1. Computer Vision Tasks

Computer vision is a field of AI that enables machines to interpret and process visual data. The key tasks in computer vision include:

1.1 Image Classification

- **Definition:** Assigning a label to an image from a predefined set of categories.
- **Example:** Classifying images into "cat" or "dog."

Mathematical Explanation

Given an input image XX, a classification model outputs a probability distribution over CC classes:

P(y|X)=f(X,θ)P(y | X) = f(X, \theta)

where ff is a deep neural network with parameters θ\theta, and yy is the predicted class.

The model is trained using **Cross-Entropy Loss**:

L=−∑i=1Cyilog(y^i)L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)

where yiy_i is the true label and y^i\hat{y}_i is the predicted probability.

Python Code for Image Classification using CNN

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers


# Load dataset (CIFAR-10)

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize images


# Define CNN model

model = keras.Sequential([

    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),

    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),

    layers.MaxPooling2D((2,2)),

    layers.Flatten(),
```

```
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes in CIFAR-10
])

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

# Train model
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
```

1.2 Object Detection

- **Definition:** Identifying and locating objects in an image by drawing bounding boxes around them.
- **Example:** Detecting multiple objects like "person" and "car" in an image.

Mathematical Explanation

Object detection involves:

- **Classification** (which object is present)
- **Localization** (where the object is)

A neural network predicts:

(x,y,w,h,c1,c2,...,cC)(x, y, w, h, c_1, c_2, ..., c_C)

where:

- (x,y)(x, y) is the center of the bounding box.
- (w,h)(w, h) is the width and height.
- cic_i are the class probabilities.

The **Intersection over Union (IoU)** metric evaluates detection accuracy:

IoU=Area of OverlapArea of UnionIoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}

Python Code for Object Detection using YOLO

```
from ultralytics import YOLO
```

```
# Load YOLOv8 pre-trained model
model = YOLO("yolov8n.pt")  # Download model automatically

# Run inference on an image
results = model("image.jpg")

# Display results
results.show()
```

1.3 Image Segmentation

- **Definition:** Assigning each pixel of an image to a specific class.
- **Example:** Segmenting "sky," "road," and "car" in a driving scene.

Mathematical Explanation

Segmentation models output a probability map PP of shape (H,W,C)(H, W, C), where each pixel gets a class label:

y^i,j=argmaxcPi,j,c\hat{y}_{i,j} = \arg\max_{c} P_{i,j,c}

Loss function: **Dice Coefficient Loss**

L=1−2∑(P·Y)∑P+∑YL = 1 - \frac{2 \sum (P \cdot Y)}{\sum P + \sum Y}

Python Code for Image Segmentation using U-Net

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D, concatenate

# Define U-Net architecture
def unet_model():
    inputs = keras.Input(shape=(128, 128, 3))
    conv1 = Conv2D(64, (3, 3), activation="relu", padding="same")(inputs)
    pool1 = MaxPooling2D((2, 2))(conv1)
```

```
    conv2 = Conv2D(128, (3, 3), activation="relu", padding="same")(pool1)
    up1 = UpSampling2D((2, 2))(conv2)

    outputs = Conv2D(1, (1, 1), activation="sigmoid")(up1)
    return keras.Model(inputs, outputs)

model = unet_model()
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

---

2. Image Data Handling

Before training, images must be **loaded, resized, and normalized**.

2.1 Loading and Resizing Images

```
import cv2
import numpy as np

# Load image
image = cv2.imread("image.jpg")

# Resize to 224x224
resized_image = cv2.resize(image, (224, 224))
```

2.2 Normalizing Images

```
# Convert to float and normalize to range [0,1]
normalized_image = resized_image / 255.0
```

---

3. Data Augmentation

Improves generalization by applying transformations like rotation, flipping, cropping, and color jittering.

3.1 Rotation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(rotation_range=30)
```

3.2 Flipping

```
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
```

3.3 Cropping

```
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1)
```

3.4 Color Jittering

```
datagen = ImageDataGenerator(brightness_range=[0.5, 1.5])
```

3.5 Full Data Augmentation Example

```
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    brightness_range=[0.5, 1.5]
)

# Apply to an image
image = np.expand_dims(image, 0)  # Expand dimensions for batch
augmented_image = datagen.flow(image, batch_size=1)
```

Conclusion

- **Image Classification** assigns a single label to an image.
- **Object Detection** localizes multiple objects using bounding boxes.
- **Image Segmentation** assigns each pixel a class.
- **Image Data Handling** includes loading, resizing, and normalizing.
- **Data Augmentation** improves generalization.