**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

A red and yellow logo with a star

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**HỌC PHẦN: LẬP TRÌNH VỚI PYTHON**

**ĐỀ TÀI: BÀI TOÁN PHÂN BIỆT HÌNH ẢNH CHÂN DUNG ĐƯỢC TẠO BỞI AI HAY NGƯỜI THẬT SỬ DỤNG CNN**

Nhóm sinh viên thực hiện: Nhóm: 15

Lớp: D22-114

Nhóm sinh viên thực hiện :

B22DCCN896 Đoàn Quốc Việt

B22DCCN548 Văn Nhật Minh

B22DCCN812 Lê Quang Thắng

B22DCCN836 Nguyễn Quý Thọ

Giảng viên hướng dẫn: ThS. Vũ Minh Mạnh

**HÀ NỘI 2024**

**PHÂN CÔNG NHIỆM VỤ NHÓM THỰC HIỆN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TT** | **Công việc / Nhiệm vụ** | **SV thực hiện** | **Thời hạn hoàn thành** | **Mức đóng góp** |
| 1 | Tìm hiểu chung về mạng neuron và CNN | Cả nhóm | 20/10/2024 | 10% |
| 2 | Phân chia công việc và thuyết trình | Đoàn Quốc Việt |  | 5% |
| 3 | Thu thập dữ liệu ảnh (AI-generated & Real portraits) | Nguyễn Quý Thọ | 30/10/2024 | 10% |
| 4 | Xây dựng mô hình CNN và viết code Python | Đoàn Quốc Việt | 05/11/2024 | 25% |
| 5 | Huấn luyện mô hình, tối ưu hoá | Đoàn Quốc Việt | 08/11/2024 | 15% |
| 6 | Phân tích model thu được. viết báo cáo, trình bày kết quả | Lê Quang Thắng  Văn Nhật Minh | 15/11/2024 | 20%  (Thắng 70)  (Minh 30) |
| 7 | Lập trình GUI cơ bản để ứng dụng và model đã huấn luyện | Lê Quang Thắng  Văn Nhật Minh  Nguyễn Quý Thọ | 15/11/2024 | 10%  (Thắng 50)  (Minh 25)  (Thọ 25) |
| 8 | Làm slide thuyết trình | Lê Quang Thắng | 19/11/2024 | 5% |

**NHÓM THỰC HIỆN TỰ ĐÁNH GIÁ**

|  |  |  |
| --- | --- | --- |
| **TT** | **SV thực hiện** | **Thái độ tham gia** |
| 1 | Đoàn Quốc Việt | 4 |
| 2 | Văn Nhật Minh | 3 |
| 3 | Lê Quang Thắng | 3 |
| 4 | Nguyễn Quý Thọ | 2 |

MỤC LỤC

[**MỤC LỤC** 3](#_Toc182613398)

[**DANH MỤC CÁC HÌNH VẼ** 4](#_Toc182613399)

[**DANH MỤC CÁC TỪ VIẾT TẮT** 5](#_Toc182613400)

[**MỞ ĐẦU** 6](#_Toc182613401)

[**LỜI CẢM ƠN** 7](#_Toc182613402)

[**CHƯƠNG 1. GIỚI THIỆU VỀ MẠNG NƠ-RON TÍCH CHẬP (CNN)** 8](#_Toc182613403)

[1.1 Khái niệm mạng nơ-ron: 8](#_Toc182613404)

[1.2 Mạng nơ-ron tích chập (CNN): 8](#_Toc182613405)

[1.2.1 Khái quát: 8](#_Toc182613406)

[1.2.2 Các thành phần trong một mạng tích chập CNN: 9](#_Toc182613407)

[**CHƯƠNG 2. XÂY DỰNG ỨNG DỤNG** 16](#_Toc182613408)

[2.1 Thư viện, các API cần thiết và tài nguyên hỗ trợ: 16](#_Toc182613409)

[2.2 Khởi tạo AI: 17](#_Toc182613410)

[2.2.1 Xây dựng: 17](#_Toc182613411)

[2.2.2 Đánh giá: 31](#_Toc182613412)

[2.3 Khởi tạo giao diện: 32](#_Toc182613413)

[2.3.1 Xây dựng: 32](#_Toc182613414)

[2.3.2 Hiển thị giao diện GUI: 38](#_Toc182613415)

[**CHƯƠNG 3. KẾT LUẬN** 39](#_Toc182613416)

[3.1 Các kết quả đạt được: 39](#_Toc182613417)

[3.2 Hướng phát triển dự định: 39](#_Toc182613418)

[**TÀI LIỆU THAM KHẢO** 40](#_Toc182613419)

# DANH MỤC CÁC HÌNH VẼ

[*Hình 1- Hình ảnh mảng ma trận RGB 6x6x3* 9](#_Toc182612612)

[*Hình 2 - Toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.* 9](#_Toc182612613)

[*Hình 3 - Minh họa các bộ lọc (kernel) trích suất ra đặc trưng của ảnh.* 10](#_Toc182612614)

[*Hình 4 – Minh hoạ lớp tích chập* 11](#_Toc182612615)

[*Hình 5 – Ma trận lớp tích chập* 11](#_Toc182612616)

[*Hình 6 – Minh hoạ lọc trong ma trận tích chập* 12](#_Toc182612617)

[*Hình 7 - Hình ảnh tích chập khác nhau sau khi áp dụng các Kernel (bộ lọc) khác nhau* 12](#_Toc182612618)

[*Hình 8 - Lớp gộp* 13](#_Toc182612619)

[*Hình 9 – Bước nhảy* 14](#_Toc182612620)

[*Hình 10 – Hàm ReLU* 15](#_Toc182612621)

[*Hình 11 - Biểu đồ đánh giá kết quả huấn luyện.* 31](#_Toc182612622)

[*Hình 12 – Giao diện ứng dụng.* 38](#_Toc182612623)

[*Hình 13 – Thử nghiệm với ảnh AI.* 38](#_Toc182612624)

[*Hình 14 – Thử nghiệm với ảnh người.* 39](#_Toc182612625)

**DANH MỤC CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **Từ  viết tắt** | **Thuật ngữ tiếng Anh/Giải thích** | **Thuật ngữ tiếng Việt/Giải thích** |
| API | Application Programming Interface | Giao diện lập trình ứng dụng |
| AI | Artificial Intelligence | Trí tuệ nhân tạo |
| CNN | Convolutional Neural Network | Mạng nơ-ron tích chập |
| ReLU | Rectified Linear Unit | Hàm kích hoạt tuyến tính chỉnh lưu |
| wb | Write Binary | Ghi dữ liệu nhị phân vào file |
| GUI | Graphical User Interface | Giao diện người dùng đồ họa |

MỞ ĐẦU

Hiện nay, với sự phát triển nhanh chóng của công nghệ trí tuệ nhân tạo, đặc biệt là trong lĩnh vực **AI tạo ảnh chân dung** (AI-generated portraits), việc phân biệt giữa ảnh chân dung thực và ảnh do AI tạo ra trở nên vô cùng quan trọng. Công nghệ AI có thể tạo ra những hình ảnh chân thực đến mức gần như không thể phân biệt được bằng mắt thường, gây ra nhiều **nguy cơ tiềm ẩn** về bảo mật, quyền riêng tư và thông tin sai lệch. Các hình ảnh giả mạo có thể bị lợi dụng trong việc **tạo tin giả, lừa đảo, mạo danh** hoặc thậm chí xâm phạm đến danh tính của người khác mà không được kiểm soát chặt chẽ.

Do đó, việc phát triển một **Ứng dụng phân biệt ảnh chân dung AI và ảnh thực** là cần thiết nhằm hỗ trợ người dùng, các doanh nghiệp, và các tổ chức trong việc nhận diện và ngăn chặn các hành vi gian lận. Ứng dụng đơn giản này sẽ giúp người dùng có thể xác định nhanh chóng loại của một bức ảnh, đảm bảo độ tin cậy của thông tin cũng như bảo vệ quyền lợi cá nhân của họ. Ngoài ra, trong bối cảnh công nghệ AI ngày càng tinh vi, việc sở hữu công cụ phân biệt ảnh thực và ảnh do AI tạo ra còn giúp các cơ quan pháp lý dễ dàng phát hiện và xử lý các vi phạm, đóng góp vào việc xây dựng một **môi trường kỹ thuật số an toàn và đáng tin cậy** hơn.

Báo cáo này là kết quả của quá trình tìm hiểu, học hỏi và phát triển để triển khai đề tài “Phân biệt hình ảnh chân dung do AI tạo ra hay con người”. Đề tài này đại diện cho nỗ lực của nhóm 15 trong việc tạo ra một ứng dụng phân biệt hình ảnh dễ sử dụng và hiệu quả, với sự tập trung vào việc phân biệt hình ảnh chính xác cùng với đưa ra độ tin cậy về hình ảnh để giúp đỡ người dùng trong nhiều mục đích.

Báo cáo sẽ xoay quanh quá trình thu thập thông tin của từng phần cấu thành để xây dựng nên ứng dụng hoàn trình mà nhóm 15 đã thưc hiện. Chúng em hi vọng rằng báo cáo này sẽ truyền đạt được sự nỗ lực và đóng góp của cả nhóm trong việc xây dựng ứng dụng phân biệt hình ảnh đơn giản này và giúp mọi người hiểu rõ về quá trình phát triển một ứng dụng đơn giản có sử dụng AI.

# LỜI CẢM ƠN

Lời đầu tiên chúng em xin gửi lời cảm ơn chân thành nhất đến thầy Vũ Ninh Mạnh – giảng viên môn Lập trình với Python đã tận tình hướng dẫn và cung cấp nguồn tri thức hữu ích giúp chúng em trong suốt quá trình làm báo cáo này.

Để hoàn thành tốt bài báo cáo này, nhóm chúng em đã nỗ lực hết sức. Nhưng vì kiến thức của cả nhóm còn hạn chế, trong suốt quá trình thực hiện sẽ có những thiếu sót không mong muốn. Chúng em rất mong nhận được sự chỉ bảo, đóng góp của thầy và các bạn để có bổ sung và nâng cao kiến thức của các thành viên trong nhóm, hoàn thành tốt hơn những bài báo cáo sau này.

Cuối cùng, chúng em xin kính chúc thầy có thật nhiều sức khỏe, niềm vui và đạt được nhiều thành công tốt đẹp trong công việc.

Chúng em xin chân thành cảm ơn!

1. gIỚI THIỆU VỀ MẠNG NƠ-RON TÍCH CHẬP (CNN)
   1. Khái niệm mạng nơ-ron:

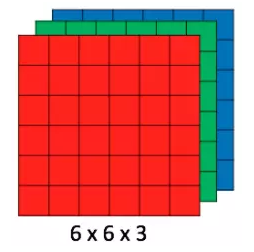
Mạng nơ-ron là một phương thức trong lĩnh vực trí tuệ nhân tạo (AI) được sử dụng để dạy máy tính xử lý dữ liệu theo cách mô phỏng bộ não con người Mạng nơ-ron có thể giúp máy tính đưa ra các quyết định thông minh chỉ với sự hỗ trợ hạn chế của con người. Mạng nơ-ron có thể hiểu rõ dữ liệu phi cấu trúc và đưa ra các nhận xét chung mà không cần đào tạo cụ thể.

**Cấu trúc cơ bản** của mạng nơ-ron bao gồm các lớp:

* **Lớp đầu vào** (Input Layer): Nhận dữ liệu đầu vào.
* **Lớp ẩn** (Hidden Layers): Là nơi mà mạng học và xử lý thông tin.
* **Lớp đầu ra** (Output Layer): Cung cấp kết quả hoặc dự đoán của mạng.
* **Liên kết (Weights) và Biases:**
  + **Weights (trọng số):** Điều chỉnh mức độ ảnh hưởng của đầu vào đến đầu ra tại mỗi kết nối.
  + **Biases (độ chệch):** Giúp dịch chuyển giá trị đầu ra để tối ưu hóa khả năng học của mạng.
  1. Mạng nơ-ron tích chập (CNN):
     1. Khái quát:

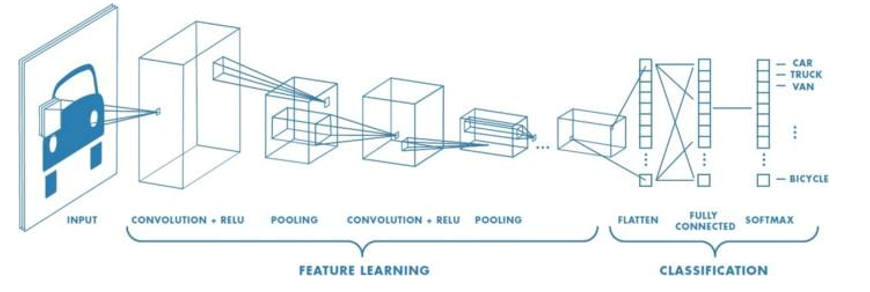
Mạng nơ-ron tích chập (CNN) là một loại mạng nơ-ron đặc biệt, được thiết kế để xử lý dữ liệu có cấu trúc dạng lưới, đặc biệt là hình ảnh. CNN là một phát triển mở rộng của mạng nơ-ron truyền thống nhưng với các thành phần và cấu trúc khác biệt, giúp nó có thể học các đặc trưng không gian trong dữ liệu như hình ảnh.

CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định. Máy tính coi hình ảnh đầu vào là 1 mảng pixel và nó phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy H x W x D (H: Chiều cao, W: Chiều rộng, D: Độ dày).



*Hình 1- Hình ảnh mảng ma trận RGB 6x6x3*

Về kỹ thuật, mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernals), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1.

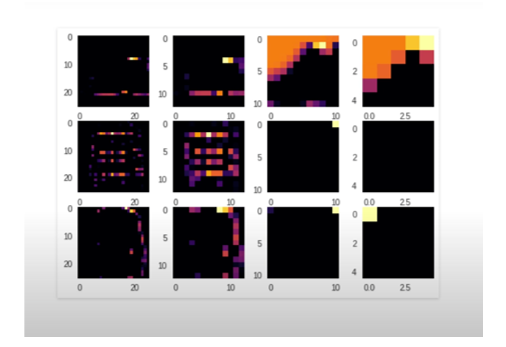


Hình 2 - Toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.

* + 1. Các thành phần trong một mạng tích chập CNN:
       1. Các bộ lọc:

Ở giai đoạn đầu, các bộ lọc (Kernel) trong các lớp tích chập (Convolutional Layer) sẽ được khởi tạo với các giá trị ngẫu nhiên. Mỗi bộ lọc này là một ma trận nhỏ (thường là 3x3 hoặc 5x5) chứa các trọng số ngẫu nhiên khi cho ảnh vào sẽ có 1 số giá trị đầu ra.

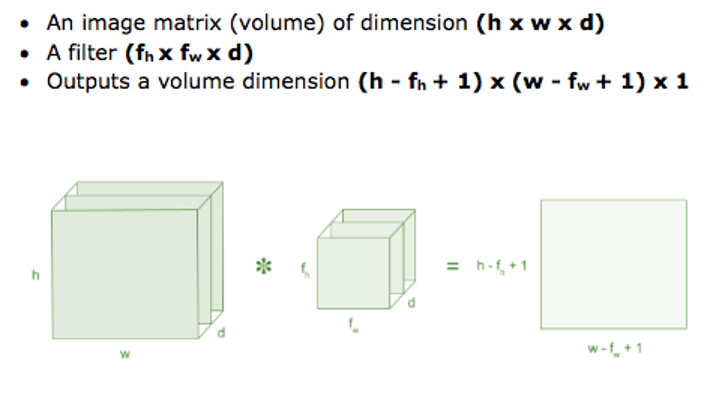
Sau đó ta sẽ cho tiếp các giá trị đó qua các lớp tiếp theo trong mạng nơ ron ,Cuối cùng mạng nơ ron đưa ra dự đoán cho nhãn bức ảnh, máy tính sẽ so sánh với nhãn trong tập huấn luyện và điều chỉnh tham số bộ lọc để kết quả nhãn gần hơn với nhãn đúng -> học được cách gán nhãn tốt nhất - > triết suất được đặc trưng.



Hình 3 - Minh họa các bộ lọc (kernel) trích suất ra đặc trưng của ảnh.

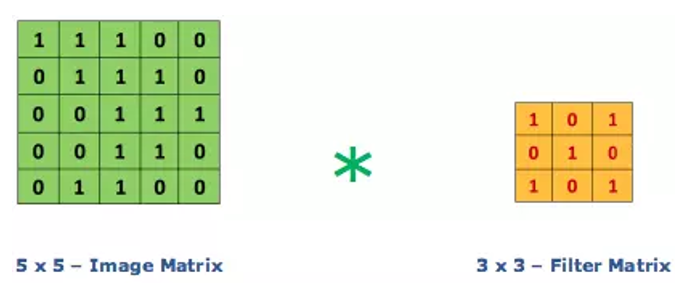
* + - 1. Lớp tích chập - Convolution Layer:

Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vương nhỏ của dữ liệu đầu vào. Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân.



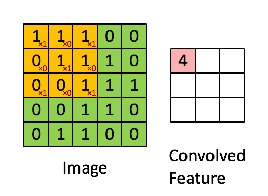
Hình 4 – Minh hoạ lớp tích chập

Sau đó, lớp tích chập của ma trận hình ảnh 5 x 5 nhân với ma trận bộ lọc 3 x 3 gọi là 'Feature Map' như hình bên dưới.



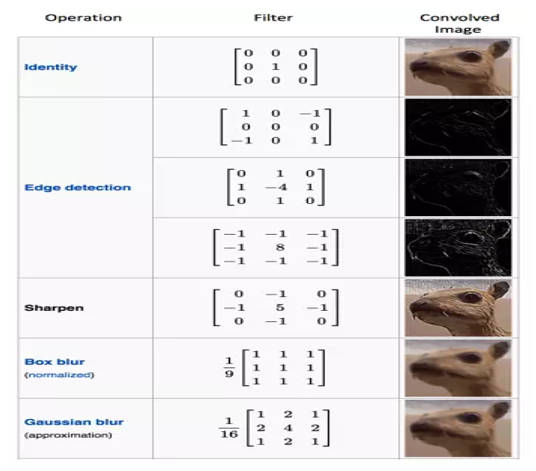
Hình 5 – Ma trận lớp tích chập

Sau đó, lớp tích chập của ma trận hình ảnh 5 x 5 nhân với ma trận bộ lọc 3 x 3 gọi là 'Feature Map' như hình bên dưới.



Hình 6 – Minh hoạ lọc trong ma trận tích chập

Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc.

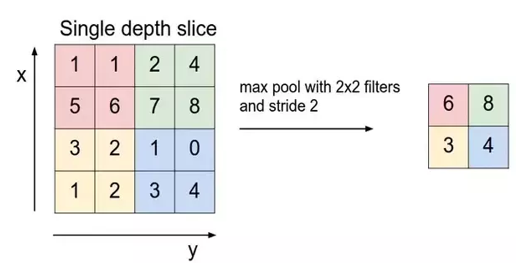


*Hình 7 - Hình ảnh tích chập khác nhau sau khi áp dụng các Kernel (bộ lọc) khác nhau*

* + - 1. Lớp gộp - Pooling Layer:

Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng.

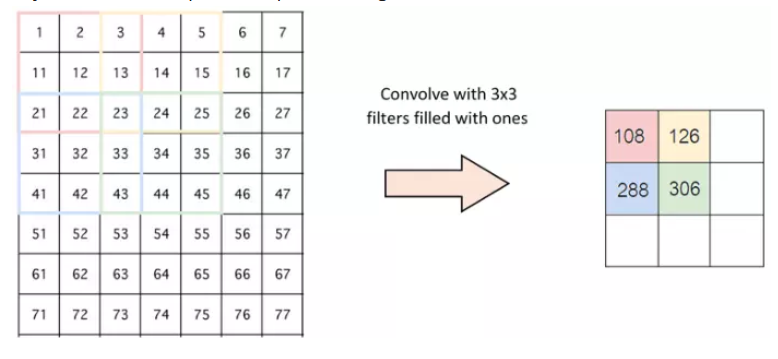
Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling.



Hình 8 - Lớp gộp

* + - 1. Bước nhảy – Stride:

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.



Hình 9 – Bước nhảy

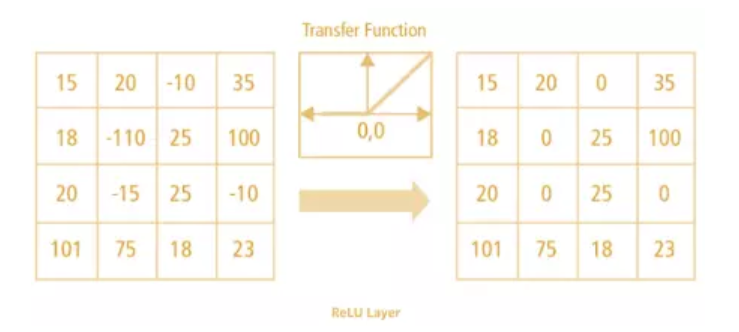
* + - 1. Đường viền – Padding:

Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:

* Chèn các số 0 và 4 đường biên của hình ảnh (padding).
* Cắt bớt hình ảnh tạo những điểm không phù hợp với bộ lọc.
  + - 1. Hàm ReLU:

ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là: ƒ (x) = max (0, x).

ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.



Hình 10 – Hàm ReLU

* + - 1. Hàm Flatten( ):

Hàm Flatten chủ yếu được sử dụng khi bạn chuyển đầu ra từ các lớp tích chập và pooling (có nhiều chiều) sang một dạng mà có thể được đưa vào các lớp fullyconnected (lớp dày đặc) để thực hiện các tác vụ như phân loại hoặc hồi quy.

Các lớp fully connected chỉ nhận vào một vector 1 chiều làm đầu vào, vì vậy Flatten giúp chuyển đổi dữ liệu trước khi đưa vào lớp này.

* + - 1. Lớp Fully Connected (FC):

Lớp Fully Connected (hoặc Dense Layer) là một lớp nơ-ron truyền thống trong mạng nơ-ron, nơi mỗi nơ-ron trong lớp này kết nối đến tất cả các nơ-ron trong lớp trước. Đặc điểm của lớp này là nó chỉ nhận vào một vector 1 chiều làm đầu vào, và không chấp nhận ma trận đa chiều.

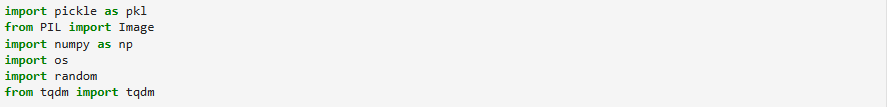
Khi bạn muốn đưa đầu ra của các lớp tích chập và pooling vào các lớp fully connected, bạn cần chuyển đổi dữ liệu từ một ma trận đa chiều thành một vector 1 chiều. Đây chính là lúc bạn cần sử dụng hàm Flatten.

1. Xây dựng ứng dụng
   1. Thư viện, các API cần thiết và tài nguyên hỗ trợ:

Trong dự án có sử dụng một số thư viện sau:

* **selenium**: Đây là thư viện chính được dùng để tự động hóa trình duyệt web. Nó cung cấp các API để tương tác với các phần tử trên trang web.
* **TensorFlow và Keras**:
  + **TensorFlow**: Là thư viện chính để xây dựng và huấn luyện mô hình máy học, đặc biệt cho các mô hình mạng nơ-ron. Các API của TensorFlow được dùng để xây dựng (như tf.keras.Sequential, tf.keras.layers), biên dịch (model.compile), huấn luyện (model.fit), và lưu mô hình (model.save).
  + **Keras** (được tích hợp trong TensorFlow): Sử dụng các lớp và hàm Keras để tạo mô hình (như layers.Conv2D, layers.MaxPooling2D, layers.Dense) và tối ưu hóa mô hình (tf.keras.optimizers.Adam).
* **NumPy**: Đoạn mã sử dụng NumPy để kiểm tra cấu trúc của dữ liệu hình ảnh và nhãn (thông qua isinstance(image\_data, np.ndarray)), đảm bảo dữ liệu được lưu trong định dạng mảng Numpy để TensorFlow xử lý.
* **Pickle**: Sử dụng API của pickle (pkl.load, pkl.dump) để tải dữ liệu từ các file .pickle và lưu các kết quả huấn luyện.
* **tqdm**: Sử dụng tqdm để hiển thị thanh tiến trình khi chạy qua các epoch, giúp người dùng dễ dàng theo dõi quá trình huấn luyện mô hình.
* **OS**: Dùng os.listdir để lấy danh sách các file trong thư mục train\_batches, và os.path.splitext để kiểm tra phần mở rộng của các file nhằm đảm bảo rằng chỉ các file .pickle hợp lệ được sử dụng.
* **Garbage Collection (gc)**: Dùng gc.collect() để dọn dẹp bộ nhớ, đặc biệt hữu ích khi huấn luyện mô hình trong các vòng lặp lớn, giúp tránh tình trạng tràn bộ nhớ.
* **tkinter**: Là thư viện GUI chuẩn của Python để xây dựng giao diện người dùng (GUI).
* **filedialog và messagebox**: Là các module con trong tkinter, dùng để mở hộp thoại chọn tệp và hiển thị thông báo trong GUI. Cũng là một phần của thư viện tkinter,.
* **PIL (Pillow)**: Đây là một thư viện xử lý hình ảnh trong Python sử dụng để mở, chỉnh sửa và hiển thị ảnh.
  1. Khởi tạo AI:
     1. Xây dựng:
        1. Tiền xử lý ảnh:

a. Nhập các thư viện cần thiết:



* **pickle**: Được sử dụng để lưu trữ dữ liệu dưới dạng tệp .pickle.
* **PIL (Python Imaging Library)**: Được dùng để xử lý ảnh như mở ảnh, thay đổi kích thước và chuyển đổi định dạng.
* **numpy**: Thư viện cho các thao tác trên mảng, giúp dễ dàng thao tác dữ liệu ảnh.
* **os**: Hỗ trợ các thao tác liên quan đến hệ thống tệp.
* **webp**: Được sử dụng để làm việc với định dạng ảnh WebP.
* **random**: Dùng để chọn ngẫu nhiên các tệp trong tập dữ liệu.
* **tqdm**: Tạo thanh tiến trình hiển thị khi chương trình xử lý dữ liệu.

b. Hàm **get\_random\_samples( )**:



* Khai báo hàm với tên **get\_random\_samples** với hai tham số đầu vào:
  + **file\_list**: danh sách các phần tử (các tệp hoặc đối tượng) mà ta muốn chọn ngẫu nhiên từ đó.
  + **sample\_size**: số lượng mẫu mà ta muốn lấy ngẫu nhiên từ file\_list.
* **random.sample( )**: Sử dụng hàm sample của thư viện random để lấy ngẫu nhiên một danh sách con từ file\_list có kích thước là sample\_size sau đó trả về danh sách con chứa các mẫu đã được chọn.
  + random.sample() đảm bảo rằng các mẫu được chọn là ngẫu nhiên và không bị trùng lặp trong mỗi lần lấy.

c. Hàm **get\_remaining\_samples( )**:

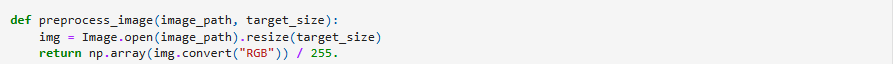


* Hàm **get\_remaining\_samples** nhận vào hai tham số:
  + **file\_list**: Danh sách gốc chứa tất cả các tệp.
  + **selected\_samples**: Danh sách con chứa các tệp đã được chọn từ file\_list.

Lọc các phần tử:

* Sử dụng **list comprehension** để tạo ra danh sách mới chỉ chứa các phần tử từ file\_list mà không có trong selected\_samples:
  + **sample for sample in file\_list**: Duyệt qua từng phần tử sample trong file\_list.
  + **if sample not in selected\_samples**: Kiểm tra xem sample có nằm ngoài selected\_samples không. Nếu sample không nằm trong selected\_samples, nó sẽ được thêm vào danh sách kết quả.

d. Hàm **preprocess\_image( )**:



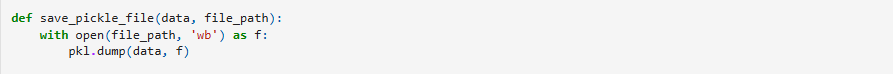
* Hàm **preprocess\_image** được định nghĩa với hai tham số:
  + **image\_path**: Đường dẫn đến file ảnh cần xử lý.
  + **target\_size**: Kích thước mong muốn của ảnh sau khi thay đổi kích thước, thường là một tuple (chiều rộng, chiều cao), ví dụ (256, 256).

Tải và thay đổi kích thước ảnh:

* **Image.open(image\_path)**: Mở file ảnh từ image\_path bằng thư viện PIL (Pillow).
* **.resize(target\_size)**: Thay đổi kích thước ảnh theo kích thước mong muốn (target\_size).

Chuyển sang dạng RGB và chuẩn hoá:

* **img.convert("RGB")**: Chuyển ảnh sang định dạng RGB (đảm bảo rằng ảnh có 3 kênh màu: Đỏ, Xanh lá, Xanh dương).
* **np.array( )**: Chuyển đổi ảnh từ định dạng PIL sang numpy array để dễ dàng xử lý bằng các thao tác trên mảng.
* **/ 255**.: Chia toàn bộ mảng pixel cho 255 để chuẩn hóa giá trị pixel từ khoảng [0, 255] về khoảng [0, 1]. Điều này giúp quá trình huấn luyện mô hình học máy hiệu quả hơn.

e. Hàm **save\_pickle\_file( )**:

* Hàm **save\_pickle\_fil**e nhận vào hai tham số:
  + **data**: Dữ liệu cần lưu trữ.
  + **file\_path**: Đường dẫn tệp nơi dữ liệu sẽ được lưu.

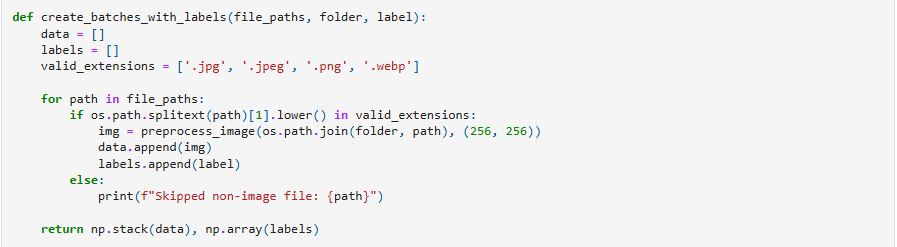
Mở tệp để ghi dữ liệu:

* **open(file\_path, 'wb')**: Hàm open mở tệp ở đường dẫn file\_path với chế độ 'wb':
* Câu lệnh with đảm bảo rằng tệp sẽ được đóng lại tự động khi khối mã kết thúc, ngay cả khi có lỗi xảy ra.

Lưu dữ liệu vào tệp pickle:

* **pkl.dump(data, f)**: Hàm dump của module pickle được sử dụng để lưu dữ liệu vào tệp nhị phân.
  + data: Là dữ liệu cần lưu.
  + f: Là đối tượng file mà chúng ta đã mở trong chế độ ghi.

f. Hàm **create\_batches\_with\_labels( )**:



* Hàm **create\_batches\_with\_labels** với ba tham số:
  + **file\_paths**: Danh sách các đường dẫn hoặc tên tệp hình ảnh cần được xử lý.
  + **folder**: Thư mục chứa các tệp hình ảnh.
  + **label**: Nhãn sẽ được gán cho tất cả các hình ảnh trong danh sách này.

Khởi tạo danh sách và xác định phần mở rộng hợp lệ:

* **data**: Danh sách lưu trữ dữ liệu hình ảnh sau khi chúng được tiền xử lý.
* l**abels**: Danh sách lưu trữ nhãn tương ứng với mỗi hình ảnh trong data.
* **valid\_extensions**: Danh sách chứa các phần mở rộng tệp hợp lệ. Chỉ các tệp hình ảnh có các phần mở rộng này sẽ được xử lý.

Duyệt qua danh sách các đường dẫn tệp: Lặp qua tất cả các tệp trong danh sách file\_paths. Mỗi tệp sẽ được kiểm tra xem có phải là hình ảnh hợp lệ hay không.

Kiểm tra phần mở rộng của tệp:

* **os.path.splitext(path)[1]**: Tách phần mở rộng của tệp từ đường dẫn path.
* **.lower( )**: Đảm bảo rằng phần mở rộng được kiểm tra không phân biệt chữ hoa chữ thường.
* **in valid\_extensions**: Kiểm tra xem phần mở rộng có thuộc danh sách các phần

mở rộng hợp lệ (valid\_extensions) hay không.

Tiền xử lý hình ảnh hợp lệ:

* **os.path.join(folder, path)**: Kết hợp thư mục folder với đường dẫn tệp path để tạo đường dẫn đầy đủ đến tệp hình ảnh.
* **preprocess\_image( )**: Đây là một hàm giả định sẽ tải và xử lý hình ảnh. Trong trường hợp này, hình ảnh sẽ được thay đổi kích thước thành (256, 256).
* **data.append(img)**: Thêm hình ảnh đã xử lý vào danh sách data.
* **labels.append(label)**: Gán nhãn label cho hình ảnh và thêm vào danh sách labels.

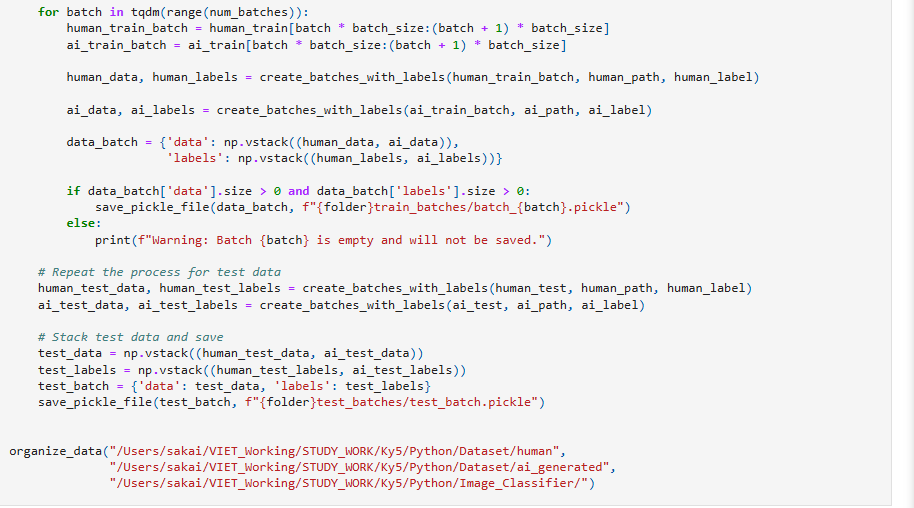
**Xử lý các tệp không phải hình ảnh:** Nếu phần mở rộng của tệp không hợp lệ, hàm sẽ in ra một thông báo để thông báo rằng tệp đó đã bị bỏ qua.

Trả về kết quả:

* **np.stack(data)**: Chuyển danh sách data thành một mảng numpy. Mảng này sẽ chứa các hình ảnh đã được xử lý và thay đổi kích thước.
* **np.array(labels)**: Chuyển danh sách labels thành một mảng numpy, chứa các nhãn tương ứng với từng hình ảnh trong data.
* Hàm trả về cả hai mảng numpy: mảng data chứa hình ảnh và mảng labels chứa nhãn.

g. Hàm chính **organize\_data(** **):**





* hàm **organize\_data()** nhận vào ba tham số:
  + **human\_path**: Đường dẫn tới thư mục chứa ảnh của con người.
  + **ai\_path**: Đường dẫn tới thư mục chứa ảnh AI.
  + **folder**: Đường dẫn tới thư mục chứa các dữ liệu đầu ra.

Khởi tạo các đường dẫn và tạo thư mục cần thiết

* **os.path.exists(folder + 'file\_names/')**: Kiểm tra xem thư mục file\_names/ trong thư mục đích đã tồn tại chưa. Nếu chưa tồn tại, nó sẽ tạo mới thư mục này.
* Thực hiện tương tự cho các thư mục train\_batches/ và test\_batches/.

Lấy danh sách ảnh từ thư mục và chia thành bộ huấn luyện và kiểm tra:

* **os.listdir(human\_path)**: Lấy danh sách tất cả các tệp trong thư mục human\_path (chứa ảnh của con người).
* **get\_random\_samples(human\_files, 8400)**: Hàm này sẽ chọn ngẫu nhiên 8400 ảnh từ danh sách các ảnh người làm mẫu huấn luyện.
* **get\_remaining\_samples(human\_files, human\_train)**: Hàm này sẽ lấy phần còn lại của danh sách ảnh người sau khi đã chọn 8400 ảnh cho bộ huấn luyện, làm dữ liệu kiểm tra.
* Tương tự cho ảnh AI, mã sẽ lấy danh sách ảnh từ thư mục ai\_path, chọn ngẫu nhiên 8400 ảnh cho bộ huấn luyện và phần còn lại cho bộ kiểm tra.

Lưu tên các mẫu huấn luyện và kiểm tra vào file pickle:

* **save\_pickle\_file([human\_train, ai\_train], folder + 'file\_names/train.pickle')**: Lưu danh sách tên các mẫu huấn luyện (bao gồm ảnh người và ảnh AI) vào tệp train.pickle.
* **save\_pickle\_file([human\_test, ai\_test], folder + 'file\_names/test.pickle')**: Lưu danh sách tên các mẫu kiểm tra vào tệp test.pickle.

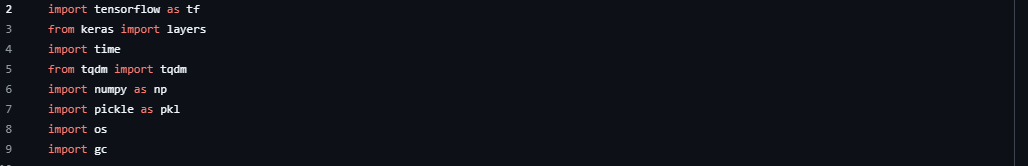
Tạo các batch huấn luyện và lưu vào file pickle:

* **num\_batches = 14**: Xác định số lượng batch (14 batch).
* **batch\_size = 600**: Mỗi batch chứa 600 ảnh.
* **human\_train[batch \* batch\_size:(batch + 1) \* batch\_size]**: Lấy dữ liệu huấn luyện của người trong khoảng từ (batch \* batch\_size) đến ((batch + 1) \* batch\_size).
* **ai\_train[batch \* batch\_size:(batch + 1) \* batch\_size]**: Lấy dữ liệu huấn luyện của AI tương tự.
* **create\_batches\_with\_labels()**: Hàm này lấy các tệp ảnh trong mỗi batch, gán nhãn tương ứng cho chúng (nhãn 0, 1, 0 cho ảnh người và 1, 0, 0 cho ảnh AI), và chuẩn bị dữ liệu cho mô hình.
* **np.vstack()**: Dùng để gộp dữ liệu từ ảnh người và AI vào một mảng.
* **save\_pickle\_file(data\_batch, f"{folder}train\_batches/batch\_{batch}.pickle")**: Lưu batch dữ liệu vào tệp pickle.

Tạo và lưu batch kiểm tra:

* **create\_batches\_with\_labels()**: Dùng để xử lý dữ liệu kiểm tra, gán nhãn cho các ảnh người và AI.
* **np.vstack()**: Gộp dữ liệu và nhãn của ảnh người và AI lại thành một mảng duy nhất.
* **save\_pickle\_file(test\_batch, f"{folder}test\_batches/test\_batch.pickle")**: Lưu dữ liệu kiểm tra vào tệp pickle test\_batch.pickle.
  + - 1. Tạo model huấn luyện:

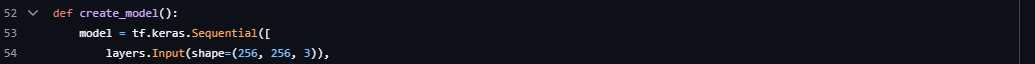
a. Nhập các thư viện cần thiết:



* **tensorflow**: thư viện mã nguồn mở phổ biến để xây dựng và huấn luyện các mô hình học sâu.
* **keras**: API cấp cao để xây dựng và huấn luyện các mô hình học sâu, được tích hợp vào TensorFlow.
* **tqdm**: thư viện hiển thị thanh tiến trình khi chạy các vòng lặp.
* **numpy**: thư viện chính cho các phép toán trên mảng và các thao tác tính toán trong Python.
* **pickle**: thư viện trong Python dùng để lưu trữ và nạp dữ liệu thành các file nhị phân.
* **os**: thư viện tiêu chuẩn của Python để tương tác với hệ điều hành.
* **gc**: thư viện quản lý bộ nhớ trong Python, giúp thu gom bộ nhớ không còn được sử dụng.

b. Hàm **create\_model**:

Hàm **create\_model** được để xây dựng một mô hình mạng nơ-ron tích chập (CNN) dành cho việc phân loại hình ảnh thành ba lớp khác nhau.



* **tf.keras.Sequential**: Đây là API dùng để tạo một mô hình "tuần tự", trong đó các lớp được thêm vào theo thứ tự. Mỗi lớp truyền dữ liệu đầu ra của mình cho lớp kế tiếp.
* **layers.Input(shape=(256, 256, 3))**: Xác định kích thước đầu vào của mô hình là hình ảnh 256 x 256 pixel với 3 kênh màu (RGB).



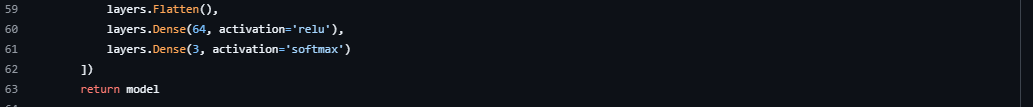
* **layers.Conv2D(32, (3, 3), activation='relu')**: Đây là lớp tích chập với:
  + **32 bộ lọc (filters)**: Số lượng ma trận được dùng để trích xuất các đặc trưng từ ảnh đầu vào.
  + **(3, 3) kernel size**: Kích thước của bộ lọc là 3x3.
  + **activation='relu'**: Hàm kích hoạt ReLU giúp đưa đầu ra của lớp này về dạng phi tuyến.



* **layers.MaxPooling2D((2, 2))**: Lớp lấy mẫu tối đa với:
  + **(2, 2) pool size**: Mỗi lần lấy mẫu, khối 2x2 của đầu ra từ lớp trước sẽ được giảm thành một giá trị lớn nhất, giúp giảm kích thước không gian của đặc trưng.

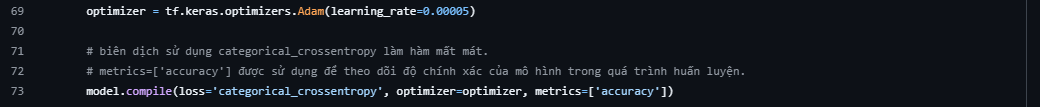
****

* **layers.Conv2D và layers.MaxPooling2D** lần hai:
  + Tiếp tục thêm một cặp lớp tích chập và lớp lấy mẫu nữa với các thông số tương tự. Cặp lớp này giúp trích xuất thêm đặc trưng từ ảnh và làm giảm thêm kích thước của dữ liệu đầu vào, tối ưu hóa việc tính toán.

****

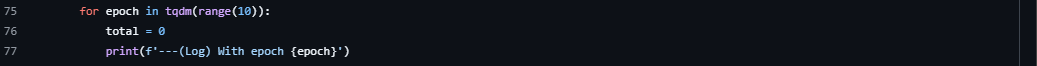
* **layers.Flatten()**: Chuyển đổi đầu ra từ các lớp tích chập thành một vector một chiều để kết nối với các lớp fully connected (lớp dày) tiếp theo.
* **layers.Dense(64, activation='relu')**: Đây là lớp fully connected với:
  + **64 nơ-ron**: Số lượng nơ-ron trong lớp này.
  + **activation='relu'**: Hàm kích hoạt ReLU giúp mạng học được các đặc trưng phức tạp hơn từ các đặc trưng đã trích xuất.
* **layers.Dense(2, activation='softmax')**: Đây là lớp đầu ra với:
  + **2 nơ-ron**: Mỗi nơ-ron đại diện cho một lớp phân loại.
  + **activation='softmax'**: Hàm kích hoạt softmax giúp chuyển đầu ra thành xác suất, dùng trong các bài toán phân loại đa lớp.

c. Hàm **train\_model**:



Khởi tạo và biên dịch mô hình:

* **optimizer = tf.keras.optimizers.Adam(learning\_rate=0.00005)**: Khởi tạo một optimizer với learning rate (tốc độ học) là 0.00005
* **model.compile(...)**:
  + **loss='categorical\_crossentropy'**: Hàm categorical\_crossentropy tính toán sự khác biệt giữa xác suất dự đoán của mô hình và nhãn thật.
  + **optimizer=optimizer**: Sử dụng optimizer Adam đã được khởi tạo ở trên.
  + **metrics=['accuracy']**: Đánh giá độ chính xác (accuracy) của mô hình trong quá trình huấn luyện và kiểm tra.



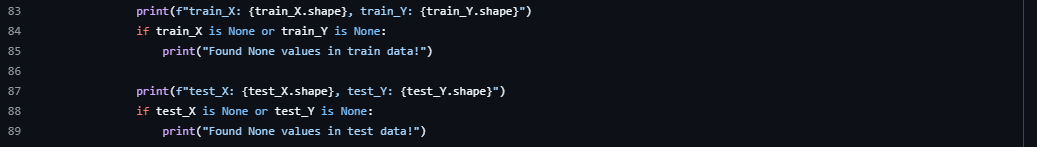
Tạo vòng lặp để huấn luyện:

* **for epoch in tqdm(range(10))**: Chạy vòng lặp huấn luyện qua 10 epoch.
* **total = 0**: Khởi tạo biến total.



Tạo vòng lặp qua các batch của dữ liệu huấn luyện:

* **for batch in train\_data**: Duyệt qua các batch dữ liệu trong train\_data.
* **train\_X, train\_Y**: Dữ liệu huấn luyện (train\_X) và nhãn của dữ liệu huấn luyện (train\_Y).
* **test\_X, test\_Y**: Dữ liệu kiểm tra (test\_X) và nhãn của dữ liệu kiểm tra (test\_Y).



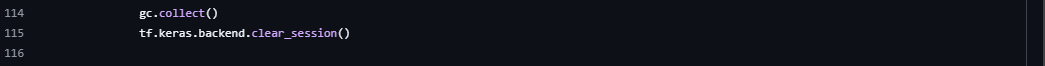
Kiểm tra xem dữ liệu huấn luyện và kiểm tra có bị thiếu giá trị (None) không. Nếu có giá trị None, chương trình sẽ in thông báo cảnh báo.



Huấn luyện cho mô hình:

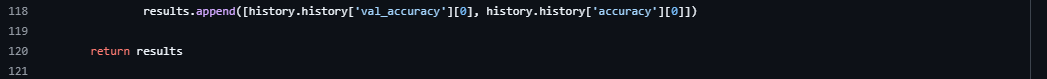
* **model.fit(...)**:
  + **train\_X, train\_Y**: Dữ liệu và nhãn huấn luyện.
  + **epochs=1**: Đặt số epoch trong một lần gọi fit() là 1, có nghĩa là mô hình sẽ được huấn luyện qua toàn bộ dữ liệu huấn luyện một lần.
  + **validation\_data=(test\_X, test\_Y)**: Dữ liệu kiểm tra sẽ được sử dụng để đánh giá hiệu suất của mô hình sau mỗi epoch.

**fit()** trả về một đối tượng history chứa thông tin về quá trình huấn luyện, bao gồm độ chính xác và mất mát cho cả dữ liệu huấn luyện và kiểm tra.



Dọn dẹp bộ nhớ:

* **gc.collect()**: Gọi hàm thu gom bộ nhớ trong Python để giải phóng bộ nhớ không sử dụng.
* **tf.keras.backend.clear\_session()**: Giải phóng tài nguyên của phiên Keras hiện tại, tránh việc sử dụng lại mô hình mà không giải phóng bộ nhớ.



Lưu kết quả:

* **history.history['val\_accuracy'][0]**: Lấy độ chính xác trên dữ liệu kiểm tra (validation accuracy) trong epoch đầu tiên.
* Kết quả được lưu vào danh sách results dưới dạng một cặp giá trị gồm độ chính xác của mô hình trên dữ liệu kiểm tra và huấn luyện.

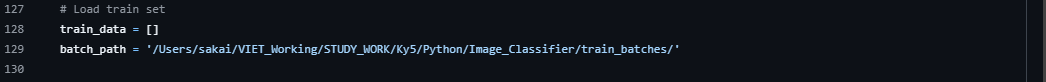
Hàm trả về danh sách results, chứa thông tin về độ chính xác (accuracy) trên dữ liệu huấn luyện và kiểm tra trong mỗi epoch.

d. Tải dữ liệu từ đường dẫn:



Đọc dữ liệu kiểm tra từ file pickle test\_batch.pickle và lưu vào biến test\_data.

e. Load train set:



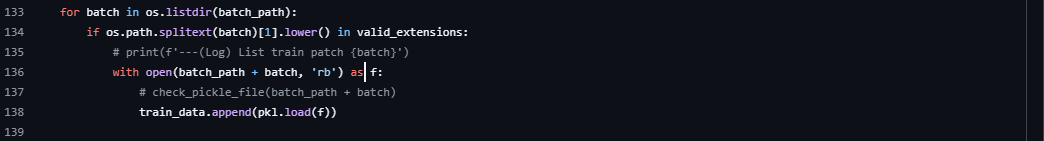
Khởi tạo biến train\_data và định nghĩa đường dẫn:

* **train\_data = []**: Đây là một danh sách trống, nơi sẽ lưu trữ tất cả dữ liệu huấn luyện được tải vào.
* Biến này chứa đường dẫn đến thư mục chứa các tệp dữ liệu huấn luyện.



Định nghĩa phần mở rộng hợp lệ cho tệp pickle:

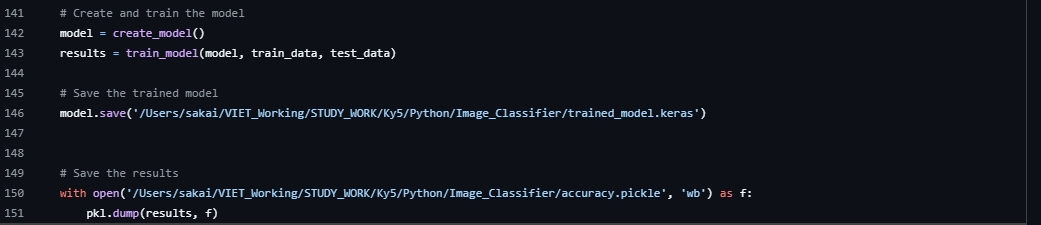
* **valid\_extensions = ['.pickle']**: Biến chứa danh sách phần mở rộng tệp hợp lệ mà bạn muốn xử lý trong thư mục batch\_path. Trong trường hợp này, chỉ các tệp có phần mở rộng .pickle mới được chọn.



Duyệt qua các tệp trong thư mục và tải dữ liệu:

* **for batch in os.listdir(batch\_path):**:
  + **os.listdir(batch\_path)**: Hàm trả về danh sách tất cả các tệp và thư mục có trong thư mục batch\_path.
* **if os.path.splitext(batch)[1].lower() in valid\_extensions:**:
  + **os.path.splitext(batch)**: Hàm tách một tên tệp thành hai phần: phần tên tệp và phần mở rộng (extension).
  + **[1]**: Truy cập vào phần mở rộng của tệp, trong ví dụ này là .pickle.
  + **in valid\_extensions**: Kiểm tra xem phần mở rộng có nằm trong danh sách các phần mở rộng hợp lệ hay không.
* **with open(batch\_path + batch, 'rb') as f:**:
  + Nếu tệp có phần mở rộng hợp lệ, hàm này sẽ mở tệp theo chế độ.
  + **batch\_path + batch** là đường dẫn đầy đủ đến tệp.
* **train\_data.append(pkl.load(f))**:
  + **pkl.load(f)**: Hàm load của module pickle được sử dụng để giải nén dữ liệu từ tệp pickle.
  + **train\_data.append(...)**: Dữ liệu giải nén từ tệp pickle sẽ được thêm vào danh sách train\_data. Sau khi vòng lặp hoàn tất, train\_data sẽ chứa tất cả các batch dữ liệu từ các tệp pickle trong thư mục batch\_path.

f. Xử lý mô hình:



Tạo và huấn luyện mô hình:

* **model = create\_model()**:
  + Hàm create\_model() được gọi để khởi tạo và trả về một mô hình CNN đã được cấu hình sẵn.
* **results = train\_model(model, train\_data, test\_data)**:
  + Hàm train\_model() nhận các tham số model, train\_data, và test\_data để thực hiện quá trình huấn luyện mô hình.
  + train\_data là tập dữ liệu huấn luyện và test\_data là tập dữ liệu kiểm tra.
  + Trong quá trình huấn luyện, hàm này sẽ theo dõi và ghi lại độ chính xác của mô hình trên cả tập huấn luyện và tập kiểm tra. Kết quả này được lưu trong results.

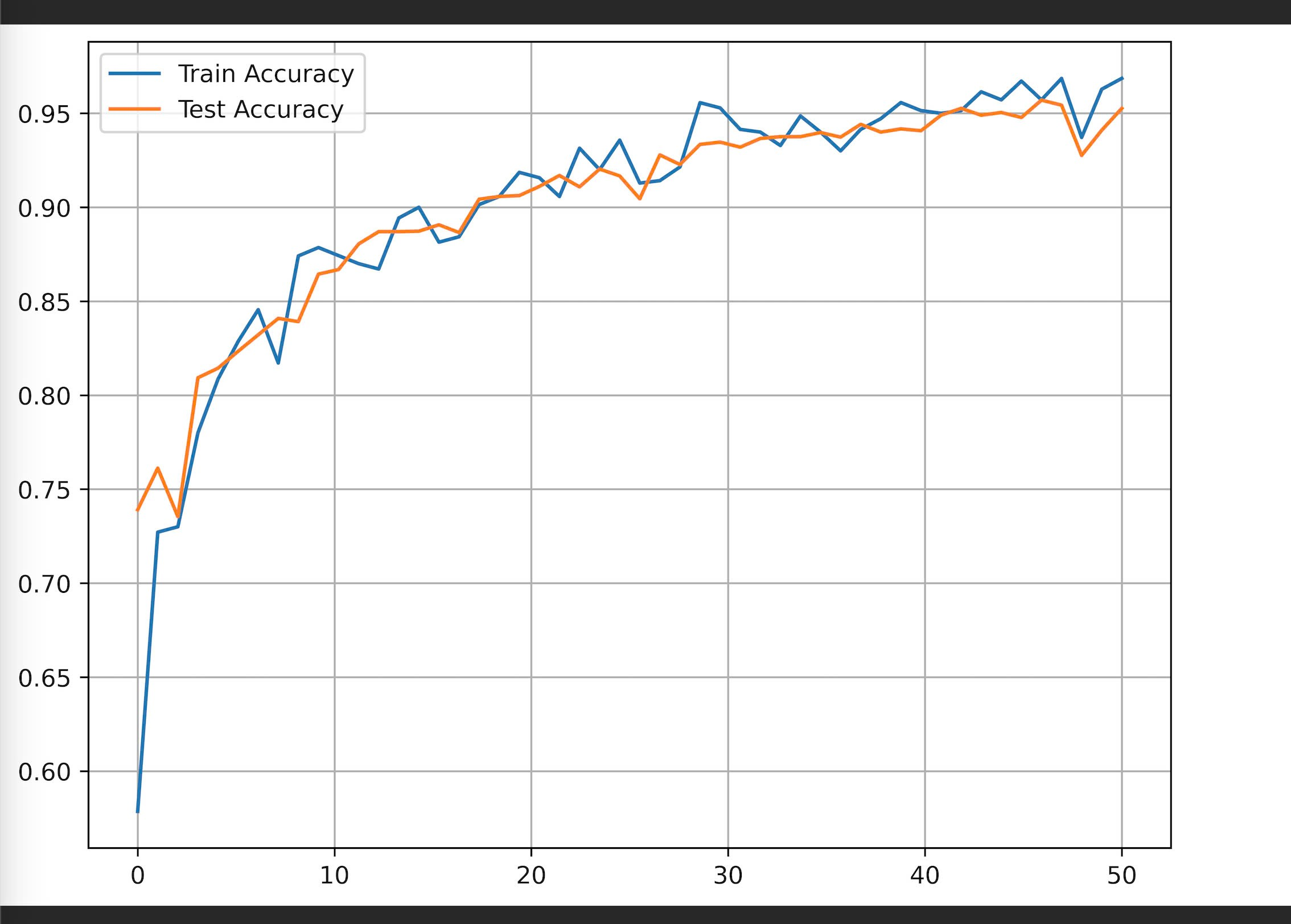
Lưu mô hình đã huấn luyện:

* **model.save(...)**:
  + Sau khi quá trình huấn luyện hoàn tất, mô hình đã học được các trọng số tối ưu từ dữ liệu huấn luyện.
  + Hàm save() sẽ lưu mô hình (bao gồm cấu trúc và các trọng số) vào một tệp .keras tại đường dẫn chỉ định.
  + Việc lưu này cho phép tải lại mô hình trong tương lai để thực hiện các dự đoán mới hoặc kiểm tra mô hình mà không cần phải huấn luyện lại từ đầu.

Lưu kết quả về độ chính xác:

* **with open(...) as f:**:
  + Mở một tệp mới hoặc ghi đè lên tệp cũ tại đường dẫn được cho.
  + Tệp này được mở ở chế độ wb (write-binary) vì dữ liệu sẽ được ghi dưới dạng nhị phân.
* **pkl.dump(results, f)**:
  + pkl.dump() là hàm của thư viện pickle để lưu trữ dữ liệu vào tệp.
  + Tham số results là danh sách chứa thông tin về độ chính xác trên tập huấn luyện và tập kiểm tra tại mỗi epoch trong quá trình huấn luyện.
  + results được lưu vào tệp accuracy.pickle, giúp có thể tải lại và phân tích kết quả huấn luyện sau này mà không cần huấn luyện lại mô hình.
    1. Đánh giá:

Biểu đồ kết quả huấn luyện:



*Hình 11 - Biểu đồ đánh giá kết quả huấn luyện.*

Phân tích chi tiết biểu đồ, ta nhận thấy:

**Trục X:** Đại diện cho số lượng epoch, tức là số lần mà mô hình được huấn luyện qua toàn bộ tập dữ liệu huấn luyện.

**Trục Y:** Đại diện cho độ chính xác (accuracy) của mô hình. Giá trị dao động từ 0.70 đến 1.00, với giá trị càng cao càng thể hiện độ chính xác càng lớn.

**Đường Train Accuracy (màu xanh):** Đây là độ chính xác của mô hình trên tập huấn luyện trong mỗi epoch.

* Ban đầu: Độ chính xác của mô hình trên tập huấn luyện bắt đầu ở mức khoảng 0.60.
* Giai đoạn tăng trưởng: Đường này tăng nhanh chóng qua các epoch đầu tiên, đạt gần 0.90 sau khoảng 10 epoch.
* Giai đoạn ổn định: Từ epoch thứ 10 trở đi, độ chính xác tiếp tục tăng dần và dao động nhẹ, ổn định quanh mức 0.95 - 0.97.

**Đường Test Accuracy (màu cam):** Đây là độ chính xác của mô hình trên tập kiểm thử trong mỗi epoch.

* Ban đầu: Độ chính xác trên tập kiểm thử bắt đầu gần giống với tập huấn luyện, khoảng 0.60 - 0.70.
* Giai đoạn tăng trưởng: Test Accuracy cũng tăng nhanh chóng trong những epoch đầu tiên, đạt gần 0.90 sau khoảng 10 epoch.
* Giai đoạn ổn định: Sau đó, nó tiếp tục tăng dần và dao động nhẹ quanh mức 0.95 - 0.97, rất gần với đường Train Accuracy.

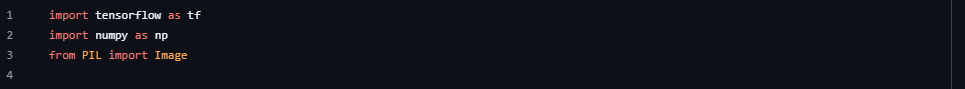
**Sự tương đồng giữa hai đường:** Đường Train Accuracy và Test Accuracy có xu hướng rất gần nhau, điều này cho thấy mô hình không bị overfitting (quá khớp) hoặc underfitting (chưa khớp) mà đạt được một mức độ tổng quát tốt.

**Giá trị cực đại:** Mô hình đạt độ chính xác tối đa gần 1.00 cho cả tập huấn luyện và kiểm thử, cho thấy mô hình có khả năng phân loại tốt trên cả hai tập dữ liệu.

**Kết luận**: Biểu đồ này cho thấy mô hình đang hoạt động hiệu quả, với độ chính xác cao trên cả tập huấn luyện và tập kiểm thử. Điều này có thể phản ánh mô hình đã được tối ưu hóa tốt và có thể hoạt động tốt trên dữ liệu mới.

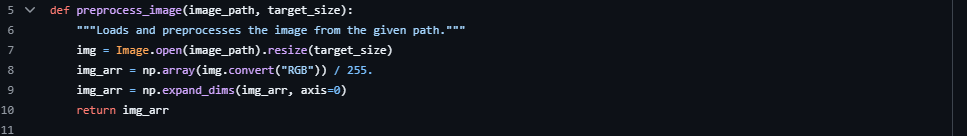
* 1. Khởi tạo giao diện:
     1. Xây dựng:
        1. Hàm cho các nút (image\_detector):

a. Các thư viện cần thiết:



* **import tensorflow as tf**: Nhập TensorFlow để xây dựng và huấn luyện mô hình học sâu.
* **import numpy as np**: Nhập NumPy để xử lý mảng và tính toán khoa học.
* **from PIL import Image**: Nhập Image từ PIL để mở, xử lý, và lưu hình ảnh.

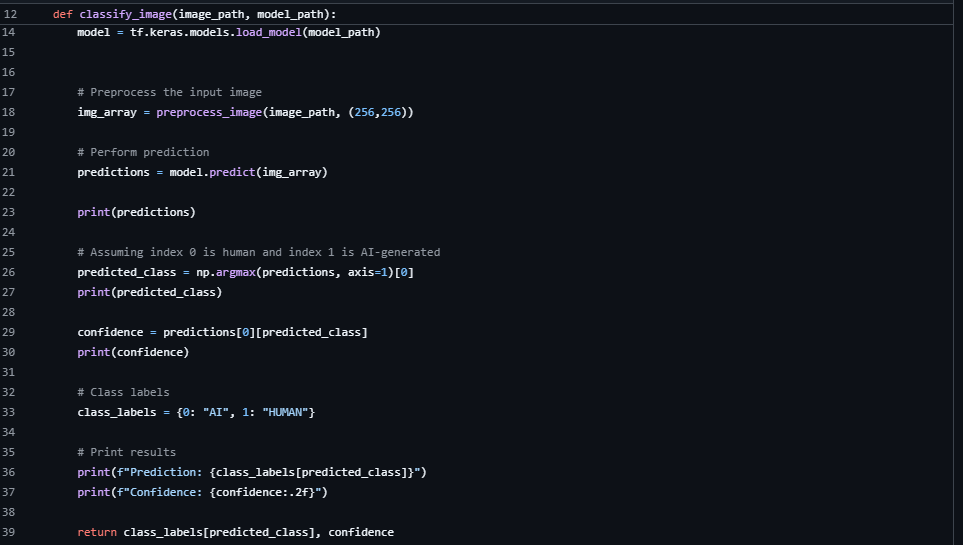
b. Hàm **preprocess\_image()**:



Trả về một mảng img\_arr chuẩn hóa, với kích thước phù hợp cho đầu vào mô hình (1, height, width, 3).

* **Image.open(image\_path)**: Mở ảnh từ đường dẫn image\_path sử dụng thư viện PIL. Hàm này trả về một đối tượng Image.
* **.resize(target\_size)**: Thay đổi kích thước ảnh thành target\_size.
* **img.convert("RGB")**: Chuyển đổi ảnh sang dạng RGB.
* **np.array(img.convert("RGB"))**: Chuyển đối tượng Image thành mảng NumPy.
* **/ 255.**: Chuẩn hóa giá trị pixel của ảnh từ phạm vi [0, 255] về phạm vi [0, 1] bằng cách chia cho 255.
* **np.expand\_dims(img\_arr, axis=0)**: Hàm này thêm một chiều vào đầu mảng, biến ảnh thành một "batch" có kích thước 1.

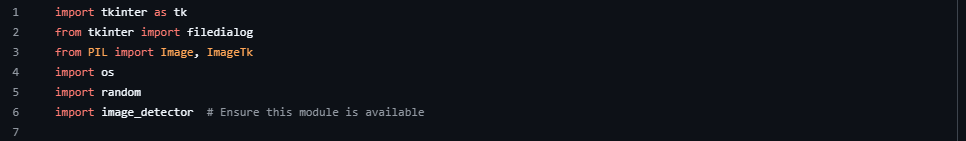
c. Hàm **classify\_image( )**:



Tải mô hình, xử lý ảnh, và dự đoán xem ảnh là "AI" hoặc "HUMAN".

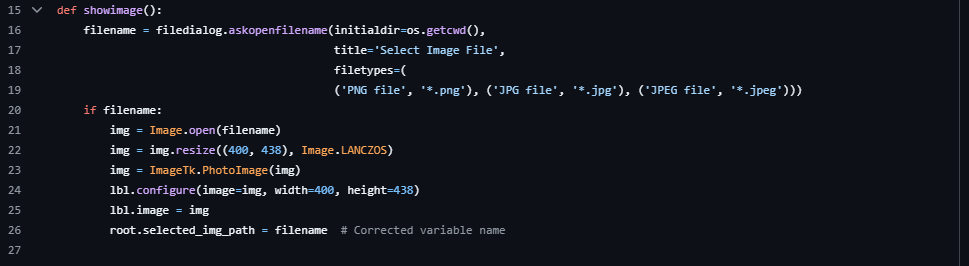
* **model = tf.keras.models.load\_model(model\_path)**: Tải mô hình đã được huấn luyện trước đó từ tệp model\_path.
* **img\_array = preprocess\_image(image\_path, (256, 256))**: Gọi hàm preprocess\_image để tải và tiền xử lý ảnh đầu vào từ image\_path. Hàm này trả về ảnh đã được thay đổi kích thước và chuẩn hóa.
* **predictions = model.predict(img\_array)**: Dùng phương thức predict của mô hình để dự đoán lớp của ảnh đã xử lý. Kết quả predictions là một mảng xác suất cho từng lớp.
* **print(predictions)**: In ra mảng xác suất dự đoán của mô hình cho các lớp.
* **predicted\_class = np.argmax(predictions, axis=1)[0]**: Tìm chỉ số của lớp có xác suất cao nhất trong mảng predictions sử dụng np.argmax.
* **confidence = predictions[0][predicted\_class]**: Lấy giá trị xác suất của lớp được dự đoán từ mảng predictions tại chỉ số predicted\_class.
* **class\_labels = {0: "AI", 1: "HUMAN"}**: Tạo một từ điển ánh xạ chỉ số lớp (0 hoặc 1) đến nhãn lớp tương ứng ("AI" hoặc "HUMAN").
* **print(f"Prediction: {class\_labels[predicted\_class]}")**: In ra nhãn của lớp dự đoán, sử dụng từ điển class\_labels.
* **print(f"Confidence: {confidence:.2f}")**: In ra mức độ tự tin của mô hình đối với dự đoán, làm tròn đến 2 chữ số thập phân.
* **return class\_labels[predicted\_class], confidence**: Trả về nhãn của lớp dự đoán và độ tự tin (confidence) tương ứng.
  + - 1. Giao diện UI:

a. Các thư viện cần thiết:



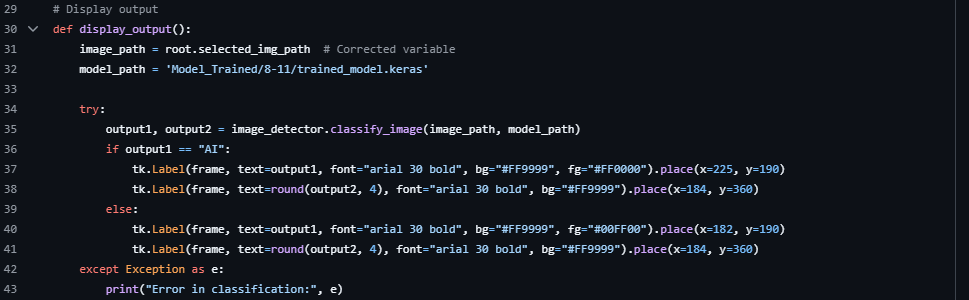
* **tkinter**: Thư viện chuẩn trong Python để tạo giao diện người dùng đồ họa.
* **filedialog**: Module trong tkinter dùng để mở hộp thoại chọn tệp (file dialog) cho người dùng chọn hình ảnh.
* **PIL (Python Imaging Library)**: Sử dụng Image và ImageTk để mở, xử lý và hiển thị hình ảnh.
* **os**: Để tương tác với hệ điều hành.
* **image\_detector**: Hàm ở mục (a) để gán vào các nút.

b. **Hàm showimage( )** - Chọn và hiển thị hình ảnh:



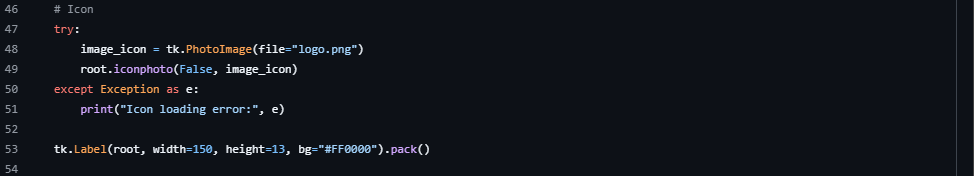
* **Mở hộp thoại chọn tệp**: filedialog.askopenfilename() mở một cửa sổ chọn tệp, cho phép người dùng chọn tệp hình ảnh với các định dạng .png, .jpg, .jpeg.
* **Mở hình ảnh**: Image.open(filename) mở hình ảnh từ đường dẫn người dùng chọn.
* **Thay đổi kích thước hình ảnh**: img.resize((400, 438), Image.LANCZOS) thay đổi kích thước của hình ảnh thành 400x438 pixel.
* **Chuyển đổi thành PhotoImage**: ImageTk.PhotoImage(img) chuyển đổi đối tượng hình ảnh PIL thành một đối tượng PhotoImage có thể hiển thị trong tkinter.
* **Hiển thị hình ảnh**: lbl.configure(image=img, width=400, height=438) thay đổi nội dung hình ảnh trong nhãn lbl để hiển thị hình ảnh đã chọn.
* **Lưu đường dẫn hình ảnh**: root.selected\_img\_path = filename lưu đường dẫn của tệp hình ảnh vào thuộc tính của cửa sổ root để có thể sử dụng sau.

c. **Hàm display\_output( )** - Phân loại hình ảnh và hiển thị kết quả



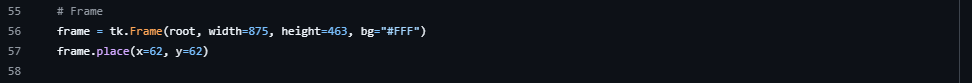
* **Lấy đường dẫn hình ảnh**: image\_path = root.selected\_img\_path lấy đường dẫn của hình ảnh người dùng đã chọn từ thuộc tính root.selected\_img\_path.
* **Đường dẫn mô hình AI**: model\_path = 'Model\_Trained/8 11/trained\_model.keras' xác định đường dẫn đến mô hình học sâu đã huấn luyện (tệp Keras).
* **Phân loại hình ảnh**: output1, output2 = image\_detector.classify\_image(image\_path, model\_path) gọi hàm classify\_image từ module image\_detector để phân loại hình ảnh.
  + output1: Kết quả phân loại, có thể là "AI" hoặc "Not AI".
  + output2: Độ tin cậy của phân loại (ví dụ: 0.95 nghĩa là 95% tin cậy vào kết quả).
* **Hiển thị kết quả**:
  + Nếu kết quả là "AI", văn bản "AI" sẽ được hiển thị với màu đỏ (fg="#FF0000"), độ tin cậy sẽ được hiển thị với màu nền hồng (bg="#FF9999").
  + Nếu kết quả là "Human", văn bản "Human" sẽ được hiển thị với màu xanh lá (fg="#00FF00").
  + Các nhãn (Label) được tạo và hiển thị tại các vị trí cố định trên cửa sổ (place(x=..., y=...)).
* **Xử lý lỗi**: Nếu có lỗi xảy ra trong quá trình phân loại , nó sẽ được in ra bằng câu lệnh print("Error in classification:", e).

d. Biểu tượng và giao diện chính:



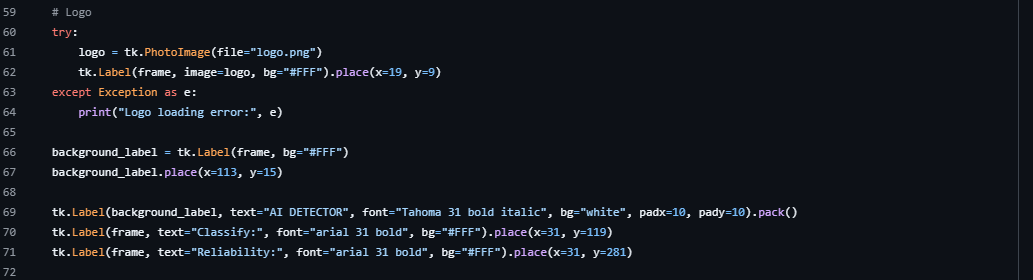
* Thử tải một biểu tượng ứng dụng từ tệp logo.png và đặt biểu tượng đó cho cửa sổ ứng dụng bằng root.iconphoto(False, image\_icon).
* **tk.Label( ).pack( )**: Thêm một nhãn màu đỏ ở đầu cửa sổ.

e. Khung giao diện người dùng:

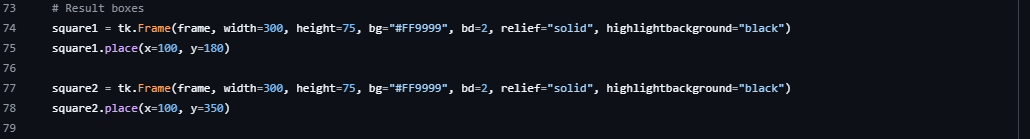


* Tạo một khung (Frame) chính chứa các phần tử giao diện người dùng, với kích thước 875x463 pixel và nền trắng (bg="#FFF").
* frame.place(x=62, y=62) đặt khung tại tọa độ (62, 62) trong cửa sổ chính.

f. Logo và các nhãn:

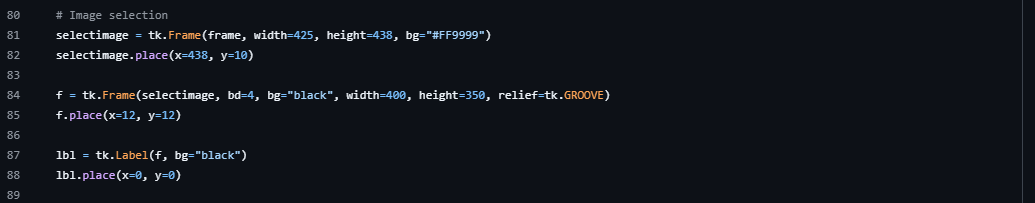


* Hiển thị logo trong khung frame.
* Tạo tiêu đề “AI DETECTOR” của ứng dụng
* Tạo các nhãn “Classify” và “Reliability”



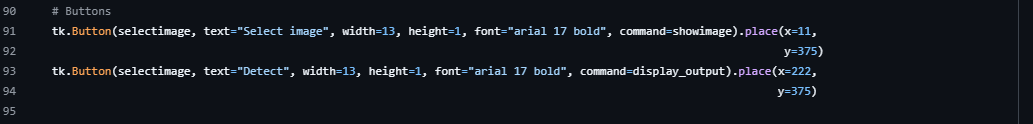
* Tạo các khung có màu nền hồng để hiển thị kết quả phân loại và độ tin cậy.

h. Khung chọn ảnh:

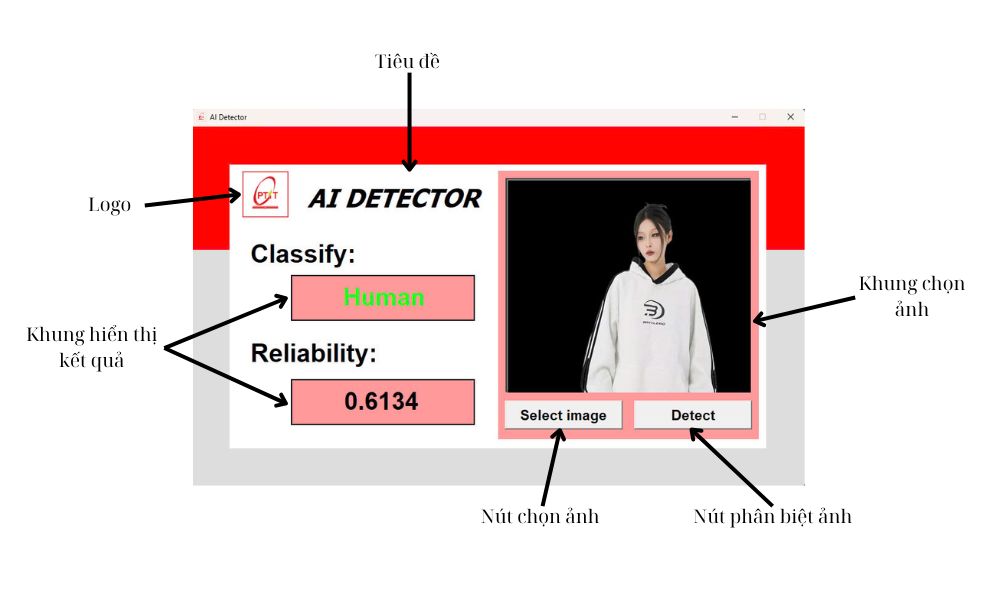


* Tạo khung để người dùng có thể chọn và xem hình ảnh đã tải lên.

i. Các nút:

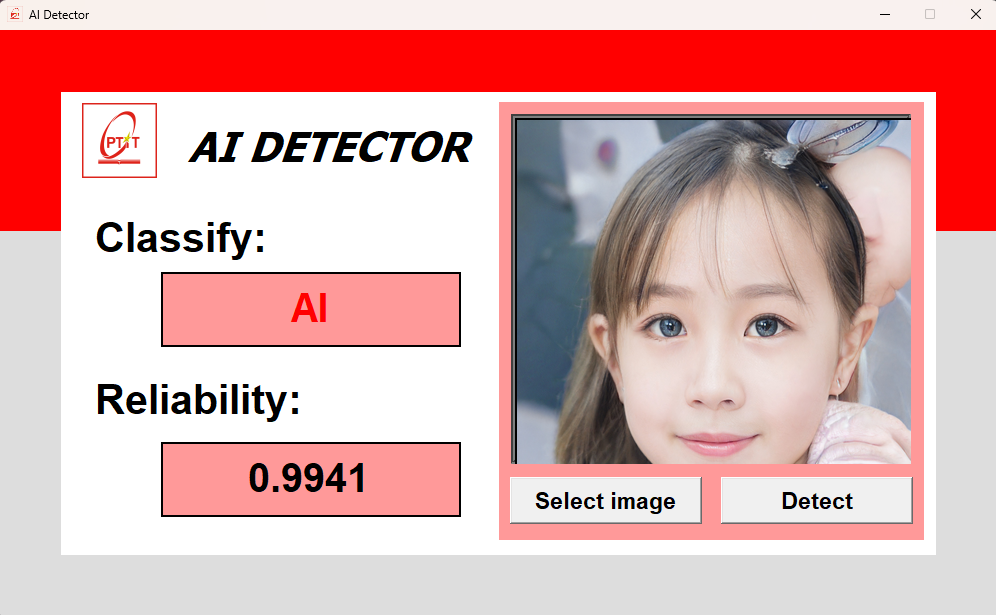


* Nút **“Select image”**: Cho phép người dùng chọn hình ảnh.
* Nút **“Detect”**: Kích hoạt việc phân loại và hiển thị kết quả.
  + 1. Hiển thị giao diện GUI:

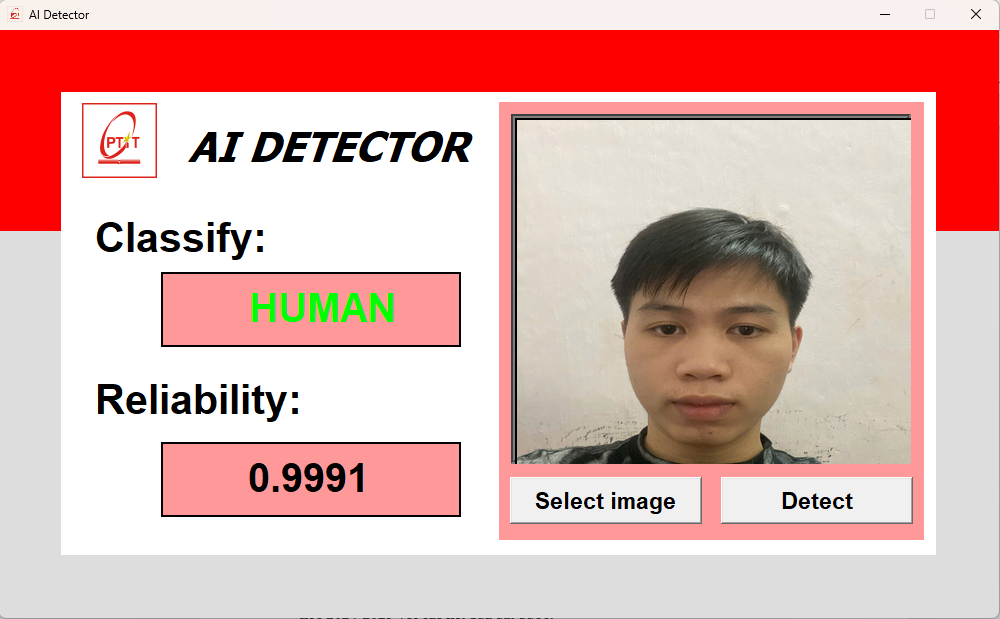


Hình 12 – Giao diện ứng dụng.

2.4 Thử nghiệm ứng dụng:



Hình 13 – Thử nghiệm với ảnh AI.



Hình 14 – Thử nghiệm với ảnh người.

1. KẾT LUẬN
   1. Các kết quả đạt được:

Nhóm thực hiện đề tài “Bài toán phân biệt ảnh chân dung được tạo bới AI hay người thật với CNN” đã hoàn thành bài tập lớn bộ môn lập trình với Python kì 1 năm học 2024-2025 với các kết quả đạt được:

* Tạo ra và huấn luyện một AI phân biệt hình ảnh đơn giản
* Tạo ra một giao diện đơn giản để kết hợp với AI đã tạo để tạo thành được một ứng dụng phân biệt hình ảnh đơn giản.
* Hoàn thành được bài toán đã đề ra.
  1. Hướng phát triển dự định:

Chúng em có dự định ứng dụng có thể phát triển theo các hướng sau:

* Nâng cấp khả năng cũng như tăng cường độ chính xác trong việc phân tích và phân biệt hình ảnh của AI.
* Nâng cấp thẩm mĩ cho giao diện của ứng dụng cũng như thêm các tính năng mới cho ứng dụng.

TÀI LIỆU THAM KHẢO:

1. <https://www.youtube.com/watch?v=NVswJrXv9I&list=PLQY2H8rRoyvxNqk9EV5VP5fS0cWEXW5QQ>
2. <https://aws.amazon.com/vi/what-is/neural-network/>
3. <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>
4. <https://phamdinhkhanh.github.io/2019/08/22/convolutional-neural-network.html>
5. <https://aws.amazon.com/vi/ai/>
6. <https://link.springer.com/chapter/10.1007/978-3-030-87664-7_9>
7. <https://keras.io/>