**BÀI TẬP LỚN MÔN HỌC: XỬ LÝ ẢNH**

**MÃ MÔN HỌC: INT13146**

**Sandipan Dey 2018 Hands-On Image Processing with Python**

**BÀI LÀM**

### **Chương 1: Getting Started with Image Processing**

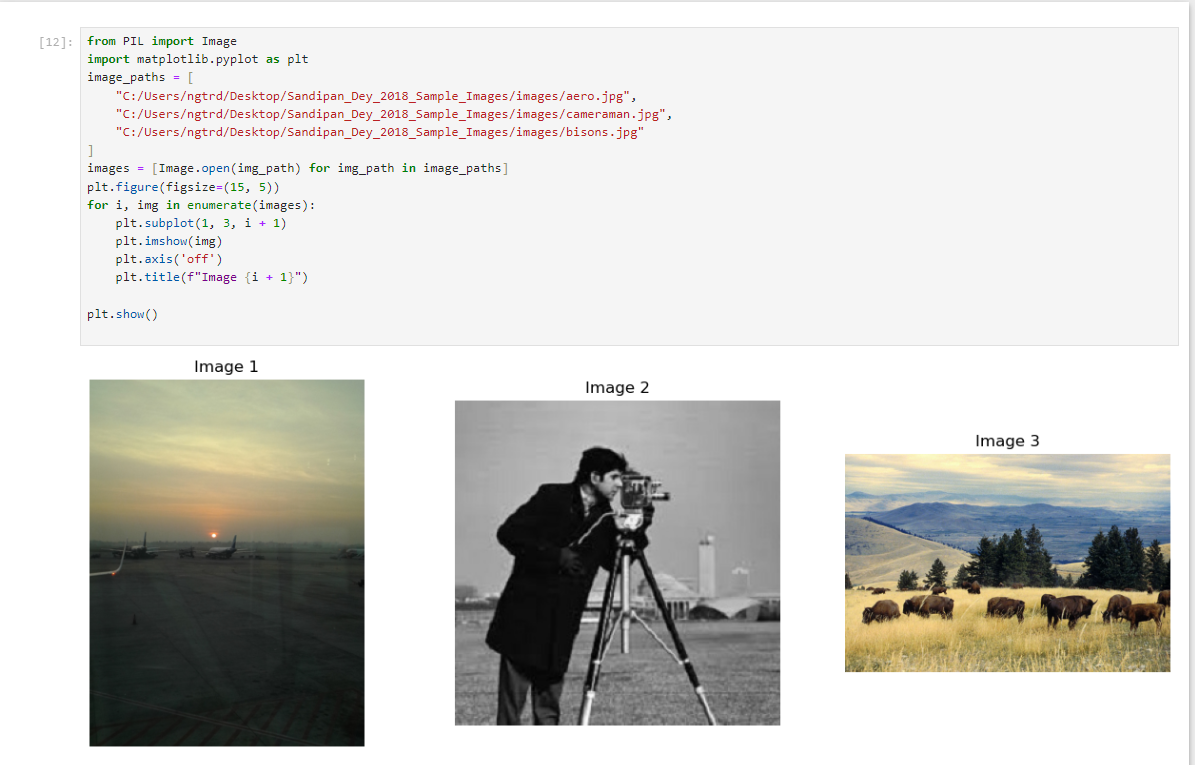
**Thuật toán**: Đọc và hiển thị ảnh

Trong chương này, bạn sẽ học cách đọc và hiển thị ảnh bằng Python và các thư viện như **Pillow** và **Matplotlib**.

**Giải thích**:

* **Dấu / trong đường dẫn**: Python tự động hiểu đường dẫn với dấu / nên bạn không cần thay đổi gì thêm.
* **Đọc ảnh**: Image.open() từ Pillow được sử dụng để mở ảnh từ danh sách image\_paths.
* **Hiển thị ảnh**: plt.subplot(1, 3, i + 1) đặt các ảnh trên một hàng (1 dòng, 3 cột), giúp hiển thị tất cả cùng lúc.

**Kết quả:**

****

### **Chương 2: Sampling, Fourier Transform, and Convolution**

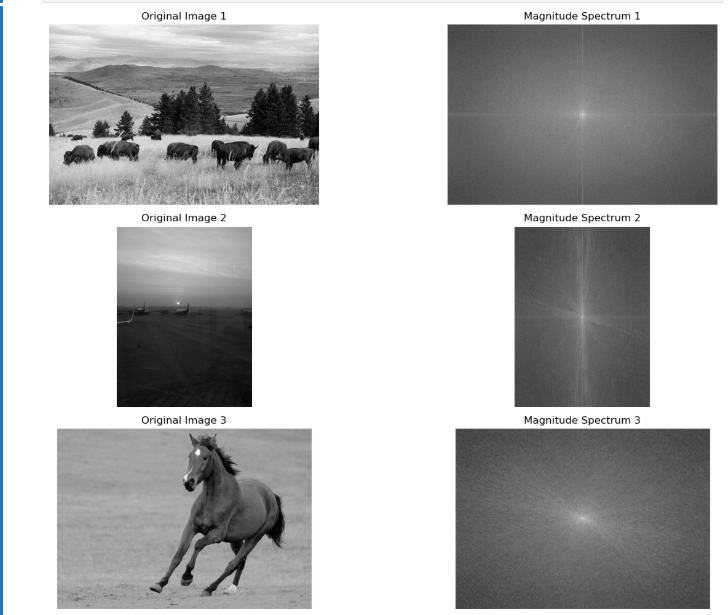
**Thuật toán**: Biến đổi Fourier rời rạc (DFT)

Chương này dạy cách sử dụng **biến đổi Fourier** để phân tích các thành phần tần số trong ảnh.

**** **Giải thích mã nguồn**:

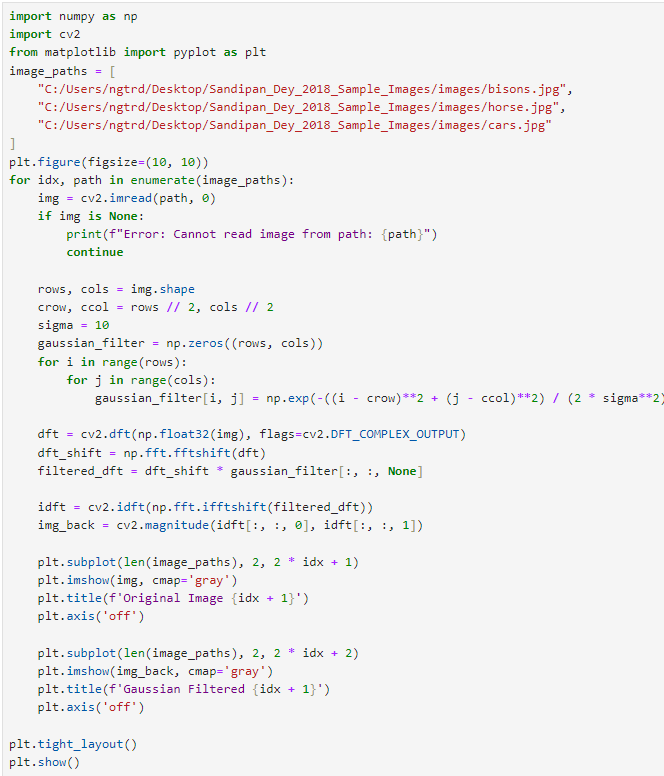
1. **Đường dẫn tới ảnh**:
   * image\_paths chứa danh sách đường dẫn của 3 ảnh cần xử lý.
2. **Xử lý DFT và phổ tần số**:
   * Với mỗi ảnh, thực hiện biến đổi Fourier rời rạc (DFT) bằng cv2.dft và dịch chuyển phổ tần số với np.fft.fftshift.
   * Tính phổ tần số bằng cv2.magnitude.
3. **Hiển thị ảnh**:
   * Sử dụng plt.subplot để tạo lưới 3x2:
     + Cột 1 hiển thị ảnh gốc.
     + Cột 2 hiển thị phổ tần số tương ứng.
4. **Tùy chỉnh hiển thị**:
   * plt.tight\_layout() điều chỉnh bố cục để các hình không bị chồng chéo.

**Kết quả:**

****

### **Chương 3: Convolution and Frequency Domain Filter**

**Thuật toán**: Lọc Gaussian trong miền tần số

**Giải thích mã nguồn**:

1. **Tạo Gaussian filter**:
   * Bộ lọc Gaussian được tính toán dựa trên kích thước của ảnh và giá trị sigma (độ rộng của bộ lọc).
2. **Biến đổi Fourier và lọc**:
   * cv2.dft() thực hiện biến đổi Fourier rời rạc.
   * np.fft.fftshift() dịch phổ tần số để căn giữa tần số thấp.
   * Bộ lọc Gaussian được nhân với phổ tần số để lọc ảnh.
3. **Chuyển đổi ngược**:
   * cv2.idft() chuyển đổi ngược về miền không gian.
   * cv2.magnitude() tính giá trị cường độ để tạo ảnh sau khi lọc.
4. **Hiển thị ảnh**:
   * Mỗi ảnh gốc và ảnh sau lọc được hiển thị trên lưới (plt.subplot()), với mỗi hàng tương ứng với một ảnh

**Kết quả:**

****

### **Chương 4: Image Enhancement**

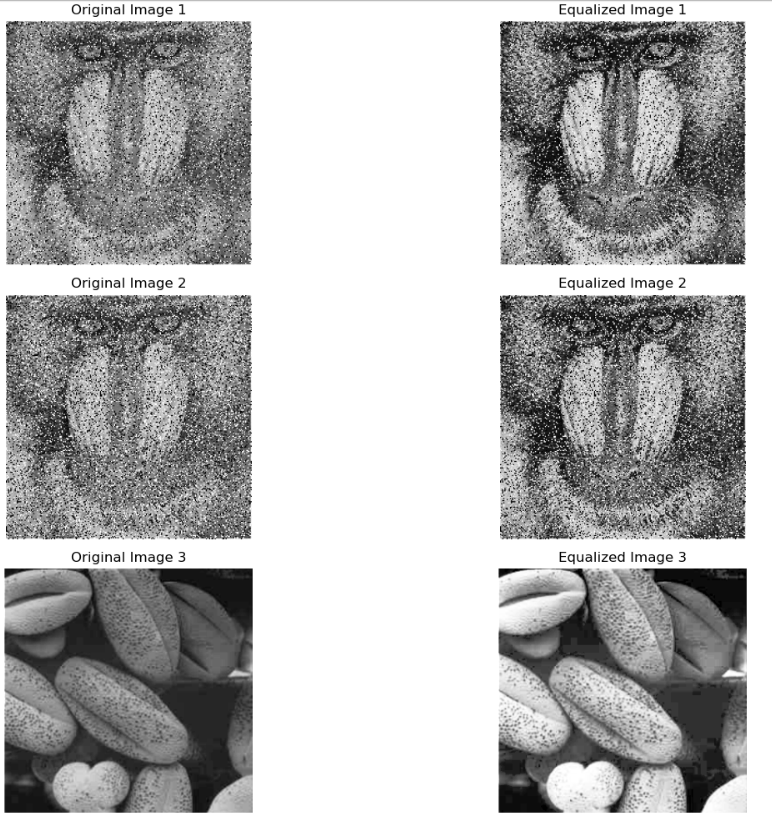
**Thuật toán**: Histogram Equalization



**Giải thích mã nguồn**:

1. **Đường dẫn ảnh**:
   * image\_paths chứa danh sách các đường dẫn tới 3 ảnh cần xử lý.
2. **Histogram Equalization**:
   * cv2.equalizeHist(img) cải thiện độ tương phản của ảnh bằng cách làm đều histogram.
3. **Hiển thị ảnh**:
   * Sử dụng plt.subplot():
     + Cột 1 hiển thị ảnh gốc.
     + Cột 2 hiển thị ảnh sau khi cải thiện.
   * Số hàng của lưới (len(image\_paths)) tương ứng với số ảnh.
4. **Tùy chỉnh hiển thị**:
   * plt.tight\_layout() đảm bảo các hình không bị chồng chéo.

**Kết quả:**

****

Nếu bạn gặp lỗi hoặc cần tối ưu mã nguồn, hãy cho mình biết!

### **Chương 5: Image Enhancement Using Derivatives**

**Thuật toán**: Phát hiện biên cạnh với Sobel

****

**Giải thích mã nguồn**:

1. **Đọc ảnh**:
   * cv2.imread(path, 0) đọc ảnh ở chế độ grayscale (đen trắng).
2. **Phát hiện biên với Sobel filter**:
   * cv2.Sobel() tính toán gradient theo cả hai hướng x và y.
   * cv2.magnitude() tính độ lớn của gradient để phát hiện biên cạnh.
3. **Hiển thị ảnh**:
   * **Cột 1**: Ảnh gốc.
   * **Cột 2**: Biên cạnh phát hiện được bằng Sobel filter.
   * plt.subplot(len(image\_paths), 3, 3 \* idx + 1) sắp xếp các ảnh trong một lưới 3 cột.
4. **Tùy chỉnh hiển thị**:
   * plt.tight\_layout() đảm bảo các hình không bị chồng chéo.

**Kết quả:**

****

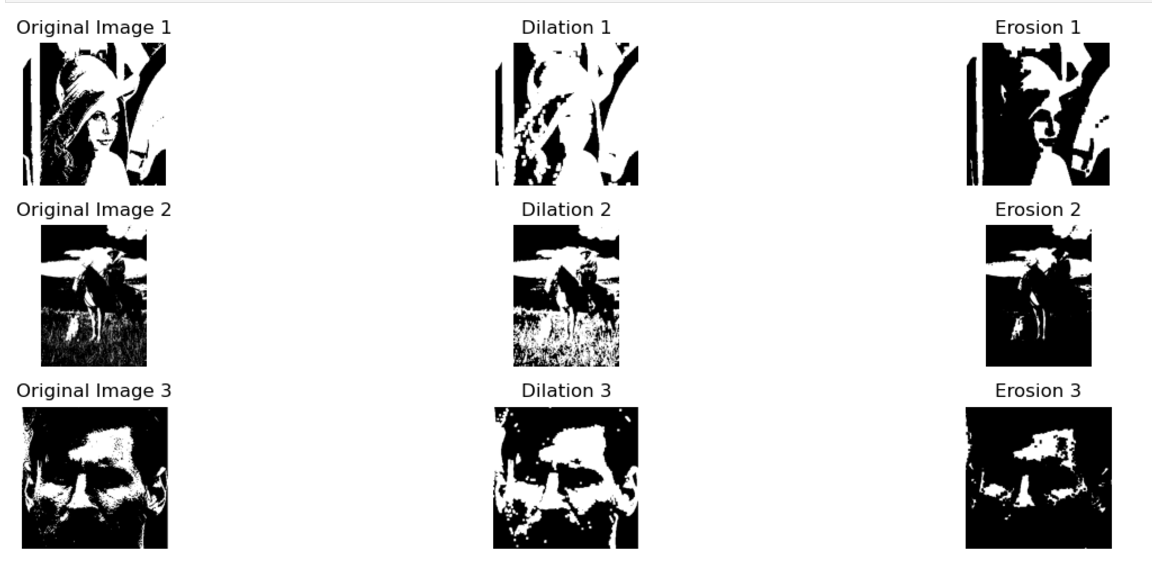
### **Chương 6: Morphological Image Processing**

**Thuật toán**: Dilation và Erosion

**Giải thích mã nguồn**:

1. **Đường dẫn ảnh**:
   * image\_paths chứa danh sách các đường dẫn tới 3 ảnh cần xử lý.
2. **Chuyển ảnh sang nhị phân**:
   * cv2.threshold() chuyển ảnh sang dạng nhị phân, với giá trị ngưỡng 127.
3. **Áp dụng Dilation và Erosion**:
   * cv2.dilate() mở rộng các vùng sáng (dilation).
   * cv2.erode() làm mờ các vùng sáng (erosion).
4. **Hiển thị ảnh**:
   * **Cột 1**: Ảnh gốc sau khi chuyển sang nhị phân.
   * **Cột 2**: Ảnh sau khi áp dụng Dilation.
   * **Cột 3**: Ảnh sau khi áp dụng Erosion.
5. **Tùy chỉnh hiển thị**:
   * plt.tight\_layout() giúp bố cục không bị chồng chéo.

**Kết quả:**

****

### **Chương 7: Extracting Image Features and Descriptors**

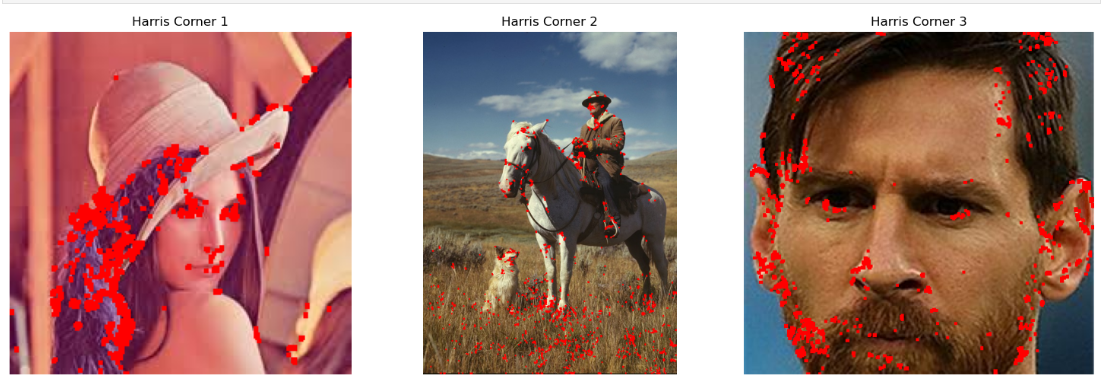
**Thuật toán**: Harris Corner Detection



**Giải thích mã nguồn**:

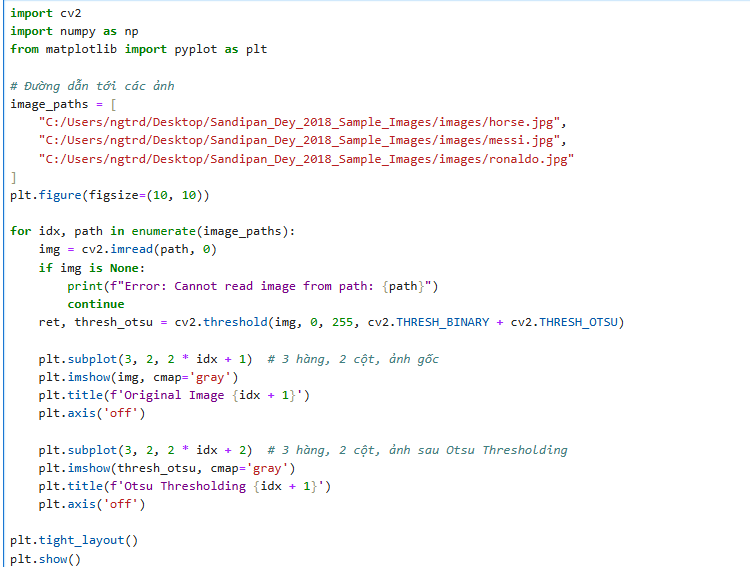
1. **Đọc ảnh**:
   * cv2.imread(path) đọc từng ảnh từ danh sách image\_paths.
   * cv2.cvtColor() chuyển ảnh sang **grayscale** (đen trắng) để xử lý với Harris corner detection.
2. **Harris Corner Detection**:
   * cv2.cornerHarris() phát hiện các góc trong ảnh.
   * cv2.dilate() tăng cường các góc phát hiện được.
3. **Tô đỏ các góc**:
   * img[dst > 0.01 \* dst.max()] = [0, 0, 255] tô đỏ các điểm góc trong ảnh.
4. **Hiển thị kết quả**:
   * plt.subplot(len(image\_paths), 3, 3 \* idx + 1) hiển thị mỗi ảnh trong lưới với 3 cột.
   * cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) chuyển ảnh từ **BGR** sang **RGB** để hiển thị đúng màu trong **Matplotlib**.

**Kết quả:**

****

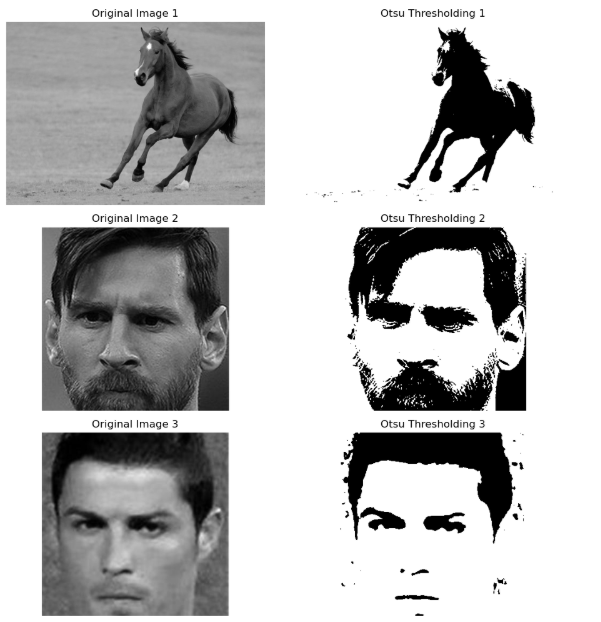
### **Chương 8: Image Segmentation**

**Thuật toán**: Otsu's Thresholding



1. **Đường dẫn ảnh**:
   * image\_paths chứa đường dẫn tới 3 ảnh bạn muốn xử lý. Bạn có thể thay thế các đường dẫn này bằng các đường dẫn chính xác từ máy tính của bạn.
2. **Chuyển ảnh sang grayscale**:
   * cv2.imread(path, 0) đọc ảnh và chuyển thành ảnh đen trắng (grayscale).
3. **Áp dụng Otsu's Thresholding**:
   * cv2.threshold() với tham số cv2.THRESH\_BINARY + cv2.THRESH\_OTSU tự động tìm ngưỡng tốt nhất cho phân đoạn ảnh.
4. **Hiển thị ảnh**:
   * **Cột 1**: Ảnh gốc (chuyển sang nhị phân).
   * **Cột 2**: Ảnh sau khi áp dụng **Otsu's Thresholding**.
   * plt.subplot(3, 2, ...) giúp tạo lưới 3 hàng và 2 cột để hiển thị ảnh.
5. **Tùy chỉnh hiển thị**:

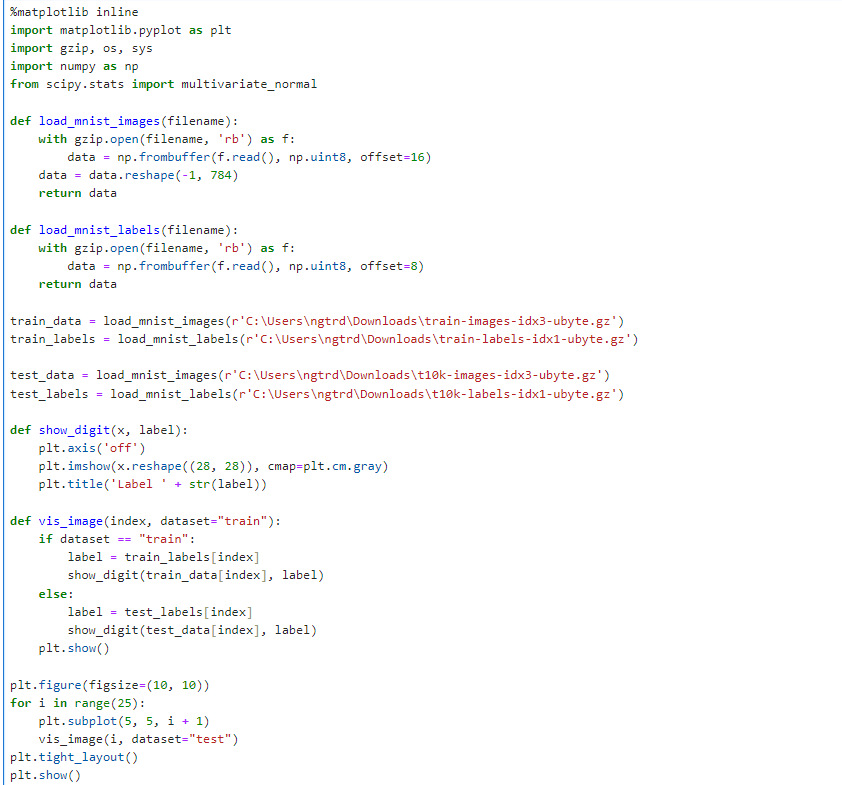
* plt.tight\_layout() giúp căn chỉnh các ảnh sao cho không bị chồng

chéo.

### **Chương 9: Supervised Machine Learning - Image Classification**

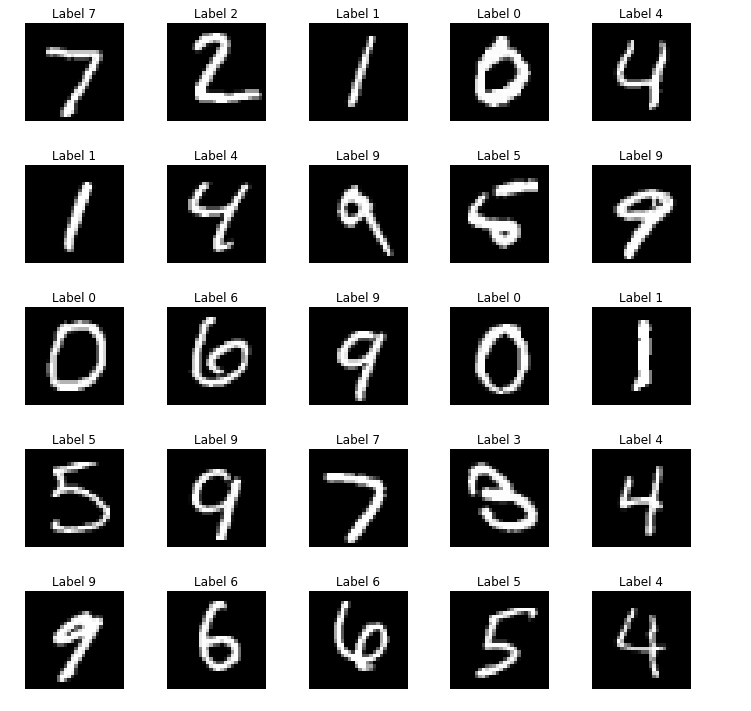
**Thuật toán**: Phân loại ảnh với k-NN

Ứng dụng phân loại chữ viết tay sử dụng bộ dữ liệu MNIST và thuật toán k-NN.

****

**Giải thích ngắn gọn mã đã cung cấp:**

1. **Tải dữ liệu MNIST**:
   * load\_mnist\_images: Mở tệp ảnh MNIST (train-images-idx3-ubyte.gz hoặc t10k-images-idx3-ubyte.gz), đọc dữ liệu và chuyển đổi chúng thành mảng numpy với kích thước 784 (ảnh 28x28).
   * load\_mnist\_labels: Mở tệp nhãn MNIST (train-labels-idx1-ubyte.gz hoặc t10k-labels-idx1-ubyte.gz), đọc và trả về mảng nhãn tương ứng.
2. **Hiển thị ảnh**:
   * show\_digit: Chuyển đổi ảnh từ vector 784 chiều về dạng 28x28 và hiển thị ảnh với nhãn trên đầu.
   * vis\_image: Lựa chọn ảnh từ tập huấn luyện hoặc kiểm tra và gọi show\_digit để hiển thị ảnh với nhãn.
3. **Hiển thị 25 ảnh từ tập kiểm tra**:
   * plt.subplot(5, 5, i + 1): Tạo bảng 5x5 để hiển thị 25 ảnh.
   * plt.tight\_layout(): Căn chỉnh không gian giữa các ảnh để chúng không chồng lên nhau.



### **Chương 10: Deep Learning in Image Processing - Image Classification**

**Thuật toán**: Convolutional Neural Network (CNN)





**Giải thích mã nguồn**:

1. **Mô hình CNN**:
   * Mô hình CNN có 2 lớp Conv2D (mỗi lớp đi kèm với một lớp MaxPooling2D) để trích xuất các đặc trưng của ảnh.
   * Lớp Flatten chuyển các đặc trưng thành một vector.
   * Cuối cùng, có 2 lớp Dense: một lớp với 128 neuron và một lớp cuối cùng với 3 neuron (vì bạn có 3 ảnh, mỗi ảnh sẽ có nhãn tương ứng).
2. **Chuẩn bị dữ liệu**:
   * Đọc và chuẩn hóa 3 ảnh với kích thước (64, 64).
   * Sử dụng tf.keras.utils.to\_categorical() để chuyển nhãn thành dạng **one-hot encoding**.
3. **Huấn luyện mô hình**:
   * Mô hình được huấn luyện trên 3 ảnh với 3 nhãn.
   * Mỗi ảnh có nhãn 0, 1, hoặc 2 để biểu diễn các lớp.
4. **Hiển thị ảnh**:
   * Sau khi huấn luyện, **3 ảnh được hiển thị** với matplotlib.

**Kết quả:**

****

### **Chương 11: Deep Learning in Image Processing - Object Detection**

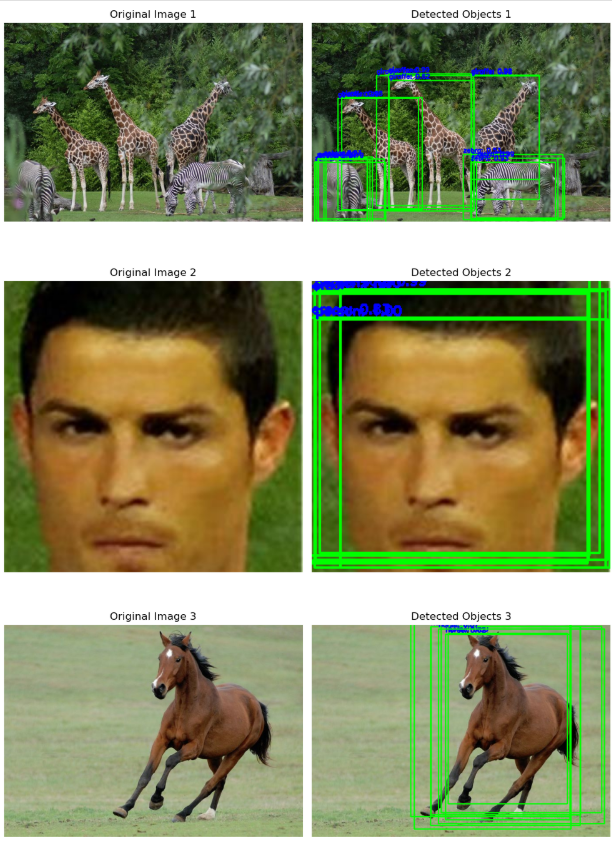
**Thuật toán**: Phát hiện đối tượng với YOLO



**Giải thích mã nguồn**:

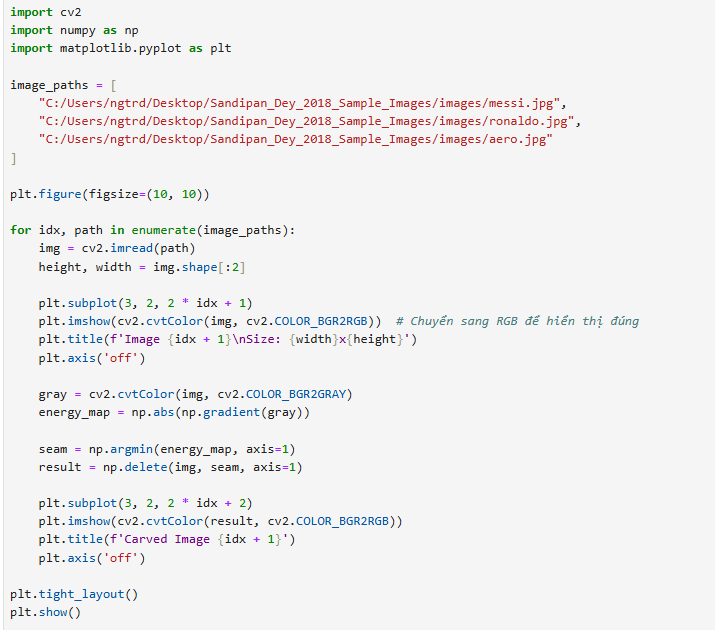
1. **Tải mô hình YOLO**:
   * cv2.dnn.readNet() được sử dụng để tải mô hình YOLO (yolov3.weights và yolov3.cfg), cùng với các nhãn lớp từ tệp coco.names.
2. **Đọc ảnh và tạo blob**:
   * cv2.imread(path) đọc ảnh.
   * cv2.dnn.blobFromImage() chuyển đổi ảnh thành dạng blob, chuẩn bị cho quá trình dự đoán.
3. **Thực hiện dự đoán**:
   * net.forward(layer\_names) thực hiện dự đoán trên ảnh và trả về các phát hiện đối tượng.
4. **Xử lý kết quả**:
   * Với mỗi đối tượng phát hiện, confidence được so sánh với ngưỡng (0.5) để chỉ giữ các đối tượng có độ tin cậy cao.
   * cv2.rectangle() vẽ hình chữ nhật quanh đối tượng và cv2.putText() thêm nhãn cho đối tượng.
5. **Hiển thị ảnh**:
   * cv2.imshow() hiển thị kết quả cho từng ảnh
6. **Chạy cho 3 ảnh**:
   * Mã sẽ lặp qua 3 ảnh và thực hiện phát hiện đối tượng cho từng ảnh.

**Kết quả:**



### **Chương 12: Additional Problems in Image Processing**

**Thuật toán**: Seam Carving - Thay đổi kích thước ảnh thông minh



**Giải thích mã nguồn**:

1. **Đọc và xử lý ảnh**:
   * cv2.imread() đọc ảnh và img.shape[:2] trả về kích thước của ảnh (chiều cao và chiều rộng).
   * cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) chuyển ảnh từ **BGR** (định dạng mặc định của OpenCV) sang **RGB** để hiển thị đúng trong matplotlib.
2. **Hiển thị ảnh và kích thước**:
   * Sử dụng plt.subplot(3, 2, 2 \* idx + 1) để xếp ảnh thành 3 hàng và 2 cột. Cột 1 sẽ chứa ảnh gốc và thông tin kích thước của ảnh (chiều rộng và chiều cao).
   * plt.title(f'Image {idx + 1}\nSize: {width}x{height}') hiển thị thông tin kích thước ảnh.
3. **Seam Carving (tùy chọn)**:
   * Đoạn mã sau chuyển ảnh sang **grayscale** và tính toán **energy map** bằng cách sử dụng np.gradient().
   * Sau đó, np.argmin() tìm đường seam có năng lượng thấp nhất và np.delete() loại bỏ seam khỏi ảnh.
   * Kết quả ảnh sau **Seam Carving** sẽ được hiển thị trong cột thứ 2.
4. **Hiển thị**:
   * plt.tight\_layout() đảm bảo các hình ảnh không bị chồng chéo nhau khi hiển thị.

**Kết quả:**

