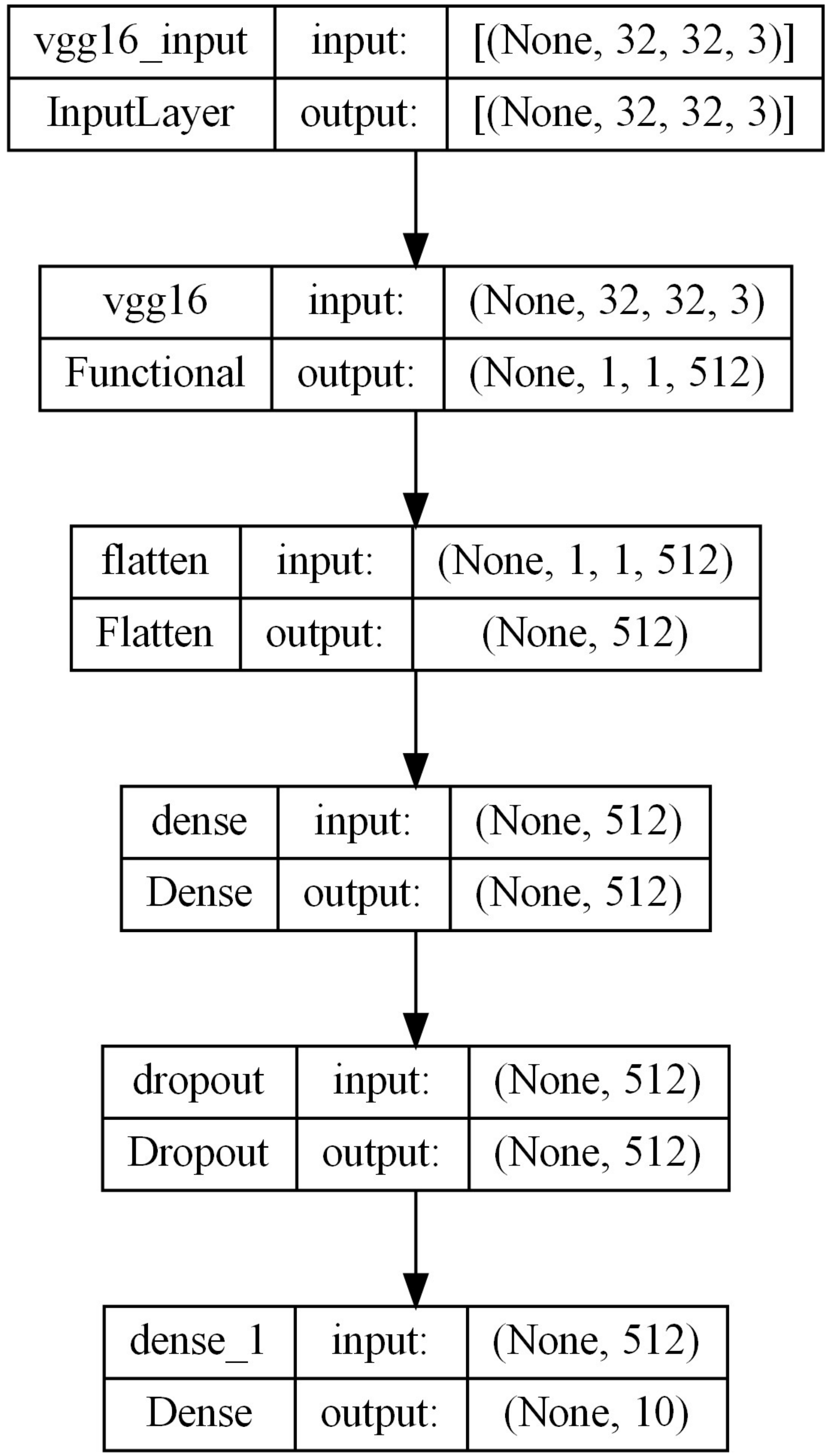
Procedure:

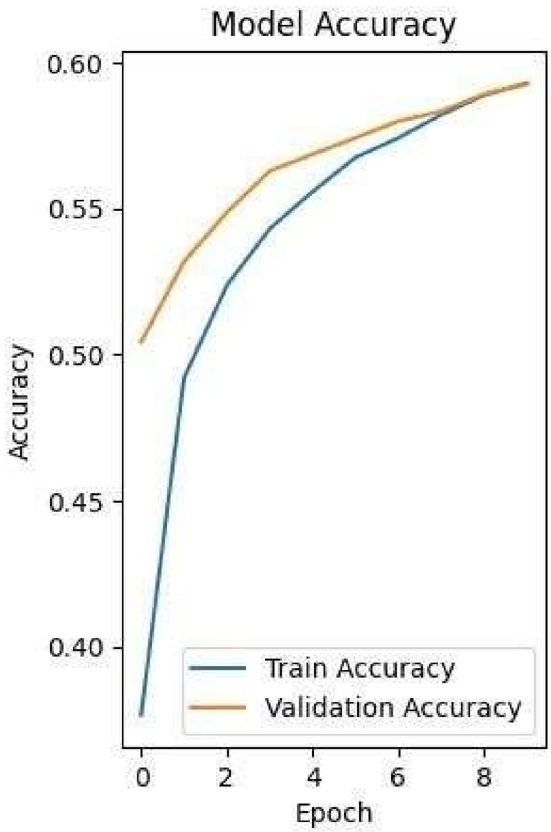
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

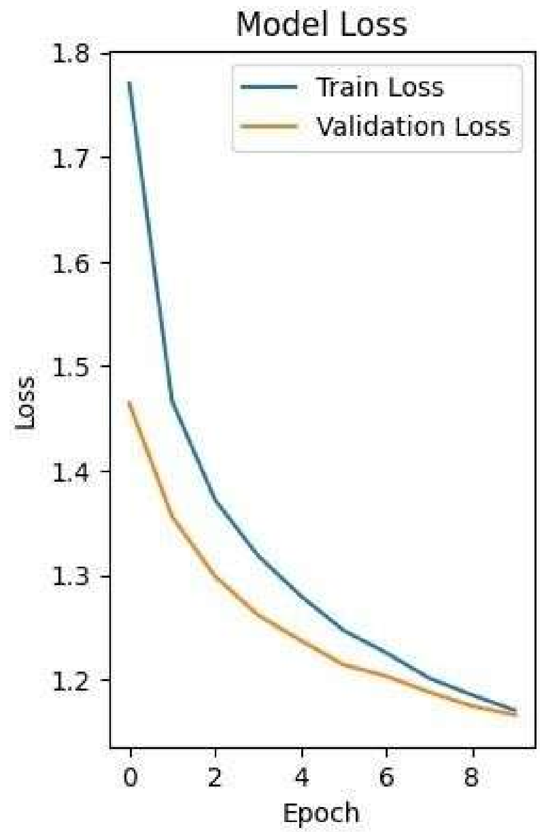
```
conda install -c conda-forge python-graphviz -y
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets 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:


vgg16_input
InputLayer
input:
output:
[(None, 32, 32, 3)]
[(None, 32, 32, 3)]
vgg16
Functional
input:
output:
(None, 32, 32, 3)
(None, 1, 1, 512)
flatten
Flatten
input:
output:
(None, 1, 1, 512)
(None, 512)
dense
Dense
input:
output:
(None, 512)
(None, 512)
dropout
Dropout
input:
output:
(None, 512)
(None, 512)
dense_1
Dense
input:
output:
(None, 512)
(None, 10)





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.

EX NO: 6 SENTIMENT CLASSIFICATION OF TEXT USING RNN

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]:

```python
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.

Ex No: 7 BUILD AUTOENCODERS WITH KERAS/TENSORFLOW

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.

Ex No: 8 **OBJECT DETECTION WITH YOLO3**

Aim:

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

Procedure:

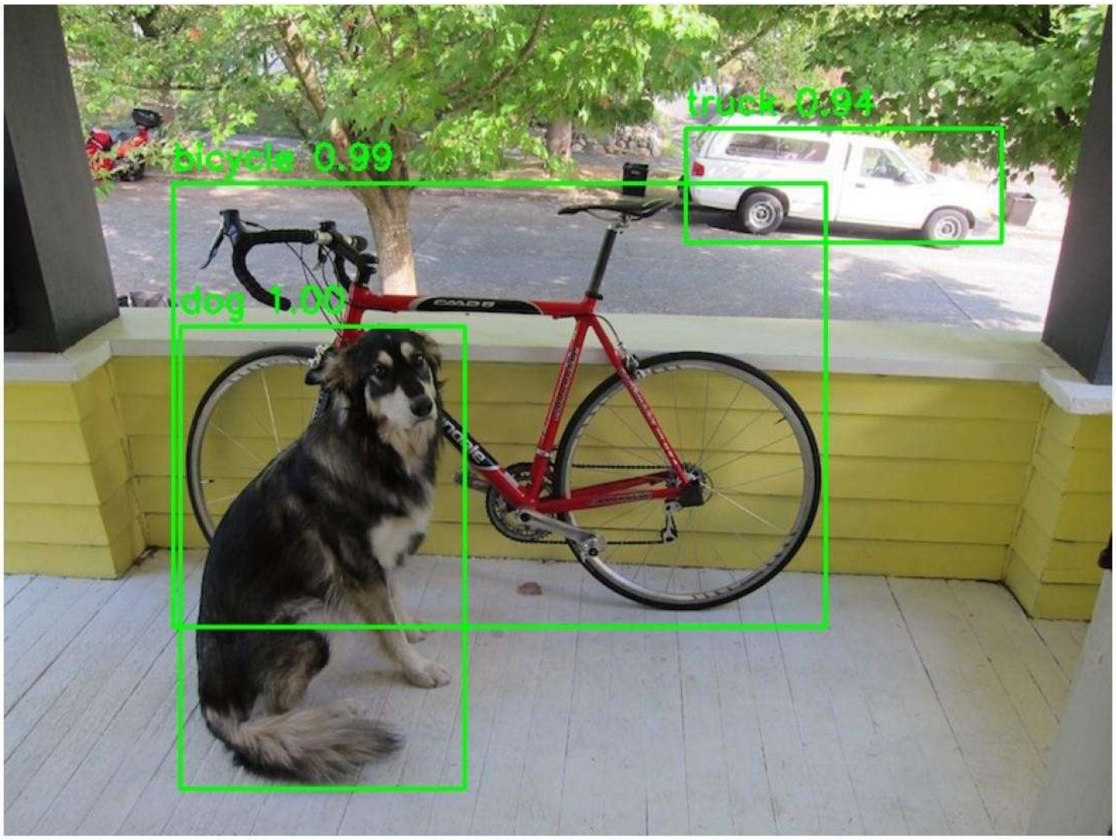
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

```
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:

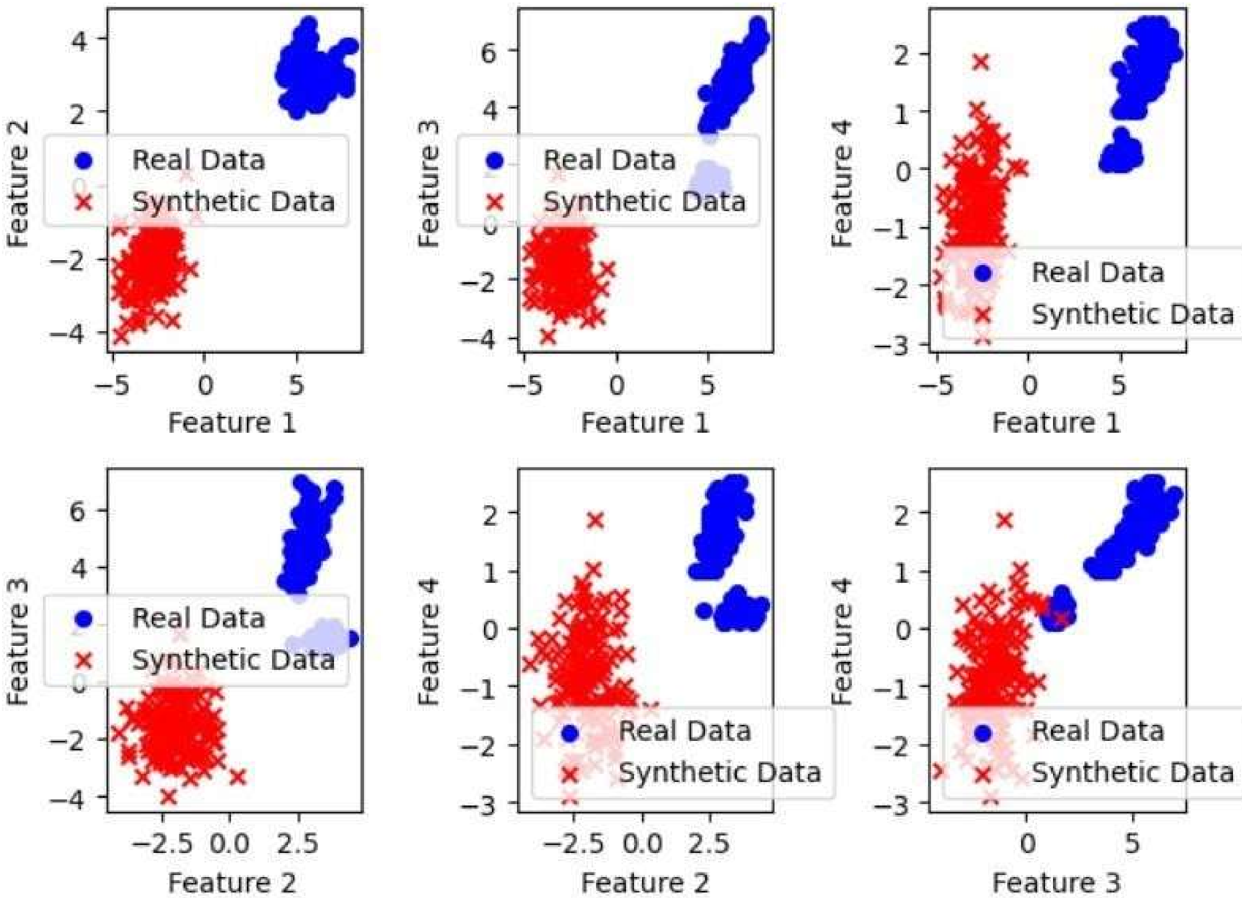
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

```
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.

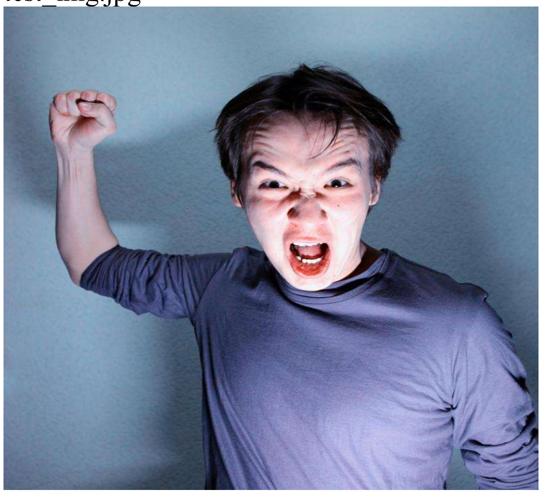
Ex No: 10 MINI PROJECT – CNN OR RNN BASED APPLICATION

Aim:
To develop an application that is based on a convolutional neural network in Keras/TensorFlow.
Mini Project Title:
FACIAL EMOTION RECOGNITION USING VGG16
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
train_dir = "dataset/train"
test_dir = "dataset/test"
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
)


test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    color_mode='rgb',
    class_mode='categorical'
)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    color_mode='rgb',
    class_mode='categorical'
)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(7, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()

callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ModelCheckpoint("best_vgg16_fer_model.h5", save_best_only=True)
]
history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=25,
    callbacks=callbacks
)
model.save("final_vgg16_fer_model.h5")
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('Epochs')
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('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
for layer in base_model.layers[-6:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5), loss='categorical_crossentropy',
metrics=['accuracy'])

fine_tune_history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=5
)
model.save("fine_tuned_vgg16_fer_model.h5")
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
img_path = "test_img.jpg"  # Path to your test image
img = load_img(img_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0
pred = model.predict(img_array)
emotion = emotion_labels[np.argmax(pred)]
```

print("\nPredicted Emotion:", emotion)
Output:
test_img.jpg

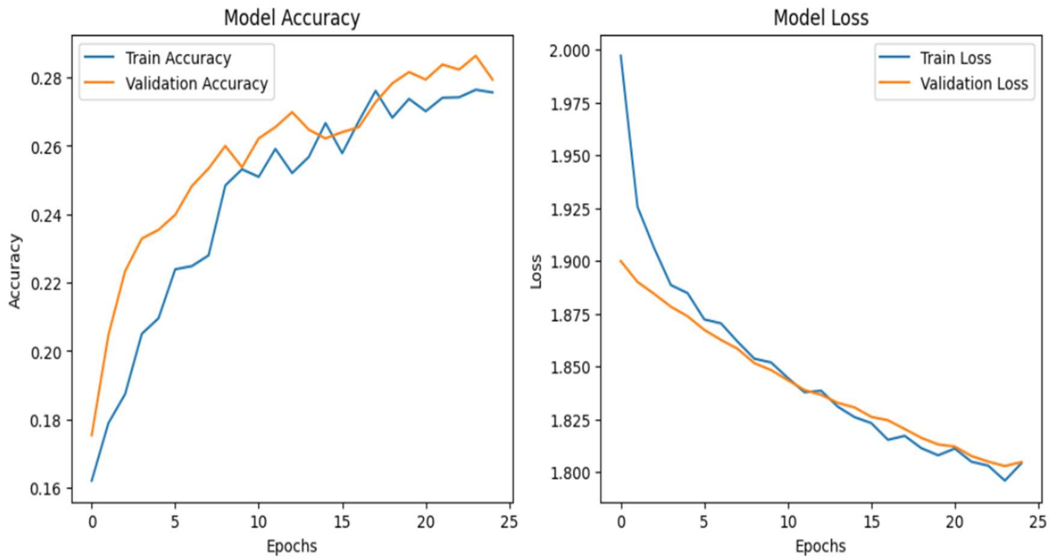


Found 6360 images belonging to 7 classes.
Found 2731 images belonging to 7 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ **2s** 0us/step
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 128)	65,664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 7)	903

```
 Total params: 14,781,255 (56.39 MB)
 Trainable params: 66,567 (260.03 KB)
 Non-trainable params: 14,714,688 (56.13 MB)
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `*
  self._warn_if_super_not_called()
Epoch 1/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 501ms/step - accuracy: 0.1559 - loss: 2.0339WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 157s 711ms/step - accuracy: 0.1559 - loss: 2.0337 - val_accuracy: 0.1754 - val_loss: 1.8999
Epoch 2/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 453ms/step - accuracy: 0.1743 - loss: 1.9361WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 106s 532ms/step - accuracy: 0.1743 - loss: 1.9361 - val_accuracy: 0.2047 - val_loss: 1.8901
Epoch 3/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 452ms/step - accuracy: 0.1833 - loss: 1.9089WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 528ms/step - accuracy: 0.1834 - loss: 1.9089 - val_accuracy: 0.2234 - val_loss: 1.8844
Epoch 4/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 450ms/step - accuracy: 0.2113 - loss: 1.8891WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 527ms/step - accuracy: 0.2112 - loss: 1.8891 - val_accuracy: 0.2329 - val_loss: 1.8783
Epoch 5/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step - accuracy: 0.2091 - loss: 1.8879WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 106s 533ms/step - accuracy: 0.2091 - loss: 1.8879 - val_accuracy: 0.2354 - val_loss: 1.8738
Epoch 6/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 452ms/step - accuracy: 0.2197 - loss: 1.8731WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 528ms/step - accuracy: 0.2198 - loss: 1.8731 - val_accuracy: 0.2398 - val_loss: 1.8673
Epoch 7/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 451ms/step - accuracy: 0.2331 - loss: 1.8630WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 528ms/step - accuracy: 0.2331 - loss: 1.8630 - val_accuracy: 0.2483 - val_loss: 1.8626
Epoch 8/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 454ms/step - accuracy: 0.2246 - loss: 1.8617WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 106s 531ms/step - accuracy: 0.2246 - loss: 1.8617 - val_accuracy: 0.2534 - val_loss: 1.8583
Epoch 9/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 453ms/step - accuracy: 0.2475 - loss: 1.8540WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 106s 531ms/step - accuracy: 0.2475 - loss: 1.8540 - val_accuracy: 0.2600 - val_loss: 1.8516
Epoch 10/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 457ms/step - accuracy: 0.2516 - loss: 1.8514WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 143s 535ms/step - accuracy: 0.2516 - loss: 1.8514 - val_accuracy: 0.2538 - val_loss: 1.8483
Epoch 11/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.2521 - loss: 1.8435WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 104s 523ms/step - accuracy: 0.2521 - loss: 1.8435 - val_accuracy: 0.2622 - val_loss: 1.8435
Epoch 12/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 451ms/step - accuracy: 0.2629 - loss: 1.8361WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 527ms/step - accuracy: 0.2629 - loss: 1.8361 - val_accuracy: 0.2655 - val_loss: 1.8388
Epoch 13/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 444ms/step - accuracy: 0.2445 - loss: 1.8449WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
```

```
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 444ms/step - accuracy: 0.2445 - loss: 1.8449WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(r
199/199 ━━━━━━━━━━━━━━━━━━━━ 104s 521ms/step - accuracy: 0.2445 - loss: 1.8449 - val_accuracy: 0.2699 - val_loss: 1.8365
Epoch 14/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 436ms/step - accuracy: 0.2559 - loss: 1.8268WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 102s 513ms/step - accuracy: 0.2559 - loss: 1.8268 - val_accuracy: 0.2647 - val_loss: 1.8327
Epoch 15/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step - accuracy: 0.2726 - loss: 1.8233WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 101s 507ms/step - accuracy: 0.2726 - loss: 1.8233 - val_accuracy: 0.2622 - val_loss: 1.8306
Epoch 16/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 434ms/step - accuracy: 0.2625 - loss: 1.8200WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 102s 511ms/step - accuracy: 0.2625 - loss: 1.8200 - val_accuracy: 0.2640 - val_loss: 1.8261
Epoch 17/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 444ms/step - accuracy: 0.2680 - loss: 1.8103WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 104s 522ms/step - accuracy: 0.2680 - loss: 1.8104 - val_accuracy: 0.2655 - val_loss: 1.8245
Epoch 18/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 435ms/step - accuracy: 0.2781 - loss: 1.8242WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 102s 513ms/step - accuracy: 0.2781 - loss: 1.8241 - val_accuracy: 0.2728 - val_loss: 1.8203
Epoch 19/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 441ms/step - accuracy: 0.2729 - loss: 1.8068WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 103s 519ms/step - accuracy: 0.2728 - loss: 1.8068 - val_accuracy: 0.2783 - val_loss: 1.8162
Epoch 20/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 431ms/step - accuracy: 0.2772 - loss: 1.7999WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 101s 509ms/step - accuracy: 0.2771 - loss: 1.7999 - val_accuracy: 0.2816 - val_loss: 1.8130
Epoch 21/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 431ms/step - accuracy: 0.2657 - loss: 1.8154WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 101s 507ms/step - accuracy: 0.2657 - loss: 1.8154 - val_accuracy: 0.2794 - val_loss: 1.8120
Epoch 22/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step - accuracy: 0.2727 - loss: 1.8065WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 101s 507ms/step - accuracy: 0.2727 - loss: 1.8065 - val_accuracy: 0.2838 - val_loss: 1.8075
Epoch 23/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step - accuracy: 0.2757 - loss: 1.8024WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 101s 506ms/step - accuracy: 0.2757 - loss: 1.8024 - val_accuracy: 0.2823 - val_loss: 1.8049
Epoch 24/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 0s 434ms/step - accuracy: 0.2696 - loss: 1.7960WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file for
199/199 ━━━━━━━━━━━━━━━━━━━━ 102s 510ms/step - accuracy: 0.2697 - loss: 1.7960 - val_accuracy: 0.2863 - val_loss: 1.8028
Epoch 25/25
199/199 ━━━━━━━━━━━━━━━━━━━━ 106s 531ms/step - accuracy: 0.2705 - loss: 1.8100 - val_accuracy: 0.2794 - val_loss: 1.8047
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format
```




```
Epoch 1/5
199/199 ━━━━━━━━━━━━━━━━━━━━ 124s 569ms/step - accuracy: 0.2846 - loss: 1.7756 - val_accuracy: 0.3354 - val_loss: 1.7010
Epoch 2/5
199/199 ━━━━━━━━━━━━━━━━━━━━ 104s 525ms/step - accuracy: 0.3371 - loss: 1.7136 - val_accuracy: 0.3523 - val_loss: 1.6620
Epoch 3/5
199/199 ━━━━━━━━━━━━━━━━━━━━ 104s 523ms/step - accuracy: 0.3501 - loss: 1.6719 - val_accuracy: 0.3808 - val_loss: 1.6197
Epoch 4/5
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 527ms/step - accuracy: 0.3726 - loss: 1.6330 - val_accuracy: 0.3881 - val_loss: 1.5884
Epoch 5/5
199/199 ━━━━━━━━━━━━━━━━━━━━ 105s 526ms/step - accuracy: 0.3672 - loss: 1.6189 - val_accuracy: 0.3940 - val_loss: 1.5588
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

Predicted Emotion: Angry
```

Result: Thus, to develop an application that is based on a convolutional neural network in Keras/TensorFlow implemented successfully and output is verified.