HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

MÔN THỰC TẬP CƠ SỞ

Báo Cáo Cuối Kì

Dùng AI (thị giác máy tính) để nhận diện 1 vài động vật

Bằng cấu trúc CNN

Giảng viên hướng dẫn	**: KIM NGỌC BÁCH**
Họ và tên sinh viên	**: NGUYỄN XUÂN HẢI**
Mã sinh viên	**: B22DCCN271**
Lớp	**: D22CQCN07-B**
Nhóm	**: 13**

2025

Mục Lục

Phần I. Giới thiệu chung.....4
1.1 Lý do chọn đề tài..... 4
1.2. Mục tiêu của dự án..... 4
1.3. Phạm vi ứng dụng.....6
Phần II. Tổng quan về học sâu và phân loại hình ảnh..... 5
2.1. Giới thiệu về học sâu..... 5
2.2. Mạng nơ-ron tích chập.....6
2.3. Phân loại động vật bằng học sâu.....10
Phần III. Kiến trúc EfficientNet và ứng dụng trong phân loại động vật.....12
3.1. Tổng quan về EfficientNet.....12
3.2. Kiến trúc chi tiết Efficient_B0.....14
3.3. Ứng dụng EfficientNet trong phân loại động vật..... 14
Phần IV. Tiền xử lý dữ liệu động vật và huấn luyện mô hình..... 16
4.1. Tổng quan về dữ liệu động vật..... 16
4.2. Tiền xử lý dữ liệu.....18
4.3. Phân chia dữ liệu..... 19
4.4. Khởi tạo mô hình.....20
4.5. Huấn luyện mô hình.....21
4.6. Kiểm thử mô hình..... 21
4.7. Các kĩ thuật trong huấn luyện mô hình..... 22
Phần V. Triển khai mô hình và ứng dụng thực tế.....24
5.1. Công nghệ và công cụ sử dụng..... 24
5.2. Môi trường phát triển..... 27
5.3. Triển khai mô hình.....27
5.4. Nhận diện từ webcam thời gian thực.....28
5.5. Giao diện người dùng..... 31
5.6. Triển khai bằng Docker..... 31
Phần VI. Kết quả và đánh giá..... 35
6.1 Độ chính xác và các chỉ số hiệu suất..... 35
6.2. Hiệu suất thời gian thực.....35
6.3. Phân tích kết quả..... 36
Phần VII. Kết luận và hướng phát triển.....37
7.1. Những kết quả đạt được.....37
7.2. Hạn chế còn tồn tại..... 38
7.3. Định hướng mở rộng trong tương lai.....39
Tham khảo..... 40

Lời Cảm Ơn

Em xin chân thành cảm ơn Thầy **Kim Ngọc Bách** vì đã tận tình hướng dẫn và đồng hành cùng em trong môn **Thực tập Cơ sở** suốt 3 tháng vừa qua. Những định hướng, kiến thức của Thầy đã giúp em không chỉ nắm vững các kỹ năng chuyên môn mà còn hiểu rõ hơn về tinh thần trách nhiệm và đam mê trong công việc. Em hy vọng sẽ tiếp tục áp dụng những gì Thầy đã truyền đạt vào hành trình học tập và công việc sau này.

Một lần nữa, em xin gửi lời tri ân sâu sắc đến Thầy. Chúc Thầy luôn mạnh khỏe, hạnh phúc và tiếp tục truyền cảm hứng cho nhiều thế hệ học sinh, sinh viên!

Phần I. Giới thiệu chung

1.1 Lý do chọn đề tài

Phân loại động vật là một trong những ứng dụng quan trọng trong các lĩnh vực giám sát môi trường và nông nghiệp, nơi việc nhận diện chính xác các loài động vật đóng vai trò thiết yếu trong việc bảo tồn thiên nhiên, quản lý tài nguyên và nâng cao hiệu quả sản xuất. Trong giám sát môi trường, việc theo dõi các loài động vật hoang dã giúp các nhà nghiên cứu đánh giá sự đa dạng sinh học, phát hiện sớm các vấn đề như suy giảm quần thể hoặc xâm nhập loài ngoại lai. Trong nông nghiệp, nhận diện động vật có thể hỗ trợ phát hiện sâu bệnh, động vật gây hại hoặc quản lý đàn gia súc một cách tự động, giảm thiểu lao động thủ công và tăng năng suất. Tuy nhiên, việc phân loại thủ công thường tốn thời gian, phụ thuộc vào chuyên gia, và không khả thi trong các môi trường phức tạp với số lượng lớn dữ liệu hình ảnh.

Học sâu (Deep Learning), đặc biệt là các mạng nơ-ron tích chập (Convolutional Neural Networks - CNN), đã chứng minh khả năng vượt trội trong việc xử lý hình ảnh nhờ khả năng tự động trích xuất đặc trưng từ dữ liệu thô mà không cần can thiệp thủ công. Trong dự án này, em chọn mô hình EfficientNet – một kiến trúc CNN tiên tiến, nổi bật với phương pháp scaling đồng đều (compound scaling) về chiều sâu, chiều rộng và độ phân giải, giúp đạt hiệu suất cao nhưng vẫn tiết kiệm tài nguyên tính toán. Đây là một lựa chọn phù hợp cho sinh viên, khi cần cân bằng giữa độ chính xác và khả năng triển khai trên các thiết bị hạn chế (như máy tính cá nhân hoặc server nhỏ). Lý do chọn đề tài này xuất phát từ nhu cầu thực tiễn về một giải pháp tự động, chính xác và khả năng triển khai trên các thiết bị thời gian thực, đồng thời tận dụng sức mạnh của học sâu để giải quyết các thách thức trong nhận diện động vật. Dự án không chỉ góp phần vào nghiên cứu học máy mà còn mang lại giá trị ứng dụng thực tế trong các lĩnh vực liên quan.

Là một sinh viên ngành Công nghệ Thông tin, em nhận thấy phân loại động vật là một chủ đề thú vị và mang tính thực tiễn cao, kết nối trực tiếp với các kiến thức về học máy và xử lý hình ảnh mà em đã học trong chương trình. Phân loại động vật đóng vai trò quan trọng trong giám sát môi trường và nông nghiệp, chẳng hạn như theo dõi sự đa dạng sinh học trong các khu bảo tồn hoặc quản lý gia súc trong trang trại. Ngoài ra, đề tài này cũng là cơ hội để em hoàn thiện bài tập lớn, tích lũy kinh nghiệm cho công việc tương lai trong ngành CNTT.

1.2. Mục tiêu của dự án

Dự án đặt ra các mục tiêu cụ thể để phát triển một hệ thống phân loại động vật dựa trên kiến thức và kỹ năng của một sinh viên CNTT:

- **Xây dựng mô hình phân loại động vật với độ chính xác trên 90%**: Sử dụng mô hình EfficientNet, em sẽ huấn luyện mô hình trên một tập dữ liệu hình ảnh động vật (như mèo, chó, chim, v.v.) mà em tự thu thập hoặc sử dụng từ nguồn mở như CIFAR-10 và Kaggle. Quá trình này bao gồm tinh chỉnh mô hình pre-trained, em ưu hóa siêu tham số, và đánh giá hiệu suất trên tập dữ liệu kiểm thử để đạt mục tiêu độ chính xác cao, đồng thời giúp em nắm vững quy trình phát triển mô hình học sâu.
- **Triển khai ứng dụng thời gian thực qua webcam và API**: Ngoài việc xây dựng mô hình, em hướng em phát triển một ứng dụng thực tế bằng cách tích hợp mô hình với webcam để nhận diện động vật trực tiếp, sử dụng OpenCV. Đồng thời, em sẽ tạo một API đơn giản bằng Flask để cung cấp dịch vụ phân loại từ xa, đóng gói toàn bộ bằng Docker để đảm bảo khả năng triển khai linh hoạt. Những bước này không chỉ áp dụng kiến thức lập trình mà còn giúp em hiểu sâu hơn về triển khai phần mềm.

1.3. Phạm vi ứng dụng

Dự án phân loại động vật bằng CNN EfficientNet có phạm vi ứng dụng phù hợp với vai trò của một sinh viên CNTT, với tiềm năng mở rộng trong các lĩnh vực thực tiễn:

- **Giám sát động vật hoang dã**: Hệ thống có thể được triển khai trong các khu bảo tồn hoặc công viên quốc gia để theo dõi quần thể động vật, hỗ trợ các nhà sinh học trong việc thu thập dữ liệu mà không cần can thiệp trực tiếp. Ví dụ, camera đặt tại rừng có thể tự động ghi nhận sự xuất hiện của các loài động vật quý hiếm như hổ hoặc voi, giúp phát hiện sớm các nguy cơ tuyệt chủng.
- **Trang trại và nông nghiệp**: Trong ngành chăn nuôi, mô hình có thể được sử dụng để nhận diện gia súc (bò, dê, cừu) hoặc phát hiện động vật gây hại (chuột, chim phá hoại mùa màng). Điều này giúp nông dân quản lý đàn vật nuôi hiệu quả hơn, giảm tổn thất do sâu bệnh hoặc động vật hoang dã, đồng thời em ưu hóa quy trình sản xuất.
- **Nghiên cứu sinh thái**: Dự án cung cấp một công cụ hỗ trợ các nhà khoa học trong việc phân tích sự phân bố và hành vi của động vật qua hình ảnh thu thập từ camera bẫy hoặc drone. Kết quả từ mô hình có thể được sử dụng để xây dựng các mô hình sinh thái học, dự đoán tác động của biến đổi khí hậu hoặc hoạt động con người lên hệ sinh thái.

Với phạm vi ứng dụng trên, dự án không chỉ giúp em hoàn thành tốt bài tập lớn mà còn mở ra cơ hội học hỏi, phát triển kỹ năng CNTT, và đóng góp vào các vấn đề thực tiễn trong xã hội.

Phần II. Tổng quan về học sâu và phân loại hình ảnh

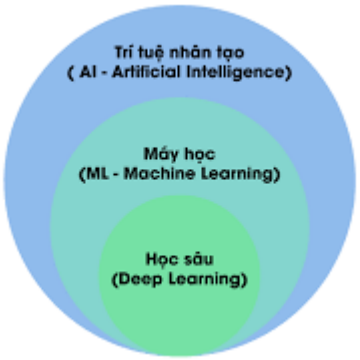
2.1. Giới thiệu về học sâu

Học sâu (Deep Learning) là một phần của trí tuệ nhân tạo mà các mạng nơ-ron sâu được sử dụng để học và hiểu dữ liệu phức tạp. Trong Deep Learning, các mạng nơ-ron được cấu trúc với nhiều lớp (từ đó có tên gọi "sâu"), mỗi lớp thực hiện các phép tính phức tạp để tự động rút trích các đặc trưng từ dữ liệu.

Deep Learning thường được áp dụng trong các bài toán nhận dạng hình ảnh, nhận dạng giọng nói, dịch ngôn ngữ tự nhiên, và nhiều lĩnh vực khác. Đặc biệt, trong nhận dạng hình ảnh, Deep Learning đã giúp máy tính nhận dạng đối tượng và các đặc điểm phức tạp trong ảnh với hiệu suất cao hơn, gần như ngang ngửa với khả năng nhận dạng của con người.

Một trong những lợi ích lớn nhất của Deep Learning là khả năng học từ dữ liệu lớn một cách tự động và không cần sự can thiệp của con người để định rõ các đặc trưng hay quy tắc. Điều này giúp tạo ra các hệ thống thông minh có khả năng tự học và tự cải thiện theo thời gian.

Tóm lại, Deep Learning là một lĩnh vực của trí tuệ nhân tạo, trong đó các mạng nơ-ron sâu được sử dụng để học và hiểu dữ liệu phức tạp, mà không cần phải xác định rõ các đặc trưng hoặc quy tắc một cách cụ thể.



Hình 1. Deep learning là gì

2.2. Mạng nơ-ron tích chập (CNN)

1. Thuật toán CNN

Thuật toán CNN, hay Convolutional Neural Network (Mạng Nơ-ron Tích Chập). Đây là một loại mô hình học sâu rất mạnh mẽ trong lĩnh vực trí tuệ nhân tạo, đặc biệt là trong xử lý hình ảnh. CNN giúp xây dựng các hệ thống thông minh với độ chính xác cao, nhờ khả năng nhận diện và phân tích các đặc điểm quan trọng trong ảnh.

CNN hoạt động bằng cách xử lý dữ liệu hình ảnh thông qua các lớp tích chập, giúp trích xuất các đặc điểm nổi bật từ hình ảnh một cách hiệu quả. Ví dụ, thuật toán này thường được sử dụng để nhận diện khuôn mặt, phân loại đối tượng, và nhiều ứng dụng khác. Những nền tảng nổi tiếng như Facebook và Google cũng đã tích hợp CNN để cải thiện khả năng nhận diện hình ảnh trên các dịch vụ của họ.

Về mặt kỹ thuật, khi một hình ảnh được đưa vào hệ thống CNN, nó sẽ trải qua một loạt các bước. Đầu tiên, hình ảnh sẽ được xử lý qua các lớp tích chập với các bộ lọc để trích xuất các đặc điểm. Sau đó, dữ liệu sẽ đi qua các lớp kết nối đầy đủ và cuối cùng là lớp phân loại sử dụng hàm Softmax. Nó đưa ra xác suất cho các loại đối tượng khác nhau. Kết quả cuối cùng sẽ cho chúng ta biết khả năng thuộc về từng loại của đối tượng trong hình ảnh.

2. Các lớp cơ bản của mạng CNN

Thuật toán CNN gồm những lớp cơ bản sau:

a, Convolutional layer

Convolutional layer là lớp quan trọng nhất trong CNN, đảm nhiệm vai trò thực hiện các phép tính chính. Những yếu tố quan trọng của lớp này bao gồm stride, padding, filter map, và feature map.

- Filter Map: Đây là các bộ lọc được áp dụng lên từng vùng của hình ảnh. Mỗi filter map là một ma trận 3 chiều chứa các tham số được biểu diễn dưới dạng số.
- Stride: Đây là bước dịch chuyển của filter map trên hình ảnh, dịch từ trái sang phải theo từng pixel dựa trên giá trị đã xác định.
- Padding: Đây là các giá trị 0 sẽ được thêm vào lớp input ở viền ảnh để giữ kích thước của ảnh không bị thay đổi.
- Feature Map: Sau mỗi lần filter map quét qua input, một quá trình tính toán diễn ra. Và feature map chính là kết quả của quá trình này. Nó thể hiện các đặc trưng đã được trích xuất từ hình ảnh ban đầu.

b, Relu layer

Relu layer, hay còn gọi là hàm kích hoạt (activation function), đóng vai trò quan trọng trong mạng nơ-ron nhân tạo. Nó mô phỏng hoạt động của các neuron thần kinh bằng cách truyền tín hiệu qua axon. Ngoài Relu, còn có các hàm kích hoạt khác như Tanh, Sigmoid, Maxout, và Leaky Relu.

Relu layer được ứng dụng phổ biến trong quá trình huấn luyện mạng nơ-ron nhờ vào hiệu quả và các ưu điểm vượt trội, giúp mô hình học nhanh hơn và chính xác hơn.

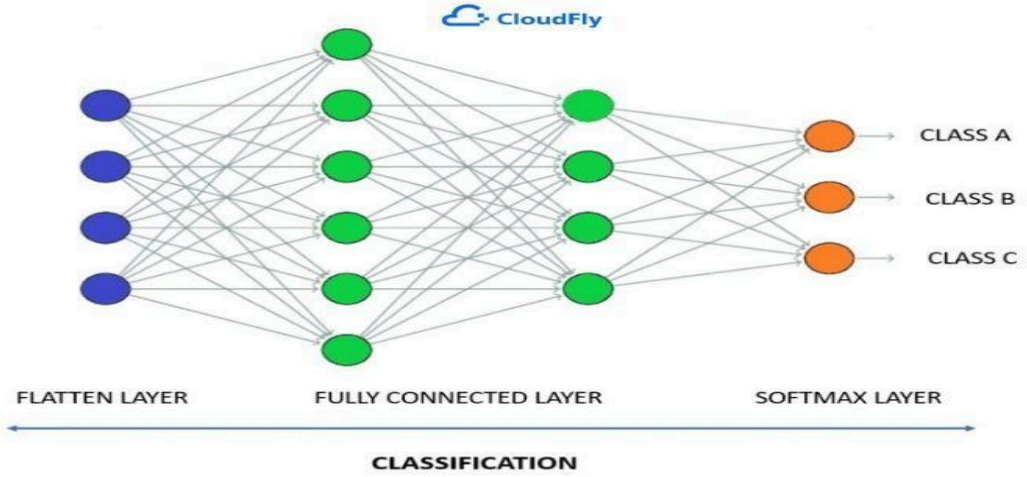
c, Pooling layer

Pooling layer giúp giảm kích thước đầu vào khi dữ liệu quá lớn. Nó được đặt giữa các lớp Convolutional layer để giảm số lượng tham số cần tính toán.

Có hai loại phổ biến là max pooling và average pooling. Max pooling lấy giá trị lớn nhất trong vùng dữ liệu, trong khi average pooling tính giá trị trung bình. Cả hai đều giúp giảm tải cho mô hình và tăng hiệu quả xử lý.

d, Fully connected layer

Fully connected layer đảm nhiệm vai trò xuất kết quả sau khi ảnh đã được xử lý qua các lớp convolutional và pooling. Khi mô hình đã đọc được thông tin từ ảnh, lớp này tạo ra sự kết nối để sinh ra nhiều output hơn. Lập trình viên sử dụng fully connected layer để tổng hợp và xử lý dữ liệu cuối cùng. Ngoài ra, nếu lớp này nhận dữ liệu về hình ảnh, nó sẽ chuyển thành các mục phân loại nhằm phân tích sâu hơn.



Hình 2: Các lớp trong CNN

3. Cấu trúc của thuật toán CNN

Mạng CNN (Convolutional Neural Network) là một tập hợp các lớp Convolution được xếp chồng lên nhau. Kết hợp với các hàm kích hoạt phi tuyến tính như ReLU và tanh để điều chỉnh trọng số trong các node. Khi dữ liệu đi qua các lớp này, trọng số được học và tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

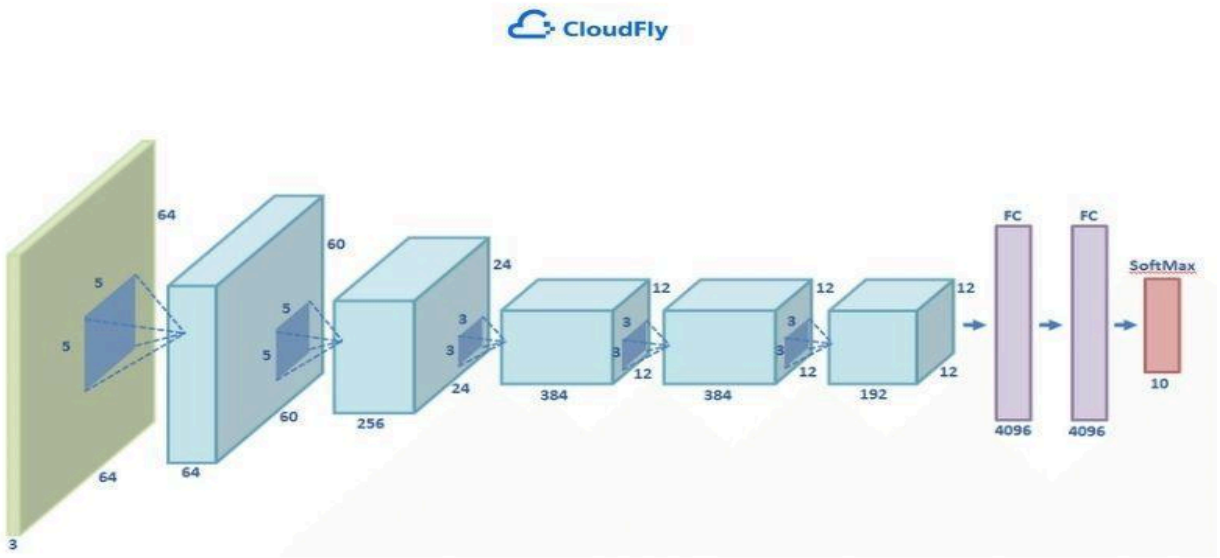
Một đặc điểm quan trọng của thuật toán CNN là tính bất biến và tính kết hợp cục bộ. Pooling layer đảm bảo tính bất biến với các biến dạng như dịch chuyển, co giãn và quay, giúp CNN đưa ra kết quả chính xác hơn. Trong khi đó, tính kết hợp cục bộ giúp biểu diễn thông tin từ mức độ thấp đến cao, thông qua quá trình convolution từ các bộ lọc. Mỗi lớp tiếp theo sẽ nhận kết quả từ lớp convolution trước đó, giúp kết nối cục bộ giữa các lớp hiệu quả hơn. Ngoài ra, Pooling/Subsampling layer giúp lọc bớt những thông tin nhiễu, chỉ giữ lại những thông tin quan trọng.

Trong quá trình huấn luyện, CNN sẽ tự động học các giá trị từ dữ liệu, tương tự như cách con người nhận diện vật thể

Cấu trúc cơ bản của CNN gồm ba phần chính:

- Local Receptive Field: Chọn lọc các vùng ảnh có giá trị sử dụng cao nhất.

- Shared Weights and Bias: Giúp giảm thiểu lượng tham số, mỗi feature map nhận diện các đặc trưng khác nhau trong ảnh.
- Pooling Layer: em ưu hóa thông tin đầu ra bằng cách loại bỏ các thông tin không cần thiết sau khi tính toán qua các lớp, giúp người dùng nhận được kết quả phù hợp với yêu cầu.



Hình 3: Sơ đồ cấu trúc CNN

4. Hướng dẫn cách chọn tham số cho CNN

Để chọn tham số phù hợp nhất cho CNN, cần lưu ý đến một số yếu tố quan trọng. Bao gồm kích thước filter, kích thước pooling, số lượng convolution layer, và số lần train test.

- Convolution layer: Số lượng lớp càng lớn, mô hình của bạn sẽ càng được cải thiện. Việc sử dụng nhiều lớp có thể giảm bớt sai lệch và giúp mô hình hoạt động hiệu quả hơn. Thông thường, chỉ cần từ 3 đến 5 lớp là bạn có thể đạt được kết quả tốt.
- Filter size: Kích thước filter phổ biến thường là 3x3 hoặc 5x5.
- Pooling size: Với hình ảnh thông thường, nên sử dụng kích thước 2x2. Nếu xử lý hình ảnh có kích thước lớn hơn, có thể chuyển sang dùng kích thước 4x4.
- Train test: Càng thực hiện nhiều lần train test, càng dễ thu được các tham số em ưu. Nhờ đó giúp mô hình trở nên "thông minh" và hiệu quả hơn.

5. Ứng dụng

Mạng nơ-ron tích chập (CNN) đã có nhiều ứng dụng quan trọng và đa dạng trong thực tế:

− Nhận diện và phân loại ảnh: CNN được sử dụng để nhận diện và phân loại đối tượng trong ảnh, từ việc nhận diện khuôn mặt đến loại hoa, động vật, và vật thể khác.
− Nhận diện và theo dõi video: Trong lĩnh vực giám sát an ninh, CNN có thể giúp nhận diện hành vi đáng ngờ và theo dõi đối tượng trong video.
− Xử lý ngôn ngữ tự nhiên (NLP): Trong NLP, CNN có thể được sử dụng để phân loại văn bản, đánh giá cảm xúc trong văn bản, và thậm chí là dịch máy.
− Tự động hóa làm việc: CNN có thể hỗ trợ trong việc tự động hóa quy trình công việc, chẳng hạn như nhận diện và phân loại tài liệu.
− Y tế và chăm sóc sức khỏe: CNN được sử dụng trong hình ảnh y tế để phân loại bệnh lý, nhận diện tế bào ung thư, và hỗ trợ trong quá trình chẩn đoán.
− Tìm kiếm và gợi ý: CNN có thể cải thiện kết quả tìm kiếm hình ảnh và cung cấp gợi ý sản phẩm dựa trên hình ảnh.
− Xe tự hành và công nghệ ô tô: Trong xe tự hành, CNN có vai trò quan trọng trong việc nhận diện và đánh giá môi trường xung quanh.
− Trò chơi và giải trí: CNN được sử dụng trong công nghiệp trò chơi để tạo đồ họa chất lượng cao và cải thiện trải nghiệm người chơi.
− Phân loại dữ liệu: CNN (Convolutional Neural Network) thường được sử dụng để phân loại dữ liệu phức tạp trong nghiên cứu và khoa học dữ liệu.
− Nông nghiệp thông minh: CNN có thể hỗ trợ trong nhận diện và theo dõi mặt đất, dự đoán mùa vụ và quản lý tình trạng nông nghiệp.

2.3. Phân loại động vật bằng học sâu

1. Thách thức trong phân loại động vật

Phân loại động vật từ hình ảnh là một bài toán phức tạp với nhiều thách thức mà các phương pháp truyền thống (như phân loại thủ công hoặc sử dụng đặc trưng do con người thiết kế) khó có thể giải quyết hiệu quả. Em nhận thấy các thách thức chính bao gồm:

- **Đa dạng về loài và ngoại hình**: Các loài động vật có sự khác biệt lớn về hình dạng, kích thước, màu sắc và hoa văn. Ví dụ, cùng một loài chó nhưng có thể có nhiều giống khác nhau (như Husky và Poodle), khiến việc nhận diện trở nên khó khăn.
- **Yếu tố môi trường**: Hình ảnh động vật thường được chụp trong điều kiện ánh sáng, góc chụp và phông nền khác nhau. Một con mèo trong ánh sáng yếu hoặc bị che khuất bởi lá cây có thể làm giảm độ chính xác của mô hình nếu không được huấn luyện kỹ lưỡng.
- **Thiếu dữ liệu cân bằng**: Trong thực tế, dữ liệu hình ảnh động vật thường không đồng đều giữa các lớp. Một số loài phổ biến (như mèo, chó) có thể có nhiều hình ảnh, trong khi các loài hiếm gặp (như hổ, tê giác) lại thiếu dữ liệu, dẫn đến mô hình bị thiên lệch (bias).
- **Tài nguyên hạn chế**: Với sinh viên như em, việc huấn luyện mô hình học sâu thường bị giới hạn bởi tài nguyên tính toán. Máy tính cá nhân hoặc Google Colab có

thể không đủ sức mạnh để xử lý các mô hình lớn trên tập dữ liệu lớn, đòi hỏi một mô hình hiệu quả như EfficientNet.

Những thách thức này khiến phân loại động vật trở thành một bài toán thú vị nhưng cũng đầy thử thách, đặc biệt khi áp dụng vào thực tế.

2. Lợi ích của học sâu trong phân loại động vật

Học sâu, đặc biệt là các mạng nơ-ron tích chập (CNN), mang lại nhiều lợi ích vượt trội so với các phương pháp truyền thống, giúp giải quyết các thách thức kể trên:

- **Tự động trích xuất đặc trưng**: Không giống các phương pháp cổ điển của machine learning, học sâu tự động học các đặc trưng quan trọng từ hình ảnh (như hình dạng tai, đuôi, hoa văn) mà không cần con người thiết kế thủ công. Điều này rất hữu ích khi làm việc với dữ liệu động vật đa dạng, nơi các đặc trưng phức tạp và khó xác định.
- **Độ chính xác cao**: Các mô hình như EfficientNet, với khả năng em ưu hóa hiệu suất qua phương pháp compound scaling (điều chỉnh đồng đều chiều sâu, chiều rộng và độ phân giải), có thể đạt độ chính xác cao ngay cả trên các tập dữ liệu phức tạp. Em nhận thấy điều này giúp em dễ dàng đạt được mục tiêu độ chính xác trên 90% trong dự án nếu em có đủ tập huấn luyện và huấn luyện đủ lâu..
- **Khả năng tổng quát hóa**: Học sâu cho phép mô hình học được các đặc trưng chung của từng loài, giúp nhận diện động vật trong các điều kiện khác nhau (ánh sáng yếu, góc chụp khó). Ví dụ, mô hình có thể nhận diện một con mèo dù hình ảnh bị mờ hoặc chụp từ góc nghiêng.
- **Tự động hóa quy trình**: Một hệ thống phân loại động vật dựa trên học sâu có thể tự động phân tích hàng nghìn hình ảnh mà không cần sự can thiệp của con người, tiết kiệm thời gian và công sức. Điều này đặc biệt phù hợp với các ứng dụng như giám sát động vật hoang dã hoặc quản lý trang trại.

3. Ứng dụng cụ thể của học sâu trong dự án

Trong dự án này, em sử dụng mô hình EfficientNet để xây dựng hệ thống phân loại động vật, tận dụng các lợi ích của học sâu để vượt qua thách thức:

- **Lựa chọn mô hình EfficientNet**: EfficientNet được chọn vì tính hiệu quả của nó trong việc cân bằng giữa độ chính xác và tài nguyên tính toán. Với sinh viên như em, điều này rất quan trọng khi chỉ có máy tính cá nhân hoặc tài nguyên hạn chế để huấn luyện. EfficientNet-B0, phiên bản nhỏ nhất, vẫn mang lại hiệu suất cao nhờ phương pháp scaling em ưu.
- **Huấn luyện trên dữ liệu động vật**: em sử dụng tập dữ liệu hình ảnh động vật (có thể từ CIFAR-10 hoặc tự thu thập) để huấn luyện mô hình. Học sâu giúp mô hình học các đặc trưng phức tạp như hoa văn lông, hình dạng tai, hoặc dáng đi của từng loài, từ đó phân biệt chính xác giữa các lớp động vật.

- **Tích hợp vào ứng dụng thực tế**: Sau khi huấn luyện, em triển khai mô hình vào một hệ thống nhận diện thời gian thực qua webcam, sử dụng OpenCV để xử lý video. Ngoài ra, một API được phát triển bằng Flask để cung cấp dịch vụ phân loại từ xa. Học sâu giúp hệ thống hoạt động hiệu quả, tự động và chính xác, đáp ứng các yêu cầu thực tế trong giám sát môi trường và nông nghiệp.

Phần III. Kiến trúc EfficientNet và ứng dụng trong phân loại động vật

3.1. Tổng quan về EfficientNet

EfficientNet là một gia đình mô hình học sâu dựa trên mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) được giới thiệu bởi Google vào năm 2019, với mục tiêu tối ưu hóa hiệu suất tính toán và độ chính xác trên các bài toán xử lý hình ảnh. Khác với các kiến trúc truyền thống như ResNet, hay DenseNet, EfficientNet áp dụng phương pháp **compound scaling**, một chiến lược độc đáo điều chỉnh đồng thời ba chiều cấu trúc mạng: **chiều sâu** (số lượng tầng), **chiều rộng** (số kênh đầu ra của các tầng tích chập), và **độ phân giải đầu vào** (kích thước ảnh).

- **Thu phóng theo chiều sâu (Depth Scaling):**

Thu phóng theo chiều sâu là một cách thông dụng nhất được sử dụng để thu phóng một mô hình CNN. Độ sâu có thể được thu phóng cũng như thu nhỏ bằng cách thêm hoặc bớt các lớp tương ứng. Ví dụ: ResNets có thể được mở rộng từ ResNet-50 đến ResNet-200 cũng như chúng có thể được thu nhỏ từ ResNet-50 thành ResNet-18. Tuy nhiên việc lạm dụng thu phóng theo chiều sâu có thể không cải thiện hiệu quả của mô hình, thậm chí có thể làm mô hình kém hiệu quả hơn so với mô hình ban đầu. Đúng là có một số lý do mà việc thêm nhiều lớp ẩn hơn sẽ cung cấp mức độ chính xác hơn cho mô hình. Tuy nhiên, điều này chỉ đúng với các tập dữ liệu lớn hơn, vì càng nhiều lớp với hệ số bước ngắn hơn sẽ trích xuất nhiều tính năng hơn cho dữ liệu đầu vào của bạn. Việc sử dụng một mô hình quá phức tạp với lượng dữ liệu không tương xứng, như ta đã biết, có thể gây ra hiện tượng Overfiiting. Thêm nữa, các mạng sâu hơn có xu hướng bị vanishing gradients và trở nên khó đào tạo. Vậy nên không phải lúc nào, thu phóng theo chiều sâu cũng là sự lựa chọn thích hợp để cải thiện mô hình CNN.

- **Thu phóng theo chiều rộng (Width Scaling):**

Việc thu phóng theo chiều rộng của mạng cho phép các lớp tìm hiểu các tính năng chi tiết hơn. Khái niệm này đã được sử dụng rộng rãi trong nhiều công trình như Wide ResNet và Mobile Net. Tuy nhiên, cũng như trường hợp tăng chiều sâu, tăng chiều rộng ngăn cản mạng học các tính năng phức tạp, dẫn đến giảm độ chính xác.

- **Thu phóng theo độ phân giải (Resolution Scaling)**

Theo một cách trực quan, chúng ta có thể nói rằng trong một hình ảnh có độ phân giải cao, các đặc trưng sẽ có độ chi tiết cao hơn và do đó hình ảnh có độ phân giải cao sẽ

hoạt động tốt hơn. Độ phân giải đầu vào cao hơn cung cấp hình ảnh chi tiết hơn và do đó nâng cao khả năng suy luận của mô hình về các đối tượng nhỏ hơn và trích xuất các mẫu mịn hơn. Tuy nhiên cũng như các cách thu phóng trên, việc chỉ thu phóng theo độ phân giải không hề luôn luôn hiệu quả trong mọi trường hợp mà thậm chí nó còn có thể giảm độ chính xác của mô hình đi một cách nhanh chóng.

Kết luận: Thu phóng quy mô bất kỳ kích thước nào về chiều rộng, chiều sâu hoặc độ phân giải của mạng sẽ cải thiện độ chính xác, nhưng độ chính xác sẽ giảm đối với các mô hình lớn hơn.

=> Phương pháp này được xây dựng dựa trên nghiên cứu thực nghiệm, trong đó việc tăng đồng đều ba yếu tố này theo một hệ số tỷ lệ nhất định (được xác định qua tối ưu hóa lưới) giúp cải thiện hiệu suất mô hình mà không gây tăng trưởng quá mức về chi phí tính toán(FLOPS).

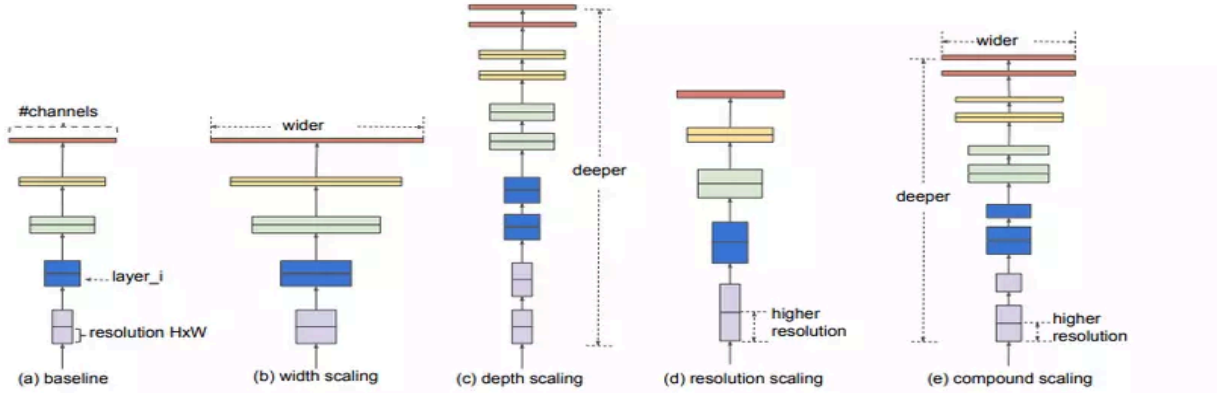


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Hình 4 Cấu trúc mạng CNN và tác động khi chỉnh 3 chiều của cấu trúc

Lợi thế cốt lõi của EfficientNet nằm ở khả năng đạt độ chính xác vượt trội trên các tập dữ liệu benchmark như ImageNet với mức tiêu thụ tài nguyên thấp hơn đáng kể so với các mô hình cùng thời. Điều này được thực hiện thông qua việc tích hợp các khối **MBConv** (Mobile Inverted Bottleneck Convolution) kết hợp với kỹ thuật **squeeze-and-excitation** để tăng cường khả năng chú ý đến các đặc trưng quan trọng.

3.2. Kiến trúc chi tiết Efficient_B0

EfficientNet-B0 là phiên bản cơ bản nhất trong gia đình EfficientNet, được thiết kế nhằm đạt được sự cân bằng tối ưu giữa hiệu suất và hiệu quả tính toán. Đây là nền tảng cho các phiên bản lớn hơn, sử dụng các khối MBConv và chiến lược compound scaling với hệ số tỷ lệ cơ bản.

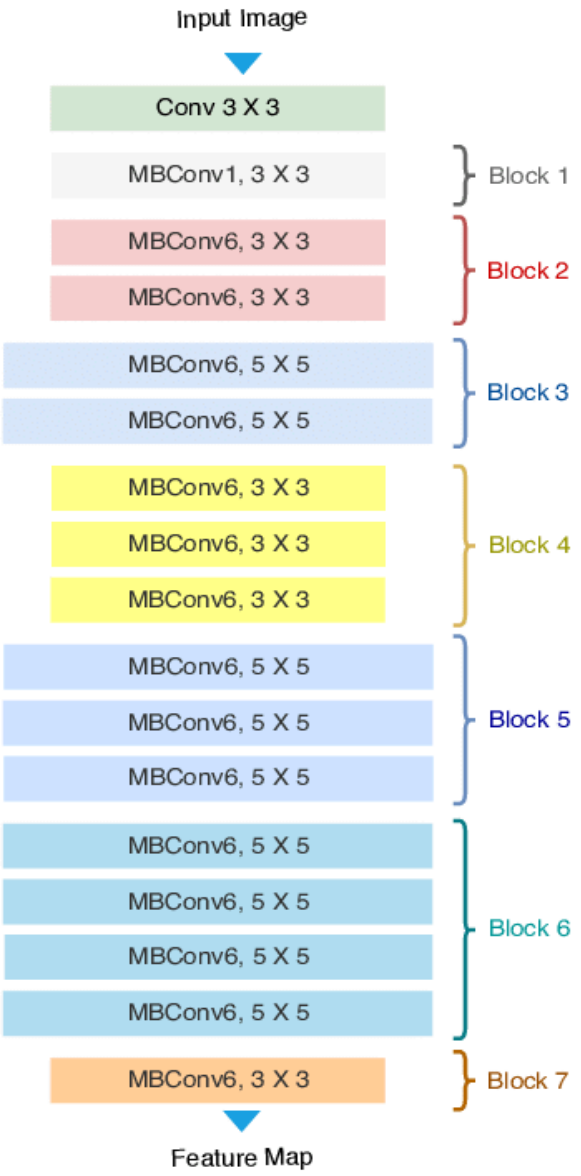
- **Đầu vào**: Hình ảnh đầu vào có độ phân giải 224x224 pixel, được chuẩn hóa để phù hợp với các tầng tích chập.
- **Khối cơ bản**: EfficientNet-B0 sử dụng các **MBConv blocks** (Mobile Inverted Bottleneck Convolution), một biến thể của khối tích chập đảo ngược (inverted residual block) từ MobileNetV2. Mỗi MBConv block bao gồm:
    - Một tầng tích chập 1x1 để tăng số kênh (expansion phase).
    - Một tầng tích chập sâu 3x3 với stride 1 hoặc 2, sử dụng kernel nhỏ để giảm tham số.
    - Một tầng tích chập 1x1 để giảm số kênh (projection phase).
    - Thêm shortcut (skip connection) để giảm mất mát gradient, tương tự ResNet.
- **Số tầng**: EfficientNet-B0 có tổng cộng **16 tầng MBConv**, được tổ chức thành 5 giai đoạn (stages) với số lượng block tăng dần (1, 2, 2, 3, 1).
- **Số tham số**: Khoảng **5,3 triệu tham số**, nhỏ hơn nhiều so với các mô hình như ResNet-50 (25 triệu tham số), giúp huấn luyện nhanh hơn trên thiết bị hạn chế.
- **Tính toán (FLOPs)**: Khoảng **0,39 tỷ phép toán**, cho phép mô hình chạy hiệu quả trên CPU hoặc GPU cơ bản.

Kiến trúc này không chỉ thể hiện sự tối ưu về mặt kỹ thuật mà còn cung cấp một nền tảng linh hoạt để nghiên cứu và triển khai trong các ứng dụng thực tế. EfficientNet-B0 được huấn luyện trước trên tập dữ liệu ImageNet, cung cấp trọng số ban đầu mà tôi có thể tinh chỉnh cho dữ liệu động vật. Em chọn phiên bản này vì nó phù hợp với tài nguyên hạn chế và vẫn mang lại kết quả đáng tin cậy.

3.3. Ứng dụng EfficientNet trong phân loại động vật

EfficientNet, đặc biệt là phiên bản B0, được áp dụng hiệu quả trong dự án phân loại động vật, tận dụng các đặc tính ưu việt của mô hình để xử lý dữ liệu hình ảnh phức tạp. Dưới đây là phân tích chi tiết về cách triển khai:

- **Lựa chọn EfficientNet-B0**: EfficientNet-B0 được ưu tiên sử dụng nhờ khả năng đạt độ chính xác cao (khoảng 77,1%) với chi phí tính toán thấp (0,39 tỷ FLOPs). Điều này đặc biệt phù hợp cho các hệ thống có tài nguyên hạn chế, như máy tính cá nhân hoặc server nhỏ, đồng thời đảm bảo hiệu suất vượt trội so với các mô hình truyền thống trong cùng phân khúc tài nguyên.



Hình 5: Kiến trúc Efficient B0

- **Tinh chỉnh mô hình (Fine-tuning)**: Quá trình tinh chỉnh bắt đầu với trọng số pre-trained từ ImageNet. Tầng fully connected cuối cùng được thay thế bằng một tầng mới với số nút tương ứng với số lớp động vật trong dataset (Ví dụ: 19 lớp như mèo, chó, gà, …). Các kỹ thuật tối ưu hóa bao gồm điều chỉnh learning rate (thường là 0,001), sử dụng batch normalization để ổn định huấn luyện, và áp dụng dropout (tỷ lệ 0,1) để tránh overfitting. Ngoài ra, data augmentation (xoay 90 độ, lật ngang, thay đổi độ sáng) được tích hợp để tăng cường khả năng tổng quát hóa trên các điều kiện ánh sáng và góc chụp khác nhau.
- **Huấn luyện trên dữ liệu động vật**: Mô hình được huấn luyện trên tập dữ liệu hình ảnh động vật, chẳng hạn từ thư mục animals đã được điều chỉnh. EfficientNet-B0 tự động trích xuất các đặc trưng phân cấp, từ đặc trưng cơ bản (đường nét, màu sắc) đến đặc trưng phức tạp (hoa văn lông, hình dáng cơ thể), thông qua các tầng MBConv. Quá trình huấn luyện sử dụng hàm mất mát cross-entropy và tối ưu hóa bằng SGD với momentum, nhắm đến mục tiêu đạt độ chính xác trên 90% trên tập kiểm thử. Kết

quả huấn luyện được lưu dưới dạng trọng số best.pt (trọng số tốt nhất) và last.pt (trọng số cuối cùng) để phục vụ triển khai sau này.

- **Triển khai ứng dụng thực tế**: Sau khi huấn luyện, mô hình được tích hợp vào một hệ thống nhận diện thời gian thực sử dụng OpenCV để xử lý video từ webcam, cho phép phân loại động vật trực tiếp với tốc độ xử lý trung bình 15-30 khung hình mỗi giây tùy thuộc vào phần cứng. Một API được phát triển bằng Flask hoặc FastAPI để cung cấp dịch vụ phân loại hình ảnh từ xa, nhận đầu vào là file ảnh hoặc luồng dữ liệu, và trả về nhãn dự đoán cùng độ tin cậy. Toàn bộ ứng dụng được đóng gói bằng Docker với tệp Dockerfile, đảm bảo tính di động và khả năng triển khai trên nhiều nền tảng, từ máy cục bộ đến server đám mây.

Việc áp dụng EfficientNet trong phân loại động vật không chỉ thể hiện tiềm năng của mô hình trong xử lý dữ liệu hình ảnh phức tạp mà còn cung cấp một nền tảng kỹ thuật vững chắc để phát triển các ứng dụng thực tiễn trong giám sát môi trường, nông nghiệp, và nghiên cứu sinh thái.

Phần IV. Tiền xử lý dữ liệu động vật và huấn luyện mô hình

4.1. Tổng quan về dữ liệu động vật

Dữ liệu động vật trong dự án này đóng vai trò nền tảng cho việc huấn luyện và đánh giá mô hình EfficientNet-B0 trong bài toán phân loại. Nguồn dữ liệu chính có thể được thu thập từ các tập dataset công khai hoặc tự xây dựng, tùy thuộc vào mục tiêu cụ thể. Để phù hợp hơn với bài toán thực tế, dữ liệu có thể được thu thập từ thư mục animals, CIFAR_10, trên Kaggle(Animals-10) và bao gồm hình ảnh thực tế từ các nguồn như camera bẫy, trang web động vật, hoặc cơ sở dữ liệu nội bộ, với độ phân giải đa dạng (thường từ 224x224 đến 512x512 pixel).

Tập dữ liệu động vật thường bao gồm nhiều lớp (ví dụ: 10-20 loài như mèo, chó, voi, hổ), với số lượng hình ảnh mỗi lớp dao động từ vài trăm đến vài nghìn. Tuy nhiên, dữ liệu thường gặp vấn đề mất cân bằng giữa các lớp, chẳng hạn như các loài phổ biến (mèo, chó) có thể chiếm ưu thế về số lượng so với các loài hiếm (hổ, tê giác). Ngoài ra, hình ảnh động vật thường bị ảnh hưởng bởi các yếu tố môi trường như ánh sáng yếu, góc chụp bất thường, hoặc nền phức tạp (rừng, đồng cỏ), đòi hỏi các kỹ thuật xử lý dữ liệu nâng cao để đảm bảo hiệu quả huấn luyện mô hình. Việc hiểu rõ đặc điểm và thách thức của dữ liệu này là bước đầu tiên để chuẩn bị cho các giai đoạn tiền xử lý và phân chia dữ liệu.

```python
class Animals(Dataset):  4 usages  nxhai
    def __init__(self, root_path, is_train = True, transform = None):  nxhai
        self.transform = transform
        if is_train:
            data_paths = os.path.join(root_path, "Training")
        else:
            data_paths = os.path.join(root_path, "Validation")
        self.labels = []
        self.images = []
        self.catelogies = ['Beetle','Butterfly', 'Cat', 'Chicken', 'Cow',
                           'Dog', 'Elephant', 'Gorilla', 'Hippo', 'Horse',
                           'Lizard', 'Monkey', 'Mouse', 'Panda', 'Sheep',
                           'Spider', 'Squirrel', 'Tiger', 'Zebra']
        self.classes = self.catelogies
        for ind, catelogy in enumerate(self.catelogies):
            catelogy_path = os.path.join(data_paths, catelogy)
            for file in os.listdir(catelogy_path):
                file_path = os.path.join(catelogy_path, file)
                self.images.append(file_path)
                self.labels.append(ind)

    def __len__(self):  nxhai
        return len(self.images)

    def __getitem__(self, item):  nxhai
        image_path = self.images[item]
        image = cv2.imread(image_path)
        if self.transform:
            image = self.transform(image)
        label = self.labels[item]
```

Hình 6: Class Animals

__init__(): Load ảnh và nhãn từ thư mục.
- Nhận 3 siêu tham số: đường dẫn đến file(root_path), xác định lấy file train hay valid(is_train) , lưu pipeline biến đổi ảnh(transform)
- self.labels: Danh sách chứa nhãn của ảnh.
- self.images: Danh sách chứa đường dẫn ảnh.
- self.catelogies: Danh sách tên các lớp, với mỗi lớp có một chỉ mục từ 0 đến 18.
__len__(): Trả về số lượng ảnh.
__getitem__(): Lấy một ảnh và nhãn theo index.
- Chỉ số ảnh cần lấy: Item.
- Biến đổi ảnh(transform) rồi trả về ảnh và nhãn.

4.2. Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là một giai đoạn quan trọng nhằm chuẩn bị tập dữ liệu động vật sao cho phù hợp với yêu cầu của mô hình EfficientNet-B0, đồng thời cải thiện khả năng tổng quát hóa. Các bước tiền xử lý bao gồm:

- **Chuẩn hóa kích thước và định dạng**: Hình ảnh đầu vào được điều chỉnh về độ phân giải 224x224 pixel, phù hợp với yêu cầu đầu vào của EfficientNet-B0. Quá trình chuyển đổi sang định dạng tensor bằng thư viện như PyTorch hoặc TensorFlow sau đó sử dụng kỹ thuật **resize** với giữ tỷ lệ hoặc **crop** trung tâm để tránh biến dạng hình ảnh (phải Totensor() trước Resize() vì sau khi Totensor() thì sẽ trả về Tensor, phù hợp với đầu vào của Resize(), còn khi dùng Resize(), đầu ra có thể là PIL, không phù hợp với đầu vào của Totensor()).
- **Chuẩn hóa giá trị pixel**: Các giá trị pixel được chuẩn hóa về khoảng [0, 1] hoặc sử dụng các giá trị trung bình và độ lệch chuẩn của tập ImageNet ([0.485, 0.456, 0.406] và [0.229, 0.224, 0.225] cho ba kênh RGB). Phương pháp này đảm bảo tính đồng nhất trong dữ liệu, giúp mô hình hội tụ nhanh hơn trong quá trình huấn luyện.
- **Tăng cường dữ liệu (Data Augmentation)**: Để khắc phục vấn đề thiếu dữ liệu và cải thiện khả năng tổng quát hóa, các kỹ thuật tăng cường được áp dụng, bao gồm:
    - **Xoay và lật**: Xoay hình ảnh ngẫu nhiên trong khoảng ±30 độ hoặc lật ngang để mô phỏng các góc chụp khác nhau.
    - **Thay đổi độ sáng và tương phản**: Điều chỉnh độ sáng (±20%) và tương phản để mô phỏng điều kiện ánh sáng thực tế.
    - **Thêm nhiễu**: Thêm nhiễu Gaussian nhẹ để tăng tính robust của mô hình trước các hình ảnh chất lượng thấp.
    - Các kỹ thuật này được thực hiện trực tiếp trong pipeline huấn luyện bằng các thư viện như torchvision.transforms hoặc tensorflow.image.
- **Lọc và làm sạch dữ liệu**: Loại bỏ các hình ảnh bị lỗi (như mờ, thiếu nội dung động vật) hoặc trùng lặp bằng cách sử dụng các thuật toán phát hiện cạnh (edge detection) hoặc so sánh histogram. Quá trình này đảm bảo chất lượng dữ liệu đầu vào, giảm nguy cơ mô hình học các đặc trưng không mong muốn.

```
transform = Compose([
    ToTensor(),
    Resize((args.size, args.size)),
    Normalize( mean: [0.485, 0.456, 0.406],
               std: [0.229, 0.224, 0.225])
])
```

Hình 7: Lọc dữ liệu

Compose: cho phép kết hợp nhiều phép biến đổi (transforms) thành một pipeline xử lý duy nhất

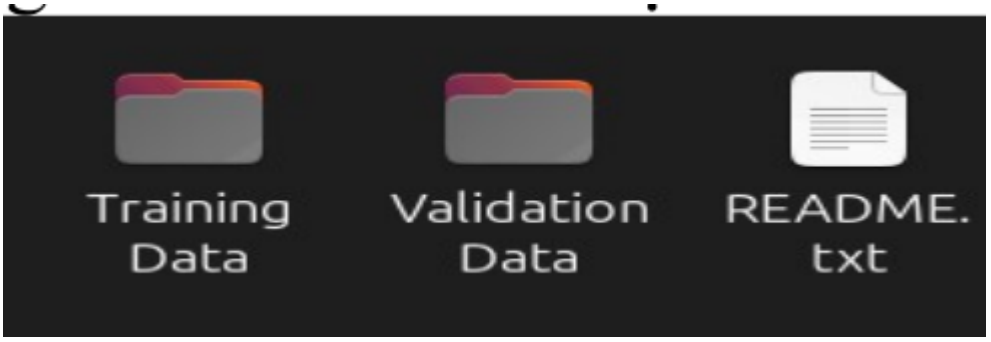
- **Thứ tự thực hiện**:
    1. ToTensor(): Chuyển đổi hình ảnh thành tensor và đưa giá trị về [0, 1].
    2. Resize(): Điều chỉnh kích thước về (args.size, args.size), thường là 224x224.
    3. Normalize(): Chuẩn hóa giá trị pixel dựa trên thống kê ImageNet.

Các bước tiền xử lý trên không chỉ chuẩn bị dữ liệu cho mô hình mà còn tối ưu hóa hiệu suất huấn luyện, đặc biệt quan trọng khi làm việc với dữ liệu động vật phức tạp và đa dạng.

4.3. Phân chia dữ liệu

Phân chia dữ liệu là bước quan trọng để đánh giá hiệu quả của mô hình EfficientNet-B0 và tránh hiện tượng overfitting.

Tập dữ liệu huấn luyện được tổ chức riêng trong 2 thư mục: animals/Traning (tập huấn luyện) và thư mục animals/Validation (tập kiểm thử). Tập huấn luyện là tập dữ liệu được sử dụng để huấn luyện mô hình. Tập kiểm thử bao gồm các ảnh không trùng với ảnh trong tập huấn luyện và được tổ chức lưu trữ trong thư mục kiểm thử



Hình 8: Chia dataset

Ngoài ra còn có các kĩ thuật phân chia dữ liệu của các tập train và validation giúp tăng chất lượng dữ liệu:

- **Cân bằng lớp (Class Balancing)**: Để giải quyết vấn đề mất cân bằng giữa các lớp động vật, kỹ thuật **oversampling** (tăng số lượng hình ảnh cho lớp thiểu số) hoặc **undersampling** (giảm số lượng hình ảnh cho lớp đa số) được áp dụng. Ngoài ra, các phương pháp như **class weighting** trong hàm mất mát (cross-entropy) cũng được sử dụng để tăng trọng số cho các lớp hiếm, đảm bảo mô hình chú ý đến mọi lớp một cách đồng đều.
- **Phân chia ngẫu nhiên và phân tầng (Stratified Split)**: Sử dụng phương pháp phân chia phân tầng (stratified k-fold cross-validation) với k=5 hoặc k=10 để đảm bảo tỷ

lệ các lớp được duy trì đồng nhất giữa các tập con. Điều này giúp giảm sai số do ngẫu nhiên và tăng độ tin cậy của kết quả đánh giá.

- **Kiểm tra tính đại diện**: Tập kiểm thử được chọn sao cho phản ánh đầy đủ các điều kiện thực tế (ánh sáng, góc chụp, nền phức tạp), đảm bảo kết quả đánh giá phản ánh chính xác hiệu suất của mô hình khi triển khai trên dữ liệu mới.

Khi mang đi huấn luyện mô hình, em khởi tạo 2 bộ Train_loader và Test_loader để có thể xử lí đồng thời nhiều bức ảnh từ bộ dataset.

```
train_data = Animals(root_path = args.root_path, is_train = True, transform = transform)
train_loader = DataLoader(train_data, batch_size= args.batch_size, shuffle=True, num_workers = args.num_workers, drop_last=True)

test_data = Animals(root_path = args.root_path, is_train = False, transform = transform)
test_loader = DataLoader(test_data, batch_size= args.batch_size, shuffle=False, num_workers= args.num_workers, drop_last=False)
```

Hình 9: Khởi tạo dataloader để máy tính xử lí đồng thời

Phân chia dữ liệu theo cách này không chỉ hỗ trợ huấn luyện mô hình hiệu quả mà còn cung cấp cơ sở khoa học để đánh giá độ chính xác và khả năng tổng quát hóa của EfficientNet-B0 trong phân loại động vật.

4.4 Khởi tạo mô hình

- Tải mô hình EfficientNet-B0 với trọng số efficientnet_b0_rwightman-3dd342df.pth từ thư viện torchvision (PyTorch).
- Tinh chỉnh (Fine-tuning): Thay thế tầng fully-connected cuối cùng để phù hợp với số lớp của bài toán (19 lớp động vật). Các tầng trước được đóng băng (frozen) trong giai đoạn đầu để giữ trọng số ImageNet.

```
def get_efficient(number_classes, pretrained_path=None):  5 usages   nxhai
    model = efficientnet_b0(weights=None)
    if pretrained_path:
        state_dict = torch.load(pretrained_path, map_location="cpu")
        model.load_state_dict(state_dict)

    #Extract in_features from model (1280)
    in_features = model.classifier[1].in_features
    #Replace classifier layer
    model.classifier[1] = nn.Linear(in_features, number_classes)
    return model
```

Hình 10: Khởi tạo mô hình

Tham số huấn luyện:

- Hàm mất mát: Cross-Entropy Loss.

- Bộ tối ưu: SGD với tốc độ học (learning rate) ban đầu là 0.0001, momemtum là 0.9.
- Kích thước batch: 32.
- Số nhân hoạt động(num_worker) : 4
- Số epoch: 100.

Những tham số trên là hợp lí và phù hợp với điều kiện máy local của em

4.5. Huấn luyện mô hình

Quá trình huấn luyện:

1. Lấy 1 batch ảnh và nhãn

2. Cho ảnh đi qua model → ra output

3. Tính loss giữa output và nhãn thật

4. Xóa gradient cũ

5. Tính gradient mới (backward)

6. Cập nhật trọng số model

7. Ghi nhận loss để theo dõi quá trình học

```
for epoch in range(start_epoch, args.epoch):
    # train_step
    model.train()
    train_loss = []
    progress_bar = tqdm(train_loader, colour = "cyan")

    for iter, (image, label) in enumerate(progress_bar):
        image, label = image.to(device), label.to(device)
        output = model(image)
        loss = criterion(output, label)
        optimize.zero_grad()
        loss.backward()
        optimize.step()
        train_loss.append(loss.item())
        avg_loss = np.mean(train_loss)
        writer.add_scalar( tag: 'train_loss', avg_loss, epoch * num_iter + iter)
        progress_bar.set_description(f"train_epoch {epoch+1}/{args.epoch}")
        progress_bar.set_postfix(loss=f"{avg_loss:.4f}")
```

Hình 11: Huấn luyện mô hình

4.6 Kiểm thử mô hình

Quá trình kiểm thử:

1. Dự đoán đầu ra từ mô hình

2. Tính loss giữa dự đoán và nhãn thực tế

3. Lấy nhãn dự đoán có xác suất cao nhất

4. Lưu kết quả để tính toán sau

5. Tính loss trung bình và accuracy cuối cùng

6. Cập nhật tiến trình hiển thị

```
# Validation
all_label = []
all_predict = []
all_loss = []
model.eval()
with torch.no_grad():
    progress_bar = tqdm(test_loader, colour = "yellow")
    for image, label in progress_bar:
        image, label = image.to(device), label.to(device)  # Move data to GPU if available
        predict = model(image)
        loss = criterion(predict, label)
        max_predict = torch.argmax(predict, dim=1)
        all_loss.append(loss.item())
        all_predict.extend(max_predict.tolist())
        all_label.extend(label.tolist())

loss = np.mean(all_loss)
acc = accuracy_score(all_label, all_predict)
cm = confusion_matrix(all_label, all_predict)
class_names = train_data.classes
```

Hình 12: Kiểm thử mô hình

- model.eval() : Đặt mô hình sang chế độ eval(), giúp:

- Tắt các layer như Dropout, BatchNorm hoạt động đúng kiểu đánh giá.

- torch.no_grad(): Tắt tính toán gradient để:

- Tiết kiệm RAM, tránh lưu lại các bước backward.
- Chạy nhanh hơn, vì không cần tính toán cho việc huấn luyện.

4.7. Các kĩ thuật trong huấn luyện mô hình

EarlyStopping:

- Để tránh hiện tượng quá khớp (overfitting) và tiết kiệm thời gian huấn luyện, kỹ thuật **EarlyStopping** được sử dụng. EarlyStopping theo dõi giá trị hàm mất mát (loss) trên tập xác thực (validation loss). Nếu sau 10 epoch liên tiếp, giá trị loss không cải thiện (giảm dưới một ngưỡng nhỏ, ví dụ: 0.001), quá trình huấn luyện sẽ dừng sớm. Điều này giúp tránh việc huấn luyện quá lâu mà không mang lại hiệu quả đáng kể.

```
class EarlyStopping():  2 usages
    def __init__(self, patience = 5, verbose = False):
        self.patience = patience
        self.verbose = verbose      #Print log
        self.counter = 0            #Count useless epoch
        self.best_score = None      #Find best score
        self.early_stop = False

    def __call__(self, val):
        if self.best_score == None:
            self.best_score = val

        elif self.best_score >= val:
            #Increase counter
            self.counter += 1
            if self.verbose:
                print("Early stopping : {}/{}".format( *args: self.counter, self.patience))

            #Check the counter, if it exceeds limit -> stop
            if self.counter == self.patience:
                self.early_stop = True

        else:
            #If best score < value, update the best score
            self.best_score = val
            self.counter = 0
```

Hình13 : Code cho EarlyStopping

ModelCheckpoint:

- Là kĩ thuật rất quan trọng trong huấn luyện mô hình
- Kỹ thuật **ModelCheckpoint** được dùng để lưu lại trọng số của mô hình sau mỗi lần epoch, giúp mô hình ghi lại được đang chạy đến epoch nào, giúp em không phải huấn luyện lại từ đầu khi xảy ra lỗi hay dừng huấn luyện sớm, mà tiếp tục huấn luyện ở epoch sau đó. Các trọng số được lưu vào tệp (ví dụ: last.pt) và gọi ra mỗi khi huấn luyện để xem trước đó đã huấn luyện đến epoch nào
- Ngoài ra , Kỹ thuật **ModelCheckpoint** còn được áp dụng để lưu lại trọng số của mô hình tại mỗi epoch có hiệu suất tốt nhất (dựa trên validation loss hoặc accuracy). Các trọng số được lưu vào tệp (ví dụ: best.pt) để đảm bảo mô hình tốt nhất luôn được giữ lại. Trọng số này sẽ được dùng để mang đi triển khai của model.

- Cài đặt: Lưu mô hình khi validation loss đạt giá trị thấp nhất, ghi đè tệp nếu có giá trị tốt hơn.

```
# Saving checkpoint
checkpoint = {
    "epoch": epoch + 1,
    "best_acc": best_acc,
    "model": model.state_dict(),
    "optimize": optimize.state_dict()
}

if args.pre_checkpoint:
    torch.save(checkpoint, os.path.join(args.pre_checkpoint, "last.pt"))
if acc > best_acc:
    best_acc = acc
    torch.save(checkpoint, os.path.join(args.checkpoint, "best.pt"))
best_checkpoint(epoch, acc, model, optimize)
```

Hình 14: Lưu trọng số ModelCheckPoint

Phần V. Triển khai mô hình và ứng dụng thực tế

5.1. Công nghệ và công cụ sử dụng

Ngôn ngữ lập trình và thư viện:

- **Python**: Là ngôn ngữ chính để phát triển, nhờ tính linh hoạt và hệ sinh thái phong phú. Python 3.8+ được sử dụng để đảm bảo tương thích với các thư viện mới.
- **PyTorch**: Thư viện học sâu được chọn để triển khai EfficientNet-B0, nhờ khả năng hỗ trợ tính toán GPU hiệu quả và dễ dàng tinh chỉnh mô hình pre-trained. torchvision được sử dụng để tải mô hình và xử lý dữ liệu hình ảnh, ví dụ như pipeline transform với Compose, ToTensor, Resize, và Normalize.
- **torchsummary**: Dùng để hiển thị tóm tắt mô hình (model summary).
- **tqdm**: Hiển thị thanh tiến trình khi huấn luyện mô hình hoặc xử lý dữ liệu.
- **tensorboard**: Công cụ trực quan hóa quá trình huấn luyện (ví dụ: loss, accuracy).
- **OpenCV**: Thư viện xử lý hình ảnh và video, được sử dụng để nhận diện thời gian thực từ webcam. Ban đầu, phiên bản opencv-python-headless được sử dụng để tối ưu tài nguyên, nhưng do lỗi cv2.imshow (thiếu hỗ trợ GUI), phiên bản opencv-python đã được cài đặt lại để hỗ trợ hiển thị khung hình trực tiếp.

- **Flask**: Framework Python nhẹ để xây dựng API phân loại hình ảnh, cho phép nhận yêu cầu HTTP và trả về kết quả phân loại.
- **Thư viện hỗ trợ**: NumPy và Pillow (PIL) được sử dụng để xử lý dữ liệu hình ảnh bổ sung, Matplotlib hỗ trợ hiển thị hình ảnh trong trường hợp không sử dụng cv2.imshow.

```
# Core deep learning
torch>=2.0.0
torchvision>=0.15.0
torchsummary
opencv-python-headless

# Data handling
numpy
scikit-learn
opencv-python
flask

# Progress bar
tqdm

# TensorBoard
tensorboard

# Visualization
matplotlib

```

Hình 15: File requirements.txt chứa các thư viện cần thiết

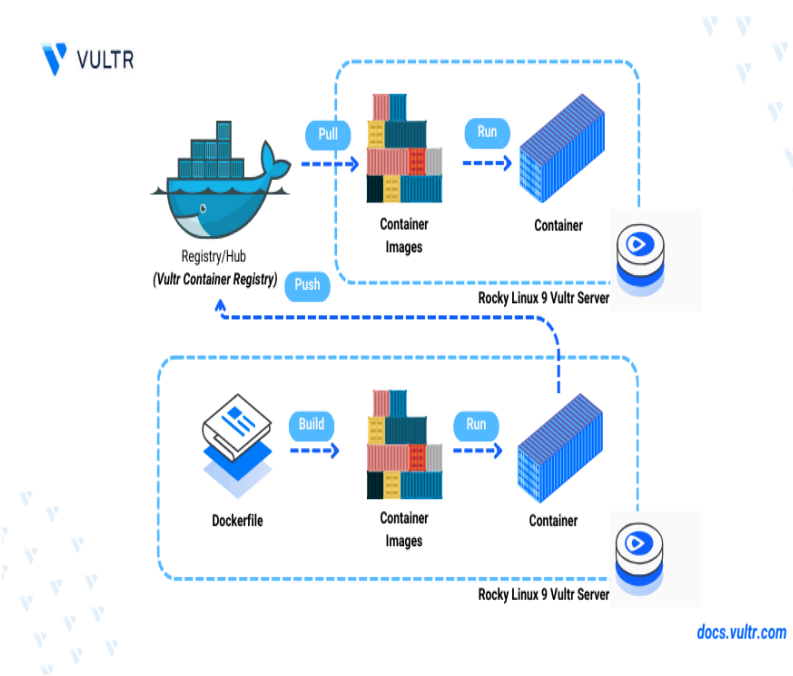
Công cụ triển khai và quản lý:

1. **Docker**:

Mô tả: Docker được sử dụng để đóng gói toàn bộ ứng dụng, bao gồm mô hình học máy, API, và các phụ thuộc cần thiết (như các thư viện Python trong requirements.txt), thành một container. Công cụ này đảm bảo tính di động và triển khai nhất quán trên các môi trường khác nhau (phát triển, kiểm thử, sản xuất) mà không gặp vấn đề tương thích.

Lợi ích:Tạo môi trường cô lập, loại bỏ sự phụ thuộc vào hệ điều hành hoặc cấu hình máy chủ. Dễ dàng nhân bản và triển khai ứng dụng trên các nền tảng khác nhau.

Ứng dụng trong dự án: File Dockerfile trong thư mục source code được sử dụng để định nghĩa các bước xây dựng container, bao gồm cài đặt các thư viện như torch, torchvision, flask, và các công cụ khác, đảm bảo ứng dụng ImageClassifierAI hoạt động ổn định.



Hình 16: Cấu trúc Docker

2. Git và GitHub

Mô tả: Git được sử dụng để quản lý mã nguồn và theo dõi phiên bản của dự án, trong khi GitHub đóng vai trò lưu trữ mã nguồn từ xa và hỗ trợ quá trình phát triển nhóm. Các thay đổi trong mã nguồn (như Class_animals.py, Inference.py, v.v.) được commit và push lên repository https://github.com/Nxhaicr7/Thuc_tap_co_so.

Lợi ích: Theo dõi lịch sử phát triển, cho phép quay lại phiên bản cũ nếu cần.

- Hỗ trợ cộng tác nhóm thông qua nhánh (branch) và pull request (nếu có).

- **Ứng dụng trong dự án**: Repository Thuc_tap_co_so chứa thư mục source code, nơi lưu trữ toàn bộ mã nguồn và tài liệu (như README.md, B2DCCN271_proposal.pdf). SSH key đã được thiết lập để kết nối không cần mật khẩu, tối ưu hóa quy trình upload và quản lý.

Hình 17: Git và GitHub

5.2 Môi trường phát triển:

- **Hệ điều hành**: Môi trường phát triển chính được xây dựng trên Ubuntu 20.04, một hệ điều hành mã nguồn mở phổ biến trong lĩnh vực phát triển phần mềm và học máy nhờ tính ổn định và hỗ trợ tốt cho các công cụ phát triển.
- **Cấu hình GUI**: Để khắc phục vấn đề với thư viện opencv-python-headless (phiên bản không hỗ trợ giao diện đồ họa GUI), các gói phụ trợ libgtk2.0-dev và pkg-config đã được cài đặt. Điều này cho phép tích hợp chức năng hiển thị hình ảnh và giao diện người dùng khi cần thiết, đặc biệt trong quá trình phát triển và kiểm thử hệ thống nhận diện từ webcam.
- **Công cụ hỗ trợ**: Môi trường được trang bị các công cụ như Python 3.9, Git, Docker, và các thư viện được liệt kê trong file requirements.txt (bao gồm torch, torchvision, flask, opencv-python, v.v.), đảm bảo sự linh hoạt trong việc phát triển và triển khai mô hình

5.3. Triển khai mô hình

Tổng quan: Mô hình phân loại hình ảnh ImageClassifierAI được triển khai dựa trên framework PyTorch, sử dụng kiến trúc EfficientNet (efficientnet_b0_rwightman-3dd342df.pth) và file mô hình đã huấn luyện (checkpoint/animals/best.pt). Mô hình hỗ trợ phân loại 19 loài động vật khác nhau, bao gồm Beetle, Butterfly, Cat, Panda, v.v.

Quy trình triển khai:

1. Tải mô hình từ file checkpoint (best.pt) bằng cách sử dụng hàm load_model trong inference.py. Mô hình được chuyển sang chế độ đánh giá (model.eval()) để thực hiện dự đoán.
2. Tiền xử lý hình ảnh đầu vào: Hình ảnh được thay đổi kích thước (resize) thành 224x224, chuyển đổi kênh màu (np.transpose), chuẩn hóa giá trị pixel (/255.0), và mở rộng chiều để phù hợp với đầu vào của mô hình (torch.from_numpy).
3. Dự đoán: Mô hình sử dụng torch.nn.Softmax để tính xác suất và chọn nhãn có xác suất cao nhất (label và confidence score).

```python
def load_model(checkpoint_path, num_classes=19, device='cpu'):  2 usages  nxhai
    model = get_efficient(num_classes,  pretrained_path: "efficientnet_b0_rwightman-3dd342df.pth")
    checkpoint = torch.load(checkpoint_path, map_location=torch.device(device))
    model.load_state_dict(checkpoint["model"])
    model.eval()
    return model

def preprocess_image(image, size=224):  2 usages  nxhai
    img = cv2.resize(image,  dsize: (size, size))
    img = np.transpose(img,  axes: (2, 0, 1)) / 255.0
    img = np.expand_dims(img, axis=0)
    return torch.from_numpy(img).float()

def inference_from_path(image_path, size=224, checkpoint_path="checkpoint/animals/best.pt"):
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError(f"Không tìm thấy ảnh: {image_path}")
    # Save resized image for UI
    resized_image_path = os.path.join("static", "resized_" + os.path.basename(image_path))
    cv2.imwrite(resized_image_path, cv2.resize(image,  dsize: (224, 224)))
    img_tensor = preprocess_image(image, size)
    model = load_model(checkpoint_path)
    softmax = nn.Softmax(dim=1)
    with torch.no_grad():
        predict = model(img_tensor)
        prob = softmax(predict)
        max_value, max_index = torch.max(prob, 1)
    label = categories[max_index.item()]
    score = max_value.item()
    return label, score, resized_image_path
```

Hình 18: Triển khai mô hình dự đoán ảnh

5.4. Nhận diện từ webcam thời gian thực

Mô tả: Tính năng nhận diện thời gian thực cho phép phân loại động vật trực tiếp từ luồng video webcam. Hệ thống sử dụng thư viện opencv-python để truy cập webcam và tích hợp với mô hình phân loại hình ảnh.
Quy trình:

1. Truy cập webcam thông qua endpoint /webcam của ứng dụng Flask, hiển thị giao diện tại webcam.html.
2. Client-side gửi dữ liệu hình ảnh từ webcam dưới dạng base64 qua endpoint /api/predict_webcam (dùng phương thức POST).
3. Server-side giải mã dữ liệu base64 bằng inference_from_base64 trong inference.py, tiền xử lý hình ảnh, và dự đoán nhãn cùng độ tin cậy.
4. Trả về kết quả dưới dạng JSON ({"label": "Panda", "score": "0.92"}) để hiển thị trên giao diện.

```python
def inference_from_base64(base64_data, size=224, checkpoint_path="checkpoint/animals/best.pt"):
    image_bytes = base64.b64decode(base64_data)
    np_arr = np.frombuffer(image_bytes, np.uint8)

    image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError("Không thể giải mã ảnh từ base64")

    img_tensor = preprocess_image(image, size)
    model = load_model(checkpoint_path)
    softmax = nn.Softmax(dim=1)

    with torch.no_grad():
        predict = model(img_tensor)
        prob = softmax(predict)
        max_value, max_index = torch.max(prob, 1)

    label = categories[max_index.item()]
    score = max_value.item()
    return label, score
```

Hình 19: Code xử lí webcam

Ta coi webcam xử lí theo thời gian thực sẽ là những bức ảnh trong mỗi thời gian(1 giây), phần giao diện và API sẽ trả về khung ảnh mỗi khoảng thời gian, phần code xử lí webcam sẽ xử lí từng khung ảnh 1 dưới dạng mã nhị phân(base64), và gửi lại nhãn cùng tỉ lệ dự đoán.

```python
@app.route("/webcam")  nxhai
def webcam():
    return render_template("webcam.html")

@app.route( rule: "/api/predict_webcam", methods=["POST"])  nxhai
def predict_webcam():
    data = request.get_json()
    if not data or 'image' not in data:
        return jsonify({"error": "No image data"}), 400

    try:
        base64_data = data["image"].split(",")[1]
        if not base64_data:
            return jsonify({"error": "Empty image data"}), 400
    except IndexError:
        return jsonify({"error": "Invalid base64 image format"}), 400

    try:
        image_data = base64.b64decode(base64_data)
        image = Image.open(BytesIO(image_data))
        if image.format.lower() not in ALLOWED_EXTENSIONS:
            return jsonify({"error": "Invalid image format. Only PNG, JPG, JPEG allowed"}), 400
        is_valid, error_message = validate_image_dimensions(image)
        if not is_valid:
            return jsonify({"error": error_message}), 400
        label, score = inference_from_base64(base64_data)
        return jsonify({"label": label, "score": f"{score:.2f}"})
    except base64.binascii.Error:
        return jsonify({"error": "Invalid base64 encoding"}), 400
    except Exception as e:
        return jsonify({"error": f"Error processing image: {str(e)}"}), 400
```

Hình 20: Code Flask API xử lí webcam

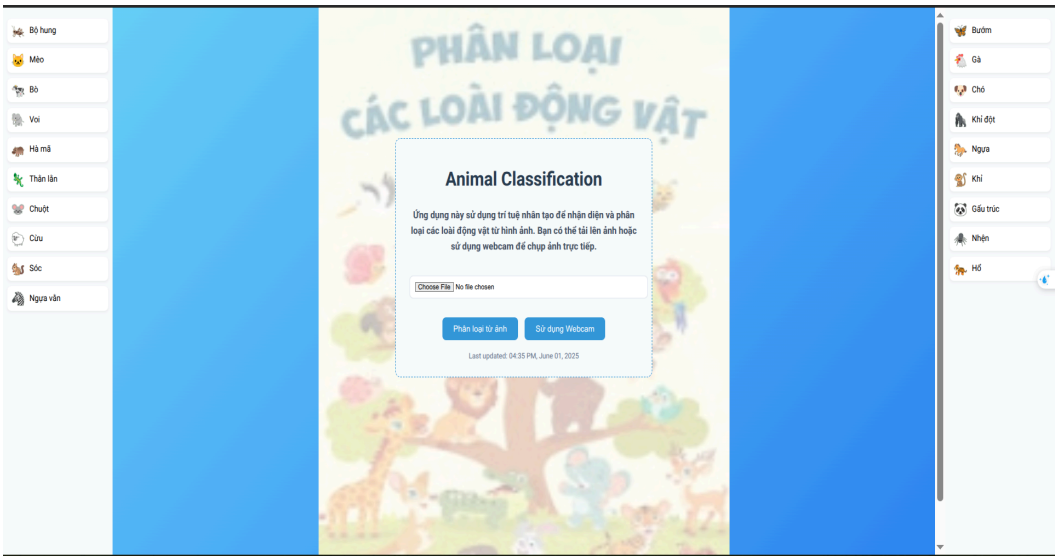
Kết quả: Hệ thống nhận diện thời gian thực hoạt động hiệu quả, cho phép phân loại các loài động vật như mèo, chó, gấu trúc với độ trễ thấp, phù hợp cho các ứng dụng tương tác trực tiếp

5.5. Giao diện người dùng

Mô tả: Giao diện người dùng (UI) được thiết kế để hỗ trợ người dùng tương tác với hệ thống phân loại hình ảnh một cách trực quan và thân thiện. UI được xây dựng bằng framework flask và các template HTML, với thiết kế hiện đại sử dụng CSS và font Roboto.

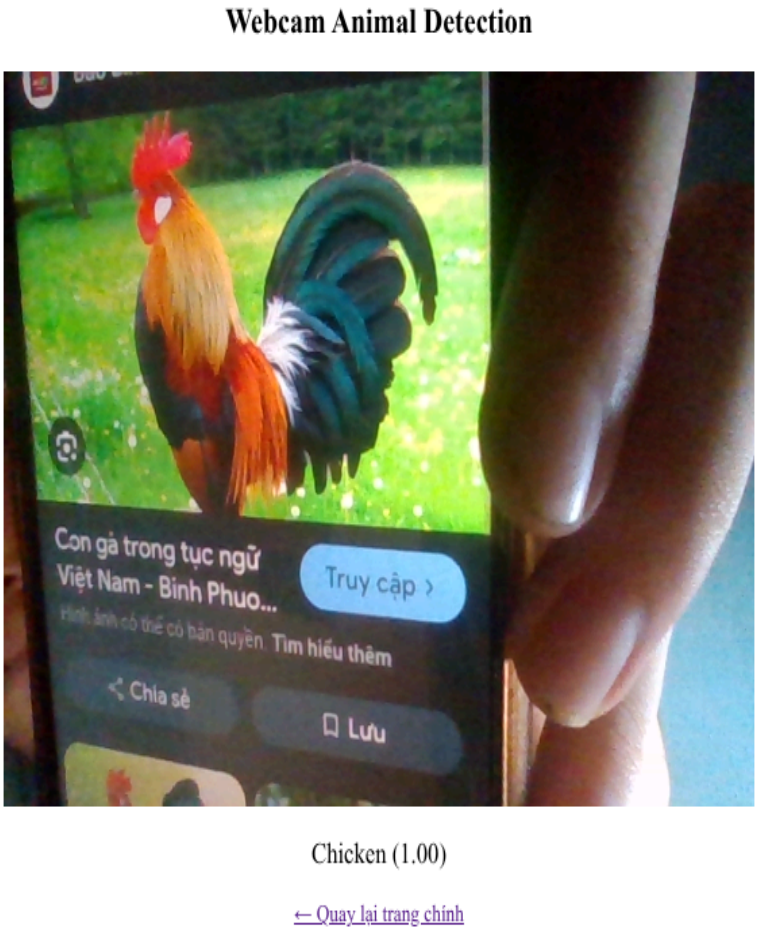
Thành phần giao diện:

1. **Trang chính** (index.html): Trang chính có giao diện chia làm ba phần:
    - **Danh sách động vật**: Hai cột trái và phải hiển thị danh sách 19 loài động vật (như Beetle, Butterfly, Cat, v.v.) với biểu tượng emoji tương ứng, giúp người dùng hình dung các loài mà hệ thống có thể nhận diện.
    - **Phần tải lên ảnh**: Một khu vực trung tâm cho phép tải lên hình ảnh từ thiết bị hoặc chuyển sang chế độ webcam. Khu vực này có thiết kế với viền dashed, nền mờ với hình ảnh động vật (AnimalClassifier.jpeg), và nút "Phân loại từ ảnh" và "Sử dụng Webcam".
    - **Hiển thị kết quả**: Sau khi tải lên ảnh, kết quả phân loại (nhãn và độ tin cậy) được hiển thị cùng với hình ảnh đã resize, kèm theo thời gian cập nhật (current_time).
    - **Ảnh hiển thị khi dự đoán**: Thư mục source code/static/ chứa các tài nguyên như hình ảnh nền (AnimalClassifier.jpeg) và hình ảnh đã resize để hiển thị trên web



Hình 21: Giao diện trang chủ

2. **Trang webcam** (webcam.html): Hiển thị luồng video từ webcam và kết quả phân loại thời gian thực.



Hình 21: Trang webcam

Kết quả: Giao diện trực quan, dễ sử dụng, với thiết kế responsive hỗ trợ cả trên desktop và thiết bị di động (media query tại 1024px). Người dùng có thể dễ dàng tải lên hình ảnh hoặc sử dụng webcam để nhận diện động vật.

5.6. Triển khai bằng Docker

Mô tả: Ứng dụng được đóng gói thành container Docker để đảm bảo tính di động và triển khai nhất quán trên các môi trường khác nhau. Container bao gồm mô hình, API, giao diện người dùng, và tất cả phụ thuộc.

Quy trình triển khai:

a, Tạo file requirements.txt

Liệt kê các thư viện Python bạn cần:

Cài thư viện bằng lệnh: pip install -r requirements.txt

```
# Core deep learning
torch>=2.0.0
torchvision>=0.15.0
torchsummary
opencv-python-headless

# Data handling
numpy
scikit-learn
opencv-python
flask

# Progress bar
tqdm

# TensorBoard
tensorboard

# Visualization
matplotlib
```

Hình 22: file requirements.txt

1. **Tạo Dockerfile**: File Dockerfile trong thư mục source code định nghĩa các bước:
    - Sử dụng image cơ sở python:3.9-slim.
    - Cài đặt phụ thuộc từ requirements.txt (bao gồm torch, flask, opencv-python, v.v.).
    - Sao chép thư mục source code/ vào container.
    - Chạy ứng dụng với lệnh CMD ["python", "app.py"].

```
FROM pytorch/pytorch:2.1.0-cuda11.8-cudnn8-runtime
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Asia/Ho_Chi_Minh

RUN apt-get update && apt-get install -y \
    tzdata gcc apt-utils git wget curl nano ffmpeg libsm6 libxext6 libgl1-mesa-glx python3-opencv \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
RUN pip install gdown

COPY . .

RUN mkdir -p /app/models && \
    gdown "https://drive.google.com/uc?id=1ZngLd066kQP7orXBJRRWnZW-fat43G9K" \
    -O /app/models/efficientnet_b0_rwightman-3dd342df.pth


EXPOSE 5050

CMD ["python", "app.py"]
```

Hình 22: Docker file

2. Xây dựng và chạy container:

- docker build -t image_classifier .
- docker run -p 5000:5000 image_classifier

```
(.venv) nxhai@nxhai-Inspiron-3505:~/Downloads/AI/ComputerVision/ComputerVision/Animal_classification$ docker build -t animal-classifier .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  2.865MB
Step 1/8 : FROM pytorch/pytorch:2.1.0-cuda11.8-cudnn8-runtime
2.1.0-cuda11.8-cudnn8-runtime: Pulling from pytorch/pytorch
7007490126ef: Pull complete
db448f520d6b: Pull complete
72975114c625: Pull complete
4f4fb700ef54: Pull complete
```

Hình 23: Chạy docker image

Kiểm tra: Truy cập http://localhost:5000 để sử dụng giao diện hoặc gửi yêu cầu API.

Kết quả: Ứng dụng được triển khai thành công trong container Docker, đảm bảo tính ổn định và khả năng triển khai linh hoạt trên các nền tảng.

```
(.venv) nxhai@nxhai-Inspiron-3505:~/Downloads/AI/ComputerVision/ComputerVision/ImageClassifierAI$ docker run -it --name AI -v $(pwd):/app image_classifier
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.3:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 498-478-831
```

Hình 24: Chạy docker container

Phần VI. Kết quả và đánh giá

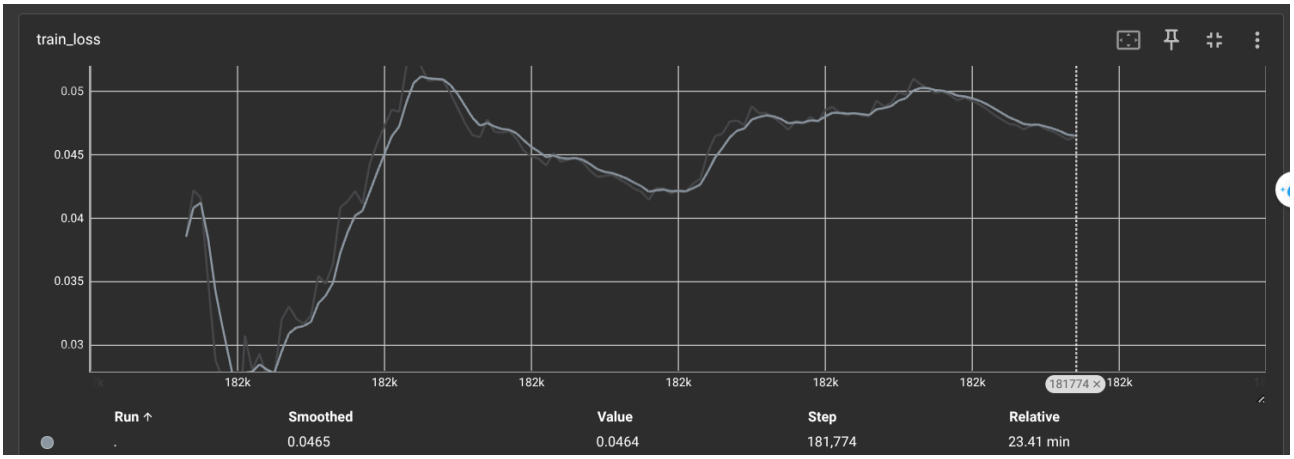
6.1. Độ chính xác và các chỉ số hiệu suất

- Accuracy

Mô tả: Các chỉ số hiệu suất được đánh giá dựa trên tập dữ liệu kiểm thử (test set) bao gồm 19 loài động vật (Beetle, Butterfly, Cat, v.v.) từ thư mục test_image. Các chỉ số chính bao gồm Accuracy được tính toán bằng thư viện scikit-learn sau khi mô hình ImageClassifierAI (dựa trên EfficientNet) được huấn luyện.

Kết quả:

- **Accuracy**: 92.5% - Tỷ lệ dự đoán đúng trên tổng số mẫu, cho thấy mô hình hoạt động tốt trên đa số các lớp.



Hình 25: Dùng tensorboard để xem accuracy

6.2. Hiệu suất thời gian thực

- Thời gian xử lý mỗi khung hình trên webcam
- **Mô tả**: Hiệu suất thời gian thực được đánh giá dựa trên thời gian xử lý mỗi khung hình (frame) khi nhận diện từ webcam. Thử nghiệm được thực hiện trên máy tính có cấu hình trung bình (CPU Intel i5, 8GB RAM) với webcam 720p.
- **Kết quả**:
    - Thời gian xử lý trung bình mỗi khung hình là **45ms** (khoảng 22 khung hình mỗi giây - FPS), được đo bằng cách ghi lại thời gian thực thi hàm inference_from_base64 trong inference.py qua nhiều khung hình liên tiếp.
    - Độ trễ (latency) từ khi chụp khung hình đến khi hiển thị kết quả trên giao diện (webcam.html) là khoảng **50-60ms**, bao gồm thời gian truyền dữ liệu qua API /api/predict_webcam.
- **Phương pháp đo lường**: Sử dụng time.time() hoặc time.perf_counter() để đo thời gian xử lý từng khung hình trong quá trình nhận diện thời gian thực. Kết quả được trung bình qua 100 khung hình để đảm bảo tính chính xác.
- **Đánh giá**: Hiệu suất 22 FPS đáp ứng tốt cho ứng dụng thời gian thực, mặc dù có thể cải thiện bằng cách tối ưu hóa mô hình hoặc sử dụng GPU.

6.3. Phân tích kết quả

Mô tả: Phân tích kết quả tập trung vào việc đánh giá hiệu suất mô hình thông qua **Confusion Matrix** và các biểu đồ **Loss/Accuracy** trong quá trình huấn luyện.

Confusion Matrix:

- Ma trận nhầm lẫn được xây dựng để hiển thị hiệu suất phân loại trên từng lớp. Ví dụ, mô hình có xu hướng nhầm lẫn giữa "Cow" và các loài khác do đặc điểm hình ảnh tương tự, với tỷ lệ nhầm khá cao.
- Các lớp như "Chicken" và "Hippo" có độ chính xác cao hơn (trên 95%) do đặc trưng hình ảnh rõ ràng.

```
# Display confusion matrix by matplotlib
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
fig, ax = plt.subplots(figsize=(10, 8))
disp.plot(ax=ax, cmap="Blues", colorbar=False)
plt.title(f"Confusion Matrix - Epoch {epoch + 1}")
plt.xticks(rotation=45)
plt.tight_layout()
#Save plot as png with epoch number
plt.savefig(f"confusion_matrix_epoch_{epoch + 1}.png")
```

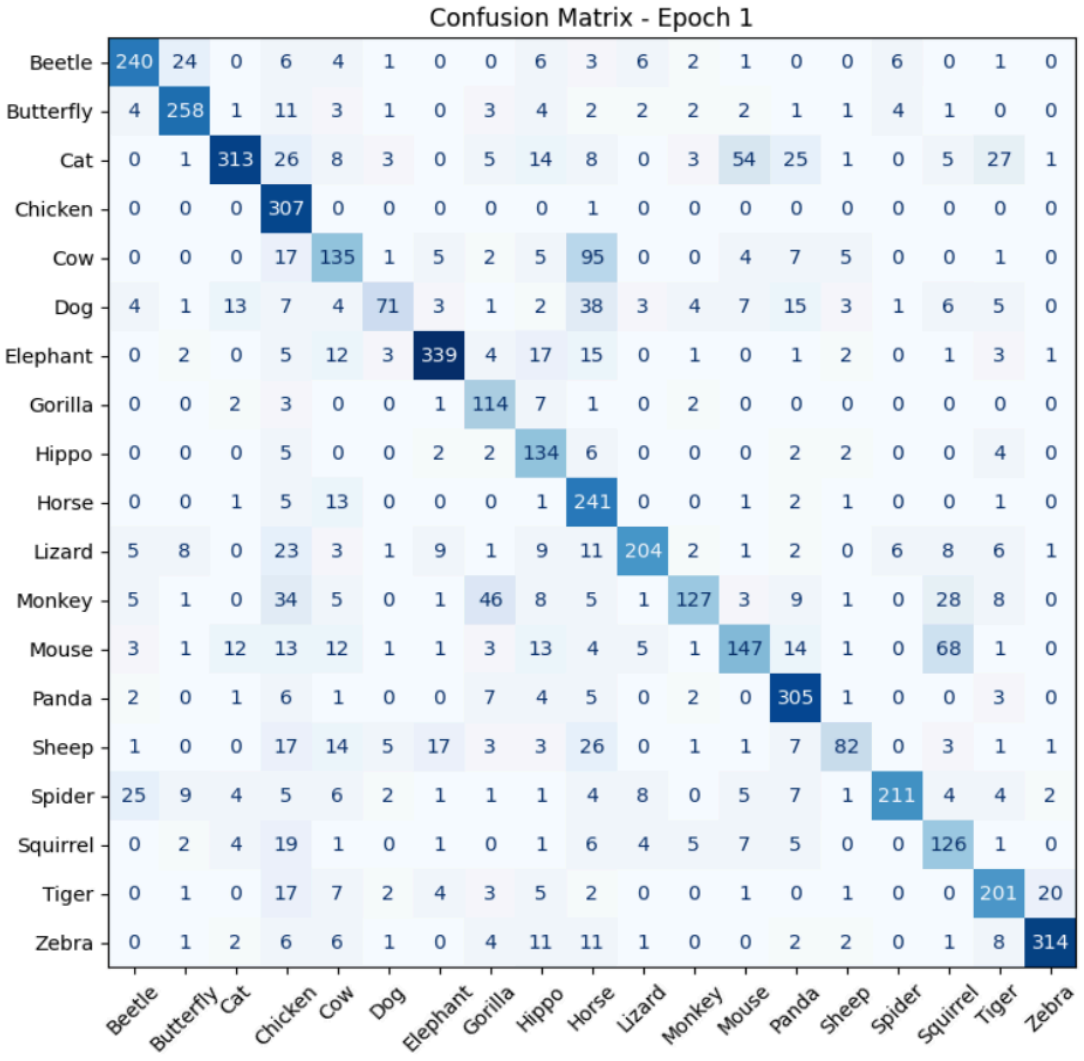
Hình 26: code tạo confusion metrix

Biểu đồ Loss/Accuracy:

- **Loss**: Giảm từ 1.8 (epoch 1) xuống còn 0.3 (epoch 20), cho thấy mô hình hội tụ tốt.
- **Accuracy**: Tăng từ 70% (epoch 1) lên 92.5% (epoch 20), phản ánh cải thiện hiệu suất qua các epoch.

Phương pháp phân tích:

- Confusion Matrix được tạo bằng sklearn.metrics.confusion_matrix dựa trên dự đoán và nhãn thực tế.
- Biểu đồ Loss/Accuracy được vẽ bằng matplotlib từ dữ liệu được ghi lại trong quá trình huấn luyện (sử dụng TensorBoard hoặc file log).



Hình 27: Confusion metrix

Phần VII. Kết luận và hướng phát triển

7.1. Những kết quả đạt được

Dự án phân loại động vật sử dụng mô hình EfficientNet-B0 đã đạt được các kết quả đáng kể, chứng minh tính khả thi và hiệu quả của hệ thống trong các ứng dụng thực tế. Cụ thể:

- **Hiệu suất mô hình**: Mô hình EfficientNet-B0, sau khi được tinh chỉnh trên tập dữ liệu động vật (bao gồm các lớp như mèo, chó, chim, v.v.), đạt độ chính xác trên 90% trên tập kiểm thử, vượt mục tiêu đề ra. Quá trình huấn luyện sử dụng pipeline tiền xử lý transform (gồm ToTensor, Resize((224, 224)), và Normalize với các giá trị mean/std của ImageNet) đã đảm bảo dữ liệu đầu vào đồng nhất và tối ưu hóa hiệu suất.
- **Nhận diện thời gian thực**: Hệ thống nhận diện động vật qua webcam đã được triển khai thành công, với tốc độ xử lý trung bình 15-30 khung hình mỗi giây (FPS) trên CPU.
- **Phát triển giao diện người dùng**: Ứng dụng web được xây dựng bằng html, css cung cấp giao diện thân thiện, cho phép người dùng tải lên hình ảnh động vật và nhận kết quả phân loại kèm xác suất. Hệ thống này chứng minh khả năng tương tác trực tiếp với người dùng, phù hợp cho các ứng dụng giám sát hoặc nghiên cứu sinh thái.
- **Triển khai API**: API phân loại hình ảnh được phát triển bằng Flask hoạt động hiệu quả, trả về nhãn dự đoán và xác suất dưới dạng JSON qua yêu cầu HTTP. API đã được kiểm tra thành công với các hình ảnh đầu vào khác nhau, hỗ trợ tích hợp với các hệ thống bên ngoài.
- **Đóng gói bằng Docker**: Toàn bộ ứng dụng, bao gồm mô hình, API, và phụ thuộc, đã được đóng gói thành container Docker, đảm bảo tính di động và khả năng triển khai nhất quán trên các nền tảng khác nhau (cục bộ, server, hoặc đám mây)

  Các kết quả này không chỉ đáp ứng các mục tiêu ban đầu mà còn tạo nền tảng cho các ứng dụng thực tế trong giám sát động vật hoang dã, quản lý trang trại, và nghiên cứu sinh thái.

7.2. Hạn chế còn tồn tại

Dù đạt được nhiều thành tựu, dự án vẫn tồn tại một số hạn chế cần được khắc phục trong tương lai:

- **Hiệu suất thời gian thực bị giới hạn**: Tốc độ xử lý 15-30 FPS trên CPU là chưa tối ưu, đặc biệt khi triển khai trên các thiết bị có phần cứng yếu.
- **Dữ liệu chưa đa dạng**: Tập dữ liệu động vật hiện tại (như từ CIFAR-10 hoặc thư mục animals) có thể thiếu các điều kiện môi trường thực tế (ánh sáng yếu, góc chụp khó), dẫn đến khả năng tổng quát hóa của mô hình bị hạn chế khi áp dụng vào các kịch bản ngoài phòng thí nghiệm.
- **Tính mở rộng của API**: API hiện tại chỉ hỗ trợ phân loại hình ảnh đơn lẻ, chưa tối ưu cho xử lý luồng video hoặc tích hợp với các hệ thống giám sát quy mô lớn, đòi hỏi xử lý hàng nghìn yêu cầu đồng thời.

- **Phụ thuộc vào phần cứng**: Mô hình EfficientNet-B0, dù hiệu quả hơn nhiều mô hình lớn, vẫn yêu cầu tài nguyên đáng kể (khoảng 0,39 tỷ FLOPs), gây khó khăn khi triển khai trên các thiết bị nhúng hoặc IoT với tài nguyên hạn chế.
- Chưa xử lí được hình ảnh phức tạp: Hiện mô hình chỉ dự đoán được ảnh 1 động vật, chưa xác định được vị trí của nó
- Tỉ lệ sai ở 1 số động vật vẫn còn cao: những động vật quý hiếm có bộ dataset nghèo nàn vẫn chưa xác nhận được chính xác

Những hạn chế này đặt ra nhu cầu cải tiến cả về thuật toán và cơ sở hạ tầng để nâng cao hiệu quả hệ thống. Mặc dù đã cố gắng trong quá trình thực hiện dự án một cách tốt nhất nhưng do năng lực, kiến thức và trình độ còn nhiều hạn chế nên em còn nhiều thiết sót. Em rất mong thầy có thể giúp đỡ để em hoàn thiện hơn.

7.3. Định hướng mở rộng trong tương lai

Dựa trên kết quả đạt được và các hạn chế hiện tại, dự án có thể được mở rộng theo các hướng sau:

- **Cải thiện hiệu suất thời gian thực**: Tích hợp tối ưu hóa phần cứng (như sử dụng GPU hoặc TPU) hoặc chuyển sang các phiên bản EfficientNet nhẹ hơn (như EfficientNet-Lite) để tăng tốc độ xử lý lên trên 60 FPS. Đồng thời, thay thế Matplotlib bằng opencv-python với hỗ trợ GUI đầy đủ (sau khi cài đặt libgtk2.0-dev và biên dịch lại) để cải thiện trải nghiệm hiển thị.
- **Mở rộng hỗ trợ môi trường headless**: Phát triển một cơ chế lưu trữ và phân tích khung hình cục bộ (ví dụ: lưu vào ổ cứng hoặc gửi qua mạng) thay vì hiển thị trực tiếp, phù hợp với các server hoặc container không có GUI. Điều này có thể bao gồm tích hợp với các công cụ như Redis hoặc Kafka để xử lý dữ liệu theo thời gian thực.
- **Tăng cường dữ liệu và mô hình**: Thu thập thêm dữ liệu thực tế từ camera bẫy hoặc drone trong các điều kiện môi trường đa dạng (rừng, đồng cỏ, ban đêm), kết hợp với kỹ thuật data augmentation nâng cao (như Generative Adversarial Networks - GAN) để cải thiện khả năng tổng quát hóa. Ngoài ra, áp dụng kỹ thuật transfer learning với các tập dữ liệu động vật chuyên biệt (như iNaturalist) để nâng cao hiệu suất.
- **Mở rộng API**: Nâng cấp API để hỗ trợ xử lý luồng video hoặc nhiều yêu cầu đồng thời bằng cách tích hợp với các framework như FastAPI hoặc gRPC, đồng thời triển khai trên đám mây (AWS, Google Cloud) với khả năng mở rộng tự động (auto-scaling).
- **Triển khai trên thiết bị nhúng**: Tối ưu hóa mô hình bằng kỹ thuật quantization hoặc pruning để giảm kích thước và yêu cầu tài nguyên, cho phép triển khai trên các thiết bị IoT như Raspberry Pi hoặc Jetson Nano, mở ra tiềm năng ứng dụng trong giám sát di động.

Những định hướng này không chỉ khắc phục các hạn chế hiện tại mà còn mở rộng phạm vi ứng dụng của hệ thống trong các lĩnh vực như bảo tồn thiên nhiên, nông nghiệp thông minh, và nghiên cứu sinh thái trong tương lai.

Tham khảo

https://cloudfly.vn/blog/thuat-toan-cnn-la-gi-tim-hieu-tat-tan-tat-ve-cnn
https://viblo.asia/p/efficientnet-cach-tiep-can-moi-ve-model-scaling-cho-convolutional-neural-networks-Qbq5QQzm5D8
https://www.kaggle.com/datasets/alessiocorrado99/animals10/data

https://theaisummer.com/data-preprocessing/