ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

======***======

BÁO CÁO BTL THUỘC HỌC PHẦN:

TRÍ TUỆ NHÂN TẠO

XÂY DỰNG HỆ THỐNG NHẬN DIỆN KHUÔN MẶT BẰNG CNN

GVHD: Mai Thanh Hồng

Nhóm - Lớp: 4 - 20244IT6121001

Thành viên: Đỗ Phương Nam- 2023602305

Đoàn Bá Hoàng- 2023601704

Nguyễn Đình Đại- 2023602486

Trần Quốc Toản- 2023602364

Hà Nội, Năm 2025

LỜI CẢM ƠN

Lời đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến cô Mai Thanh Hồng. Trong suốt thời gian thực hiện bài tập lớn học phần Trí tuệ nhân tạo, cô đã luôn tận tình hướng dẫn, giải đáp thắc mắc và định hướng cho chúng em phương pháp nghiên cứu khoa học, giúp chúng em có thể từng bước giải quyết các vấn đề và hoàn thành đề tài "Xây dựng hệ thống nhận diện khuôn mặt bằng CNN".

Chúng em cũng xin gửi lời tri ân đến các thầy cô giáo trong Đại học Công nghiệp Hà Nội. Những kiến thức nền tảng vững chắc và sự chỉ bảo tận tình của các thầy cô trong suốt quá trình học tập tại trường chính là cơ sở quan trọng giúp chúng em có đủ khả năng để tiếp cận và thực hiện dự án này.

Mặc dù cả nhóm đã nỗ lực hết mình để hoàn thành bài báo cáo với kết quả tốt nhất, song do giới hạn về mặt thời gian và kinh nghiệm thực tiễn còn hạn chế, bài làm chắc chắn không thể tránh khỏi những thiếu sót. Chúng em rất mong nhận được những ý kiến đóng góp, nhận xét quý báu từ Cô để chúng em có thể khắc phục, rút kinh nghiệm và hoàn thiện hơn kiến thức của mình trong tương lai.

Chúng em xin chân thành cảm ơn!

1. PHIẾU HỌC TẬP CÁ NHÂN/NHÓM

I. Thông tin chung

1. Tên lớp: 20244IT6121001 Khóa: 8
2. Họ và tên sinh viên: Đỗ Phương Nam
3. Tên nhóm: Nhóm 4
4. Thành viên trong nhóm:

Họ và tên	Mã sinh viên
Đỗ Phương Nam	2023602305
Đoàn Bá Hoàng	2023601704
Trần Đình Đại	2023602766
Trần Quốc Toản	2023602364

II. Nội dung học tập

1. Tên chủ đề: Xây dựng hệ thống nhận diện khuôn mặt/người bằng mạng nơ-ron tích chập (CNN).

2. Hoạt động của sinh viên

- Hoạt động/Nội dung 1: Tìm hiểu lý thuyết về mạng CNN và bài toán nhận diện
    + Mục tiêu/chuẩn đầu ra: Sinh viên hiểu được kiến trúc mạng LeNet, các lớp tích chập (Convolution), lớp gộp (Pooling), và các hàm kích hoạt (ReLU, Sigmoid) phù hợp cho bài toán phân loại nhị phân.
- Hoạt động/Nội dung 2: Thu thập và tiền xử lý dữ liệu (Data Preprocessing)

+ Mục tiêu/chuẩn đầu ra: Thu thập bộ dữ liệu Natural Images, phân chia thành hai lớp "Person" và "No_person". Thực hiện chuẩn hóa kích thước ảnh (150x150 pixels), chuẩn hóa giá trị pixel và tăng cường dữ liệu (Augmentation) để tránh overfitting.

- Hoạt động/Nội dung 3: Huấn luyện mô hình, Serving và xây dựng giao diện

    + Mục tiêu/chuẩn đầu ra: Xây dựng thành công mô hình LeNet trên Keras/TensorFlow, đạt độ chính xác trên 85%. Triển khai model thành API bằng FastAPI và xây dựng giao diện người dùng bằng Streamlit để demo nhận diện từ hình ảnh upload.

3. Sản phẩm nghiên cứu Một hệ thống nhận diện người hoàn chỉnh, bao gồm:

- Mô hình CNN (kiến trúc LeNet) đã được huấn luyện và lưu trọng số (.h5).
- Bộ dữ liệu đã được xử lý và chia tập Train/Val/Test hợp lý.
- Source code ứng dụng Web (Streamlit) kết nối với API (FastAPI) để dự đoán kết quả.
- Báo cáo chi tiết quy trình thực hiện và kết quả đánh giá.

III. Nhiệm vụ học tập

1. Hoàn thành Báo cáo Bài tập lớn theo đúng thời gian quy định.
2. Báo cáo sản phẩm và demo ứng dụng trước giảng viên.

IV. Học liệu thực hiện

1. Tài liệu học tập:

- Giáo trình Trí tuệ nhân tạo, Thị giác máy tính.
- Tài liệu tham khảo về kiến trúc LeNet-5 và thư viện Keras/TensorFlow.
- Documentation của FastAPI và Streamlit.

2. Phương tiện: Máy tính cài đặt Python, môi trường Google Colab hoặc Jupyter Notebook.

2. KẾ HOẠCH THỰC HIỆN BÀI TẬP LỚN

Tên chủ đề: Xây dựng hệ thống nhận diện khuôn mặt/người bằng CNN

Tuần	**Người thực hiện**	**Nội dung công việc**	**Phương pháp thực hiện**
1	Đỗ Phương Nam	Tổ chức họp nhóm, xác định cấu trúc báo cáo và phân công nhiệm vụ.	Đã thống nhất đề tài và bảng phân công công việc..
	Đoàn Bá Hoàng	Tìm hiểu tổng quan về đề tài và ứng dụng thực tế của Computer Vision.	Hoàn thành nội dung Chương 1 (Lý do chọn đề tài).
	Trần Đình Đại	Tìm hiểu nguyên lý hoạt động của mạng CNN.	Đã nắm được khái niệm Convolution và Pooling.
	Trần Quốc Toản	Thiết lập Drive chung và tìm kiếm tài liệu tham khảo.	Đã tạo cấu trúc thư mục và tải lên các giáo trình cần thiết.
2	Đỗ	Viết cơ sở lý thuyết về	Hoàn thành bản thảo phần

	Phương Nam	kiến trúc LeNet và lớp Fully Connected.	kiến trúc mạng LeNet.
	Đoàn Bá Hoàng	Nghiên cứu về hàm mất mát (Loss) và thuật toán tối ưu (Optimizer).	Đã viết xong phần lý thuyết về Binary Crossentropy và Adam.
	Trần Đình Đại	Viết phần Tổng quan dự án và Mục tiêu nghiên cứu.	Hoàn thành mục 1.2 và 1.3 của báo cáo
	Trần Quốc Toản	Viết lý thuyết chi tiết về các lớp Tích chập và Gộp (Pooling).	Hoàn thành nội dung lý thuyết các lớp trong CNN.
3	Đỗ Phương Nam	Xây dựng kịch bản Augmentation (xoay, lật ảnh) để tăng dữ liệu.	Đã có code sinh thêm dữ liệu train thành công.
	Đoàn Bá Hoàng	Thu thập bộ dữ liệu Natural Images và kiểm tra nhãn	Đã tải và lọc sạch bộ dữ liệu (Person/No_person).
	Trần Đình Đại	Viết code tiền xử lý (Resize, Normalize).	Đã đưa toàn bộ ảnh về kích thước 150x150 chuẩn.
	Trần Quốc Toản	Phân chia dữ liệu thành 3 tập Train/Val/Test.	Đã chia dữ liệu theo tỷ lệ 70/15/15.
4	Đỗ Phương	Code và huấn luyện mô hình LeNet trên Colab.	Mô hình chạy tốt, không lỗi cú pháp.

	Nam		
	Đoàn Bá Hoàng	Tinh chỉnh Learning Rate và Batch size.	Đã tìm ra tham số tối ưu giúp model hội tụ nhanh
	Trần Đình Đại	Đánh giá mô hình trên tập Test.	Model đạt độ chính xác >95% trên tập kiểm thử.
	Trần Quốc Toản	Cấu hình Callbacks (lưu model, early stopping).	Đã lưu được file trọng số .h5 tốt nhất.
5	Đỗ Phương Nam	Xây dựng API Serving Model với FastAPI.	API hoạt động, nhận ảnh và trả về JSON kết quả.
	Đoàn Bá Hoàng	Thiết kế giao diện Web với Streamlit.	Giao diện hiển thị tốt, có nút upload ảnh.
	Trần Đình Đại	Kiểm thử API với các trường hợp ảnh khác nhau.	API xử lý tốt cả ảnh người và vật.
	Trần Quốc Toản	Kết nối Frontend (Streamlit) với Backend (API).	Hệ thống đã chạy đồng bộ, dự đoán chính xác.
6	Đỗ Phương Nam	Viết chương Kết luận và Hướng phát triển.	Hoàn thành chương cuối của báo cáo.

	Đoàn Bá Hoàng	Chụp ảnh demo hệ thống đưa vào báo cáo.	Đã có đủ hình ảnh minh họa cho Chương V.
	Trần Đình Đại	Viết báo cáo chi tiết phần Kết quả thực nghiệm.	Đã phân tích các biểu đồ Loss/Accuracy
	Trần Quốc Toản	Rà soát, format văn bản và mục lục.	Báo cáo đã đúng chuẩn định dạng của trường.
7	Đỗ Phương Nam	Chuẩn bị slide thuyết trình	Slide đã hoàn thiện, đủ nội dung chính.
	Đoàn Bá Hoàng	Dọn dẹp source code và upload lên Drive/Git.	Code đã được tổ chức gọn gàng, có comment.
	Trần Đình Đại	Luyện tập thuyết trình phần Kết quả.	Tự tin trình bày phần đánh giá.
	Trần Quốc Toản	In ấn và nộp quyển báo cáo.	Đã nộp báo cáo đúng hạn.

Ngày tháng năm 2025.

XÁC NHẬN CỦA GIẢNG VIÊN

(Ký, ghi rõ họ tên)

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

CHƯƠNG I: TỔNG QUAN VỀ DỰ ÁN

1.1. Lý do chọn đề tài

1.1.1. Bối cảnh phát triển của AI và Computer Vision

Trí tuệ nhân tạo (AI) và Học sâu (Deep Learning) đang tạo nên cuộc cách mạng trong kỷ nguyên số, thay đổi cách con người tương tác với máy móc và dữ liệu. Trong đó, Thị giác máy tính (Computer Vision) là một trong những lĩnh vực phát triển nhanh nhất, cho phép máy tính thu nhận, xử lý và hiểu được nội dung từ hình ảnh hoặc video giống như thị giác của con người.

Mạng nơ-ron tích chập (CNN - Convolutional Neural Networks) đã trở thành công nghệ cốt lõi trong các bài toán thị giác máy tính nhờ khả năng trích xuất đặc trưng vượt trội, mang lại độ chính xác cao trong các tác vụ như phân loại ảnh, nhận diện khuôn mặt và phát hiện đối tượng.

1.1.2. Ý nghĩa của bài toán nhận diện người

Trong thực tế, nhu cầu tự động hóa việc giám sát và phân tích hành vi con người ngày càng cấp thiết. Từ các hệ thống an ninh thông minh, xe tự hành, cho đến các cửa hàng bán lẻ không người bán, việc phát hiện chính xác sự hiện diện của con người trong khung hình là bước xử lý đầu tiên và quan trọng nhất.

Các phương pháp giám sát truyền thống phụ thuộc hoàn toàn vào sự tập trung của con người, dẫn đến hiệu suất không ổn định và tốn kém nhân lực khi quy mô giám sát mở rộng. Do đó, việc ứng dụng mạng CNN để xây dựng hệ thống tự động nhận diện người là một hướng tiếp cận hiện đại, giúp nâng cao hiệu quả giám sát, đảm bảo an ninh và mở ra nhiều ứng dụng thực tiễn trong đời sống.

1.2. Mục tiêu đề tài

1.2.1. Mục tiêu tổng quát

Xây dựng và hoàn thiện một mô hình Trí tuệ nhân tạo (AI) có khả năng phân loại nhị phân (Binary Classification) đối với hình ảnh đầu vào, nhằm xác định sự hiện diện của con người.

1.2.2. Mục tiêu cụ thể

Để đạt được mục tiêu tổng quát, nhóm thực hiện tập trung vào các nhiệm vụ cụ thể sau:

Xây dựng mô hình: Huấn luyện thành công mô hình CNN dựa trên kiến trúc LeNet – một kiến trúc mạng tích chập kinh điển, phù hợp cho việc học tập và nhận diện các đặc trưng cơ bản.

Hiệu suất mô hình: Đạt độ chính xác (Accuracy) trên 85% khi kiểm thử với tập dữ liệu Natural Images.

Sản phẩm ứng dụng: Xây dựng script Python hoàn chỉnh để đưa ảnh vào mô hình và trả về kết quả nhãn dự đoán là "Person" (Có người) hoặc "No Person" (Không có người).

1.3. Phương pháp nghiên cứu

1.3.1. Phương pháp lý thuyết

Nghiên cứu kiến trúc mạng nơ-ron tích chập (CNN) để hiểu cơ chế trích xuất đặc trưng.

Tìm hiểu sâu về các hàm kích hoạt phi tuyến tính như ReLU (Rectified Linear Unit) giúp tăng tốc độ hội tụ, và Sigmoid dùng cho lớp đầu ra của bài toán phân loại nhị phân.

Nghiên cứu thuật toán tối ưu Adam để cập nhật trọng số hiệu quả trong quá trình huấn luyện.

1.3.2. Phương pháp thực nghiệm

Quy trình thực nghiệm được tiến hành qua các bước:

- Thu thập dữ liệu: Sử dụng bộ dữ liệu Natural Images.

- Xử lý dữ liệu (Data preprocessing): Chuẩn hóa kích thước ảnh, cân bằng dữ liệu.

- Huấn luyện (Training): Đưa dữ liệu vào mô hình LeNet.

- Đánh giá (Evaluation): Kiểm tra độ chính xác trên tập test.

1.3.3. Công cụ hỗ trợ

- Ngôn ngữ lập trình: Python.

- Thư viện: PyTorch, Torchvision, Matplotlib

- Môi trường phát triển: Google Colab hoặc Jupyter Notebook.

1.4. Đối tượng và phạm vi nghiên cứu

1.4.1. Đối tượng nghiên cứu

Các đặc trưng hình ảnh đại diện cho con người: Bao gồm khuôn mặt, dáng người, tư thế.

Các vật thể nền (Background) thường gặp để phân biệt: Xe cộ, hoa, quả, động vật, v.v.

1.4.2. Phạm vi nghiên cứu

Tập trung giải quyết bài toán phân loại 2 lớp (Binary Classification).

Sử dụng bộ dữ liệu Natural Images đã được xử lý cân bằng số lượng mẫu giữa hai lớp (Person và Non-person) để tránh hiện tượng mô hình học lệch (bias).

1.5. Tình hình nghiên cứu hiện tại

Hiện nay, các ứng dụng nhận diện con người và khuôn mặt đang trở nên vô cùng phổ biến trong đời sống:

- Hệ thống điểm danh/chấm công: Sử dụng FaceID tại các công ty, trường học.

- An ninh sân bay/cửa khẩu: Hệ thống e-KYC tự động xác minh danh tính.

- Thiết bị cá nhân: Mở khóa điện thoại, laptop bằng khuôn mặt.

Tuy nhiên, hầu hết các hệ thống thương mại đều sử dụng các mô hình rất phức tạp và nặng (như ResNet, Yolo). Đề tài này tập trung vào

việc nghiên cứu mô hình LeNet đơn giản hơn để hiểu rõ bản chất cốt lõi của việc nhận diện, phù hợp cho các bài toán yêu cầu tài nguyên phần cứng thấp.

1.6. Tiềm năng và hướng phát triển

Dựa trên kết quả của mô hình nhận diện cơ bản, đề tài có thể mở rộng theo các hướng sau:

Tích hợp thời gian thực: Áp dụng mô hình vào luồng video (webcam) để phát hiện người di chuyển qua lại.

Nâng cấp mô hình: Chuyển sang các kiến trúc sâu hơn như VGG16 hoặc MobileNet để tăng độ chính xác trong môi trường thiếu sáng hoặc bị che khuất.

Mở rộng bài toán: Phát triển từ phân loại (có người hay không) sang bài toán đếm số lượng người trong ảnh (Crowd Counting).

Chương II: Cơ sở lý thuyết của bài toán

2.1. Bài toán nhận diện người

2.1.1 Mô tả bài toán

Nhận diện người (Person Detection) là một bài toán thuộc lĩnh vực Thị giác máy tính (Computer Vision), trong đó mục tiêu của mô hình là xác định xem trong ảnh đầu vào có chứa đối tượng "người" hay không và là dạng bài toán phân loại nhị phân (binary classification).

Input: Một ảnh màu (RGB), đã được resize về kích thước chuẩn (ví dụ: 64×64).

Output:

- Nhãn 0: Không phải người
- Nhãn 1: Là người

Bài toán tưởng chừng đơn giản nhưng thực tế chứa nhiều thách thức do sự phong phú của hình ảnh đời thực, ảnh hưởng của ánh sáng, độ phân giải, góc chụp và các yếu tố nhiễu trong ảnh.

2.1.2 Mô hình áp dụng: LeNet-5 (Phiên bản cải tiến)

Lý do chọn LeNet-5: LeNet-5 là một trong những kiến trúc CNN đầu tiên và có ảnh hưởng lớn. Dù đơn giản, nó vẫn giữ đủ cấu trúc cơ bản của một mạng CNN hiện đại, gồm:

- Lớp tích chập (Convolution)
- Lớp gộp(Pooling Layer)
- Lớp kết nối đầy đủ (Fully Connected)

Do đó, LeNet phù hợp để:

- Giải thích rõ ràng các nguyên lý học sâu.
- Thử nghiệm nhanh nhờ số lượng tham số nhỏ.
- Tối ưu tốt với các kích thước ảnh nhỏ đến trung bình (32–64 px).

Các điều chỉnh so với phiên bản gốc

Bản gốc LeNet-5 (1998) dùng:

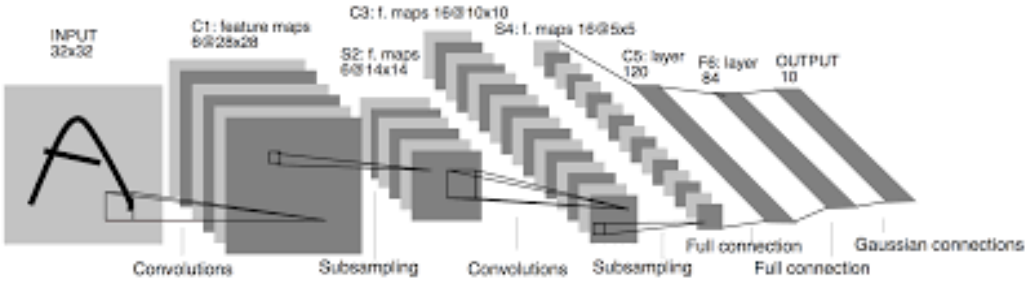
- Kích thước input 32×32,

- Hàm kích hoạt Sigmoid/Tanh,

Để phù hợp hơn với ảnh thực tế trong bài toán Nhận diện người, mô hình được cải tiến:

- Tăng kích thước đầu vào lên 64×64 RGB → Giảm mất mát thông tin khi resize, giúp nhận diện đặc trưng hình dạng người tốt hơn.

- Thay activations Tanh/Sigmoid bằng ReLU → ReLU giúp chống biến mất gradient, hội tụ nhanh hơn, phù hợp với các mô hình hiện đại.



Hình 2.1. Kiến trúc tổng quát của mạng LeNet-5.

2.1.3. Đặc điểm của dữ liệu và Thách thức

Bộ dữ liệu: Natural Images

Bộ dữ liệu Natural Images chứa 8 lớp: person, dog, cat, fruits, airplane, car,flowers, others

Trong bài toán này, dữ liệu được gộp lại thành 2 lớp chính:

- Person (là người)

- Non-person (tổng hợp các lớp còn lại)

Thách thức chính:

-Sự đa dạng cao của lớp Non-person

- Lớp này bao gồm cả động vật (dog, cat), phương tiện (car, airplane), cây cối (flowers), đồ vật ngẫu nhiên.

- Mức độ nhiễu lớn, không đồng nhất → mô hình khó phân biệt nếu đặc trưng bị trùng lặp.

- Ảnh trong lớp Person cũng rất đa dạng

- Khác nhau về dáng người, trang phục, góc chụp, tư thế.

- Có trường hợp người chỉ chiếm một phần nhỏ trong ảnh → gây khó cho mô hình.

- Điều kiện môi trường khác nhau

- Độ sáng mạnh/yếu

- Background phức tạp

- Nhiễu hoặc mờ

- Resize ảnh xuống 64×64 → Dễ làm mất chi tiết quan trọng, nhất là hình người ở xa.

Những thách thức này đòi hỏi cần:

- Tiền xử lý đầy đủ,

- Augmentation hợp lý,

Kiến trúc CNN đủ mạnh (dù vẫn giữ dạng LeNet-based).

2.1.4. Quy trình xử lý bài toán

Bước 1: Thu thập và tiền xử lý dữ liệu:

Dữ liệu được lựa chọn từ bộ Natural Images và được gom lại thành hai nhóm chính: Person và Non-person.

Các ảnh được đưa về kích thước đồng nhất (64×64, RGB) và được chuẩn hóa để phù hợp với mô hình học sâu.

Nhằm tăng tính đa dạng và giúp mô hình tổng quát tốt hơn, một số kỹ thuật tăng cường dữ liệu (data augmentation) được áp dụng như lật ảnh, xoay nhẹ, điều chỉnh độ sáng–tương phản, hoặc thay đổi kích thước.

Bước 2: Xây dựng mô hình:

Mô hình được sử dụng là LeNet-5 phiên bản cải tiến, gồm hai khối tích chập – gộp, theo sau là các lớp kết nối đầy đủ.

Hàm kích hoạt ReLU được sử dụng ở các lớp ẩn để tăng tốc độ hội tụ, và lớp cuối dùng Sigmoid để thực hiện phân loại nhị phân.

Mô hình được tối ưu bằng các thuật toán tối ưu phổ biến như Adam hoặc SGD cùng với hàm mất mát Binary Cross-Entropy phù hợp cho bài toán nhị phân.

Bước 3: Huấn luyện mô hình:

Mô hình được huấn luyện trong nhiều vòng lặp (epochs) với sự hỗ trợ của các kỹ thuật cải thiện quá trình học như Early Stopping, điều chỉnh tốc độ học tự động và lưu lại mô hình tốt nhất.

Trong quá trình huấn luyện, các chỉ số như accuracy, loss, và các thước đo đánh giá phân loại khác (precision, recall, F1-score) được theo dõi để đảm bảo mô hình học ổn định.

Bước 4: Đánh giá mô hình:

Sau khi huấn luyện, mô hình được đánh giá trên tập kiểm thử độc lập nhằm kiểm tra khả năng tổng quát.

Các đồ thị như Confusion Matrix và ROC Curve được sử dụng để trực quan hóa hiệu quả phân loại.

Ngoài ra, mô hình được phân tích thêm thông qua các trường hợp nhầm lẫn điển hình (false positive và false negative), đặc biệt giữa người và các đối tượng có hình dạng tương tự.

2.2. Cơ sở lý thuyết về mạng CNN

2.2.1. Convolutional Neural Network (CNN)

2.2.1.1 Tổng quan về CNN

Convolutional Neural Network (CNN) là một loại mạng nơ-ron chuyên biệt được thiết kế để xử lý dữ liệu dạng lưới, mà ảnh số là ví dụ điển hình (ma trận 2D chứa giá trị pixel). CNN tỏ ra vượt trội hơn mạng nơ-ron truyền thống (ANN – Artificial Neural Network) trong xử lý ảnh nhờ hai đặc điểm quan trọng:

- (1) Khả năng chia sẻ tham số (Parameter Sharing): Trong ANN, mỗi neuron kết nối với toàn bộ pixel đầu vào → số lượng tham số rất lớn, dẫn đến:

+ Dễ overfitting

+ Tốn nhiều bộ nhớ

+ Khó huấn luyện

Ngược lại, CNN sử dụng cùng một bộ lọc (kernel) quét trên toàn ảnh để trích xuất đặc trưng.

→ Mọi vị trí trong ảnh dùng chung tham số bộ lọc.

→ Giảm mạnh số lượng tham số, tăng tốc độ học và khả năng khái quát hóa.

- (2) Khả năng trích xuất đặc trưng cục bộ (Local Receptive Fields): CNN tập trung vào các vùng nhỏ (patch) trong ảnh, ví dụ 3×3 hoặc 5×5 pixel. Điều này phù hợp với thực tế:

+ Đặc trưng ảnh như cạnh (edge), góc (corner), hoặc texture thường mang tính cục bộ.

+ Mỗi lớp convolution học một loại đặc trưng nhất định:

+ Lớp đầu học cạnh và đường thẳng

+ Lớp giữa học hình dạng (shape)

+ Lớp sâu học đối tượng (object-level features)

+ Nhờ hai tính chất trên, CNN trở thành kiến trúc chuẩn trong các bài toán nhận dạng ảnh, phân loại, phát hiện đối tượng, phân đoạn ảnh,…

2.2.1.2. Kiến trúc chi tiết của một mạng CNN

Convolutional Neural Network (CNN) là một kiến trúc mạng nơ-ron chuyên biệt cho xử lý ảnh, được xây dựng từ nhiều lớp tích chập xếp chồng lên nhau kết hợp với các hàm kích hoạt phi tuyến như ReLU hoặc Tanh. Qua từng lớp, mạng học được các biểu diễn đặc trưng ngày càng trừu tượng hơn, giúp mô hình tự động phát hiện cạnh, họa tiết, hình dạng và các đặc trưng mức cao khác của đối tượng trong ảnh.

Khác với mạng nơ-ron truyền thống (Fully Connected Neural Network), nơi mỗi neuron kết nối với toàn bộ neuron của lớp trước, CNN chỉ kết nối cục bộ thông qua phép tích chập (convolution). Điều này giúp giảm mạnh số lượng tham số và cho phép mô hình tập trung học các đặc trưng trong từng vùng của ảnh. Các đặc trưng được học theo trình tự từ

thấp đến cao: pixel → cạnh → hình dạng → cấu trúc → đặc trưng ngữ nghĩa. Cuối cùng, các đặc trưng này được đưa vào lớp kết nối đầy đủ để thực hiện phân loại.

CNN thường bao gồm ba loại lớp chính:

a. Lớp tích chập (Convolutional Layer)

- Lớp tích chập là nền tảng của CNN, chịu trách nhiệm trích xuất đặc trưng từ ảnh đầu vào. Mỗi lớp tích chập sử dụng nhiều bộ lọc (filter/kernels) nhỏ, chẳng hạn 3×3 hoặc 5×5, trượt qua toàn bộ ảnh để tạo ra các bản đồ đặc trưng (feature maps).

- Ở lớp nông, mô hình học đặc trưng đơn giản như cạnh, góc, đường thẳng.

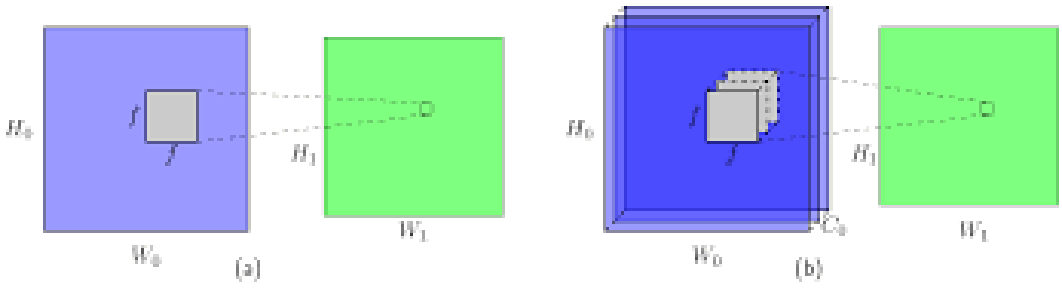
- Ở lớp sâu hơn, các filter học những đặc trưng phức tạp như đường cong, cấu trúc hình học hay một phần cơ thể người.

- Ở lớp rất sâu, CNN học được các đặc trưng trừu tượng như hình dạng tổng thể của người hoặc các pattern ngữ nghĩa liên quan đến đối tượng.

- Nhờ cơ chế kết nối cục bộ và chia sẻ tham số, conv layer hiệu quả hơn rất nhiều so với fully connected khi xử lý ảnh kích thước lớn.

$$H_1 = \frac{H_0 + 2P - f}{S} + 1$$

$$W_1 = \frac{W_0 + 2P - f}{S} + 1$$



Hình 2.2. Mô phỏng quá trình trượt bộ lọc trên ảnh đầu vào để tạo Feature Map.

b. Lớp gộp (Pooling Layer)

- Pooling là lớp dùng để giảm kích thước của feature map, giúp mô hình:

- Giảm số lượng tham số và chi phí tính toán

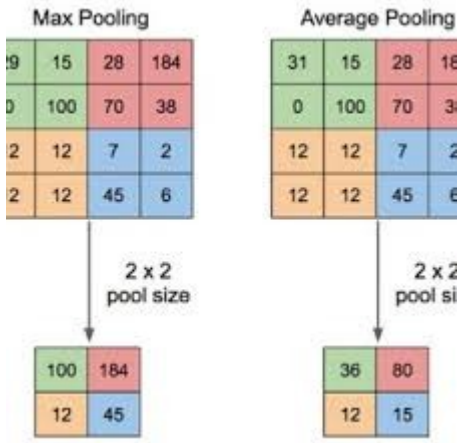
- Giữ lại đặc trưng quan trọng

- Tăng tính bất biến khi ảnh bị dịch chuyển, xoay nhẹ hoặc nhiễu

- Loại phổ biến nhất là Max Pooling, lấy giá trị lớn nhất trong mỗi vùng nhỏ (ví dụ 2×2).

- Điều này giúp mô hình giữ lại các tín hiệu mạnh nhất trong feature map và loại bỏ nhiễu không cần thiết.

- Ví dụ: một vùng 2×2 trong feature map sẽ được rút gọn thành 1 giá trị duy nhất — giá trị quan trọng nhất của vùng đó.



Hình 2.3. Minh họa hoạt động của lớp Pooling giúp giảm chiều dữ liệu.

c. Lớp kết nối đầy đủ (Fully Connected Layer)

- Sau khi trải qua nhiều lớp convolution và pooling, ảnh ban đầu được biến đổi thành một vector đặc trưng giàu thông tin.

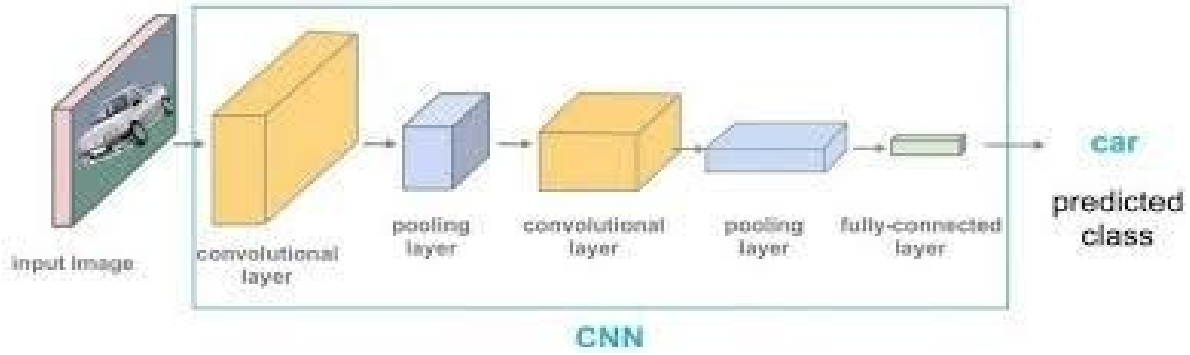
- Lớp Fully Connected (FC) sẽ:

+ Nhận toàn bộ vector đặc trưng

+ Kết hợp các đặc trưng lại với nhau

+ Đưa ra dự đoán cuối cùng

Trong bài toán Person Detection, lớp FC cuối cùng thường có 1 neuron với hàm kích hoạt Sigmoid, trả về một giá trị từ 0 đến 1 biểu thị xác suất ảnh có chứa người.



Hình 2.4. Minh họa luồng xử lý đầy đủ của một mạng CNN.

2.2.1.3. Các siêu tham số quan trọng

(1) Kích thước bộ lọc – Kernel/Filter Size

- Thường dùng: 3×3 (phổ biến nhất), 5×5

- Kernel nhỏ giúp:

+ Giảm số tham số

+ Trích xuất đặc trưng tinh hơn

+Kernel lớn (7×7, 11×11) thường ít dùng trong các mô hình nhỏ.

(2) Stride (Bước nhảy)

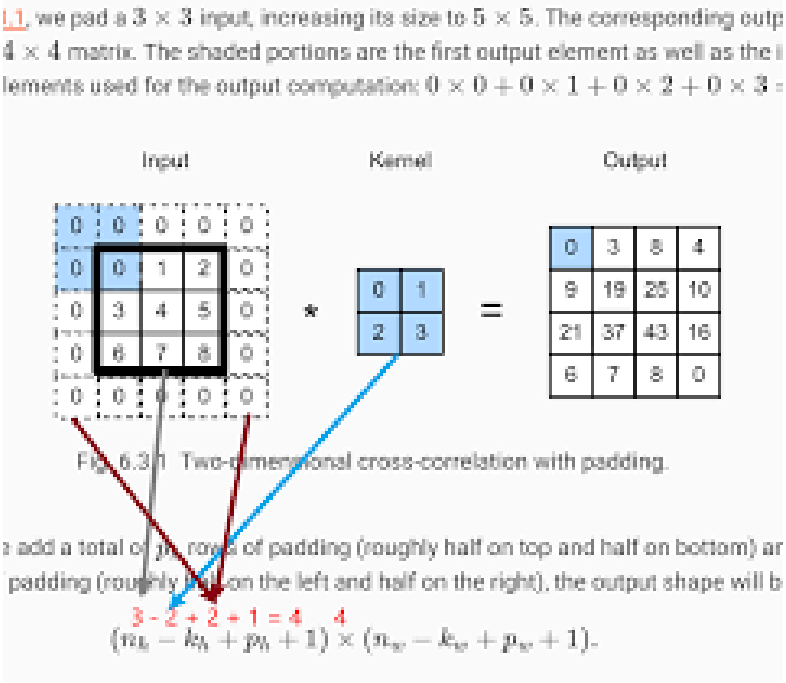
- Mặc định: 1

-Stride lớn (2, 3) → giảm kích thước feature map → mô hình chạy nhanh hơn nhưng dễ mất thông tin.

(3) Padding (Đệm viền ảnh)

- Valid padding: không thêm viền, làm giảm kích thước sau từng lớp.

- Same padding: thêm viền để giữ nguyên kích thước đầu ra.



Hình 2.5. Padding và Stride.

2.2.1.4. Các hàm kích hoạt (Activation Function) trong bài

Trong mô hình CNN, hàm kích hoạt đóng vai trò thêm tính phi tuyến (non-linearity) vào mạng. Điều này giúp mô hình có khả năng học các quan hệ phức tạp trong dữ liệu thay vì chỉ là các phép biến đổi tuyến tính. Trong bài toán Person Detection, hai hàm kích hoạt chính được sử dụng là ReLU và Sigmoid, mỗi hàm được dùng cho mục đích khác nhau.

a, ReLU (Rectified Linear Unit)

Công thức:

$$f(x)=\max(0,x)$$

- Vai trò và ưu điểm:

- Dùng cho các lớp ẩn (hidden layers) trong mô hình.

- Giúp tránh hiện tượng triệt tiêu đạo hàm (vanishing gradient) – một vấn đề thường gặp khi dùng Tanh hoặc Sigmoid, khiến việc học diễn ra chậm hoặc không học được ở các lớp sâu.

- Tăng tốc độ hội tụ: ReLU cho gradient lớn và ổn định hơn, giúp mô hình học nhanh hơn trong quá trình huấn luyện.

- Tính đơn giản: chỉ lấy phần dương → giúp tiết kiệm chi phí tính toán.

b, Sigmoid

- Công thức:

$$\sigma(x)=\frac{1}{1+e^{-x}}$$

- Vai trò

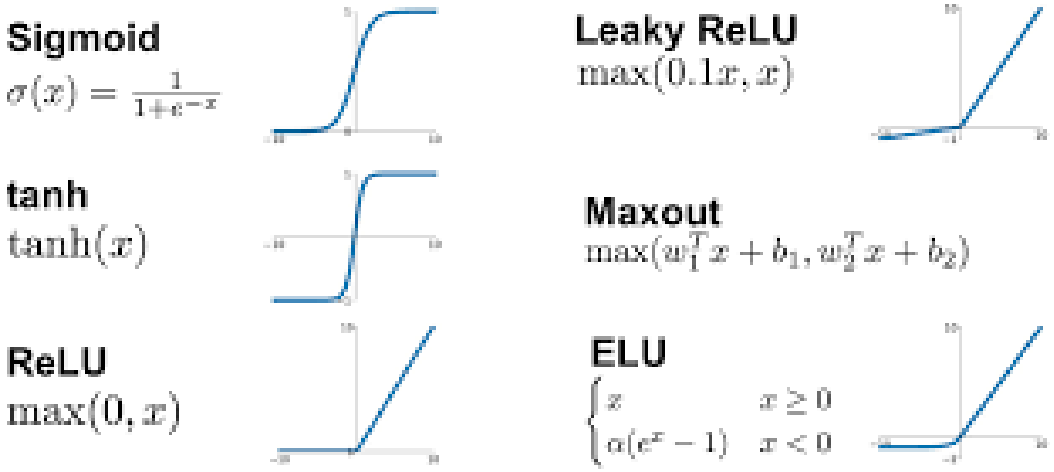
- Dùng cho lớp Output của mô hình.

- Chuyển đầu ra thành một xác suất nằm trong khoảng (0, 1).

- Ý nghĩa trong bài toán Person Detection

- Nếu đầu ra > 0.5 → mô hình dự đoán lớp 1 (Person)

- Nếu đầu ra ≤ 0.5 → mô hình dự đoán lớp 0 (Non-person)



Hình 2.6. Đồ thị biểu diễn hàm kích hoạt ReLU và Sigmoid.

2.2.1.5. Hàm mất mát và Tối ưu hóa

a, Hàm mất mát: Binary Cross-Entropy (BCE)

BCE đo mức độ khác biệt giữa xác suất dự đoán và nhãn thật trong bài toán phân loại nhị phân.

Công thức:

$$BCE=-[y\log(p)+(1-y)\log(1-p)]$$

- Trong đó:

+ $y \in \{0,1\}$ là nhãn thật

+ $p$là xác suất mô hình dự đoán (đầu ra của Sigmoid)

- Ý nghĩa: Khi dự đoán đúng (p gần y), giá trị BCE nhỏ → mô hình "phạt ít". Khi dự đoán sai (p xa y), BCE lớn → mô hình "phạt mạnh".

b, Bộ tối ưu hóa: Adam (Adaptive Moment Estimation)

Adam là một thuật toán tối ưu dựa trên gradient, kết hợp ưu điểm của hai kỹ thuật phổ biến:

Momentum: giúp mô hình di chuyển "mượt" hơn trong không gian tham số.

RMSProp: điều chỉnh learning rate dựa trên độ lớn gradient theo từng tham số.

Ưu điểm của Adam

Tự động điều chỉnh learning rate cho từng trọng số, giúp mô hình học ổn định hơn.

Hội tụ nhanh, phù hợp với bài toán có dữ liệu đa dạng và nhiều nhiễu.

Rất hiệu quả với các mô hình nhỏ hoặc trung bình như LeNet cải tiến.

Ít cần tinh chỉnh thủ công → phù hợp trong bối cảnh học thuật và thực nghiệm.

Vì vậy, Adam được lựa chọn vì tốc độ nhanh, ổn định, ít phụ thuộc vào tuning, và hoạt động tốt trên dữ liệu ảnh có độ biến thiên cao.

CHƯƠNG III: DỮ LIỆU VÀ TIỀN XỬ LÝ

3.1. Bộ dữ liệu sử dụng (Natural Images)

Bộ dữ liệu chính được sử dụng là Natural Images (có thể là một tập con hoặc biến thể của các bộ dữ liệu ảnh tự nhiên lớn hơn như ImageNet hoặc Open Images, được tinh chỉnh cho bài toán nhận diện khuôn mặt).

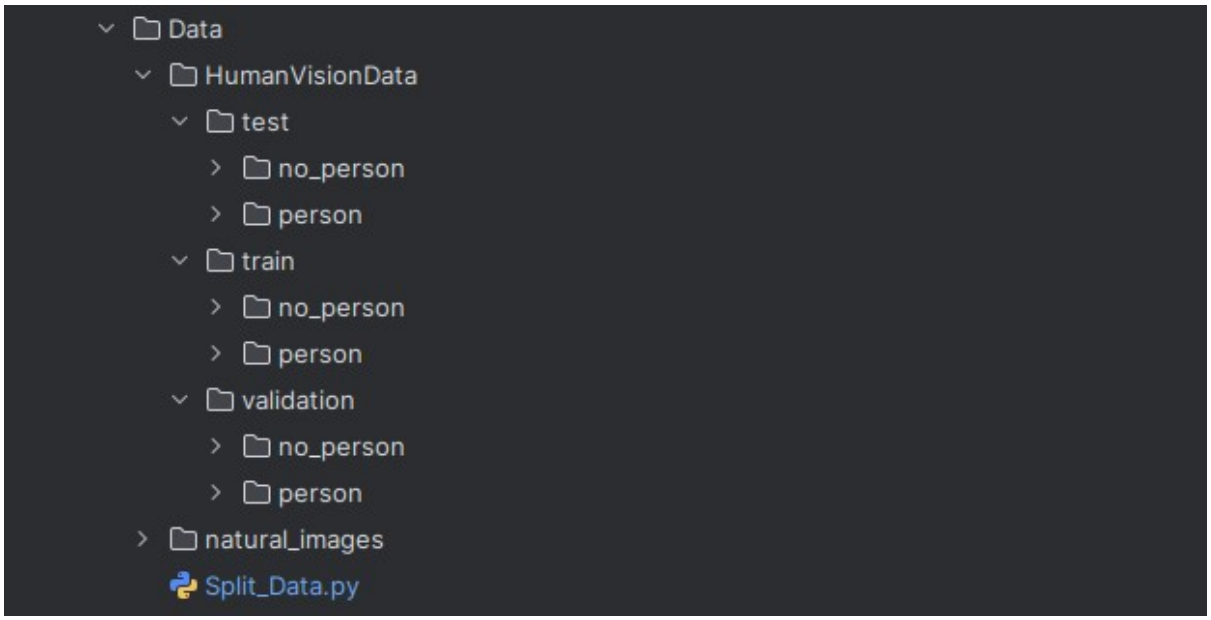
Mô tả: Bộ dữ liệu bao gồm các hình ảnh thực tế, đa dạng về bối cảnh, ánh sáng, góc chụp và độ phân giải, được phân thành hai lớp chính: person: Chứa các hình ảnh có khuôn mặt người (hoặc người) xuất hiện. no_person: Chứa các hình ảnh không có khuôn mặt người (hoặc người) xuất hiện.

Mục đích: Đảm bảo mô hình CNN học được các đặc trưng phong phú và có khả năng phân biệt hiệu quả giữa hai lớp đối tượng trong môi trường thực tế.

3.2. Chia tập & Cấu trúc thư mục

Dữ liệu được chia thành ba tập con Train, Validation và Test với tỷ lệ đã xác định để phục vụ cho các giai đoạn khác nhau của quá trình phát triển mô hình.

Cấu trúc thư mục



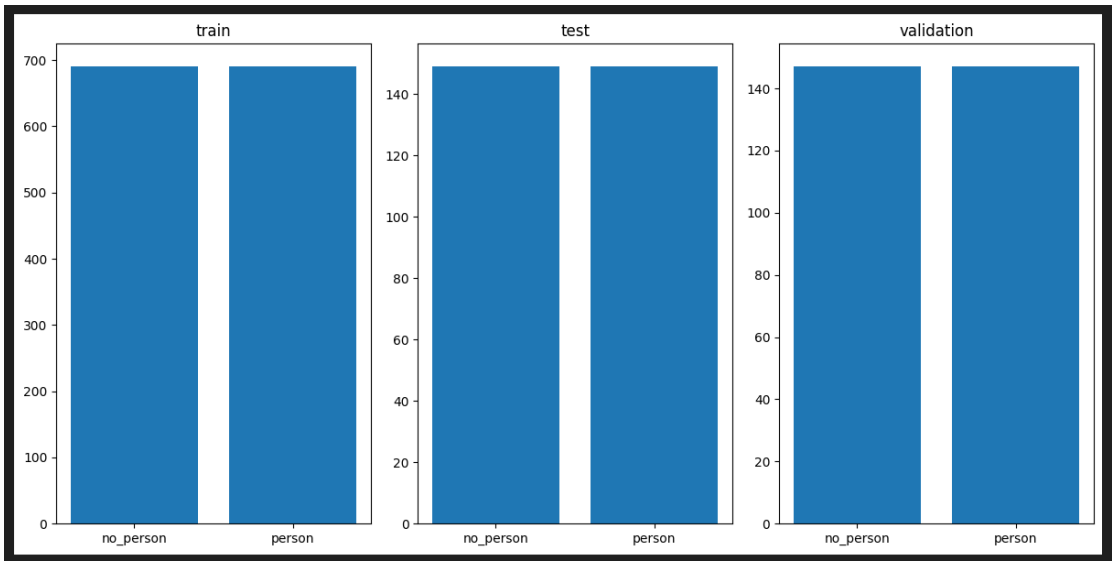
Hình 3.1. Cấu trúc thư mục tổ chức dữ liệu.

Bảng 3.1. Thống kê phân bố số lượng ảnh trong các tập dữ liệu.

Tập dữ liệu	Lớp person(Có khuôn mặt	Lớp no_person(Không khuôn mặt)	Tổng số mẫu	Tỷ lệ(Gần đúng)
Train	690	690	1380	70%
Validation	147	147	294	15%
Test	149	149	298	15%

- Tổng cộng: 1380 + 294 + 298 = 1972 hình ảnh.

- Đánh giá: Sự phân chia này đảm bảo tính cân bằng giữa các lớp (person và no_person) trong mỗi tập, tránh hiện tượng mô hình thiên vị một lớp nào đó.



Hình 3.2. Biểu đồ cân bằng dữ liệu giữa hai lớp Person và No-person.

3.3. Tiền xử lý và Tăng cường dữ liệu

Tiền xử lý là bước bắt buộc để đưa dữ liệu đầu vào về định dạng tiêu chuẩn mà mô hình học sâu có thể xử lý hiệu quả.

3.3.1. Chuẩn hoá kích thước (Resizing)

Mô tả: Tất cả các hình ảnh trong ba tập Train, Validation, Test đều được điều chỉnh về cùng một kích thước cố định, ví dụ: 224 x 224 pixels (tùy thuộc vào kiến trúc CNN được chọn).

Mục đích: Đảm bảo đầu vào của lớp đầu tiên trong mạng CNN là đồng nhất, giúp mô hình hoạt động ổn định và có thể thực hiện phép toán ma trận chính xác.

3.3.2. Chuẩn hoá giá trị Pixel (Normalization)

Mô tả: Giá trị pixel gốc (thường nằm trong khoảng 0-255) được chuẩn hóa về khoảng [0, 1] hoặc về phân phối chuẩn (mean=0, std=1).

Mục đích: Giúp tốc độ học của mô hình nhanh hơn và ổn định hơn. Các giá trị nhỏ hơn giúp thuật toán tối ưu hóa (như Gradient Descent) hội tụ nhanh chóng, giảm thiểu lỗi số học.

3.3.3. Tăng cường (Augmentation) cho tập Train

Kỹ thuật tăng cường dữ liệu chỉ được áp dụng cho tập Train để mở rộng bộ dữ liệu một cách nhân tạo, giúp mô hình tăng cường khả năng tổng quát hóa và chống lại hiện tượng quá khớp (overfitting).

Các kỹ thuật áp dụng:

- Xoay ngẫu nhiên (Random Rotation): Xoay ảnh trong một phạm vi nhỏ (ví dụ: ± 10 độ).

- Lật ngang ngẫu nhiên (Horizontal Flip): Giúp mô hình nhận dạng khuôn mặt độc lập với chiều ngang.

- Cắt xén ngẫu nhiên (Random Crop/Zoom): Phóng to hoặc cắt một phần ngẫu nhiên của ảnh.

- Thay đổi độ sáng/độ tương phản (Brightness/Contrast Adjustment): Giúp mô hình ít nhạy cảm hơn với điều kiện ánh sáng khác nhau.

3.4. Thống kê sau Tiền xử lý

Sau khi áp dụng các bước tiền xử lý, dữ liệu có các thông số kỹ thuật đầu vào đồng nhất:

Kích thước ảnh: 224 x 224 x 3 (chiều rộng x chiều cao x số kênh màu RGB).

Dải giá trị pixel: [0, 1] (hoặc theo phân phối chuẩn).

Số lượng mẫu: Giữ nguyên số lượng mẫu gốc trong các tập Validation và Test. Số lượng mẫu trong tập Train tăng lên đáng kể (về mặt hiệu quả học tập) do áp dụng Augmentation.

Code:

```
img_size = 150 # img.shape[0] = 227
output_size = 1 #Use sigmoid function
batch_size = 256
# Đây là phần tăng cường dữ liệu cho tập huấn luyện
train_datagen = ImageDataGenerator( rescale=1.0/255.0,
                                    rotation_range=30,
                                    zoom_range=0.15,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    shear_range=0.15,
                                    horizontal_flip=True,
                                    fill_mode="nearest" )
# Dữ liệu validation và test chỉ được rescale, không tăng cường
val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
# Chuẩn bị các iterators để tải dữ liệu
train_dataloader = train_datagen.flow_from_directory(train_data,
                                                     class_mode='binary',
                                                     batch_size=batch_size,
```

```
                                                target_size=(img_size, img_size))
    valid_dataloader = val_datagen.flow_from_directory(valid_data,
                                        class_mode='binary',
                                        batch_size=batch_size,
                                        target_size=(img_size, img_size))
    test_dataloader = test_datagen.flow_from_directory(test_data,
                                         class_mode='binary',
                                         batch_size=batch_size,
                                         target_size=(img_size, img_size))
```

3.5. Nhận xét

Việc lựa chọn và tiền xử lý dữ liệu đóng vai trò quan trọng trong thành công của hệ thống nhận diện khuôn mặt:

Chất lượng dữ liệu: Bộ dữ liệu Natural Images với sự đa dạng bối cảnh giúp mô hình học các đặc trưng mạnh mẽ.

Tính cân bằng: Sự cân bằng mẫu giữa hai lớp person và no_person trong tất cả các tập giúp mô hình tránh thiên vị và cải thiện độ chính xác tổng thể.

Vai trò của Augmentation: Tăng cường dữ liệu là chìa khóa để chống quá khớp, đặc biệt khi quy mô tập dữ liệu ban đầu còn hạn chế, làm cho mô hình nhận diện khuôn mặt có khả năng tổng quát hóa tốt hơn với các biến thể trong thực tế (góc nhìn, ánh sáng, kích thước).

Chương IV: Thiết kế và huấn luyện mô hình

4.1. Cấu trúc mô hình LeNet

Mô hình được sử dụng là một biến thể của kiến trúc LeNet-5, được xây dựng bằng thư viện Keras của TensorFlow. Mô hình này được thiết kế để thực hiện bài toán phân loại nhị phân (có người hoặc không có người) trên các ảnh màu có kích thước 150x150 pixels.

Cấu trúc chi tiết của mô hình bao gồm các lớp sau:

- Lớp tích chập (Conv2D):

+ Số bộ lọc: 6

+ Kích thước kernel: (5, 5)

+ Hàm kích hoạt: ReLU

+ Input shape: (150, 150, 3)

- Lớp gộp trung bình (AveragePooling2D):

+ Kích thước pool: (2, 2)

+ Strides: 2

- Lớp tích chập (Conv2D):

+ Số bộ lọc: 16

+ Kích thước kernel: (5, 5)

+ Hàm kích hoạt: ReLU

- Lớp gộp trung bình (AveragePooling2D):

+ Kích thước pool: (2, 2)

+ Strides: 2

+ Lớp duỗi thẳng (Flatten): Chuyển đổi dữ liệu từ dạng ma trận 2D thành vector 1D để cung cấp cho các lớp kết nối đầy đủ.

+ Lớp kết nối đầy đủ (Dense):

+ Số units: 120

+ Hàm kích hoạt: ReLU

- Lớp kết nối đầy đủ (Dense):

+ Số units: 84

+ Hàm kích hoạt: ReLU

- Lớp đầu ra (Dense):

+ Số units: 1

+ Hàm kích hoạt: Sigmoid (phù hợp cho bài toán phân loại nhị phân)

Tổng số tham số của mô hình: 2,232,761.

4.2. Cấu hình huấn luyện

- Tập dữ liệu: Dữ liệu được chia thành ba tập:

+ Huấn luyện (Train): 1380 ảnh

+ Kiểm định (Validation): 294 ảnh

+ Kiểm tra (Test): 298 ảnh

- Tăng cường dữ liệu (Data Augmentation): Chỉ áp dụng cho tập huấn luyện để tăng sự đa dạng và giảm thiểu overfitting, bao gồm các phép biến đổi:

+ Chuẩn hóa giá trị pixel về khoảng [0, 1] (rescale=1.0/255.0).

+ Xoay ảnh ngẫu nhiên trong khoảng 30 độ.

+ Phóng to ảnh ngẫu nhiên (zoom_range=0.15).

+ Dịch chuyển ảnh theo chiều rộng và chiều cao (width/height_shift_range=0.2).

+ Biến dạng ảnh (shear_range=0.15).

+ Lật ảnh theo chiều ngang.

- Siêu tham số (Hyperparameters):

+ Kích thước ảnh: 150x150 pixels.

+ Kích thước lô (Batch size): 256.

+ Số epochs: 10.

+ Trình tối ưu hóa (Optimizer): Adam.

+ Hàm mất mát (Loss function): binary_crossentropy.

+ Thước đo (Metrics): accuracy.

Code:

```
# Biên dịch mô hình với optimizer, hàm loss và metrics
lenet_model.compile(optimizer='adam',loss="binary_crossentropy",
metrics=['accuracy'])
lenet_model.summary()
# Số lượng epochs để huấn luyện
num_epochs = 10
# Đường dẫn để lưu trọng số của mô hình tốt nhất
checkpoint_path =
"/content/save_weight/human_prediction(2)_.weights.h5"
# Callback để lưu lại mô hình có val_accuracy tốt nhất
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
        filepath=checkpoint_path,
        verbose=1,
        save_weights_only=True,
        monitor='val_accuracy',
        mode='max',
        save_best_only=True)
# Callback để giảm learning rate khi val_accuracy không cải thiện
learning_rate_reduction =
tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                     patience = 2,
                                     verbose=1,
                                     factor=0.5,
                                     min_lr=0.00001)
# Bắt đầu quá trình huấn luyện
history = lenet_model.fit(train_dataloader,
                  epochs = num_epochs,
                  validation_data = valid_dataloader,
```

```
callbacks=[model_checkpoint, learning_rate_reduction])
```

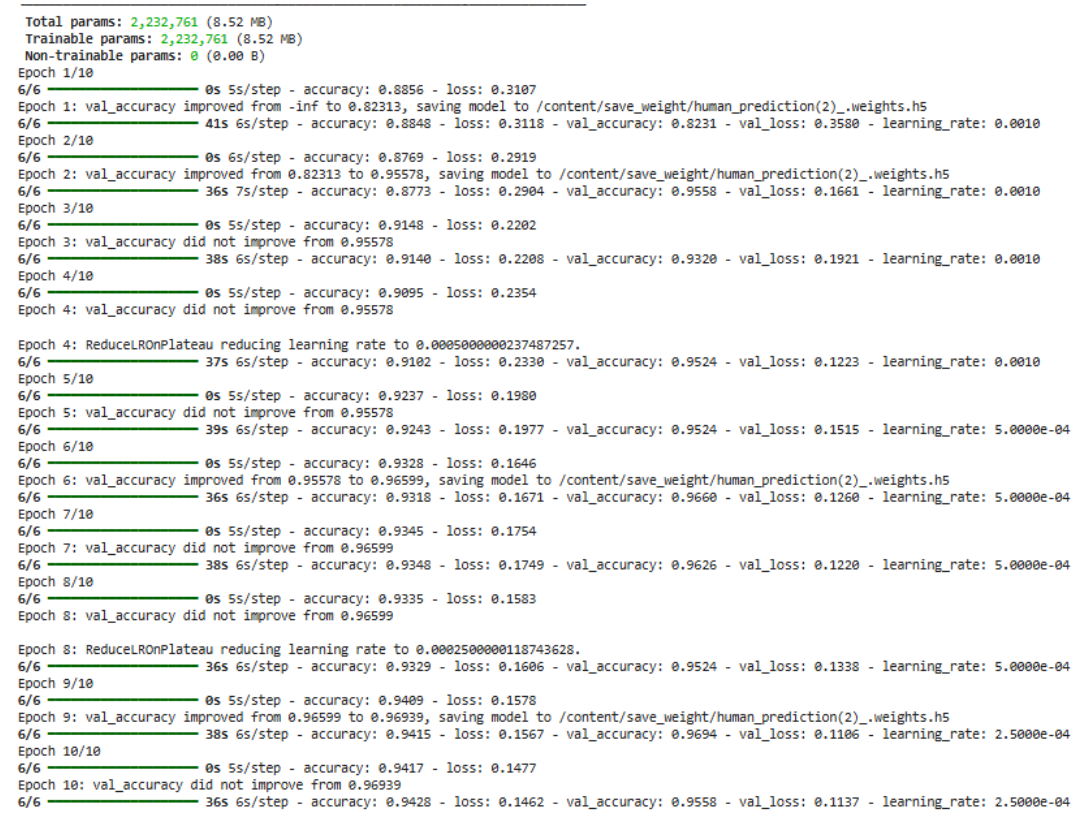
4.3. Quá trình huấn luyện

Mô hình được huấn luyện trong 10 epochs. Trong quá trình huấn luyện, hai callbacks đã được sử dụng để tối ưu hóa hiệu suất và lưu lại kết quả tốt nhất:

ModelCheckpoint: Callback này theo dõi độ chính xác trên tập kiểm định (val_accuracy). Nó sẽ tự động lưu lại trọng số của mô hình mỗi khi độ chính xác này đạt một giá trị cao mới. Trọng số tốt nhất được lưu vào tệp human_prediction(2)_.weights.h5.

ReduceLROnPlateau: Callback này giúp điều chỉnh tốc độ học (learning rate). Nếu val_accuracy không được cải thiện sau 2 epochs liên tiếp, tốc độ học sẽ được giảm đi một nửa (factor=0.5). Trong quá trình huấn luyện, tốc độ học đã được giảm 2 lần, cho thấy cơ chế này đã hoạt động để giúp mô hình hội tụ tốt hơn.

Quá trình huấn luyện cho thấy độ chính xác trên cả tập huấn luyện và tập kiểm định đều tăng dần, trong khi hàm mất mát giảm, cho thấy mô hình đã học được các đặc trưng quan trọng từ dữ liệu.



Hình 4.1. Nhật ký quá trình huấn luyện mô hình trên Google Colab.

4.4. Đánh giá và nhận xét

Hiệu suất: Mô hình đạt được độ chính xác rất cao trên tập kiểm định, với giá trị tốt nhất lên đến 96.94% ở epoch thứ 9. Điều này cho thấy mô hình có khả năng phân loại tốt giữa ảnh có người và không có người.

Độ ổn định: Mặc dù có một vài biến động nhỏ, val_accuracy nhìn chung có xu hướng tăng ổn định trong suốt quá trình huấn luyện.

Kiểm soát Overfitting: Việc sử dụng data augmentation và callback ReduceLROnPlateau đã tỏ ra hiệu quả trong việc giúp mô hình tổng quát hóa tốt hơn và hạn chế hiện tượng overfitting.

4.5. Nhận xét kết quả

Kết quả huấn luyện rất tích cực. Mô hình LeNet, mặc dù có kiến trúc tương đối đơn giản so với các mô hình hiện đại, đã chứng tỏ hiệu quả cao cho tác vụ này. Độ chính xác đạt được là một minh chứng cho thấy: Việc lựa chọn kiến trúc mô hình là phù hợp.

Quá trình tiền xử lý và tăng cường dữ liệu đã được thực hiện tốt, giúp mô hình học hiệu quả.

Các siêu tham số được cấu hình hợp lý.

Mô hình sau khi huấn luyện đã được lưu lại trọng số tốt nhất, sẵn sàng cho việc tích hợp vào các ứng dụng thực tế như nhận diện người qua webcam hoặc phân loại ảnh tự động.

Chương V: Serving Model

5.1. Mục tiêu

Mục tiêu của chương này là triển khai mô hình LeNet đã được huấn luyện thành một dịch vụ web (web service). Dịch vụ này sẽ cung cấp một giao diện lập trình ứng dụng (API) cho phép các ứng dụng bên ngoài (ví dụ: ứng dụng web, di động) gửi một hình ảnh và nhận lại kết quả dự đoán là "có người" (person) hay "không có người" (no_person). Việc "serving" mô hình giúp tách biệt logic AI khỏi ứng dụng người dùng cuối và cho phép mô hình được tái sử dụng một cách hiệu quả.

5.2. Cấu trúc dự án

Để phục vụ mô hình, dự án được tổ chức trong thư mục app/ với cấu trúc rõ ràng, tách biệt các thành phần:

```
app/
├──main.py            # Điểm khởi đầu của ứng dụng FastAPI
├──api/
│   ├──api.py         # Định nghĩa các API endpoints (ví dụ: /predict)
│   └──__init__.py
└──model/
    ├──lenet.py       # Định nghĩa lớp, tạo kiến trúc và tải trọng số mô hình
    ├──__init__.py
    └──weights/
        └──saved_models/
            └──human_prediction(2)_.weights.h5 # File trọng số đã huấn luyện
```

- main.py: Khởi tạo server và các cấu hình chung.

- api/api.py: Chứa logic xử lý các yêu cầu HTTP, nhận dữ liệu, gọi mô hình và trả về kết quả.

- model/lenet.py: Chịu trách nhiệm tái tạo kiến trúc mô hình LeNet và tải các trọng số đã được huấn luyện từ tệp .h5.

- model/weights/saved_models/: Thư mục chứa các tệp trọng số đã được lưu lại từ quá trình huấn luyện.

5.3. Tạo API sử dụng FastAPI

Dự án sử dụng FastAPI, một web framework hiện đại của Python, để xây dựng API. FastAPI được chọn vì hiệu năng cao, khả năng xử lý bất đồng bộ, và tính năng tự động tạo tài liệu API (documentation).

Tệp app/main.py là nơi ứng dụng FastAPI được khởi tạo:

- Nó tạo một đối tượng FastAPI với tiêu đề là "Human Vision Prediction".

- Nó cấu hình CORSMiddleware để cho phép các yêu cầu từ mọi nguồn (cần thiết khi phát triển giao diện người dùng trên một domain khác).

- Quan trọng nhất, nó sử dụng app.include_router() để tích hợp các API endpoint được định nghĩa trong tệp app/api/api.py vào ứng dụng chính với tiền tố (prefix) là /api.

Cấu trúc API

Logic chính của hệ thống được cài đặt tại tập tin app/api/api.py với cấu trúc như sau:

- Thông tin Endpoint:
    + URL: /api/predict
    + Phương thức: POST
    + Đầu vào (Input): Tệp hình ảnh được gửi dưới dạng multipart/form-data (UploadFile).
- Quy trình xử lý dữ liệu:
    1. Tiếp nhận và Xác thực: Hệ thống đọc dữ liệu từ request và xác thực tệp tin tải lên đúng định dạng hình ảnh.

2. Tiền xử lý (Preprocessing): Ảnh đầu vào trải qua các bước biến đổi để phù hợp với yêu cầu của mô hình:
    - Chuyển đổi không gian màu sang RGB (sử dụng thư viện PIL).
    - Thay đổi kích thước (Resize) về 150x150 pixels.
    - Chuẩn hóa dữ liệu (Normalization): Chuyển ảnh thành mảng NumPy và đưa giá trị pixel về khoảng [0, 1].
    - Định hình lại (Reshape): Mở rộng chiều của mảng để tạo thành một batch có kích thước (1, 150, 150, 3).
3. Dự đoán (Inference): Gọi phương thức predict() của mô hình đã huấn luyện trên dữ liệu ảnh vừa xử lý.
4. Hậu xử lý (Post-processing):
    - Kết quả trả về là điểm tin cậy (confidence score) trong khoảng [0, 1].
    - Ngưỡng phân loại: Nếu score > 0.5 gán nhãn "person", ngược lại gán nhãn "no_person".
5. Phản hồi (Response): Trả về client định dạng JSON chứa nhãn dự đoán (prediction) và độ tin cậy (confidence).

```python
from api.api import api_router
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

# Khởi tạo ứng dụng FastAPI
app = FastAPI(title='Human Vision Prediction', version='1.0.0')

# Cấu hình CORSMiddleware để cho phép truy cập từ các domain khác
app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=['*'],
    allow_headers=['*'],
)

# Tích hợp các API endpoints từ tệp api.py với tiền tố /api
app.include_router(api_router, prefix='/api')
```

Hình 5.1. Mã nguồn khởi tạo ứng dụng FastAPI.

5.4. Tạo mô hình LeNet

Việc tạo và tải mô hình được thực hiện trong tệp app/model/lenet.py để đảm bảo mô hình chỉ được tải một lần khi ứng dụng khởi động, giúp tiết kiệm tài nguyên và tăng tốc độ xử lý request.

Lớp HumanVisionModel: Một lớp được tạo ra để đóng gói tất cả logic liên quan đến mô hình.

Khởi tạo kiến trúc: Trong phương thức __init__, kiến trúc của mô hình LeNet được định nghĩa lại giống hệt như kiến trúc đã sử dụng trong quá trình huấn luyện. Điều này là bắt buộc vì tệp .h5 chỉ lưu trọng số chứ không lưu kiến trúc.

Tải trọng số: Sau khi tạo kiến trúc, phương thức load_weights() được gọi để tải các tham số đã được huấn luyện từ tệp app/model/weights/saved_models/human_prediction(2)_.weights.h5 v ào mô hình.

Đối tượng Singleton: Một đối tượng duy nhất của lớp HumanVisionModel được tạo ở cuối tệp: human_vision_model =

HumanVisionModel(). Đối tượng này sau đó được import và sử dụng trong api.py. Nhờ vậy, mô hình được tải vào bộ nhớ chỉ một lần và được tái sử dụng cho tất cả các yêu cầu dự đoán tiếp theo.

```python
class HumanVisionModel:
    def __init__(self):
        # Khởi tạo kiến trúc LeNet giống hệt lúc huấn luyện
        self.model = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(150, 150, 3)),
            tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
            tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'),
            tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(units=120, activation='relu'),
            tf.keras.layers.Dense(units=84, activation='relu'),
            tf.keras.layers.Dense(units=1, activation='sigmoid')
        ])

        try:
            # Đường dẫn đến file trọng số
            base_dir = os.path.dirname(os.path.abspath(__file__))
            weights_path = os.path.join(base_dir, 'weights', 'saved_models', 'human_prediction(2)_.weights.h5')

            # Tải trọng số đã huấn luyện
            self.model.load_weights(weights_path)
            print(f"✅ Tải trọng số thành công từ: {weights_path}")
        except Exception as e:
            print(f"❌ Lỗi khi tải trọng số: {e}")

    def predict(self, img: np.ndarray):
        """
        Hàm dự đoán trên một ảnh đã được tiền xử lý.
        """
        return self.model.predict(img)

# Tạo một đối tượng Singleton để được import và tái sử dụng
human_vision_model = HumanVisionModel()
```

Hình 5.2. Lớp HumanVisionModel định nghĩa và tải trọng số mô hình.

5.5. Triển khai mô hình trên Server

Để triển khai ứng dụng FastAPI, một máy chủ ASGI (Asynchronous Server Gateway Interface) là cần thiết. Dự án này sử dụng Uvicorn, một máy chủ ASGI cực nhanh.

Trong tệp app/main.py, đoạn mã sau được dùng để khởi chạy server cho mục đích phát triển:

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)
```

- uvicorn.run("main:app", ...): Lệnh này yêu cầu Uvicorn chạy ứng dụng app bên trong tệp main.py.

- host="0.0.0.0": Cho phép server có thể được truy cập từ bất kỳ địa chỉ IP nào, không chỉ localhost.

- port=8000: Server sẽ lắng nghe các yêu cầu trên cổng 8000.

- reload=True: Tự động khởi động lại server mỗi khi có thay đổi trong mã nguồn (hữu ích cho việc phát triển).

Khi server khởi động, nó sẽ tải ứng dụng FastAPI, ứng dụng này sẽ import và khởi tạo mô hình LeNet. Từ đó, server sẵn sàng nhận các yêuTệp api.py chứa logic cốt lõi của API. Nó định nghĩa một endpoint POST /predict nhận một tệp ảnh, xử lý trước và sau đó sử dụng human_vision_model để đưa ra dự đoán.

Chương VI: Thiết kế hệ thống giao diện trực quan

6.1. Công cụ được lựa chọn

Để tạo giao diện người dùng (UI) một cách nhanh chóng và hiệu quả, dự án đã lựa chọn Streamlit. Streamlit là một framework mã nguồn mở của Python, được thiết kế đặc biệt để xây dựng và chia sẻ các ứng dụng web cho khoa học dữ liệu và học máy.

Lý do lựa chọn Streamlit:

- Đơn giản và nhanh chóng: Cho phép xây dựng một giao diện web tương tác chỉ bằng vài dòng mã Python mà không cần kiến thức về HTML, CSS, hay JavaScript.

- Tập trung vào dữ liệu: Cung cấp các thành phần (widgets) được tối ưu hóa cho việc trực quan hóa dữ liệu và tương tác với mô hình AI, ví dụ như tải tệp lên, hiển thị ảnh, và các nút bấm.

- Tương thích cao: Tích hợp liền mạch với các thư viện khoa học dữ liệu phổ biến như-là NumPy, Pandas, và Pillow (PIL), những thư viện đã được sử dụng trong dự án.

6.2. Các tính năng chính của Streamlit

Streamlit biến các tập lệnh Python thành các ứng dụng web tương tác. Các tính năng nổi bật được sử dụng trong dự án này bao gồm:

Widget tương tác:

- st.title() và st.write(): Dễ dàng thêm văn bản, tiêu đề và các mô tả vào ứng dụng.

- st.file_uploader(): Cung cấp một giao diện cho phép người dùng kéo-thả hoặc duyệt để tải lên tệp hình ảnh từ máy tính.

- st.image(): Hiển thị hình ảnh đã được tải lên một cách trực quan.

- st.button(): Tạo một nút bấm để kích hoạt hành động dự đoán.

- st.spinner(): Hiển thị một thông báo chờ trong khi ứng dụng đang xử lý (ví dụ: gửi yêu cầu đến API).

- st.success() và st.error(): Hiển thị kết quả thành công hoặc thông báo lỗi với định dạng màu sắc dễ nhận biết.

- Luồng thực thi đơn giản: Streamlit chạy lại toàn bộ kịch bản từ trên xuống dưới mỗi khi người dùng tương tác với một widget, giúp việc quản lý trạng thái trở nên đơn giản.

6.3. Triển khai ứng dụng

Ứng dụng giao diện được triển khai trong tệp streamlit_app/app.py. Điểm đáng chú ý trong kiến trúc này là sự tách biệt rõ ràng giữa giao diện người dùng (front-end) và dịch vụ mô hình (back-end).

Kiến trúc Client-Server:

- Client (Front-end): Ứng dụng Streamlit đóng vai trò là client. Nó không chứa logic của mô hình AI mà chỉ chịu trách nhiệm về giao diện và tương tác với người dùng.

- Server (Back-end): Ứng dụng FastAPI (đã mô tả ở Chương V) đóng vai trò là server, nơi mô hình được tải và thực hiện các phép tính dự đoán.

Luồng hoạt động của ứng dụng:

1, Khởi tạo giao diện: Ứng dụng hiển thị tiêu đề "Human Presence Detector" và một widget để người dùng tải ảnh lên.

2. Tải ảnh lên: Người dùng chọn một tệp ảnh (PNG, JPG, hoặc JPEG).

3. Hiển thị ảnh và nút bấm: Ngay sau khi ảnh được tải lên, ứng dụng sẽ hiển thị ảnh đó trên trang cùng với một nút "Predict".

4. Gửi yêu cầu dự đoán: Khi người dùng nhấp vào nút "Predict":

- Ứng dụng hiển thị một vòng quay chờ (spinner).

- Nó sử dụng thư viện requests của Python để gửi một yêu cầu POST đến endpoint của FastAPI: http://localhost:8000/api/predict.

- Tệp hình ảnh được đính kèm trong body của request.

5. Nhận và hiển thị kết quả:

- Ứng dụng chờ phản hồi từ API.

- Nếu nhận được mã trạng thái 200 (thành công), nó sẽ phân tích phản hồi JSON để lấy nhãn dự đoán (prediction) và độ tin cậy (confidence).

- Kết quả được hiển thị trong một hộp thông báo thành công, ví dụ: "Prediction: PERSON (Confidence: 0.98)".

- Nếu có lỗi xảy ra (ví dụ: không kết nối được API, API trả về lỗi), một thông báo lỗi chi tiết sẽ được hiển thị.

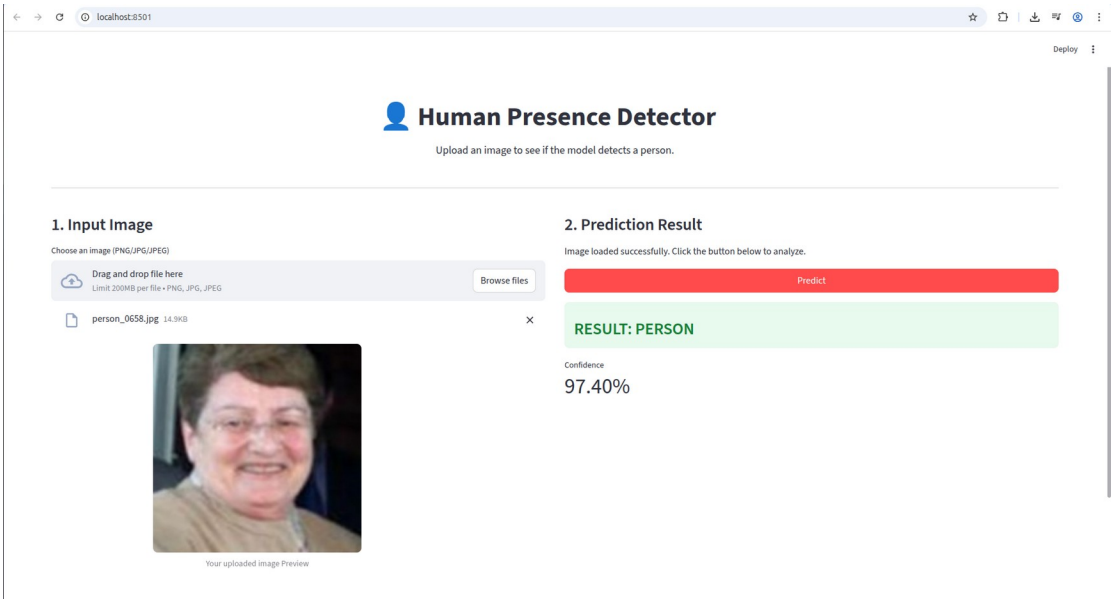
- Để chạy ứng dụng này, người dùng chỉ cần thực thi lệnh sau trong terminal:

- streamlit run streamlit_app/app.py

```python
# URL của API back-end (FastAPI)
API_URL = "http://localhost:8000/api/predict"

# Cấu hình trang
st.set_page_config(
    page_title="Human Presence Detector",
    layout="centered",
)

# 1. Khởi tạo giao diện
st.title("🕵️ Human Presence Detector")
st.write("Tải lên một ảnh để xem mô hình có phát hiện người hay không.")
st.write("---")

# 2. Tải ảnh Lên
uploaded_file = st.file_uploader(
    label="Chọn một tệp ảnh (PNG, JPG, JPEG)",
    type=["png", "jpg", "jpeg"]
)

# 3. Hiển thị ảnh và nút bấm
if uploaded_file is not None:
    try:
        image = Image.open(uploaded_file).convert("RGB")
        st.image(image, caption="Ảnh bạn đã tải lên", use_column_width=True)

        if st.button("Dự đoán"):
            # 4. Gửi yêu cầu dự đoán
            with st.spinner("Đang gửi ảnh đến API và chờ kết quả..."):

                # Chuẩn bị file để gửi đi
                buffer = io.BytesIO()
                image.save(buffer, format='JPEG')
                buffer.seek(0)

                files = {"file": (uploaded_file.name, buffer, "image/jpeg")}

                try:
                    # Gửi request POST đến API
                    response = requests.post(API_URL, files=files)
                except requests.exceptions.RequestException as e:
                    st.error(f"Lỗi kết nối đến API: {e}")
                else:
                    # 5. Nhận và hiển thị kết quả
                    if response.status_code == 200:
                        data = response.json()
                        prediction = data.get("prediction", "Không xác định")
                        confidence = data.get("confidence", 0.0)
                        st.success(f"Kết quả: **{prediction.upper()}** (Độ tin cậy: {confidence:.2f})")
                    else:
                        st.error(f"API trả về lỗi (mã {response.status_code})")
    except Exception as e:
        st.error(f"Lỗi khi xử lý ảnh: {e}")
else:
    st.info("Vui lòng tải lên một tệp ảnh để bắt đầu dự đoán.")
```

Hình 6.1. Mã nguồn xử lý giao diện người dùng với Streamlit.

6.4. Đánh giá và nhận xét hệ thống



Trải nghiệm người dùng: Giao diện được xây dựng bằng Streamlit rất sạch sẽ, trực quan và dễ sử dụng. Người dùng không cần kiến thức kỹ thuật vẫn có thể dễ dàng tương tác với mô hình AI.

Hình 6.2. Hình ảnh giao diện

Kiến trúc tách biệt: Việc tách biệt front-end (Streamlit) và back-end (FastAPI) là một lựa chọn thiết kế rất tốt. Nó mang lại nhiều lợi ích: Linh hoạt: Giao diện và mô hình có thể được phát triển, nâng cấp, và triển khai độc lập với nhau.

Tái sử dụng: API back-end có thể phục vụ nhiều loại client khác nhau (ví dụ: ứng dụng di động, một trang web khác) chứ không chỉ riêng ứng dụng Streamlit này.

Hiệu quả: Mô hình AI chỉ cần được tải một lần trên server FastAPI, giúp tối ưu hóa tài nguyên và đảm bảo các yêu cầu dự đoán được xử lý nhanh chóng.

Mục đích trình diễn: Hệ thống này là một công cụ trình diễn (demo) hoàn hảo, cho phép người khác thấy được khả năng và hiệu suất của mô hình một cách trực quan và sinh động.

KẾT LUẬN

Trong bài báo cáo này, nhóm đã xây dựng thành công hệ thống nhận diện người sử dụng mô hình LeNet-5 cải tiến trên bộ dữ liệu Natural Images. Kết quả thực nghiệm rất khả quan với độ chính xác đạt 96.94% trên tập kiểm định, chứng minh hiệu quả của mô hình trong bài toán phân loại nhị phân.

Dù gặp thách thức về sự đa dạng của dữ liệu ảnh, việc áp dụng các kỹ thuật tiền xử lý và tăng cường dữ liệu (Data Augmentation) như xoay, lật ảnh đã giúp mô hình học đặc trưng tốt hơn và hạn chế hiện tượng overfitting .

Hệ thống đã được triển khai hoàn chỉnh thành một ứng dụng thực tế. Nhóm sử dụng FastAPI để xây dựng API xử lý và Streamlit để tạo giao diện người dùng trực quan, cho phép dự đoán kết quả nhanh chóng từ ảnh tải lên. Đây là cơ sở tốt để phát triển các ứng dụng giám sát an ninh hoặc nhà thông minh.

Hướng phát triển tiếp theo:

- Tích hợp thời gian thực: Áp dụng mô hình vào luồng video (webcam) để phát hiện người liên tục.
- Nâng cấp mô hình: Sử dụng các kiến trúc mạng sâu hơn như VGG16 hoặc MobileNet để tăng độ chính xác trong môi trường phức tạp.
- Mở rộng bài toán: Phát triển thêm tính năng đếm số lượng người hoặc khoanh vùng đối tượng thay vì chỉ phân loại.

Với kết quả đạt được, nhóm hy vọng đề tài sẽ đóng góp một phần nhỏ vào việc ứng dụng thị giác máy tính trong thực tiễn, giúp tự động hóa các quy trình giám sát và nhận diện.

TÀI LIỆU THAM KHẢO

[1] Trần Hùng Cường, Nguyễn Phương Nga, Giáo trình Trí tuệ nhân tạo, 2014

[2]. Lương Chi Mai, *Nhập môn Xử lý ảnh*. Viện Công nghệ Thông tin – Viện Hàn lâm Khoa học và Công nghệ Việt Nam.

[3]. Đinh Mạnh Tường, *Giáo trình Xử lý ảnh số*. Nhà xuất bản Đại học Quốc gia Hà Nội, 2013.

[4]. TensorFlow Team, "TensorFlow Core v2.x API Documentation". [Online]. Available: https://www.tensorflow.org/api_docs.

[5]. Keras Team, "Keras Documentation: The Python Deep Learning API". [Online]. Available: https://keras.io/api/.

[6]. François Chollet, *Deep Learning with Python*. Manning Publications, 2017. (Tác giả của thư viện Keras - rất hợp vì bạn dùng Keras trong bài).

[7]. Richard Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Springer, 2022. (Dùng cho phần tổng quan thị giác máy tính).