**TRƯỜNG ĐẠI HỌC SƯ PHẠM KĨ THUẬT THÀNH PHỐ HỒ CHÍ MINH**

🟑🟑🟑🙡🕮🙣🟑🟑🟑

**KHOA CƠ KHÍ CHẾ TẠO MÁY**

**BỘ MÔN CƠ ĐIỆN TỬ**



**BÁO CÁO CUÔI KÌ**

**TRÍ TUỆ NHÂN TẠO**

**NHẬN DIỆN, PHÂN LOẠI ĐỘNG VẬT BẰNG MẠNG NƠ RON NHÂN TẠO**

**GVHD: PGS.TS Nguyễn Trường Thịnh**

**MHP: ARIN337629\_22\_2\_08**

**SVTH: Vũ Đức Bình**

**MSSV: 20146478**

**MỤC LỤC**

[CHƯƠNG I: TỔNG QUAN 3](#_Toc135519298)

[**1.1** **Giới thiệu về đề tài** 3](#_Toc135519299)

[**1.2** **Nguyên do chọn đề tài** 4](#_Toc135519300)

[**Mục tiêu nghiên cứu:** 5](#_Toc135519301)

[**1.3** **Phương pháp nghiên cứu:** 5](#_Toc135519302)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 5](#_Toc135519303)

[**2.1 Giới thiệu về mô hình mạng nơ ron tích chập CNN (Convolution Neural Network)** 5](#_Toc135519304)

[**2.1.1 Thuật toán CNN (Convolution Neural Network) là gì?** 5](#_Toc135519305)

[**2.1.2 Các tham số chính trong CNN** 7](#_Toc135519306)

[**b. Bước nhảy - Stride** 9](#_Toc135519307)

[**c. Đường viền - Padding** 9](#_Toc135519308)

[**d. Hàm phi tuyến - ReLU** 9](#_Toc135519309)

[**e. Lớp gộp - Pooling Layer** 10](#_Toc135519310)

[**2.1.3 Cấu trúc bên trong mạng CNN** 11](#_Toc135519311)

[**2.2 Một số loại thư viện được sử dụng** 12](#_Toc135519312)

[**2.2.1 Thư viện Keras** 12](#_Toc135519313)

[**2.2.2 Thư viện Sciit-learn (sklearn)** 14](#_Toc135519314)

[**2.2.3 Thư viện keras** 15](#_Toc135519315)

[CHƯƠNG III: ỨNG DỤNG 15](#_Toc135519316)

[**3.1 XÂY DỰNG MÔ HÌNH CHO TRAINING MODEL** 15](#_Toc135519317)

[**1.** **Tạo ra tệp dữ liệu (datasets) cho việc training model** 15](#_Toc135519318)

[**2.** **Khai báo các thư viện cần thiết** 15](#_Toc135519319)

[**3.** **Lấy dữ liệu đã được upload trên google drive** 16](#_Toc135519320)

[**4.** **Thể hiện ngẫu nhiên một số ảnh có trong tệp dữ liệu** 16](#_Toc135519321)

[**5.** **Tăng cường tệp dữ liệu** 17](#_Toc135519322)

[**Xây dựng mô hình Convolutional neural network (CNN)** 19](#_Toc135519323)

[**6.** **Huấn luyện mô hình** 23](#_Toc135519324)

[**7.** **Xây dựng web-app** 26](#_Toc135519325)

[CHƯƠNG 4: KẾT LUẬN 26](#_Toc135519326)

[**4.1 Mô hình chẩn đoán** 27](#_Toc135519327)

[**4.1.1 Mô hình nhận diện động vật** 27](#_Toc135519328)

[**4.1.2 Webapp được xây dựng trên Streamlit** 27](#_Toc135519329)

[**4.2 Ưu và nhược điểm** 30](#_Toc135519330)

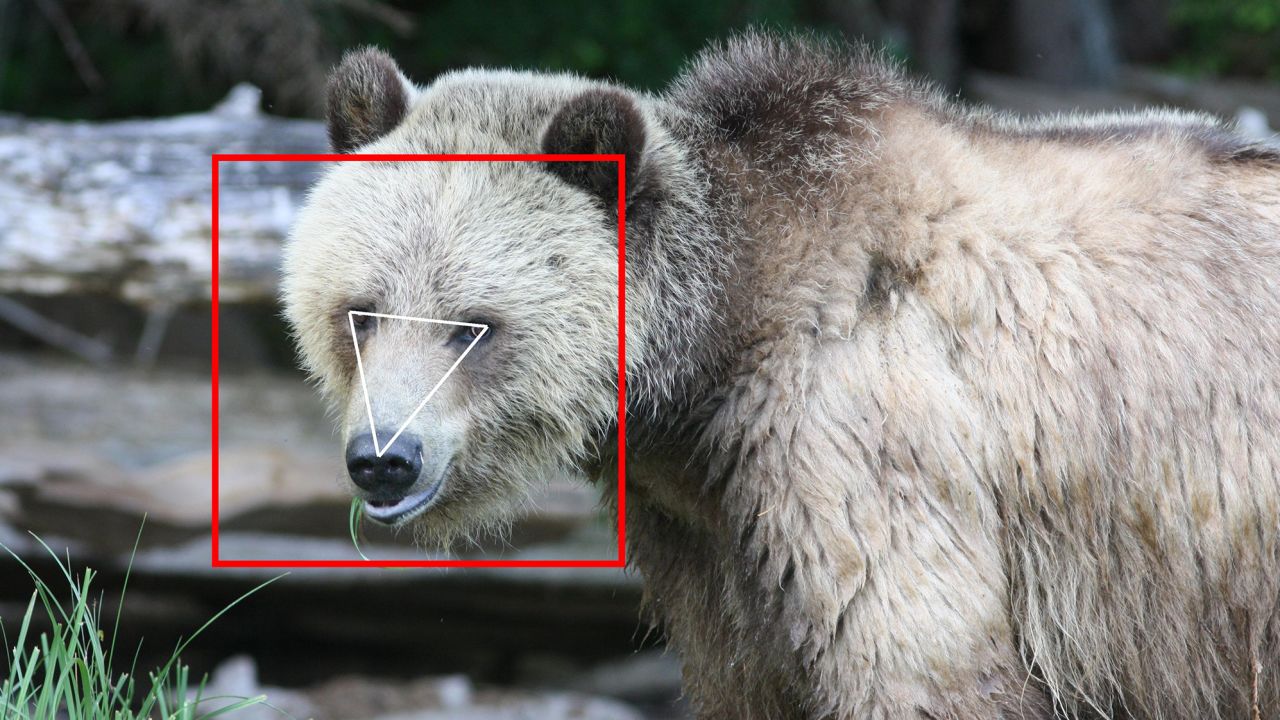
[**4.3 Hướng phát triển** 31](#_Toc135519331)

[TÀI LIỆU THAM KHẢO 31](#_Toc135519332)

# CHƯƠNG I: TỔNG QUAN

* 1. **Giới thiệu về đề tài**

Nhận dạng động vật là một trong những lĩnh vực nghiên cứu quan trọng trong khoa học máy tính và trí tuệ nhân tạo, với mục đích phát triển các phương pháp và công nghệ để nhận dạng và phân loại các loài động vật thông qua hình ảnh, âm thanh hoặc các dữ liệu khác.



Hình 1.1 Hình ảnh nhận dạng gấu xám (Nguồn CNN Business)

Ngày nay, một số loài động vật đang dần đi đến bờ vực của sự tuyệt chủng, việc phát triển và quản lý công nghệ nhận dạng động vật có thể giúp cho việc kiểm soát, quản lý nguồn tài nguyên động vật, thực vật hoang dã và bảo vệ các loài động vật đang bị đe dọa. Công nghệ này cũng có thể được áp dụng để giúp nghiên cứu về hành vi và sinh thái học của các loài động vật.

Các phương pháp nhận dạng động vật có thể được thực hiện bởi các thuật toán máy học và trí tuệ nhân tạo, bao gồm mạng neural và học sâu. Những phương pháp này có thể được sử dụng để phát triển các mô hình dự đoán và phân loại cho các loài động vật, dựa trên các đặc trưng và thuộc tính của chúng.

Để thực hiện đề tài nhận dạng động vật, người nghiên cứu có thể sử dụng các công cụ, thư viện được phát triển bằng ngôn ngữ Python như OpenCV, TensorFlow, và PyTorch. Các nghiên cứu về đề tài này cũng có thể áp dụng trên nhiều lĩnh vực, từ nghiên cứu sinh thái học đến bảo vệ động vật hoang dã, và đóng góp tích cực cho việc quản lý tài nguyên và bào vệ môi trường.

* 1. **Nguyên do chọn đề tài**

Có nhiều lý để phát triển công nghệ nhận đạng động vật:

1. Theo dõi động vật hoang dã: Nhận dạng động vật có thể giúp các nhà khoa học và nhân viên quản lý tài nguyên động vật hoang dã theo dõi các hoạt động của động vật, bao gôm số lượng, phân bố và di chuyển của chúng trong môi trường sông tự nhiên. Qua đó giúp các nhà tài nguyên tự nhiên có thể xác định khu vực đang bị ảnh hưởng và đưa ra các biện pháp giải quyết tốt hơn.
2. Bảo vệ động vật hoang dã: Công nghệ nhận dạng động vật cũng có thể giúp phát hiện các hoạt động săn băn trái phép hoặc buôn bán động vật trái phép, từ đó giúp bảo vệ các loài động vật hoang dã đang bị đe dọa.
3. Giám sát hành vi động vật: Các phương pháp nhận dạng động vật cũng có thể giúp nhà khoa học nghiên cứu hành vi và sinh thái học cùa các loài động vật, từ đó đưa ra những phương án quẩn lý và bảo vệ tôi ưu cho chúng.
4. Giám sát vệ sinh thực phẩm: Nhận dạng động vật cũng có thể được sử dụng trong giám sát vệ sinh thực phẩm, để phát hiện các loại thực phẩm bị nhiêm khuẩn hoặc bị ô nhiễm bởi các tác nhân có hại.
5. Nghiên cứu về bệnh tật của động vật: Công nghệ nhận dạng động vật cũng có thể giúp các nhà khoa học nghiên cứu và chẩn đoán các bệnh tật của động vật, từ đó giúp phát hiện và phòng ngừa các dịch bệnh động vật trong môi trường sống tự nhiên.
6. Phát triển kinh tế địa phương: Nhận dạng động vật cũng có thể giúp các địa phương phát triển về kinh tế. Ví dụ như việc nhận dạng các loài cá trong các con suối hoặc song có thể giúp tăng sản lượng, đánh bắt hải sản một cách hiệu quả và bền vững

Tóm lại, đề tài nhận dạng động vật có nhiêu ứng dụng hữu ích và đóng vai trò quan trọng trong việc quản lý tài nguyên động vật, bảo vệ môi trường và cải thiện chất lượng cuộc sống.

## **Mục tiêu nghiên cứu:**

* Xây dựng mô hình phân loại, nhận dạng động vật bằng phương pháp mạng nơ-ron tích chập (CNN)
* Tạo ra một website cơ bản, với giao diện dễ sử dụng để nhiều người có thể tiếp cận
  1. **Phương pháp nghiên cứu:**
* Tiếp cận, thu thập dữ liệu cụ thể là hình ảnh của các loài động vật.
* Xây dựng model phù hợp với độ chính xác cao, cụ thể là CNN.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## **2.1 Giới thiệu về mô hình mạng nơ ron tích chập CNN (Convolution Neural Network)**

### **2.1.1 Thuật toán CNN (Convolution Neural Network) là gì?**

Trong mạng neural, mô hình mạng neural tích chập (CNN) là 1 trong những mô hình để nhận dạng và phân loại hình ảnh. Trong đó, xác định đối tượng và nhận dạng khuôn mặt là 1 trong số những lĩnh vực mà CNN được sử dụng rộng rãi.

CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định (Ví dụ: Chó, Mèo, Hổ, ...). Máy tính coi hình ảnh đầu vào là 1 mảng pixel và nó phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy H x W x D (H: Chiều cao, W: Chiều rộng, D: Độ dày). Ví dụ: Hình ảnh là mảng ma trận RGB 6x6x3 (3 ở đây là giá trị RGB).

A picture containing rectangle, square, colorfulness, line

Description automatically generated

Hình 2.1 Ma trận RGB 6x6x3

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

A picture containing sketch, diagram, drawing, plan

Description automatically generated

Hình 2.2 Sơ đồ toàn bộ luồng CNN trong xử lý ảnh

### **2.1.2 Các tham số chính trong CNN**

**a. Lớp tích chập- Convoluntion Layer**

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

  Description automatically generated

Hình 2.3 Ảnh minh họa cho nhân tích chập

* Xem xét 1 ma trận 5 x 5 có giá trị pixel là 0 và 1. Ma trận bộ lọc 3 x 3 như hình bên dưới.

A picture containing screenshot, rectangle, diagram, square

Description automatically generated

Hình 2.4 Ảnh ví dụ

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

A picture containing square, screenshot, rectangle, text

Description automatically generated

* Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng Kernal khác nhau.

A picture containing text, screenshot

Description automatically generated

Hình 2.5 Kết quả

**b. Bước nhảy - Stride**

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

A picture containing text, diagram, screenshot, line

Description automatically generated

Hình 2.6 Giải thích về bước nhảy

**c. Đường viền - Padding**

* Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:
* Chèn thêm các số 0 vào 4 đường biên hình ảnh(padding)
* Cắt bớt hình ảnh tại những điểm không phù hợp với kernel

**d. Hàm phi tuyến - ReLU**

* ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là: ƒ (x) = max (0, x).
* Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.

A picture containing line, number, rectangle, diagram

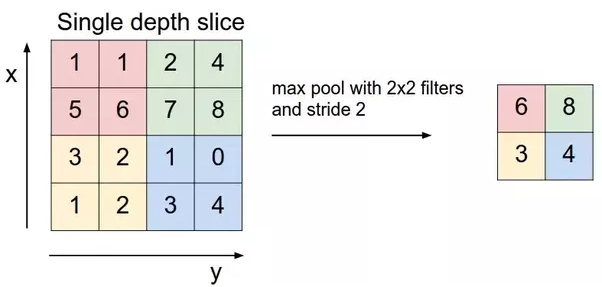
Description automatically generated

Có 1 số hà phi tuyến khác như tanh, sigmoid cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

**e. Lớp gộp - Pooling Layer**

* Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:
* Max Pooling
* Average Pooling
* Sum Pooling

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



Hình 2.6 Giải thích về pooling layer

### **2.1.3 Cấu trúc bên trong mạng CNN**

Cấu trúc mạng CNN bao gồm các lớp chính sau:

1. Lớp Input: Lớp này nhận đâu vào là ảnh đẩu vào và truyẽn nó đến các lớp tiếp theo.

2. Lớp Convolutional: Lớp này sử dụng các bộ lọc (kernel) để thực hiện các phép tích chập trên ảnh đẩu vào. Các bộ lọc này sẽ tìm kiếm các đặc trưng trong ảnh như các cạnh, đường cong, hoặc các đôi tượng khác. Kết quả cùa lớp convolutional là các feature map (bản đồ đặc trưng).

3. Lớp Activation: Lớp này thêm vào hàm kích hoạt phi tuyến tính như ReLU (Rectified Linear Unit) vào các feature map đã được tính toán. Điếu này giúp mạng CNN học được các đặc trưng phi tuyên tính và làm tăng độ chính xác của mô hình.

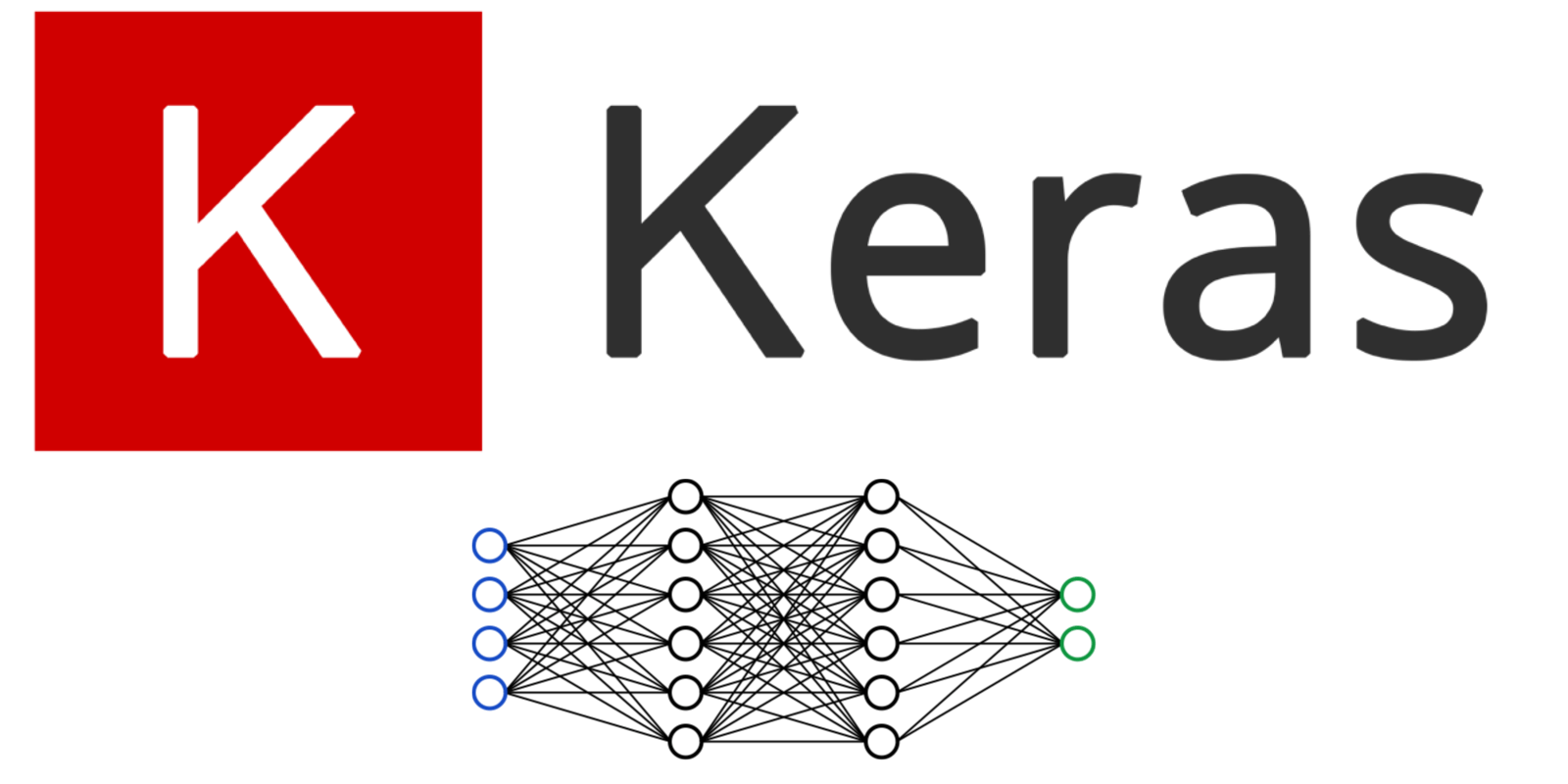
4. Lớp Pooling: Lớp này giảm kích thước của feature map băng cách thực hiện phép lấy mẩu, thường là max pooling hoặc average pooling. Điểu này giúp giảm sô' lượng tham số của mạng, tăng tốc độ tính toán và giảm khả năng overfitting.

5. Lớp Fully Connected: Lớp này kết nối các feature map đã được thu nhỏ với nhau để đưa ra dự đoán cuối cùng. Lớp này thường kết thúc băng một lớp softmax để tính toán xác suẩt của các lớp đôi tượng khác nhau.

Các lớp này thường được sử dụng trong một số kết hợp khác nhau để tạo thành một mạng CNN hoàn chỉnh. Qua quá trình huấn luyện, các trọng số trong các lớp này sẽ được tối ưu để tăng độ chính xác của mô hình trong việc phân loại các đôi tượng trong ảnh.

## **2.2 Một số loại thư viện được sử dụng**

### **2.2.1 Thư viện Keras**



Hình 2.7 Hình ảnh Keras

Keras là một thư viện mã nguôn mở cho deep learning, cung cấp các công cụ để xây dựng các mô hình neural network với tính năng cao và dễ sử dụng. Keras được phát triển bởi François Chollet và được tích hợp sấn trong thư viện Tensorflow.

Keras cung cấp cho người dùng một API đơn giản để xây dựng mô hình neural network. Với Keras, người dùng có thể dễ dàng tạo các lớp cho mô hình neural network, chỉ định số lượng lớp, số lượng nơ-ron, hàm kích hoạt và các siêu tham sô' khác để điều chỉnh mô hình.

Ngoài ra, Keras cung cấp các lớp tiêu chuẩn để xây dựng các mạng CNN, RNN và mạng nơ- ron cơ bản khác. Keras cũng hỗ trợ nhiêu kỹ thuật khác nhau để tăng tốc độ và tăng độ chính xác cùa mô hình như tôi ưu hóa, regularization và dropout.

Keras cũng được sử dụng rộng rãi trong cộng đông deep learning để xây dựng các ứng dụng phức tạp như nhận dạng hình ảnh, dịch vụ nhận diện giọng nói và các hệ thông tự động đánh giá. Do tính dễ sử dụng và hiệu quà cùa nó, Keras đã trở thành một trong những thư viện deep learning phổ biên nhất hiện nay.

Thêm vào đó, Keras có tính linh hoạt cao trong việc kết hợp với các thư viện khác của Python như NumPy, Pandas, Scikit-learn và Matplotlib để thực hiện các tác vụ phân tích dữ liệu phức tạp.

Keras cung cấp cho người dùng nhiêu kiểu mô hình khác nhau, bao gồm mạng nơ-ron đa tâng (Multi-Layer Perceptron), mạng nơ-ron tích chập (Convolutional Neural Network), mạng nơ-ron hồi quy (Recurrent Neural Network), và nhiêu kiểu mô hình khác.

Với Keras, người dùng có thể lựa chọn nhiêu kiểu tôi ưu hóa khác nhau cho mô hình, bao gôm Adam, RMSprop và SGD. Keras cũng hổtrỢ nhiêu hàm mat mát và các phương pháp regularization khác nhau để giảm overfitting.

Ngoài ra, Keras còn hỗ trợ tính năng tự động xác định kích thước đẩu vào cho mô hình và cũng cung cấp các công cụ để đánh giá hiệu suất của mô hình trên tập dữ liệu đào tạo và kiếm tra.

Keras còn hỗ trợ tính năng lưu và tải lại mô hình đã được đào tạo, cho phép người dùng lưu mô hình và tái sử dụng sau này mà không cân phải đào tạo lại từ đâu.

Điểu đáng chú ý là Keras cũng có thể được sử dụng trên nhiêu nên tảng khác nhau, bao gô'm CPU và GPU. Với sự hổtrự của GPU, Keras có thể tăng tốc độ đào tạo mô hình lên đáng kể.

Tóm lại, Keras là một thư viện deep learning tuyệt vời với nhiêu tính năng tiên tiên và dểsử dụng. Với Keras, người dùng có thể xây dựng các mô hình deep learning phức tạp chỉ trong vài dòng mã. Nó đượcsửdụng rộng rãi trong cộng đông deep learning và là một trong những công cụ phổ biến nhất cho việc xây dựng các ứng dụng AI.

### **2.2.2 Thư viện Scikit-learn (sklearn)**

**A picture containing logo, graphics, font, clipart

Description automatically generated**

Hình 2.8 Logo của Scikit-learn và Python

Scikit-learn là một trong những thư viện Python phổ biến nhất cho việc xây dựng các mô hình machine learning. Thư viện này cung cấp một loạt các công cụ và chức năng để phân tích và xử lý dữ liệu, huân luyện và kiểm tra các mô hình machine learning, và đánh giá hiệu suẩt của các mô hình.

Scikit-learn hỗ trợ nhiều loại mô hình machine learning, bao gồm phần loại, hôi quy, gom nhóm và giảm chiều dữ liệu. Nó cũng cung cáp các công cụ cho việc xử lý dữ liệu, bao gồm phân tích thành phẩn chính, chuẩn hoá dữ liệu và chuyển đổi đặc trưng.

Scikit-learn được xây dựng trên NumPy, SciPy và Matplotlib, các thư viện Python khác phổ biến cho phân tích dữ liệu và đồ họa. Do đó, Scikit-learn tương thích tốt với các thư viện này và cho phép người dùng kết hợp chúng để xây dựng các ứng dụng phân tích dữ liệu phức tạp.

Một trong những tính năng đáng chú ý của Sclkit-learn là khả năng chọn mô hình và tôi ưu hóa siêu tham sô'cho các mô hinh machine learning. Nó cung cấp các công cụ cho việc đánh giá và so sánh các mô hình khác nhau và tìm kiếm siêu tham số tối ưu cho mô hình.

Ngoài ra, Scikit-learn cũng cung cấp các tính năng cho việc chia tập dữ liệu thành các tập huấn luyện và kiểm tra, và cho phép người dùng đánh giá hiệu suất của các mô hình trên các tập dữ liệu kiểm tra khác nhau.

Scikit-learn cũng được sử dụng rộng rãi trong cộng đông machine learning để giải quyết các bài toán thực tế, bao gôm phân loại ảnh, nhận dạng văn bản, dự đoán giá cổ phiếu, và nhiêu bài toán khác.

Tóm lại, Scikit-learn là một thư viện Python râ't mạnh mẽvà đa dụng cho việc xây dựng các mô hình machine learning. Nó cung cấp các công cụ cho việc xử lý dữ liệu, chọn mô hình, tôi Ưu hóa siêu tham sô' và đánh glá hiệu suất của các mô hình.

### **2.2.3 Thư viện keras**

# CHƯƠNG III: ỨNG DỤNG

## **3.1 XÂY DỰNG MÔ HÌNH CHO TRAINING MODEL**

Việc xây dựng mô hình cho training model được thực hiện trên google colab , sau đó mô hình được lưu dưới file hdf5 và được nhúng trong Streamlit.

* 1. **Tạo ra tệp dữ liệu (datasets) cho việc training model**

Để tạo ra tệp dữ liệu, tôi thu thập hàng loạt các ảnh từ các nguồn khác nhau như: google images, pinterest, . Do số lượng loài vật trên thế giới rất nhiều nên tôi chỉ lấy 10 loài động vật tiêu biểu bao gồm: . Dữ liệu được chia ra thành 2 thư mục: Training datasets và Test datasets. Với training gồm 7178 ảnh và testing gồm 1393 ảnh.

Link training\_datasets: <https://drive.google.com/drive/folders/1HVkIDJub4y1iA4e-lp7UuyNzZQTWWPvl?usp=share_link>

Link testing\_datasets:

<https://drive.google.com/drive/folders/1HVkIDJub4y1iA4e-lp7UuyNzZQTWWPvl?usp=share_link>

* 1. **Khai báo các thư viện cần thiết**

import numpy as np

import os

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,UpSampling2D,InputLayer,Reshape

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Dropout,Activation,BatchNormalization

from tensorflow.keras.preprocessing.image import load\_img

from tensorflow.keras.preprocessing.image import img\_to\_array

from tensorflow.keras.preprocessing.image import array\_to\_img

import matplotlib.pyplot as plt

from tensorflow.keras.layers import LeakyReLU

from pathlib import Path

from skimage.io import imread

from skimage.transform import resize

from sklearn.utils import Bunch

import math

import pandas as pd

from sklearn.model\_selection import train\_test\_split

* 1. **Lấy dữ liệu đã được upload trên google drive**

num\_classes=10

img\_rows,img\_cols=128,128

batch\_size=128

train\_data\_dir="/content/drive/MyDrive/ANIMAL\_IMAGES/Training\_dataset"

test\_data\_dir= "/content/drive/MyDrive/ANIMAL\_IMAGES/Testing\_dataset"

* 1. **Thể hiện ngẫu nhiên một số ảnh có trong tệp dữ liệu**

for i in range(5):  # Display the first 5 batches of images

    # Get the next batch of images and labels

    batch = train\_generator.next()

    images = batch[0]

    # Plot the images

    fig, axs = plt.subplots(4, 8, figsize=(15, 8))

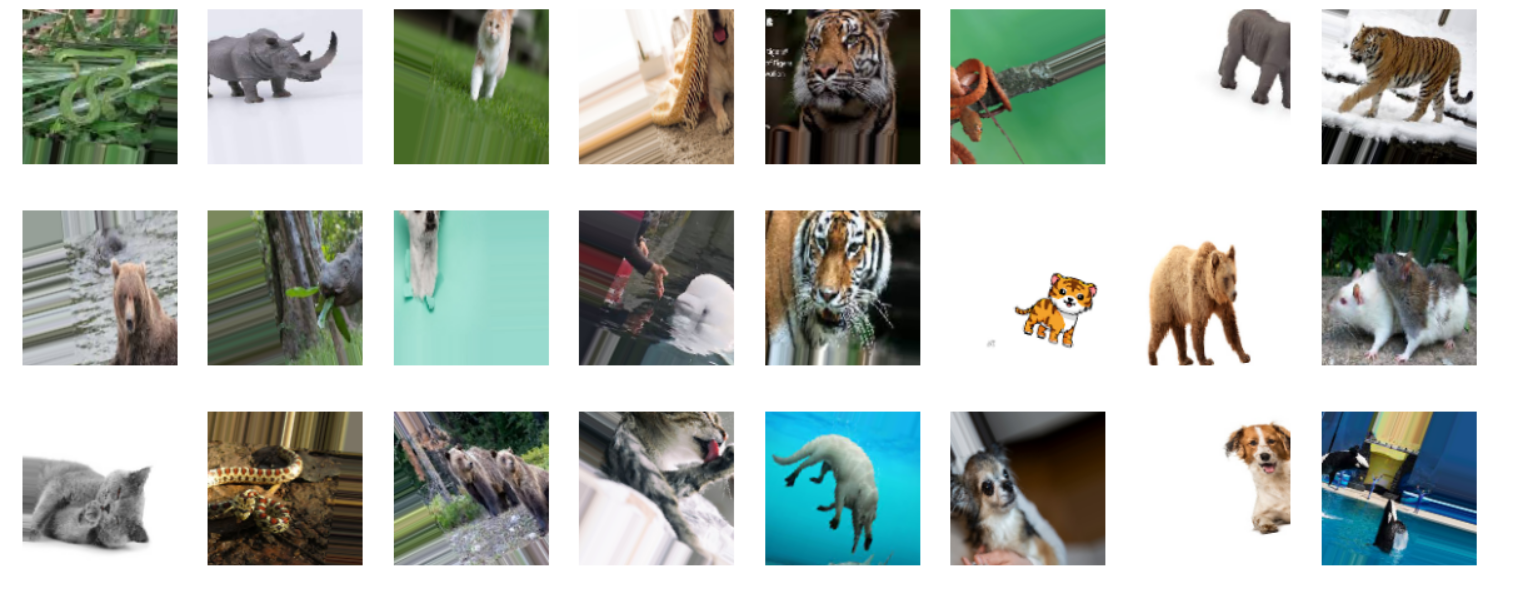
    axs = axs.ravel()

    for j in range(32):

        axs[j].imshow(images[j])

        axs[j].axis('off')

    plt.show()



* 1. **Tăng cường tệp dữ liệu**

Tăng cường dữ liệu là quá trình tạo ra thêm dữ liệu nhằm để cải thiện hiệu suất của quá trinh học máy. Bằng cách biến đổi các dữ liệu gốc ban đầu, tệp dữ liệu sẽ được mở rộng thêm.

Trong phần coding có những câu lệnh được sử dụng nhiều:

Train\_data\_dir là đường dẫn tới thư mục chứa dữ liệu huấn luyện.Thư mục này phải chứa các thư mục con đại diện cho từng lớp hoặc nhãn của dữ liệu.

target\_size=(img\_rows, img.cols) chỉ định kích thước mà các hình ảnh đâu vào sẽ được thay đồi đến. Ở đây, 'img\_rows' và 'img\_cols' là chiểu cao và chiểu rộng mong muôn của hình ảnh.

Batch\_size là kích thước của các batch dữ liệu được tạo ra.

Class\_mode= 'categorical' cho biết rang mục tiêu huấn luyện là một biến đẩu ra dạng one-hot encoded (phân loại nhiêu lớp).

shuffle=True chỉ định răng dữ liệu huân luyện se~được xáo trộn ngẩu nhiên trước mõi epoch.

'rescale=l./255': Giá trị pixel của hình ảnh sẽ được chia cho 255 để đưa vê'khoáng giá trị từ 0 đến 1. Điêu này thực hiện việc chuẩn hóa dữ liệu băng cách đảm bảo rẩng tất cả các giá trị đêu nă'm trong khoáng này.

'rotation\_range=30': Hình ẩnh sẽ được xoay một góc ngẩu nhiên trong khoáng từ-30 đến 30 độ.

'shear\_range=0.3': Hình ảnh sẽ được biến dạng băng cách kéo giãn một phẩn của nó theo một hướng ngầu nhiên trong khoảng từ -0.3 đến 0.3.

'zoom\_range=0.3': Hình ảnh sẽ được thu phóng ngẫu nhiên trong khoảng từ 0.7đến 1.3 lần kích thước ban đâu.

'width\_shift\_range=0.3': Hình ảnh sẽ được dịch chuyển ngang một phẩn của nó ngẫu nhiên trong khoảng từ -0.3 đến 0.3 lân chiếu rộng.

'height\_shift\_range=0.4': Hình ảnh sẽ được dịch chuyển dọc một phân của nó ngẫu nhiên trong khoảng từ -0.4 đến 0.4 lân chiều cao.

' horizontal\_flip=True': Hình ảnh có thể được lật ngang theo chiêu ngang. 'fill\_mode='nearest'. Khi thực hiện các biến đổi và mở rộng hình ảnh, các vị trí mới được tạo ra có thể chứa các pixel bị thiếu. Tham số này chỉ định cách điển vào các vị trí thiếu bằng cách sao chép giá trị pixel gân nhất.

train\_datagen=ImageDataGenerator(

                rescale=1./255,

                rotation\_range=30,

                shear\_range=0.3,

                zoom\_range=0.3,

                width\_shift\_range=0.3,

                height\_shift\_range=0.4,

                horizontal\_flip=True,

                fill\_mode='nearest'

)

test\_datagen=ImageDataGenerator(rescale=1./255)

train\_generator= train\_datagen.flow\_from\_directory(

    train\_data\_dir,

    target\_size=(img\_rows,img\_cols),

    batch\_size=batch\_size,

    class\_mode='categorical',

    shuffle=True

)

test\_generator=test\_datagen.flow\_from\_directory(

  test\_data\_dir,

  target\_size=(img\_rows,img\_cols),

  batch\_size=batch\_size,

  class\_mode='categorical',

  shuffle=True

)

### **Xây dựng mô hình Convolutional neural network (CNN)**

Mô hình CNN (Convolutional Neural Network) trong đoạn mã trên là một mô hình tuân tự (Sequential) được xây dựng băng cách sử dụng các lớp khác nhau như Conv2D, Activation, BatchNormalization, MaxPooling2D, Dropout, Flatten và Dense. Dưới đầy là mô tả chi tiết vể từng lớp trong mô hình:

* LớpConv2D:
  + - * Sô lượng: 32
      * Kích thước kernel: (3,3)
      * Padding: 'same’
      * Đâu vào: (img.rows, img.cols, 3)
      * Hàm kích hoạt: 'relu'
      * BatchNormalization: Cân bằng lại các trọng số và định vị trước khi áp dụng hàm kích hoạt.
      * MaxPooling2D: Phép gộp tối đa với kích thước cửa sổ (3,3).
      * Dropout: Tỳ lệ loại bỏ ngấu nhiên 25% các đơn vị đầu ra.
* Lớp Conv2D:
  + - * Sô lượng: 64
      * Kích thước kernel: (3,3)
      * Padding: 'same’
      * Hàm kích hoạt: 'relu'
      * BatchNormalization: Cân bằng lại các trọng số và định vị trước khi áp dụng hàm kích hoạt.
      * MaxPooling2D: Phép gộp tối đa với kích thước cửa sổ (2,2)
      * Dropout: Tỳ lệ loại bỏ ngấu nhiên 25% các đơn vị đầu ra.
* Lớp Conv2D:
  + - * Sô lượng: 128
      * Kích thước kernel: (3,3)
      * Padding: 'same’
      * Hàm kích hoạt: 'relu'
      * BatchNormalization: Cân bằng lại các trọng số và định vị trước khi áp dụng hàm kích hoạt.
      * MaxPooling2D: Phép gộp tối đa với kích thước cửa sổ (2,2).
      * Dropout: Tỳ lệ loại bỏ ngẫu nhiên 25% các đơn vị đầu ra.
* Lớp Flatten: Chuyển đổi đâu vào từ một tensor 2D thành một vector 1D để truyền tiếp cho các lớp Dense.
* Lớp Dense:
  + - * Số lượng đơn vị: 1024
      * Hàm kích hoạt: 'relu'
      * BatchNormalization
      * Dropout: Tỷ lệ loại bỏ ngẫu nhiên 50% các đơn vị đầu ra.
* Lớp Dense cuối cùng:
  + - Số lượng đơn vị: num.classes (sô'lượng lớp đâu ra)
    - Hàm kích hoạt: 'softmax'

model=Sequential()

#Nhân tích chập 32 lần

model.add(Conv2D(32,(3,3),padding='same',input\_shape=(img\_rows,img\_cols,3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(3,3)))

model.add(Dropout(0.25))

#Nhân tích chập 64 lần

model.add(Conv2D(64,(3,3),padding='same'))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

#Nhân tích chập 128 lần

model.add(Conv2D(128,(3,3),padding='same'))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(num\_classes))

model.add(Activation('softmax'))

print(model.summary())

Cuối cùng sử dụng hàm model.summary() nhằm tổng hợp lại mô hình mạng vừa xây dựng.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 128, 128, 32) 896

activation (Activation) (None, 128, 128, 32) 0

batch\_normalization (BatchN (None, 128, 128, 32) 128

ormalization)

max\_pooling2d (MaxPooling2D (None, 42, 42, 32) 0

)

dropout (Dropout) (None, 42, 42, 32) 0

conv2d\_1 (Conv2D) (None, 42, 42, 64) 18496

activation\_1 (Activation) (None, 42, 42, 64) 0

batch\_normalization\_1 (Batc (None, 42, 42, 64) 256

hNormalization)

max\_pooling2d\_1 (MaxPooling (None, 21, 21, 64) 0

2D)

dropout\_1 (Dropout) (None, 21, 21, 64) 0

conv2d\_2 (Conv2D) (None, 21, 21, 128) 73856

activation\_2 (Activation) (None, 21, 21, 128) 0

batch\_normalization\_2 (Batc (None, 21, 21, 128) 512

hNormalization)

max\_pooling2d\_2 (MaxPooling (None, 10, 10, 128) 0

2D)

dropout\_2 (Dropout) (None, 10, 10, 128) 0

flatten (Flatten) (None, 12800) 0

dense (Dense) (None, 1024) 13108224

activation\_3 (Activation) (None, 1024) 0

batch\_normalization\_3 (Batc (None, 1024) 4096

hNormalization)

dropout\_3 (Dropout) (None, 1024) 0

dense\_1 (Dense) (None, 10) 10250

activation\_4 (Activation) (None, 10) 0

=================================================================

Total params: 13,216,714

Trainable params: 13,214,218

Non-trainable params: 2,496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

* 1. **Huấn luyện mô hình**

Mô hình được huấn luyện với số lần học 30, ta có như sau:

/Epoch 1/100

17/57 [=======>......................] - ETA: 33:53 - loss: 3.2301 - accuracy: 0.2167/usr/local/lib/python3.10/dist-packages/PIL/Image.py:975: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images

warnings.warn(

57/57 [==============================] - 3542s 62s/step - loss: 2.5114 - accuracy: 0.2786 - val\_loss: 3.8423 - val\_accuracy: 0.1579

Epoch 2/100

57/57 [==============================] - 374s 7s/step - loss: 1.9949 - accuracy: 0.3564 - val\_loss: 4.1941 - val\_accuracy: 0.1472

Epoch 3/100

57/57 [==============================] - 352s 6s/step - loss: 1.8384 - accuracy: 0.3963 - val\_loss: 3.7223 - val\_accuracy: 0.1486

Epoch 4/100

57/57 [==============================] - 373s 7s/step - loss: 1.7818 - accuracy: 0.4061 - val\_loss: 2.8246 - val\_accuracy: 0.1788

Epoch 5/100

57/57 [==============================] - 351s 6s/step - loss: 1.6998 - accuracy: 0.4289 - val\_loss: 3.5085 - val\_accuracy: 0.1759

Epoch 6/100

57/57 [==============================] - 347s 6s/step - loss: 1.7398 - accuracy: 0.4231 - val\_loss: 3.4321 - val\_accuracy: 0.1852

Epoch 7/100

57/57 [==============================] - 352s 6s/step - loss: 1.6606 - accuracy: 0.4344 - val\_loss: 2.5609 - val\_accuracy: 0.2706

Epoch 8/100

57/57 [==============================] - 349s 6s/step - loss: 1.5788 - accuracy: 0.4634 - val\_loss: 3.8412 - val\_accuracy: 0.2304

Epoch 9/100

57/57 [==============================] - 350s 6s/step - loss: 1.5849 - accuracy: 0.4564 - val\_loss: 4.2414 - val\_accuracy: 0.3309

Epoch 10/100

57/57 [==============================] - 372s 7s/step - loss: 1.4987 - accuracy: 0.4762 - val\_loss: 4.1417 - val\_accuracy: 0.2383

Epoch 11/100

57/57 [==============================] - 360s 6s/step - loss: 1.6187 - accuracy: 0.4525 - val\_loss: 1.9762 - val\_accuracy: 0.3812

Epoch 12/100

57/57 [==============================] - 355s 6s/step - loss: 1.5151 - accuracy: 0.4776 - val\_loss: 4.0855 - val\_accuracy: 0.2836

Epoch 13/100

57/57 [==============================] - 349s 6s/step - loss: 1.4816 - accuracy: 0.4974 - val\_loss: 2.2428 - val\_accuracy: 0.3798

Epoch 14/100

57/57 [==============================] - 349s 6s/step - loss: 1.4408 - accuracy: 0.5008 - val\_loss: 3.5801 - val\_accuracy: 0.2886

Epoch 15/100

57/57 [==============================] - 350s 6s/step - loss: 1.4185 - accuracy: 0.5202 - val\_loss: 2.8872 - val\_accuracy: 0.3654

Epoch 16/100

57/57 [==============================] - 354s 6s/step - loss: 1.4304 - accuracy: 0.5046 - val\_loss: 3.2914 - val\_accuracy: 0.3812

Epoch 17/100

57/57 [==============================] - 351s 6s/step - loss: 1.4049 - accuracy: 0.5123 - val\_loss: 1.7862 - val\_accuracy: 0.4487

Epoch 18/100

57/57 [==============================] - 352s 6s/step - loss: 1.3511 - accuracy: 0.5325 - val\_loss: 1.6888 - val\_accuracy: 0.4415

Epoch 19/100

57/57 [==============================] - 355s 6s/step - loss: 1.3298 - accuracy: 0.5446 - val\_loss: 1.7146 - val\_accuracy: 0.4200

Epoch 20/100

57/57 [==============================] - 355s 6s/step - loss: 1.3450 - accuracy: 0.5339 - val\_loss: 2.0680 - val\_accuracy: 0.4049

Epoch 21/100

57/57 [==============================] - 353s 6s/step - loss: 1.3357 - accuracy: 0.5337 - val\_loss: 2.1166 - val\_accuracy: 0.4235

Epoch 22/100

57/57 [==============================] - 352s 6s/step - loss: 1.3231 - accuracy: 0.5403 - val\_loss: 1.4108 - val\_accuracy: 0.5126

Epoch 23/100

57/57 [==============================] - 353s 6s/step - loss: 1.3513 - accuracy: 0.5274 - val\_loss: 1.7835 - val\_accuracy: 0.4544

Epoch 24/100

57/57 [==============================] - 352s 6s/step - loss: 1.3052 - accuracy: 0.5410 - val\_loss: 3.0217 - val\_accuracy: 0.3331

Epoch 25/100

57/57 [==============================] - 351s 6s/step - loss: 1.3431 - accuracy: 0.5306 - val\_loss: 2.1730 - val\_accuracy: 0.4350

Epoch 26/100

57/57 [==============================] - 372s 7s/step - loss: 1.3195 - accuracy: 0.5429 - val\_loss: 2.3028 - val\_accuracy: 0.4013

Epoch 27/100

57/57 [==============================] - 353s 6s/step - loss: 1.3032 - accuracy: 0.5463 - val\_loss: 3.1904 - val\_accuracy: 0.3618

Epoch 28/100

57/57 [==============================] - 353s 6s/step - loss: 1.2717 - accuracy: 0.5521 - val\_loss: 1.7806 - val\_accuracy: 0.4652

Epoch 29/100

57/57 [==============================] - 355s 6s/step - loss: 1.2494 - accuracy: 0.5561 - val\_loss: 1.5837 - val\_accuracy: 0.4982

Epoch 30/100

57/57 [==============================] - 353s 6s/step - loss: 1.2381 - accuracy: 0.5678 - val\_loss: 1.5971 - val\_accuracy: 0.5047

Epoch 31/100

57/57 [==============================] - 355s 6s/step - loss: 1.2512 - accuracy: 0.5637 - val\_loss: 1.5270 - val\_accuracy: 0.5176

Epoch 32/100

57/57 [==============================] - 359s 6s/step - loss: 1.3061 - accuracy: 0.5490 - val\_loss: 1.4128 - val\_accuracy: 0.4788

Epoch 33/100

57/57 [==============================] - 360s 6s/step - loss: 1.2446 - accuracy: 0.5567 - val\_loss: 1.6348 - val\_accuracy: 0.4501

Epoch 34/100

57/57 [==============================] - 378s 7s/step - loss: 1.2090 - accuracy: 0.5815 - val\_loss: 1.7981 - val\_accuracy: 0.4731

Epoch 35/100

57/57 [==============================] - 355s 6s/step - loss: 1.1820 - accuracy: 0.5783 - val\_loss: 1.5834 - val\_accuracy: 0.4882

Epoch 36/100

57/57 [==============================] - 364s 6s/step - loss: 1.1945 - accuracy: 0.5773 - val\_loss: 1.6818 - val\_accuracy: 0.4264

Epoch 37/100

57/57 [==============================] - 356s 6s/step - loss: 1.1939 - accuracy: 0.5816 - val\_loss: 1.7651 - val\_accuracy: 0.4681

Epoch 38/100

57/57 [==============================] - 365s 6s/step - loss: 1.3147 - accuracy: 0.5412 - val\_loss: 1.9138 - val\_accuracy: 0.4099

Epoch 39/100

57/57 [==============================] - 357s 6s/step - loss: 1.1924 - accuracy: 0.5784 - val\_loss: 1.6749 - val\_accuracy: 0.4487

Epoch 40/100

57/57 [==============================] - 356s 6s/step - loss: 1.1313 - accuracy: 0.5918 - val\_loss: 1.7269 - val\_accuracy: 0.4530

Epoch 41/100

57/57 [==============================] - 354s 6s/step - loss: 1.1831 - accuracy: 0.5860 - val\_loss: 1.5346 - val\_accuracy: 0.5248

Epoch 42/100

57/57 [==============================] - 375s 7s/step - loss: 1.1576 - accuracy: 0.5815 - val\_loss: 1.5271 - val\_accuracy: 0.5068

Epoch 43/100

57/57 [==============================] - 350s 6s/step - loss: 1.1041 - accuracy: 0.6056 - val\_loss: 1.7946 - val\_accuracy: 0.4925

Epoch 44/100

57/57 [==============================] - 353s 6s/step - loss: 1.1148 - accuracy: 0.6030 - val\_loss: 1.7542 - val\_accuracy: 0.5047

Epoch 45/100

57/57 [==============================] - 354s 6s/step - loss: 1.1129 - accuracy: 0.6041 - val\_loss: 1.9293 - val\_accuracy: 0.4149

Epoch 46/100

57/57 [==============================] - 372s 7s/step - loss: 1.1960 - accuracy: 0.5770 - val\_loss: 2.9940 - val\_accuracy: 0.2642

Epoch 47/100

57/57 [==============================] - 355s 6s/step - loss: 1.1247 - accuracy: 0.6018 - val\_loss: 1.1917 - val\_accuracy: 0.6159

Epoch 48/100

57/57 [==============================] - 376s 7s/step - loss: 1.0943 - accuracy: 0.6023 - val\_loss: 1.3463 - val\_accuracy: 0.5808

Epoch 49/100

57/57 [==============================] - 353s 6s/step - loss: 1.0808 - accuracy: 0.6211 - val\_loss: 1.1565 - val\_accuracy: 0.6202

Epoch 50/100

57/57 [==============================] - 353s 6s/step - loss: 1.1593 - accuracy: 0.5926 - val\_loss: 1.5055 - val\_accuracy: 0.4989

Epoch 51/100

57/57 [==============================] - 351s 6s/step - loss: 1.1144 - accuracy: 0.6066 - val\_loss: 1.3061 - val\_accuracy: 0.5736

Epoch 52/100

57/57 [==============================] - 354s 6s/step - loss: 1.0882 - accuracy: 0.6110 - val\_loss: 1.9185 - val\_accuracy: 0.5269

Epoch 53/100

57/57 [==============================] - 357s 6s/step - loss: 1.0659 - accuracy: 0.6218 - val\_loss: 2.2748 - val\_accuracy: 0.4724

Epoch 54/100

57/57 [==============================] - 353s 6s/step - loss: 1.0610 - accuracy: 0.6223 - val\_loss: 1.5569 - val\_accuracy: 0.5642

Epoch 55/100

57/57 [==============================] - 354s 6s/step - loss: 1.0740 - accuracy: 0.6215 - val\_loss: 3.0157 - val\_accuracy: 0.3331

Epoch 56/100

57/57 [==============================] - 350s 6s/step - loss: 1.2119 - accuracy: 0.5818 - val\_loss: 5.5579 - val\_accuracy: 0.2290

Epoch 57/100

57/57 [==============================] - 352s 6s/step - loss: 1.2127 - accuracy: 0.5751 - val\_loss: 2.1577 - val\_accuracy: 0.4149

Epoch 58/100

57/57 [==============================] - 353s 6s/step - loss: 1.0817 - accuracy: 0.6167 - val\_loss: 1.4293 - val\_accuracy: 0.5312

Epoch 59/100

57/57 [==============================] - 355s 6s/step - loss: 1.0687 - accuracy: 0.6123 - val\_loss: 1.3586 - val\_accuracy: 0.5592

Epoch 60/100

57/57 [==============================] - 353s 6s/step - loss: 1.0427 - accuracy: 0.6323 - val\_loss: 1.2916 - val\_accuracy: 0.5887

Xây dựng đồ thị biểu thị loss và accuracy của mô hình:

def plot\_loss\_curves(history):

  """

  Returns separate loss curves for training and validation metrics.

  """

  loss = history.history['loss']

  val\_loss = history.history['val\_loss']

  accuracy = history.history['accuracy']

  val\_accuracy = history.history['val\_accuracy']

  epochs = range(len(history.history['loss']))

  # Plot loss

  plt.plot(epochs, loss, label='training\_loss')

  plt.plot(epochs, val\_loss, label='val\_loss')

  plt.title('Loss')

  plt.xlabel('Epochs')

  plt.legend()

  # Plot accuracy

  plt.figure()

  plt.plot(epochs, accuracy, label='training\_accuracy')

  plt.plot(epochs, val\_accuracy, label='val\_accuracy')

  plt.title('Accuracy')

  plt.xlabel('Epochs')

  plt.legend();

plot\_loss\_curves(history)

A graph of loss and loss

Description automatically generated with low confidenceA picture containing text, plot, line, diagram

Description automatically generated

Hình 3.1 Đồ thị thể hiện loss và accuracy của mô hình

* 1. **Xây dựng web-app**

Streamlit là thư viện mã nguồn mở của thư viện Python giúp chúng ta tạo ra ứng dụng web cho khoa học dữ liệu và các dự án máy học một cách đơn giản, cho phép xây dựng các ứng dụng web tương tác nhanh chóng bằng python.Với streamlit, ta có thể tạo ra các ứng dụng tương tác trực tiếp với python mà không cần phải sử dụng một số ngôn ngữ như HTML hoặc CSS. Một số đăc điểm chính của streamlit:

* Đơn giản và dễ sử dụng
* Tương tác thời gian thực
* Xử lí và thể hiện
* Triển khai dễ dàng

Với đề tài trên, sau khi mô hình được huấn luyện và lưu dưới dạng file có đuôi là hdf5, webapp sẽ được xây dựng tại VisualStudio Code và được deploy thông qua streamlit.

# CHƯƠNG 4: KẾT LUẬN

## **4.1 Mô hình chẩn đoán**

### **4.1.1 Mô hình nhận diện động vật**

Mô hình nhận diện động vật với độ chính xác khoảng 60%, tuy nhiên việc nhận diện vẫn chưa được chính xác tuyệt đối, đặc biệt với 2 loài động vật là “chó” và “mèo” bởi nhiều điểm tương đồng của 2 động vật làm cho mô hình nhầm lẫn trong sự nhận dạng giữa 2 loài này.

### **4.1.2 Webapp được xây dựng trên Streamlit**

Giao diện web-app trên Streamlit:

A screenshot of a computer

Description automatically generated

Hình 4.1 Giao diện của web-app

Một số hình ảnh về nhận diện con vật trên web-app:

A rhinoceros with horns

Description automatically generated with low confidence

Hình 4.2 Nhận diện tê giác trên web-app

A bear standing on a white background

Description automatically generated with low confidence

Hình 4.3 Nhận diện gấu trên web-app

A picture containing mammal, text, rodent, rat

Description automatically generated

Hình 4.4 Nhận diện chuột trên web-app

## **4.2 Ưu và nhược điểm**

Ưu điểm: tốc độ dự đoán nhanh, cùng với những ảnh rõ rang, không có nhiều yếu tố ngoại cảnh thì mô hình dự đoán một cách dễ dàng

Nhược điểm: với những động vật nó nhiều đặc điểm giống nhau như giữa mèo và chó thì mô hình bị nhầm lẫn còn nhiều, tỉ lệ dự đoán chính xác vẫn còn ở mức trung bình với những ảnh chứa nhiều ngoại vật

## **4.3 Hướng phát triển**

Trong tương lai, nếu bộ dữ liệu đủ lớn để có thể bao quát được những loài động vật phổ biến cũng như quý hiếm trên thế giới thì đây là là một công cụ tuyêt vời nhằm bảo tồn, bảo vệ và giữ gìn sự đa dạng của các loài động vật trên thế giới.

# TÀI LIỆU THAM KHẢO

[1] Mô hình CNN là gì? Link: <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>

[2] Các đặc điểm trên mô hình CNN. Link: <https://vietnix.vn/cnn-la-gi/>

[3] Thu thập hình ảnh. Link: <https://www.google.com/imghp?hl=vi&tab=wi>