Day 2: Image Preprocessing & Classical CV

Augmentation + Feature Detection + Object Recognition

made with ♡, by Aryan

1 Session Overview

Welcome to Day 2! Today we move beyond basic image operations to learn professional preprocessing techniques, data augmentation strategies, and classical computer vision methods that are still widely used in production systems. These techniques form the foundation for modern deep learning pipelines.

2 Learning Objectives

- Understand why image preprocessing is critical for ML success
- Master normalization and standardization techniques
- Implement data augmentation to improve model robustness
- Apply classical CV methods: edge detection, feature extraction, template matching
- Build a real-time face detection webcam application

3 Part 1: Why Preprocessing Matters

Image preprocessing transforms raw images into a format optimal for machine learning models. Poor preprocessing can doom even the best models to failure.

3.1 Key Reasons for Preprocessing

1. Consistency
Models expect consistent input dimensions, scales, and formats.

- All images must be the same size (e.g., 224x224)
- Pixel values must be in consistent range (0-1 or -1 to 1)
- Color space must be standardized

2. Numerical Stability
Neural networks work better with normalized inputs.

- Prevents gradient explosion or vanishing
- Speeds up convergence during training
- Reduces sensitivity to initialization

3. Feature Enhancement
Preprocessing can make important features more visible.

- Contrast adjustment reveals hidden details
- Noise reduction improves signal
- Edge enhancement sharpens boundaries

4. Data Efficiency
Augmentation artificially expands limited datasets.

- Reduces overfitting with more training variations
- Improves model robustness to real-world variations
- Balances class distributions

4 Part 2: Normalization and Standardization

Scaling pixel values to appropriate ranges is fundamental to deep learning success.

4.1 Min-Max Normalization

Scales values to a fixed range, typically [0, 1].

```
import cv2
import numpy as np

# Load image (values are 0-255)
image = cv2.imread('image.jpg')
print(f'Original range: {image.min()} to {image.max()}')  # 0 to 255

# Min-Max normalization to [0, 1]
normalized = image.astype(np.float32) / 255.0
print(f'Normalized range: {normalized.min():.2f} to {normalized.max()
    :.2f}')

# General min-max formula: (x - min) / (max - min)
min_val = image.min()
max_val = image.max()
normalized_general = (image - min_val) / (max_val - min_val)

# Scale to different range, e.g., [-1, 1]
normalized_neg = (image.astype(np.float32) / 127.5) - 1.0
print(f'[-1, 1] range: {normalized_neg.min():.2f} to {normalized_neg.
    max():.2f}')
```

4.2 Z-Score Standardization

Transforms data to have mean=0 and std=1. Common for pretrained models.

```
# Z-score standardization: (x - mean) / std
image = cv2.imread('image.jpg').astype(np.float32)

# Calculate per-channel statistics
mean = np.mean(image, axis=(0, 1))  # Mean per channel
std = np.std(image, axis=(0, 1))    # Std per channel

print(f'Mean: {mean}')
print(f'Std: {std}')

# Standardize
standardized = (image - mean) / std

print(f'Standardized mean: {standardized.mean():.6f}')  # ~0
print(f'Standardized std: {standardized.std():.6f}')    # ~1

```

```python
# ImageNet statistics (commonly used for transfer learning)
IMAGENET_MEAN = np.array([0.485, 0.456, 0.406])
IMAGENET_STD = np.array([0.229, 0.224, 0.225])

# First normalize to [0, 1], then standardize with ImageNet stats
image_norm = image / 255.0
image_standardized = (image_norm - IMAGENET_MEAN) / IMAGENET_STD
```

Teaching Note: Explain that ImageNet statistics are used because many pretrained models were trained on ImageNet with these values. Using the same normalization ensures the model receives inputs in the expected distribution.

5 Part 3: Data Augmentation

Data augmentation creates variations of training images to improve model generalization. It's one of the most effective techniques to prevent overfitting.

5.1 Geometric Transformations

```python
import cv2
import numpy as np

image = cv2.imread('image.jpg')
height, width = image.shape[:2]

# 1. Horizontal Flip
flipped_h = cv2.flip(image, 1)  # 1 = horizontal

# 2. Vertical Flip
flipped_v = cv2.flip(image, 0)  # 0 = vertical

# 3. Both Flips
flipped_both = cv2.flip(image, -1)  # -1 = both

# 4. Rotation
# Get rotation matrix
center = (width // 2, height // 2)
angle = 45  # degrees
scale = 1.0
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# Apply rotation
rotated = cv2.warpAffine(image, rotation_matrix, (width, height))

# 5. Random Rotation
random_angle = np.random.uniform(-30, 30)  # Random angle between -30
    and 30
rotation_matrix = cv2.getRotationMatrix2D(center, random_angle, 1.0)
random_rotated = cv2.warpAffine(image, rotation_matrix, (width, height)
    )

# 6. Scaling (Zoom in/out)
scale_factor = 1.2  # Zoom in
scaled = cv2.resize(image, None, fx=scale_factor, fy=scale_factor)
# Crop back to original size
start_y = (scaled.shape[0] - height) // 2
start_x = (scaled.shape[1] - width) // 2
```

```
zoomed = scaled[start_y:start_y+height, start_x:start_x+width]

# 7. Translation (Shift)
tx, ty = 50, 30  # Shift 50 pixels right, 30 pixels down
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
translated = cv2.warpAffine(image, translation_matrix, (width, height))

# 8. Shearing
shear_factor = 0.2
shear_matrix = np.float32([[1, shear_factor, 0], [0, 1, 0]])
sheared = cv2.warpAffine(image, shear_matrix, (width, height))
```

5.2 Color Augmentations

```
# 1. Brightness Adjustment
brightness_factor = 50  # Add 50 to all pixels
brighter = np.clip(image.astype(np.int16) + brightness_factor, 0, 255).
    astype(np.uint8)

# Random brightness
random_brightness = np.random.randint(-50, 50)
adjusted = np.clip(image.astype(np.int16) + random_brightness, 0, 255).
    astype(np.uint8)

# 2. Contrast Adjustment
contrast_factor = 1.5  # Increase contrast
contrasted = np.clip(image.astype(np.float32) * contrast_factor, 0,
    255).astype(np.uint8)

# 3. Saturation Adjustment (requires HSV)
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).astype(np.float32)
saturation_factor = 1.5
hsv[:, :, 1] = np.clip(hsv[:, :, 1] * saturation_factor, 0, 255)
saturated = cv2.cvtColor(hsv.astype(np.uint8), cv2.COLOR_HSV2BGR)

# 4. Hue Shift
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).astype(np.float32)
hue_shift = 30  # Shift hue by 30 degrees
hsv[:, :, 0] = (hsv[:, :, 0] + hue_shift) % 180
hue_shifted = cv2.cvtColor(hsv.astype(np.uint8), cv2.COLOR_HSV2BGR)

# 5. Add Gaussian Noise
mean = 0
std = 25
gaussian_noise = np.random.normal(mean, std, image.shape)
noisy = np.clip(image + gaussian_noise, 0, 255).astype(np.uint8)
```

5.3 Comprehensive Augmentation Function

```
def augment_image(image, augmentation_probability=0.5):
    """
    Apply random augmentations to an image.
    Each augmentation is applied with probability
    augmentation_probability.
    """
    height, width = image.shape[:2]
```

```
    result = image.copy()

    # Random horizontal flip
    if np.random.random() < augmentation_probability:
        result = cv2.flip(result, 1)

    # Random rotation
    if np.random.random() < augmentation_probability:
        angle = np.random.uniform(-15, 15)
        center = (width // 2, height // 2)
        M = cv2.getRotationMatrix2D(center, angle, 1.0)
        result = cv2.warpAffine(result, M, (width, height))

    # Random brightness
    if np.random.random() < augmentation_probability:
        brightness = np.random.randint(-40, 40)
        result = np.clip(result.astype(np.int16) + brightness, 0, 255).
astype(np.uint8)

    # Random contrast
    if np.random.random() < augmentation_probability:
        contrast = np.random.uniform(0.7, 1.3)
        result = np.clip(result.astype(np.float32) * contrast, 0, 255).
astype(np.uint8)

    # Random noise
    if np.random.random() < augmentation_probability:
        noise = np.random.normal(0, 15, result.shape)
        result = np.clip(result + noise, 0, 255).astype(np.uint8)

    return result
```

6 Part 4: Classical CV Techniques

Before deep learning dominated computer vision, classical techniques were the only approach. Many remain useful today for preprocessing, feature engineering, and computationally-constrained applications.

6.1 Advanced Edge Detection

```
import cv2
import numpy as np

image = cv2.imread('image.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 1. Sobel Edge Detection (gradient-based)
# Detects horizontal edges
sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)

# Detects vertical edges
sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)

# Combine both directions
sobel_combined = np.sqrt(sobelx**2 + sobely**2)
sobel_combined = np.uint8(sobel_combined)
```

```
17
18 # 2. Laplacian Edge Detection (second derivative)
19 laplacian = cv2.Laplacian(gray, cv2.CV_64F)
20 laplacian = np.uint8(np.absolute(laplacian))
21
22 # 3. Canny (still the best for most cases)
23 canny = cv2.Canny(gray, 50, 150)
```

6.2 Contour Detection

Contours are curves joining continuous points along a boundary. Useful for shape analysis and object detection.

```
1  # Find contours requires binary image
2  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
3  ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
4
5  # Find contours
6  contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
       CHAIN_APPROX_SIMPLE)
7
8  print(f'Found {len(contours)} contours')
9
10 # Draw all contours
11 result = image.copy()
12 cv2.drawContours(result, contours, -1, (0, 255, 0), 2)  # -1 means all
       contours
13
14 # Filter contours by area
15 large_contours = [c for c in contours if cv2.contourArea(c) > 1000]
16 print(f'Large contours: {len(large_contours)}')
17
18 # Draw bounding boxes around contours
19 for contour in large_contours:
20     x, y, w, h = cv2.boundingRect(contour)
21     cv2.rectangle(result, (x, y), (x + w, y + h), (255, 0, 0), 2)
22
23     # Get contour properties
24     area = cv2.contourArea(contour)
25     perimeter = cv2.arcLength(contour, True)
26     print(f'Area: {area:.2f}, Perimeter: {perimeter:.2f}')
```

6.3 Hough Transform - Line and Circle Detection

```
1  # Detect lines
2  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
3  edges = cv2.Canny(gray, 50, 150)
4
5  # Hough Line Transform
6  lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=100,
7                          minLineLength=50, maxLineGap=10)
8
9  # Draw detected lines
10 result = image.copy()
11 if lines is not None:
12     for line in lines:
13         x1, y1, x2, y2 = line[0]
```

```
        cv2.line(result, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Detect circles
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)  # Reduce noise

circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, dp=1, minDist=50,
                           param1=50, param2=30, minRadius=10,
    maxRadius=100)

# Draw detected circles
if circles is not None:
    circles = np.uint16(np.around(circles))
    for circle in circles[0, :]:
        cx, cy, radius = circle
        # Draw circle outline
        cv2.circle(result, (cx, cy), radius, (0, 255, 0), 2)
        # Draw center point
        cv2.circle(result, (cx, cy), 2, (0, 0, 255), 3)
```

6.4 Feature Detection with ORB

ORB (Oriented FAST and Rotated BRIEF) is a fast, free alternative to SIFT and SURF. It detects keypoints and computes descriptors for image matching.

```
# Create ORB detector
orb = cv2.ORB_create(nfeatures=500)  # Detect up to 500 keypoints

# Detect keypoints and compute descriptors
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
keypoints, descriptors = orb.detectAndCompute(gray, None)

print(f'Found {len(keypoints)} keypoints')
print(f'Descriptor shape: {descriptors.shape}')  # (n_keypoints, 32)

# Draw keypoints
result = cv2.drawKeypoints(image, keypoints, None,
                           flags=cv2.
    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Match keypoints between two images
image2 = cv2.imread('image2.jpg')
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
keypoints2, descriptors2 = orb.detectAndCompute(gray2, None)

# Create BFMatcher (Brute Force Matcher)
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(descriptors, descriptors2)

# Sort matches by distance (lower is better)
matches = sorted(matches, key=lambda x: x.distance)

# Draw top 50 matches
match_img = cv2.drawMatches(image, keypoints, image2, keypoints2,
                            matches[:50], None,
                            flags=cv2.
    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

6.5 Template Matching

Find a small template image within a larger image. Good for detecting known objects without training.

```
# Load template and main image
template = cv2.imread('template.jpg', 0)  # Grayscale
image = cv2.imread('image.jpg', 0)

template_h, template_w = template.shape

# Perform template matching
result = cv2.matchTemplate(image, template, cv2.TM_CCOEFF_NORMED)

# Find best match location
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

# Draw rectangle around match
top_left = max_loc
bottom_right = (top_left[0] + template_w, top_left[1] + template_h)

image_rgb = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
cv2.rectangle(image_rgb, top_left, bottom_right, (0, 255, 0), 2)

print(f'Best match confidence: {max_val:.4f}')
print(f'Match location: {top_left}')
```

6.6 Face Detection with Haar Cascades

Haar Cascades are pre-trained classifiers for detecting faces and other objects. Fast and effective for real-time applications.

```
# Load pre-trained face detector
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
)

# Load eye detector (optional)
eye_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + 'haarcascade_eye.xml'
)

# Detect faces
image = cv2.imread('faces.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor=1.1,  # How much to reduce image size at each scale
    minNeighbors=5,    # How many neighbors required to retain
   detection
    minSize=(30, 30)   # Minimum face size
)

print(f'Found {len(faces)} faces')

# Draw rectangles around faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

```
    # Detect eyes within face region
    face_roi_gray = gray[y:y+h, x:x+w]
    face_roi_color = image[y:y+h, x:x+w]

    eyes = eye_cascade.detectMultiScale(face_roi_gray)

    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(face_roi_color, (ex, ey), (ex + ew, ey + eh), (0,
    255, 0), 2)
```

7 Part 5: Live Coding - Real-Time Face Detection

Let's build a complete webcam application that detects faces in real-time.

```
import cv2
import numpy as np

def detect_faces_webcam():
    """
    Real-time face detection using webcam.
    Press 'q' to quit, 's' to save current frame.
    """
    # Load face cascade
    face_cascade = cv2.CascadeClassifier(
        cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
    )

    # Open webcam (0 is default camera)
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print('Error: Could not open webcam')
        return

    # Set camera resolution (optional)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

    frame_count = 0

    print('Starting face detection... Press q to quit, s to save frame'
   )

    while True:
        # Read frame
        ret, frame = cap.read()

        if not ret:
            print('Error: Failed to capture frame')
            break

        # Convert to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces
        faces = face_cascade.detectMultiScale(
```

```
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30)
        )

        # Draw rectangles and labels
        for (x, y, w, h) in faces:
            # Draw rectangle
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0),
    2)

            # Add label
            cv2.putText(frame, 'Face', (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Add info text
        info_text = f'Faces detected: {len(faces)} | Press q to quit, s
     to save'
        cv2.putText(frame, info_text, (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

        # Display frame
        cv2.imshow('Face Detection', frame)

        # Handle key presses
        key = cv2.waitKey(1) & 0xFF

        if key == ord('q'):
            print('Quitting...')
            break
        elif key == ord('s'):
            filename = f'face_detection_{frame_count}.jpg'
            cv2.imwrite(filename, frame)
            print(f'Saved {filename}')
            frame_count += 1

    # Cleanup
    cap.release()
    cv2.destroyAllWindows()
    print('Face detection completed')


if __name__ == '__main__':
    detect_faces_webcam()
```

8 Hands-On Tasks

8.1 Task 1: Augmentation Pipeline

Create a script that loads images from a folder and generates 5 augmented versions of each image using different techniques. Save all augmented images to an output folder.

8.2 Task 2: Feature Matching

Take two photos of the same object from different angles. Use ORB to detect keypoints and match them between the two images. Display the matches and count how many good matches you found.

9 Assignment: Circle Detector

Build a program that detects circles in images using the Hough Circle Transform.

Requirements:

- Load an image containing circular objects
- Apply appropriate preprocessing (grayscale, blur)
- Detect all circles
- Draw circles and label them with radius
- Print statistics (number of circles, average radius)
- Save the result

Bonus: Extend your face detection webcam app to also detect smiles using `haarcascade_smile.xml`

Excellent work today! Tomorrow we dive into deep learning!