|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.3

“Áp dụng mô hình học sâu CNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2023

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.5.3

“Áp dụng mô hình học sâu CNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc129729519)

[DANH MỤC HÌNH ẢNH 2](#_Toc129729520)

[CNN: Convolutional neural network 3](#_Toc129729521)

[1.1. Phép tính Convolution [2] 3](#_Toc129729522)

[1.2. Padding 4](#_Toc129729523)

[1.3. Stride 5](#_Toc129729524)

[1.4. Convolution neural network [1] 6](#_Toc129729525)

[1..4.1 Convolutional layer 6](#_Toc129729526)

[1.4.2. Pooling layer 10](#_Toc129729527)

[1.4.3. Fully connected layer 11](#_Toc129729528)

[1.5. Thực nghiệm 12](#_Toc129729529)

[TÀI LIỆU THAM KHẢO 19](#_Toc129729530)

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Phép tính Convolution 3](#_Toc129729498)

[Hình 1.2: padding của ma trận 4](#_Toc129729499)

[Hình 1.3: Các bước thực hiện convolution cho ma trận X với kernel K 4](#_Toc129729500)

[Hình 1.4: Ma trận X khi thêm viền 0 bên ngoài 4](#_Toc129729501)

[Hình 1.5: stride = 1, padding = 1 5](#_Toc129729502)

[Hình 1.6: padding = 1, stride = 2 5](#_Toc129729503)

[Hình 1.7: Mô hình neural network. 6](#_Toc129729504)

[Hình 1.8: Input layer và hidden layer 1 6](#_Toc129729505)

[Hình 1.9: Phép tính convolution ảnh xám biểu diễn ảnh dạng ma trận 7](#_Toc129729506)

[Hình 1.10: Phép tính convolution trên ảnh màu với k = 3 8](#_Toc129729507)

[Hình 1.11: Tensor X, W 3 chiều được viết dưới dạng 3 matrix [1] 8](#_Toc129729508)

[Hình 1.12: Thực hiện phép tính convolution trên ảnh màu 9](#_Toc129729509)

[Hình 1.13: Convolution layer đầu tiên 9](#_Toc129729510)

[Hình 1.14: Tính toán trong convolutional 9](#_Toc129729511)

[Hình 1.15: max pooling layer với size = (3,3), stride = 1, padding = 0 10](#_Toc129729512)

[Hình 1.16: Sau pooling layer (2\*2) 11](#_Toc129729513)

[Hình 1.17: Ví dụ về pooling layer 11](#_Toc129729514)

[Hình 1.18: Chuyển feature map sang flatten 11](#_Toc129729515)

[Hình 1.19: Cài đặt các thư viện cho chương trình 12](#_Toc129729516)

[Hình 1.20: Thông tin dữ liệu huấn luyện sau khi nạp vào chương trình 13](#_Toc129729517)

[Hình 1.21: Thông tin mô hình trong quá trình huấn luyện 16](#_Toc129729518)

# CNN: Convolutional neural network

## 1.1. Phép tính Convolution [2]

Để cho dễ hình dung tôi sẽ lấy ví dụ trên ảnh xám, tức là ảnh được biểu diễn dưới dạng ma trận A kích thước m\*n.

Ta định nghĩa kernel là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9... Ví dụ kernel kích thước 3\*3.

W =

Kí hiệu phép tính convolution (⊗), kí hiệu Y = X ⊗W

Với mỗi phần tử xi j trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử xi j làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính element-wise của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

Chart, scatter chart

Description automatically generated

Hình 1.1. Phép tính Convolution

Ví dụ khi tính tại x22 (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có x22 làm trung tâm có màu nền da cam như trong hình. Sau đó tính y11 = sum(A⊗W) = x11 ∗w11 +x12 ∗w12 + x13 ∗w13 +x21 ∗w21 +x22 ∗ w22 +x23 ∗w23 +x31 ∗w31 +x32 ∗w32 +x33 ∗w33 = 4. Và làm tương tự với các phần tử còn lại trong ma trận.

Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như x11? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X.

A picture containing chart

Description automatically generated

Hình 1.2: padding của ma trận

Nên bạn để ý thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là (m-k+1) \* (n-k+1).

A picture containing diagram

Description automatically generated

Hình 1.3: Các bước thực hiện convolution cho ma trận X với kernel K

## 1.2. Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X => Tìm cách giải quyết cho các phần tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.

A picture containing different

Description automatically generated

Hình 1.4: Ma trận X khi thêm viền 0 bên ngoài

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử x11 , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu. Phép tính này gọi là convolution với padding=1. Padding=k nghĩa là thêm k vector 0 vào mỗi phía (trên, dưới, trái, phải) của ma trận.

1.3. Stride

Như ở trên ta thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

A picture containing shelf

Description automatically generated

Hình 1.5: stride = 1, padding = 1

Tuy nhiên nếu stride=k (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử x1+i\*k, 1+j\*k.  Ví dụ k = 2

Text, table

Description automatically generated

Hình 1.6: padding = 1, stride = 2

Hiểu đơn giản là bắt đầu từ vị trí x11 sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X. Kích thước của ma trận Y là 3\*3 đã giảm đi đáng kể so với ma trận X. Công thức tổng quát cho phép tính convolution của ma trận X kích thước m\*n với kernel kích thước k\*k, stride = s, padding = p ra ma trận Y kích thước ( + 1) ∗ ( +1).

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

## 1.4. Convolution neural network [1]

### 1..4.1 Convolutional layer

Diagram

Description automatically generated

Hình 1.7: Mô hình neural network.

Mỗi hidden layer được gọi là fully connected layer, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là fully connected neural network (FCN).

Vấn đề của fully connected neural network với xử lý ảnh Như bài trước về xử lý ảnh, thì ảnh màu 64\*64 được biểu diễn dưới dạng 1 tensor 64\*64\*3. Nên để biểu thị hết nội dung của bức ảnh thì cần truyền vào input layer tất cả các pixel (64\*64\*3 = 12288). Nghĩa là input layer giờ có 12288 nodes.

Diagram, schematic

Description automatically generated

Hình 1.8: Input layer và hidden layer 1

Giả sử số lượng node trong hidden layer 1 là 1000. Số lượng weight W giữa

input layer và hidden layer 1 là 12288\*1000 = 12288000, số lượng bias là 1000 => tổng số parameter là: 12289000. Đấy mới chỉ là số parameter giữa input layer và hidden layer 1, trong model còn nhiều layer nữa, và nếu kích thước ảnh tăng, ví dụ 512\*512 thì số lượng parameter tăng cực kì nhanh => Cần giải pháp tốt hơn!!!

**Nhận xét:**

* Trong ảnh các pixel ở cạnh nhau thường có liên kết với nhau hơn là những pixel ở xa. Ví dụ như phép tính convolution trên ảnh ở bài trước. Để tìm các đường trong ảnh, ta áp dụng sobel kernel trên mỗi vùng kích thước 3\*3. Hay làm nét ảnh ta áp dụng sharpen kernel cũng trên vùng có kích thước 3\*3.
* Với phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh chia sẻ hệ số với nhau. => Áp dụng phép tính convolution vào layer trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.
* **Convolutional layer đầu tiên**

Bài trước phép tính convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận

Shape

Description automatically generated with low confidence

Hình 1.9: Phép tính convolution ảnh xám biểu diễn ảnh dạng ma trận

Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.

Diagram, engineering drawing

Description automatically generated

Hình 1.10: Phép tính convolution trên ảnh màu với k = 3

Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám

Box and whisker chart

Description automatically generated

Hình 1.11: Tensor X, W 3 chiều được viết dưới dạng 3 matrix [1]

Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là xijk. y11 = b + (x111 ∗ w111 + x121 ∗ w121 + x131 ∗ w131 + x211 ∗ w211 + x221 ∗ w221 + x231 ∗ w231 + x311 ∗ w311+x321 ∗w321+x331 ∗w331)+ (x112 ∗w112+x122 ∗w122+x132 ∗w132+x212 ∗w212+x222 ∗w222+ x232 ∗w232 +x312 ∗w312 +x322 ∗w322 +x332 ∗w332)+ (x113 ∗w113 +x123 ∗w123 +x133 ∗w133 +x213 ∗ w213 +x223 ∗w223 +x233 ∗w233 +x313 ∗w313 +x323 ∗w323 +x333 ∗w333) = −25

* Nhận xét:
* Output Y của phép tính convolution trên ảnh màu là 1 matrix.
* Có 1 hệ số bias được cộng vào sau bước tính tổng các phần tử của phép tính element-wise.

Các quy tắc đối với padding và stride hoàn toàn tương tự như ở bài trước

Với mỗi kernel khác nhau ta sẽ học được những đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được nhiều thuộc tính của ảnh. Vì mỗi kernel cho ra output là 1 matrix nên k kernel sẽ cho ra k output matrix. Ta kết hợp k output matrix này lại thành 1 tensor 3 chiều có chiều sâu k.

Calendar

Description automatically generated

Hình 1.12: Thực hiện phép tính convolution trên ảnh màu

Diagram

Description automatically generated

Hình 1.13: Convolution layer đầu tiên

**Convolutional layer tổng quát**

Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D. Kernel có kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

Convolutional layer áp dụng K kernel=> Output của layer là tensor 3 chiều có kích thước: + 1) \* ( + 1) \* K

Diagram

Description automatically generated

Hình 1.14: Tính toán trong convolutional

* **Lưu ý:**
* Output của convolutional layer sẽ qua hàm non-linear activation function trước khi trở thành input của convolutional layer tiếp theo.
* Tổng số parameter của layer: Mỗi kernel có kích thước F\*F\*D và có 1 hệ số bias, nên tổng parameter của 1 kernel là F\*F\*D + 1. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là K \* (F\*F\*D + 1).

### 1.4.2. Pooling layer

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu giúp giảm các phép tính toán trong model.

Bên cạnh đó, với phép pooling kích thước ảnh giảm, do đó lớp convolution học được các vùng có kích thước lớn hơn. Ví dụ như ảnh kích thước 224\*224 qua pooling về 112\*112 thì vùng 3\*3 ở ảnh 112\*112 tương ứng với vùng 6\*6 ở ảnh ban đầu. Vì vậy qua các pooling thì kích thước ảnh nhỏ đi và convolutional layer sẽ học được các thuộc tính lớn hơn.

Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.

A screenshot of a game

Description automatically generated with low confidence

Hình 1.15: max pooling layer với size = (3,3), stride = 1, padding = 0

Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên.

A picture containing text, clock, screenshot

Description automatically generated

Hình 1.16: Sau pooling layer (2\*2)

Diagram

Description automatically generated

Hình 1.17: Ví dụ về pooling layer

Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.

### 1.4.3. Fully connected layer

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,...) thì tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W\*D, 1)

Graphical user interface

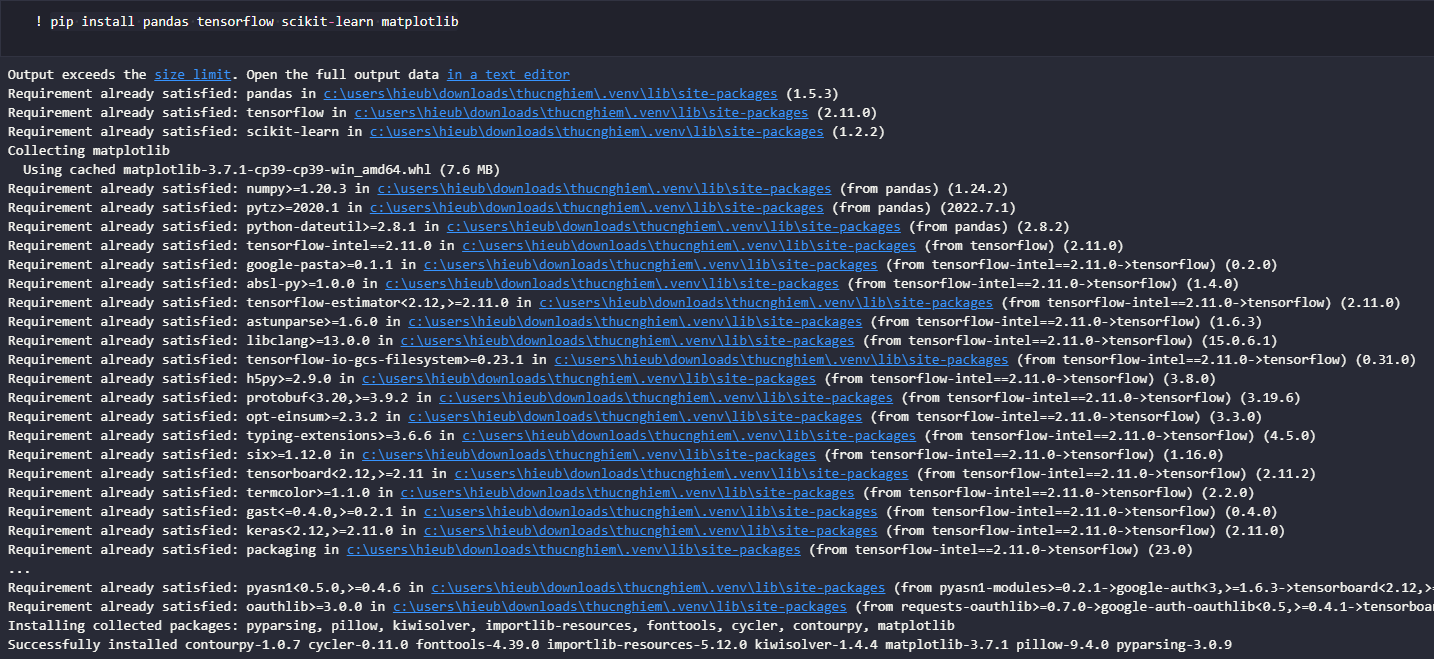
Description automatically generated

Hình 1.18: Chuyển feature map sang flatten

Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

1.5. Thực nghiệm

Dựa trên các đặc điểm trên, chúng ta sẽ áp dụng mô hình học máy CNN để huấn luyện cũng như chạy thực nghiệm việc phân loại mã độc với dữ liệu đầu vào là các đặc trưng. Để có thể sử dụng được thư viện, ta tiến hành cài đặt các thư viện cần thiết thông qua câu lệnh *pip install pandas tensorflow scikit-learn matplotlib*.



Hình 1.19: Cài đặt các thư viện cho chương trình

Sau khi cài đặt xong, ta tiến hành nạp các thư viện cần thiết vào chương trình.

import pandas as pd

import csv

import keras

from keras.layers import Input

from keras.models import Model

from keras.layers import Dense

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.pooling import MaxPooling2D

from keras.layers.merging import concatenate

from keras.utils import to\_categorical

from keras.callbacks import TensorBoard, Callback

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import numpy as np

Tiếp theo, ta tiến hành nạp dữ liệu sử dụng để huấn luyện và kiểm tra từ file có định dạng .csv vào chương trình thông qua thư viện *pandas*.

train\_df=pd.read\_csv(r"dataset/train-0.csv",header=None,skiprows=1)

train\_df= train\_df.sample(frac=1)

val\_df = pd.read\_csv(r"dataset/file-0.csv", header=None, skiprows=1)

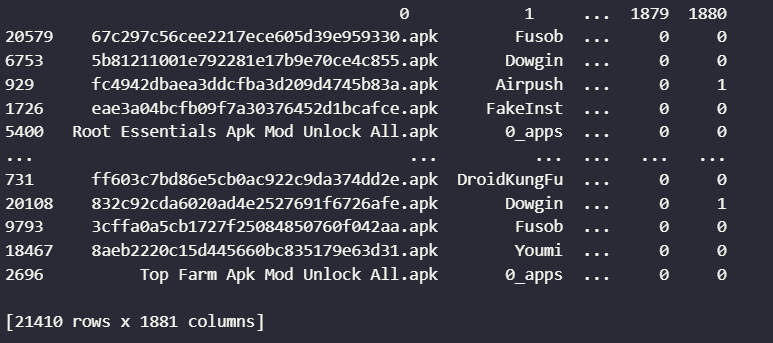
val\_df = val\_df.sample(frac=1)

test\_df = pd.read\_csv(r"dataset/file-1.csv",header=None,skiprows=1)

test\_df = test\_df.sample(frac=1)

print(train\_df)

Kết quả sau khi nạp dữ liệu ta sẽ có bộ dữ liệu dùng để huấn luyện bao gồm 21410 bản ghi với 1881 cột chứa thông tin.



Hình 1.20: Thông tin dữ liệu huấn luyện sau khi nạp vào chương trình

Tiếp đó, ta tiếp tục tử dụng thư viện *pandas* để lọc các đặc trưng và nhãn từ dữ liệu đã được nào vào, đồng thời thêm padding vào dữ liệu chứa đặc trưng nhằm chuẩn hóa dữ liệu thành một ma trận vuông có kích thước 44x44.

train\_y = np.array(train\_df.iloc[:, 2]) #lấy cột 2 là cột nhãn

train\_x = np.array(train\_df.iloc[:,3:]) #lấy cột 3 làm đặc trưng

#thêm padding các số 0 vào để tạo thành ma trạn vuông.

train\_x = np.concatenate((train\_x,np.zeros((train\_x.shape[0],28))),1)

unique, counts = np.unique(train\_y, return\_counts=True)

print("LABEL TRAIN: ", len(unique))

val\_y = np.array(val\_df.iloc[:, 2]) #lấy cột 2 là cột nhãn

val\_x = np.array(val\_df.iloc[:,3:]) #lấy cột 3 làm đặc trưng

#thêm padding các số 0 vào để tạo thành ma trạn vuông.

val\_x = np.concatenate((val\_x, np.zeros((val\_x.shape[0], 28))), 1)

unique\_v, counts\_ = np.unique(val\_y, return\_counts=True)

print("LABEL VAL: ", len(unique\_v))

test\_y = np.array(test\_df.iloc[:, 2]) #lấy cột 2 là cột nhãn

test\_x = np.array(test\_df.iloc[:,3:]) #lấy cột 3 làm đặc trưng

#thêm padding các số 0 vào để tạo thành ma trạn vuông.

test\_x = np.concatenate((test\_x, np.zeros((test\_x.shape[0],28))), 1)

unique\_t, counts\_t = np.unique(test\_y, return\_counts=True)

print("LABEL TEST: ", len(unique\_t))

Sau đó, ta khởi tạo các hằng số của mô hình.

BATCH\_SIZE = 32

SIZE = 44 # kích thước ma trận(44x44)

N\_CLASSES = 228 # số lượng kết quả đầu ra

LR = 0.001

N\_EPOCHS = 15 # Số vòng lặp huấn luyện

Tiếp theo, biến đổi dữ liệu ma trận về dạng ma trận phù hợp để chuẩn bị huấn luyện mô hình.

train\_x = train\_x.reshape(train\_x.shape[0],SIZE, SIZE, 1)

val\_x = val\_x.reshape(val\_x.shape[0],SIZE, SIZE, 1)

test\_x = test\_x.reshape(test\_x.shape[0],SIZE, SIZE, 1)

LABELS = np.unique(train\_y)

original\_test\_y = test\_y

train\_y = to\_categorical(train\_y, N\_CLASSES)

val\_y = to\_categorical(val\_y, N\_CLASSES)

test\_y = to\_categorical(test\_y, N\_CLASSES)

Sau khi đã chuẩn bị xong các thông số cũng như dữ liệu huấn luyện cho mô hình, ta tiến hành xây dựng mô hình để huấn luyện. Đầu tiên, ta khởi tạo một mô hình CNN.

input0 = Input(shape=(SIZE,SIZE,1))

conv1 = Conv2D(

filter=32, kernel\_size=2, activation='relu', padding="same",

input\_shape=(SIZE, SIZE, 1)

)(input0)

pool1 = MaxPooling2D(pool\_size=(2, 2)) (conv1)

conv2 = Conv2D(

filter=32, kernel\_size=2, activation='relu', padding="same"

)(pool1)

pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

conv3 = Conv2D(

filter=64,kernel\_size=2,activation='relu',padding="same"

)(pool2)

pool3 = MaxPooling2D(pool\_size=(2, 2))(conv3)

flatten\_per = Flatten()(pool3)

hidden1 = Dense(1024, activation='relu') (flatten\_per)

output = Dense(N\_CLASSES, activation='softmax') (hidden1)

model = Model(inputs=input0, outputs=output)

model.compile(

loss='categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy']

)

Ở đây, ta sử dụng 3 lớp với mỗi lớp kết hợp của Conv2D và MaxPooling2D. Sau đó được làm phẳng với lớp Flatten và lọc lấy 228 kết quả có xác xuất cao nhất thông qua thuật toán softmax. Sau khi đã có mô hình, ta khởi tạo một callback được sử dụng để xử lý sự kiện khi kết thúc huấn luyện mô hình.

class Metrics(Callback):

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y if (y.ndim == 1 or y.shape[1] == 1) else np.argmax(y, axis=1)

self.reports = []

def on\_epoch\_end(self, epoch, logs={}):

y\_hat = np.asarray(self.model.predict(self.x))

y\_hat = np.where(y\_hat > 0.5, 1, 0) if (y\_hat.ndim == 1 or y\_hat.shape[1] == 1) else np.argmax(y\_hat, axis=1)

report = classification\_report(self.y,y\_hat,output\_dict=True)

self.reports.append(report)

return

# Utility method

def get(self, metrics, of\_class):

return [report[str(of\_class)][metrics] for report in self.reports]

Kế tiếp, ta tiến hành huấn luyện mô hình.

metrics\_multiclass = Metrics(train\_x, train\_y)

history = model.fit(

x=train\_x,

y=train\_y,

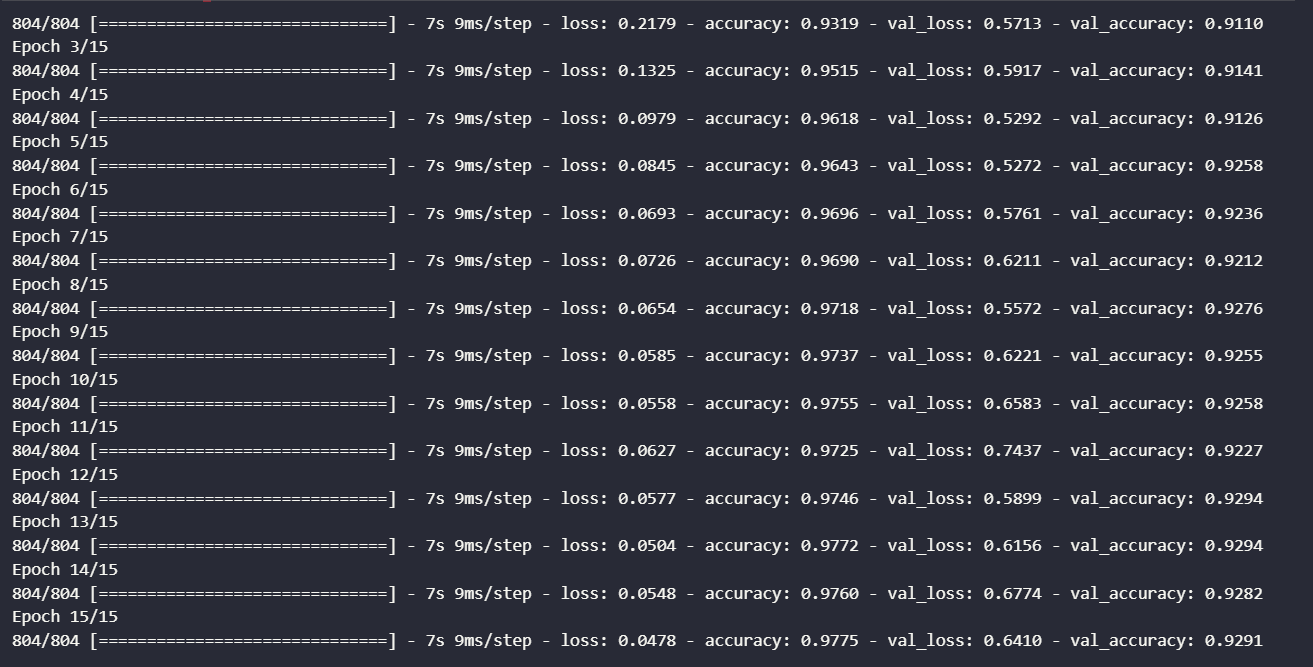
epochs=N\_EPOCHS,

batch\_size = BATCH\_SIZE ,

validation\_data=(val\_x, val\_y),

callbacks=[metrics\_multiclass]

)



Hình 1.21: Thông tin mô hình trong quá trình huấn luyện

Sau khi huấn luyện mô hình hoàn tất, ta hiển thị thông số độ chính xác và độ mất mát dữ liệu của mô hình thông qua sử dụng thư viện matplotlib.

acc = history.history['accuracy']

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

loss=history.history['loss']

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

epochs\_range = range(N\_EPOCHS)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

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

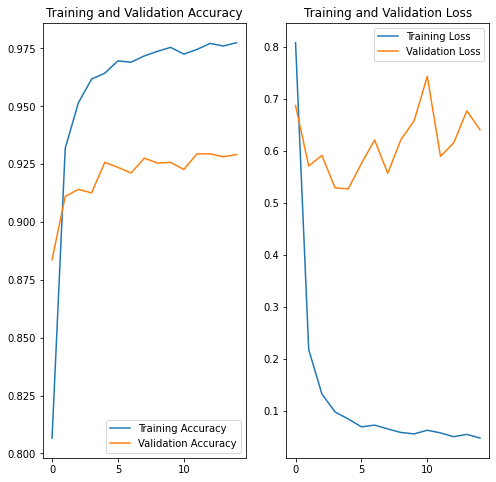
plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

plt.savefig('/content/drive/My Drive/test.png')

plt.close('all')



Hình 1.22: Biểu đồ độ chính xác và độ mất mát của mô hình

Tiếp theo, ta lưu mô hình đã được huấn luyện dưới dạng file .h5, sau đó chuyển đổi file chứa mô hình đã lưu sang định dạng TensorFlow Lite thông qua hàm *tf.lite.TFLiteConverter.from\_keras\_model* của thư viện TensorFlow Lite, hoàn tất quá trình huấn luyện mô hình.

from tf.keras.models import load\_model

from tf.lite.TFLiteConverter import from\_keras\_model

h5\_model\_path = 'out/models/model.h5'

h5\_model = load\_model(model)

converter = from\_keras\_model(h5\_model)

tflite\_model = converter.convert()

open("AMDDrebinModel.tflite", "wb").write(tflite\_model)

Cuối cùng, ta tiến hành đánh giá độ chính xác của mô hình dựa trên dữ liệu kiểm tra đã nạp trước đó.

tmp = list(dict.fromkeys(original\_test\_y))

y\_pred1 = model.predict(test\_x)

y\_pred = np.argmax(y\_pred1, axis=1)

# Print f1, precision, and recall scores

cnf\_matrix=confusion\_matrix(original\_test\_y, y\_pred, labels=tmp, normalize='true')

plt.close('all')

def plot\_confusion\_matrix(cm, classes,

title='Confusion matrix',

cmap=plt.cm.Blues):

plt.figure(figsize=(20, 20))

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=tmp, title='Normalized confusion matrix')

plt.show()

plt.close('all')

TÀI LIỆU THAM KHẢO

1. Nguyễn Anh Tuấn, “Deep learning cơ bản”, <https://drive.google.com/file/d/1lNjzISABdoc7SRq8tg-xkCRRZRABPCKi/view>
2. Deep Learning based Vehicle Detection in Aerial, <https://library.oapen.org//bitstream/handle/20.500.12657/53149/9783731511137.pdf?sequence=1&isAllowed=y>
3. <https://viblo.asia/p/generative-adversarial-networksgan-va-ung-dung-cua-no-trong-deepfakes-jvElagwdKkw>
4. <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>
5. <https://d2l.ai/chapter_recurrent-modern/lstm.html>
6. Generative Adversarial Nets (neurips.cc), <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>
7. MNIST Reborn, Restored and Expanded: Additional 50K Training Samples | Synced (syncedreview.com), <https://syncedreview.com/2019/06/19/mnist-reborn-restored-and-expanded-additional-50k-training-samples/>
8. <https://machinelearningcoban.com/2017/04/09/smv/>