| LẬP TRÌNH ÂM THANH  Spring 2023- D20PT | | | | | |
| --- | --- | --- | --- | --- | --- |
| Nhóm Bài tập lớn: SNDPROGSP23PRJ09 | | | Bài: Final Project | | |
| Họ và tên: Nguyễn Quốc Đạt (NT) | Mã sinh viên: B20DCPT053 | | | Nhóm lớp: 01 | 10 |
| Họ và tên : Nguyễn Hồng Nhung | Mã sinh viên: B20DCPT150 | | | Nhóm lớp: 01 | 9 |
| Họ và tên : Nguyễn Văn Tuấn | Mã sinh viên: B20DCPT182 | | | Nhóm lớp: 01 | 9.5 |
| Mức độ hoàn thành: 88% | | | | | |
| Đánh giá cụ thể: Nhóm đã thực hiện training model để phân biệt loài chim, ở đây vì số lượng dữ liệu quá lớn, nhóm chỉ thực hiện phân loại 5 loài chim, và xây dựng một giao diện để phân loại chim. Chi tiết được trình bày trong báo cáo dưới đây. | | | | | |
| Hạng mục | | Mức độ thực hiện | | | |
| Tìm hiểu cơ bản về đề tài | | 100% | | | |
| Tổng quan đề tài | | 100% | | | |
| Clone Project | | 90% | | | |
|  | |  | | | |

1. Phân loại âm thanh:

* Phân loại âm thanh đề cập đến việc phân loại các âm thanh dựa trên các đặc điểm, phong cách và các quy ước chung. Có nhiều thể loại âm nhạc khác nhau, mỗi loại có đặc điểm và đặc trưng riêng. Một số thể loại bao gồm: Pop, Rock, Hip-hop, R&B, Country, Jazz, Classical, Electronic…
* Để có thể phân loại các thể loại âm thanh, người ta dựa trên 1 số các đặc điểm về âm thanh sau để phân loại:
* Nhịp: Nhịp nhanh hoặc chậm.
* Giai điệu: Lãng mạn, mạnh mẽ, phức tạp, êm ái, cảm động…
* Hòa hợp: Các hợp âm và hòa âm phức tạp hoặc đơn giản
* Kết cấu và hình thức: Kết cấu đơn giản verse-chorus hoặc phức tạp, có thể gồm nhiều phần và các đoạn khác nhau
* Âm sắc: Âm sắc mạnh mẽ hay truyền thống

1. Tổng quan:

* Xác định các loài chim dựa trên tiếng kêu, giọng hót và âm thanh của chúng trong bản ghi âm là một nhiệm vụ quan trọng trong giám sát động vật hoang dã mà việc chú thích sẽ tốn thời gian nếu thực hiện thủ công. Với sự xuất hiện của các mạng thần kinh tích chập (CNN, ConvNets), việc xử lý tự động các bản ghi trường đã tạo ra một bước tiến vượt bậc. Tuy nhiên, việc xử lý các bộ dữ liệu lớn chứa hàng trăm lớp khác nhau vẫn còn rất nhiều thách thức. Trong những năm qua, nhiều kiến trúc CNN đột phá đã phát triển từ các chiến dịch đánh giá như TREC, CLEF hoặc ILSVRC. Thích nghi với các kiến trúc cho mục đích phát hiện sự kiện âm thanh đã trở thành một thông lệ phổ biến mặc dù các lĩnh vực đầu vào hình ảnh và âm thanh rất khác nhau. Tạo các tính năng sâu sắc dựa trên hình ảnh biểu diễn của bản ghi âm đã được chứng minh là rất hiệu quả.hữu ích khi áp dụng để phân loại các sự kiện âm thanh như tiếng chim.
* Phân loại tiếng chim là quá trình nhận dạng và phân loại các âm thanh của những loài chim. Chủ đề này thuộc lĩnh vực xử lý âm thanh và trí tuệ nhân tạo, ứng dụng nhiều trong các nghiên cứu về động vật, sinh thái học và giám sát môi trường.
* Việc phân loại âm thanh của loài chim đòi hỏi phải có hiểu biết về các loài chim và có khả năng phân biệt âm thanh.

1. Mục tiêu:

Các tiếng kêu của chim có đặc điểm riêng biệt và đa dạng giữa các loài, việc phân loại nhằm xác định loài chim thông qua tiếng chim của chúng.

1. Các thư viện sử dụng:

* Librosa: Thư viện Python phổ biến trong lĩnh vực xử lý âm thanh và xử lý âm nhạc. Nó cung cấp các công vụ và chức năng mạnh mẽ để đọc, xử lý và trích xuất đặc trưng từ tệp âm thanh. Libro hỗ trợ các tính toán tâm thanh như trích xuất mel spectrogram, chroma feature, beat detection…
* Keras: Là một thư viện học sâu mã nguồn mở được viết bằng Python. Nó cung cấp 1 giao diện dễ sử dụng và linh hoạt để xây dựng các mô hình mạng neural network. Keras hỗ trợ việc triển khai các mô hình học sâu trên nền tảng Tensorflow và Theano. Nó cung cấp các lớp, các thuật toán tối ưu hóa, và các hàm mất mát phổ biến để xây dựng và huấn luyện mạng neural network.
* EfficientNet: Một kiến trúc mạng neural network hiệu quả và tiết kiệm tài nguyên. Các mô hình EfficientNet có khả năng đạt được độ chính xác cao với số lượng tham số và tài nguyên tính toán tối thiểu.
* TensorFlow: Là 1 thư viện mã nguồn mở cho học máy và học sâu, phát triển bởi Google Brain team. Nó cung cấp 1 giao diện linh hoạt để xây dựng và huấn luyện các mô hình machine learning và deep learning. TensorFlow hỗ trợ việc xử lý dữ liệu số, tính toán đại số tuyến tính và xây dựng các mạng neural network phức tạp. Nó có thể chạy trên nhiều nền tảng, gồm desktop, server, thiết bị di động và đám mây.
* Scikit-learn: Một thư viện Python phổ biến cho machine learning. Nó cung cấp một tập hợp đa dạng các thuật toán học máy và các công cụ hữu ích để tiền xử lý dữ liệu, lựa chọn và đánh giá mô hình, và thực hiện các tác vụ như phân loại, hồi quy, gom cụm và suy luận. Nó cung cấp 1 giao diện đơn giản và thân thiện cho người dùng để áp dụng các thuật toán học máy vào các tác vụ thực tế.
* Flask: Là một framework phát triển ứng dụng web Python nhẹ nhàng và linh hoạt. Nó cho phép xây dựng các ứng dụng web đơn giản đến phức tạp, từ ứng dụng web tĩnh đến web động có tích hợp cơ sở dữ liệu/ Với Flask. có thể xây dựng và triển khai mô hình nhận diện âm thanh, cung cấp giao diện người dùng tương tác và hiển thị kết quả nhận diện cho người dùng. Nó cung cấp 1 cách dễ dàng để kết hợp các thành phần của 1 hệ thống nhận diện âm thanh và tạo ra 1 ứng dụng web hoàn chỉnh để phục vụ và tương tác với người dùng.
* Noisereduce: Một thư viện được sử dụng để giảm nhiễu trong tín hiệu âm thanh. Nó cung cấp các phương pháp và công cụ để xử lý và loại bỏ nhiễu không mong muốn từ tín hiệu âm thanh, cải thiện chất lượng và độ rõ nét của tín hiệu.

1. Triển khai hệ thống:

Hệ thống phân loại tiếng chim sử dụng các kỹ thuật trí tuệ nhân tạo, đặc biệt là mạng nơ-ron tích chập (CNN) và các thuật toán về học máy. Quy trình làm việc bao gồm bốn bước chính. Đầu tiên, tiến hành trích xuất quang phổ từ tất cả các bản ghi âm. Thứ hai, thực hiện mở rộng tập huấn luyện của mình thông qua việc tăng tập dữ liệu mở rộng. Tiếp theo, tiến hành tìm kiến trúc CNN tốt nhất về số lượng lớp, số lượng mẫu và tính đa dạng của dữ liệu. Cuối cùng, đào tạo các mô hình bằng cách sử dụng tiêu thụ phần cứng và các bộ công cụ và khung mã nguồn mở.

1. Tiền xử lý dữ liệu:

* Dataset là thư viện mã nguồn mở để tải xuống và chuẩn bị dataset từ tất cả các miền. API tối giản của nó cho phép người dùng tải xuống và chuẩn bị dataset chỉ trong một dòng mã Python, với một bộ chức năng cho phép xử lý trước hiệu quả. Số lượng dataset có sẵn là vô hạn, với tất cả các dataset học máy phổ biến nhất đều có sẵn để tải xuống. Không chỉ điều này, mà dataset được chuẩn bị sẵn với nhiều tính năng dành riêng cho âm thanh giúp làm việc với dataset dễ dàng đối với các nhà nghiên cứu cũng như người thực hành.
* Trong xử lý dữ liệu, dataset là tập hợp các dữ liệu thu thập được từ các nguồn khác nhau và được tổ chức thành 1 đơn vị. Mỗi dữ liệu trong dataset thường có cấu trúc và thông tin liên quan đến 1 đối tượng hoặc sự kiện cụ thể. Dataset đóng vai trò quan trọng trong quá trình xử lý dữ liệu vì nó cung cấp nguồn thông tin cho việc phân tích, mô hình hóa và đưa ra các quyết định. Dựa trên dataset, các thuật toán máy học và các kỹ thuật xử lý dữ liệu có thể được áp dụng để tìm ra mẫu, khám phá thông tin, tạo ra các dự đoán và thực hiện các tác vụ khác.
* Các bước xử lý: bao gồm tải dữ liệu xuống, tách mẫu âm thanh thành những đoạn nhỏ ví dụ từ 5-10s, trích xuất đặc trưng từ mỗi phân đoạn MFCC, MelSpectrogram, chuẩn hóa đặc trưng để đưa về cùng phạm vi, chia tập dữ liệu thành các mục như train, valid, test.
* Cụ thể: đầu tiên là chuẩn bị dữ liệu: dữ liệu được lấy từ nhiều trang web mã nguồn mở như <https://xeno-canto.org/>, <https://www.kaggle.com/datasets>, … Cách tải dữ liệu xuống, trước tiên down json từ api:

url = = 'https://www.xeno-canto.org/api/2/recordings?query={0}&page={1}'.format(*searchTerms*.replace(' ', '%20'),page)

Sau đó đọc từng file json, sau đó tải xuống theo truy vấn từng page, từng tên con chim, tên khu vực, quốc gia, … Do lượng dữ liệu lớn, nên mỗi loài, ta chỉ tải từ 50-100 file. Các file tải xuống được chia vào thư mục ./xeno-canto-dataset/

* Tiếp sau đó là tách mẫu âm thanh: mỗi âm thanh được tách từ 5-10s, sau đó trích xuất đặc trưng, cách đặt tên theo tên file âm thanh.

size **=** {'desired': 5, # [seconds]

'minimum': 4, # [seconds]

'stride' : 0, # [seconds]

'name': 5 # [number of letters]

} # stride should not be bigger than desired length

Number of directories to check and cut: 27

Processing 0 . Parus Major ...

HBox(children=(IntProgress(value=1, bar\_style='info', max=1), HTML(value='')))

Processing 1 . Turduspilaris ...

HBox(children=(IntProgress(value=1, bar\_style='info', max=1), HTML(value='')))

* Trích xuất đặc trưng:
* Mel spectrogram là một biểu đồ tần số của một tín hiệu âm thanh. Trong đó, các tần số được hiển thị trên trục dọc, thời gian được hiển thị trên trục ngang và màu sắc thể hiện mức độ năng lượng của tần số đó tại thời điểm đó. Mel Spectrogram là một công cụ quan trọng trong các giải thuật trí tuệ nhân tạo như deep learning, neural network và mạng nơ-ron học sâu để xử lý âm thanh và giúp máy tính đạt được khả năng xử lý tín hiệu âm thanh tương đương với con người.
* Cách thực hiện:

def saveMel(signal, directory):

gc.enable()

# MK\_spectrogram modified

N\_FFT = 1024 #N\_FFT chiều dài của cửa sổ FFT

HOP\_SIZE = 1024 #HOP\_SIZE khoảng cách giữa các khung của tín hiệu âm thanh

N\_MELS = 128 #N\_MELS số lượng băng thông Mel được tạo ra

WIN\_SIZE = 1024 #WIN\_SIZE chiều dài của cửa sổ

WINDOW\_TYPE = 'hann' #WINDOW\_TYPE kiểu cửa sổ

FEATURE = 'mel' #FEATURE loại đặc trưng

FMIN = 1400 #FMIN giá trị tần số thấp nhất

fig = plt.figure(1,frameon=False)

fig.set\_size\_inches(6,6)

ax = plt.Axes(fig, [0., 0., 1., 1.])

ax.set\_axis\_off()

fig.add\_axes(ax)

S = librosa.feature.melspectrogram(y=signal, sr=sr,

n\_fft=N\_FFT,

hop\_length=HOP\_SIZE,

n\_mels=N\_MELS,

htk=True,

fmin=FMIN,

fmax=sr/2)

librosa.display.specshow(librosa.power\_to\_db(S\*\*2,ref=np.max), fmin=FMIN) #power = S\*\*2

#librosa.display.spechow() hiển thị Mel Spectrograms và sử dụng

#librosa.power\_to\_db() chuyển đổi đơn vị tính từ power sang dB

fig.savefig(directory) #lưu trữ dưới dạng hình ảnh

plt.ioff()

#plt.show(block=False)

fig.clf()

ax.cla()

plt.clf()

plt.close('all')

* Tách các thư mục về từng folder: tách theo tỉ lệ mẫu dữ liệu gồm 8:1:1 ứng với từng mục train(huấn luyện), valid(xác thực), test(kiểm tra). Chúng ta không thể sử dụng các chức năng được lập trình sẵn để làm điều đó, bởi vì khi đã chia từng tệp của mình thành các tệp nhỏ hơn khác (tức là một âm thanh thành sáu hình ảnh). Việc đặt các hình ảnh được tạo từ cùng một tệp mp3 có thể dẫn đến rò rỉ dữ liệu và khiến kết quả không đáng tin cậy và sai lệch.

Selected random files number:

train: 1544 / 1544 , val: 193 / 193 , test: 194 / 194

Selected random files number:

train: 98 / 98 , val: 12 / 12 , test: 13 / 13

Selected random files number:

train: 307 / 307 , val: 38 / 38 , test: 39 / 39

Selected random files number:

train: 316 / 316 , val: 39 / 39 , test: 41 / 41

Selected random files number:

train: 414 / 414 , val: 51 / 51 , test: 53 / 53

Selected random files number:

train: 1269 / 1269 , val: 158 / 158 , test: 160 / 160

Selected random files number:

train: 44 / 44 , val: 5 / 5 , test: 7 / 7

Selected random files number:

train: 286 / 286 , val: 35 / 35 , test: 37 / 37

Selected random files number:

train: 108 / 108 , val: 13 / 13 , test: 15 / 15

Selected random files number:

train: 146 / 146 , val: 18 / 18 , test: 19 / 19

Selected random files number:

train: 71 / 71 , val: 8 / 8 , test: 10 / 10

Selected random files number:

train: 311 / 311 , val: 38 / 38 , test: 40 / 40

+) Tổng file ảnh của mỗi loài:

Found 2778 files for 0Acroc.

Found 2206 files for 1Ember.

Found 1085 files for 2Parus.

Found 1320 files for 3Phyll.

Found 1782 files for 4Sylvi.

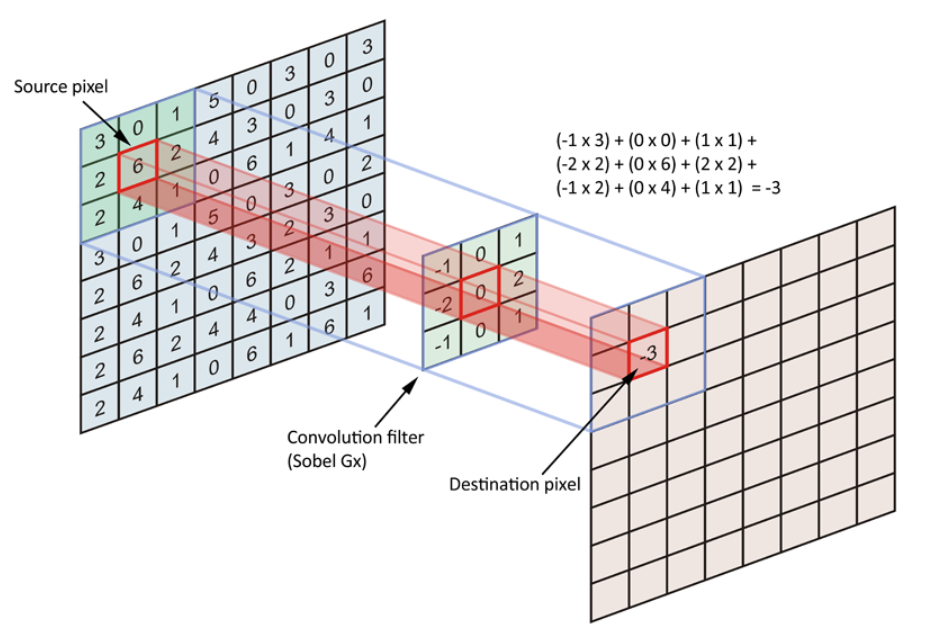
1. Mô hình hệ thống:

* Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.
* CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh, xử lý ngôn ngữ tự nhiên, phân loại âm thanh, xử lý video, xử lý tín hiệu, dữ liệu khoa học, hệ thống tự động lái xe, trong y học….
* **Quy trình thực hiện của CNN:**
* *Khởi tạo CNN*

classifier **=** Sequential()

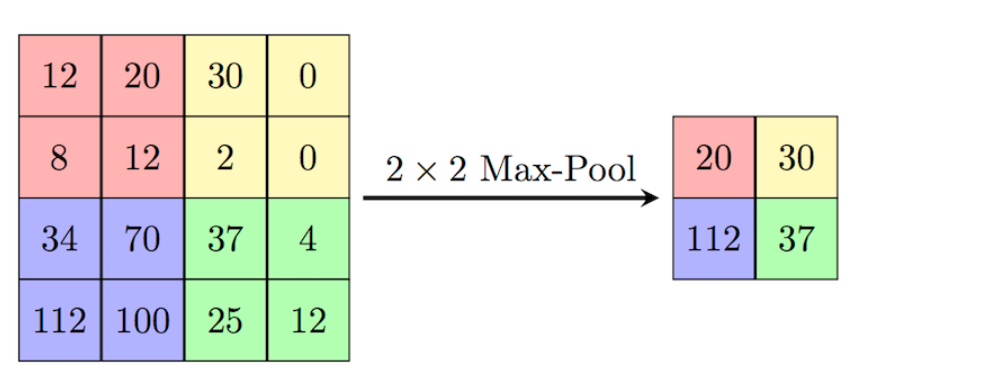
* *Nhân tích chập*

classifier**.**add(Conv2D(32, (3, 3), input\_shape **=** (64, 64, 3), activation **=** 'relu'))



* *Gộp và lấy giá trị lớn nhất ở các góc*

classifier**.**add(MaxPooling2D(pool\_size **=** (2, 2)))



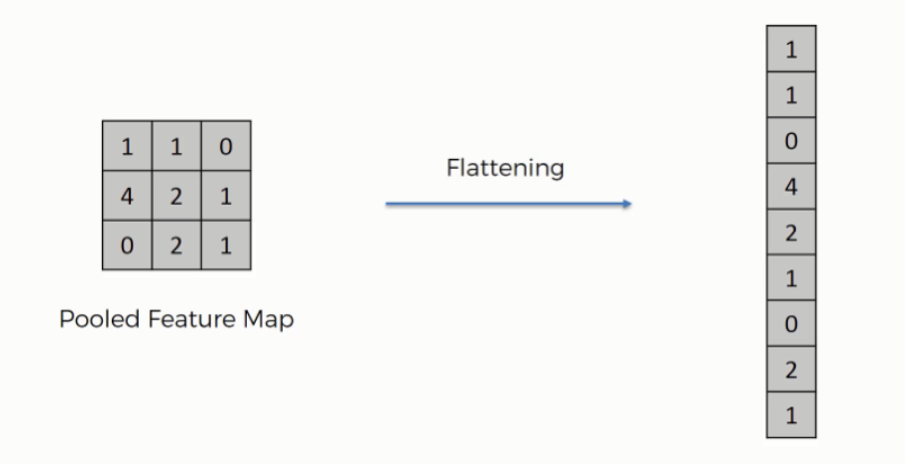
* *Thêm 1 lớp tích chập thứ 2*

classifier**.**add(Conv2D(32, (3, 3), activation **=** 'relu'))

classifier**.**add(MaxPooling2D(pool\_size **=** (2, 2)))

* *Làm dẹt ra*

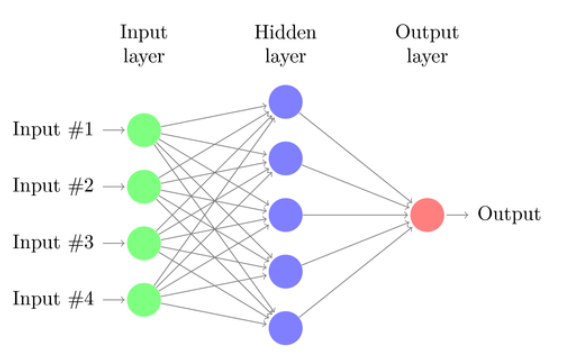
classifier**.**add(Flatten())



* *Kết nối đầy đủ*

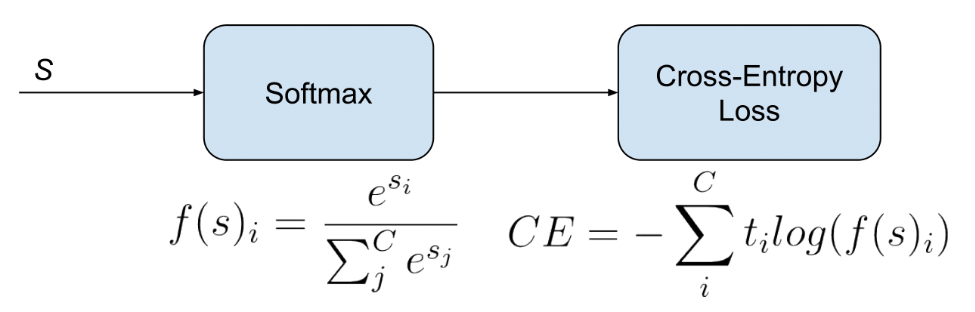
classifier**.**add(Dense(units **=** 128, activation **=** 'relu'))

classifier**.**add(Dense(units **=** 5, activation **=** 'sigmoid'))



* *Tổng hợp CNN*

classifier**.**compile(optimizer **=** 'adam', loss **=** 'categorical\_crossentropy', metrics **=** ['accuracy'])



* *Lắp CNN vào hình ảnh*

**from** keras.preprocessing.image **import** ImageDataGenerator

* train\_datagen **=** ImageDataGenerator(rescale **=** 1.**/**255,

shear\_range **=** 0.2,

zoom\_range **=** 0.2,

horizontal\_flip **=** **False**)

* test\_datagen **=** ImageDataGenerator(rescale **=** 1.**/**255)
* training\_set **=** train\_datagen**.**flow\_from\_directory('data/5\_class/train',

target\_size **=** (64, 64),

batch\_size **=** 32,

class\_mode **=** 'categorical', shuffle**=True**)

* test\_set **=** test\_datagen**.**flow\_from\_directory('data/5\_class/val',

target\_size **=** (64, 64),

batch\_size **=** 32,

class\_mode **=** 'categorical')

* classifier**.**fit\_generator(training\_set,

epochs **=** 25,

validation\_data **=** test\_set)

* **EfficientNet:**
* EfficientNet là một mạng nơ-ron tích chập (CNN) xuất hiện vào năm 2019. Mạng EfficientNet được thiết kế để đạt được hiệu suất cao trong việc phân loại hình ảnh với mức độ phức tạp khác nhau và tiết kiệm tài nguyên tính toán.
* Đặc điểm nổi bật của EfficientNet là khả năng tự động điều chỉnh độ sâu (depth), độ rộng (width) và độ phân giải (resolution) của mạng, dựa trên một siêu tham số gọi là "hệ số mở rộng" (compound scaling). Qua quá trình tăng hoặc giảm hệ số mở rộng, EfficientNet có thể thích ứng với kích thước và độ phức tạp của tập dữ liệu huấn luyện.
* Việc sử dụng EfficientNet giúp tối ưu hóa việc huấn luyện mô hình CNN trên các tài nguyên tính toán có hạn, như bộ nhớ và thời gian tính toán. Mạng EfficientNet đã đạt được những kết quả ấn tượng trên nhiều bài toán phân loại hình ảnh và đã trở thành một trong những kiến trúc CNN phổ biến và hiệu quả.
* Cách sử dụng EfficientNet:
* Tải và cài đặt thư viện keras\_efficientnet: Đầu tiên, bạn cần tải và cài đặt thư viện keras\_efficientnet, một gói mở rộng của Keras cho EfficientNet. Bạn có thể cài đặt thư viện này bằng cách sử dụng lệnh pip: pip install keras\_efficientnet.
* Xây dựng mô hình EfficientNet: Bạn có thể sử dụng lớp EfficientNet trong keras\_efficientnet để xây dựng mô hình EfficientNet. Cung cấp các thông số như kích thước đầu vào, số lớp đầu ra và cấu hình hệ số mở rộng để tạo mô hình EfficientNet phù hợp cho nhiệm vụ của bạn.
* Huấn luyện mô hình: Sau khi xây dựng mô hình, bạn có thể sử dụng dữ liệu huấn luyện để huấn luyện mô hình. Bạn cần chuẩn bị dữ liệu huấn luyện, xác định hàm mất mát (loss function) và thuật toán tối ưu hóa để huấn luyện mô hình.
* Đánh giá và sử dụng mô hình: Sau khi huấn luyện xong, bạn có thể đánh giá mô hình bằng cách sử dụng dữ liệu kiểm tra hoặc dữ liệu mới. Bạn cũng có thể sử dụng mô hình đã huấn luyện để dự đoán cho các dữ liệu mới.
* Lợi ích của EfficientNet:
* Hiệu suất cao: EfficientNet đã được thiết kế để đạt hiệu suất cao trong việc phân loại hình ảnh. Với kiến trúc linh hoạt và cấu hình hệ số mở rộng, EfficientNet có thể đáp ứng các nhiệm vụ phân loại với độ phức tạp khác nhau.
* Tối ưu tài nguyên tính toán: EfficientNet có khả năng điều chỉnh độ sâu, độ rộng và độ phân giải của mạng dựa trên hệ số mở rộng. Điều này giúp tối ưu hóa việc sử dụng tài nguyên tính toán như bộ nhớ và thời gian tính toán.
* Tích hợp dễ dàng: EfficientNet có sẵn trong các thư viện và gói mở rộng phổ biến như Keras và PyTorch. Điều này giúp dễ dàng sử dụng và tích hợp EfficientNet vào quy trình làm việc của bạn.
* Đạt kết quả tốt trên nhiều bài toán: EfficientNet đã được kiểm chứng và đạt kết quả ấn tượng trên nhiều bài toán phân loại hình ảnh, bao gồm cả các bài toán phức tạp và tập dữ liệu lớn.

⇒ Với các lợi ích trên, EfficientNet đã trở thành một lựa chọn phổ biến và hiệu quả trong lĩnh vực phân loại hình ảnh và có thể được áp dụng cho các bài toán từ nhỏ đến lớn trong nhiều ngành công nghiệp khác nhau.

* Giải thích 1 số hàm trong EfficientNet:

train\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.1,

fill\_mode='nearest')

Đoạn mã được sử dụng để tạo ra các phiên bản tăng cường dữ liệu từ tập dữ liệu huấn luyện ảnh.

preprocessing\_fuction: Hàm xử lý trước được áp dụng cho mỗi ảnh trước khi áp dụng các phép biến đổi tăng cường dữ liệu. Trong trường hợp này, preprocess\_input là 1 hàm xử lý trước được áp dụng để chuẩn hóa giá trị pixel của ảnh theo 1 phương pháp xử lý trước cụ thể (thường sử dụng cho mạng EfficientNet).

width\_shift\_range, height\_shift\_range: Khoảng chuyển đổi chiều dọc và ngang của ảnh. Giá trị trong khoảng này được áp dụng để dịch chuyển các pixel của ảnh theo 1 phạm vi ngẫu nhiên.

shear\_range: Khoảng biến đổi cắt góc của ảnh. Cắt góc của ảnh bằng cách kéo các điểm trong ảnh theo các góc khác nhau.

zoom\_range: Khoảng biến đổi thu phóng của ảnh. Giá trị trong khoảng này được sử dụng để tạo ra các phiên bản thu nhỏ hoặc phóng to của ảnh.

fill\_mode: Đây là chế độ điền vào khi thực hiện các biến đổi dịch chuyển hoặc cắt góc mà ảnh có thể bị thay đổi kích thước. Trong trường hợp này, chế độ ‘nearest’ được sử dụng để điền các pixel bị thay đổi bằng các pixel gần nhất.

train\_batches = train\_datagen.flow\_from\_directory('./5\_class/train',

classes=birds\_classes,

target\_size=IM\_SIZE,

class\_mode='categorical',shuffle=True,

batch\_size=16)

‘./5\_class/train’: Đường dẫn đến thư mục chứa tập dữ liệu huấn luyện. Các ảnh hưởng trong thư mục này sẽ được sử dụng để tạo batch dữ liệu.

‘classes=birds\_classes’: Đây là danh sách lớp mà bạn muốn xác định. Trong trường hợp này là 1 danh sách các lớp của chim

‘target\_size=IM\_SIZE’: Kích thước mà ảnh sẽ được thay đổi thành. IM\_SIZE là 1 tuple (chiều rộng, chiều cao) đại diện cho kích thước mong muốn.

class\_mode=’categorical’: Chế độ phân loại lớp cho dữ liệu huấn luyện. Trong trường hợp này, sử dụng ‘categorical’ để đảm bảo rằng nhãn của mỗi ảnh được mã hóa dưới dạng one-hot-vectors.

shuffle=True: Xáo trộn dữ liệu huấn luyện trước khi tạo batch. Điều này giúp tránh hiện tượng mô hình học theo thứ tự các lớp.

‘batch\_size=16’: Số lượng ảnh có trong mỗi batch dữ liệu.

ModelCheck = ModelCheckpoint('models/AM\_efficientnet\_5classes\_callbackb4.h5',

monitor='val\_loss',

verbose=0,

save\_best\_only=False,

save\_weights\_only=True,

mode='auto',

save\_freq=1)

‘models/AM\_efficientnet\_5classes\_callbackb4.h5’: Đường dẫn và tên tệp tin để lưu trữ trọng số của mô hình. Ở đây, nó sẽ được lưu vào tệp tin có tên ‘AM\_efficientnet\_5classes\_callbackb4.h5’ trong thư mục ‘models/’

‘monitor=’val\_loss’’: Giá trị để theo dõi trong quá trình huấn luyện. Ở đây, val\_loss được sử dụng để theo dõi giá trị hàm mất mát trên tập validation.

‘verbose=0’: Cấp độ thông báo. Giá trị 0 nghĩa là không có thông báo nào được hiển thị.

‘save\_best\_only=False’: Xác định liệu có lưu trữ trọng số chỉ khi giá trị được theo dõi là tốt nhất hay không. Trong trường hợp này, False có nghĩa là lưu trữ trọng số sau mỗi chu kỳ huấn luyện.

‘save\_weights\_only=True’: Xác định liệu chỉ lưu trữ trọng số của mô hình hay lưu trữ cả kiến trúc máy tính. Ở đây, True có nghĩa là chỉ lưu trữ trọng số.

‘mode=’auto’’: Xác định cách xác định giá trị tốt nhất. Giá trị ‘auto’ tự động xác định cách dựa trên tên của thông số theo dõi.

‘save\_freq=1’: Tần suất lưu trữ trọng số. Trong trường hợp này, trọng số để được lưu trữ sau mỗi chu kỳ huấn luyện.

net = efn.EfficientNetB4(include\_top=False,

weights='imagenet',

input\_tensor=None,

input\_shape=(224,224,3))

Một mô hình EfficientNetB4 được tạo ra bằng cách sử dụng hàm ‘efn.EfficientNetB4’ từ thư viện ‘efficientnet.keras’

‘include\_top=False’: Tham số này xác định liệu mô hình có bao gồm các tầng fully connected layer ở đầu cuối của mạng hay không. Ở đây, ‘False’ có nghĩa là chỉ tạo ra một phiên bản của EfficientNetB4 mà không bao gồm các tầng fully connected layer.

‘weights=’imagenet’’: Xác định trọng số mô hình được sử dụng. Giá trị ‘imagenet’ có nghĩa là sử dụng trọng số được huấn luyện trước trên tập dữ liệu ImageNet.

‘input\_tensor=None’: Đây là tensor đầu vào của mô hình. Trong trường hợp này, ‘None’ có nghĩa là mô hình sẽ tự tạo ra tensor đầu vào.

‘input\_shape=(224,224,3)’: Xác định kích thước đầu vào của mô hình. Trong trường hợp này, ‘input\_shape’ được đặt ra (224,224,3), đại diện cho chiều rộng, chiều cao và số kênh(RGB) của ảnh đầu vào.

class\_weights = class\_weight.compute\_class\_weight(

class\_weight='balanced',

classes=np.unique(train\_batches.classes),

y=train\_batches.classes)

Đoạn mã sử dụng hàm ‘compute\_class\_weight’ từ module ‘class\_weight’ trong thư viện scikit-learn để tính toán trọng số lớp cân bằng cho mô hình huấn luyện. Trọng số lớp cân bằng để điều chỉnh sự ảnh hưởng của các lớp trong quá trình huấn luyện, đặc biệt là khi các lớp có số lượng mẫu không cân bằng.

‘class\_weight=’balanced’’: Xác định cách tính toán trọng số lớp. Trong trường hợp này, giá trị ‘balanced’ có nghĩa là tính toán trọng số sao cho tỷ lệ mẫu của các lớp được cân bằng.

‘classes=np.unique(train\_batches.classes)’: Trả về danh sách các lớp duy nhất trong tập dữ liệu huấn luyện.

‘y=train\_batches.classes’: Nhãn của các mẫu trong tập dữ liệu huấn luyện. Đây là thông tin để tính toán trọng số lớp dựa trên phân phối của các lớp trong tập dữ liệu.

net\_final.fit(train\_batches,

validation\_data = valid\_batches,

epochs = 20,

steps\_per\_epoch= 400,

class\_weight=class\_weights\_dic,

callbacks=[ModelCheck,ReduceLR])

Mô hình ‘net\_final’ được huấn luyện bằng phương thức ‘fit’ trên các dữ liệu huấn luyện và dữ liệu validation.

‘train\_batches’: Đây là luồng dữ liệu chứa các batch dữ liệu huấn luyện.

‘validation\_data=valid\_batches’: Đây là luồng dữ liệu chứa các batch dữ liệu validation.

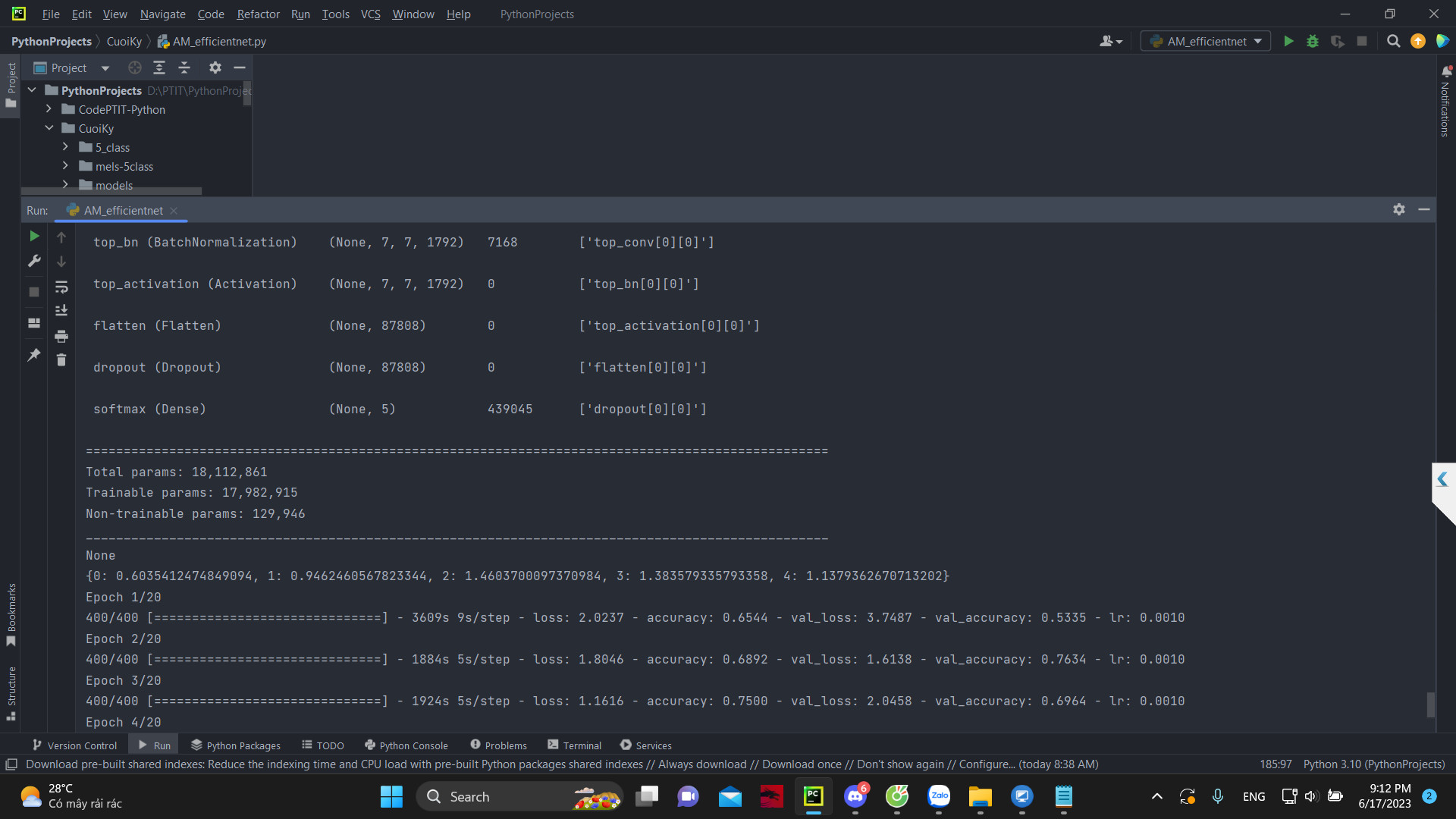
‘epochs=20’: Số lượng epoch (chu kỳ huấn luyện) mà mô hình sẽ được huấn luyện.

‘steps\_per\_epoch=400’: Số lượng bước (steps) huấn luyện trong mỗi epoch. Mỗi bước tương ứng với 1 lần cập nhật trọng số của mô hình.

‘class\_weight=class\_weights\_dic’: Trọng số lớp được sử dụng trong quá trình huấn luyện để điều chỉnh sự ảnh hưởng của các lớp. ‘class\_weights\_dic’ là 1 từ điển chứa trọng số tương ứng với từng lớp.

‘callbacks=[ModelCheck,ReduceLR]’: Các callback được sử dụng trong quá trình huấn luyện. Ở đây, ‘ModelCheck’ là callback để lưu trữ trọng số của mô hình khi có tiêu chí được đáp ứng, và ‘ReduceLR’ là callback để giảm learning rate trong quá trình huấn luyện.

* Kết quả sau training:



Epoch 1/20

400/400 [==============================] - 3609s 9s/step - loss: 2.0237 - accuracy: 0.6544 - val\_loss: 3.7487 - val\_accuracy: 0.5335 - lr: 0.0010

Epoch 2/20

400/400 [==============================] - 1884s 5s/step - loss: 1.8046 - accuracy: 0.6892 - val\_loss: 1.6138 - val\_accuracy: 0.7634 - lr: 0.0010

Epoch 3/20

400/400 [==============================] - 1924s 5s/step - loss: 1.1616 - accuracy: 0.7500 - val\_loss: 2.0458 - val\_accuracy: 0.6964 - lr: 0.0010

Epoch 4/20

400/400 [==============================] - 1847s 5s/step - loss: 1.3005 - accuracy: 0.7514 - val\_loss: 2.6265 - val\_accuracy: 0.7087 - lr: 0.0010

Epoch 5/20

400/400 [==============================] - 1801s 5s/step - loss: 0.6003 - accuracy: 0.8281 - val\_loss: 0.5163 - val\_accuracy: 0.8292 - lr: 3.0000e-04

Epoch 6/20

400/400 [==============================] - 1826s 5s/step - loss: 0.4505 - accuracy: 0.8630 - val\_loss: 0.4091 - val\_accuracy: 0.8661 - lr: 3.0000e-04

Epoch 7/20

400/400 [==============================] - 1838s 5s/step - loss: 0.4306 - accuracy: 0.8660 - val\_loss: 0.4099 - val\_accuracy: 0.8705 - lr: 3.0000e-04

Epoch 8/20

400/400 [==============================] - 1866s 5s/step - loss: 0.3801 - accuracy: 0.8788 - val\_loss: 0.5172 - val\_accuracy: 0.8426 - lr: 3.0000e-04

Epoch 9/20

400/400 [==============================] - 1850s 5s/step - loss: 0.3590 - accuracy: 0.8876 - val\_loss: 0.5418 - val\_accuracy: 0.8371 - lr: 3.0000e-04

Epoch 10/20

400/400 [==============================] - 1870s 5s/step - loss: 0.3782 - accuracy: 0.8826 - val\_loss: 0.6354 - val\_accuracy: 0.8337 - lr: 3.0000e-04

Epoch 11/20

400/400 [==============================] - 1799s 4s/step - loss: 0.3290 - accuracy: 0.8938 - val\_loss: 0.5659 - val\_accuracy: 0.8382 - lr: 3.0000e-04

Epoch 12/20

400/400 [==============================] - 1790s 4s/step - loss: 0.3445 - accuracy: 0.8894 - val\_loss: 0.5944 - val\_accuracy: 0.8449 - lr: 3.0000e-04

Epoch 13/20

400/400 [==============================] - 2226s 6s/step - loss: 0.3479 - accuracy: 0.8939 - val\_loss: 0.9950 - val\_accuracy: 0.7868 - lr: 3.0000e-04

Epoch 14/20

400/400 [==============================] - 1877s 5s/step - loss: 0.3358 - accuracy: 0.9005 - val\_loss: 0.4958 - val\_accuracy: 0.8661 - lr: 3.0000e-04

Epoch 15/20

400/400 [==============================] - 1857s 5s/step - loss: 0.3439 - accuracy: 0.8968 - val\_loss: 0.5783 - val\_accuracy: 0.8326 - lr: 3.0000e-04

Epoch 16/20

400/400 [==============================] - 1841s 5s/step - loss: 0.3316 - accuracy: 0.9027 - val\_loss: 0.4200 - val\_accuracy: 0.8717 - lr: 3.0000e-04

Epoch 17/20

400/400 [==============================] - 1860s 5s/step - loss: 0.2902 - accuracy: 0.9124 - val\_loss: 0.4284 - val\_accuracy: 0.8616 - lr: 3.0000e-04

Epoch 18/20

400/400 [==============================] - 1885s 5s/step - loss: 0.3111 - accuracy: 0.9066 - val\_loss: 0.5368 - val\_accuracy: 0.8683 - lr: 3.0000e-04

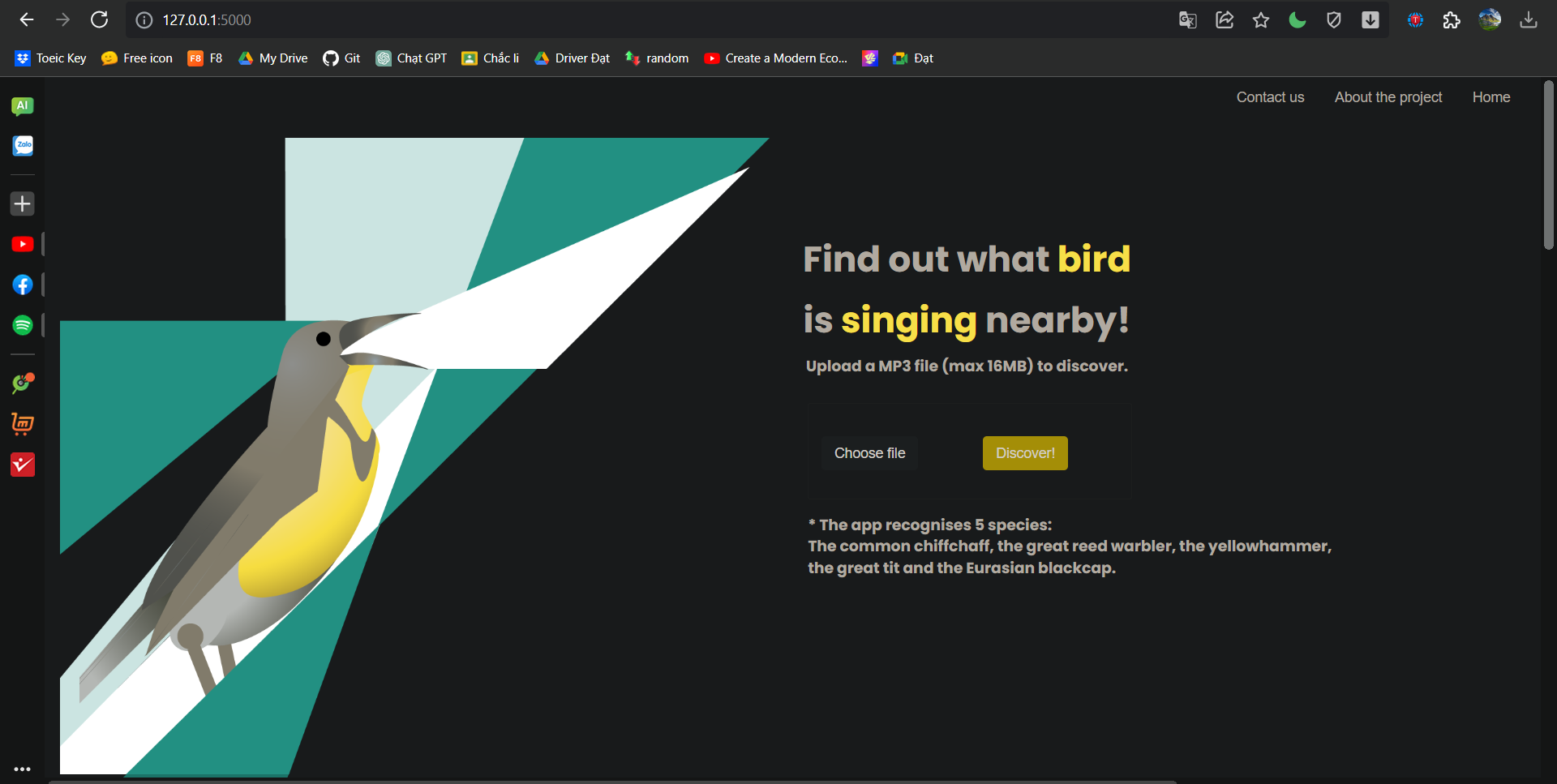
Epoch 19/20

400/400 [==============================] - 1860s 5s/step - loss: 0.2849 - accuracy: 0.9157 - val\_loss: 0.5646 - val\_accuracy: 0.8549 - lr: 3.0000e-04

Epoch 20/20

400/400 [==============================] - 1869s 5s/step - loss: 0.2667 - accuracy: 0.9155 - val\_loss: 0.4226 - val\_accuracy: 0.8672 - lr: 3.0000e-04

* Giao diện hệ thống dự đoán chim dựa vào file âm thanh:



* Chọn file âm thanh và tải lên, sau đó ấn discover, chim phát hiện sẽ được dự đoán và hiển thị kết quả.

1. Đánh giá kết quả:

Mức độ hoàn thành của dự án mới chỉ ở mức training dữ liệu thành công, phần dự đoán còn sơ xuất.

1. Link tham khảo:

<https://github.com/wimlds-trojmiasto/birds/tree/master/notebooks>

<https://github.com/m-kortas/Sound-based-bird-species-detection>

<https://ceur-ws.org/Vol-1866/paper_143.pdf>

<https://www.mathworks.com/company/newsletters/articles/compressing-neural-networks-using-network-projection.html?s_eid=psm_brj&source=15308>

<https://www.youtube.com/watch?v=Q-3VA5oCNFk&t=494s&ab_channel=NandeiTechnologies>

<https://github.com/wimlds-trojmiasto/birds/tree/master/notebooks>

<https://github.com/m-kortas/Sound-based-bird-species-detection>

<https://www.sciencedirect.com/science/article/abs/pii/S0003682X23000439>