🎓 Foundations of Computer Vision

🎯 Learning Objectives | Outcomes

- What is Computer Vision?
- Understand Basic Concepts in CV.
- Use cases in Healthcare, Security, Entertainment, etc.
- Understand Digital Image Formats (RGB vs Grayscale vs Binary Images).
- Digital Image Structures (Pixel values, Grids, Color Channels, Color Spaces, Image Shapes).
- Load, Display, and save Images using OpenCV.
- Perform Image Manipulations (Resizing, Cropping, Rotation) using OpenCV.
- Apply real-time Filters: Blurring, Edge Detection and etc.
- Build an Interactive Filter app in Python.
- Apply basic Statistics and Visualizations Data Plots & Histograms from Visual Datasets.
- Grasp the impact of preprocessing for ML/DL pipelines.
- Build simple ML models to Classify image data
- Construct and Train beginner-level CNNs using Python frameworks
- Translate theory into working projects using Python, OpenCV, PyTorch, and more.

1️⃣ Introduction to Computer Vision & Image Types

Topics Covered:

- What is Computer Vision? Scope and applications.
- Image formats: RGB vs. Grayscale vs. Binary, Image Channels and Pixel Intensity.
- Image Histograms, Formats (JPG, PNG, BMP), Resolution Concepts.

Tools Used: Python, OpenCV, Pillow

Visualization: Pixel matrix Visualizer, Histogram Plots

🔍 What is Computer Vision?

Definition:
Computer Vision is a field of Artificial Intelligence that empowers machines to interpret images and videos, enables computers to understand and interpret visual information from the world (Images and Videos), Simulating Human Vision.

Examples and Definitions of real-world Applications:

- Face Recognition (like unlocking phones)
- Medical Imaging Analysis (detecting tumors)
- Self-Driving | Autonomous Driving Cars (lane and object detection)

🖼️ Image Types

❖ Color Channels (RGB) vs. Grayscale Images
❖ Pixel Matrix Representation (Image as Matrices of Pixels)

RGB Images:

- Made of 3 channels: Red, Green, Blue
- Each pixel has 3 values (e.g. `[255, 0, 0]` for red)

Grayscale Images:

- Only 1 channel, where each pixel holds brightness
- Values range from `0` (black) to `255` (white)

2 Image Manipulation with OpenCV

Tutorial Steps with Code Sample Preview (Python):

- Read, Load and display images in OpenCV (cv2.imread, cv2.imshow)
- Convert Images to Grayscale
- Color Space Conversions (RGB ↔ HSV, Grayscale)
- Inspect Pixel Values and Dimensions
- Resizing, Cropping, Rotating Images (using Transformation Matrix)
- Applying Filters: Blur, Canny Edge Detection (cv2.GaussianBlur, cv2.Canny)
- Drawing Shapes, Text Overlays
- Understand basic Image Transformations and Visual Effects.
- **Hands-on a Mini Project:**
  Build a **Simple Photo Filter GUI App***:*
  Design a Python app that loads an image and lets users apply filters (grayscale, blur, edge detection) via button clicks using **tkinter + OpenCV**.

  - UI with sliders for filter intensity (e.g., brightness, blur, contrast)
  - Use OpenCV to apply effects in real-time and (filter selection)
  - Real-time display using OpenCV **imshow** and callbacks
  - Save filtered images to disk

Tools & Setup

- **Language**: Python
- **Dependencies (Libraries)**:

  From Bash (Terminal, CMD or Shell) run: pip install **opencv-python matplotlib numpy**

- **Environment**: Jupyter Notebook or Google Colab (recommended for ease of use)

Load an Image, Printing Shape & Pixel Dept in a Point, Converting RGB to Grayscale, Displaying the Images.

```
import cv2 # Import OpenCV Library

rgb_img = cv2.imread('photo.jpg')  # Loads in BGR Image by default
print(rgb_img.shape) # Show image dimensions
print(rgb_img[100, 150]) # Print Pixel intensity at (100, 150)
gray_img = cv2.cvtColor(rgb_img, cv2.COLOR_BGR2GRAY) # Covert BGR to Grayscale
print(gray_img[100, 150]) # Print Pixel intensity at (100, 150)

cv2.imshow('RGB', rgb_img) # Show RGB Image
cv2.imshow('Grayscale', gray_img) # Show Grayscale Image
cv2.waitKey(0) # Keeping Window Open Until Pressing a Key on the Keyboard
cv2.destroyAllWindows() # Closing All Windows
```

📚 Visualize an Image in a Plot

```
import cv2
import matplotlib.pyplot as plt # Import Plot Library

image = cv2.imread("photo.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb) # Show Plot of the Image
plt.title("Good Photo") # Setting Title for the Plot
plt.axis("off") # Hides both the x-axis and the y-axis of the Plot
plt.show() # Show the Plot
```

✂️ Resize, Crop, Filter, Rotate, Edge Detection, Color Space Conversions, Convert Formats (BGR ↔ HSV)

```
import cv2

img = cv2.imread('sample.jpg')
resized = cv2.resize(img, (100, 100))  # Resized
cropped = img[50:150, 100:200]  # Cropped dimension range:  y-range, x-range
blurred_filtered = cv2.GaussianBlur(img, (5, 5), 0) # Applied GaussianBlur Filter
rotated = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE) # Rotate 90 Degree Clockwise
edges = cv2.Canny(img, 100, 200) # Edge Detection by Canny
RGB_to_HSV = cv2.cvtColor(img , cv2.COLOR_RGB2HSV) # Convert rgb to hsv
HSV_to_RGB = cv2.cvtColor(img , cv2.COLOR_HSV2RGB) # Convert hsv to rgb
cv2.imshow("NameOfImage", oneOfRawImageOrProcessedImagesAbove)
cv2.waitKey(0)
```

3 Basic Statistics & Data Visualization for Vision Datasets

💉 Objective:

- Explore pixel data distributions and summarize image information.
- Perform basic statistics on image datasets.
- Visualize class distributions.
- Detect anomalies or imbalance.

📈 Libraries:

- Numpy
- Matplotlib
- seaborn

👩‍🏫 Tutorial Steps:

- ✓ Extract image channels and compute statistics
- ✓ Visualize grayscale histograms
- ✓ Compare pixel intensity distributions across images
- ✓ Understanding Data & Visualize Dataset Stats
- ✓ Dataset shape, summary statistics (mean, variance, std), class distribution
- ✓ Data distributions and outlier detection
- ✓ Explore vision datasets (MNIST, CIFAR)
- ✓ Exploratory Data Analysis (EDA) of image datasets
- ✓ Plotting (visualization) with Matplotlib & Seaborn
- ✓ Histograms and box plots
- ✓ Plot distributions, calculate mean/std, Mean and variance of pixel intensity

Mini Project: Visualize class-wise image distribution in MNIST or CIFAR-10

📊 Image Dataset Analysis

Understanding dataset before training helps identify class imbalance, image anomalies, or corrupted samples.

✂️ Visualize Dataset Distribution

```
import matplotlib.pyplot as plt
import seaborn as sns

labels = ['Cat']*500 + ['Dog']*1000
sns.countplot(x=labels)
plt.title("Class Distribution")
plt.show()
```

🗂️ Summary Stats

```
import numpy as np
import cv2

image = cv2.imread("photo.jpg")
gray_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
mean_pixel = np.mean(gray_img)
std_pixel = np.std(gray_img)
print(f"Mean: {mean_pixel:.2f}, Std: {std_pixel:.2f}")
```

🧩 Histogram Plot

```
import matplotlib.pyplot as plt
import cv2

image = cv2.imread("photo.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_flat = gray.flatten()
plt.hist(gray_flat, bins=50, color='gray')
plt.title("Pixel Intensity Distribution")
plt.xlabel("Intensity")
plt.ylabel("Frequency")
plt.show()
```

📚 Visualize an Image Matrices in a Plot

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

# Load image
img = cv2.imread("photo.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Show image matrices
plt.figure(figsize=(8,4))
plt.subplot(1,2,1); plt.title("RGB"); plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.subplot(1,2,2); plt.title("Grayscale"); plt.imshow(gray, cmap='gray')
plt.show()
```

🗂️ Summary Stats

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Simulate labels
labels = ['Cat']*300 + ['Dog']*700
sns.countplot(x=labels)
plt.title("Class Distribution")
plt.show()

# Image stats
from PIL import Image
gray = Image.open("photo.png").convert("L")
pixels = np.array(gray).flatten()
print("Mean:", np.mean(pixels))
print("Std Dev:", np.std(pixels))
```

4 Fundamentals of Supervised Machine Learning (ML) for CV

🚀 **Objective**:

- ❖ Introduce ML concepts and apply them to visual data features.
- ❖ Learn linear regression and decision trees
- ❖ Train models on image-derived features

Definition:
Learning from labeled data where each input (image) has a known output (label). The model finds patterns to predict labels on new unseen images.

- **Topics Covered:**
  - Linear Regression for pixel intensity prediction
  - Decision Trees on image features
  - Extract features from images (pixel intensity, color histograms, mean pixel value, edge count)
  - Create a dataset for classification (e.g., cat vs dog)
  - Train, test, split, cross-validation, evaluation metrics and model (accuracy, precision)
  - Train a decision tree classifier
- **Libraries:** Scikit-learn, NumPy
- **Code Example:** with scikit-learn (sklearn**)**

💡 **Key Algorithms:**

- ➢ LinearRegression
- ➢ DecisionTreeClassifier

📐 Example Algorithms

Linear Regression (for image-derived features):

```
from sklearn.linear_model import LinearRegression

X = [[20], [40], [60]]  # Feature: brightness
y = [1, 2, 3]            # Target label

model = LinearRegression()
model.fit(X, y) # model.fit(X_train, y_train)
print("Prediction for brightness=50:", model.predict([[50]]))
print(f'Accuracy: {model.score(X, y):.2f}') # model.score(X_test, y_test)
```

Decision Trees:

```
from sklearn.tree import DecisionTreeClassifier

X = [[128, 0], [200, 1]]  # Features: avg_brightness, edge_flag
y = [0, 1]                # Classes: Cat=0, Dog=1

clf = DecisionTreeClassifier()
clf.fit(X, y)
print("Class Prediction:", clf.predict([[150, 0]]))
```

Linear Regression + Decision Trees:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier

# Toy data
X = [[0], [100], [150]]  # brightness
y = [0, 1, 1]             # label

# Linear model
lr = LinearRegression()
lr.fit(X, y)
print("Prediction for 120:", lr.predict([[120]]))

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X, y)
print("Tree Prediction:", dt.predict([[120]]))
```

5️⃣ Intro to Convolutional Neural Network (CNN): Architecture & Use Cases

🧠 Objective:

- ❖ Understand basic structure of CNNs layers and their applications.
- ❖ Understand MNIST digit classifier with PyTorch
- ❖ Train a classifier on MNIST

🤖 What is a CNN?

Definition:
A Convolutional Neural Network (CNN) is a specialized deep learning model for image data. It uses convolution filters to detect patterns like edges, shapes, and textures.

- **Topics Covered:**
  - What are Convolutions, Pooling, FC layers?
  - Activation functions, Flattening layers, Kernel, Stride, Padding
  - CNN layers: Conv2D, MaxPooling, Fully Connected (FC)
  - Popular architectures: LeNet, AlexNet (intro-level)
  - Feed-forward architecture for image classification
  - Evaluation metrics: accuracy, confusion matrix
  - Building a basic CNNs with Keras or PyTorch
  - Train on MNIST dataset
  - Visualize predictions and misclassifications
- **Libraries:** MNIST digit classifier using PyTorch (Torchvision, Torchvision.Datasets.MNIST)
- **Code Snippet:**

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, 3)
        self.fc = nn.Linear(16*26*26, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

🧱 Common Layers Explained:

- **Conv2D**: Applies filters to extract features
- **ReLU:** Introduces non-linearity
- **MaxPooling (MaxPool2D):** Reduces image size while preserving info (DownSampling)
- **Fully Connected:** Final decision-making
- **Linear:** Classification

🔢 **MNIST Digit Classifier (**Classify Handwritten Digits by MNIST**)**

- Load dataset using **`torchvision`** or **keras.datasets** （Load MNIST data and preprocess）
- Build and Train a simple CNN with 2–3 layers
- Evaluate accuracy and visualize predictions
- Plot misclassified digits and confusion matrix

🛠️ Hands-On Projects

📥 **Example: MNIST Digit Classifier**

```
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3)  # 1-channel grayscale input
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16*13*13, 10)  # Output 10 digits

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 16*13*13)
        x = self.fc1(x)
        return x
```

Use with PyTorch's **`torchvision.datasets.MNIST`** for training.

💻 Sample Code (Keras)

```
from keras import models, layers, datasets
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()
x_train = x_train[..., None] / 255.0
x_test = x_test[..., None] / 255.0
model = models.Sequential([ layers.Conv2D(32, (3, 3),
                            activation='relu',
                            input_shape=(28,28,1)),
                            layers.MaxPooling2D((2, 2)),
                            layers.Flatten(),
                            layers.Dense(100, activation='relu'),
                            layers.Dense(10, activation='softmax')
                                                                ])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

🧪 Hands-on Projects

Project	Tools	Description
Photo Filter App	OpenCV + tkinter	GUI for image filtering
MNIST Classifier	PyTorch	Train CNN to classify digits
Class Visualizer	Matplotlib	EDA on dataset distribution
Image Feature Predictor	Scikit-learn	ML with handcrafted features

📚 Tools & Dependencies

- Python ≥ 3.8

- OpenCV (`opencv-python`)

- PyTorch + torchvision

- Matplotlib + Seaborn

- Scikit-learn

🧠 ML Algorithms & CNNs

- Train a Decision Tree on feature-extracted images
- Build & train a CNN for MNIST
- 🌟 Challenge: “Replace ReLU with LeakyReLU and compare results”

```
import torch.nn as nn
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(1, 16, 3)
        self.fc = nn.Linear(16*26*26, 10)

    def forward(self, x):
        x = F.relu(self.conv(x))
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

🧠 Why Preprocess?

- Ensuring consistent input for CNNs
- Highlighting meaningful features

🎓 Mini Project: Build Your Own Image Editor

Create a notebook where students upload a photo, apply transformations, and save the result. Bonus challenge: add a GUI with `cv2.imshow()` + keyboard controls!

🛠️ **Interactive Tasks**

1. Crop the image to center the face.
2. Apply different kernel sizes for blurring.
3. Build a tkinter GUI to toggle filters.

🛠️ **Suggested Projects (Progressively Challenging)**

Project	Skills Practiced	Tools
MNIST Classifier	CNN basics	PyTorch, torchvision
Dog vs. Cat Classifier	Data pipelines, fine-tuning	PyTorch, transfer learning
Custom YOLOv5 Detector	Object detection	Ultralytics YOLO repo
Semantic Segmentation	Masking, pixel-wise classification	PyTorch + OpenCV

📘 Crystal-Clear Documentation Style

Module Doc Example

📊 Visual Flow Diagram

[Insert block diagram showing image flow from file → preprocessing → display]

💻 Extra Code Examples and Notebooks

- [x] CNN training with PyTorch (starter + intermediate)

- [x] Object detection pipeline with pre-trained YOLOv5

- [x] Semantic segmentation using U-Net

- [x] Jupyter notebook: Data Augmentation walkthrough

- [x] Custom dataset loader tutorial (images + annotations)