# BLG 453E – Computer Vision
# Homework 4

**Student:** Emil Huseynov
**Student ID:** 150210906
**Course:** BLG453E

**Assignment:** Homework 4
**Due Date:** January 27, 2025

# Contents

# 1 Introduction

This report is prepared as the submission for **Assignment IV** of the *BLG 453E – Computer Vision* course.

The assignment is composed of four questions:

- **Q1 (20 pts):** Region-Growing Segmentation
- **Q2 (20 pts):** Diffusion Models for Text-to-Image, Text-to-Video, Image-to-Image, and Image-to-Video
- **Q3 (30 pts):** Optical Flow for Robot Arm Tracking
- **Q4 (30 pts):** Principal Component Analysis and Transfer Learning

In the following sections, each question is addressed with a detailed description of the methodology, the code, and relevant discussion of any results and issues encountered. All code listings are provided in separate code blocks for clarity.

# 2 Q1 – Region-Growing

## 2.1 Problem Statement

The task is to implement a simple region-growing algorithm for image segmentation. A single seed point (or multiple seeds) is selected in the `lion.jpg` image, and the region is grown by adding neighboring pixels based on a color similarity threshold. The goals are:

- Select a single seed point to segment the lion, annotating the seed with a red dot.
- Repeat the process with three separate seed points.
- Optionally, showcase three different scenarios using three seed points simultaneously and compare the results.

## 2.2 Methodology

The key steps in my solution are:

1. Read the input image using `OpenCV` in `BGR` format.

2. Define a `get_neighbors` function to retrieve the 8-connected neighbors of a given pixel.
3. Maintain a `visited` mask and a `queue` to perform a Breadth-First Search (BFS) type region growing:
   - Initialize the region with the seed pixel.
   - Compute color difference against the *current mean color* of the region; add neighbors if within threshold.
4. Create a visualization by zeroing out all pixels outside the region and placing a red circle on the seed coordinate.
5. Repeat the process for multiple seeds.

## 2.3 Code

```python
import cv2 # For images
import matplotlib.pyplot as plt

def show_image_in_colab(image_bgr, title="Output"):
    """
    Displays a BGR image in Colab using Matplotlib (converted to
        RGB).
    """
    image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(6, 6))
    plt.imshow(image_rgb)
    plt.title(title)
    plt.axis('off')
    plt.show()

def abs_value(a):
    """Manual absolute value (since abs() is a built-in)."""
    if a < 0:
        return -a
    return a

def color_difference(c1, c2):
    """Difference between two BGR colors c1=(B1,G1,R1), c2=(B2,G2,
        R2)."""
    return (abs_value(c1[0] - c2[0]) +
            abs_value(c1[1] - c2[1]) +
            abs_value(c1[2] - c2[2]))

def get_neighbors(x, y, width, height):
```

```
    """Return the 8-connected neighbors of (x,y) within image
        bounds."""
    neighbors = []
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                continue
            nx = x + dx
            ny = y + dy
            if 0 <= nx < width and 0 <= ny < height:
                neighbors.append((nx, ny))
    return neighbors

def region_grow_segment(image, seed_x, seed_y, threshold=30):
    """
    Perform region growing on 'image' from seed (seed_x, seed_y).
    Returns:
        mask: 2D list of booleans
        out_image: a copy of original image with region in color,
                   background black, and seed marked with a red
                       circle.
    """
    height, width, _ = image.shape

    # Prepare visited and mask
    visited = []
    mask = []
    for _h in range(height):
        visited.append([False]*width)
        mask.append([False]*width)

    # Initialize BFS queue
    queue = []

    # Starting color from seed
    seed_color = image[seed_y, seed_x] # BGR
    sum_b = int(seed_color[0])
    sum_g = int(seed_color[1])
    sum_r = int(seed_color[2])
    region_size = 1

    # Enqueue seed
    queue.append((seed_x, seed_y))
    visited[seed_y][seed_x] = True
    mask[seed_y][seed_x] = True

    # BFS
    while len(queue) > 0:
        x, y = queue.pop(0)
```

```

        # Current region mean color
        mean_b = sum_b // region_size
        mean_g = sum_g // region_size
        mean_r = sum_r // region_size

        # Check neighbors
        for nx, ny in get_neighbors(x, y, width, height):
            if not visited[ny][nx]:
                visited[ny][nx] = True
                pix_color = image[ny, nx]
                diff = color_difference(pix_color, (mean_b, mean_g, mean_r))
                if diff < threshold:
                    # Accept into region
                    mask[ny][nx] = True
                    queue.append((nx, ny))

                    # Update region color sums
                    sum_b += int(pix_color[0])
                    sum_g += int(pix_color[1])
                    sum_r += int(pix_color[2])
                    region_size += 1

    # Prepare output visualization
    out_image = image.copy()
    for row_i in range(height):
        for col_i in range(width):
            if not mask[row_i][col_i]:
                out_image[row_i, col_i] = [0, 0, 0] # black background

    # Draw a small red circle on the seed point
    cv2.circle(out_image, (seed_x, seed_y), 5, (0, 0, 255), -1)

    return mask, out_image

def region_grow_multiple_seeds(image, seeds, threshold=30):
    """
    Region growing with multiple initial seeds simultaneously.
    seeds: list of (x, y) points
    Returns:
        mask: 2D list of booleans
        out_image: region in color, background black, seeds in red circles.
    """
    height, width, _ = image.shape

    visited = []
```

```
    mask = []
    for _h in range(height):
        visited.append([False]*width)
        mask.append([False]*width)

    queue = []

    # Initialize region color sums
    sum_b, sum_g, sum_r = 0, 0, 0
    region_size = 0

    # Add seeds
    for (sx, sy) in seeds:
        seed_color = image[sy, sx]
        sum_b += int(seed_color[0])
        sum_g += int(seed_color[1])
        sum_r += int(seed_color[2])
        region_size += 1

        visited[sy][sx] = True
        mask[sy][sx] = True
        queue.append((sx, sy))

    # BFS
    while len(queue) > 0:
        x, y = queue.pop(0)

        mean_b = sum_b // region_size
        mean_g = sum_g // region_size
        mean_r = sum_r // region_size

        for nx, ny in get_neighbors(x, y, width, height):
            if not visited[ny][nx]:
                visited[ny][nx] = True
                pix_color = image[ny, nx]
                diff = color_difference(pix_color, (mean_b, mean_g,
                     mean_r))
                if diff < threshold:
                    mask[ny][nx] = True
                    queue.append((nx, ny))
                    sum_b += int(pix_color[0])
                    sum_g += int(pix_color[1])
                    sum_r += int(pix_color[2])
                    region_size += 1

    out_image = image.copy()
    for row_i in range(height):
        for col_i in range(width):
            if not mask[row_i][col_i]:
```

```python
                out_image[row_i, col_i] = [0, 0, 0]

    # Mark each seed with a small red circle
    for (sx, sy) in seeds:
        cv2.circle(out_image, (sx, sy), 5, (0, 0, 255), -1)

    return mask, out_image

# Read input lion image
img = cv2.imread('Lion.jpg')
if img is None:
    print("Could not find 'lion.jpg'. Please upload the image to Colab.")
else:
    # 1) Single seed
    seed_x, seed_y = 150, 250
    mask_single, result_single = region_grow_segment(img, seed_x, seed_y, threshold=30)
    show_image_in_colab(result_single, title="Single Seed Segmentation")

    # 2) Three seeds, separate calls
    seeds_individual = [(200, 400), (400, 200), (900, 600)]
    for i, (sx, sy) in enumerate(seeds_individual):
        msk_sep, res_sep = region_grow_segment(img, sx, sy, threshold=25)
        show_image_in_colab(res_sep, title=f"Separate Seed {i+1} Segmentation")

    # 3) BONUS: Multiple seeds in one run
    seeds_scenario_1 = [(250, 280), (500, 210), (1050, 400)]
    mask_multi_1, result_multi_1 = region_grow_multiple_seeds(img, seeds_scenario_1, threshold=30)
    show_image_in_colab(result_multi_1, title="Multi-Seed Scenario 1")

    seeds_scenario_2 = [(300, 620), (700, 500), (1100, 780)]
    mask_multi_2, result_multi_2 = region_grow_multiple_seeds(img, seeds_scenario_2, threshold=30)
    show_image_in_colab(result_multi_2, title="Multi-Seed Scenario 2")

    seeds_scenario_3 = [(50, 100), (600, 400), (1150, 700)]
    mask_multi_3, result_multi_3 = region_grow_multiple_seeds(img, seeds_scenario_3, threshold=50)
    show_image_in_colab(result_multi_3, title="Multi-Seed Scenario 3")
```

Listing 1: Region-Growing Code for Q1.

## 2.4 Discussion

- The region-growing threshold is a sensitive parameter. Depending on the threshold value, more or fewer background pixels might be included.
- By selecting different seed points, we can segment slightly different portions of the image. For the lion, we focus on areas with similar fur color.
- The bonus scenario shows how using multiple seeds can expand or shrink the final segmented region depending on their initial positions and the threshold.

# 3 Q2 – Diffusion Models: Text-to-Image, Text-to-Video, Image-to-Image, Image-to-Video

## 3.1 Problem Statement

This question involves experimenting with modern diffusion models (e.g., Stable Diffusion, ZeroScope) to generate synthetic images and videos. Specifically, we:

- Generate images from text prompts (Text-to-Image).
- Generate videos from text prompts (Text-to-Video).
- Modify existing images via text prompts (Image-to-Image).
- Generate videos from input images (Image-to-Video).

We utilized publicly available models on Hugging Face, including:

- `runwayml/stable-diffusion-v1-5` for Text-to-Image and Image-to-Image
- `cerspense/zeroscope_v2_576w` for Text-to-Video
- `stabilityai/stable-video-diffusion-img2vid-xt` for Image-to-Video (optionally)

## 3.2 Methodology and Code

Below is the code used in Google Colab for the text-to-image, text-to-video, and image-to-image experiments. GPU acceleration (e.g., T4 runtime) is recommended due to the complexity of diffusion models.

```python
import os
os.environ["HUGGINGFACE_TOKEN"] = "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"
import torch
import IPython
from PIL import Image
from IPython.display import display, Image as IPyImage

from diffusers import StableDiffusionPipeline

# 1) Load the Stable Diffusion text-to-image pipeline
pipe_t2i = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5", # or stable-diffusion-v1-5
    torch_dtype=torch.float16,
    use_auth_token=True
)
pipe_t2i.to("cuda")

# 2) Provide a prompt
prompt_t2i = "A student tries to fly using homemade wings in a sunny day from a mountain"

# 3) Generate the image
image_t2i = pipe_t2i(
    prompt=prompt_t2i,
    num_inference_steps=30,
    guidance_scale=7.5
).images[0]

# 4) Show result
display(image_t2i)


# -------------------------------------------------------
# TEXT TO VIDEO using ZeroScope
# -------------------------------------------------------
import torch
import imageio
import numpy as np
from IPython.display import Image as IPyImage, display
from diffusers import DiffusionPipeline

# Load ZeroScope
```

```
pipe_t2v = DiffusionPipeline.from_pretrained(
    "cerspense/zeroscope_v2_576w",
    torch_dtype=torch.float16,
)
pipe_t2v.to("cuda")

prompt = "A cat dancing in a disco, neon lights"

result = pipe_t2v(
    prompt=prompt,
    num_inference_steps=25,
    guidance_scale=7.5,
    num_frames=16,
    width=576,
    height=320
)

raw_frames = result.frames
arr_4d = raw_frames[0] # shape (16, 320, 576, 3)

video_frames = []
num_total_frames = arr_4d.shape[0]

for i in range(num_total_frames):
    frame = arr_4d[i]
    if frame.dtype != np.uint8:
        frame = frame.astype(np.uint8)
    video_frames.append(frame)

# Save frames as a GIF
gif_path = "zeroscope_output.gif"
imageio.mimsave(gif_path, video_frames, fps=4)
display(IPyImage(filename=gif_path))


# ---------------------------------------------------------
# IMAGE-TO-IMAGE with Stable Diffusion
# ---------------------------------------------------------
from diffusers import StableDiffusionImg2ImgPipeline

pipe_i2i = StableDiffusionImg2ImgPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16,
    use_auth_token=True
).to("cuda")

init_image_path = "tesla.jpg" # Example: some input image
init_img = Image.open(init_image_path).convert("RGB")
init_img = init_img.resize((512, 512)) # resize if needed
```

```python

prompt_i2i = "Make␣it␣look␣like␣a␣fantasy␣landscape␣by␣Studio␣
    Ghibli,␣vibrant␣colors"

image_i2i = pipe_i2i(
    prompt=prompt_i2i,
    image=init_img,
    strength=0.7,
    guidance_scale=8
).images[0]

display(image_i2i)
```

Listing 2: Text-to-Image and Image-to-Image Diffusion Code.

### 3.3 Discussion

- The text-to-image task allows creative generation of images from pure text prompts.
- ZeroScope was used to convert text prompts into short video clips (Text-to-Video). The process can be slow but generates interesting, small video sequences.
- Image-to-image diffusion transforms a given image to match a text description in style or content.
- Depending on GPU resources and inference steps, results may vary significantly.

## 4 Q3 – Optical Flow

### 4.1 Problem Statement

Optical flow is a method to estimate motion between consecutive frames in a video. The assignment focuses on tracking the Baxter robot's arm as it moves. The steps involve:

- Extract frames from a given video at a suitable frame rate.
- Detect feature points (e.g., via Shi–Tomasi corner detection) in the first frame.
- Use Lucas–Kanade optical flow to track these points across subsequent frames.

- Visualize the tracked points on the frames.

## 4.2 Methodology

1. A video file (e.g., `opticalflow_video.mp4`) is loaded. Frames are extracted at a rate of approximately 1 frame per second (or any suitable rate).
2. `cv2.goodFeaturesToTrack` is used on the first frame.
3. `cv2.calcOpticalFlowPyrLK` tracks the motion of these features.
4. For visualization, lines are drawn from old to new positions of each tracked feature.

## 4.3 Code

```python
import cv2 as cv
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import glob

############################
# FRAME EXTRACTION PART
############################

path = "opticalflow_video.mp4"
frames_save_path = "/frames"

def frames_from_video(video_path):
    cam = cv.VideoCapture(video_path) # read video file

    if not os.path.exists(frames_save_path):
        os.makedirs(frames_save_path)

    original_fps = cam.get(cv.CAP_PROP_FPS)
    # We want to save about 1 frame each second
    extraction_rate = int(original_fps) if original_fps > 0 else
        25

    currframe = 0
    while True:
        ret, frame = cam.read()
        if not ret:
```

```python
            break

        if currframe % extraction_rate == 0:
            frame_name = os.path.join(frames_save_path, f"frame_{currframe}.png")
            cv.imwrite(frame_name, frame)
            print(f"Saved: {frame_name}")
        currframe += 1

    cam.release()
    cv.destroyAllWindows()

frames_from_video(path)

#############################
# OPTICAL FLOW PART
#############################
def optical_flow(frames_saved_path):
    # 1) Gather & sort all frames
    frames_path = sorted(glob.glob(os.path.join(frames_saved_path, "*.png")),
                         key=lambda x: int(os.path.splitext(os.path.basename(x))[0].split("_")[1]))
    frames_images = [Image.open(f) for f in frames_path]

    # 2) Parameters
    feature_params = dict(
        maxCorners=100,
        qualityLevel=0.3,
        minDistance=7,
        blockSize=7
    )

    lk_params = dict(
        winSize=(15, 15),
        maxLevel=2,
        criteria=(cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03)
    )

    color = np.random.randint(0, 255, (100, 3))

    # 3) Prepare the first frame
    first_frame_pil = frames_images[0]
    first_frame_pil = first_frame_pil.resize((244, 224))
    first_frame = np.array(first_frame_pil)
    first_frame_gray = cv.cvtColor(first_frame, cv.COLOR_RGB2GRAY)

    # 4) Find corners in the first frame
```

```
    p0 = cv.goodFeaturesToTrack(first_frame_gray, mask=None, **feature_params)

    # 5) Mask for drawing
    mask = np.zeros_like(first_frame)
    flow_points = []

    # 6) Process subsequent frames
    for idx in range(1, len(frames_images)):
        frame_pil = frames_images[idx].resize((244, 224))
        frame_np = np.array(frame_pil)
        frame_gray = cv.cvtColor(frame_np, cv.COLOR_RGB2GRAY)

        p1, st, err = cv.calcOpticalFlowPyrLK(first_frame_gray, frame_gray, p0, None, **lk_params)

        if p1 is not None and st is not None:
            good_new = p1[st == 1]
            good_old = p0[st == 1]

            for i, (new, old) in enumerate(zip(good_new, good_old)):
                a, b = new.ravel()
                c, d = old.ravel()
                mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), 2)
                frame_np = cv.circle(frame_np, (int(a), int(b)), 5, color[i].tolist(), -1)

            img = cv.add(frame_np, mask)
            plt.figure(figsize=(6, 4))
            plt.imshow(img)
            plt.title(f"Optical Flow - Frame {idx}")
            plt.axis('off')
            plt.show()

            first_frame_gray = frame_gray.copy()
            p0 = good_new.reshape(-1, 1, 2)
            flow_points.append(p0)
        else:
            print("No features found in this frame. Stopping.")
            break

optical_flow(frames_save_path)
```

Listing 3: Optical Flow Tracking Code for Q3.

### 4.4 Discussion

- The number of features that remain trackable decreases as more frames are processed (some features move out of view or become occluded).
- If no features are detected in a frame, the process stops or must be reinitialized.
- The optical flow lines give an intuitive visualization of how points move from frame to frame, which helps track the robot arm effectively.

## 5 Q4 – Principal Component Analysis (PCA) and Transfer Learning

### 5.1 Problem Statement

We have a dataset of fruits and vegetables (Kaggle: Fruit-and-Vegetable Image Recognition). The task is split into two main parts:

1. **Basic Feature Extraction & PCA Visualization:** Flatten image pixels (RGB) into a vector and apply PCA to reduce dimensionality, then visualize in 2D.
2. **Transfer Learning & PCA Visualization:** Fine-tune a ResNet-50 model on the training set, then extract features from the penultimate layer, apply PCA, and compare the 2D visual clusters with the basic approach.

### 5.2 Methodology

**Part 1: Basic Feature Extraction.**

1. Resize the images to $224 \times 224$.
2. Flatten each image into a vector of size $3 \times 224 \times 224$.
3. Perform PCA to reduce from $3 \times 224 \times 224$ to 2 dimensions.
4. Cluster the PCA-transformed data using `KMeans` and visualize.

**Part 2: Transfer Learning.**

1. Load a pre-trained ResNet-50 and replace the final fully-connected layer to match the number of classes.

2. Freeze most of the layers except the last block (layer4) and the new fc layer to allow partial fine-tuning.

3. Train on the `train` set, validate on the `validation` set, and save the best model.

4. Use the best model as a feature extractor by taking the output from the penultimate layer (pooling layer).

5. Again, perform PCA on these extracted features and compare the results to the basic features.

## 5.3 Code

```python
import kagglehub

# Download dataset
path = kagglehub.dataset_download("kritikseth/fruit-and-vegetable-image-recognition")
print("Path to dataset files:", path)

import os
print(os.listdir("/root/.cache/kagglehub/datasets/kritikseth/fruit-and-vegetable-image-recognition/versions/8"))

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms, datasets
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)

train_data_path = "/root/.cache/kagglehub/datasets/kritikseth/fruit-and-vegetable-image-recognition/versions/8/train"
val_data_path = "/root/.cache/kagglehub/datasets/kritikseth/fruit-and-vegetable-image-recognition/versions/8/validation"
```

```python
test_data_path = "/root/.cache/kagglehub/datasets/kritikseth/fruit-and-vegetable-image-recognition/versions/8/test"

###################################
# PART 1: BASIC FEATURE EXTRACTION
###################################
basic_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

test_dataset_basic = datasets.ImageFolder(test_data_path, transform=basic_transform)

basic_test_images = []
basic_test_labels = []

for img, label in test_dataset_basic:
    flattened = img.view(-1).numpy() # shape (3*224*224,)
    basic_test_images.append(flattened)
    basic_test_labels.append(label)

basic_test_images = np.array(basic_test_images)
basic_test_labels = np.array(basic_test_labels)

print("Shape of raw pixel feature matrix:", basic_test_images.shape)

# Apply PCA (2D)
pca_basic = PCA(n_components=2)
basic_test_pca = pca_basic.fit_transform(basic_test_images)

# KMeans
number_of_classes = len(test_dataset_basic.classes)
kmeans_basic = KMeans(n_clusters=number_of_classes, random_state=42)
clusters_basic = kmeans_basic.fit_predict(basic_test_pca)

# Visualization
plt.figure(figsize=(8,6))
scatter = plt.scatter(basic_test_pca[:, 0],
                      basic_test_pca[:, 1],
                      c=clusters_basic,
                      cmap='tab10')
plt.title("PCA of Flattened-Pixel Features (Basic Extraction)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(scatter, label="Cluster ID")
plt.show()
```

```python

# Optional function to visualize cluster images
def visualize_cluster_examples(basic_test_dataset, cluster_ids,
    kmeans_obj, k=3):
    import random
    unique_clusters = np.unique(cluster_ids)
    for c in unique_clusters:
        indices = np.where(cluster_ids == c)[0]
        if len(indices) == 0:
            continue
        print(f"\nCluster {c}: showing up to {k} random images")
        chosen = np.random.choice(indices, size=min(k, len(indices)
            ), replace=False)

        fig, axs = plt.subplots(1, max(1, len(chosen)), figsize
            =(15, 3))
        if len(chosen) == 1:
            axs = [axs]

        fig.suptitle(f"Cluster {c} sample images", fontsize=16)
        for i, idx in enumerate(chosen):
            path, label_idx = basic_test_dataset.samples[idx]
            img = plt.imread(path)
            axs[i].imshow(img)
            axs[i].axis('off')
        plt.show()

visualize_cluster_examples(
    test_dataset_basic,
    clusters_basic,
    kmeans_basic,
    k=3
)


###################################
# PART 2: TRANSFER LEARNING
###################################
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

train_dataset = datasets.ImageFolder(train_data_path, transform=
    data_transforms)
val_dataset = datasets.ImageFolder(val_data_path, transform=
    data_transforms)
```

```
test_dataset = datasets.ImageFolder(test_data_path, transform=
    data_transforms)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=
    True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=
    False)

num_classes = len(train_dataset.classes)

model = models.resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False

# Unfreeze the last block
for name, param in model.layer4.named_parameters():
    param.requires_grad = True

# Replace final FC layer
in_features = model.fc.in_features
model.fc = nn.Linear(in_features, num_classes)
for param in model.fc.parameters():
    param.requires_grad = True

model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.
    parameters()), lr=0.001)

def validate(model, val_loader):
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    avg_loss = val_loss / len(val_loader)
    accuracy = 100.0 * correct / total
    return avg_loss, accuracy
```

```python

def train_model(model, train_loader, val_loader, epochs=3):
    best_val_acc = 0.0
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

        train_loss = running_loss / len(train_loader)
        train_acc = 100.0 * correct / total

        val_loss, val_acc = validate(model, val_loader)
        print(f"[Epoch {epoch+1}/{epochs}] "
              f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}% | "
              f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            print("  -> Saving best model...")
            torch.save(model.state_dict(), "best_model.pth")

train_model(model, train_loader, val_loader, epochs=3)

# Load best model
best_model = models.resnet50(pretrained=False)
best_model.fc = nn.Linear(in_features, num_classes)
best_model.load_state_dict(torch.load("best_model.pth", map_location=device))
best_model.to(device)
best_model.eval()

# Feature Extractor
feature_extractor = nn.Sequential(*list(best_model.children())[:-1])
feature_extractor.to(device)
```

```python
feature_extractor.eval()

features = []
test_labels = []

with torch.no_grad():
    for imgs, lbls in test_loader:
        imgs = imgs.to(device)
        feats = feature_extractor(imgs)
        feats = feats.view(feats.size(0), -1)
        features.append(feats.cpu().numpy())
        test_labels.extend(lbls.numpy())

features = np.concatenate(features, axis=0) # shape: [N, 2048]
test_labels = np.array(test_labels)

print("Shape of extracted features (ResNet penultimate):",
    features.shape)

# PCA
pca_resnet = PCA(n_components=2)
pca_features = pca_resnet.fit_transform(features)

kmeans_resnet = KMeans(n_clusters=num_classes, random_state=42)
clusters_resnet = kmeans_resnet.fit_predict(pca_features)

plt.figure(figsize=(8,6))
scatter = plt.scatter(pca_features[:, 0],
                    pca_features[:, 1],
                    c=clusters_resnet,
                    cmap='tab10')
plt.title("PCA of Fine-Tuned ResNet-50 Features")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(scatter, label="Cluster ID")
plt.show()

def visualize_cluster_examples_resnet(feature_dataset, cluster_ids
    , kmeans_obj, k=3):
    unique_clusters = np.unique(cluster_ids)
    for c in unique_clusters:
        indices = np.where(cluster_ids == c)[0]
        if len(indices) == 0:
            continue
        print(f"\n[ResNet-FineTuned] Cluster {c}: showing up to {k}
            random images")
        chosen = np.random.choice(indices, size=min(k, len(indices)
            ), replace=False)

```

```
    fig, axs = plt.subplots(1, max(1, len(chosen)), figsize
        =(15, 3))
    if len(chosen) == 1:
        axs = [axs]

    fig.suptitle(f"Cluster {c} sample images", fontsize=16)
    for i, idx in enumerate(chosen):
        path, label_idx = feature_dataset.samples[idx]
        img = plt.imread(path)
        axs[i].imshow(img)
        axs[i].axis('off')
    plt.show()

visualize_cluster_examples_resnet(test_dataset, clusters_resnet,
    kmeans_resnet, k=3)
```

Listing 4: PCA + Transfer Learning Code for Q4.

### 5.4 Discussion

- The raw pixel flattening often does not capture discriminative features effectively, leading to less cohesive clusters.
- Fine-tuned ResNet-50 features show significantly better cluster separation in PCA space.
- Transfer learning can improve classification accuracy and yield more meaningful embeddings.

## 6 Conclusion

This report covers four key topics in computer vision:

1. **Region-Growing Segmentation:** Demonstrated a BFS-based algorithm that expands a region from seed points. Different thresholds and multiple seeds can greatly influence the final segmentation outcome.
2. **Diffusion Models (Text-to-Image/Video, Image-to-Image/Video):** Explored the power of modern diffusion approaches for generative tasks using publicly available pipelines from Hugging Face.
3. **Optical Flow for Motion Tracking:** Implemented Lucas–Kanade optical flow to track feature points in a video, highlighting the movement of a robot arm.

4. **PCA and Transfer Learning:** Showed how PCA can visualize high-dimensional data, and how transfer learning (fine-tuning a ResNet-50) improves the feature representations compared to raw-pixel flattening.

Overall, each of these tasks demonstrates essential computer vision and deep learning techniques, ranging from classical segmentation and optical flow to state-of-the-art generative models and representation learning with deep neural networks.