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

BÁO CÁO CHUYÊN ĐỀ SỐ 4.6.1

“Mô hình Wide & Deep CNN cải tiến mô hình CNN truyền thống”

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 4.6.1

“Mô hình Wide & Deep CNN cải tiến mô hình CNN truyền thống”

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](#_Toc129698961)

[DANH MỤC HÌNH ẢNH 1](#_Toc129698962)

[MÔ HÌNH WIDE&DEEP CNN CẢI TIẾN MÔ HÌNH CNN TRUYỀN THỐNG 2](#_Toc129698963)

[1.1. Mô hình toán học sử dụng Wide&Deep CNN 2](#_Toc129698964)

[1.2. Phân hoạch dữ liệu và triển khai mô hình Wide&Deep cho bài toán phát hiện mã độc 3](#_Toc129698965)

[1.2.1. Phân hoạch dữ liệu 3](#_Toc129698966)

[1.2.2. Xây dựng mô hình Wide&DeepCNN cho bài toán 6](#_Toc129698967)

[1.3. Code mô hình Wide&Deep CNN 7](#_Toc129698968)

# DANH MỤC HÌNH ẢNH

[Hình 1: Sơ đồ hoạt động mô hình wide&deepCNN 5](file:///D:\DT_Ban%202022\bản%20word\Báo%20cáo%20tiến%20độ\Nội%20dung%204\BC_CĐ%204.6.1.%20Mô%20hình%20Wide%20and%20Deep%20CNN%20cải%20tiến%20mô%20hình%20CNN%20truyền%20thống.docx#_Toc129698955)

[Hình 2: Cấu trú và tham số của mô hình WDCNN 6](file:///D:\DT_Ban%202022\bản%20word\Báo%20cáo%20tiến%20độ\Nội%20dung%204\BC_CĐ%204.6.1.%20Mô%20hình%20Wide%20and%20Deep%20CNN%20cải%20tiến%20mô%20hình%20CNN%20truyền%20thống.docx#_Toc129698956)

# MÔ HÌNH WIDE&DEEP CNN CẢI TIẾN MÔ HÌNH CNN TRUYỀN THỐNG

## Mô hình toán học sử dụng Wide&Deep CNN

Trong phần này, chúng tôi xây dựng mô hình toán học tổng thể của bài toán để phát triển phương pháp cũng như làm tiền đề để mở rộng, cải tiến trong các nghiên cứu tiếp theo. Mô hình học sâu-rộng gồm hai thành phần là học rộng và học sâu. Mô hình này nhằm kết hợp khả năng phân lớp nhanh của học rộng và khả năng khái quát hoá đặc trưng của học sâu. Do đó, mỗi tập đặc trưng đầu vào sẽ được tách thành hai tập con tương ứng. Phần học sâu cũng có thể sử dụng nhiều CNN. Mỗi CNN cũng có thể dùng cả các lớp tích chập và lớp kết nối đầy đủ để khái quát hoá đặc trưng, giảm số chiều hoặc chỉ sử dụng các lớp tích chập (bao gồm cả tích chập và lọc). Đồng thời, việc phân hoạch tập đặc trưng sâu, rộng cũng cần được thực hiện trong phương pháp. Để xây dựng mô hình toán học, đầu tiên, chúng tôi đưa ra định nghĩa sau:

**Định nghĩa 1 – Tập đặc trưng ban đầu**

Tập đặc trưng ban đầu, ký hiệu là F0, là tập chứa mọi đặc trưng được đưa vào mô hình phân lớp.

**Định nghĩa 2 – Tập đặc trưng rộng**

Tập đặc trưng rộng, ký hiệu là Fw, là tập con của F0, được sử dụng cho thành phần học rộng trong mô hình học rộng-sâu.

**Định nghĩa 3 – Tập đặc trưng sâu**

Tập đặc trưng sâu, ký hiệu là Fd, là tập con của F0, được sử dụng để khái quát hoá đặc trưng trong thành phần học sâu của mô hình học rộng-sâu.

Trên cơ sở các định nghĩa trên, chúng tôi xây dựng mô hình toán học tổng thể của bài toán như trong công thức (1).

|  |  |
| --- | --- |
|  | (1) |

Trong đó,

* L là tập nhãn, bao gồm nhãn sạch và các nhãn mã độc
* là hàm phân hoạch, để chia tập *F0* thành *Fd* và *Fw*
* l à ánh xạ từ tập đặc trưng rộng sang vector
* là ánh xạ từ tập đặc trưng rộng sang vector
* là vector tổng hợp cho học rộng-sâu

## Phân hoạch dữ liệu và triển khai mô hình Wide&Deep cho bài toán phát hiện mã độc

### Phân hoạch dữ liệu

Để đánh giá và phân hoạch tập đặc trưng ban đầu thành tập đặc trưng sâu và tập đặc trưng rộng, chúng tôi đưa ra các định nghĩa sau:

**Định nghĩa 4 – Đặc trưng thô**

Đặc trưng thô, ký hiệu là r, là đặc trưng không thể hiện được hoặc không mang đầy đủ nghĩa về một hành vi, thao tác hay thuộc tính của mã độc. Ví dụ, một byte trong file .dex, một pixel.

**Định nghĩa 5 – Đặc trưng tổng quát**

Đặc trưng tổng quát, ký hiệu là a, là đặc trưng thể hiện được một hành vi, thao tác hay một thuộc tính của mã độc. Ví dụ, một quyền (permission) hoặc một API.

**Định nghĩa 6 – Đặc trưng khái quát mức nhóm**

Đặc trưng tổng quát mức nhóm, ký hiệu là g, là đặc trưng đại diện cho một nhóm hành vi, thao tác hay một nhóm thuộc tính của mã độc. Ví dụ, nhóm quyền truy xuất bộ nhớ, nhóm API thao tác file.

Một vấn đề quan trọng cần giải quyết trong phương pháp đề xuất là việc phân chia tập F thành Fd và Fw. Căn cứ vào ba định nghĩa trên, các đặc trưng thô được đưa vào tập Fd vì một đặc trưng thô thường không có ý nghĩa đầy đủ nên cần được tổng quát hoá để tạo thành đặc trưng khái quát hoặc đặc trưng khái quát mức nhóm để giảm số chiều. Các đặc trưng khái quát mức nhóm thường được đưa vào Fw vì các đặc trưng này đã mang ý nghĩa và tính chất khái quát của mã độc. Tuy nhiên, khi hệ thống có tập đặc trưng khái quát mức nhóm lớn, vẫn có thể đưa vào Fd để giảm bớt số chiều. Tuỳ theo ngữ cảnh bài toán và mức độ tổng quát mà các đặc trưng tổng quát và các đặc trưng mức nhóm có thể được đưa vào Fd hoặc Fw.

**Phân hoạch tập đặc trưng**

Một phân hoạch là một phép phân chia tập đặc trưng ban đầu thành tập đặc trưng rộng và tập đặc trưng sâu phù hợp với ngữ cảnh bài toán và đặc điểm của tập đặc trưng. Để thực hiện phân hoạch, chúng tôi xây dựng thuật toán như sau.

**Thuật toán 1 – Thuật toán phân hoạch**

**Input:**

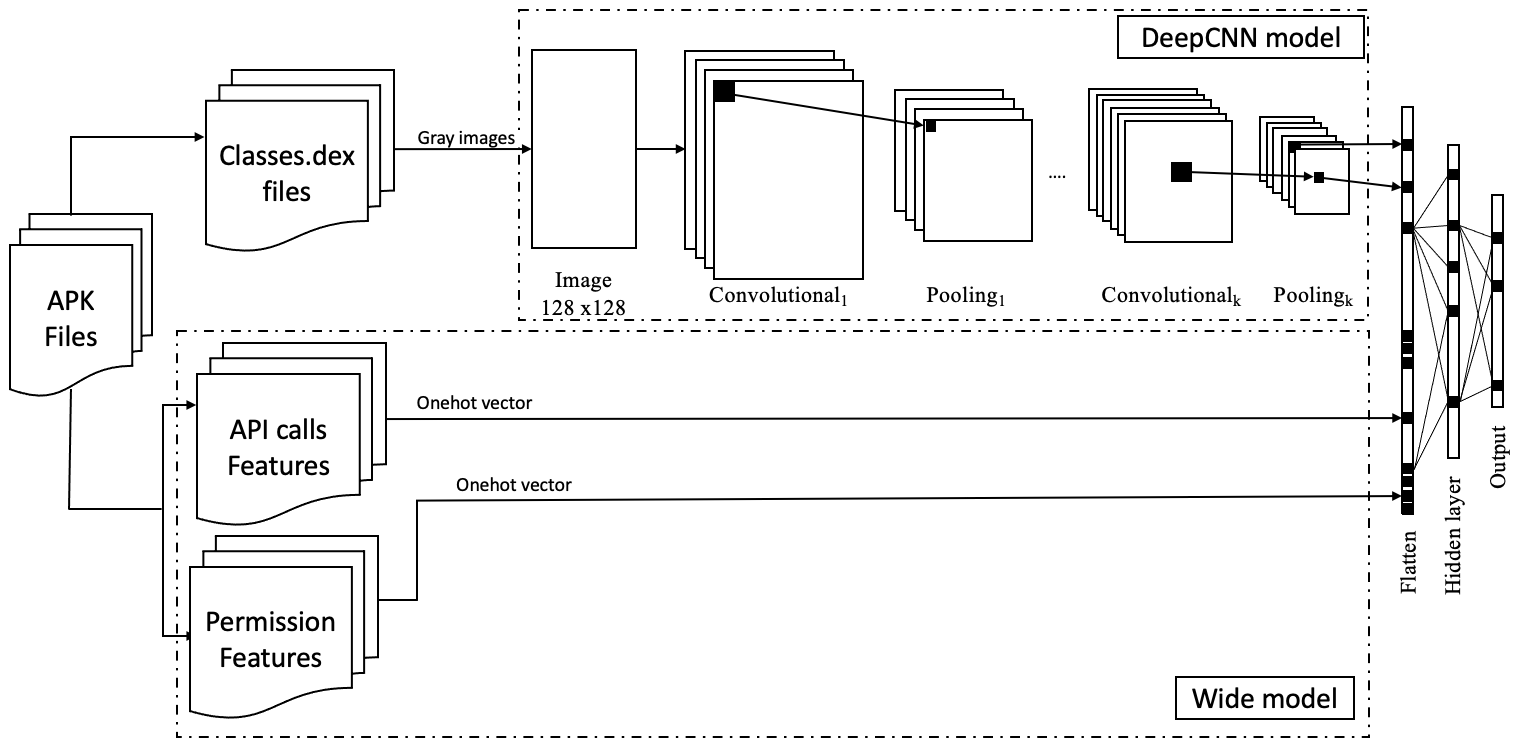
* F0: tập đặc trưng ban đầu
* A: Tập hành vi/thao tác/thuộc tính
* G: Tập nhóm hành vi/thao tác/thuộc tính
* R: Tập luật (rule) để phân chia

**Output:** Fw, Fd

**Thuật toán:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Duyệt mỗi đặc trưng fi thuộc F0  If fi thoả mãn một trong các hành vi/thao tác/thuộc tính của G  Đưa fi vào tập đặc trưng khái quát mức nhóm  Else, if fi thoả mãn A  Đưa fi vào tập đặc trưng khái quát  Else  Đưa fi vào tập đặc trưng thô  Gán Fd bằng tập đặc trưng thô  Duyệt với mỗi đặc trưng thuộc tập đặc trưng khái quát  If thoả mãn R thì đưa vào Fw  Else, đưa vào Fd  Duyệt với mỗi đặc trưng thuộc tập đặc trưng khái quát mức nhóm  If thoả mãn R thì đưa vào Fw  Else, đưa vào Fd |

Để cụ thể hoá mô hình toán học tổng thể, trong phạm vi nghiên cứu của bài báo, chúng tôi tập trung vào tập đặc trưng ban đầu gồm ba tập đặc trưng con: tập quyền hạn, tập API và file ảnh tương ứng với phần code trong file \*.dex. Theo mô tả trên, tập các pixel trong file ảnh chính là tập đặc trưng thô vì pixel không có ý nghĩa đầy đủ cho một hành vi, thao tác hay thuộc tính của mã độc. Tập các quyền và tập các API chính là hai tập đặc trưng khái quát vì mỗi đặc trưng thể hiện được một hành vi hoặc thao tác của mã độc. Tình huống nghiên cứu trong bài báo được thực hiện trên 2 tập đặc trưng trừu tượng và 1 tập đặc trưng thô; với luật phân hoạch đơn giản là các đặc trưng trừu tượng đựa được vào thành phần rộng và các đặc trưng thô được đưa vào thành phần học sâu. Thực hiện Thuật toán 1 trong phạm vi bài báo, chúng tôi lựa chọn Fd là tập đặc trưng thô và Fw là tập mọi đặc trưng khái quát gồm permision và API. Trên cơ sở phân chia tập đặc trưng này, mô hình wide & deep trong nghiên cứu được mô tả trong phần tiếp theo.

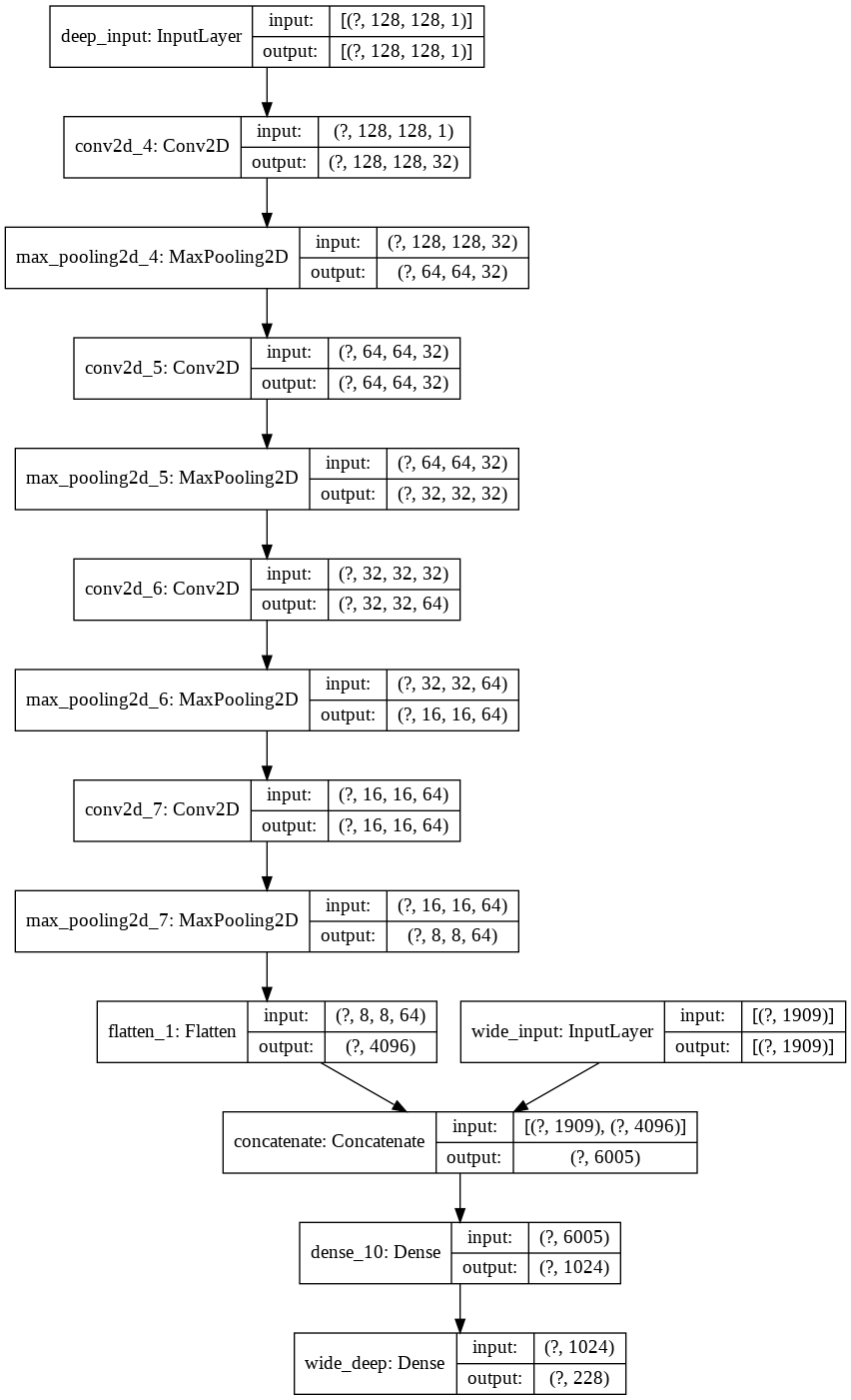


Hình 1: Sơ đồ hoạt động mô hình wide&deepCNN

### Xây dựng mô hình Wide&DeepCNN cho bài toán

Trong mô hình wide&DeepCNN có hai thành phần phần chính:

* DeepCNN model: Bản chất là mạng CNN, bên trong là các lớp convolutional và pooling được kết hợp với nhau để huấn luyện các đặc trưng đầu vào. CNN có thể là 1CNN khi sử dụng cho một nhóm đặc trưng như hình 1, cũng có thể là 2CNN khi áp dụng cho hai nhóm đặc trưng khác nhau và các thông số trong CNN cũng thiết lập khác nhau.



Hình 2: Cấu trú và tham số của mô hình WDCNN

* Wide model: Là các đặc trưng bổ sung cho mô hình wide&deepCNN. Trong Hình 1 chúng tôi bổ sung thêm các đặc trưng thô (raw features) là toàn bộ Permission và rank(1000) API được sử dụng nhiều nhất trong các file APK. Do khi đưa các đặc trưng vào mô hình DeepCNN có thể trong quá trình chuẩn hoá đầu vào (từ dạng vector sang dạng ma trận) các đăc trưng không tuân theo thứ tự [bài tạp chí bách khoa] sẽ ảnh hưởng tới việc kết hợp các đặc trưng đó trong các lớp convolutional và pooling. Như vậy, các đặc trưng trong mô hình wide để thêm các thông tin cho mô hình.

Để có thể rõ hơn, ta có thể xem model Wide&Deep như Hình 2.

## Code mô hình Wide&Deep CNN

# Khai báo những thư viện cần thiết

import numpy as np

import pandas as pd

import tensorflow as tf

from keras.layers import Input, Embedding, Dense, Flatten, Activation, Dropout, concatenate

from keras.layers.advanced\_activations import ReLU

from keras.layers.normalization import BatchNormalization

from keras.models import Model

from keras.utils import to\_categorical, plot\_model

from sklearn.preprocessing import PolynomialFeatures

import matplotlib.pyplot as plt

import time

print("Bắt đầu.")

# Đường dẫn đến 2 file train và test

TRAIN\_FILE\_PATH = r'/content/drive/My Drive/Colab Notebooks/10set (đủ mã)/AMD\_Benign\_ver2(tong hop)/train-0.csv'

TEST\_FILE\_PATH = r'/content/drive/My Drive/Colab Notebooks/10set (đủ mã)/AMD\_Benign\_ver2(tong hop)/file-0.csv'

VAL\_FILE\_PATH = r'/content/drive/My Drive/Colab Notebooks/10set (đủ mã)/AMD\_Benign\_ver2(tong hop)/file-1.csv'

#FILE\_TOTAL\_PATH = r'F:\Thay Thuan\data\image128\_per\_api\_drebin\_benign\Drebin\_Benign\_128(Img\_Per\_API)\_ver3.csv'

#tt\_data = pd.read\_csv(FILE\_TOTAL\_PATH, header=None, skiprows = 1) #Test thử cùng với row 48 để chạy đủ file total

# Khai báo những tham số cần thiết

SIZE = 128

N\_CLASSES = 228

N\_EPOCHS = 50

N\_BATCH\_SIZE = 32

N\_PERMISSION\_COLUMNS = 877 #drebin 829 AMD 879 và all 908

N\_API\_COLUMNS = 1000

N\_IMAGE\_COLUMNS = 16384

# Đọc dữ liệu train và test

train\_data = pd.read\_csv(TRAIN\_FILE\_PATH, header=None, skiprows = 1)

#train\_data=train\_data.sample(100)

train\_data.dropna(how='any', axis=0)

test\_data = pd.read\_csv(TEST\_FILE\_PATH, header=None, skiprows = 1)

#test\_data=test\_data.sample(100)

test\_data.dropna(how='any', axis=0)

val\_data = pd.read\_csv(VAL\_FILE\_PATH,header=None, skiprows = 1)

#val\_data=val\_data.sample(100)

val\_data.dropna(how='any', axis=0)

# #test\_data = np.concatenate((test\_data,val\_data))

# train\_data = pd.concat([train\_data, val\_data])

# #train\_data = pd.concat([train\_data, test\_data])

# # Tổng số cột của một dòng

# cols = train\_data.columns.values

# #train\_data=tt\_data.sample(frac = 0.9)

# #test\_data=tt\_data.sample(frac = 0.1) #Test thử cùng với row 20,21 để chạy đủ file total

# print("ab=",test\_data.shape)

# # Cột đầu tiên là label, tên cột là '2'.

# # Những cột còn lại đóng vai trò là feature

# LABEL\_COLUMN = cols[2]

# FEATURE\_COLUMNS = cols[3:]

# print("feature\_columns = ", FEATURE\_COLUMNS.shape) #Tính tổng số đặc trưng trong file csv

# # Số lượng cột Permission feature được lấy từ cột thứ 0 đến cột N\_PERMISSION\_COLUMNS-1

# # Số lượng cột API feature được lấy từ cột thứ N\_PERMISSION\_COLUMNS đến hết.

# #TH1: sử dụng cho file có các giá trị ảnh

# IMAGE\_COLUMNS = FEATURE\_COLUMNS[0: N\_IMAGE\_COLUMNS]

# PERMISSION\_COLUMNS = FEATURE\_COLUMNS[N\_IMAGE\_COLUMNS: N\_IMAGE\_COLUMNS+N\_PERMISSION\_COLUMNS]

# API\_COLUMNS = FEATURE\_COLUMNS[N\_IMAGE\_COLUMNS+N\_PERMISSION\_COLUMNS:]

# print('PERMISSION\_COLUMNS = ', PERMISSION\_COLUMNS)

# print('API\_COLUMNS = ', API\_COLUMNS)

# print('IMAGE\_COLUMNS = ', IMAGE\_COLUMNS)

# '''

# #TH2: Sử dụng cho các file không có giá trị ảnh

# #IMAGE\_COLUMNS = FEATURE\_COLUMNS[0: N\_IMAGE\_COLUMNS]

# PERMISSION\_COLUMNS = FEATURE\_COLUMNS[:N\_PERMISSION\_COLUMNS]

# API\_COLUMNS = FEATURE\_COLUMNS[N\_PERMISSION\_COLUMNS:N\_PERMISSION\_COLUMNS+1833]

# print("PERMISSION\_COLUMNS: ",PERMISSION\_COLUMNS.shape)

# print("API\_COLUMNS: ",API\_COLUMNS.shape)

# # In ra thông tin các cột đã lấy

# #print('PERMISSION\_COLUMNS = ', PERMISSION\_COLUMNS)

# #print('API\_COLUMNS = ', API\_COLUMNS)

# '''

# # Hàm này để xử lý data trước khi thực hiện các bước sau

# def preprocessing():

# all\_data = pd.concat([train\_data, test\_data])

# # Cột đầu tiên (cột số 2) được định nghĩa là label

# all\_data[LABEL\_COLUMN] = all\_data[LABEL\_COLUMN].astype(int)

# y = all\_data[LABEL\_COLUMN].values

# print("y:",y)

# all\_data.pop(0) # remove file name

# all\_data.pop(1) # remove label name

# all\_data.pop(LABEL\_COLUMN) # remove label id

# print("gia tri lon nhan cua nhan:", max(y))

# # Chia data train và data test từ data đã đưa vào

# train\_size = len(train\_data)

# x\_train = all\_data.iloc[:train\_size]

# y\_train = y[:train\_size]

# x\_test = all\_data.iloc[train\_size:]

# y\_test = y[train\_size:]

# print(x\_train.shape)

# # Chia tập x\_train, x\_test thành 2 tập con là per và api

# x\_train\_per = np.array(x\_train[PERMISSION\_COLUMNS])

# x\_train\_api = np.array(x\_train[API\_COLUMNS])

# x\_test\_per = np.array(x\_test[PERMISSION\_COLUMNS])

# x\_test\_api = np.array(x\_test[API\_COLUMNS])

# print("train per:", x\_train\_per.shape)

# #Đầu vào dành cho wide

# x\_train\_per\_w = np.array(x\_train[N\_IMAGE\_COLUMNS])

# x\_train\_api\_w = np.array(x\_train[PERMISSION\_COLUMNS+API\_COLUMNS])

# x\_test\_per\_w = np.array(x\_test[N\_IMAGE\_COLUMNS])

# x\_test\_api\_w = np.array(x\_test[PERMISSION\_COLUMNS+API\_COLUMNS])

# print("train per for wide:", x\_train\_per\_w.shape)

# return (x\_train, y\_train, x\_test, y\_test, x\_train\_per, x\_test\_per, x\_train\_api, x\_test\_api, x\_train\_per\_w, x\_test\_per\_w, x\_train\_api\_w, x\_test\_api\_w, all\_data)

# def plot\_graphs(history):

# plt.plot(history.history['accuracy'])

# plt.xlabel("Epochs")

# plt.ylabel('accuracy')

# plt.legend(['accuracy'])

# plt.savefig('plot\_graphs.png')

# plt.close()

# class Wide\_and\_Deep:

# # Định nghĩa class Wide\_and\_Deep và khai báo, khởi tạo những property cần thiết

# def \_\_init\_\_(self, mode='wide and deep'):

# self.mode = mode

# x\_train, y\_train, x\_test, y\_test, x\_train\_per, x\_test\_per, x\_train\_api, x\_test\_api, x\_train\_per\_w, x\_test\_per\_w, x\_train\_api\_w, x\_test\_api\_w, all\_data = preprocessing()

# self.x\_train = x\_train

# self.y\_train = y\_train

# self.x\_test = x\_test

# self.y\_test = y\_test

# self.x\_train\_per = x\_train\_per

# self.x\_test\_per = x\_test\_per

# self.x\_train\_api = x\_train\_api

# self.x\_test\_api = x\_test\_api

# self.x\_train\_per\_w = x\_train\_per\_w

# self.x\_test\_per\_w = x\_test\_per\_w

# self.x\_train\_api\_w = x\_train\_api\_w

# self.x\_test\_api\_w = x\_test\_api\_w

# self.all\_data = all\_data

# self.poly = PolynomialFeatures(degree=2, interaction\_only=True)

# '''self.x\_train\_per\_poly = self.poly.fit\_transform(x\_train\_per)

# self.x\_test\_per\_poly = self.poly.transform(x\_test\_per)

# self.x\_train\_api\_poly = self.poly.fit\_transform(x\_train\_api)

# self.x\_test\_api\_poly = self.poly.transform(x\_test\_api)

# '''

# self.per\_input = None

# self.api\_input = None

# self.deep\_component\_outlayer = None

# self.logistic\_input = None

# self.logistic\_per\_input = None

# self.logistic\_api\_input = None

# # self.logistic\_api\_input = None

# self.model = None

# self.history = None

# # Hàm này để tạo deep model

# def deep\_component(self):

# per\_inputs = []

# api\_inputs = []

# per\_embeds = []

# api\_embeds = []

# dims\_per = []

# dims\_api = []

# for i in range(len(PERMISSION\_COLUMNS)):

# input\_i = Input(shape=(1,), dtype='int32')

# dim = len(np.unique(self.all\_data[PERMISSION\_COLUMNS[i]]))

# embed\_dim = int(np.ceil(dim \*\* 0.25))

# dims\_per.append(embed\_dim)

# embed\_i = Embedding(dim, embed\_dim, input\_length=1)(input\_i)

# flatten\_i = Flatten()(embed\_i)

# per\_inputs.append(input\_i)

# per\_embeds.append(flatten\_i)

# for i in range(len(API\_COLUMNS)):

# input\_i = Input(shape=(1,), dtype='int32')

# dim = len(np.unique(self.all\_data[API\_COLUMNS[i]]))

# embed\_dim = int(np.ceil(dim \*\* 0.25))

# dims\_api.append(embed\_dim)

# embed\_i = Embedding(dim, embed\_dim, input\_length=1)(input\_i)

# flatten\_i = Flatten()(embed\_i)

# api\_inputs.append(input\_i)

# api\_embeds.append(flatten\_i)

# print(per\_embeds)

# print(api\_embeds)

# print(dims\_per)

# print(dims\_api)

# # Đưa input per vào model

# deep\_per\_input = Input(shape=(N\_PERMISSION\_COLUMNS,))

# deep\_per\_dense = Dense(128, use\_bias=False)(deep\_per\_input)

# # Đưa input api vào model

# deep\_api\_input = Input(shape=(N\_API\_COLUMNS,))

# deep\_api\_dense = Dense(128, use\_bias=False)(deep\_api\_input)

# # Sau khi các input đã đi qua Full connected thì merge lại với nhau

# # concat\_embeds = concatenate(per\_embeds + api\_embeds)

# concat\_embeds = concatenate([deep\_per\_dense] + [deep\_api\_dense])

# concat\_embeds = Activation('relu')(concat\_embeds)

# bn\_concat = BatchNormalization()(concat\_embeds)

# # Tiến hành cho qua những lớp ẩn

# fc1 = Dense(2048, use\_bias=False)(bn\_concat)

# ac1 = ReLU()(fc1)

# bn1 = BatchNormalization()(ac1)

# fc2 = Dense(1024, use\_bias=False)(bn1)

# ac2 = ReLU()(fc2)

# bn2 = BatchNormalization()(ac2)

# fc3 = Dense(512, use\_bias=False)(bn2)

# ac3 = ReLU()(fc3)

# bn3 = BatchNormalization()(ac3)

# fc4 = Dense(256)(bn3)

# dropout = Dropout(0.25)

# ac4 = ReLU()(fc4)

# self.per\_input = deep\_per\_input

# self.api\_input = deep\_api\_input

# # Đầu ra cuối cùng của deep model

# self.deep\_component\_outlayer = ac4

# print('Finish create deep model')

# # Hàm này để tạo wide model

# def wide\_model(self):

# # Đưa input per vào model

# dim\_per = self.x\_train\_per\_w.shape[1]

# print("x\_train\_per\_w = ", self.x\_train\_per\_w.shape)

# print("x\_train\_per = ", self.x\_train\_per.shape)

# self.logistic\_per\_input = Input(shape=(dim\_per,))

# print("logistic\_per\_input = ",self.logistic\_per\_input.shape)

# # Đưa input api vào model

# dim\_api = self.x\_train\_api\_w.shape[1]

# print("x\_train\_api\_w = ", self.x\_train\_api\_w.shape)

# self.logistic\_api\_input = Input(shape=(dim\_api,))

# print("logistic\_api\_input = ",self.logistic\_api\_input.shape)

# # Hàm này thực hiện merge wide & deep model lại với nhau

# def create\_model(self):

# print('Starting create model ...')

# self.deep\_component()

# self.wide\_model()

# if self.mode == 'wide and deep':

# # Output layer của model cần phải có các thông tin output của từng model đã tạo

# out\_layer = concatenate([self.deep\_component\_outlayer, self.logistic\_per\_input, self.logistic\_api\_input])

# # Inputs cũng cần phải được đưa vào những thông tin đúng thứ tự đã định nghĩa của từng model

# # Nên xem summary để biết thứ tự đưa vào các input hoặc xem thứ tự gọi các lệnh input của từng model

# inputs = [self.per\_input] + [self.api\_input] + [self.logistic\_per\_input] + [self.logistic\_api\_input]

# elif self.mode == 'deep':

# out\_layer = self.deep\_component\_outlayer

# inputs = [self.per\_input] + [self.api\_input]

# else:

# print('wrong mode')

# return

# # Output cuối cùng của model

# output = Dense(N\_CLASSES, activation='softmax')(out\_layer)

# print(f'input = {len(inputs)}')

# # Tạo model với thông tin inputs và output đã định nghĩa

# self.model = Model(inputs=inputs, outputs=output)

# print(self.model.summary())

# # Hàm này dùng để train model, ứng thứ tự các input đã khai báo, cần đưa vào model đúng thứ tự tập input train

# def train\_model(self):

# if not self.model:

# print('You have to create model first')

# return

# if self.mode == 'wide and deep':

# print("chạy vào train wide and deep")

# input\_data = [self.x\_train\_per] + [self.x\_train\_api] + [self.x\_train\_per\_w] + [self.x\_train\_api\_w]

# print("xxxxxxxx")

# elif self.mode == 'deep':

# input\_data = [self.x\_train\_per] + [self.x\_train\_api]

# else:

# print('wrong mode')

# return

# # Do bài toán ở đây là multi classification nên loss phải là 'categorical\_crossentropy'

# self.model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# print("qua compile")

# # Chuyển đối số chiều của tập y\_train từ [len(y\_train), None] về [len(y\_train), N\_CLASSES]

# self.y\_train = to\_categorical(self.y\_train, N\_CLASSES)

# # Định nghĩa callbacks theo 'accuracy'

# es = tf.keras.callbacks.EarlyStopping(monitor='accuracy', mode='max')

# # callbacks = [es]

# callbacks = []

# # Trong quá trình train có thể sử dụng callbacks hoặc không.

# self.history = self.model.fit(input\_data, self.y\_train, epochs=N\_EPOCHS, batch\_size=N\_BATCH\_SIZE)

# # Hàm này dùng để đánh giá kết quả model đã train bằng tập test

# # Các tham số đưa vào được tính toán tương tự với khi train

# def evaluate\_model(self):

# if not self.model:

# print('You have to create model first')

# return

# if self.mode == 'wide and deep':

# print("chạy vào test wide and deep")

# input\_data = [self.x\_test\_per] + [self.x\_test\_api] + [self.x\_test\_per\_w] + [self.x\_test\_api\_w]

# elif self.mode == 'deep':

# input\_data = [self.x\_test\_per] + [self.x\_test\_api]

# else:

# print('wrong mode')

# return

# self.y\_test = to\_categorical(self.y\_test,N\_CLASSES)

# loss, acc = self.model.evaluate(input\_data, self.y\_test)

# print(f'\ntest\_loss: {loss} - test\_acc: {acc}')

# # Lưu lại model dưới dạng .h5 (keras output format) để sử dụng về sau

# def save\_model(self, filename='wide\_and\_deep.h5'):

# self.model.save(filename)

# if \_\_name\_\_ == '\_\_main\_\_':

# # Khai báo 1 đối tượng Wide\_and\_Deep() và sử dụng các hàm đã định nghĩa của Wide\_and\_Deep

# t0 = time.time()

# print("bắt đầu chạy winde and deep")

# wide\_deep\_net = Wide\_and\_Deep()

# t1 = time.time()

# print("chạy xong wide and deep")

# print("bắt đầu chạy create model")

# wide\_deep\_net.create\_model()

# print("chạy xong create model")

# print("bắt đầu chạy train model")

# wide\_deep\_net.train\_model()

# print("chạy xong train model")

# print("bắt đầu chạy evaluate\_model")

# wide\_deep\_net.evaluate\_model()

# print("chạy xong evaluate\_model")

# wide\_deep\_net.save\_model()

# #plot\_graphs(wide\_deep\_net.history)

# # Lưu lại thông tin của model dứa dạng file .png

# #plot\_model(wide\_deep\_net.model, to\_file='model.png', show\_shapes=True, show\_layer\_names=False)

# t2 = time.time()

# print("thoi gian chạy model không tính comatrix: %.2f phút." %((t2-t1)/60))

# print("thoi gian chạy xong model: %.2f phút." %((t2-t0)/60))

# print('Hoàn thành việc huấn luyện và kiểm tra!')

# In[ ]:

#test\_data = np.concatenate((test\_data,val\_data))

train\_data = pd.concat([train\_data, val\_data])

#train\_data = pd.concat([train\_data, test\_data])

# Tổng số cột của một dòng

cols = train\_data.columns.values

#train\_data=tt\_data.sample(frac = 0.9)

#test\_data=tt\_data.sample(frac = 0.1) #Test thử cùng với row 20,21 để chạy đủ file total

print("ab=",test\_data.shape)

# Cột đầu tiên là label, tên cột là '2'.

# Những cột còn lại đóng vai trò là feature

LABEL\_COLUMN = cols[2]

FEATURE\_COLUMNS = cols[3:]

print("feature\_columns = ", FEATURE\_COLUMNS.shape) #Tính tổng số đặc trưng trong file csv

# Số lượng cột Permission feature được lấy từ cột thứ 0 đến cột N\_PERMISSION\_COLUMNS-1

# Số lượng cột API feature được lấy từ cột thứ N\_PERMISSION\_COLUMNS đến hết.

#TH1: sử dụng cho file có các giá trị ảnh

IMAGE\_COLUMNS = FEATURE\_COLUMNS[0: N\_IMAGE\_COLUMNS]

PERMISSION\_COLUMNS = FEATURE\_COLUMNS[N\_IMAGE\_COLUMNS: ]

API\_COLUMNS = FEATURE\_COLUMNS[N\_IMAGE\_COLUMNS+N\_PERMISSION\_COLUMNS:]

print('PERMISSION\_COLUMNS = ', PERMISSION\_COLUMNS)

print('API\_COLUMNS = ', API\_COLUMNS)

print('IMAGE\_COLUMNS = ', IMAGE\_COLUMNS)

'''

#TH2: Sử dụng cho các file không có giá trị ảnh

#IMAGE\_COLUMNS = FEATURE\_COLUMNS[0: N\_IMAGE\_COLUMNS]

PERMISSION\_COLUMNS = FEATURE\_COLUMNS[:N\_PERMISSION\_COLUMNS]

API\_COLUMNS = FEATURE\_COLUMNS[N\_PERMISSION\_COLUMNS:N\_PERMISSION\_COLUMNS+1833]

print("PERMISSION\_COLUMNS: ",PERMISSION\_COLUMNS.shape)

print("API\_COLUMNS: ",API\_COLUMNS.shape)

# In ra thông tin các cột đã lấy

#print('PERMISSION\_COLUMNS = ', PERMISSION\_COLUMNS)

#print('API\_COLUMNS = ', API\_COLUMNS)

'''

# Hàm này để xử lý data trước khi thực hiện các bước sau

def preprocessing():

all\_data = pd.concat([train\_data, test\_data])

# Cột đầu tiên (cột số 2) được định nghĩa là label

all\_data[LABEL\_COLUMN] = all\_data[LABEL\_COLUMN].astype(int)

y = all\_data[LABEL\_COLUMN].values

print("y:",y)

all\_data.pop(0) # remove file name

all\_data.pop(1) # remove label name

all\_data.pop(LABEL\_COLUMN) # remove label id

print("gia tri lon nhan cua nhan:", max(y))

# Chia data train và data test từ data đã đưa vào

train\_size = len(train\_data)

x\_train = all\_data.iloc[:train\_size]

y\_train = y[:train\_size]

x\_test = all\_data.iloc[train\_size:]

y\_test = y[train\_size:]

print(x\_train.shape)

# Chia tập x\_train, x\_test thành 2 tập con là per và api

x\_train\_per = np.array(x\_train[PERMISSION\_COLUMNS])

x\_train\_api = np.array(x\_train[API\_COLUMNS])

x\_test\_per = np.array(x\_test[PERMISSION\_COLUMNS])

x\_test\_api = np.array(x\_test[API\_COLUMNS])

print("train per:", x\_train\_per.shape)

#Đầu vào dành cho wide

x\_train\_per\_w = np.array(x\_train[N\_IMAGE\_COLUMNS])

x\_train\_api\_w = np.array(x\_train[PERMISSION\_COLUMNS.tolist()+API\_COLUMNS.tolist()])

x\_test\_per\_w = np.array(x\_test[N\_IMAGE\_COLUMNS])

x\_test\_api\_w = np.array(x\_test[PERMISSION\_COLUMNS.tolist()+API\_COLUMNS.tolist()])

print("train per for wide:", x\_train\_per\_w.shape)

return (x\_train, y\_train, x\_test, y\_test, x\_train\_per, x\_test\_per, x\_train\_api, x\_test\_api, x\_train\_per\_w, x\_test\_per\_w, x\_train\_api\_w, x\_test\_api\_w, all\_data)

def plot\_graphs(history):

plt.plot(history.history['accuracy'])

plt.xlabel("Epochs")

plt.ylabel('accuracy')

plt.legend(['accuracy'])

plt.savefig('plot\_graphs.png')

plt.close()

class Wide\_and\_Deep:

# Định nghĩa class Wide\_and\_Deep và khai báo, khởi tạo những property cần thiết

def \_\_init\_\_(self, mode='wide and deep'):

self.mode = mode

x\_train, y\_train, x\_test, y\_test, x\_train\_per, x\_test\_per, x\_train\_api, x\_test\_api, x\_train\_per\_w, x\_test\_per\_w, x\_train\_api\_w, x\_test\_api\_w, all\_data = preprocessing()

self.x\_train = x\_train

self.y\_train = y\_train

self.x\_test = x\_test

self.y\_test = y\_test

self.x\_train\_per = x\_train\_per

self.x\_test\_per = x\_test\_per

self.x\_train\_api = x\_train\_api

self.x\_test\_api = x\_test\_api

self.x\_train\_per\_w = x\_train\_per\_w

self.x\_test\_per\_w = x\_test\_per\_w

self.x\_train\_api\_w = x\_train\_api\_w

self.x\_test\_api\_w = x\_test\_api\_w

self.all\_data = all\_data

self.poly = PolynomialFeatures(degree=2, interaction\_only=True)

'''self.x\_train\_per\_poly = self.poly.fit\_transform(x\_train\_per)

self.x\_test\_per\_poly = self.poly.transform(x\_test\_per)

self.x\_train\_api\_poly = self.poly.fit\_transform(x\_train\_api)

self.x\_test\_api\_poly = self.poly.transform(x\_test\_api)

'''

self.per\_input = None

self.api\_input = None

self.deep\_component\_outlayer = None

self.logistic\_input = None

self.logistic\_per\_input = None

self.logistic\_api\_input = None

# self.logistic\_api\_input = None

self.model = None

self.history = None

# Hàm này để tạo deep model

def deep\_component(self):

per\_inputs = []

api\_inputs = []

per\_embeds = []

api\_embeds = []

dims\_per = []

dims\_api = []

for i in range(len(PERMISSION\_COLUMNS)):

input\_i = Input(shape=(1,), dtype='int32')

dim = len(np.unique(self.all\_data[PERMISSION\_COLUMNS[i]]))

embed\_dim = int(np.ceil(dim \*\* 0.25))

dims\_per.append(embed\_dim)

embed\_i = Embedding(dim, embed\_dim, input\_length=1)(input\_i)

flatten\_i = Flatten()(embed\_i)

per\_inputs.append(input\_i)

per\_embeds.append(flatten\_i)

for i in range(len(API\_COLUMNS)):

input\_i = Input(shape=(1,), dtype='int32')

dim = len(np.unique(self.all\_data[API\_COLUMNS[i]]))

embed\_dim = int(np.ceil(dim \*\* 0.25))

dims\_api.append(embed\_dim)

embed\_i = Embedding(dim, embed\_dim, input\_length=1)(input\_i)

flatten\_i = Flatten()(embed\_i)

api\_inputs.append(input\_i)

api\_embeds.append(flatten\_i)

print(per\_embeds)

print(api\_embeds)

print(dims\_per)

print(dims\_api)

# Đưa input per vào model

deep\_per\_input = Input(shape=(N\_PERMISSION\_COLUMNS,))

deep\_per\_dense = Dense(128, use\_bias=False)(deep\_per\_input)

# Đưa input api vào model

deep\_api\_input = Input(shape=(N\_API\_COLUMNS,))

deep\_api\_dense = Dense(128, use\_bias=False)(deep\_api\_input)

# Sau khi các input đã đi qua Full connected thì merge lại với nhau

# concat\_embeds = concatenate(per\_embeds + api\_embeds)

concat\_embeds = concatenate([deep\_per\_dense] + [deep\_api\_dense])

concat\_embeds = Activation('relu')(concat\_embeds)

bn\_concat = BatchNormalization()(concat\_embeds)

# Tiến hành cho qua những lớp ẩn

fc1 = Dense(2048, use\_bias=False)(bn\_concat)

ac1 = ReLU()(fc1)

bn1 = BatchNormalization()(ac1)

fc2 = Dense(1024, use\_bias=False)(bn1)

ac2 = ReLU()(fc2)

bn2 = BatchNormalization()(ac2)

fc3 = Dense(512, use\_bias=False)(bn2)

ac3 = ReLU()(fc3)

bn3 = BatchNormalization()(ac3)

fc4 = Dense(256)(bn3)

dropout = Dropout(0.25)

ac4 = ReLU()(fc4)

self.per\_input = deep\_per\_input

self.api\_input = deep\_api\_input

# Đầu ra cuối cùng của deep model

self.deep\_component\_outlayer = ac4

print('Finish create deep model')

# Hàm này để tạo wide model

def wide\_model(self):

# Đưa input per vào model

dim\_per = self.x\_train\_per\_w.shape

print("x\_train\_per\_w = ", self.x\_train\_per\_w.shape)

print("x\_train\_per = ", self.x\_train\_per.shape)

self.logistic\_per\_input = Input(shape=(dim\_per,))

print("logistic\_per\_input = ",self.logistic\_per\_input.shape)

# Đưa input api vào model

dim\_api = self.x\_train\_api\_w.shape[1]

print("x\_train\_api\_w = ", self.x\_train\_api\_w.shape)

self.logistic\_api\_input = Input(shape=(dim\_api,))

print("logistic\_api\_input = ",self.logistic\_api\_input.shape)

# Hàm này thực hiện merge wide & deep model lại với nhau

def create\_model(self):

print('Starting create model ...')

self.deep\_component()

self.wide\_model()

if self.mode == 'wide and deep':

# Output layer của model cần phải có các thông tin output của từng model đã tạo

out\_layer = concatenate([self.deep\_component\_outlayer, self.logistic\_per\_input, self.logistic\_api\_input])

# Inputs cũng cần phải được đưa vào những thông tin đúng thứ tự đã định nghĩa của từng model

# Nên xem summary để biết thứ tự đưa vào các input hoặc xem thứ tự gọi các lệnh input của từng model

inputs = [self.per\_input] + [self.api\_input] + [self.logistic\_per\_input] + [self.logistic\_api\_input]

elif self.mode == 'deep':

out\_layer = self.deep\_component\_outlayer

inputs = [self.per\_input] + [self.api\_input]

else:

print('wrong mode')

return

# Output cuối cùng của model

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

print(f'input = {len(inputs)}')

# Tạo model với thông tin inputs và output đã định nghĩa

self.model = Model(inputs=inputs, outputs=output)

print(self.model.summary())

# Hàm này dùng để train model, ứng thứ tự các input đã khai báo, cần đưa vào model đúng thứ tự tập input train

def train\_model(self):

if not self.model:

print('You have to create model first')

return

if self.mode == 'wide and deep':

print("chạy vào train wide and deep")

input\_data = [self.x\_train\_per] + [self.x\_train\_api] + [self.x\_train\_per\_w] + [self.x\_train\_api\_w]

print("xxxxxxxx")

elif self.mode == 'deep':

input\_data = [self.x\_train\_per] + [self.x\_train\_api]

else:

print('wrong mode')

return

# Do bài toán ở đây là multi classification nên loss phải là 'categorical\_crossentropy'

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

print("qua compile")

# Chuyển đối số chiều của tập y\_train từ [len(y\_train), None] về [len(y\_train), N\_CLASSES]

self.y\_train = to\_categorical(self.y\_train, N\_CLASSES)

# Định nghĩa callbacks theo 'accuracy'

es = tf.keras.callbacks.EarlyStopping(monitor='accuracy', mode='max')

# callbacks = [es]

callbacks = []

# Trong quá trình train có thể sử dụng callbacks hoặc không.

self.history = self.model.fit(input\_data, self.y\_train, epochs=N\_EPOCHS, batch\_size=N\_BATCH\_SIZE)

# Hàm này dùng để đánh giá kết quả model đã train bằng tập test

# Các tham số đưa vào được tính toán tương tự với khi train

def evaluate\_model(self):

if not self.model:

print('You have to create model first')

return

if self.mode == 'wide and deep':

print("chạy vào test wide and deep")

input\_data = [self.x\_test\_per] + [self.x\_test\_api] + [self.x\_test\_per\_w] + [self.x\_test\_api\_w]

elif self.mode == 'deep':

input\_data = [self.x\_test\_per] + [self.x\_test\_api]

else:

print('wrong mode')

return

self.y\_test = to\_categorical(self.y\_test,N\_CLASSES)

loss, acc = self.model.evaluate(input\_data, self.y\_test)

print(f'\ntest\_loss: {loss} - test\_acc: {acc}')

# Lưu lại model dưới dạng .h5 (keras output format) để sử dụng về sau

def save\_model(self, filename='wide\_and\_deep.h5'):

self.model.save(filename)

if \_\_name\_\_ == '\_\_main\_\_':

# Khai báo 1 đối tượng Wide\_and\_Deep() và sử dụng các hàm đã định nghĩa của Wide\_and\_Deep

t0 = time.time()

print("bắt đầu chạy winde and deep")

wide\_deep\_net = Wide\_and\_Deep()

t1 = time.time()

print("chạy xong wide and deep")

print("bắt đầu chạy create model")

wide\_deep\_net.create\_model()

print("chạy xong create model")

print("bắt đầu chạy train model")

wide\_deep\_net.train\_model()

print("chạy xong train model")

print("bắt đầu chạy evaluate\_model")

wide\_deep\_net.evaluate\_model()

print("chạy xong evaluate\_model")

wide\_deep\_net.save\_model()

#plot\_graphs(wide\_deep\_net.history)

# Lưu lại thông tin của model dứa dạng file .png

#plot\_model(wide\_deep\_net.model, to\_file='model.png', show\_shapes=True, show\_layer\_names=False)

t2 = time.time()

print("thoi gian chạy model không tính comatrix: %.2f phút." %((t2-t1)/60))

print("thoi gian chạy xong model: %.2f phút." %((t2-t0)/60))

print('Hoàn thành việc huấn luyện và kiểm tra!')