Computer Vision:

Introduction to Computer Vision

Lecturer: **SEK SOCHEAT**
E-mail: ssocheat@puthisastra.edu.kh
Mobile: +855 17 879 967

MSIT, T4 (2023-2024)

Contents

The Concept of Computer Vision

- The concept of Computer Vision revolves around the ability of machines to interpret and understand the visual data from the world, similarly to how humans use their sight.
- Essentially, it's about automating the capabilities of the human visual system to identify, process, and make decisions based on the information gathered from visual inputs.
- The introduction of computer vision as a concept generally wraps these elements into a cohesive overview, setting the stage for more detailed exploration of techniques, applications, and challenges in subsequent parts of a curriculum or discussion.
- This foundational knowledge helps learners and professionals understand not only how computer vision works but also its potential impact on industries and society at large.

Key Aspects

1. Purpose and Goal

2. How It Works

3. Applications

4. Challenges

5. Technological Integration

6. Ethical and Societal Implications

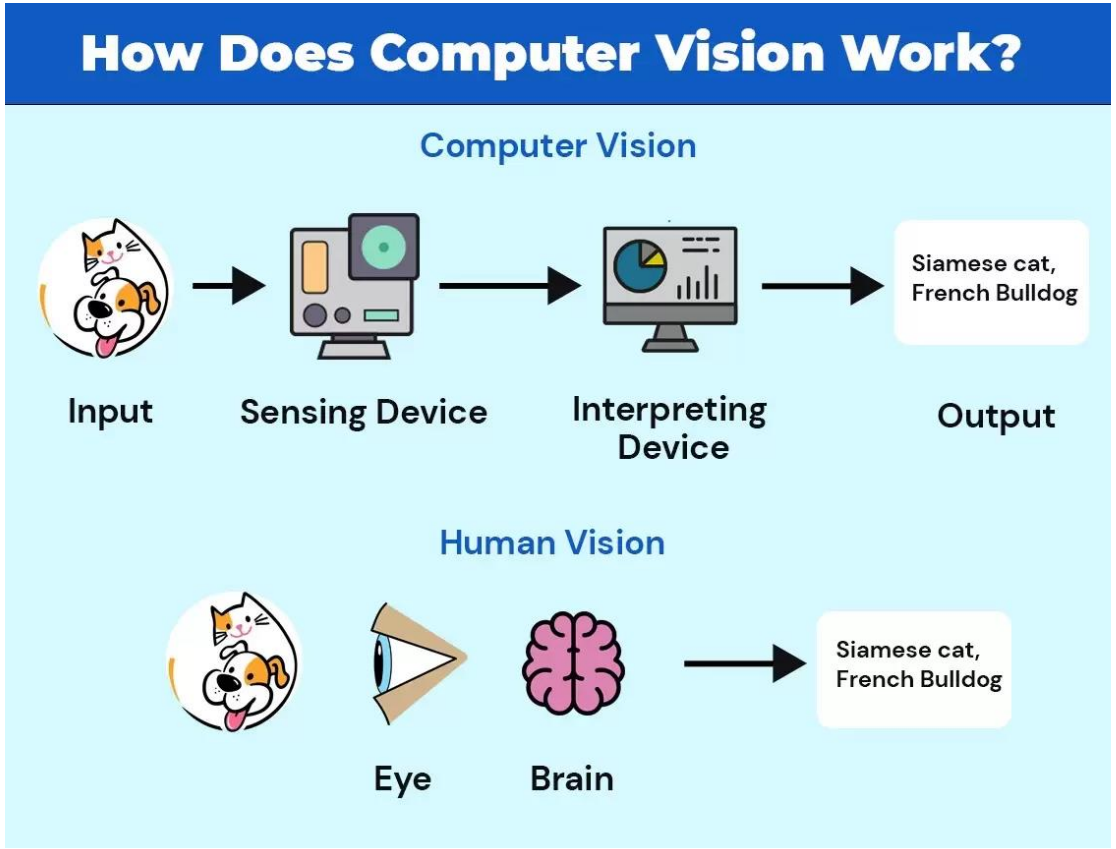
1. Purpose and Goal

- Computer vision aims to replicate and enhance human vision using digital processes.
- The ultimate goal is for computers to analyze and interpret visual data on their own.
- This involves tasks such as identifying objects, classifying them into categories, assessing their properties, and understanding their spatial relationships.

2. How It Works

At its core, computer vision processes involve several steps:

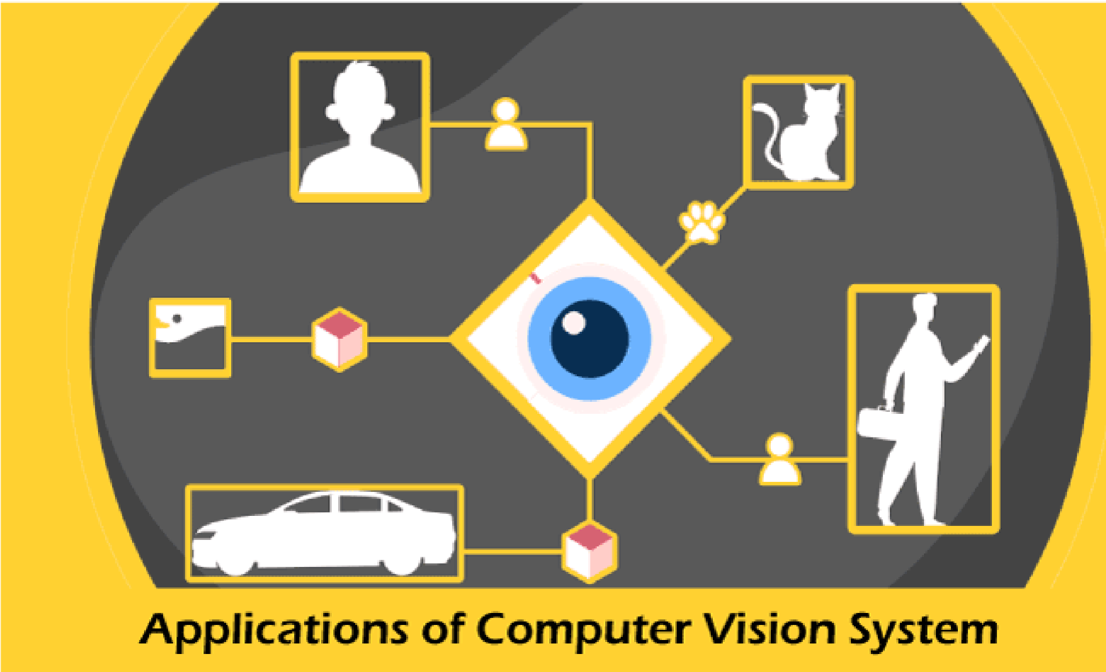
- **Image Acquisition:** Capturing image data through cameras or sensors.
- **Pre-processing:** Refining images to improve quality and enhance features that are important for analysis (e.g., adjusting brightness, filtering noise).
- **Feature Extraction:** Identifying distinctive attributes or features in images that are useful for further analysis, such as edges, textures, or specific shapes.
- **Segmentation and Grouping:** Dividing an image into segments that represent different objects or regions of interest.
- **Recognition and Interpretation:** Applying algorithms to recognize objects or scenes, classify them, and possibly relate them to other known entities.



3. Applications

The versatility of computer vision allows it to be applied in numerous fields:

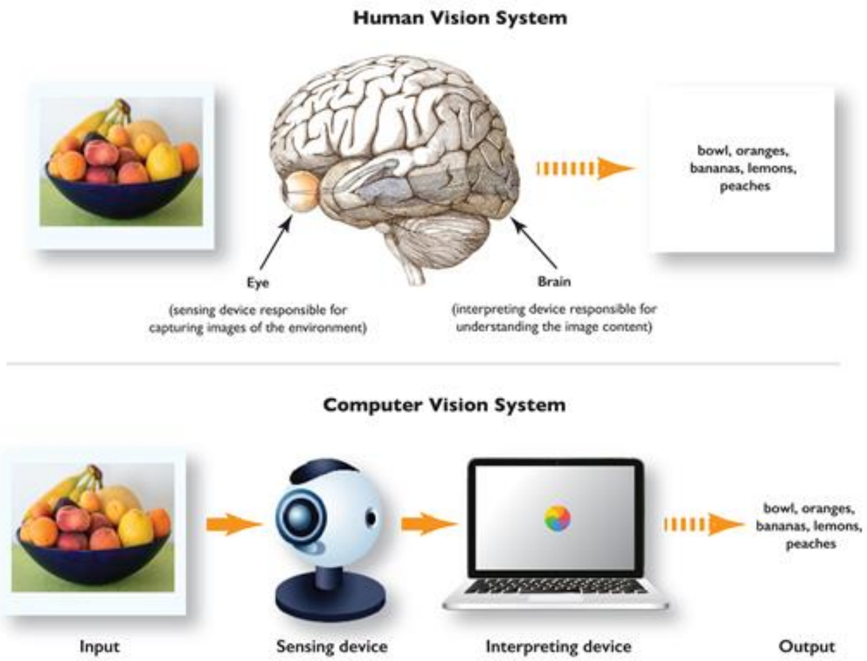
- **Automotive:** Enabling self-driving cars to "see" and navigate their environment.
- **Manufacturing:** Inspecting products for defects at high speeds.
- **Security:** Analyzing video footage in real-time for surveillance purposes.
- **Agriculture:** Monitoring crop health and automating harvesting.
- **Healthcare:** Enhancing diagnostic accuracy through better interpretation of medical imagery.



4. Challenges

Despite its advancements, computer vision faces significant challenges such as:

- **Variability in Visual Data:** Changes in lighting, angles, or obscured views can significantly affect the performance of vision algorithms.
- **Complexity of Natural Scenes:** Real-world environments are highly complex and unpredictable, which makes it difficult to design algorithms that generalize well from one scene to another.
- **Semantic Gap:** The difference between human-level understanding of images and the numerical information that machines perceive from those images presents a fundamental challenge. Bridging this gap—enabling machines not just to see, but to understand the context and nuances of visual data—is a primary goal of ongoing research.

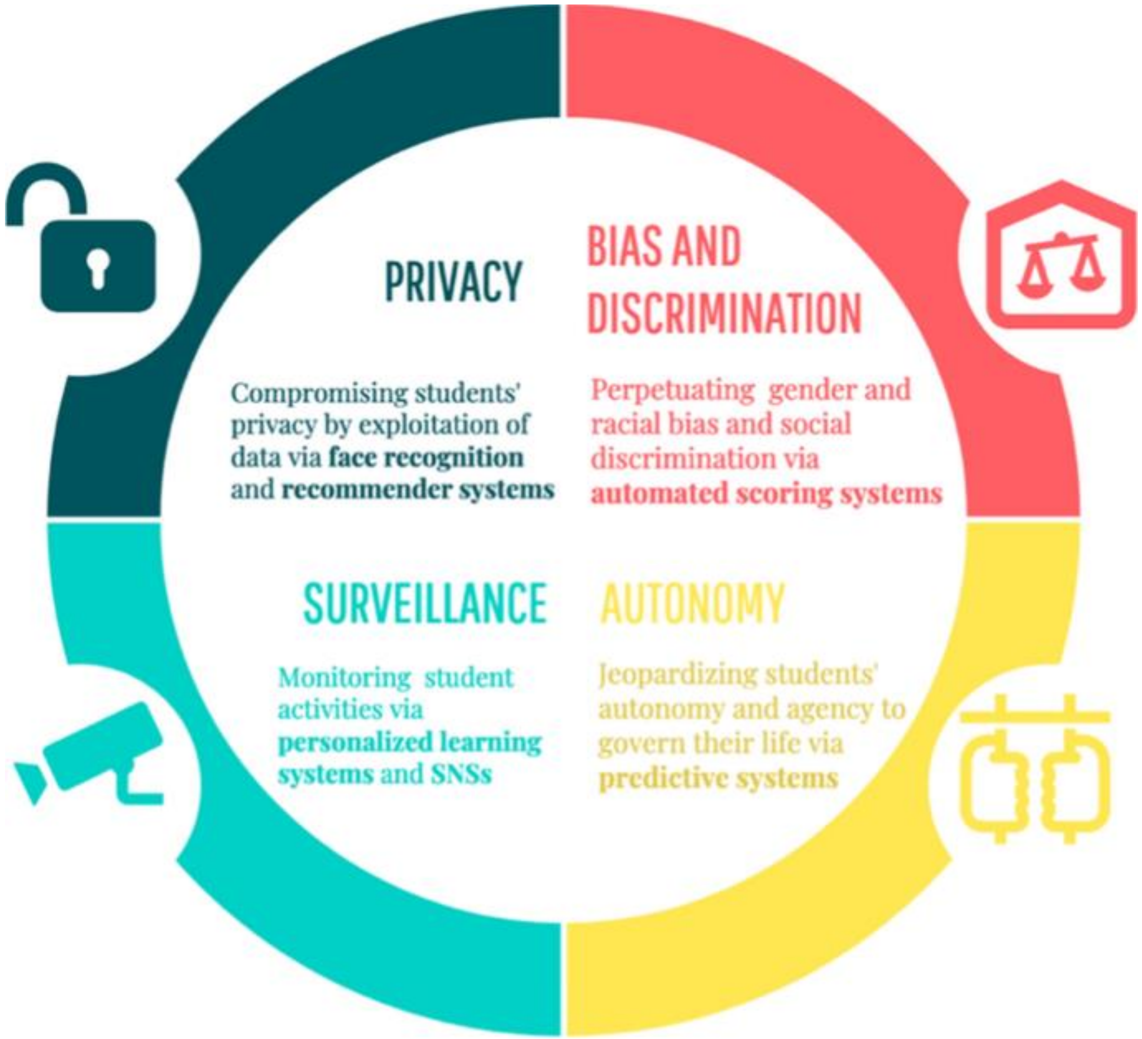


5. Technological Integration

Advances in hardware, such as GPUs and specialized processors, along with better sensors, have propelled the capabilities of computer vision. Software advancements, particularly in machine learning and neural networks, have further refined the accuracy and speed of visual data processing.

6. Ethical and Societal Implications

As computer vision systems become more pervasive, ethical considerations increasingly come to the fore. Issues such as surveillance overreach, privacy invasion, and biases in facial recognition technologies highlight the need for ethical frameworks and regulations in the deployment of computer vision technologies.



Historical Context of Computer Vision

1950s-1970s

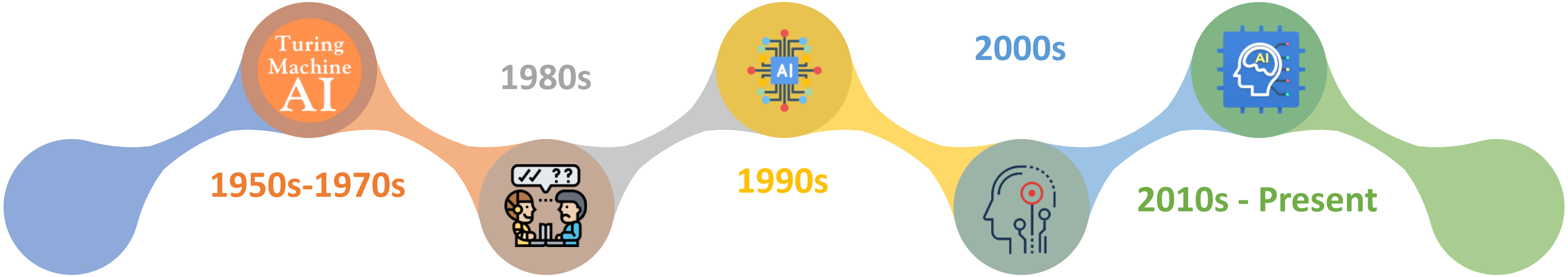
Early explorations in pattern recognition and basic image processing set the stage for future developments. Notable projects like the MIT Summer Vision Project attempted to link cameras to computers for object recognition tasks.

1990s

As computational power increased, computer vision began to see practical and commercial applications, including in facial recognition and industrial automation.

2010s - Present

The deep learning revolution, sparked by breakthroughs such as AlexNet, transformed computer vision, leading to profound advancements in applications ranging from autonomous vehicles to advanced surveillance systems.



1980s

Advances in algorithms improved capabilities in object recognition and motion tracking. The foundational ideas of neural networks were also developed during this period.

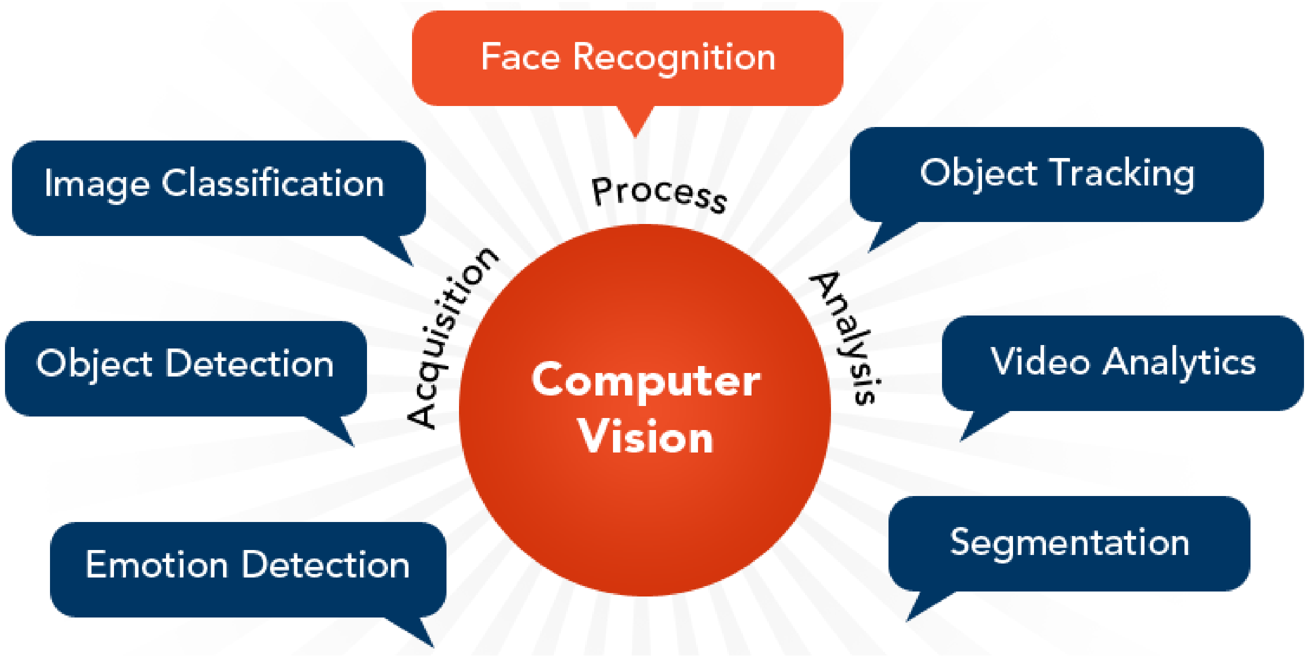
2000s

Machine learning techniques, particularly those involving feature detection like SIFT and SURF, became integrated with computer vision, enhancing its accuracy and reliability.

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



1. Image Acquisition:

- **Sensors:** The process starts with capturing visual information, typically using cameras or sensors that convert light into digital data.
- **Resolution and Depth:** Characteristics like resolution and color depth impact the amount of detail and the range of colors in captured images.

Foundational Concepts of Computer Vision

Image Acquisition

- **Image Acquisition** refers to the process of capturing an image from the real world to create a digital representation that can be processed by a computer. This process involves converting the light captured from a scene into a set of digital pixels.

- **Image Acquisition** is a crucial first step. It involves capturing digital images via cameras or other means, which can then be processed and analyzed using various algorithms.

To enhance image acquisition in Python, especially when working with the OpenCV library, there are several considerations:

- **Camera Setup and Configuration:** The initial setup of the camera is fundamental. Ensuring optimal focus, resolution, and frame rate can significantly enhance the quality of the acquired images.
- **Handling Different Light Conditions:** Good image acquisition needs to handle varying lighting. This includes implementing algorithms that can adjust the camera settings automatically based on the environment, such as automatic exposure compensation.
- **Noise Reduction:** Image noise can obscure details and affect the performance of computer vision algorithms. Implementing noise reduction techniques during or immediately after acquisition can improve image quality.
- **Real-time Processing:** Sometimes, you may need to process images as they are being captured, such as for motion detection or live video analysis. Efficient real-time processing ensures that the system can handle video streams without lag.

Foundational Concepts of Computer Vision

Example: Image Acquisition

```python
import cv2
import matplotlib.pyplot as plt
from datetime import datetime

def adjust_brightness(img, value=30):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    h, s, v = cv2.split(hsv)

    lim = 255 - value

    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))

    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img

def capture_image(frame, description):
    filename = f"images\{datetime.now().strftime('%Y%m%d_%H%M%S')}_{description}.jpg"

    cv2.imwrite(filename, cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))
    print(f"Image saved as {filename}")  # Print the file name to the console

    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    plt.imshow(image)
    plt.title(f'Captured Image: {filename}')
    plt.show()
```

1. adjust_brightness(img, value=30): Adjusts the brightness of an input image by converting it to **HSV** (**H**ue, **S**aturation, **V**alue) format, increasing the Value channel based on the specified value, and then converting it back to **BGR** format. This function returns the brightness-adjusted image.

2. capture_image(frame, description): Saves the provided image frame with a filename that includes a timestamp and a description. It saves the image in **BGR** format, prints the file location, and displays the image using **Matplotlib** with a title that indicates the filename. This function is used for capturing and visually confirming saved images.

Foundational Concepts of Computer Vision

Example: Image Acquisition

```python

cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Camera is not opened.")
else:
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame.")
            break

        frame_adjusted = adjust_brightness(frame, value=30)

        cv2.imshow('Original', frame)
        cv2.imshow('Brightness Adjusted', frame_adjusted)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
        elif key == ord('c'):
            capture_image(frame, 'original')
        elif key == ord('b'):
            capture_image(frame_adjusted, 'brightness_adjusted')

    cap.release()
    cv2.destroyAllWindows()
```

1. Opens a video camera stream using **OpenCV**.

2. Captures and processes video frames continuously in a loop:

- Adjusts the brightness of each frame.
- Displays both the original and adjusted frames.

3. Handles user inputs to:

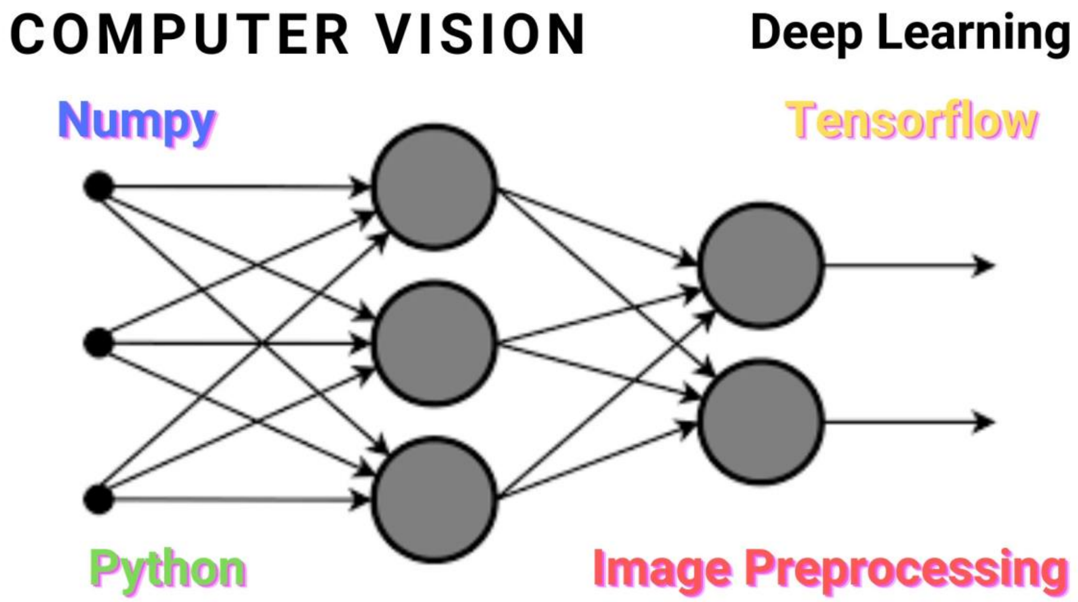
- Save the current frame ('**c**' for original, '**b**' for adjusted).
- Exit the loop and close the application ('**q**').

4. Cleans up by releasing the camera and closing all windows upon exiting.

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



2. Preprocessing:

- **Noise Reduction:** Techniques like filtering are used to reduce noise and improve image quality.

- **Normalization:** Adjusting the image data to bring it into a consistent format or range, which is crucial for further processing.

Foundational Concepts of Computer Vision

Preprocessing

- **Preprocessing** refers to the initial steps or operations applied to data to prepare it for further analysis or processing.

- In the context of image processing and computer vision, preprocessing involves transforming raw image data to enhance its quality or extract meaningful features, making it more suitable for tasks such as object detection, recognition, or analysis.

Preprocessing in image processing involves several key techniques, each aimed at enhancing the usability of images for further analysis:

1. Noise Reduction: Removes unwanted variations from the image to enhance clarity.

2. Contrast Enhancement: Improves the distinction between the brightest and darkest parts of the image.

3. Normalization: Scales pixel intensity values to a standard range, often to facilitate uniform processing.

4. Scaling and Resizing: Adjusts the image dimensions to meet the requirements of specific applications or algorithms.

5. Color Space Conversion: Transforms the image into different color formats that may be more useful for certain tasks.

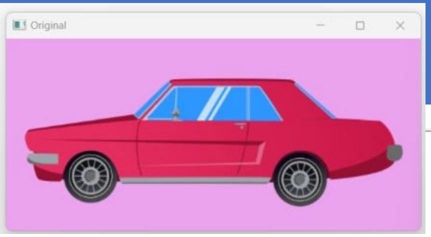
6. Edge Detection: Identifies and highlights the boundaries of objects within the image, useful for segmentation.

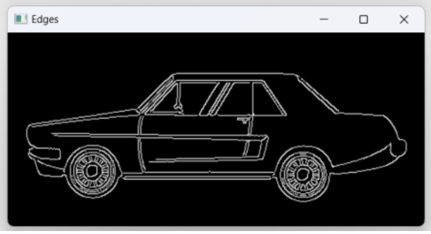
7. Geometric Transformations: Modifies the image geometry through rotation, translation, or scaling to correct alignment or augment data.

Foundational Concepts of Computer Vision

Example: Preprocessing

```python
import cv2

image = cv2.imread('images/small_car.jpg')

if image is None:
    print("Error: Image not found.")
    exit()

image_blurred = cv2.GaussianBlur(image, (5, 5), 0)

image_gray = cv2.cvtColor(image_blurred, cv2.COLOR_BGR2GRAY)

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
image_clahe = clahe.apply(image_gray)

if len(image.shape) == 3:
    image_color_corrected = cv2.cvtColor(image_clahe, cv2.COLOR_GRAY2BGR)
else:
    image_color_corrected = image_clahe

edges = cv2.Canny(image_color_corrected, 50, 150)

cv2.imshow('Original', image)
cv2.imshow('Enhanced', image_color_corrected)
cv2.imshow('Edges', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





1. Load Image: The script reads an image from a specified path.

2. Check Image Availability: It checks if the image was successfully loaded; if not, it prints an error message and exits.

3. Apply Gaussian Blur: Blurs the image to reduce noise using a Gaussian kernel of size (5,5).

4. Convert to Grayscale: Converts the blurred image to grayscale for further processing.

5. Apply CLAHE: Enhances the contrast of the grayscale image using Contrast Limited Adaptive Histogram Equalization (CLAHE).

6. Color Correction: Converts the CLAHE output back to BGR color space if the original image was in color.

7. Edge Detection: Uses the Canny algorithm to detect edges in the contrast-enhanced image.

8. Display Images: Shows the original, enhanced, and edge-detected versions of the image.

9. Wait and Cleanup: Waits for a key press to close the windows and properly shuts down the display windows.

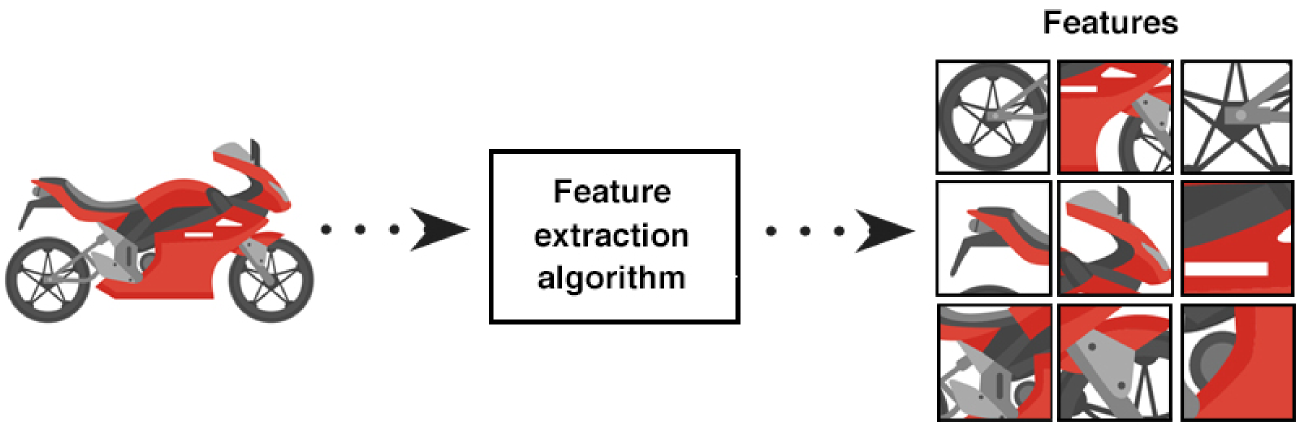
Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:

3. Feature Extraction:

- **Edges and Contours:** Detecting edges and outlines of objects within an image helps in distinguishing objects from the background and from each other.
- **Textures:** Analyzing surface textures to recognize patterns or classify materials.
- **Color:** Using color histograms or color spaces (like RGB, HSV) to identify unique properties of objects.



Foundational Concepts of Computer Vision

Feature Extraction

- Feature Extraction is a process in data analysis and machine learning where raw data is transformed or reduced to a more manageable group of variables, known as features.

- These features capture essential information from the raw data, which can be used for further processing like classification or regression.

Key Points in Feature Extraction:

1. Dimensionality Reduction: Simplifies large data sets by reducing the number of variables, enhancing the efficiency of data processing.

2. Information Preservation: Ensures that the most critical information is retained despite reducing the number of variables.

3. Noise Reduction: Eliminates irrelevant data, improving the accuracy and quality of predictions.

4. Improved Learning Efficiency: Streamlines learning in machine learning models by focusing on essential features, enhancing both speed and performance.

5. Facilitation of Visualization: Makes it easier to visualize and analyze data by reducing its complexity, which is especially useful during exploratory analysis.

6. Specific Techniques: Includes methods like Principal Component Analysis (PCA), indicator variables for categorical data, and domain-specific techniques like texture and color extraction in images or cepstral coefficients in audio processing.

Foundational Concepts of Computer Vision

Example: Feature Extraction

```python
import cv2
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

image = cv2.imread('images/group_people1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
height, width = gray.shape
step_size = height // 5

samples = [gray[x:x+step_size, :].flatten() for x in range(0, height, step_size) if x+step_size <= height]
samples = np.array(samples)

if samples.shape[0] > 1:
    pca = PCA(n_components=min(5, samples.shape[0], samples.shape[1]))
    features = pca.fit_transform(samples)
    components = pca.components_
    fig, axes = plt.subplots(pca.n_components_, 1, figsize=(5, 5 * pca.n_components_))
    for i, ax in enumerate(axes.flat):
        component_image = components[i].reshape(step_size, width)
        ax.imshow(component_image, cmap='gray')
        ax.set_title(f"Component {i+1}")
        ax.axis('off')

    plt.tight_layout()
    plt.show()

    print("PCA features shape:", features.shape)
else:
    print("Not enough samples for PCA.")
```

1. Import Libraries: Brings in required modules for image handling, numerical operations, PCA, and plotting.

2. Load and Convert Image: Loads an image and converts it from color (BGR) to grayscale.

3. Define Image Sections: Determines the image's height and width, and calculates sections by dividing the image into five equal parts vertically.

4. Create Samples: Extracts flattened pixel data from each section to create samples for analysis.

5. Prepare Data Array: Transforms the list of samples into a structured NumPy array.

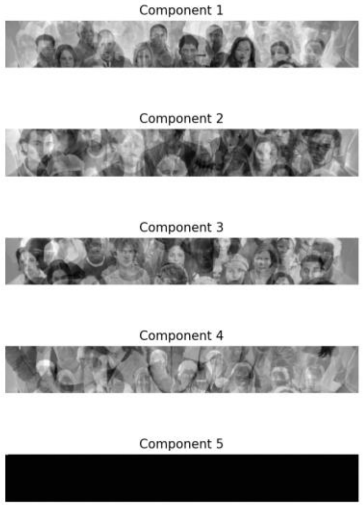
6. Check Sample Size and Apply PCA: Ensures there are enough samples for PCA and applies it to reduce dimensionality, focusing on capturing the most variance.

7. Visualize Results: Displays each principal component as an image, showing key features detected by PCA.

8. Adjust Layout and Show Plots: Arranges plots neatly and shows them, ensuring clarity in visual representation.

9. Output PCA Details: Prints the shape of the transformed data to indicate the new dimensionality.

10. Handle Insufficient Data: Provides feedback if there aren't enough samples to perform PCA effectively.



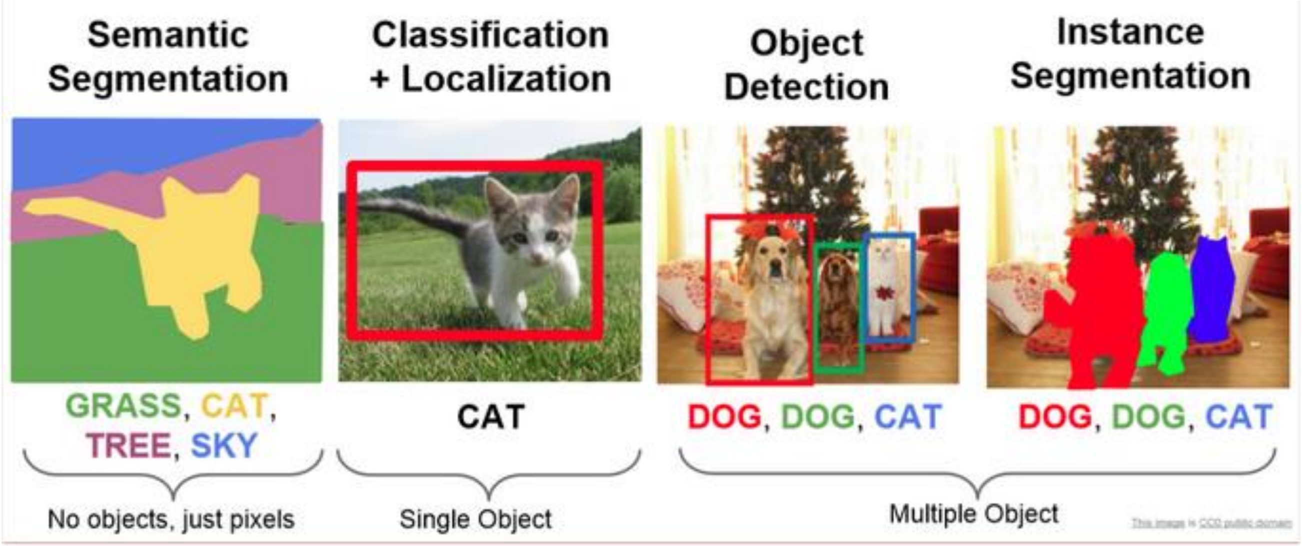
Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:

4. Segmentation:

- **Thresholding:** Separating objects from the background based on color or light intensity.
- **Clustering:** Grouping pixels or features in an image into cohesive regions.



Foundational Concepts of Computer Vision

Segmentation

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def main():
    # Load the image
    image = cv2.imread('angkor_wat.jpg')
    if image is None:
        print("Error: Could not read the image.")
        return

    # Convert the image to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Reshape the image to a 2D array of pixels
    pixel_values = image.reshape((-1, 3))
    # Convert to float
    pixel_values = np.float32(pixel_values)

    # Define criteria and apply K-means clustering
    # Criteria: (type, max_iter, epsilon)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.
                TERM_CRITERIA_MAX_ITER, 100, 0.2)

    # Number of clusters (K)
    k = 3
    _, labels, centers = cv2.kmeans(pixel_values,
                                    k, None, criteria, 10,
                                    cv2.KMEANS_RANDOM_CENTERS)

    # Convert centers to 8-bit values
    centers = np.uint8(centers)
    # Map the labels to the center colors
    segmented_image = centers[labels.flatten()]
    # Reshape to the original image shape
    segmented_image = segmented_image.reshape(image.shape)

    # Display the original and segmented images
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(image)
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title('Segmented Image with K-means')
    plt.imshow(segmented_image)
    plt.axis('off')

    plt.show()

if __name__ == "__main__":
    main()
```

Foundational Concepts of Computer Vision

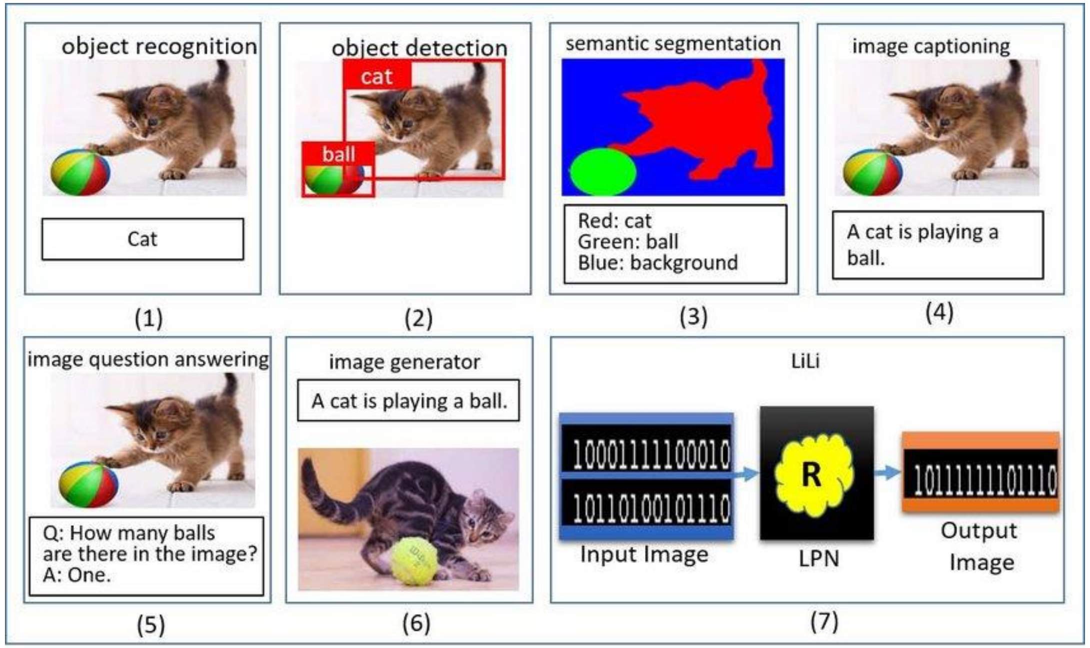
Explanation:

- **Loading the image:** The image is loaded using **cv2.imread** and converted to **RGB** using **cv2.cvtColor**.
- **Reshaping the image:** The image is reshaped into a **2D** array where each pixel is a point in a **3D** space (R, G, B).
- **K-means clustering:** The K-means algorithm clusters the pixel values into k clusters, where **k** is the number of segments we want.
- **Mapping the labels:** Each pixel is assigned the color of its corresponding cluster center.
- **Displaying the images:** The original and segmented images are displayed using **matplotlib**.

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



5. Object Detection and Recognition:

- **Template Matching:** Comparing segments of an image to predefined templates to find matches.

- **Machine Learning Models:** Training models on large datasets to recognize objects, using techniques ranging from simple linear classifiers to complex neural networks.

Foundational Concepts of Computer Vision

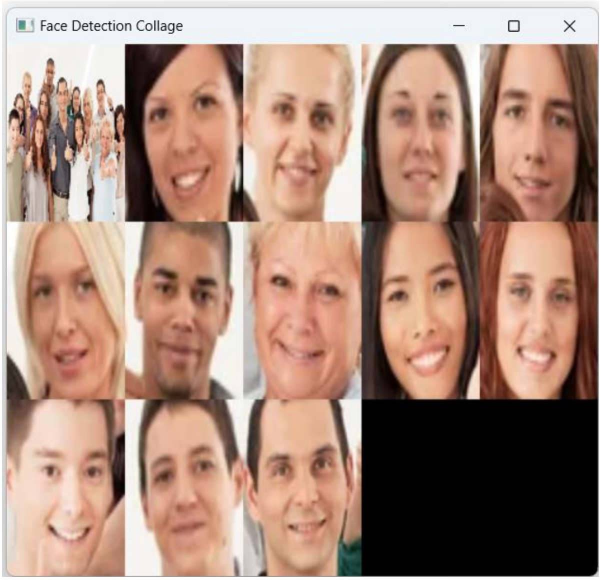
Object Detection and Recognition

```python
import cv2
import numpy as np
import os

def load_haar_cascade():
    # Check if the Haar Cascade file exists
    cascade_path = cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
    if not os.path.exists(cascade_path):
        raise FileNotFoundError("Haar Cascade file not found.")
    face_cascade = cv2.CascadeClassifier(cascade_path)
    return face_cascade

def detect_faces(img, face_cascade):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
                                          minNeighbors=5, minSize=(30, 30))
    return faces

def extract_faces(img, faces, new_width=100, new_height=150):
    face_images = []
    for (x, y, w, h) in faces:
        face = img[y:y+h, x:x+w]
        resized_face = cv2.resize(face, (new_width, new_height))
        face_images.append(resized_face)
    return face_images

```

Foundational Concepts of Computer Vision

Object Detection and Recognition

```
def create_face_collage(face_images, original_image, cols=5,
                        new_width=100, new_height=150):
    rows = (len(face_images) + 1 + cols - 1) // cols
    collage = np.zeros((rows * new_height, cols * new_width, 3), dtype=np.uint8)

    # Resize the original image to fit in the first block
    resized_original = cv2.resize(original_image, (new_width, new_height))
    collage[0:new_height, 0:new_width] = resized_original

    for idx, face in enumerate(face_images):
        row = (idx + 1) // cols
        col = (idx + 1) % cols
        collage[row * new_height:(row + 1) * new_height, col * new_width:(col + 1) * new_width] = face

    return collage

def image_detection(img_path):
    face_cascade = load_haar_cascade()
    img = cv2.imread(img_path)
    faces = detect_faces(img, face_cascade)
    face_images = extract_faces(img, faces)

    if face_images:
        collage = create_face_collage(face_images, img)
        cv2.imshow("Face Detection Collage", collage)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("No faces detected.")

if __name__ == "__main__":
    image_path = "photos.jpg"
    image_detection(image_path)
```



Foundational Concepts of Computer Vision

Explanation:

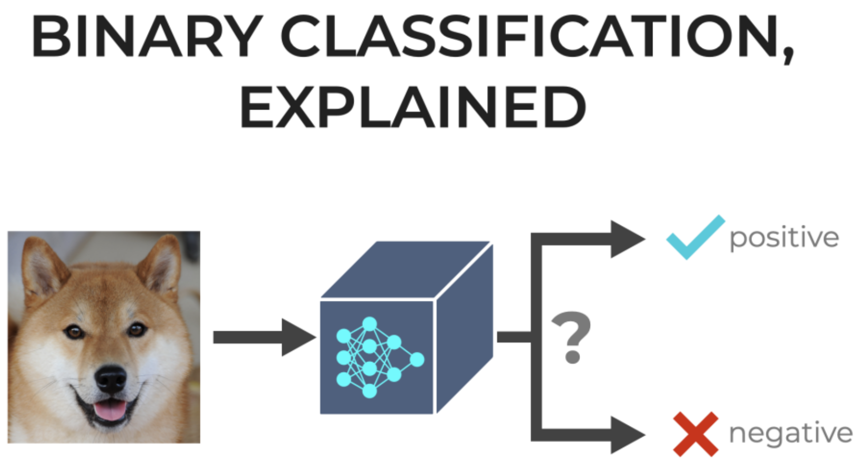
The **image_detection** function orchestrates the process of face detection in an image by:

- Loading the Haar Cascade classifier.
- Reading the image from the provided file path.
- Detecting faces using the classifier.
- Extracting and resizing the detected faces.
- Checking if any faces were detected.
- Creating and displaying a collage of the original image and the detected faces.
- Printing a message if no faces were detected.

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



6. Classification:

- **Supervised Learning:** Using labeled data to teach models to classify images into categories.
- **Unsupervised Learning:** Discovering patterns and classifying data without pre-labeled examples.

Foundational Concepts of Computer Vision

Explanation of the Cat and Dog Classifier Program

1. Dataset Preparation:

- Downloads and extracts the Cats vs. Dogs dataset.
- Organizes the data into training and validation directories.

2. Data Augmentation and Normalization:

- Uses **ImageDataGenerator** to augment and normalize the images for better model generalization.

3. Data Generators:

- Creates **train_generator** and **validation_generator** to read images in batches and apply augmentations.

4. Building the Model:

- Constructs a Convolutional Neural Network (**CNN**) using **Keras** Sequential API with layers including convolution, max pooling, flattening, and dense layers.

5. Compiling the Model:

- Compiles the model with binary cross-entropy loss, Adam optimizer, and accuracy as a metric.

6. Training the Model:

- Trains the model for 15 epochs using the training and validation datasets, repeating the dataset to avoid running out of data.

7. Visualizing Training Results:

- Plots training and validation accuracy and loss over epochs.

8. Predicting New Images:

- Defines functions to preprocess an image and predict whether it is a cat or dog.
- Displays the image with the predicted class label.

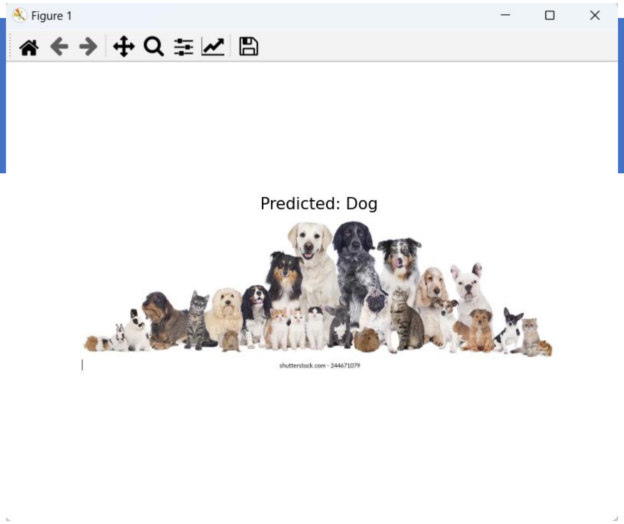
This program provides a comprehensive approach to training a CNN for binary classification of cats and dogs, including data handling, model building, training, evaluation, and prediction.

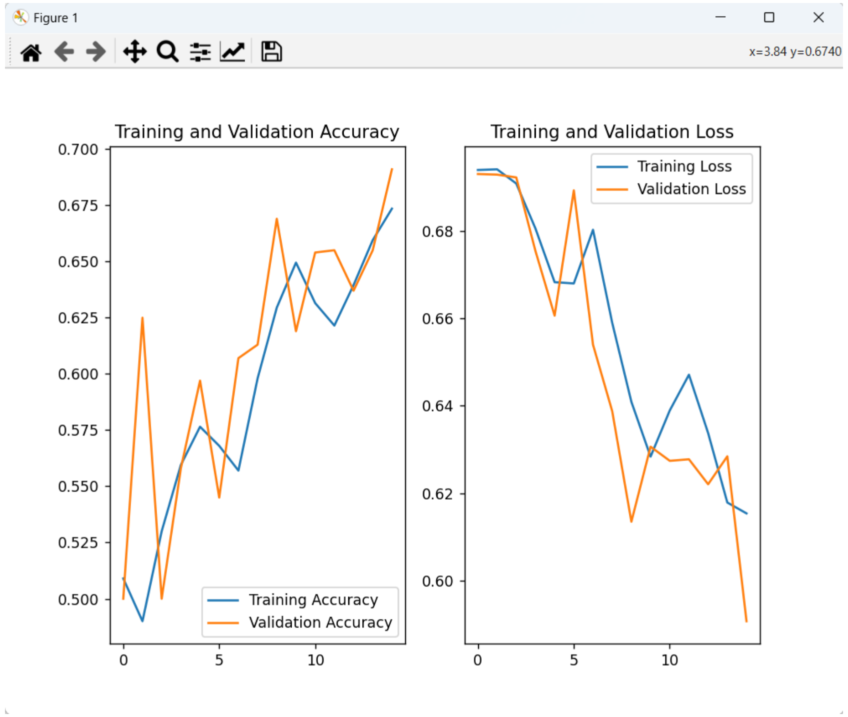
Foundational Concepts of Computer Vision

6. Classification: main_classification.py

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
import os
from PIL import Image

# Download the dataset
url = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
zip_file = tf.keras.utils.get_file('cats_and_dogs_filtered.zip', origin=url, extract=True)
base_dir = os.path.join(os.path.dirname(zip_file), 'cats_and_dogs_filtered')

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

# Data augmentation and normalization
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

pip install tensorflow matplotlib numpy pillow

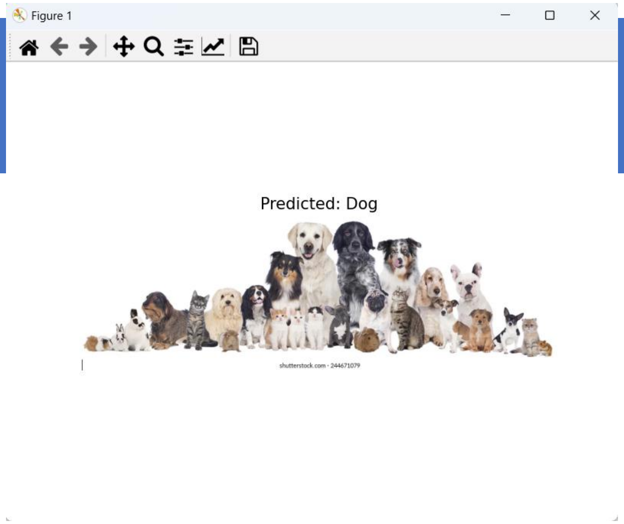


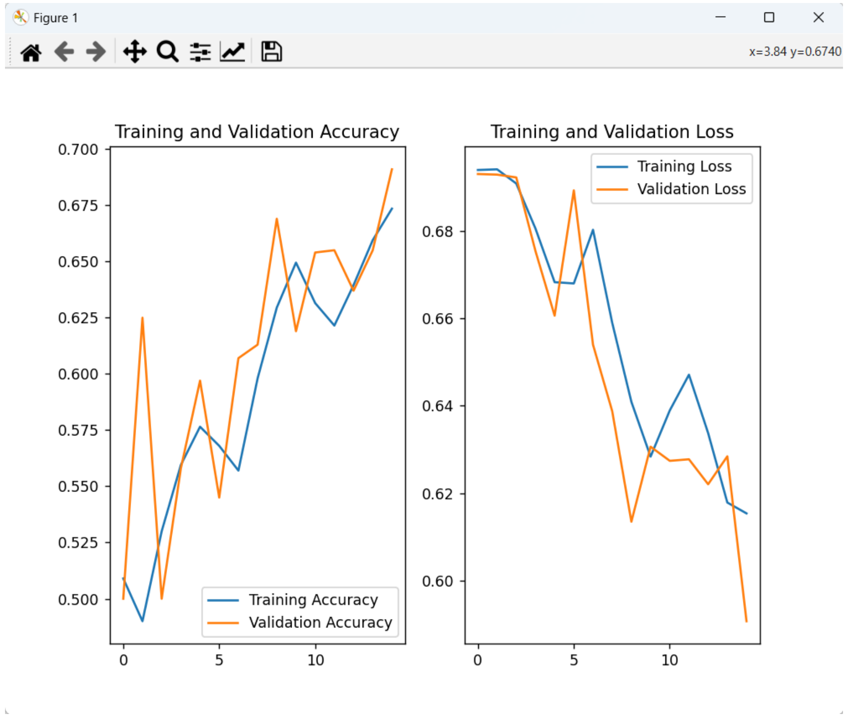


Foundational Concepts of Computer Vision

6. Classification: main_classification.py

```python
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

# Add the .repeat() method
train_dataset = tf.data.Dataset.from_generator(
    lambda: train_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes=([None, 150, 150, 3], [None])
).repeat()

validation_dataset = tf.data.Dataset.from_generator(
    lambda: validation_generator,
    output_types=(tf.float32, tf.float32),
    output_shapes=([None, 150, 150, 3], [None])
).repeat()

```

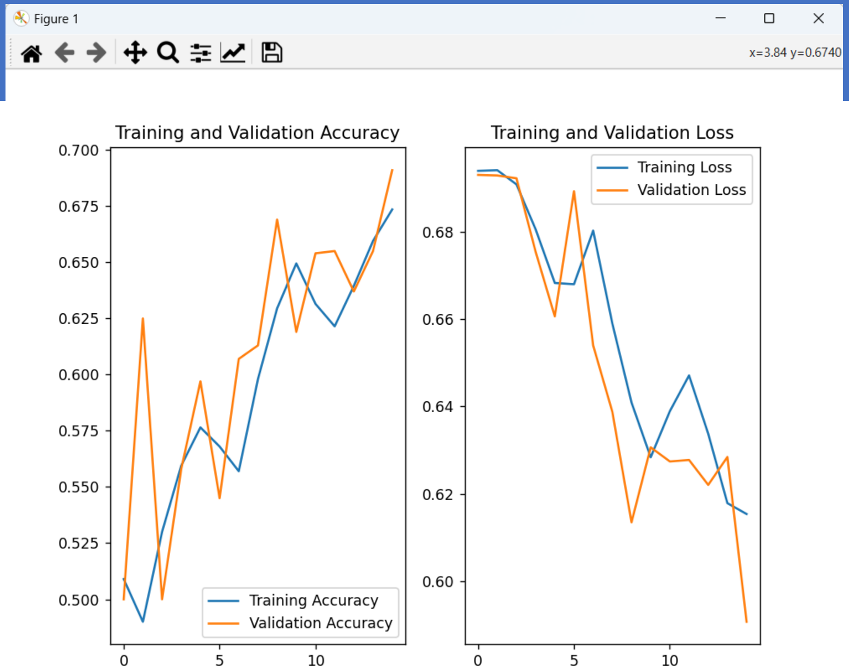


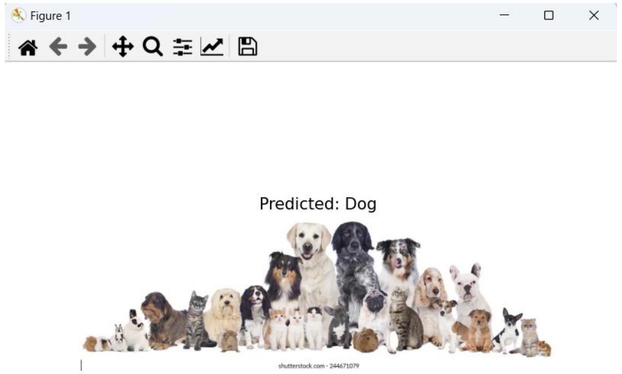


Foundational Concepts of Computer Vision

6. Classification: main_classification.py

```python
# Build the model
model = models.Sequential([
    layers.Input(shape=(150, 150, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

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

# Calculate the number of steps per epoch
num_train_samples = sum([len(files) for r, d, files in os.walk(train_dir)])
num_validation_samples = sum([len(files) for r, d, files in os.walk(validation_dir)])
steps_per_epoch = num_train_samples // 20
validation_steps = num_validation_samples // 20

# Train the model
history = model.fit(
    train_dataset,
    steps_per_epoch=steps_per_epoch,
    epochs=15,
    validation_data=validation_dataset,
    validation_steps=validation_steps
)
```

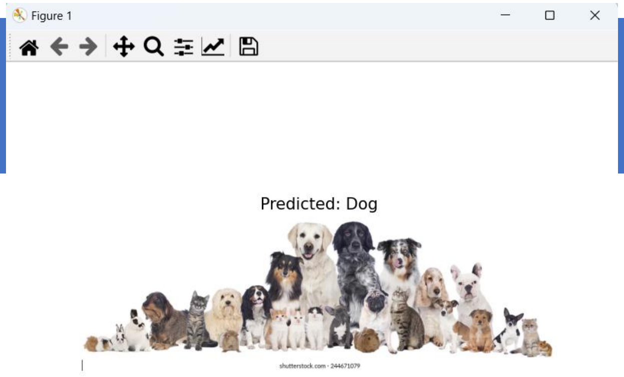


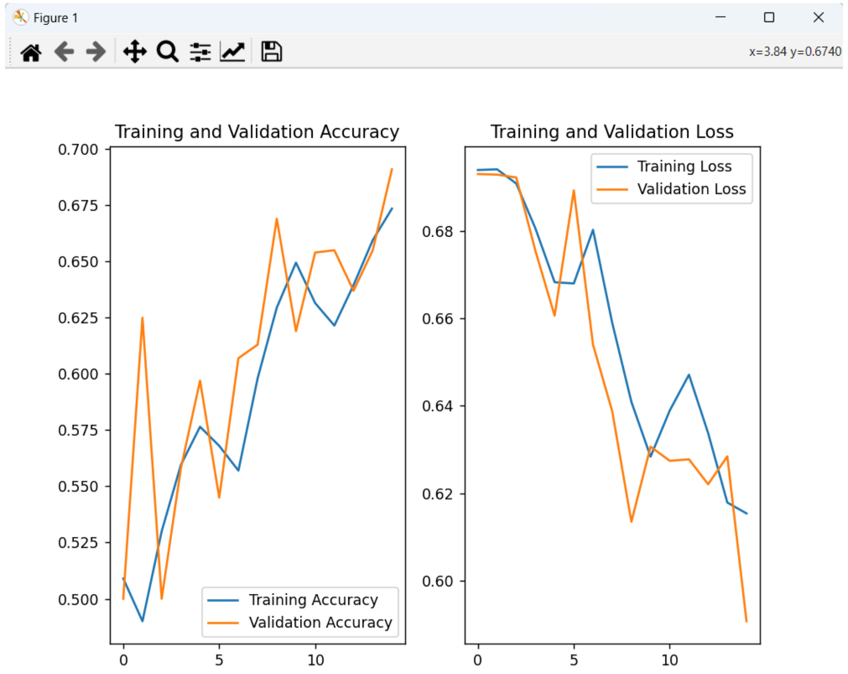


Foundational Concepts of Computer Vision

6. Classification: main_classification.py

```python
# Evaluate the model
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(15)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

def load_and_preprocess_image(img_path):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=(150, 150))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0
    return img_array

def predict_image(img_path):
    img = load_and_preprocess_image(img_path)
    prediction = model.predict(img)
    return 'Dog' if prediction[0] > 0.5 else 'Cat'

# Example usage
img_path = 'cat_dog.jpg'
predicted_class = predict_image(img_path)
print(f'The predicted class is: {predicted_class}')
```



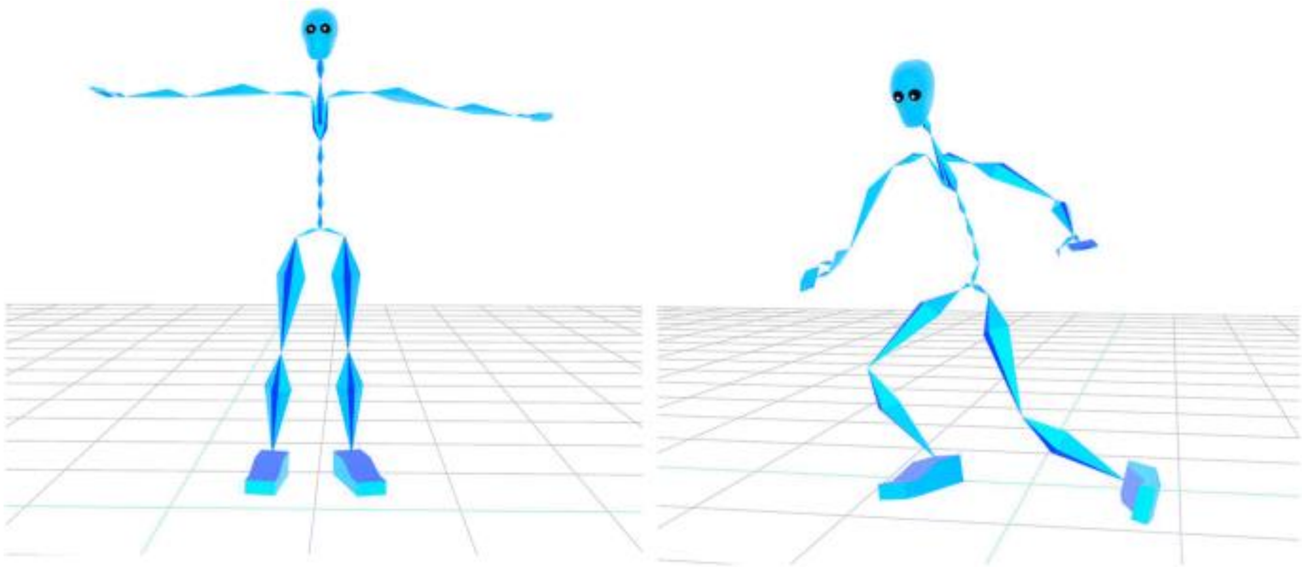



```python

img = tf.keras.preprocessing.image.load_img(img_path)
plt.imshow(img)
plt.title(f'Predicted: {predicted_class}')
plt.axis('off')
plt.show()
```

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



7. Tracking and Motion Analysis:

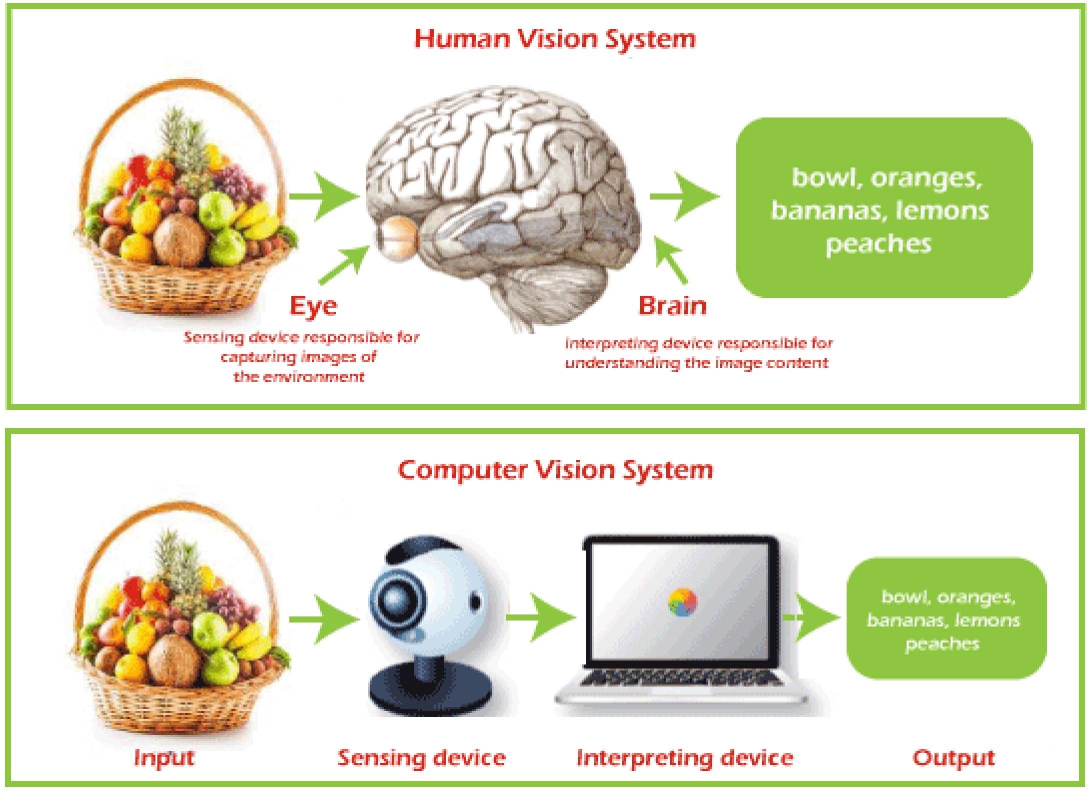
- **Optical Flow:** Calculating the motion between two image frames that are taken at different times to track movement.

- **Background Subtraction:** Identifying moving objects from the part of a video that changes over time.

Foundational Concepts of Computer Vision

Understanding the foundational concepts of computer vision is crucial for grasping how it allows computers to interpret and interact with the visual world.

Here are some of the core concepts that underpin the field:



8. High-Level Vision Tasks:

- **Scene Reconstruction:** Building a three-dimensional scene from a set of two-dimensional images.
- **Image Registration:** Aligning multiple images into a single integrated view.
- **Scene Understanding :** Interpreting complex scenes to understand the relationship and context of objects within them.

Thanks for attention,

Any Question?