

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐIỆN ĐIỆN TỬ
BỘ MÔN KỸ THUẬT MÁY TÍNH - VIỄN THÔNG



HCMUTE

BÁO CÁO CUỐI KÌ

PHÂN LOẠI TÍN HIỆU RADAR

GVHD: TS.Huỳnh Thế Thiện

Sinh viên: Hàn Anh Tú

MSSV: 21161215

Nguyễn Minh Nhựt

MSSV: 21161166

TP. HỒ CHÍ MINH – 4/2024

BẢNG NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Stt	Nội dung thực hiện	Nhận xét

Nhận xét tổng quát:

.....

.....

.....

.....

TÓM TẮT

Deep Learning là thuật toán dựa trên một số ý tưởng từ não bộ tới việc tiếp thu nhiều tầng biểu đạt, cả cụ thể lẫn trừu tượng, qua đó làm rõ nghĩa của các loại dữ liệu. Deep Learning được ứng dụng trong nhận diện hình ảnh, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên. Hiện nay rất nhiều các bài toán nhận dạng, phân loại sử dụng Deep Learning, vì nó có thể giải quyết các bài toán với số lượng lớn các biến, tham số kích thước đầu vào lớn với hiệu năng cũng như độ chính xác vượt trội so với các phương pháp phân lớp truyền thống, xây dựng những hệ thống thông minh với độ chính xác cao. Trong báo cáo này, chúng tôi nghiên cứu về đề tài “Phân loại tín hiệu radar” theo sự hướng dẫn của TS.Huỳnh Thế Thiện.

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU	1
1.1. GIỚI THIỆU	1
1.2. MỤC TIÊU ĐỀ TÀI	1
1.3. GIỚI HẠN ĐỀ TÀI	1
1.4. PHƯƠNG PHÁP NGHIÊN CỨU	1
1.5. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU	2
1.6. BỐ CỤC QUYỀN BẢO CÁO	2
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	3
2.1. GIỚI THIỆU MẠNG CNN	3
CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH	7
3.1. XÂY DỰNG MÔ HÌNH	7
3.2. CÁC BƯỚC THỰC HIỆN	7
3.2.1. Load tập dữ liệu từ Kaggle	7
3.2.2. Huấn luyện dữ liệu đưa vào	8
3.2.3 Dữ liệu đầu ra được phân loại theo các lớp như sau	13
CHƯƠNG 4: KẾT QUẢ THÍ NGHIỆM	16
4.1. KẾT QUẢ SAU KHI HUẤN LUYỆN	16
4.2 ĐỒ THỊ SỰ BIẾN THIÊN CỦA HÀM LOSS VÀ ACCURACY	22
4.3 ĐƯA HÌNH ẢNH TỪ TẬP TEST ĐỂ PHÂN LOẠI	23
KẾT LUẬN	25
PHỤ LỤC	26
TÀI LIỆU THAM KHẢO	28

DANH MỤC HÌNH

Hình 2.1 Kiến trúc cơ bản của mạng CNN.....	3
Hình 2.2 Bộ lọc tích chập được sử dụng trên ma trận điểm ảnh	4
Hình 2.3 Phương thức Average Pooling và Max Pooling	5
Hình 3.1 Sơ đồ quy trình của mô hình phân loại tín hiệu radar.....	7
Hình 3.2 Hình ảnh 2 file test_set và training_set.....	7
Hình 3.3 Hình ảnh các file tín hiệu được gán nhãn.....	8
Hình 3.4 Bảng phân tích mạng CNN được design trên Matlab.....	10
Hình 3.5 Chuyển đổi mô hình từ Matlab sang file code python.....	13
Hình 3.6 Bảng phân tích đầu ra phân loại tín hiệu.....	15
Hình 4.1 Tổng trọng số được ước tính trong mạng.....	17
Hình 4.2 Độ chính và giá trị hàm lỗi theo từng epoch	19
Hình 4.3 Đồ thị Training và Validation Accuracy/Loss	22
Hình 4.4 Hình ảnh trước khi nhận diện (ảnh từ file B-FM)	24
Hình 4.5 Hình ảnh sau khi phân loại (B-FM)	24

CÁC TỪ VIẾT TẮT

ReLu	Rectified Linear Unit
CNN	Convolutional Neural Network
MLL	Machine Learning Library
CONV	Convolution

CHƯƠNG 1: GIỚI THIỆU

1.1. GIỚI THIỆU

CNN là một trong những mô hình mạng Học sâu phổ biến nhất hiện nay, có khả năng nhận dạng và phân loại hình ảnh với độ chính xác rất cao, thậm chí còn tốt hơn con người trong nhiều trường hợp. Mô hình này đã và đang được phát triển, ứng dụng vào các hệ thống xử lý ảnh lớn của Facebook, Google hay Amazon... cho các mục đích khác nhau, như các thuật toán gắn thẻ tự động, tìm kiếm ảnh hoặc gợi ý sản phẩm cho người tiêu dùng.

Mạng CNN với kiến trúc thay đổi, có khả năng xây dựng liên kết chỉ sử dụng một phần cục bộ trong ảnh kết nối đến node trong lớp tiếp theo thay vì toàn bộ ảnh như trong mạng nơ ron truyền thẳng.

Nhờ sự phát triển mạng CNN, chúng tôi có cơ sở thực hiện đề tài “Nhận diện khuôn mặt”.

1.2. MỤC TIÊU ĐỀ TÀI

Đề tài “Phân loại các dạng tín hiệu radar” có thể áp dụng trong các hệ thống:

- Hệ thống thu phát sóng và tín hiệu
- Phân tích dữ liệu

1.3. GIỚI HẠN ĐỀ TÀI

Phạm vi giới hạn của đề tài:

Đề tài chủ yếu sử dụng các thư viện có sẵn trong lập trình python, đồng thời ứng dụng các mô hình toán học trong mạng nơ ron nhân tạo. Điều kiện đầu vào của mô hình là các tệp hình ảnh có giới hạn, đầu ra phân loại các dạng sóng được đưa vào có độ chính xác tương đối. Đối với chức năng thì có sự giới hạn do hiểu biết về lập trình và cách ứng dụng của mô hình vào các hệ thống lớn.

1.4. PHƯƠNG PHÁP NGHIÊN CỨU

Trong phần này, chúng tôi thiết kế mô hình phân loại tín hiệu dựa trên mạng CNN. Dựa trên các phương pháp nghiên cứu chính như phương pháp phân tích, tham khảo tài liệu, phương pháp tổng hợp tài liệu lý thuyết...

1.5. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU

Các đối tượng cần nghiên cứu và phạm vi nghiên cứu có thể giải quyết được đề tài:

- *Đối tượng nghiên cứu*: Trí tuệ nhân tạo, lập trình python.
- *Phạm vi nghiên cứu*: Mạng CNN, các module trong python: numpy, os, tensorflow, image.

1.6. BỐ CỤC QUYỀN BÁO CÁO

Nội dung báo cáo:

CHƯƠNG 1: GIỚI THIỆU

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH

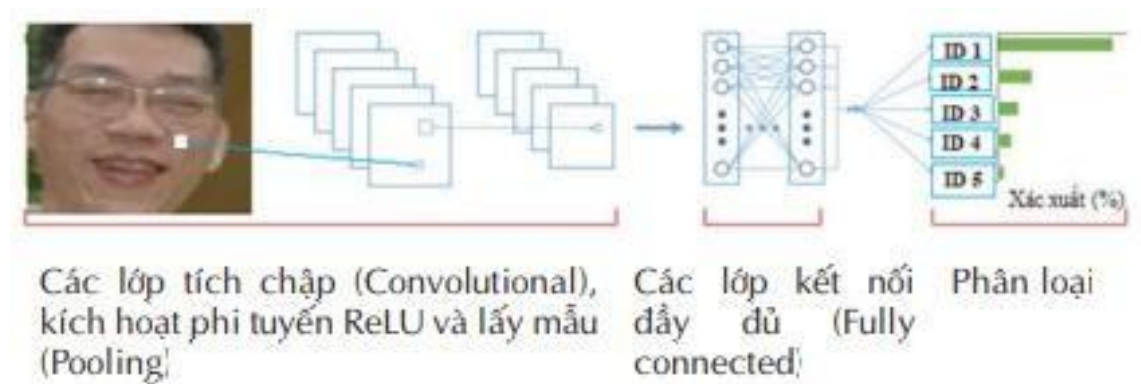
CHƯƠNG 4: KẾT QUẢ THÍ NGHIỆM

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. GIỚI THIỆU MẠNG CNN

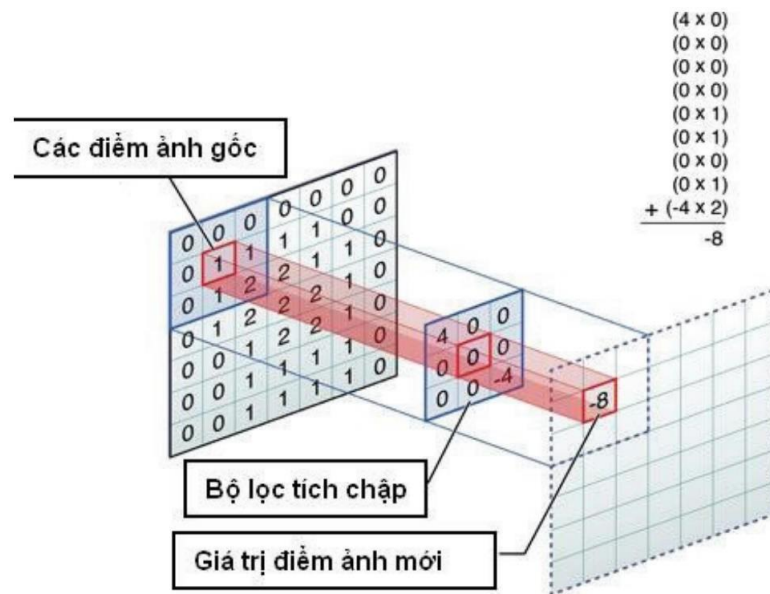
Hình 2.1 trình bày một kiến trúc mạng CNN, các lớp cơ bản trong một mạng CNN bao gồm: lớp tích chập (Convolutional); lớp kích hoạt phi tuyến ReLU (Rectified Linear Unit); lớp lấy mẫu (Pooling); lớp kết nối đầy đủ (Fully connected) được thay đổi về số lượng và cách sắp xếp để tạo ra các mô hình huấn luyện phù hợp cho từng bài toán khác nhau. Các lớp tích chập (Convolutional), kích hoạt phi tuyến ReLU và lấy mẫu (Pooling) Các lớp kết nối đầy đủ (Fully connected) Phân loại Hình 1. Kiến trúc cơ bản của một mạng CNN.

Lớp tích chập: đây là thành phần quan trọng nhất trong mạng CNN, thể hiện sự liên kết cục bộ thay vì kết nối toàn bộ các điểm ảnh. Các liên kết cục bộ được tính toán bằng phép tích chập giữa các giá trị điểm ảnh trong một vùng ảnh cục bộ với các bộ lọc filters có kích thước nhỏ.



Hình 2.1 Kiến trúc cơ bản của mạng CNN

Lớp tích chập: đây là thành phần quan trọng nhất trong mạng CNN, thể hiện sự liên kết cục bộ thay vì kết nối toàn bộ các điểm ảnh. Các liên kết cục bộ được tính toán bằng phép tích chập giữa các giá trị điểm ảnh trong một vùng ảnh cục bộ với các bộ lọc filters có kích thước nhỏ.



Hình 2.2 Bộ lọc tích chập được sử dụng trên ma trận điểm ảnh

Bộ lọc tích chập được sử dụng trên ma trận điểm ảnh. Trong hình 2.2, bộ lọc được sử dụng là một ma trận có kích thước 3x3, bộ lọc này dịch chuyển lần lượt qua từng vùng ảnh đến khi hoàn thành quét toàn bộ bức ảnh, tạo ra một bức ảnh mới có kích thước nhỏ hơn hoặc bằng với kích thước ảnh đầu vào. Kích thước này được quyết định tùy theo kích thước các khoảng trắng được thêm ở viền bức ảnh gốc và được tính theo công thức sau:

$$O = \frac{i + 2 * p - k}{s} + 1 \quad (2.1)$$

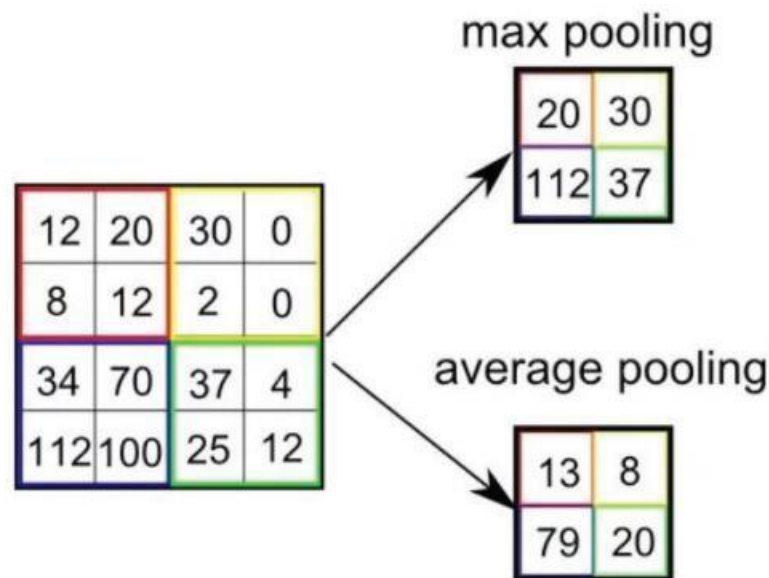
Trong đó: O: kích thước ảnh đầu ra; i: kích thước ảnh đầu vào; p: kích thước khoảng trắng phía ngoài viền của ảnh gốc; k: kích thước bộ lọc; s: bước trượt của bộ lọc. Như vậy, sau khi đưa một bức ảnh đầu vào cho lớp tích chập nhận được kết quả đầu ra là một loạt ảnh tương ứng với các bộ lọc đã được sử dụng để thực hiện phép tích chập. Các trọng số của các bộ lọc này được khởi tạo ngẫu nhiên trong lần đầu tiên và sẽ được cập nhật trong quá trình huấn luyện. - Lớp kích hoạt phi tuyến ReLU: được xây dựng để đảm bảo tính phi tuyến của mô hình huấn luyện sau khi đã thực hiện một loạt các phép tính toán tuyến tính qua các lớp tích chập. Lớp kích hoạt phi tuyến sử dụng các hàm kích hoạt phi tuyến như ReLU hoặc sigmoid, tanh... để giới hạn phạm vi biên độ cho phép của giá trị đầu ra. Trong số các hàm kích hoạt này, hàm ReLU được chọn do cài đặt đơn giản, tốc độ xử lý nhanh mà vẫn đảm bảo được tính toán hiệu quả. Phép tính toán của hàm ReLU chỉ

đơn giản là chuyển tất cả các giá trị âm thành giá trị 0. Lớp ReLU được áp dụng ngay phía sau lớp tích chập, với đầu ra là một ảnh mới có kích thước giống với ảnh đầu vào, các giá trị điểm ảnh cũng hoàn toàn tương tự, trừ các giá trị âm đã bị loại bỏ.

$$f(x) = \max(0, x) \quad (2.2)$$

Lớp lấy mẫu: được đặt sau lớp tích chập và lớp ReLU để làm giảm kích thước ảnh đầu ra trong khi vẫn giữ được các thông tin quan trọng của ảnh đầu vào. Việc giảm kích thước dữ liệu có tác dụng làm giảm được số lượng tham số cũng như tăng hiệu quả tính toán. Lớp lấy mẫu cũng sử dụng một cửa sổ trượt để quét toàn bộ các vùng trong ảnh như lớp tích chập, và thực hiện phép lấy mẫu thay vì phép tích chập, sẽ chọn lưu lại một giá trị duy nhất đại diện cho toàn bộ thông tin của vùng ảnh đó.

Hình 3 thể hiện các phương thức lấy mẫu thường được sử dụng nhất hiện nay, đó là Max Pooling (lấy giá trị điểm ảnh lớn nhất) và Average Pooling (lấy giá trị trung bình của các điểm ảnh trong vùng ảnh cục bộ).



Hình 2.3 Phương thức Average Pooling và Max Pooling

Như vậy, với mỗi ảnh đầu vào được đưa qua lấy mẫu sẽ thu được một ảnh đầu ra tương ứng, có kích thước giảm xuống đáng kể nhưng vẫn giữ được các đặc trưng cần thiết cho quá trình tính toán và nhận dạng.

Lớp kết nối đầy đủ: được thiết kế tương tự như trong mạng nơ ron truyền thống, tất cả các điểm ảnh được kết nối đầy đủ với node trong lớp tiếp theo.

So với mạng nơ ron truyền thống, các ảnh đầu vào của lớp này đã có kích thước được giảm bớt rất nhiều, đồng thời vẫn đảm bảo các thông tin quan trọng của ảnh cho việc nhận dạng. Do vậy, việc tính toán nhận dạng sử dụng mô hình truyền thẳng đã không còn phức tạp và tốn nhiều thời gian như trong mạng nơ ron truyền thống.

CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH

3.1. XÂY DỰNG MÔ HÌNH

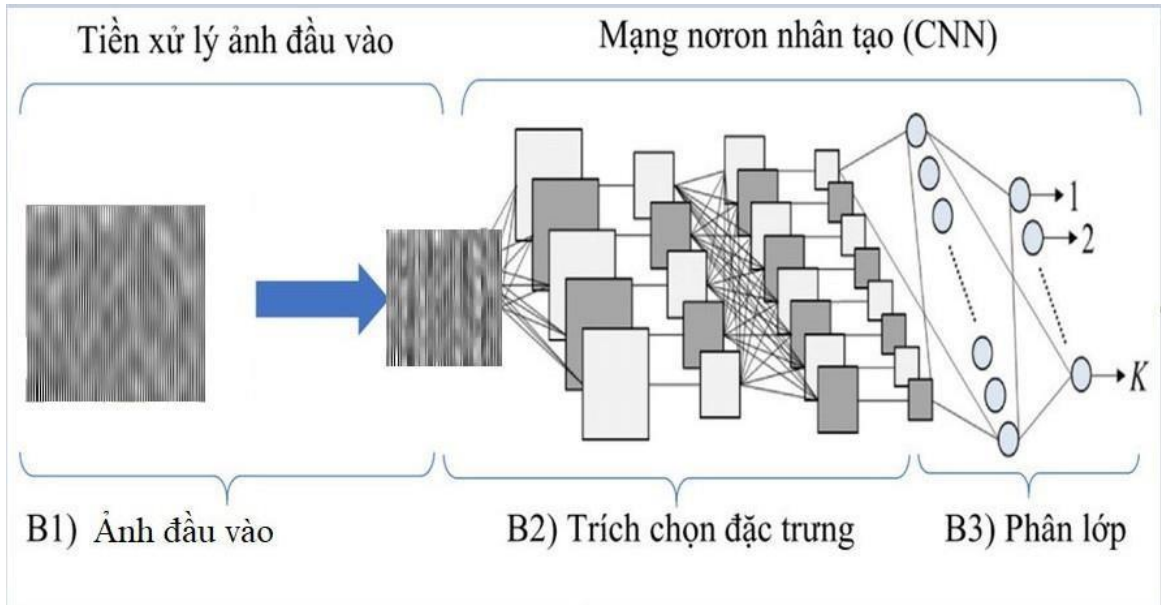
Mô tả các bước xây dựng mạng

Mô hình nhận dạng được chia thành 3 bước chính (Hình 3.1), bao gồm:

Bước 1: Load tập dữ liệu từ Kaggle .

Bước 2: Huấn luyện dữ liệu đưa vào và trích chọn các đặc trưng.

Bước 3: Phân loại ảnh dựa trên đặc trưng được trích chọn và đưa ra kết quả.

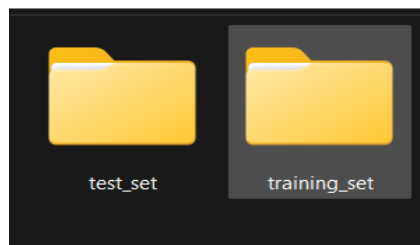


Hình 3.1 Sơ đồ quy trình của mô hình phân loại tín hiệu radar

3.2. CÁC BƯỚC THỰC HIỆN

3.2.1. Load tập dữ liệu từ Kaggle

- Gồm 2 File : train_set và test_set được tải từ Kaggle



Hình 3.2 Hình ảnh 2 file test_set và training_set

- Trong mỗi 1 file là train_set hay test_set đều chứa 8 file là các hình ảnh từ tín hiệu radar đã được gán nhãn



Hình 3.3 Hình ảnh các file tín hiệu được gán nhãn

- Load dữ liệu trên file code

```
import pathlib
training_dir = pathlib.Path('C:/MyLaptop/Spyder File Code/finalproject/training_set')
training_count = len(list(training_dir.glob('*/*.png')))
print(training_count)

test_dir = pathlib.Path('C:/MyLaptop/Spyder File Code/finalproject/test_set')
test_count = len(list(test_dir.glob('*/*.png')))
print(test_count)

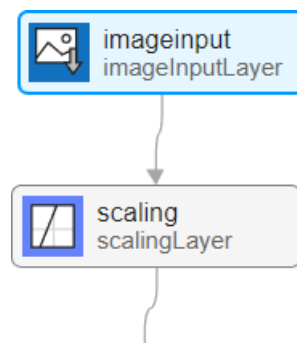
In [1]: runfile('C:/MyLaptop/Spyder File Code/finalproject/finalproject.py', wdir='C:/MyLaptop/
Spyder File Code/finalproject')
6400
4800
Found 6400 files belonging to 8 classes.
Found 4800 files belonging to 8 classes.
['B-FM', 'Barker', 'CPFSK', 'DSB-AM', 'GFSK', 'LFM', 'Rect', 'SSB-AM']
```

- Kết quả in ra ở màn hình Console cho thấy tệp train gồm 6400 mẫu và tệp test gồm 4800 mẫu , được gán nhãn phân thành 8 loại trong hình trên .

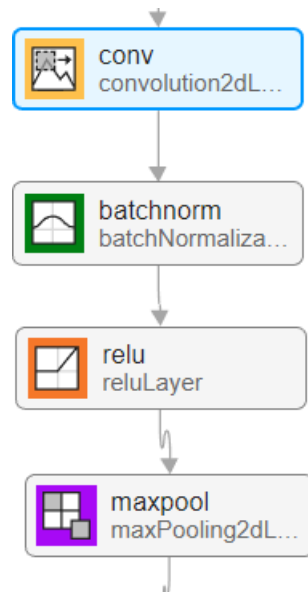
3.2.2. Huấn luyện dữ liệu đưa vào

- Mô hình mạng CNN được thiết kế bằng Matlab như sau :

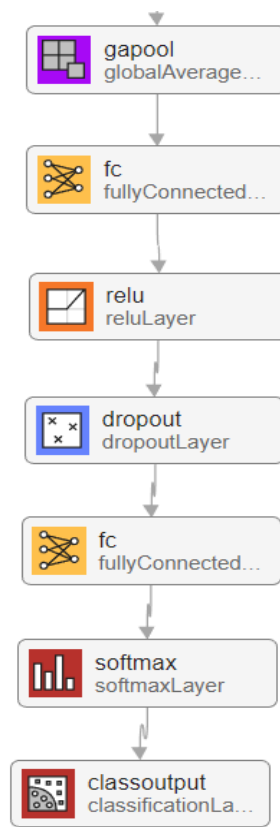
- Đây là lớp tiền xử lý dữ liệu



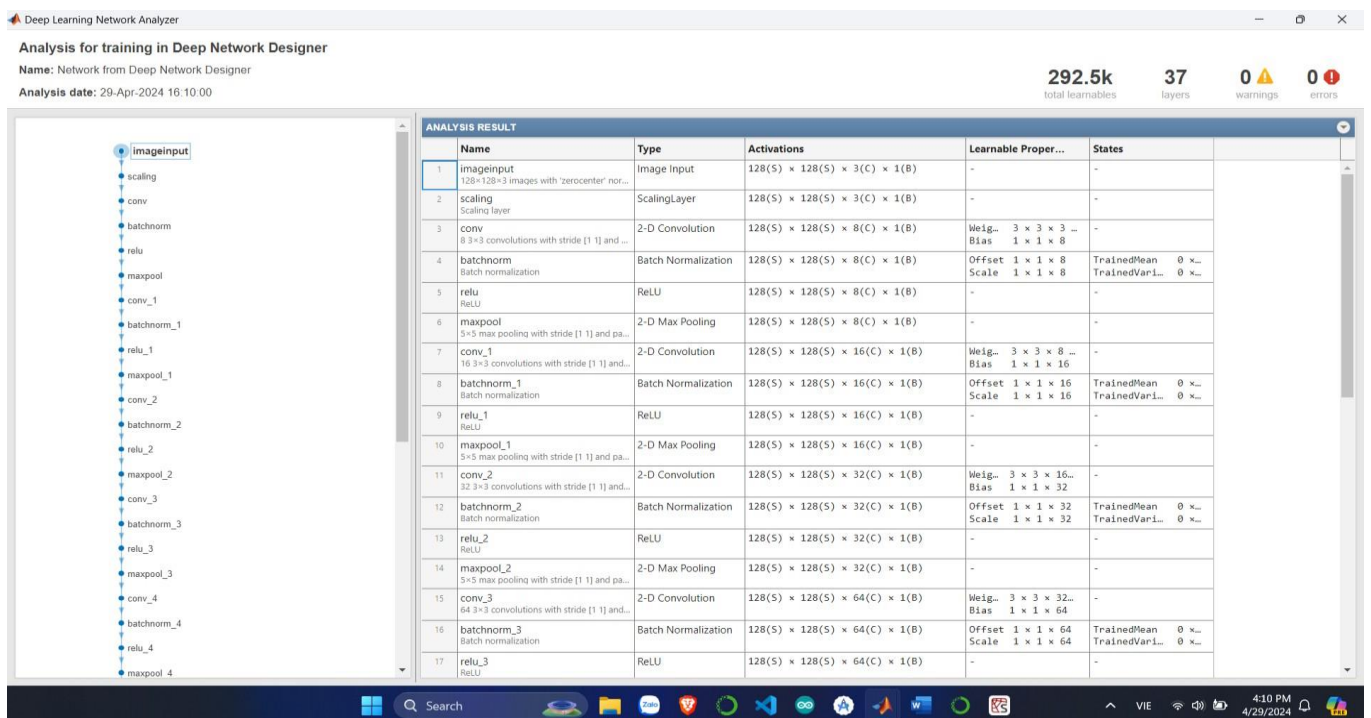
- Tiếp theo là các lớp tích chập 3x3 với số lớp liên tiếp là: 8,16,32,64,64,128,128



- Quá trình xử lý dữ liệu sau khi đã hoàn thành một quá trình xử lý chính.



- Phân tích tổng số lượng trọng số trong mạng bằng Matlab



Hình 3.4 Bảng phân tích mạng CNN được design trên Matlab

Như hình vẽ , Matlab đã thống kê được mạng CNN trên đã sử dụng tổng cộng 292.5K trọng số. Cụ thể như sau :

Lớp	Tổng trọng số
Conv	$3 \times 3 \times 3 \times 8 + 1 \times 1 \times 8 = 224$
Batchnorm	$1 \times 1 \times 8 + 1 \times 1 \times 8 = 16$
Conv1	$3 \times 3 \times 8 \times 16 + 1 \times 1 \times 16 = 1168$
Batchnorm1	$1 \times 1 \times 16 + 1 \times 1 \times 16 = 32$
Conv2	$3 \times 3 \times 16 \times 32 + 1 \times 1 \times 32 = 4640$
Batchnorm2	$1 \times 1 \times 32 + 1 \times 1 \times 32 = 64$
Conv3	$3 \times 3 \times 32 \times 64 + 1 \times 1 \times 64 = 18496$
Batchnorm3	$1 \times 1 \times 64 + 1 \times 1 \times 64 = 128$
Conv4	$3 \times 3 \times 64 \times 64 + 1 \times 1 \times 64 = 36928$
Batchnorm4	$1 \times 1 \times 64 + 1 \times 1 \times 64 = 128$
Conv5	$3 \times 3 \times 64 \times 128 + 1 \times 1 \times 128 = 73856$
Batchnorm5	$1 \times 1 \times 128 + 1 \times 1 \times 128 = 256$
Conv6	$3 \times 3 \times 128 \times 128 + 1 \times 1 \times 128 = 147584$

Batchnorm6	$1 \times 1 \times 128 + 1 \times 1 \times 128 = 256$
Fc	$64 \times 128 + 64 \times 1 = 8256$
Fc_1	$8 \times 64 + 8 \times 1 = 520$

- Phân tích các lớp trong mạng đã thiết kế

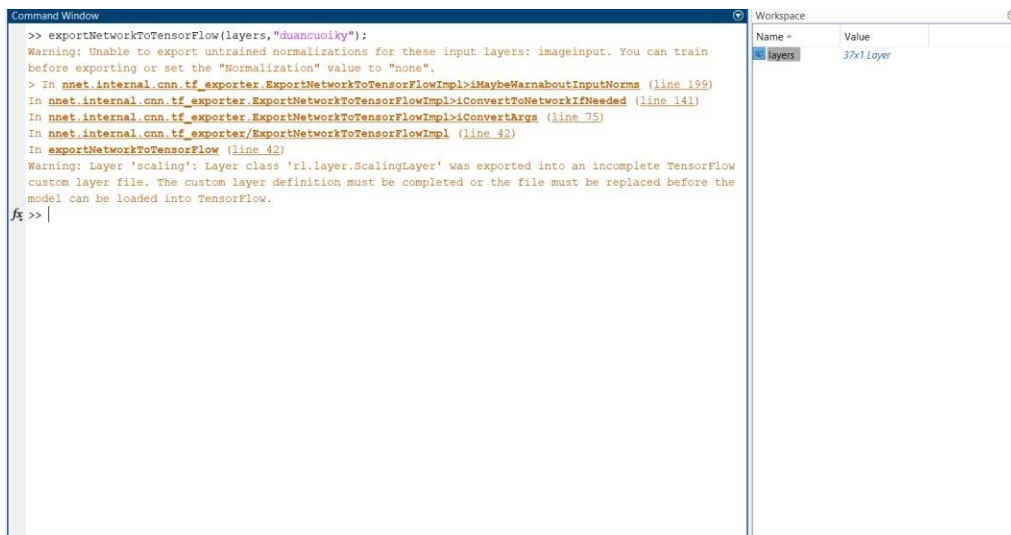
1. **Rescaling:** Lớp này điều chỉnh tỷ lệ giá trị pixel của ảnh về khoảng từ 0 đến 1 bằng cách chia mỗi giá trị pixel cho 255. Điều này giúp cải thiện hiệu suất của mô hình bằng cách làm cho dữ liệu đầu vào có dạng phân phối thích hợp hơn.
2. **Conv2D:** Lớp convolution 2D thực hiện phép tích chập trên đầu vào, sử dụng một tập hợp các bộ lọc (kernels) để trích xuất các đặc trưng từ ảnh đầu vào. Trong mạng này, lớp Conv2D được sử dụng với các kernel có kích thước (3, 3) và hàm kích hoạt là ReLU.
3. **BatchNormalization:** Lớp này chuẩn hóa đầu vào bằng cách áp dụng một phép chuẩn hóa batch, giúp giảm thiểu hiện tượng biến thiên (covariate shift) và tăng cường ổn định và tốc độ học của mô hình.
4. **Activation (ReLU):** Lớp kích hoạt ReLU thực hiện hàm kích hoạt ReLU (Rectified Linear Unit) trên đầu ra từ các lớp convolution và batch normalization, giúp giải quyết vấn đề về tính toán phi tuyến tính và tăng tính phi tuyến tính của mô hình.
5. **MaxPooling2D:** Lớp này thực hiện phép gộp giảm kích thước của đầu ra từ các lớp convolution bằng cách chọn giá trị lớn nhất từ một vùng cụ thể trong đầu vào.
6. **GlobalAveragePooling2D:** Lớp này thực hiện phép gộp trung bình trên toàn bộ các đặc trưng trong mỗi kênh của đầu ra từ các lớp convolution trước đó, giảm kích thước của tensor đầu ra xuống một vector duy nhất.
7. **Dense:** Lớp này thực hiện phép kết nối đầy đủ giữa các neuron, trong đó mỗi neuron trong lớp này kết nối với tất cả các neuron trong lớp trước đó.

Lớp Dense cuối cùng có hàm kích hoạt softmax để tạo ra xác suất dự đoán cho các lớp đầu ra.

8. **Dropout:** Lớp này áp dụng kỹ thuật dropout để ngẫu nhiên "tắt" một số lượng neuron trong quá trình huấn luyện, giúp tránh overfitting bằng cách ngăn chặn sự phụ thuộc quá mức vào một số neuron cụ thể. Trong mạng này, dropout được sử dụng với tỷ lệ dropout là 0.2.

- Chuyển sang code python và dùng thư viện Keras

Chuyển đổi mô hình mạng CNN đã xây dựng ở trên từ Matlab sang tensorflow bằng cách sử dụng hàm Matlab: [exportNetworkToTensorFlow](#).



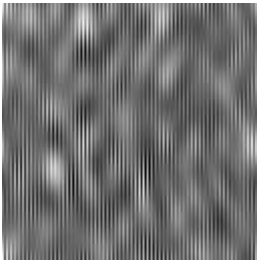
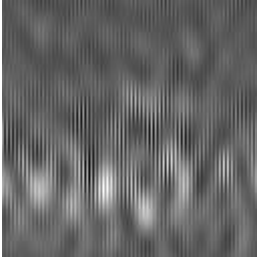
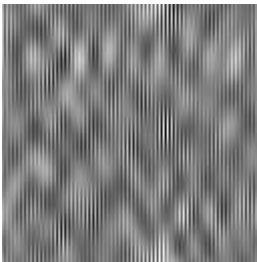
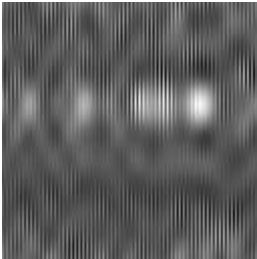
The screenshot shows the MATLAB Command Window with the following code and output:

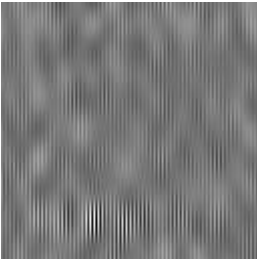
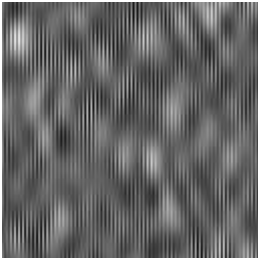
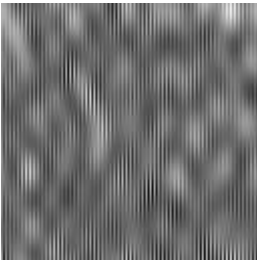
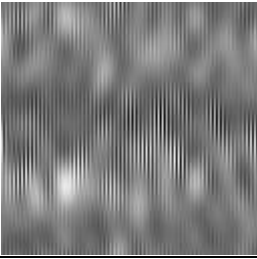
```
>> exportNetworkToTensorFlow(layers,"duancuoiky");
Warning: Unable to export untrained normalizations for these input layers: imageinput. You can train
before exporting or set the "Normalization" value to "none".
> In nnet.internal.cnn.tf_exporter.ExportNetworkToTensorFlowImpl>maybeWarnaboutInputNorms (line 189)
In nnet.internal.cnn.tf_exporter.ExportNetworkToTensorFlowImpl>convertToNetworkIfNeeded (line 141)
In nnet.internal.cnn.tf_exporter.ExportNetworkToTensorFlowImpl>convertArgs (line 75)
In nnet.internal.cnn.tf_exporter.ExportNetworkToTensorFlowImpl (line 42)
In exportNetworkToTensorFlow (line 42)
Warning: Layer 'scaling': Layer class 'rl.layer.ScalingLayer' was exported into an incomplete TensorFlow
custom layer file. The custom layer definition must be completed or the file must be replaced before the
model can be loaded into TensorFlow.
fx >> |
```

The Workspace window on the right shows a variable named 'layers' with a value of '37x1 Layer'.

Hình 3.5 Chuyển đổi mô hình từ Matlab sang file code python

3.2.3 Dữ liệu đầu ra được phân loại theo các lớp như sau

STT	Ảnh đầu vào	Tên phân loại
1		CPFSK
2		LFM
3		SSB-AM
4		Rect

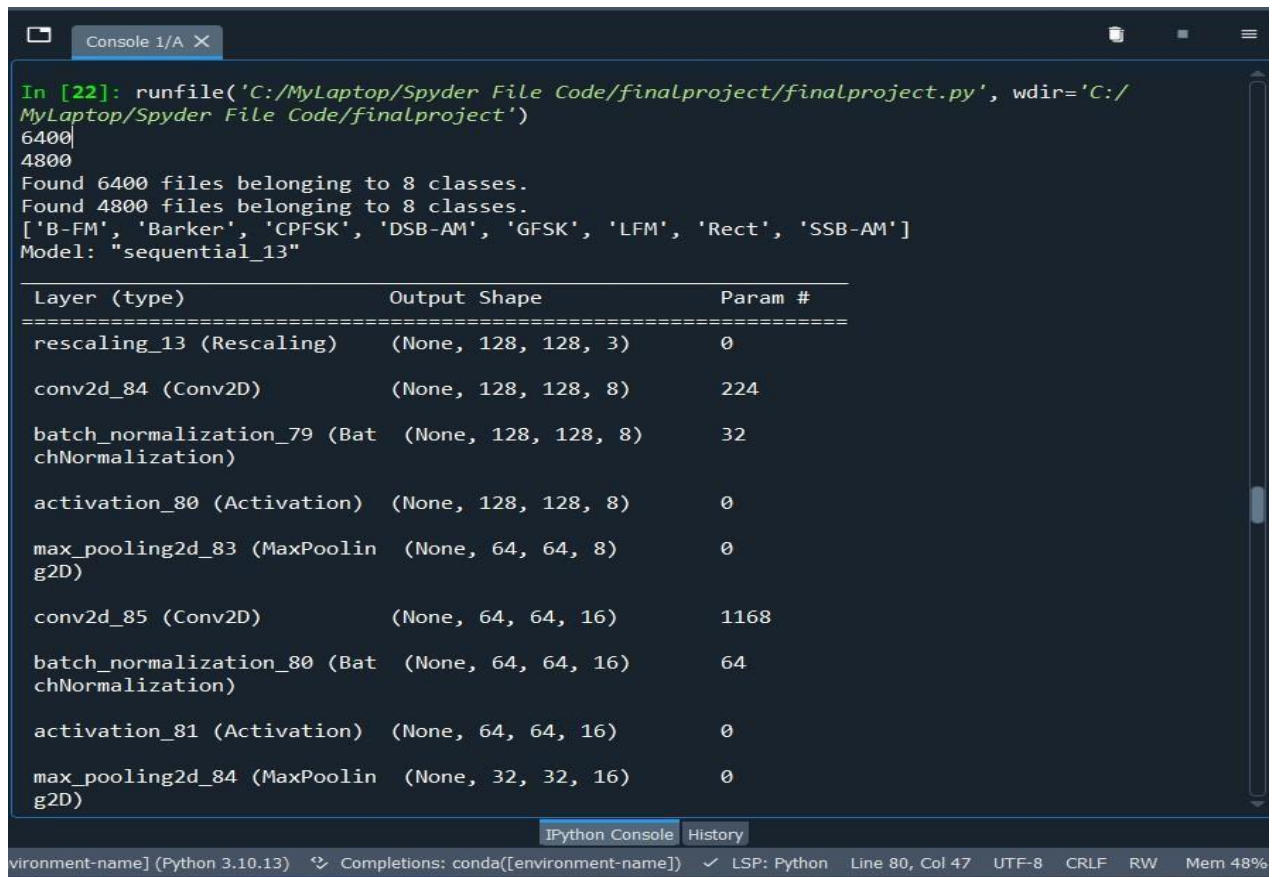
5		GFSK
6		DSB-AM
7		B-FM
8		Barker

Hình 3.6 Bảng phân tích đầu ra phân loại tín hiệu

CHƯƠNG 4: KẾT QUẢ THÍ NGHIỆM

4.1. KẾT QUẢ SAU KHI HUẤN LUYỆN

Quá trình huấn luyện tập dữ liệu sau khi chạy code Python trên spyder



```
In [22]: runfile('C:/MyLaptop/Spyder File Code/finalproject/finalproject.py', wdir='C:/MyLaptop/Spyder File Code/finalproject')
6400
4800
Found 6400 files belonging to 8 classes.
Found 4800 files belonging to 8 classes.
['B-FM', 'Barker', 'CPFSK', 'DSB-AM', 'GFSK', 'LFM', 'Rect', 'SSB-AM']
Model: "sequential_13"
```

Layer (type)	Output Shape	Param #
rescaling_13 (Rescaling)	(None, 128, 128, 3)	0
conv2d_84 (Conv2D)	(None, 128, 128, 8)	224
batch_normalization_79 (Batch Normalization)	(None, 128, 128, 8)	32
activation_80 (Activation)	(None, 128, 128, 8)	0
max_pooling2d_83 (MaxPooling2D)	(None, 64, 64, 8)	0
conv2d_85 (Conv2D)	(None, 64, 64, 16)	1168
batch_normalization_80 (Batch Normalization)	(None, 64, 64, 16)	64
activation_81 (Activation)	(None, 64, 64, 16)	0
max_pooling2d_84 (MaxPooling2D)	(None, 32, 32, 16)	0

Python Console History

environment-name] (Python 3.10.13) Completions: conda[environment-name] LSP: Python Line 80, Col 47 UTF-8 CRLF RW Mem 48%

```
Console 1/A X
max_pooling2d_84 (MaxPoolin (None, 32, 32, 16) 0
g2D)
conv2d_86 (Conv2D) (None, 32, 32, 32) 4640
batch_normalization_81 (Bat (None, 32, 32, 32) 128
chNormalization)
activation_82 (Activation) (None, 32, 32, 32) 0
max_pooling2d_85 (MaxPoolin (None, 16, 16, 32) 0
g2D)
conv2d_87 (Conv2D) (None, 16, 16, 64) 18496
batch_normalization_82 (Bat (None, 16, 16, 64) 256
chNormalization)
activation_83 (Activation) (None, 16, 16, 64) 0
max_pooling2d_86 (MaxPoolin (None, 8, 8, 64) 0
g2D)
conv2d_88 (Conv2D) (None, 8, 8, 64) 36928
batch_normalization_83 (Bat (None, 8, 8, 64) 256
chNormalization)
activation_84 (Activation) (None, 8, 8, 64) 0
max_pooling2d_87 (MaxPoolin (None, 4, 4, 64) 0
g2D)

IPython Console History
environment-name] (Python 3.10.13) Completions: conda[[environment-name]] LSP: Python Line 80, Col 47 UTF-8 CRLF RW Mem 48%

=====
Total params: 293,432
Trainable params: 292,552
Non-trainable params: 880
```

Hình 4.1 Tổng trọng số được ước tính trong mạng

- **Nhận xét** : Con số khá giống những gì đã phân tích ở phần trước

- Huấn luyện mô hình trên với tệp test_set

```

Epoch 1/40
75/75 [=====] - 16s 198ms/step - loss: 0.7186 - accuracy: 0.6973 - val_loss:
2.3053 - val_accuracy: 0.1250 - lr: 0.0010
Epoch 2/40
75/75 [=====] - 16s 208ms/step - loss: 0.3552 - accuracy: 0.8290 - val_loss:
2.6879 - val_accuracy: 0.2352 - lr: 0.0010
Epoch 3/40
75/75 [=====] - 17s 222ms/step - loss: 0.2916 - accuracy: 0.8421 - val_loss:
3.2915 - val_accuracy: 0.1298 - lr: 0.0010
Epoch 4/40
75/75 [=====] - 16s 213ms/step - loss: 0.2641 - accuracy: 0.8675 - val_loss:
1.3938 - val_accuracy: 0.3965 - lr: 0.0010
Epoch 5/40
75/75 [=====] - 15s 194ms/step - loss: 0.2514 - accuracy: 0.8660 - val_loss:
1.0241 - val_accuracy: 0.5692 - lr: 0.0010
Epoch 6/40
75/75 [=====] - 15s 198ms/step - loss: 0.2205 - accuracy: 0.8838 - val_loss:
0.2763 - val_accuracy: 0.8523 - lr: 0.0010
Epoch 7/40
75/75 [=====] - 15s 193ms/step - loss: 0.1943 - accuracy: 0.9092 - val_loss:
0.1856 - val_accuracy: 0.9294 - lr: 0.0010
Epoch 8/40
75/75 [=====] - 15s 194ms/step - loss: 0.1603 - accuracy: 0.9329 - val_loss:
0.0871 - val_accuracy: 0.9725 - lr: 0.0010
Epoch 9/40
75/75 [=====] - 14s 184ms/step - loss: 0.1060 - accuracy: 0.9592 - val_loss:
0.1287 - val_accuracy: 0.9583 - lr: 0.0010
Epoch 10/40
75/75 [=====] - 15s 200ms/step - loss: 0.1122 - accuracy: 0.9615 - val_loss:
0.1475 - val_accuracy: 0.9404 - lr: 0.0010
Epoch 11/40
75/75 [=====] - 14s 187ms/step - loss: 0.0854 - accuracy: 0.9692 - val_loss:
0.0356 - val_accuracy: 0.9890 - lr: 0.0010
Epoch 12/40
75/75 [=====] - 15s 198ms/step - loss: 0.0449 - accuracy: 0.9854 - val_loss:
0.0435 - val_accuracy: 0.9850 - lr: 0.0010
Epoch 13/40
75/75 [=====] - 15s 196ms/step - loss: 0.0359 - accuracy: 0.9877 - val_loss:
0.1547 - val_accuracy: 0.9398 - lr: 0.0010
Epoch 14/40
75/75 [=====] - ETA: 0s - loss: 0.0520 - accuracy: 0.9827
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
75/75 [=====] - 15s 196ms/step - loss: 0.0520 - accuracy: 0.9827 - val_loss:
0.0834 - val_accuracy: 0.9665 - lr: 0.0010
Epoch 15/40
75/75 [=====] - 14s 182ms/step - loss: 0.0204 - accuracy: 0.9944 - val_loss:
0.0064 - val_accuracy: 0.9990 - lr: 3.0000e-04
Epoch 16/40
75/75 [=====] - 15s 198ms/step - loss: 0.0052 - accuracy: 0.9992 - val_loss:
0.0019 - val_accuracy: 0.9998 - lr: 3.0000e-04
Epoch 17/40
75/75 [=====] - 15s 194ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss:
5.2958e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 18/40
75/75 [=====] - 14s 181ms/step - loss: 0.0022 - accuracy: 0.9998 - val_loss:
3.6114e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 19/40
75/75 [=====] - 14s 188ms/step - loss: 0.0017 - accuracy: 0.9998 - val_loss:
3.9079e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 20/40
75/75 [=====] - 14s 185ms/step - loss: 0.0024 - accuracy: 0.9998 - val_loss:
3.2166e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 21/40

```

IPython Console History

conda: [environment-name] (Python 3.10.13) Completions: conda([environment-name]) LSP: Python Line 102, Col 14 UTF-8 CRLF RW Mem 68%

Console 1/A X

```

Epoch 11/40
75/75 [=====] - 14s 187ms/step - loss: 0.0854 - accuracy: 0.9692 - val_loss:
0.0356 - val_accuracy: 0.9890 - lr: 0.0010
Epoch 12/40
75/75 [=====] - 15s 198ms/step - loss: 0.0449 - accuracy: 0.9854 - val_loss:
0.0435 - val_accuracy: 0.9850 - lr: 0.0010
Epoch 13/40
75/75 [=====] - 15s 196ms/step - loss: 0.0359 - accuracy: 0.9877 - val_loss:
0.1547 - val_accuracy: 0.9398 - lr: 0.0010
Epoch 14/40
75/75 [=====] - ETA: 0s - loss: 0.0520 - accuracy: 0.9827
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
75/75 [=====] - 15s 196ms/step - loss: 0.0520 - accuracy: 0.9827 - val_loss:
0.0834 - val_accuracy: 0.9665 - lr: 0.0010
Epoch 15/40
75/75 [=====] - 14s 182ms/step - loss: 0.0204 - accuracy: 0.9944 - val_loss:
0.0064 - val_accuracy: 0.9990 - lr: 3.0000e-04
Epoch 16/40
75/75 [=====] - 15s 198ms/step - loss: 0.0052 - accuracy: 0.9992 - val_loss:
0.0019 - val_accuracy: 0.9998 - lr: 3.0000e-04
Epoch 17/40
75/75 [=====] - 15s 194ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss:
5.2958e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 18/40
75/75 [=====] - 14s 181ms/step - loss: 0.0022 - accuracy: 0.9998 - val_loss:
3.6114e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 19/40
75/75 [=====] - 14s 188ms/step - loss: 0.0017 - accuracy: 0.9998 - val_loss:
3.9079e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 20/40
75/75 [=====] - 14s 185ms/step - loss: 0.0024 - accuracy: 0.9998 - val_loss:
3.2166e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 21/40

```

IPython Console History

conda: [environment-name] (Python 3.10.13) Completions: conda([environment-name]) LSP: Python Line 102, Col 14 UTF-8 CRLF RW Mem 68%


```
Console 1/A X
Epoch 21/40
75/75 [=====] - 14s 189ms/step - loss: 0.0018 - accuracy: 0.9998 - val_loss:
1.9249e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 22/40
75/75 [=====] - 15s 194ms/step - loss: 9.3971e-04 - accuracy: 1.0000 - val_loss:
1.3975e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 23/40
75/75 [=====] - 15s 196ms/step - loss: 0.0026 - accuracy: 0.9992 - val_loss:
3.1164e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 24/40
75/75 [=====] - ETA: 0s - loss: 0.0015 - accuracy: 1.0000
Epoch 24: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
75/75 [=====] - 13s 177ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss:
1.5125e-04 - val_accuracy: 1.0000 - lr: 3.0000e-04
Epoch 25/40
75/75 [=====] - 14s 181ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss:
1.0550e-04 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 26/40
75/75 [=====] - 16s 206ms/step - loss: 7.8895e-04 - accuracy: 1.0000 - val_loss:
9.0502e-05 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 27/40
75/75 [=====] - 15s 193ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss:
9.6182e-05 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 28/40
75/75 [=====] - 13s 172ms/step - loss: 7.3916e-04 - accuracy: 1.0000 - val_loss:
8.2525e-05 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 29/40
75/75 [=====] - ETA: 0s - loss: 0.0012 - accuracy: 0.9998
Epoch 29: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
75/75 [=====] - 14s 193ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss:
7.8540e-05 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 30/40
75/75 [=====] - 14s 191ms/step - loss: 7.8238e-04 - accuracy: 1.0000 - val_loss:
7.8540e-05 - val_accuracy: 1.0000 - lr: 9.0000e-05
Epoch 31/40
75/75 [=====] - 14s 180ms/step - loss: 8.1567e-04 - accuracy: 1.0000 - val_loss:
6.9489e-05 - val_accuracy: 1.0000 - lr: 2.7000e-05
Epoch 32/40
75/75 [=====] - ETA: 0s - loss: 7.3837e-04 - accuracy: 1.0000
Epoch 32: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
75/75 [=====] - 15s 199ms/step - loss: 7.3837e-04 - accuracy: 1.0000 - val_loss:
6.6692e-05 - val_accuracy: 1.0000 - lr: 2.7000e-05
Epoch 33/40
75/75 [=====] - 13s 174ms/step - loss: 6.9103e-04 - accuracy: 1.0000 - val_loss:
6.5466e-05 - val_accuracy: 1.0000 - lr: 8.1000e-06
Epoch 34/40
75/75 [=====] - 14s 191ms/step - loss: 6.4275e-04 - accuracy: 1.0000 - val_loss:
6.4524e-05 - val_accuracy: 1.0000 - lr: 8.1000e-06
Epoch 35/40
75/75 [=====] - ETA: 0s - loss: 7.4408e-04 - accuracy: 1.0000
Epoch 35: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
75/75 [=====] - 15s 206ms/step - loss: 7.4408e-04 - accuracy: 1.0000 - val_loss:
6.4245e-05 - val_accuracy: 1.0000 - lr: 8.1000e-06
Epoch 36/40
75/75 [=====] - 14s 185ms/step - loss: 7.8288e-04 - accuracy: 1.0000 - val_loss:
6.4268e-05 - val_accuracy: 1.0000 - lr: 2.4300e-06
Epoch 37/40
75/75 [=====] - 13s 179ms/step - loss: 8.6919e-04 - accuracy: 1.0000 - val_loss:
6.4100e-05 - val_accuracy: 1.0000 - lr: 2.4300e-06
Epoch 38/40
75/75 [=====] - ETA: 0s - loss: 6.8890e-04 - accuracy: 1.0000
Epoch 38: ReduceLROnPlateau reducing learning rate to 7.289999985005124e-07.
75/75 [=====] - 14s 182ms/step - loss: 6.8890e-04 - accuracy: 1.0000 - val_loss:
6.3613e-05 - val_accuracy: 1.0000 - lr: 2.4300e-06
Epoch 39/40
75/75 [=====] - 14s 189ms/step - loss: 6.9091e-04 - accuracy: 1.0000 - val_loss:
6.3533e-05 - val_accuracy: 1.0000 - lr: 7.2900e-07
Epoch 40/40
75/75 [=====] - 15s 200ms/step - loss: 8.2388e-04 - accuracy: 1.0000 - val_loss:
6.3459e-05 - val_accuracy: 1.0000 - lr: 7.2900e-07
75/75 [=====] - 4s 44ms/step - loss: 6.3459e-05 - accuracy: 1.0000
```

Hình 4.2 Độ chính và giá trị hàm lỗi theo từng epoch

- Nhận xét:

- Giá trị của hàm lỗi giảm dần theo từng epoch. Giá trị của hàm lỗi ở epoch 1 là 0.7186, giá trị hàm lỗi ở epoch 10 giảm xuống còn 0.1122 và đến epoch 40 là 8.2388e-04 .
- Độ chính xác tăng dần qua từng epoch, tăng lên tới đa 100% ở epoch thứ 17 và duy trì mức ngưỡng 99%-100% đến epoch 40 .

Với yêu cầu là chạy 3 lần code trên tệp test_set và ghi lại giá trị accuracy cho mỗi epoch ta có bảng tổng hợp sau đây:

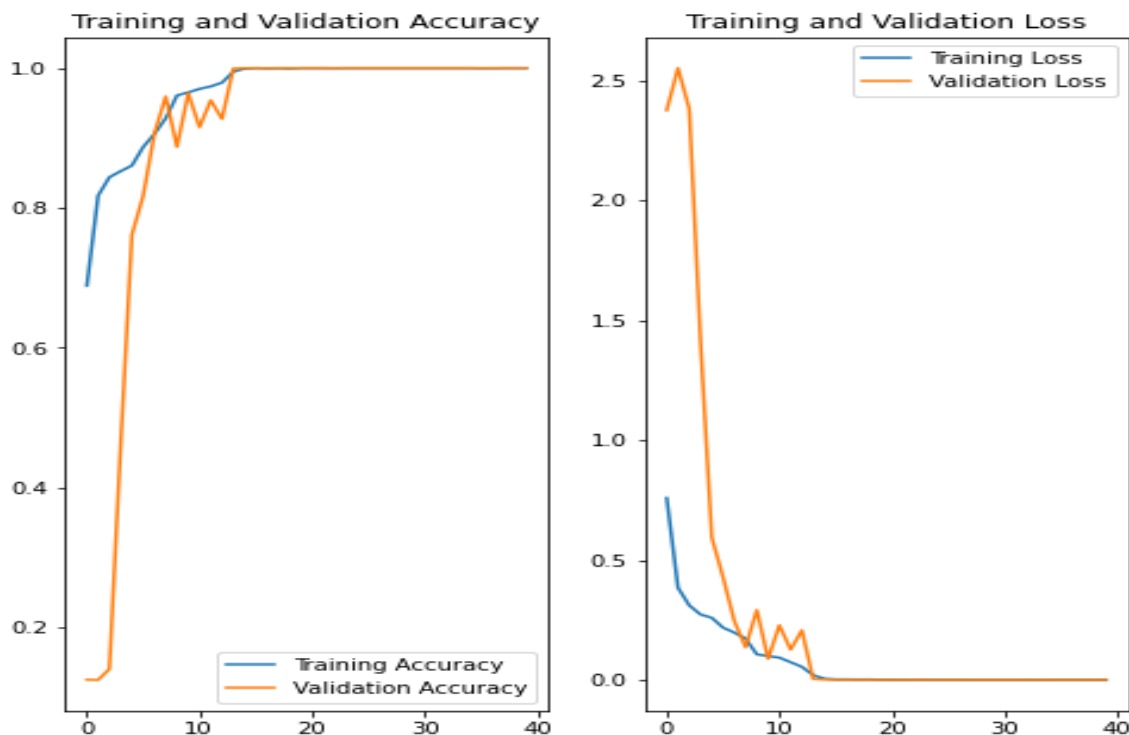
Lần chạy Epoch	Lần 1	Lần 2	Lần 3
1	0.7016	0.7127	0.7048
2	0.8161	0.8172	0.8148
3	0.8402	0.8416	0.8372
4	0.8614	0.8600	0.8587
5	0.8775	0.8719	0.8648
6	0.8917	0.8891	0.8756
7	0.9094	0.8941	0.8963
8	0.9327	0.9028	0.9408
9	0.9397	0.9262	0.9586
10	0.9517	0.9441	0.9744
11	0.9617	0.9645	0.9912
12	0.9691	0.9778	0.9956
13	0.9928	0.9712	0.9983
14	0.9978	0.9766	0.9987
15	0.9992	0.9791	0.9992
16	0.9991	0.9911	0.9989
17	0.9995	0.9975	0.9984
18	1.0000	0.9992	0.9992
19	0.9998	0.9995	0.9991
20	1.0000	0.9995	0.9991
21	0.9998	0.9998	0.9994
22	1.0000	1.0000	0.9986
23	1.0000	1.0000	0.9997

24	0.9998	1.0000	0.9995
25	0.9998	1.0000	0.9994
26	0.9998	0.9997	1.0000
27	1.0000	1.0000	1.0000
28	1.0000	0.9998	0.9991
29	1.0000	0.9997	0.9996
30	0.9998	1.0000	1.0000
31	0.9998	1.0000	0.9989
32	1.0000	1.0000	1.0000
33	1.0000	1.0000	1.0000
34	1.0000	0.9997	1.0000
35	1.0000	1.0000	1.0000
36	0.9998	0.9998	0.9996
37	1.0000	0.9998	0.9997
38	1.0000	1.0000	1.0000
39	1.0000	0.9997	1.0000
40	1.0000	1.0000	1.0000
Trung bình	0.9659	0.9628	0.9674

(Accuracy Final được tính bằng cách lấy trung bình accuracy 3 lần chạy ở epoch thứ 40)
Accuracy Final = (1.0000+ 1.0000 + 1.0000) / 3 = 1.0000

Vậy ta có thể đưa ra nhận xét tổng kết rằng độ chính xác của mạng CNN này là **100 %**

4.2 ĐỒ THỊ SỰ BIẾN THIÊN CỦA HÀM LOSS VÀ ACCURACY



Hình 4.3 Đồ thị Training và Validation Accuracy/Loss

Nhận xét

- Training Accuracy và Validation Accuracy:

+ Đường Training Accuracy tăng dần và cuối cùng đạt mức gần 1, cho thấy mô hình học được tốt trên dữ liệu huấn luyện.

+ Đường Validation Accuracy tăng lên đáng kể ở đầu và sau đó giữ ổn định ở một mức độ cao. Sự ổn định này cho thấy mô hình đang hoạt động tốt trên dữ liệu kiểm tra, không có dấu hiệu của overfitting.

- Training Loss và Validation Loss:

+ Đường Training Loss giảm dần và cuối cùng hội tụ ở mức thấp, cho thấy mô hình đang học được từ dữ liệu huấn luyện.

+ Đường Validation Loss giảm lên đầu và sau đó giữ ổn định. Điều này cho thấy mô hình đang hoạt động tốt.

=> Tổng quan, mô hình có vẻ hoạt động tốt trên tập dữ liệu kiểm tra, với độ chính xác và sự mất mát ổn định.

4.3 ĐƯA HÌNH ẢNH TỪ TẬP TEST ĐỂ PHÂN LOẠI

- Đoạn code dùng model đã xây dựng để phân loại ảnh từ tập test_set

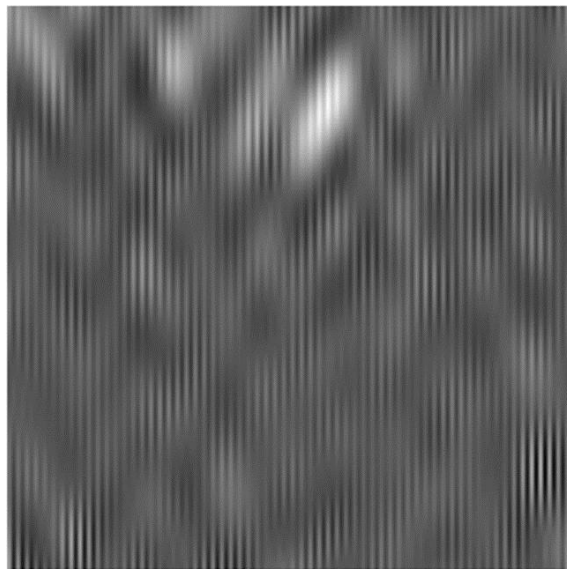
```
# Chọn ngẫu nhiên một bức ảnh từ tập test
test_image_paths = list(test_dir.glob('*/*.png'))
random_test_image_path = np.random.choice(test_image_paths)

# Đọc và tiền xử lý bức ảnh
img = keras.preprocessing.image.load_img(
    random_test_image_path, target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Thêm một chiều để phù hợp với batch size

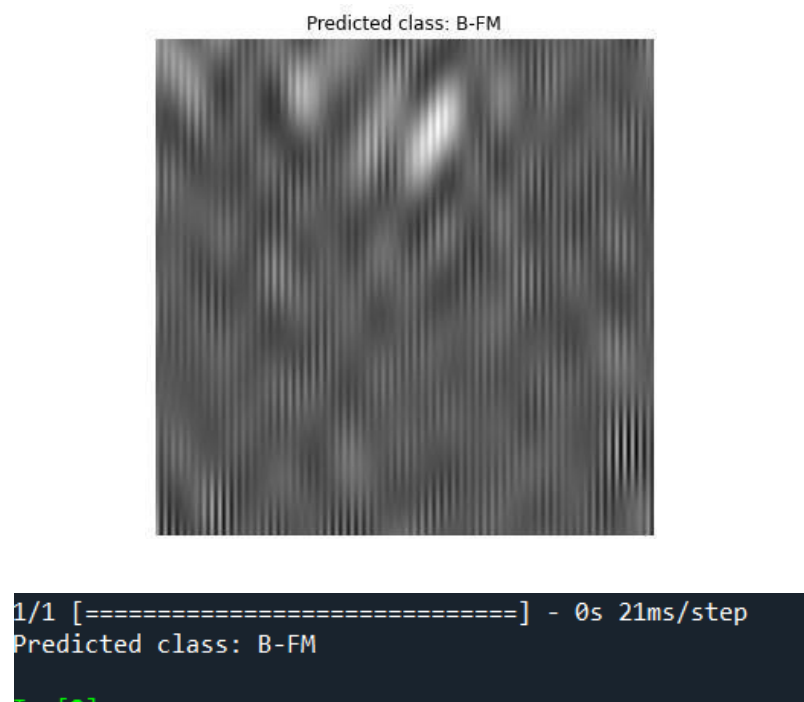
# Dự đoán lớp của bức ảnh
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions)

# In kết quả dự đoán và hiển thị hình ảnh
print("Predicted class:", class_names[predicted_class])
plt.figure(figsize=(6, 6))
plt.imshow(img)
plt.title("Predicted class: {}".format(class_names[predicted_class]))
plt.axis("off")
plt.show()
```

- So sánh kết quả trước và sau phân loại tín hiệu :



Hình 4.4 Hình ảnh trước khi nhận diện (ảnh từ file B-FM)



Hình 4.5 Hình ảnh sau khi phân loại (B-FM)

Nhận xét:

- Kết quả cho ra là chính xác đồng thời khi test với nhiều trường hợp khác thì độ chính xác cũng rất cao.

KẾT LUẬN

Trong báo cáo này, chúng tôi đã đề xuất một mô hình dựa trên mạng nơron tích chập (CNN) để phân loại các dạng tín hiệu radar . Mô hình này có 6 lớp nơron tích chập (Convolution) và 2 lớp nơron liên kết đầy đủ (Fully Connected), tổng số tham số là khoảng hơn 293 ngàn.

Như vậy, có thể khẳng định mô hình của chúng tôi có độ phức tạp ở mức vừa phải, phù hợp với các hệ thống xử lý ở mức trung bình và đem lại tiềm năng khả thi trong ứng dụng thực tiễn.

Mặc dù độ phức tạp của mô hình ở mức thấp so với các mô hình khác, nhưng kết quả thử nghiệm cho thấy tính hiệu quả của phân lớp khá cao. Hiện nay do điều kiện tính toán nên chỉ áp dụng số lần huấn luyện còn thấp, nếu được huấn luyện ở mức độ sâu hơn thì kỳ vọng sẽ đem lại kết quả cao hơn nữa.

Để phát triển thêm cho mô hình, chúng tôi sẽ tìm hiểu và thiết kế một hệ thống thu thập dữ liệu hình ảnh để tạo bộ dữ liệu huấn luyện đa dạng cho mô hình, từ đó xây dựng một ứng dụng cho bài toán thực tiễn như hệ thống đánh giá và giám sát tín hiệu radar ở trạm thu ,trong lĩnh vực quân sự, các hệ thống radar được sử dụng để phát hiện, theo dõi và phân loại các mục tiêu như máy bay, tàu thuyền, và phương tiện di động khác hoặc trong y tế và cứu thương, radar y tế có thể được sử dụng để phát hiện các biểu hiện bất thường trong cơ thể con người .

PHỤ LỤC

Chương trình được thực hiện trên môi trường Spyder (Anaconda) với chương trình như sau:

```
finalproject.py X
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow import keras
5 from tensorflow.keras import layers
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras import layers, models
8
9 import pathlib
10 training_dir = pathlib.Path('C:/MyLaptop/Spyder File Code/finalproject/training_set')
11 training_count = len(list(training_dir.glob('*/*.png')))
12 print(training_count)
13
14 test_dir = pathlib.Path('C:/MyLaptop/Spyder File Code/finalproject/test_set')
15 test_count = len(list(test_dir.glob('*/*.png')))
16 print(test_count)
17
18 batch_size = 64
19
20 img_height = 128
21 img_width = 128
22 train_ds = tf.keras.utils.image_dataset_from_directory(
23     training_dir,
24     validation_split=0,
25     seed=123,
26     image_size=(img_height, img_width),
27     batch_size=batch_size)
28
29 val_ds = tf.keras.utils.image_dataset_from_directory(
30     test_dir,
31     validation_split=0,
32     seed=113,
33     image_size=(img_height, img_width),
34     batch_size=batch_size)
35
36 class_names = train_ds.class_names
37 print(class_names)
38
39 def create_model(input_shape, num_classes):
40     model = models.Sequential([
41         layers.Rescaling(1./255, input_shape=input_shape),
42         layers.Conv2D(8, (3, 3), padding='same'),
43         layers.BatchNormalization(),
44         layers.Activation('relu'),
45         layers.MaxPooling2D(),
46         layers.Conv2D(16, (3, 3), padding='same'),
47         layers.BatchNormalization(),
48         layers.Activation('relu'),
49         layers.MaxPooling2D(),
50         layers.Conv2D(32, (3, 3), padding='same'),
51         layers.BatchNormalization(),
52         layers.Activation('relu'),
53         layers.MaxPooling2D(),
54         layers.Conv2D(64, (3, 3), padding='same'),
55         layers.BatchNormalization(),
56         layers.Activation('relu'),
57         layers.MaxPooling2D(),
58         layers.Conv2D(64, (3, 3), padding='same'),
59         layers.BatchNormalization(),
60         layers.Activation('relu'),
61         layers.MaxPooling2D(),
62         layers.Conv2D(128, (3, 3), padding='same'),
63         layers.BatchNormalization(),
64         layers.Activation('relu'),
65         layers.MaxPooling2D(),
66         layers.Conv2D(128, (3, 3), padding='same'),
67         layers.BatchNormalization(),
68         layers.Activation('relu'),
69         layers.MaxPooling2D(),
70         layers.GlobalAveragePooling2D(),
71         layers.Dense(64, activation='relu'),
72         layers.Dropout(0.2),
73         layers.Dense(num_classes, activation='softmax')
74     ])
75     return model
76
```

```
finalproject.py X
37 print(class_names)
38
39 def create_model(input_shape, num_classes):
40     model = models.Sequential([
41         layers.Rescaling(1./255, input_shape=input_shape),
42         layers.Conv2D(8, (3, 3), padding='same'),
43         layers.BatchNormalization(),
44         layers.Activation('relu'),
45         layers.MaxPooling2D(),
46         layers.Conv2D(16, (3, 3), padding='same'),
47         layers.BatchNormalization(),
48         layers.Activation('relu'),
49         layers.MaxPooling2D(),
50         layers.Conv2D(32, (3, 3), padding='same'),
51         layers.BatchNormalization(),
52         layers.Activation('relu'),
53         layers.MaxPooling2D(),
54         layers.Conv2D(64, (3, 3), padding='same'),
55         layers.BatchNormalization(),
56         layers.Activation('relu'),
57         layers.MaxPooling2D(),
58         layers.Conv2D(64, (3, 3), padding='same'),
59         layers.BatchNormalization(),
60         layers.Activation('relu'),
61         layers.MaxPooling2D(),
62         layers.Conv2D(128, (3, 3), padding='same'),
63         layers.BatchNormalization(),
64         layers.Activation('relu'),
65         layers.MaxPooling2D(),
66         layers.Conv2D(128, (3, 3), padding='same'),
67         layers.BatchNormalization(),
68         layers.Activation('relu'),
69         layers.MaxPooling2D(),
70         layers.GlobalAveragePooling2D(),
71         layers.Dense(64, activation='relu'),
72         layers.Dropout(0.2),
73         layers.Dense(num_classes, activation='softmax')
74     ])
75     return model
76
```

conda: [t


```

76
77 # Sử dụng hàm create_model để tạo mô hình
78 input_shape = (128, 128, 3)
79 num_classes = len(class_names) # Cần phải định nghĩa class_names trước khi sử dụng
80 model = create_model(input_shape, num_classes)
81 model.summary()
82
83 # Chạy Model trên
84 model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
85               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
86               metrics=['accuracy'])
87
88 # Xác định learning rate và gọi lại hàm để giảm
89 learning_rate_reduction = keras.callbacks.ReduceLROnPlateau(
90     monitor='val_loss', factor=0.3, patience=3, verbose=1, mode='auto')
91
92 # Huấn luyện mô hình trên tập test_set
93 epochs = 40
94 history = model.fit(
95     val_ds,
96     validation_data=val_ds,
97     epochs=epochs, shuffle=True, callbacks=[learning_rate_reduction])
98
99 # Đánh giá mô hình
100 accuracy = model.evaluate(val_ds)
101
102 # Vẽ kết quả
103 acc = history.history['accuracy']
104 val_acc = history.history['val_accuracy']
105 loss = history.history['loss']
106 val_loss = history.history['val_loss']
107 epochs_range = range(epochs)
108
109 plt.figure(figsize=(8, 8))
110 plt.subplot(1, 2, 1)
111 plt.plot(epochs_range, acc, label='Training Accuracy')
112 plt.plot(epochs_range, val_acc, label='Validation Accuracy')
113 plt.legend(loc='lower right')
114 plt.title('Training and Validation Accuracy')
115
116 plt.subplot(1, 2, 2)
117 plt.plot(epochs_range, loss, label='Training Loss')
118 plt.plot(epochs_range, val_loss, label='Validation Loss')
119 plt.legend(loc='upper right')
120 plt.title('Training and Validation Loss')
121 plt.show()

```

+ File code : được gửi kèm theo file pdf báo cáo

TÀI LIỆU THAM KHẢO

- [1] PGS. TS Trương Ngọc Sơn, *Trí tuệ nhân tạo cơ sở và ứng dụng*, Đại học quốc gia TP HCM, 2020.
- [2] Tài liệu online.
- [3] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, “Recent advances in convolutional neural network acceleration,” *Neurocomputing*, vol. 323, pp. 0–38, 2019, doi: 10.1016/j.neucom.2018.09.038.