EX. NO: 05	IMPLEMENTATION OF IMAGE AGUMENTATION TECHNIQUES
DATE:	

AIM:

To apply various data augmentation techniques to image datasets and analyze their effects on deep learning models.

ALGORITHM:

1. Import necessary libraries.
2. Load an image dataset.
3. Apply different data augmentation techniques:
    - Horizontal shift
    - Vertical shift
    - Horizontal flip
    - Rotation
    - Brightness adjustment
    - Zoom
4. Generate and display augmented images.
5. Analyze the visual effects of augmentation.

REQUIREMENTS:

pip install tensorflow

pip install matplotlib

PROGRAM:

 # HORIZONTAL SHIFT AUGMENTATION

```
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot
# Load the image
img = load_img('cat.jpeg')
# Convert to numpy array
data = img_to_array(img)
# Expand dimension to one sample
samples = expand_dims(data, 0)
# Create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-200,200])
# Prepare iterator
```

```
it = datagen.flow(samples, batch_size=1)
# Generate samples and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = next(it)
    image = batch[0].astype('uint8')
    pyplot.imshow(image)
pyplot.show()
```

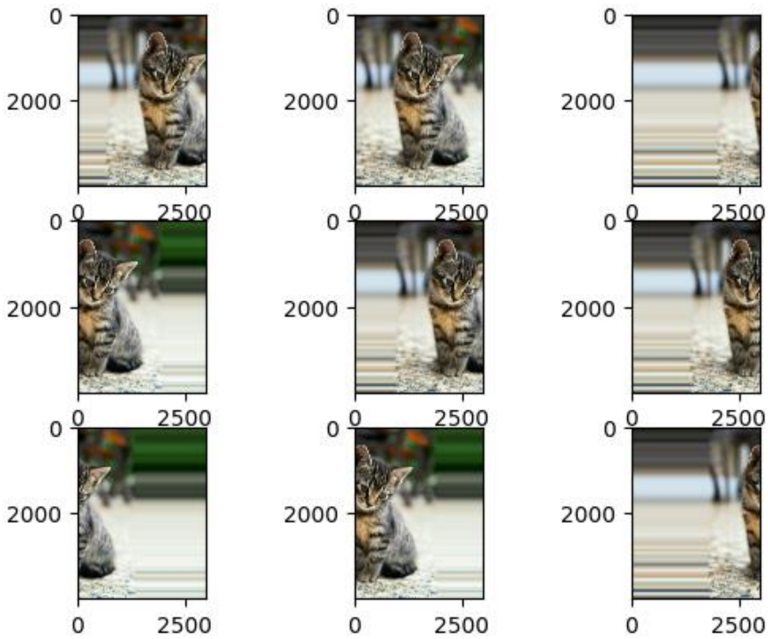
OUTPUT:

VERTICAL SHIFT AUGMENTATION

```
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot
# Load the image
img = load_img('cat.jpeg')
# Convert to numpy array
data = img_to_array(img)
# Expand dimension to one sample
samples = expand_dims(data, 0)
# Create image data augmentation generator
datagen = ImageDataGenerator(height_shift_range=0.5)
# Prepare iterator
```

```
it = datagen.flow(samples, batch_size=1)
# Generate samples and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = next(it)
    image = batch[0].astype('uint8')
 pyplot.imshow(image)
pyplot.show()
```

OUTPUT:

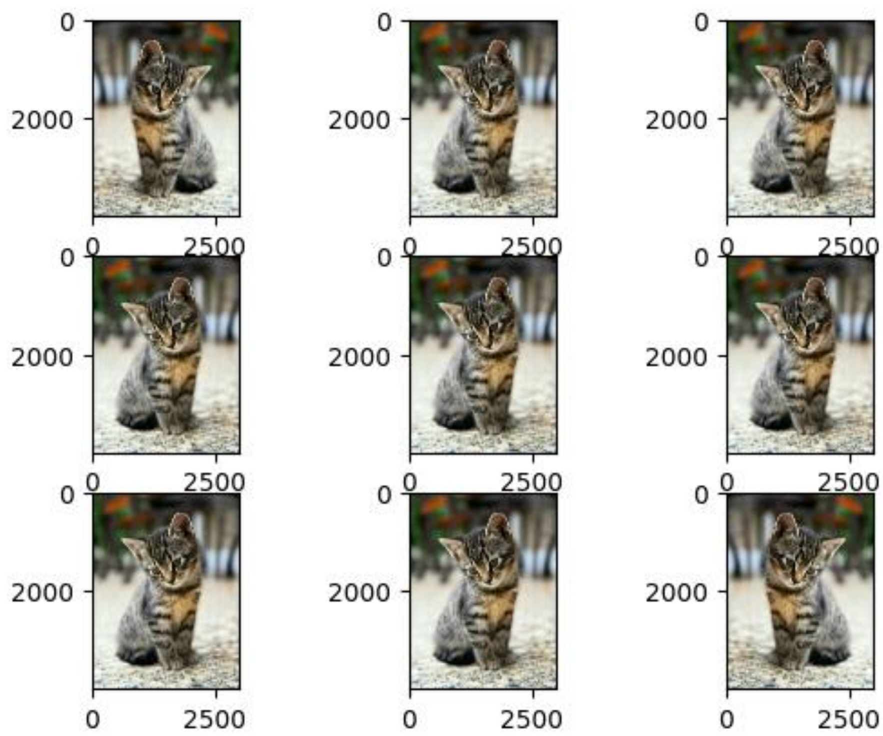


HORIZONTAL FLIP AUGMENTATION

```
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot
# Load the image
img = load_img('cat.jpeg')
# Convert to numpy array
data = img_to_array(img)
# Expand dimension to one sample
samples = expand_dims(data, 0)
# Create image data augmentation generator
datagen = ImageDataGenerator(horizontal_flip=True)
# Prepare iterator
```

```
it = datagen.flow(samples, batch_size=1)
# Generate samples and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = next(it)
    image = batch[0].astype('uint8')
pyplot.imshow(image)
pyplot.show()
```

OUTPUT:



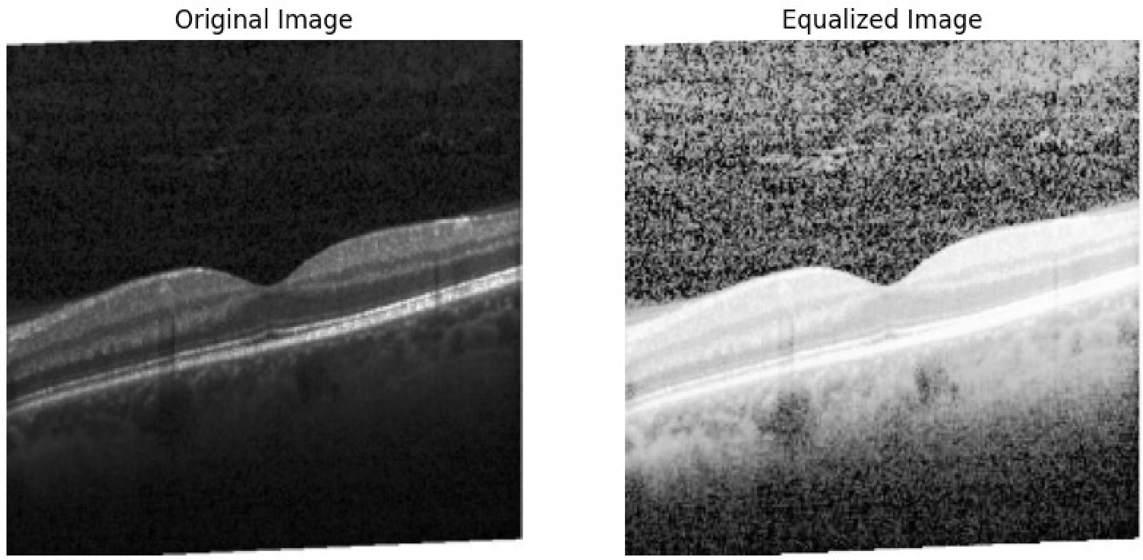
RESULT:

Thus, the Implementation of Image Augmentation Techniques has been executed successfully.

CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('image.jpeg', cv2.IMREAD_GRAYSCALE)
if image is None:
    print("Image not found!")
    exit()
equalized_image = cv2.equalizeHist(image)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')
plt.show()
cv2.imwrite('equalized_image.jpg', equalized_image)
```

OUTPUT:




```
]: True
```

Code:

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pickle
with open('tokenizer.pkl', 'rb') as f:
    tokenizer = pickle.load(f)

max_length = 34

model = load_model('model.h5')
```

```
def extract_features(image_path):
    base_model = VGG16()
    model_vgg = Model(inputs=base_model.inputs, outputs=base_model.layers[-2].output)

    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)

    feature = model_vgg.predict(image, verbose=0)
    return feature

def generate_caption(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)

        word = None
        for w, index in tokenizer.word_index.items():
            if index == yhat:
                word = w
                break
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    final_caption = in_text.replace('startseq', '').replace('endseq', '').strip()
    return final_caption

image_paths = [
    'image1.png',
    'image2.png',
    'image3.png',
    'image4.png'
]

for path in image_paths:
    photo_feature = extract_features(path)
    caption = generate_caption(model, tokenizer, photo_feature, max_length)
    print(f"{path}: {caption}")
```

output:
tensorflow not installed

EX. NO: 04	TRAINING DEEP LEARNING MODELS USING TRANSFER LEARNING TECHNIQUES ON IMAGE DATASETS
DATE	

AIM:

To implement deep learning models using transfer learning techniques on image datasets using the InceptionV3 model with TensorFlow and Keras.

ALGORITHM:

Model Training Algorithm:

1. Import necessary libraries.
2. Load and preprocess the dataset.
3. Load the pre-trained InceptionV3 model without the top layer.
4. Add custom layers for binary classification.
5. Freeze the base model layers.
6. Compile the model using Adam optimizer and binary cross-entropy loss.
7. Train and validate the model.
8. Save the trained model.
9. Plot accuracy and loss graphs.

Image Prediction Algorithm:

1. Import necessary libraries.
2. Define paths for test images and check their existence.
3. Load the trained model.
4. Preprocess input images.
5. Predict image class using the model.
6. Display the predicted class label with confidence percentage.
7. Show the image with the classification result.

REQUIREMENTS:

To execute this implementation, install the required Python libraries using:

```
pip install tensorflow
pip install matplotlib
```

PROGRAM:

MODEL TRAINING

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
```

```
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
# Load and preprocess training/testing data
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen  = ImageDataGenerator(rescale=1./255)
train_dir = "dataset/training_set/"
test_dir = "dataset/test_set/"
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(299, 299),
    batch_size=32, class_mode='binary')
test_generator = test_datagen.flow_from_directory(
    test_dir, target_size=(299, 299),
    batch_size=32, class_mode='binary')
print("Dataset successfully loaded!")
# Load pre-trained InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False)
# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)
# Create final model
model = Model(inputs=base_model.input, outputs=predictions)
# Freeze base model layers
for layer in base_model.layers:
    layer.trainable = False
# Compile model
adam = optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
# Train model
history = model.fit(
    train_generator, steps_per_epoch=len(train_generator), epochs=5,
    validation_data=test_generator, validation_steps=len(test_generator))
# Save model
model.save("cat_dog_model.h5")
```

Plot Accuracy & Loss

```
def plot_history(history):
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(epochs, acc, 'bo-', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r*-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(epochs, loss, 'bo-', label='Training Loss')
plt.plot(epochs, val_loss, 'r*-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
plot_history(history)
```

OUTPUT:

```
C:\Program Files\Python311\Lib\contextlib.py:155: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)
[1m251/251[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 1ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/5
[1m251/251[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m529s[0m 2s/step - accuracy: 0.9944 - loss: 0.0159 - val_accuracy: 0.9886 - val_loss: 0.0465
Epoch 4/5
[1m251/251[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 357us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/5
[1m251/251[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m562s[0m 2s/step - accuracy: 0.9947 - loss: 0.0148 - val_accuracy: 0.9842 - val_loss: 0.0699
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, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

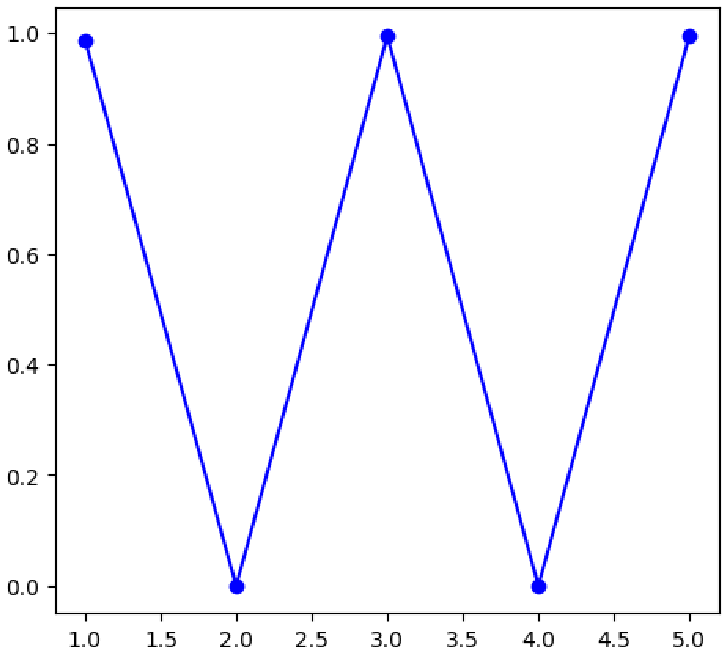


IMAGE PREDICTION:

```
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import os
# Define image paths
cat_image_path = "cat.jpeg"
dog_image_path = "dog.jpg"
# Check if images exist
for img_path in [cat_image_path, dog_image_path]:
    if not os.path.exists(img_path):
        raise FileNotFoundError(f"'{img_path}' NOT found! Upload it before running this code.")
# Load trained model
model = tf.keras.models.load_model("cat_dog_model.h5")
# Define class labels
classes = ['Cat', 'Dog']
# Preprocess input image
def preprocess_input(image_path):
    img = load_img(image_path, target_size=(299, 299))
    img_array = img_to_array(img) / 255.0
```

```
    img_array = np.expand_dims(img_array, axis=0)
    return img_array, img
# Predict function
def predict_image(image_path):
    img_array, img = preprocess_input(image_path)
    prediction = model.predict(img_array)[0][0]
    predicted_class = "Dog" if prediction > 0.5 else "Cat"
    confidence = round(prediction * 100, 2) if prediction > 0.5 else round((1 - prediction) * 100, 2)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"Prediction: {predicted_class} ({confidence}%)")
    plt.show()
# Run predictions
print("Predicting for 'cat.jpeg'...")
predict_image(cat_image_path)
print("Predicting for 'dog.jpeg'...")
predict_image(dog_image_path)
```

OUTPUT:

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

🔍 Predicting for 'cat.jpeg'...

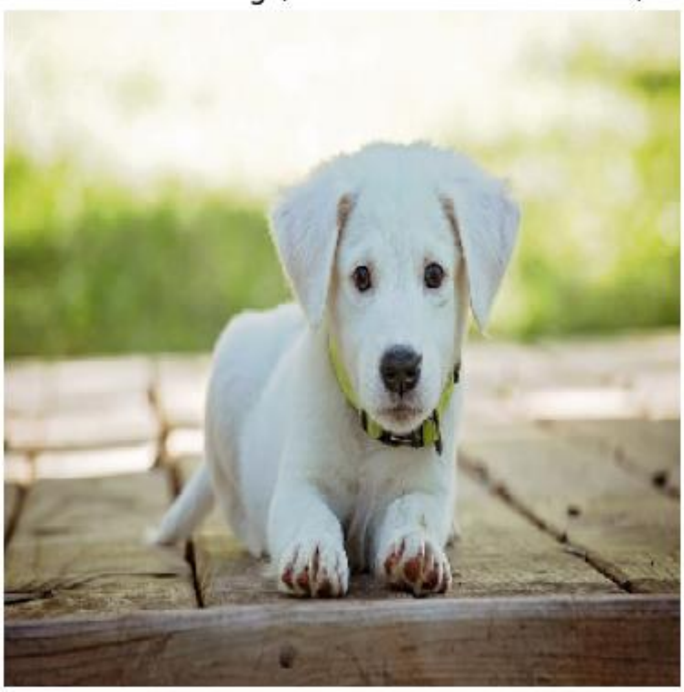
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

Prediction: Cat (100.0%)

🔍 **Predicting for 'cow.jpeg'...**

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step

Prediction: Dog (99.83999633789062%)



RESULT:

Thus, the Implement deep learning models using transfer learning techniques on image datasets.

EX. NO: 02	FEATURE POINT DETECTION USING EDGE AND CORNER DETECTION
DATE:	

AIM:

To implement feature point detection techniques using edge detection (Canny Edge Detector) and corner detection (Harris Corner Detector) in Python.

ALGORITHM:

1. Import the required Python libraries.
2. Load the input image in grayscale format.
3. Resize the image for consistent processing (optional step).
4. Apply the Canny Edge Detector to detect edges in the image.
5. Convert the grayscale image to float32 format (required for Harris Corner Detection).
6. Apply the Harris Corner Detector to identify corners in the image.
7. Dilate the detected corners for better visualization.
8. Highlight strong corners by applying a threshold.
9. Overlay the detected corners on the original image.
10. Display the processed images including edges and corners.

REQUIREMENTS:

To execute this implementation, install the required Python libraries using:

$ pip install numpy

$ pip install opencv-python

$ pip install matplotlib

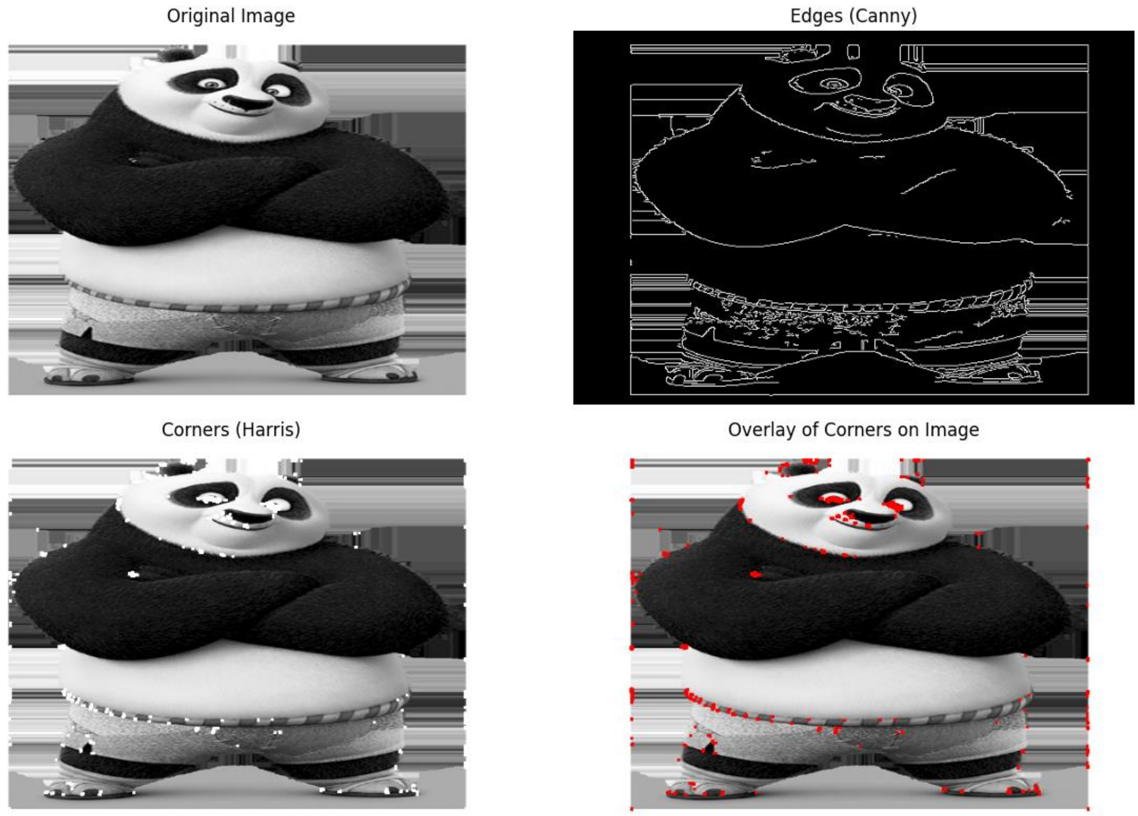
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
```

```
# Resize the image for consistent processing (optional)
image = cv2.resize(image, (600, 400))
# Step 1: Edge Detection (Canny Edge Detector)
edges = cv2.Canny(image, threshold1=100, threshold2=200)
# Step 2: Corner Detection (Harris Corner Detector)
# Convert image to float32 (required for Harris Corner Detection)
gray_float = np.float32(image)
# Apply Harris Corner Detection
corners = cv2.cornerHarris(src=gray_float, blockSize=2, ksize=3, k=0.04)
# Dilate corner detections for better visualization
corners = cv2.dilate(corners, None)
# Threshold to highlight strong corners (corner strength > threshold)
corner_image = np.copy(image)
corner_image[corners > 0.01 * corners.max()] = 255
# Step 3: Overlay detected corners on the original image
overlay = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
overlay[corners > 0.01 * corners.max()] = [0, 0, 255]  # Red corners
# Display results using Matplotlib
plt.figure(figsize=(12, 8))
# Original Image
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
# Edges
plt.subplot(2, 2, 2)
plt.title("Edges (Canny)")
plt.imshow(edges, cmap="gray")
plt.axis("off")
# Corners
plt.subplot(2, 2, 3)
plt.title("Corners (Harris)")
plt.imshow(corner_image, cmap="gray")
plt.axis("off")
# Overlay Image
plt.subplot(2, 2, 4)
plt.title("Overlay of Corners on Image")
```

```
plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

OUTPUT:



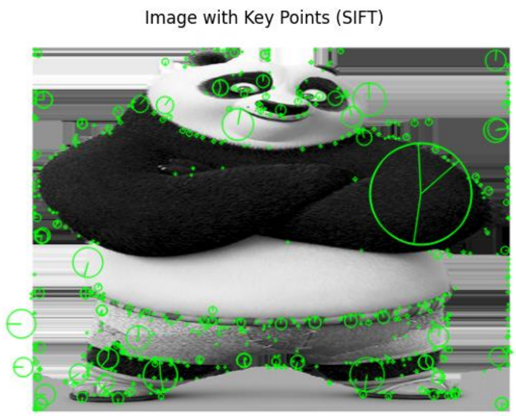
PROGRAM:

#SHIFT INVARIANT FEATURE TRANSFORM

```
import cv2
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
# Resize the image (optional, for better visualization)
image = cv2.resize(image, (600, 400))
# Initialize the SIFT detector
sift = cv2.SIFT_create()
# Detect key points and compute descriptors
keypoints, descriptors = sift.detectAndCompute(image, None)
# Draw the key points on the image
image_with_keypoints = cv2.drawKeypoints(
```

```
    image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS,
color=(0, 255, 0))
# Display the original image and the image with key points
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
# Image with Key Points
plt.subplot(1, 2, 2)
plt.title("Image with Key Points (SIFT)")
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

OUTPUT:



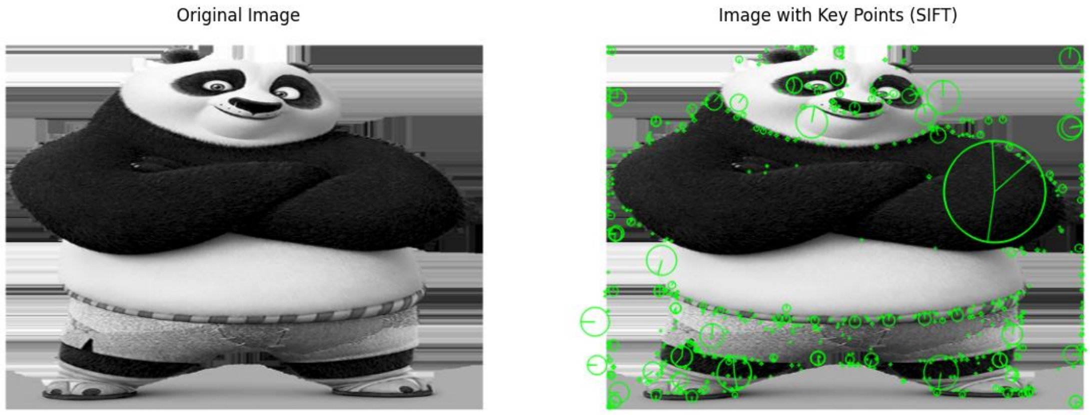
PROGRAM:

#SURF (SPEEDED-UP ROBUST FEATURES)ALGORITHM

```
import cv2
import matplotlib.pyplot as plt
# Load the image in grayscale
image = cv2.imread("panda.png", cv2.IMREAD_GRAYSCALE)
# Resize the image (optional, for better visualization)
image = cv2.resize(image, (600, 400))
# Use SIFT instead of SURF
sift = cv2.SIFT_create()
# Detect key points and compute descriptors
```

```
keypoints, descriptors = sift.detectAndCompute(image, None)
# Draw the key points on the image
image_with_keypoints = cv2.drawKeypoints(
    image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS,
color=(0, 255, 0)
)
# Display the original image and the image with key points
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
# Image with Key Points
plt.subplot(1, 2, 2)
plt.title("Image with Key Points (SIFT)")
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

Thus, the Feature Point extraction has been executed successfully.

EX.NO:01	IMAGE HANDLING AND IMAGE RELATED OPERATIONS
DATE	

AIM:

To implement edge and corner detection techniques using the Canny Edge Detector and Harris Corner Detector in Python.

ALGORITHM:

1. Import the necessary Python libraries (cv2, numpy, matplotlib).
2. Load the input image in grayscale format.
3. Resize the image for consistent processing (optional step).
4. Apply the Canny Edge Detector to detect edges in the image.
5. Convert the grayscale image to float32 format (required for Harris Corner Detection).
6. Apply the Harris Corner Detector to identify corners in the image.
7. Dilate the detected corners for better visualization.
8. Apply a threshold to highlight strong corners.
9. Overlay the detected corners on the original image by marking them in red.
10. Display the processed images, including the original image, edge-detected image, corner-detected image, and overlay image using Matplotlib.

REQUIREMENTS:

```
$ pip install numpy
$ pip install opencv-python
$ pip install matplotlib
```

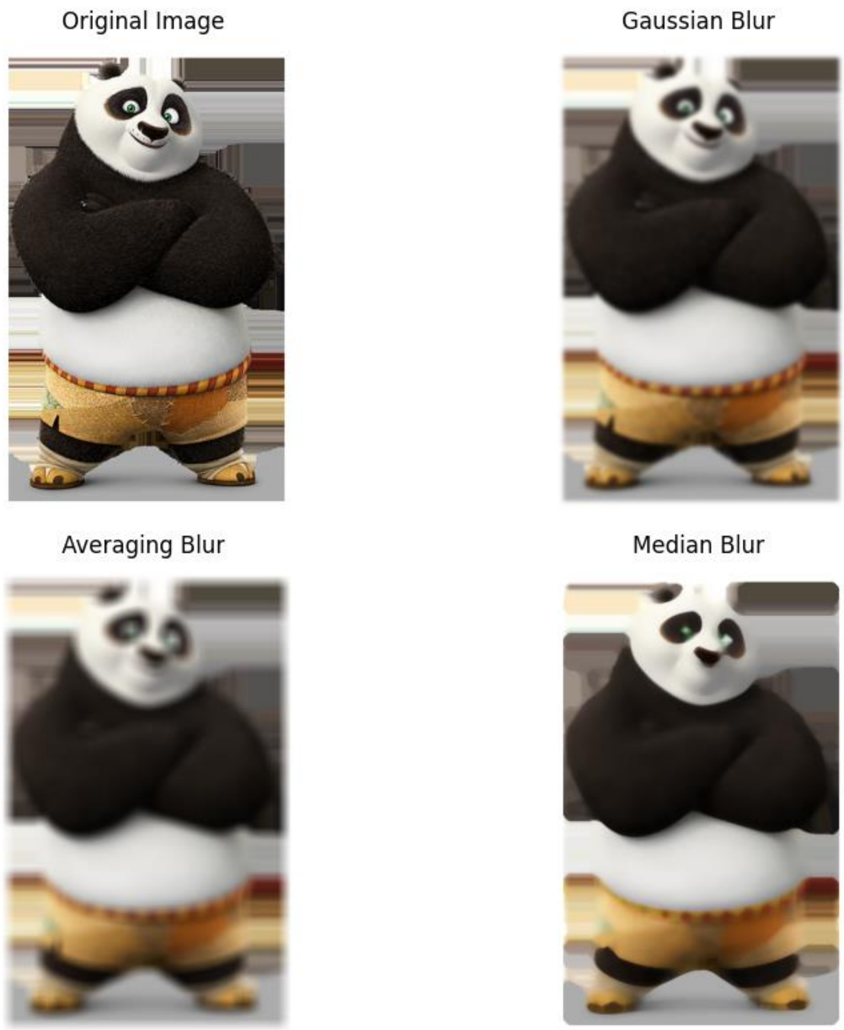
LOW PASS FILTER:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the uploaded image (Ensure the filename is correct)
image = cv2.imread('panda.png', cv2.IMREAD_COLOR)
```

```
# Check if the image is loaded correctly
if image is None:
    raise FileNotFoundError("Error: Image 'panda.png' not found. Ensure it is uploaded in the
notebook.")
# Convert the image from BGR to RGB for correct display in Matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Apply Gaussian Blur (Low-Pass Filter)
gaussian_blur = cv2.GaussianBlur(image, (15, 15), 0)  # Kernel size: 15x15
# Apply Averaging Filter (Low-Pass Filter)
averaging_blur = cv2.blur(image, (15, 15))  # Kernel size: 15x15
# Apply Median Blur
median_blur = cv2.medianBlur(image, 15)  # Kernel size: 15 (must be an odd number)
# Display the original and filtered images
plt.figure(figsize=(10, 8))
# Original image
plt.subplot(2, 2, 1)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
# Gaussian Blur
plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.axis("off")
# Averaging Blur
plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(averaging_blur, cv2.COLOR_BGR2RGB))
plt.title("Averaging Blur")
plt.axis("off")
# Median Blur
plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.axis("off")
```

```
plt.tight_layout()
plt.show()
```

OUTPUT:

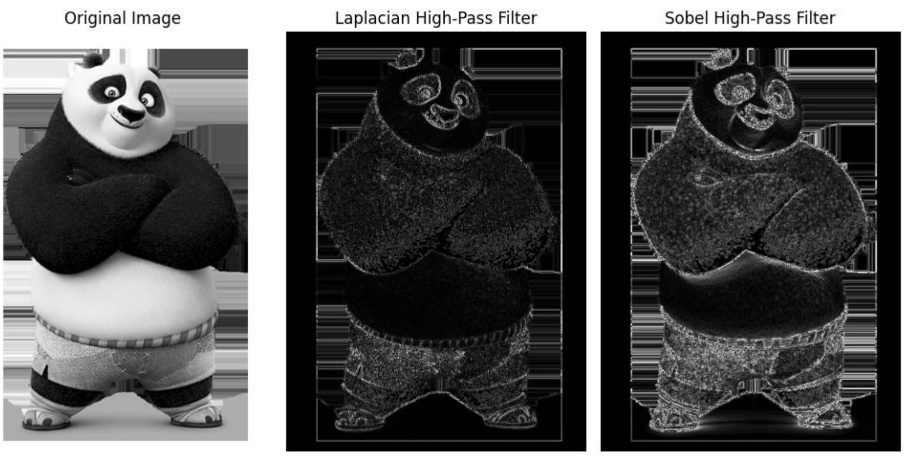


HIGH PASS FILTER:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the uploaded image in grayscale
image = cv2.imread('panda.png', cv2.IMREAD_GRAYSCALE)
# Check if the image is loaded correctly
if image is None:
    raise FileNotFoundError("Error: Image 'panda.png' not found. Ensure it is uploaded in the notebook.")
# Apply High-Pass Filter using Laplacian
```

```
laplacian = cv2.Laplacian(image, cv2.CV_64F)  # Use 64-bit float for higher precision
laplacian = np.uint8(np.absolute(laplacian))  # Convert to unsigned 8-bit
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Derivative in X direction
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Derivative in Y direction
sobel = cv2.magnitude(sobel_x, sobel_y)  # Combine Sobel X and Y
sobel = np.uint8(np.absolute(sobel))  # Convert to unsigned 8-bit
# Display the original and filtered images
plt.figure(figsize=(10, 6))
# Original image
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis("off")
# Laplacian High-Pass Filter
plt.subplot(1, 3, 2)
plt.imshow(laplacian, cmap='gray')
plt.title("Laplacian High-Pass Filter")
plt.axis("off")
plt.subplot(1, 3, 3)
plt.imshow(sobel, cmap='gray')
plt.title("Sobel High-Pass Filter")
plt.axis("off")
plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

Thus, the Image Handling has been executed successfully.

EX. NO:8	IMAGE CAPTION GENERATION
DATE:	

AIM:

To build an image caption generator using a Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) with a pre-trained VGG16 model for feature extraction and an LSTM model for caption generation.

ALGORITHM:

STEP 1: Load and Clean Captions
STEP 2: Extract Features from Images
STEP 3: Prepare Tokenizer and Sequence Data
STEP 4: Generate Training Sequences
STEP 5: Define the Model
STEP 6: Train the Model
STEP 7: Generate Caption for New Images
STEP 8: Evaluate Using BLEU Score

PROGRAM:

```
import os
import string
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from tensorflow.keras.models import Model
def load_captions(captions_path):
    captions_dict = {}
    with open(captions_path, 'r') as f:
        next(f)
        for line in f:
            tokens = line.strip().split(",")
            if len(tokens) < 2:
                continue
            img_id = tokens[0].split('.')[0]
```

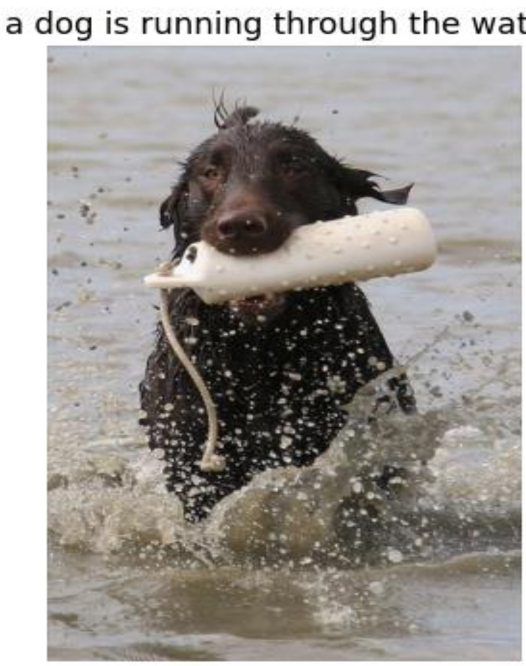
```
            caption = ",".join(tokens[1:]).strip().lower()
            caption = caption.translate(str.maketrans('', '', string.punctuation))
            caption = "startseq " + caption + " endseq"
            captions_dict.setdefault(img_id, []).append(caption)
    return captions_dict
captions_dict = load_captions("captions.txt")
def preprocess_image(image_path):
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    return preprocess_input(img)
def extract_features(image_folder):
    model = VGG16()
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
    features = {}
    for img_file in tqdm(os.listdir(image_folder)):
        img_path = os.path.join(image_folder, img_file)
        img_id = img_file.split('.')[0]
        try:
            img = preprocess_image(img_path)
            feature = model.predict(img, verbose=0)
            features[img_id] = feature
        except:
            continue
    return features
features = extract_features("Image")
captions_dict = {k: v for k, v in captions_dict.items() if k in features}
all_captions = [cap for sublist in captions_dict.values() for cap in sublist]
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(c.split()) for c in all_captions)
def create_sequences(tokenizer, max_length, captions_dict, features, vocab_size):
    X1, X2, y = [], [], []
    for img_id, desc_list in captions_dict.items():
        for desc in desc_list:
            seq = tokenizer.texts_to_sequences([desc])[0]
            for i in range(1, len(seq)):
                in_seq, out_seq = seq[:i], seq[i]
                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                X1.append(features[img_id][0])
                X2.append(in_seq)
                y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
X1, X2, y = create_sequences(tokenizer, max_length, captions_dict, features, vocab_size)
def define_model(vocab_size, max_length):
```

```
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    return model
model = define_model(vocab_size, max_length)
model.fit([X1, X2], y, epochs=10, batch_size=32)
model.save("caption_model.h5")
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
def generate_caption(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text.replace('startseq', '').replace('endseq', '').strip()
from nltk.translate.bleu_score import corpus_bleu
actual, predicted = [], []
for key in list(captions_dict.keys())[:100]:
    photo = features[key].reshape((1, 4096))
    yhat = generate_caption(model, tokenizer, photo, max_length)
    references = [d.split() for d in captions_dict[key]]
    actual.append(references)
    predicted.append(yhat.split())
print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

```
def extract_feature_for_new_image(image_path):
    model = VGG16()
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
    img = preprocess_image(image_path)
    feature = model.predict(img, verbose=0)
    return feature.reshape((1, 4096))
third_test_img_path = "Image/430803349_a66c91f64e.jpg"
third_feature = extract_feature_for_new_image(third_test_img_path)
third_caption = generate_caption(model, tokenizer, third_feature, max_length)
print("Generated Caption:", third_caption)
from PIL import Image as PILImage
plt.imshow(PILImage.open(third_test_img_path))
plt.title(third_caption)
plt.axis("off")
plt.show()
```

OUTPUT:

BLEU Scores:
BLEU-1: 0.68
BLEU-2: 0.49
BLEU-3: 0.38
BLEU-4: 0.31
Generated Caption: a dog is running through the water



RESULT:

Thus, the **IMAGE CAPTION GENERATION** has been implemented successfully.

EX. NO: 07	MULTI OBJECT DETECTION
DATE:	

AIM:

The aim of the code is to perform object detection on an image using a pre-trained MobileNet SSD model, and display the image with bounding boxes around detected objects, labeled with their class names and confidence scores.

ALGORITHM:

STEP 1: Load Required Files:

- Define the paths for the image, model configuration (prototxt), and pre-trained model weights (caffemodel).
- Check if the image and model files exist.

STEP 2 :Load the Pre-trained Model:

- Use OpenCV to load the MobileNet SSD model using cv2.dnn.readNetFromCaffe() with the specified prototxt and caffemodel files.

STEP 3 :Process the Input Image:

- Read the image using cv2.imread(), and get its dimensions.
- Resize the image to 800x600 pixels for display.

STEP 4:Create Blob for Image:

- Create a blob from the image for input into the network, which resizes and normalizes the image to 300x300 pixels.

STEP 5 :Perform Object Detection:

- Pass the blob to the model and get the detections.
- Measure the time taken for the inference.

STEP 6 :Process Detection Results:

- Loop over the detections, checking if the confidence score is above 20%.
- For each detection, extract the class label and bounding box coordinates.

STEP 7 :Draw Bounding Boxes:

- Draw bounding boxes around the detected objects on the resized image and label them with the class and confidence.

STEP 9: Display the Output Image:

- Convert the image from BGR to RGB and display it using PIL's Image.fromarray() in a Jupyter-friendly format.

PROGRAM:

```
import numpy as np
import cv2
import time
import os
from IPython.display import display
import PIL.Image as Image
# Define paths to your files
image_path = "C:\\Users\\3122246002016\\Downloads\\image.jpg"  # Update with actual image path
prototxt_path = "C:\\Users\\3122246002016\\Downloads\\MobileNetSSD_deploy.prototxt"  # Update with actual prototxt file
model_path = "C:\\Users\\3122246002016\\Downloads\\MobileNetSSD_deploy.caffemodel"  # Update with actual caffemodel file
# Check if image file exists
if not os.path.exists(image_path):
  raise FileNotFoundError("Error: Image file not found!")
# Check if model files exist
if not os.path.exists(prototxt_path) or not os.path.exists(model_path):
  raise FileNotFoundError("Error: Model files not found!")
# Load pre-trained model
print("[INFO] Loading model...")
net = cv2.dnn.readNetFromCaffe(prototxt_path, model_path)
# Load and process image
print("[INFO] Processing image...")
image = cv2.imread(image_path)
(h, w) = image.shape[:2]
# Resize image for display
display_width, display_height = 800, 600
image_resized = cv2.resize(image, (display_width, display_height))
# Compute scaling factors for bounding box transformation
x_scale = display_width / w
y_scale = display_height / h
# Create a blob for image normalization
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843, (300, 300), 127.5)
# Run object detection
print("[INFO] Running object detection...")
start_time = time.time()
```

```
net.setInput(blob)
detections = net.forward()
end_time = time.time()
print(f"[INFO] Inference Time: {end_time - start_time:.2f} seconds")
# Class Labels for MobileNet SSD
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair",
           "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train",
"tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
# Loop over detections
for i in np.arange(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.2:  # Adjust confidence threshold if needed
        idx = int(detections[0, 0, i, 1])
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        # Convert bounding box to resized image scale
        startX, startY, endX, endY = int(startX * x_scale), int(startY * y_scale), int(endX * x_scale), int(endY
* y_scale)
        # Ensure bounding box coordinates are within the image dimensions
        startX, startY, endX, endY = np.clip([startX, startY, endX, endY], 0,
                                    [display_width - 1, display_height - 1, display_width - 1, display_height - 1])
        label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
        print("[INFO] Detected:", label)
        # Draw bounding box
        cv2.rectangle(image_resized, (startX, startY), (endX, endY), COLORS[idx], 2)
        y = startY - 15 if startY - 15 > 15 else startY + 15
        cv2.putText(image_resized, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx],
2)
# Display the output image (Jupyter-friendly)
print("[INFO] Displaying output image...")
image_pil = Image.fromarray(cv2.cvtColor(image_resized, cv2.COLOR_BGR2RGB))
display(image_pil)
```

OUTPUT:

```
[INFO] Loading model...
[INFO] Processing image...
[INFO] Running object detection...
[INFO] Inference Time: 0.03 seconds
[INFO] Detected: bicycle: 76.71%
[INFO] Detected: person: 54.41%
[INFO] Detected: person: 44.39%
[INFO] Detected: person: 40.97%
[INFO] Displaying output image...
```

RESULT:

The output will be an image with bounding boxes around detected objects, labeled with their names and confidence percentages. The console will display the detected objects along with their confidence scores.

EX. NO: 06	VISION BASED OBJECT RECOGNITION USING STATE OF THE ART CNN
DATE:	

AIM:

To classify an image using multiple deep learning models from the Keras Applications module. It evaluates the image using VGG16, VGG19, ResNet50, InceptionV3, and Xception, all pretrained on ImageNet.

ALGORITHM:

STEP 1: Import the libraries

- TensorFlow/Keras models (VGG16, VGG19, ResNet50, InceptionV3, Xception)
- Image preprocessing (cv2, numpy, matplotlib, img_to_array, load_img)

STEP 2: Define Image Classification Models

- A dictionary maps model names to their respective Keras classes.

STEP 3: Load and Preprocess the Image

- Load the image using OpenCV and convert it to an RGB array.
- Resize it according to the input size required by each model.
- Normalize and preprocess the image for each model.

STEP 4: Perform Classification with Each Model

- Load the pre-trained model with weights="imagenet".
- Predict the class of the image.
- Decode the top-1 predicted class label and probability.

STEP 5: Display Results

- Show the input image using matplotlib.
- Print out the predicted label and probability for each model.

PROGRAM:

```
from tensorflow.keras.applications import ResNet50, InceptionV3, Xception, VGG16, VGG19, imagenet_utils
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np
import cv2
import matplotlib.pyplot as plt
# Path to the input image
image_path = "C:\\Users\\3122246002016\\Downloads\\soccer_ball.jpg"  # Change this to your image file
# Dictionary of models
MODELS = {
```

```
    "VGG16": VGG16,
    "VGG19": VGG19,
    "ResNet50": ResNet50,
    "InceptionV3": InceptionV3,
    "Xception": Xception,
}
# Load and preprocess image
data = {}
for model_name in MODELS.keys():
    # Set input size based on the model
    inputShape = (224, 224) if model_name not in ("InceptionV3", "Xception") else (299, 299)
    preprocess = imagenet_utils.preprocess_input if model_name not in ("InceptionV3", "Xception") else
preprocess_input
    # Load and preprocess image
    image = load_img(image_path, target_size=inputShape)
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess(image)
    data[model_name] = image
# Run classification on all models
results = {}
for model_name, ModelClass in MODELS.items():
    print(f"[INFO] Loading {model_name} model...")
    model = ModelClass(weights="imagenet")
    print(f"[INFO] Classifying image with {model_name}...")
    preds = model.predict(data[model_name])
    P = imagenet_utils.decode_predictions(preds)
    imagenetID, label, prob = P[0][0]  # Get top prediction
    results[model_name] = (label, prob * 100)
# Load the original image
orig = cv2.imread(image_path)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB)
# Display results
plt.figure(figsize=(10, 6))
plt.imshow(orig)
plt.axis("off")
plt.title("Model Predictions")
```

```
plt.show()
# Print results for each model
for model_name, (label, prob) in results.items():
    print(f"{model_name}: {label} ({prob:.2f}%)")
```

OUTPUT:

```
[INFO] Loading VGG16 model...
[INFO] Classifying image with VGG16...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step
[INFO] Loading VGG19 model...
[INFO] Classifying image with VGG19...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
[INFO] Loading ResNet50 model...
[INFO] Classifying image with ResNet50...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
[INFO] Loading InceptionV3 model...
[INFO] Classifying image with InceptionV3...
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
[INFO] Loading Xception model...
[INFO] Classifying image with Xception...
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002169DDE71A0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 754ms/step
```


```
VGG16: soccer_ball (99.97%)
VGG19: soccer_ball (99.89%)
ResNet50: soccer_ball (99.92%)
InceptionV3: soccer_ball (100.00%)
Xception: soccer_ball (98.26%)
```

RESULT:

Thus, the model classified the image with VGG16, VGG19, Resnet50, InceptionV3 and Xception.

EX. NO: 03	VISUALIZATION OF NETWORK ARCHITECTURE AND FEATURE MAP	
DATE:		

AIM:

To visualize and understand the architecture, feature extraction process, and hierarchical learning capabilities of the VGG16 model by analyzing its feature maps and filters.

ALGORITHM:

A)FILTER ONE (To Extract and Visualize Filters from VGG16)

STEP 1: Load the Pretrained VGG16 Model

- Import the required libraries.
- Load the VGG16 model with pretrained weights.

STEP 2: Extract Convolutional Filters

- Iterate through all layers in the model.
- Check if a layer is a convolutional layer.
- Retrieve the filter weights of each convolutional layer.

STEP 3: Process Filters for Visualization

- Normalize filter values to a range of 0–1 for better visualization.
- Extract individual filters (each filter is a 3D tensor: width × height × depth).
- Convert them to 2D grayscale images for visualization.

STEP 4: Display Filters Using Matplotlib

- Use matplotlib.pyplot to create a grid layout.
- Plot multiple filters from the first convolutional layer.

PROGRAM:

```
from keras.applications.vgg16 import VGG16 from matplotlib import pyplot
# load the model model = VGG16()
# summarize filter shapes for layer in model.layers:
# check for convolutional layer
if 'conv' not in layer.name: continue
# get filter weights
filters, biases = layer.get_weights() print(layer.name, filters.shape)
```

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 ━━━━━━━━━━━━━━━━━━━━ 53s 0us/step
block1_conv1 (3, 3, 3, 64)
block1_conv2 (3, 3, 64, 64)
block2_conv1 (3, 3, 64, 128)
block2_conv2 (3, 3, 128, 128)
block3_conv1 (3, 3, 128, 256)
block3_conv2 (3, 3, 256, 256)
block3_conv3 (3, 3, 256, 256)
block4_conv1 (3, 3, 256, 512)
block4_conv2 (3, 3, 512, 512)
block4_conv3 (3, 3, 512, 512)
block5_conv1 (3, 3, 512, 512)
block5_conv2 (3, 3, 512, 512)
block5_conv3 (3, 3, 512, 512)
```

B)FILTER TWO (To Extract and Visualize Filters from VGG16's Second Convolutional Layer)

ALGORITHM:

STEP 1: Load the Pre-trained Model

- Import the necessary libraries.
- Load the VGG16 model with pre-trained ImageNet weights.

STEP 2: Extract Weights from the Second Layer

- Retrieve filters (weights) and biases from the second convolutional layer.
- Normalize filter values between 0 and 1 to make them suitable for visualization.

STEP 3: Set Up Plotting Parameters

- Define the number of filters to visualize (n_filters = 6).
- Use a loop to iterate over the selected filters.

STEP 4: Visualize Filters

- Extract each filter from the layer.
- Loop through its 3 color channels (RGB).
- Plot each channel separately in grayscale.

STEP 5: Display the Plots

- Use matplotlib.pyplot to generate the plots.
- Remove axis labels for a cleaner visualization.
- Show the final figure.

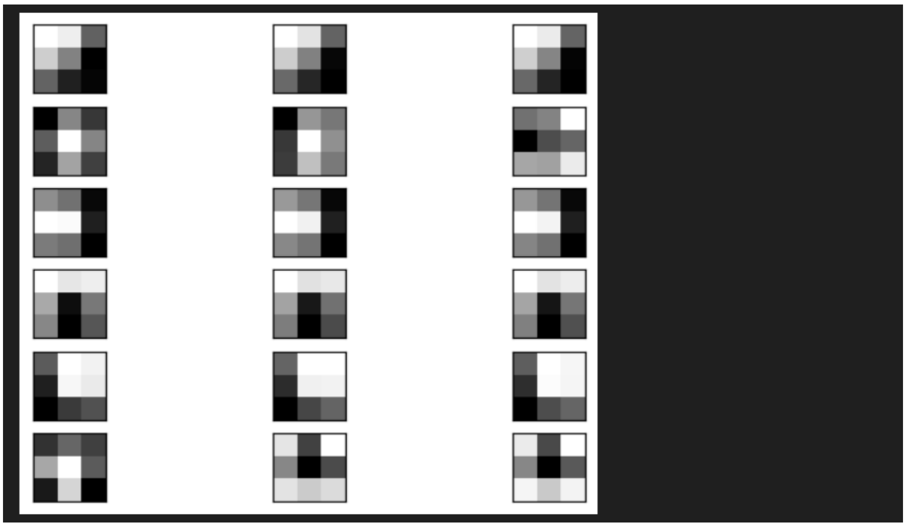
PROGRAM:

```
from keras.applications.vgg16 import VGG16 from matplotlib import pyplot

# load the model model = VGG16()

# retrieve weights from the second hidden layer filters, biases = model.layers[1].get_weights()

# normalize filter values to 0-1 so we can visualize them f_min, f_max = filters.min(), filters.max()

filters = (filters - f_min) / (f_max - f_min) # plot first few filters

n_filters, ix = 6, 1

for i in range(n_filters): # get the filter

f = filters[:, :, :, i]

# plot each channel separately for j in range(3):

# specify subplot and turn of axis ax = pyplot.subplot(n_filters, 3, ix) ax.set_xticks([])

ax.set_yticks([])

# plot filter channel in grayscale pyplot.imshow(f[:, :, j], cmap='gray')
```

ix += 1

show the figure pyplot.show()

OUTPUT:



C)FILTER THREE (For Visualizing Feature Maps of VGG16's First Convolutional Layer)

ALGORITHM:

STEP 1: Load the Pre-trained Model:

- Import necessary libraries.
- Load the VGG16 model with pre-trained ImageNet weights.

STEP 2: Modify the Model to Extract Feature Maps:

- Redefine the model to output activations from the first convolutional layer.
- Display model architecture using summary().

STEP 3: Load and Preprocess an Image:

- Load an image from the local directory.
- Resize it to the required input shape (224 × 224 pixels).
- Convert the image to an array and expand dimensions to match model input requirements.
- Apply preprocess_input() for pixel scaling.

STEP 4: Generate Feature Maps:

- Pass the preprocessed image through the modified model.
- Extract feature maps from the first convolutional layer.

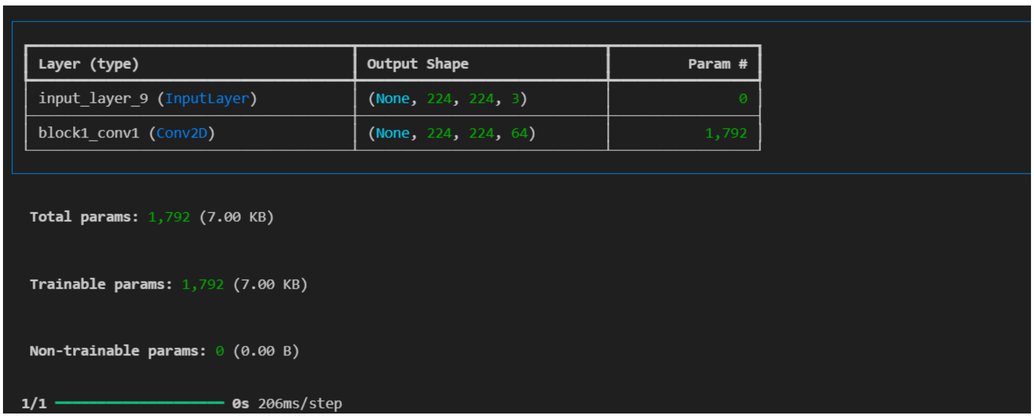
STEP 5: Visualize Feature Maps:

- Define an 8×8 grid layout to plot 64 feature maps.
- Use a loop to iterate through all 64 feature maps.
- Use matplotlib.pyplot to display each feature map in grayscale.
- Remove axis ticks for a cleaner visualization.

PROGRAM:

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array from keras.models import Model
from matplotlib import pyplot from numpy import expand_dims # load the model
model = VGG16()
# redefine model to output right after the first hidden layer
model = Model(inputs=model.inputs, outputs=model.layers[1].output) model.summary()
img = load_img("C:\\Users\\3122246002016\\Downloads\\flower.jpg", target_size=(224, 224)) # convert
the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single 'sample' img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg)
img = preprocess_input(img)
# get feature map for first hidden layer feature_maps = model.predict(img)
# plot all 64 maps in an 8x8 squares square = 8
ix = 1
for _ in range(square): for _ in range(square):
# specify subplot and turn of axis
ax = pyplot.subplot(square, square, ix) ax.set_xticks([])
ax.set_yticks([])
# plot filter channel in grayscale pyplot.imshow(feature_maps[0, :, :, ix-1], cmap='gray') ix += 1
# show the figure
pyplot.show()
```

OUTPUT:



Layer (type)	Output Shape	Param #
input_layer_9 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792

Total params: 1,792 (7.00 KB)

Trainable params: 1,792 (7.00 KB)

Non-trainable params: 0 (0.00 B)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step

D)FILTER FOUR (For Visualizing Feature Maps from Multiple Layers of VGG16)

ALGORITHM:

STEP 1: Load the Pre-trained VGG16 Model

- Import necessary libraries.
- Load the VGG16 model with pre-trained ImageNet weights.

STEP 2: Modify the Model to Extract Feature Maps from Specific Layers

- Define a list of layer indices to extract outputs from (ixs = [2, 5, 9, 13, 17]).
- Retrieve the output of these layers and redefine the model accordingly.

STEP 3: Load and Preprocess an Image

- Load the input image and resize it to (224, 224).
- Convert it into a NumPy array.
- Expand dimensions to match the input format required by VGG16.
- Apply preprocess_input() for pixel normalization.

STEP 4: Generate Feature Maps from the Selected Layers

- Pass the preprocessed image through the modified model.
- Extract feature maps from the selected convolutional layers.

STEP 5: Visualize Feature Maps from Each Layer

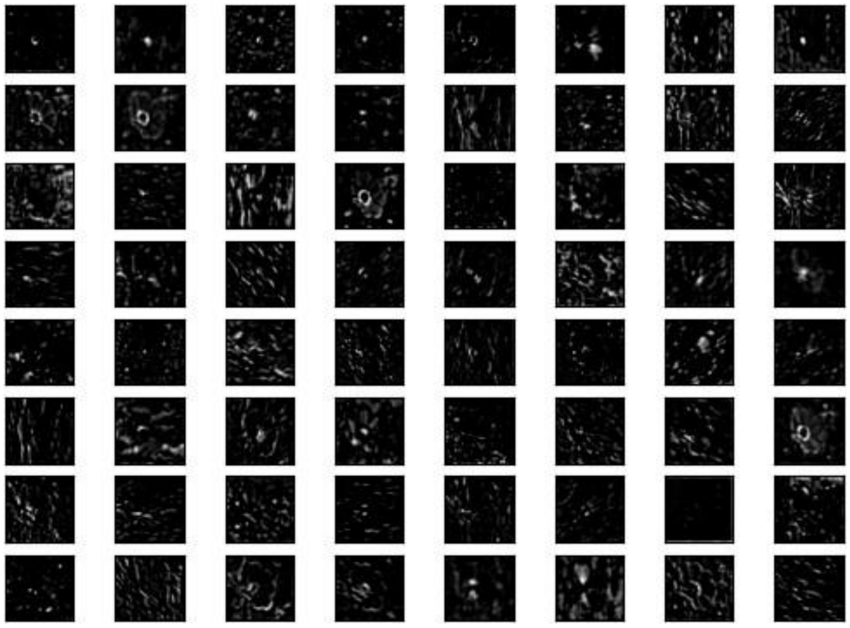
- Iterate through each extracted feature map.
- Define an 8×8 grid layout for visualization.
- Use nested loops to plot the first 64 feature maps in grayscale.
- Remove axis labels for a clean display.
- Show the figure for each layer separately.

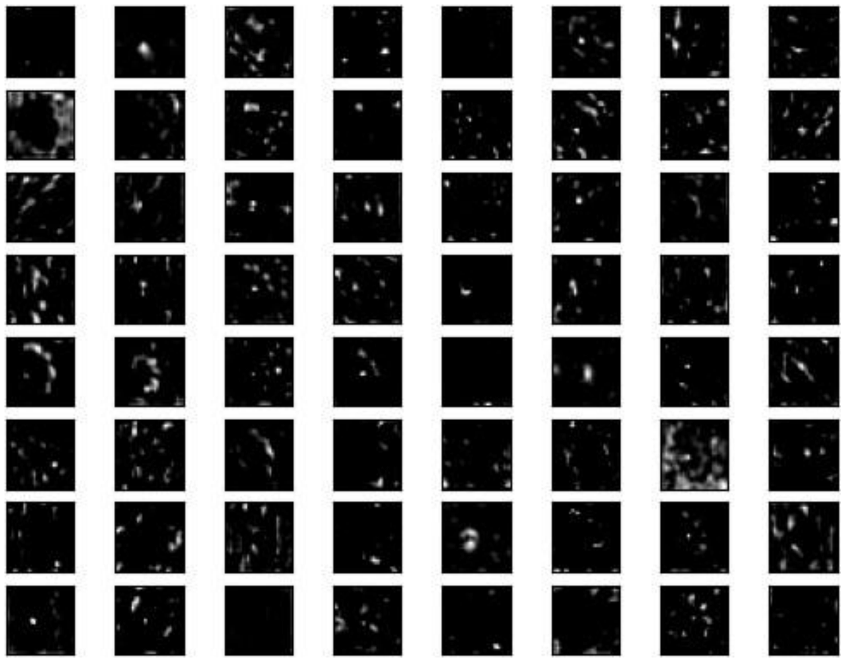
PROGRAM:

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array from keras.models import Model
from matplotlib import pyplot from numpy import expand_dims # load the model
model = VGG16()
# redefine model to output right after the first hidden layer ixs = [2, 5, 9, 13, 17]
outputs = [model.layers[i].output for i in ixs]
model = Model(inputs=model.inputs, outputs=outputs) # load the image with the required shape
img = load_img("C:\\Users\\3122246002016\\Downloads\\flower.jpg", target_size=(224, 224)) # convert the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single 'sample' img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg) img = preprocess_input(img)
```

```
# get feature map for first hidden layer
feature_maps = model.predict(img) # plot the output from each block square = 8
for fmap in feature_maps:
# plot all 64 maps in an 8x8 squares ix = 1
for _ in range(square): for _ in range(square):
# specify subplot and turn of axis
ax = pyplot.subplot(square, square, ix) ax.set_xticks([])
ax.set_yticks([])
# plot filter channel in grayscale pyplot.imshow(fmap[0, :, :, ix-1], cmap='gray') ix += 1
# show the figure
pyplot.show()
```

OUTPUT:





RESULT:

Thus, the VGG16 model's architecture and feature extraction process were successfully visualized. Feature maps and filters demonstrated how the model detects edges, textures, and patterns, showcasing its hierarchical learning capability.