

Day 1: Computer Vision Foundations

Build Base Tooling + Data Intuition
NumPy + Pandas + OpenCV Basics

made with ♡, by Aryan

Contents

1	Session Overview	3
2	Part 1: What is Computer Vision?	3
2.1	Definition	3
2.2	Real-World Applications	3
2.2.1	1. Object Detection	3
2.2.2	2. Optical Character Recognition (OCR)	3
2.2.3	3. Image Segmentation	3
2.2.4	4. Augmented Reality (AR)	4
2.2.5	5. Robotics Vision	4
2.3	Classical Image Processing vs Deep Learning Vision	4
3	Part 2: NumPy Essentials	4
3.1	Why NumPy for Computer Vision?	4
3.2	Arrays, Shape, and Dimensions	5
3.3	Array Creation Methods	5
3.4	Slicing and Indexing	5
3.5	Broadcasting	6
3.6	Element-wise Operations	7
3.7	Reshaping and Flattening	7
3.8	Vectorization vs Loops	8
3.9	Useful NumPy Functions for Images	8
4	Part 3: Pandas for Vision Datasets	9
4.1	Why Pandas for Computer Vision?	9
4.2	Loading and Exploring Data	9
4.3	Creating Dataset DataFrames	10
4.4	Selecting Rows and Columns	10
4.5	Merging and Cleaning	11
4.6	Dataset Splitting	11
4.7	Connecting Images with Labels	12
5	Part 4: OpenCV Basics	13
5.1	What is OpenCV and Why Use It?	13
5.2	Reading and Writing Images	13
5.3	Color Space Conversions	14
5.4	Resizing and Cropping	15
5.5	Blurring	16
5.6	Canny Edge Detection	16

6	Part 5: Live Coding - Edge Detection Mini-Project	17
7	Mini Tasks	20
7.1	Task 1: Image Processing Pipeline	20
7.2	Task 2: Pixel Histogram Analysis	20
8	Assignment: Pencil Sketch Effect	20
9	Resources and Next Steps	21
9.1	Documentation	21
9.2	Preview of Day 2	21

1 Session Overview

Welcome to Day 1! Today we build the foundational toolkit for computer vision work. By the end of this session, you'll understand what computer vision is, master NumPy for array operations, use Pandas for dataset management, and perform essential image processing with OpenCV.

Learning Objectives:

- Understand Computer Vision and its real-world applications
- Master NumPy: arrays, broadcasting, vectorization
- Use Pandas for vision annotation tables and data management
- Perform image operations with OpenCV
- Complete edge detection mini-project

2 Part 1: What is Computer Vision?

2.1 Definition

Computer Vision (CV) is a field of artificial intelligence that enables computers to derive meaningful information from digital images, videos, and other visual inputs. It teaches machines to "see" and understand the visual world.

2.2 Real-World Applications

2.2.1 1. Object Detection

Identifying and locating objects within images or video streams.

Examples:

- **Autonomous vehicles:** Detecting pedestrians, traffic signs, lane markers, other vehicles
- **Retail analytics:** Tracking inventory on shelves, customer behavior analysis
- **Security systems:** Identifying unauthorized persons, suspicious packages
- **Manufacturing:** Quality control, defect detection

2.2.2 2. Optical Character Recognition (OCR)

Converting images of text into machine-readable text.

Examples:

- **Document digitization:** Books, historical documents, receipts
- **License plate recognition:** Parking systems, toll booths
- **Check processing:** Banking automation
- **Text extraction:** From signs, product labels, invoices

2.2.3 3. Image Segmentation

Dividing images into meaningful regions or segments.

Examples:

- **Medical imaging:** Tumor detection, organ segmentation, disease diagnosis
- **Satellite imagery:** Land use classification, crop monitoring
- **Photo editing:** Background removal, portrait mode
- **Robotics:** Object grasping, scene understanding

2.2.4 4. Augmented Reality (AR)

Overlaying digital content on real-world views.

Examples:

- **Social media:** Snapchat/Instagram filters, facial effects
- **E-commerce:** IKEA Place app (furniture visualization), virtual try-on
- **Gaming:** Pokemon GO, AR games
- **Navigation:** Google Maps AR walking directions

2.2.5 5. Robotics Vision

Enabling robots to perceive and interact with environments.

Examples:

- **Industrial robots:** Quality inspection, assembly line automation
- **Warehouse automation:** Amazon robots, inventory management
- **Agriculture:** Crop health monitoring, automated harvesting
- **Healthcare:** Surgical robots, patient monitoring

2.3 Classical Image Processing vs Deep Learning Vision

Aspect	Classical Methods	Deep Learning
Approach	Hand-crafted algorithms	Learn from data
Features	Manual design (SIFT, HOG)	Automatic feature learning
Training	No training needed	Requires labeled data
Performance	Good for specific tasks	Excellent for complex tasks
Computational	Fast, efficient	Resource-intensive (GPU)
Scalability	Plateaus quickly	Improves with more data
Examples	Edge detection, template matching	CNNs, YOLO, Mask R-CNN

Table 1: Classical vs Deep Learning Computer Vision

When to use each:

- **Classical:** Preprocessing, simple detection, resource-constrained environments, well-defined problems
- **Deep Learning:** Complex patterns, large datasets available, end-to-end solutions, state-of-the-art accuracy needed

3 Part 2: NumPy Essentials

NumPy is the foundation of numerical computing in Python. Images are represented as NumPy arrays, making NumPy mastery essential for CV work.

3.1 Why NumPy for Computer Vision?

- **Images = Arrays:** Grayscale (2D), Color (3D)
- **Speed:** C-optimized operations, 50-100x faster than Python loops
- **Vectorization:** Apply operations to entire arrays
- **Memory efficiency:** Handles large datasets
- **Ecosystem:** Integrates with OpenCV, PIL, scikit-image

3.2 Arrays, Shape, and Dimensions

```
1 import numpy as np
2
3 # Creating arrays
4 arr_1d = np.array([1, 2, 3, 4, 5])
5 arr_2d = np.array([[1, 2, 3],
6                   [4, 5, 6],
7                   [7, 8, 9]])
8
9 # Array properties
10 print(f"Shape: {arr_2d.shape}")      # (3, 3) - rows, columns
11 print(f"Dimensions: {arr_2d.ndim}")  # 2
12 print(f"Size: {arr_2d.size}")        # 9 total elements
13 print(f>Data type: {arr_2d.dtype}")  # int64
14
15 # Image array representations
16 # Grayscale: (height, width)
17 gray_image = np.zeros((480, 640), dtype=np.uint8)
18 print(f"Grayscale shape: {gray_image.shape}") # (480, 640)
19
20 # RGB: (height, width, channels)
21 color_image = np.zeros((480, 640, 3), dtype=np.uint8)
22 print(f"Color shape: {color_image.shape}")   # (480, 640, 3)
23
24 # Batch of images: (batch_size, height, width, channels)
25 batch = np.zeros((32, 224, 224, 3), dtype=np.uint8)
26 print(f"Batch shape: {batch.shape}")        # (32, 224, 224, 3)
```

3.3 Array Creation Methods

```
1 # Zeros and ones
2 zeros = np.zeros((3, 3))
3 ones = np.ones((2, 4))
4 full = np.full((3, 3), 7) # Fill with specific value
5
6 # Identity matrix
7 identity = np.eye(4)
8
9 # Random arrays
10 random_uniform = np.random.rand(3, 3)      # Uniform [0, 1)
11 random_normal = np.random.randn(3, 3)      # Normal distribution
12 random_int = np.random.randint(0, 255, (3, 3)) # Integers
13
14 # Ranges
15 arange = np.arange(0, 10, 2)                # [0 2 4 6 8]
16 linspace = np.linspace(0, 1, 5)             # 5 evenly spaced values
17
18 # Like operations (match shape/dtype of existing array)
19 arr = np.array([[1, 2], [3, 4]])
20 zeros_like = np.zeros_like(arr)
21 ones_like = np.ones_like(arr)
```

3.4 Slicing and Indexing

```
1 arr = np.array([[10, 20, 30, 40],
2                 [50, 60, 70, 80],
3                 [90, 100, 110, 120]])
4
5 # Basic indexing
```

```

6 print(arr[0, 0])      # 10 - first element
7 print(arr[2, 3])      # 120 - last element
8 print(arr[-1, -1])    # 120 - negative indexing
9
10 # Row and column selection
11 print(arr[0, :])      # [10 20 30 40] - first row
12 print(arr[:, 1])      # [20 60 100] - second column
13 print(arr[:, -1])     # [40 80 120] - last column
14
15 # Slicing ranges
16 print(arr[0:2, 1:3])
17 # [[20 30]
18 #  [60 70]]
19
20 # Boolean indexing
21 mask = arr > 50
22 print(arr[mask])      # [60 70 80 90 100 110 120]
23
24 # Fancy indexing
25 rows = [0, 2]
26 cols = [1, 3]
27 print(arr[rows, cols]) # [20 120]
28
29 # For images - crop center region
30 image = np.random.randint(0, 255, (480, 640, 3), dtype=np.uint8)
31 h, w = image.shape[:2]
32 crop_size = 200
33 center_crop = image[
34     h//2 - crop_size//2 : h//2 + crop_size//2,
35     w//2 - crop_size//2 : w//2 + crop_size//2
36 ]
37 print(f"Original: {image.shape}, Cropped: {center_crop.shape}")

```

3.5 Broadcasting

Broadcasting allows NumPy to work with arrays of different shapes during arithmetic operations.

```

1 # Scalar broadcasting
2 arr = np.array([[1, 2, 3],
3                 [4, 5, 6]])
4 result = arr + 10 # Add 10 to every element
5 print(result)
6 # [[11 12 13]
7 #  [14 15 16]]
8
9 # 1D to 2D broadcasting
10 arr_2d = np.array([[1, 2, 3],
11                    [4, 5, 6]])
12 arr_1d = np.array([10, 20, 30])
13
14 result = arr_2d + arr_1d # arr_1d broadcasts across rows
15 print(result)
16 # [[11 22 33]
17 #  [14 25 36]]
18
19 # Column broadcasting
20 col_vector = np.array([[10],
21                         [20]])
22 result = arr_2d + col_vector # Broadcasts across columns
23 print(result)
24 # [[11 12 13]
25 #  [24 25 26]]

```

```

26
27 # Image example - adjust RGB channels independently
28 image = np.random.randint(0, 255, (100, 100, 3), dtype=np.uint8)
29 channel_adjustments = np.array([1.2, 1.0, 0.8]) # Boost red, reduce blue
30 adjusted = np.clip(image * channel_adjustments, 0, 255).astype(np.uint8)

```

Broadcasting Rules:

1. Compare dimensions from right to left
2. Dimensions must be equal, or one must be 1
3. Missing dimensions are assumed to be 1

3.6 Element-wise Operations

```

1 arr = np.array([1, 2, 3, 4, 5])
2
3 # Arithmetic
4 print(arr + 10)      # [11 12 13 14 15]
5 print(arr * 2)       # [2 4 6 8 10]
6 print(arr ** 2)      # [1 4 9 16 25]
7 print(arr / 2)       # [0.5 1. 1.5 2. 2.5]
8
9 # Array to array
10 arr2 = np.array([5, 4, 3, 2, 1])
11 print(arr + arr2)    # [6 6 6 6 6]
12 print(arr * arr2)    # [5 8 9 8 5]
13
14 # Comparison operations
15 print(arr > 3)       # [False False False True True]
16 print(arr == 3)      # [False False True False False]
17
18 # Mathematical functions
19 print(np.sqrt(arr))  # Square root
20 print(np.exp(arr))   # Exponential
21 print(np.log(arr))   # Natural log
22 print(np.sin(arr))   # Sine
23
24 # Image brightness adjustment
25 image = np.random.randint(0, 200, (100, 100), dtype=np.uint8)
26 brighter = np.clip(image + 50, 0, 255).astype(np.uint8)
27 darker = np.clip(image - 50, 0, 255).astype(np.uint8)
28 contrast = np.clip(image * 1.5, 0, 255).astype(np.uint8)

```

3.7 Reshaping and Flattening

```

1 # Reshape
2 arr = np.array([1, 2, 3, 4, 5, 6])
3 reshaped = arr.reshape(2, 3)
4 print(reshaped)
5 # [[1 2 3]
6 #   [4 5 6]]
7
8 # Automatic dimension inference with -1
9 auto = arr.reshape(3, -1) # NumPy calculates: (3, 2)
10 print(auto.shape)
11
12 # Flatten to 1D
13 flat = reshaped.flatten() # Returns copy
14 ravel = reshaped.ravel()  # Returns view (faster)

```

```

15
16 # Transpose
17 transposed = reshaped.T
18 print(transposed)
19 # [[1 4]
20 #   [2 5]
21 #   [3 6]]
22
23 # Image examples
24 # Flatten for ML model input
25 mnist_image = np.random.randint(0, 255, (28, 28), dtype=np.uint8)
26 flat_image = mnist_image.flatten()
27 print(f"Image shape: {mnist_image.shape}, Flat: {flat_image.shape}")
28 # (28, 28) -> (784,)
29
30 # Add batch dimension
31 single_image = np.random.rand(224, 224, 3)
32 batched = single_image.reshape(1, 224, 224, 3)
33 # or
34 batched = np.expand_dims(single_image, axis=0)
35
36 # Rearrange dimensions (HWC to CHW for PyTorch)
37 hwc_image = np.random.rand(224, 224, 3)
38 chw_image = np.transpose(hwc_image, (2, 0, 1))
39 print(f"HWC: {hwc_image.shape}, CHW: {chw_image.shape}")

```

3.8 Vectorization vs Loops

Vectorization is crucial for performance in CV applications.

```

1 import time
2 import numpy as np
3
4 # Create test data
5 arr = np.random.rand(1000000)
6
7 # LOOP APPROACH (SLOW)
8 start = time.time()
9 result_loop = np.zeros_like(arr)
10 for i in range(len(arr)):
11     result_loop[i] = arr[i] ** 2
12 loop_time = time.time() - start
13
14 # VECTORIZED APPROACH (FAST)
15 start = time.time()
16 result_vectorized = arr ** 2
17 vectorized_time = time.time() - start
18
19 print(f"Loop time: {loop_time:.4f}s")
20 print(f"Vectorized time: {vectorized_time:.4f}s")
21 print(f"Speedup: {loop_time / vectorized_time:.1f}x faster!")
22 # Typically 50-100x faster!
23
24 # Verify results are identical
25 print(f"Results match: {np.allclose(result_loop, result_vectorized)}")

```

Key Principle: Always prefer vectorized operations over loops for array processing.

3.9 Useful NumPy Functions for Images

```

1 image = np.random.randint(0, 255, (480, 640, 3), dtype=np.uint8)
2

```

```
3 # Statistical operations
4 print(f"Mean: {image.mean()}")
5 print(f"Std: {image.std()}")
6 print(f"Min: {image.min()}, Max: {image.max()}")
7 print(f"Median: {np.median(image)}")
8
9 # Per-channel statistics
10 print(f"Mean per channel: {image.mean(axis=(0, 1))}") # [R, G, B] means
11 print(f"Std per channel: {image.std(axis=(0, 1))}")
12
13 # Clipping values
14 clipped = np.clip(image, 50, 200) # Constrain to [50, 200]
15
16 # Normalization
17 normalized = (image - image.min()) / (image.max() - image.min())
18
19 # Standardization (zero mean, unit variance)
20 standardized = (image - image.mean()) / image.std()
21
22 # Concatenation
23 img1 = np.zeros((100, 100, 3), dtype=np.uint8)
24 img2 = np.ones((100, 100, 3), dtype=np.uint8) * 255
25
26 hstack = np.hstack([img1, img2]) # Horizontal: (100, 200, 3)
27 vstack = np.vstack([img1, img2]) # Vertical: (200, 100, 3)
28 dstack = np.dstack([img1, img2]) # Depth: (100, 100, 6)
29
30 # Stacking for batches
31 batch = np.stack([img1, img2, img1]) # (3, 100, 100, 3)
32
33 # Splitting arrays
34 split_images = np.split(batch, 3, axis=0) # Split batch into individual images
35
36 # Where (conditional selection)
37 # Create binary mask
38 mask = image > 127
39 # Apply different operations based on condition
40 output = np.where(mask, 255, 0) # White where >127, black otherwise
```

4 Part 3: Pandas for Vision Datasets

Pandas excels at organizing image metadata, file paths, labels, and dataset splits.

4.1 Why Pandas for Computer Vision?

- **Organize paths:** Manage thousands of image file locations
- **Label management:** Store and manipulate class labels
- **Dataset operations:** Train/val/test splits, filtering, sampling
- **Annotation tables:** Bounding boxes, segmentation masks, metadata
- **Analysis:** Quick dataset statistics and visualizations

4.2 Loading and Exploring Data

```
1 import pandas as pd
2 import os
3
4 # Load CSV with image annotations
5 df = pd.read_csv('annotations.csv')
6
```

```

7 # Basic exploration
8 print(df.head())           # First 5 rows
9 print(df.tail())           # Last 5 rows
10 print(df.shape)            # (rows, columns)
11 print(df.columns)          # Column names
12 print(df.dtypes)           # Data types
13 print(df.info())           # Summary info
14 print(df.describe())       # Statistical summary
15
16 # Select specific columns
17 filenames = df['filename']
18 labels = df['label']
19
20 # Select multiple columns
21 subset = df[['filename', 'label', 'width', 'height']]
22
23 # Select rows by condition
24 cats = df[df['label'] == 'cat']
25 large_images = df[df['width'] > 640]

```

4.3 Creating Dataset DataFrames

```

1 import os
2 from pathlib import Path
3
4 # Example 1: Build from folder structure
5 # Assume: dataset/class_name/image.jpg
6 dataset_path = 'dataset/'
7 data = []
8
9 for class_name in os.listdir(dataset_path):
10     class_path = os.path.join(dataset_path, class_name)
11
12     if os.path.isdir(class_path):
13         for img_file in os.listdir(class_path):
14             if img_file.endswith((' .jpg', '.jpeg', '.png')):
15                 data.append({
16                     'filepath': os.path.join(class_path, img_file),
17                     'filename': img_file,
18                     'label': class_name,
19                     'class_id': hash(class_name) % 1000 # Numeric label
20                 })
21
22 df = pd.DataFrame(data)
23 print(df.head())
24 print(f"\nDataset size: {len(df)} images")
25 print(f"Classes: {df['label'].unique()}")

```

4.4 Selecting Rows and Columns

```

1 # Select single column (returns Series)
2 labels = df['label']
3
4 # Select multiple columns (returns DataFrame)
5 subset = df[['filename', 'label']]
6
7 # Row selection by index
8 first_row = df.iloc[0]           # First row
9 last_row = df.iloc[-1]           # Last row
10 rows_1_to_5 = df.iloc[1:6]      # Rows 1-5

```

```
11
12 # Row selection by condition
13 cats = df[df['label'] == 'cat']
14 dogs_or_cats = df[df['label'].isin(['dog', 'cat'])]
15 large = df[(df['width'] > 640) & (df['height'] > 480)]
16
17 # Loc vs iloc
18 # iloc: integer position
19 # loc: label-based
20 df_indexed = df.set_index('filename')
21 specific_image = df_indexed.loc['img_001.jpg']
```

4.5 Merging and Cleaning

```
1 # Create two dataframes
2 df_images = pd.DataFrame({
3     'image_id': [1, 2, 3, 4],
4     'filepath': ['img1.jpg', 'img2.jpg', 'img3.jpg', 'img4.jpg']
5 })
6
7 df_labels = pd.DataFrame({
8     'image_id': [1, 2, 3, 4],
9     'label': ['cat', 'dog', 'cat', 'bird']
10 })
11
12 # Merge (SQL-like join)
13 df_merged = pd.merge(df_images, df_labels, on='image_id')
14 print(df_merged)
15
16 # Handle missing data
17 df_with_missing = pd.DataFrame({
18     'filename': ['img1.jpg', 'img2.jpg', None, 'img4.jpg'],
19     'label': ['cat', None, 'dog', 'bird'],
20     'width': [640, 480, 800, None]
21 })
22
23 # Check for missing values
24 print(df_with_missing.isnull().sum())
25
26 # Drop rows with any missing values
27 df_clean = df_with_missing.dropna()
28
29 # Drop rows with missing values in specific column
30 df_clean = df_with_missing.dropna(subset=['filename'])
31
32 # Fill missing values
33 df_filled = df_with_missing.fillna({'label': 'unknown', 'width': 640})
34
35 # Drop duplicates
36 df_unique = df.drop_duplicates(subset=['filename'])
```

4.6 Dataset Splitting

```
1 from sklearn.model_selection import train_test_split
2
3 # Stratified split (maintains class distribution)
4 train_df, test_df = train_test_split(
5     df,
6     test_size=0.2,          # 20% for test
7     random_state=42,        # Reproducible split
```

```

8     stratify=df['label'] # Maintain class balance
9 )
10
11 # Further split train into train + validation
12 train_df, val_df = train_test_split(
13     train_df,
14     test_size=0.2,          # 20% of train = validation
15     random_state=42,
16     stratify=train_df['label']
17 )
18
19 print(f"Train: {len(train_df)} ({len(train_df)/len(df)*100:.1f}%)")
20 print(f"Val: {len(val_df)} ({len(val_df)/len(df)*100:.1f}%)")
21 print(f"Test: {len(test_df)} ({len(test_df)/len(df)*100:.1f}%)")
22
23 # Add split column to main dataframe
24 df['split'] = 'train'
25 df.loc[df.index.isin(val_df.index), 'split'] = 'val'
26 df.loc[df.index.isin(test_df.index), 'split'] = 'test'
27
28 # Verify class distribution is maintained
29 print("\nClass distribution per split:")
30 print(df.groupby(['split', 'label']).size().unstack(fill_value=0))
31
32 # Save split dataset
33 df.to_csv('dataset_with_splits.csv', index=False)

```

4.7 Connecting Images with Labels

```

1 import cv2
2 import matplotlib.pyplot as plt
3
4 def load_image(filepath):
5     """Load and convert image to RGB"""
6     img = cv2.imread(filepath)
7     if img is None:
8         return None
9     return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
10
11 # Add image properties to dataframe
12 def get_image_info(filepath):
13     img = cv2.imread(filepath)
14     if img is None:
15         return pd.Series([None, None, None])
16     h, w = img.shape[:2]
17     channels = img.shape[2] if len(img.shape) == 3 else 1
18     return pd.Series([h, w, channels])
19
20 df[['height', 'width', 'channels']] = df['filepath'].apply(get_image_info)
21
22 # Analyze dataset
23 print("\nImage size distribution:")
24 print(df[['height', 'width']].describe())
25
26 # Find unusual images
27 mean_h, mean_w = df['height'].mean(), df['width'].mean()
28 unusual = df[
29     (df['height'] < mean_h * 0.5) |
30     (df['width'] < mean_w * 0.5) |
31     (df['height'] > mean_h * 2) |
32     (df['width'] > mean_w * 2)
33 ]

```

```

34 print(f"\nFound {len(unusual)} images with unusual dimensions")
35
36 # Visualize samples from each class
37 def plot_class_samples(df, n_samples=5):
38     classes = df['label'].unique()
39     fig, axes = plt.subplots(len(classes), n_samples, figsize=(15, 3*len(
40         classes)))
41
42     for i, class_name in enumerate(classes):
43         class_df = df[df['label'] == class_name].sample(n=min(n_samples, len(
44             class_df)))
45         for j, (_, row) in enumerate(class_df.iterrows()):
46             img = load_image(row['filepath'])
47             if img is not None:
48                 axes[i, j].imshow(img)
49                 axes[i, j].set_title(f"{class_name}")
50                 axes[i, j].axis('off')
51
52     plt.tight_layout()
53     plt.show()
54 # plot_class_samples(df, n_samples=5)

```

5 Part 4: OpenCV Basics

OpenCV (Open Source Computer Vision Library) is the most widely used library for real-time computer vision.

5.1 What is OpenCV and Why Use It?

OpenCV Features:

- 2500+ optimized algorithms
- Real-time processing capabilities
- C++ backend (fast)
- Python bindings (easy)
- Cross-platform (Windows, Linux, macOS, mobile)
- Camera and video support
- Extensive documentation and community

Installation:

```

1 pip install opencv-python
2 pip install opencv-contrib-python # Extra modules

```

5.2 Reading and Writing Images

```

1 import cv2
2 import numpy as np
3
4 # Read image (returns BGR format!)
5 image = cv2.imread('image.jpg')
6
7 # Check if loaded successfully
8 if image is None:
9     print("Error: Could not load image")
10 else:
11     print(f"Image loaded: {image.shape}")

```

```

12
13 # Read modes
14 gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE) # Grayscale
15 unchanged = cv2.imread('image.png', cv2.IMREAD_UNCHANGED) # With alpha
16
17 # Image properties
18 height, width, channels = image.shape
19 print(f"Dimensions: {width}x{height}, Channels: {channels}")
20 print(f"Data type: {image.dtype}")
21 print(f"Total pixels: {image.size}")
22
23 # Write image
24 cv2.imwrite('output.jpg', image)
25 cv2.imwrite('output.png', image) # Auto-detects format from extension
26
27 # Write with quality settings (JPEG)
28 cv2.imwrite('output.jpg', image, [cv2.IMWRITE_JPEG_QUALITY, 95])
29
30 # Write PNG with compression
31 cv2.imwrite('output.png', image, [cv2.IMWRITE_PNG_COMPRESSION, 9])

```

Important: OpenCV uses BGR color order, not RGB! Always convert for matplotlib display:

```

1 import matplotlib.pyplot as plt
2
3 # Wrong way - BGR displayed as RGB (colors wrong)
4 plt.imshow(image)
5 plt.show()
6
7 # Correct way - convert to RGB first
8 image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
9 plt.imshow(image_rgb)
10 plt.show()

```

5.3 Color Space Conversions

```

1 image = cv2.imread('image.jpg')
2
3 # BGR to RGB
4 rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
5
6 # BGR to Grayscale
7 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
8
9 # BGR to HSV (Hue, Saturation, Value)
10 hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
11
12 # BGR to LAB
13 lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
14
15 # Grayscale to BGR (3-channel grayscale)
16 gray_bgr = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
17
18 # RGB to Grayscale (if you have RGB)
19 # First ensure it's RGB, then:
20 gray_from_rgb = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
21
22 print(f"Original: {image.shape}")
23 print(f"Grayscale: {gray.shape}")
24 print(f"HSV: {hsv.shape}")

```

5.4 Resizing and Cropping

```
1 image = cv2.imread('image.jpg')
2 h, w = image.shape[:2]
3
4 # Resize to specific dimensions (width, height)
5 resized = cv2.resize(image, (320, 240))
6
7 # Resize with scale factor
8 scaled_down = cv2.resize(image, None, fx=0.5, fy=0.5)
9 scaled_up = cv2.resize(image, None, fx=2.0, fy=2.0)
10
11 # Resize maintaining aspect ratio
12 def resize_with_aspect_ratio(img, target_width=None, target_height=None):
13     h, w = img.shape[:2]
14
15     if target_width is not None:
16         aspect = h / w
17         new_h = int(target_width * aspect)
18         return cv2.resize(img, (target_width, new_h))
19
20     if target_height is not None:
21         aspect = w / h
22         new_w = int(target_height * aspect)
23         return cv2.resize(img, (new_w, target_height))
24
25     return img
26
27 resized_aspect = resize_with_aspect_ratio(image, target_width=640)
28
29 # Interpolation methods
30 # INTER_NEAREST - Fastest, lowest quality (good for pixel art)
31 # INTER_LINEAR - Default, good balance
32 # INTER_CUBIC - Slower, better quality (good for upscaling)
33 # INTER_LANCZOS4 - Best quality, slowest
34 # INTER_AREA - Best for downscaling
35
36 upscaled_hq = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC
37 )
38 downscaled = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.
39     INTER_AREA)
40
41 # Cropping (using array slicing)
42 # Format: image[y1:y2, x1:x2]
43
44 # Crop top-left 100x100
45 crop_tl = image[0:100, 0:100]
46
47 # Crop center region
48 center_x, center_y = w // 2, h // 2
49 crop_size = 200
50 center_crop = image[
51     center_y - crop_size//2 : center_y + crop_size//2,
52     center_x - crop_size//2 : center_x + crop_size//2
53 ]
54
55 # Crop bottom-right quarter
56 crop_br = image[h//2:, w//2:]
57
58 print(f"Original: {image.shape}")
59 print(f"Center crop: {center_crop.shape}")
```

5.5 Blurring

```

1 image = cv2.imread('image.jpg')
2
3 # 1. Gaussian Blur - Most common, good for noise reduction
4 blur_gaussian = cv2.GaussianBlur(image, (5, 5), 0)
5 # (5,5) = kernel size (must be odd)
6 # 0 = sigmaX (calculated automatically if 0)
7
8 # Stronger blur
9 blur_strong = cv2.GaussianBlur(image, (15, 15), 0)
10
11 # 2. Median Blur - Best for salt-and-pepper noise
12 blur_median = cv2.medianBlur(image, 5) # Kernel size (odd)
13
14 # 3. Average Blur - Simple averaging
15 blur_average = cv2.blur(image, (5, 5))
16
17 # 4. Bilateral Filter - Smooths while preserving edges
18 blur_bilateral = cv2.bilateralFilter(image, 9, 75, 75)
19 # d = diameter
20 # sigmaColor = filter sigma in color space
21 # sigmaSpace = filter sigma in coordinate space
22
23 # Compare results
24 import matplotlib.pyplot as plt
25
26 images = [image, blur_gaussian, blur_median, blur_bilateral]
27 titles = ['Original', 'Gaussian', 'Median', 'Bilateral']
28
29 fig, axes = plt.subplots(1, 4, figsize=(16, 4))
30 for img, title, ax in zip(images, titles, axes):
31     ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
32     ax.set_title(title)
33     ax.axis('off')
34 plt.tight_layout()
35 plt.show()

```

5.6 Canny Edge Detection

```

1 image = cv2.imread('image.jpg')
2 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
3
4 # Apply Gaussian blur first (reduces noise)
5 blurred = cv2.GaussianBlur(gray, (5, 5), 0)
6
7 # Canny edge detection
8 edges = cv2.Canny(blurred, threshold1=50, threshold2=150)
9 # threshold1 = lower threshold for weak edges
10 # threshold2 = upper threshold for strong edges
11 # Edge pixels between thresholds are included only if connected to strong edges
12
13 # Try different thresholds
14 edges_sensitive = cv2.Canny(blurred, 30, 100) # More edges
15 edges_strict = cv2.Canny(blurred, 100, 200) # Fewer edges
16 edges_auto = cv2.Canny(blurred, 0, 255) # Auto thresholds
17
18 # Automatic threshold calculation (Otsu's method inspiration)
19 def auto_canny(image, sigma=0.33):
20     """Automatically determine Canny thresholds"""
21     median_val = np.median(image)
22     lower = int(max(0, (1.0 - sigma) * median_val))

```

```

23     upper = int(min(255, (1.0 + sigma) * median_val))
24     return cv2.Canny(image, lower, upper)
25
26 edges_auto_calc = auto_canny(blurred)
27
28 # Display results
29 import matplotlib.pyplot as plt
30
31 fig, axes = plt.subplots(2, 3, figsize=(15, 10))
32 axes = axes.ravel()
33
34 axes[0].imshow(gray, cmap='gray')
35 axes[0].set_title('Original Grayscale')
36
37 axes[1].imshow(blurred, cmap='gray')
38 axes[1].set_title('Blurred')
39
40 axes[2].imshow(edges, cmap='gray')
41 axes[2].set_title('Canny (50, 150)')
42
43 axes[3].imshow(edges_sensitive, cmap='gray')
44 axes[3].set_title('Sensitive (30, 100)')
45
46 axes[4].imshow(edges_strict, cmap='gray')
47 axes[4].set_title('Strict (100, 200)')
48
49 axes[5].imshow(edges_auto_calc, cmap='gray')
50 axes[5].set_title('Auto Threshold')
51
52 for ax in axes:
53     ax.axis('off')
54
55 plt.tight_layout()
56 plt.show()

```

6 Part 5: Live Coding - Edge Detection Mini-Project

Build a complete project that processes a folder of images and displays edge detection results.

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5 from pathlib import Path
6
7 def detect_edges(image_path, low_threshold=50, high_threshold=150, blur_kernel
=5):
8     """
9     Apply Canny edge detection to an image with preprocessing.
10
11     Args:
12         image_path: Path to input image
13         low_threshold: Lower threshold for Canny
14         high_threshold: Upper threshold for Canny
15         blur_kernel: Gaussian blur kernel size (must be odd)
16
17     Returns:
18         original_image_rgb, edges_image
19     """
20     # Read image
21     image = cv2.imread(image_path)
22

```

```
23     if image is None:
24         print(f'Error: Could not load {image_path}')
25         return None, None
26
27     # Convert to RGB for display
28     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
29
30     # Convert to grayscale
31     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
32
33     # Apply Gaussian blur to reduce noise
34     blurred = cv2.GaussianBlur(gray, (blur_kernel, blur_kernel), 0)
35
36     # Apply Canny edge detection
37     edges = cv2.Canny(blurred, low_threshold, high_threshold)
38
39     return image_rgb, edges
40
41
42 def process_image_folder(folder_path, output_folder=None,
43                          low_threshold=50, high_threshold=150):
44     """
45     Process all images in a folder and display/save edge detection results.
46
47     Args:
48         folder_path: Path to folder containing images
49         output_folder: Optional path to save edge-detected images
50         low_threshold: Canny lower threshold
51         high_threshold: Canny upper threshold
52     """
53     # Create output folder if specified
54     if output_folder:
55         Path(output_folder).mkdir(parents=True, exist_ok=True)
56         print(f"Created output folder: {output_folder}")
57
58     # Get list of image files
59     image_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tiff']
60     image_files = [
61         f for f in os.listdir(folder_path)
62         if Path(f).suffix.lower() in image_extensions
63     ]
64
65     if not image_files:
66         print(f"No images found in {folder_path}")
67         return
68
69     print(f'Found {len(image_files)} images')
70
71     # Process each image
72     for idx, img_file in enumerate(image_files, 1):
73         img_path = os.path.join(folder_path, img_file)
74         print(f'[{idx}/{len(image_files)}] Processing: {img_file}')
75
76         # Detect edges
77         original, edges = detect_edges(img_path, low_threshold, high_threshold)
78
79         if original is None:
80             continue
81
82         # Create side-by-side display
83         fig, axes = plt.subplots(1, 2, figsize=(14, 7))
84
85         axes[0].imshow(original)
```

```

86     axes[0].set_title(f'Original: {img_file}')
87     axes[0].axis('off')
88
89     axes[1].imshow(edges, cmap='gray')
90     axes[1].set_title(f'Edge Detection (Canny)')
91     axes[1].axis('off')
92
93     plt.tight_layout()
94     plt.show()
95
96     # Save edge image if output folder specified
97     if output_folder:
98         output_name = f'edges_{Path(img_file).stem}.png'
99         output_path = os.path.join(output_folder, output_name)
100        cv2.imwrite(output_path, edges)
101        print(f'  Saved: {output_path}')
102
103        print() # Empty line for readability
104
105
106 def batch_compare_thresholds(image_path):
107     """
108     Compare different Canny thresholds on a single image.
109     """
110     image = cv2.imread(image_path)
111     if image is None:
112         print(f"Error loading {image_path}")
113         return
114
115     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
116     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
117
118     # Different threshold combinations
119     thresholds = [
120         (30, 100, 'Sensitive'),
121         (50, 150, 'Balanced'),
122         (100, 200, 'Strict'),
123         (0, 255, 'Auto')
124     ]
125
126     fig, axes = plt.subplots(2, 2, figsize=(12, 12))
127     axes = axes.ravel()
128
129     for idx, (low, high, name) in enumerate(thresholds):
130         edges = cv2.Canny(blurred, low, high)
131         axes[idx].imshow(edges, cmap='gray')
132         axes[idx].set_title(f'{name}\n(low={low}, high={high})')
133         axes[idx].axis('off')
134
135     plt.suptitle(f'Canny Edge Detection - Threshold Comparison', fontsize=16)
136     plt.tight_layout()
137     plt.show()
138
139
140 # Example usage
141 if __name__ == '__main__':
142     # Process folder
143     process_image_folder(
144         folder_path='sample_images',
145         output_folder='edge_results',
146         low_threshold=50,
147         high_threshold=150
148     )

```

```
149
150     # Compare thresholds on single image
151     # batch_compare_thresholds('sample_images/test.jpg')
```

7 Mini Tasks

7.1 Task 1: Image Processing Pipeline

Create a processing pipeline that applies multiple operations and displays them in a grid.

Requirements:

1. Load an image
2. Apply Gaussian blur (7x7 kernel)
3. Apply Canny edge detection
4. Apply binary thresholding
5. Display all 4 images in a 2x2 grid

7.2 Task 2: Pixel Histogram Analysis

Create a program that analyzes and visualizes pixel intensity distributions.

Requirements:

1. Load a grayscale image
2. Calculate histogram of pixel intensities
3. Display image and histogram side-by-side
4. Add statistics (mean, median, std)

8 Assignment: Pencil Sketch Effect

Due: Next session

Objective: Create a program that converts any image to a realistic pencil sketch effect.

Algorithm:

1. Convert image to grayscale
2. Invert grayscale: $inverted = 255 - gray$
3. Apply Gaussian blur to inverted (kernel size 21x21)
4. Invert the blurred result: $inverted_blur = 255 - blurred$
5. Divide and scale: $sketch = \frac{gray}{inverted_blur} \times 256$
6. Clip values to $[0, 255]$ range

Requirements:

- Clean, well-commented code
- Function-based design
- Display original and sketch side-by-side
- Save sketch to file

- Test with at least 3 different images
- Handle errors gracefully

Bonus Challenges:

- Add adjustable blur parameter
- Create color pencil sketch version
- Process video file frame-by-frame
- Add GUI with sliders for real-time adjustment

See separate assignment file for detailed instructions and starter code.

9 Resources and Next Steps

9.1 Documentation

- NumPy: <https://numpy.org/doc/>
- Pandas: <https://pandas.pydata.org/docs/>
- OpenCV-Python: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- Matplotlib: <https://matplotlib.org/stable/contents.html>

9.2 Preview of Day 2

Tomorrow we'll dive deeper into:

- Advanced preprocessing and augmentation
- Classical CV techniques (ORB, SIFT, template matching)
- Contour detection and shape analysis
- Real-time webcam applications
- Haar cascade face detection

Great job today! Complete the assignment and see you tomorrow!