

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



BÁO CÁO ĐỀ TÀI

**“PHÂN LOẠI ẢNH TRÊN TẬP CIFAR-10
SỬ DỤNG THƯ VIỆN KERAS”**

Giảng viên hướng dẫn: Đào Thị Thúy Quỳnh

Sinh viên: Nguyễn Quang Huy

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn đến Học viện Công nghệ Bưu chính Viễn Thông đã tạo điều kiện cho em được học môn Nhập môn trí tuệ nhân tạo . Đặc biệt, em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới cô **ĐÀO THỊ THÚY QUỲNH**, giảng viên bộ môn đã hướng dẫn và truyền đạt những kiến thức bổ ích và quý báu trong suốt thời gian học tập vừa qua.

Với vốn hiểu biết sâu rộng và kinh nghiệm nhiều năm giảng dạy cũng như làm việc trong môi trường công nghệ thông tin, cô khiến chúng em thật sự ấn tượng, khâm phục và “ngỡ ngàng” trước những hiểu biết của cô. Những kiến thức, kinh nghiệm cô chia sẻ khiến chúng em thật sự ngưỡng mộ và một phần hãnh diện khi được học dưới sự hướng dẫn của cô.

Môn học Nhập môn trí tuệ nhân tạo là môn học mang tính chuyên ngành trong chương trình đào tạo công nghệ thông tin hệ đại học. Môn học đã giúp em quen với khái niệm trí tuệ nhân tạo thông qua việc giới thiệu một số kỹ thuật và ứng dụng cụ thể. Với việc học về trí tuệ nhân tạo, em đã được làm quen với những phương pháp, cách giải quyết vấn đề không thuộc lĩnh vực toán rời rạc hoặc giải thuật truyền thống, chẳng hạn các phương pháp dựa trên heuristics, các phương pháp dựa trên tri thức, dữ liệu. Mặt khác, em lại được làm quen với khả năng ứng dụng tiềm tàng các kỹ thuật trí tuệ nhân tạo trong nhiều bài toán thực tế. Tuy nhiên do chưa có nhiều kinh nghiệm về làm báo cáo cũng như những hạn chế về mặt kiến thức , trong bài báo cáo chắc chắn cũng không thể tránh khỏi những thiếu sót . Em mong nhận được sự đóng góp , phê bình từ phía cô để báo cáo của em được hoàn thiện hơn .

Một lần nữa, em xin chân thành cảm ơn cô!

Hà Nội, Tháng 05 năm 2022

Huy

Nguyễn Quang Huy

Contents

LỜI CẢM ƠN	2
I , Giới thiệu tập CIFAR-10	4
II ,Giới thiệu về CNN	4
1.Convolutional layer	5
2.Hàm Relu	6
3.Pooling layer	7
4.Fully connected layer	8
5.Tóm tắt	8
III ,Các bước giải bài toán.....	9
1 Tiền xử lý dữ liệu (Data pre-processing)	9
1.1Tải xuống và đọc tập dữ liệu.....	9
1.2 Hình dung một phần của tập huấn luyện.....	9
1.3 Chuẩn hóa hình ảnh(Image normalize)	10
1.4 Mã hóa one-hot (One-Hot Encoding)	10
2 .Xây dựng mạng CNN	11
2.1 . Một số khái niệm liên quan.....	11
2.2 Xây dựng mô hình.....	14
2.3 Quy trình đào tạo	19
2.4 Hình dung quá trình đào tạo.....	19
2.5 Đánh giá mô hình.....	20
2.6 Kết quả dự đoán	20
2.7. Hiển thị ma trận nhầm lẫn.....	21
3. Lưu lịch sử đào tạo và mô hình.....	23
3.1 Lưu lịch sử đào tạo.....	23
3.2 Lưu mô hình.....	24
4. Đẩy mô hình lên web và thử nghiệm:	25
4.1Giới thiệu sơ qua về fast API:	25
4.2 Hướng dẫn chạy project	25
4.3 . Test thử nghiệm	28

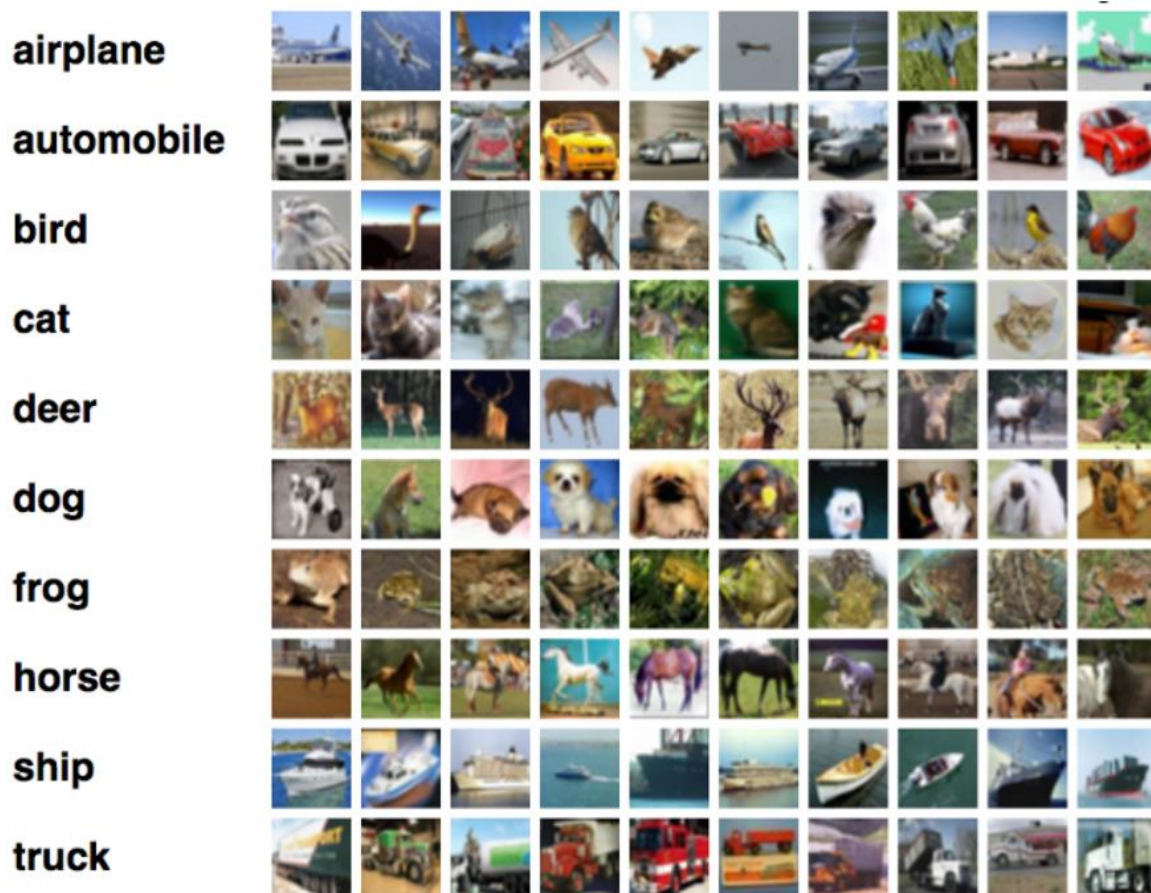
I , Giới thiệu tập CIFAR-10

-CIFAR-10 (Canadian Institute for Advanced Research) là một trong những bộ dữ liệu chuẩn cho nhiệm vụ phân loại hình ảnh được thu thập bởi Alex Krizhevsky, Vinod Nair và Geoffrey Hinton. Nó là một tập gồm 60.000 hình ảnh màu (32x32) trong đó 50000 cho training và 10000 cho testing.

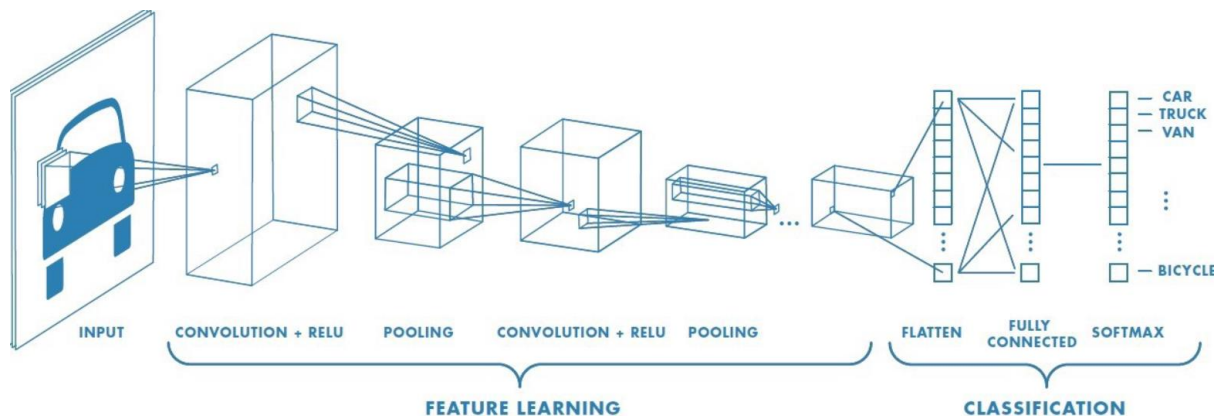
-Dataset của CIFAR-10 bao gồm 10 lớp đối tượng mỗi loại có 6000 hình ảnh là : Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck. Ảnh 3 kênh RGB có kích cỡ 32x32x3.

-Lý do chọn CIFAR-10: Một là nó đủ phức tạp để đào tạo hầu hết các khả năng của các mô hình quy mô lớn của TensorFlow, và hai là nó đủ nhỏ để đào tạo nhanh chóng, lý tưởng để thử các ý tưởng mới và thử nghiệm công nghệ mới.

- CIFAR-10 đã có sẵn trong datasets module của Keras, chúng ta chỉ việc import trực tiếp từ keras.datasets.



II ,Giới thiệu về CNN



Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và tanh để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Đây một trong những mô hình Deep Learning tiên tiến giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

CNN bao gồm những phần lớp cơ bản là:

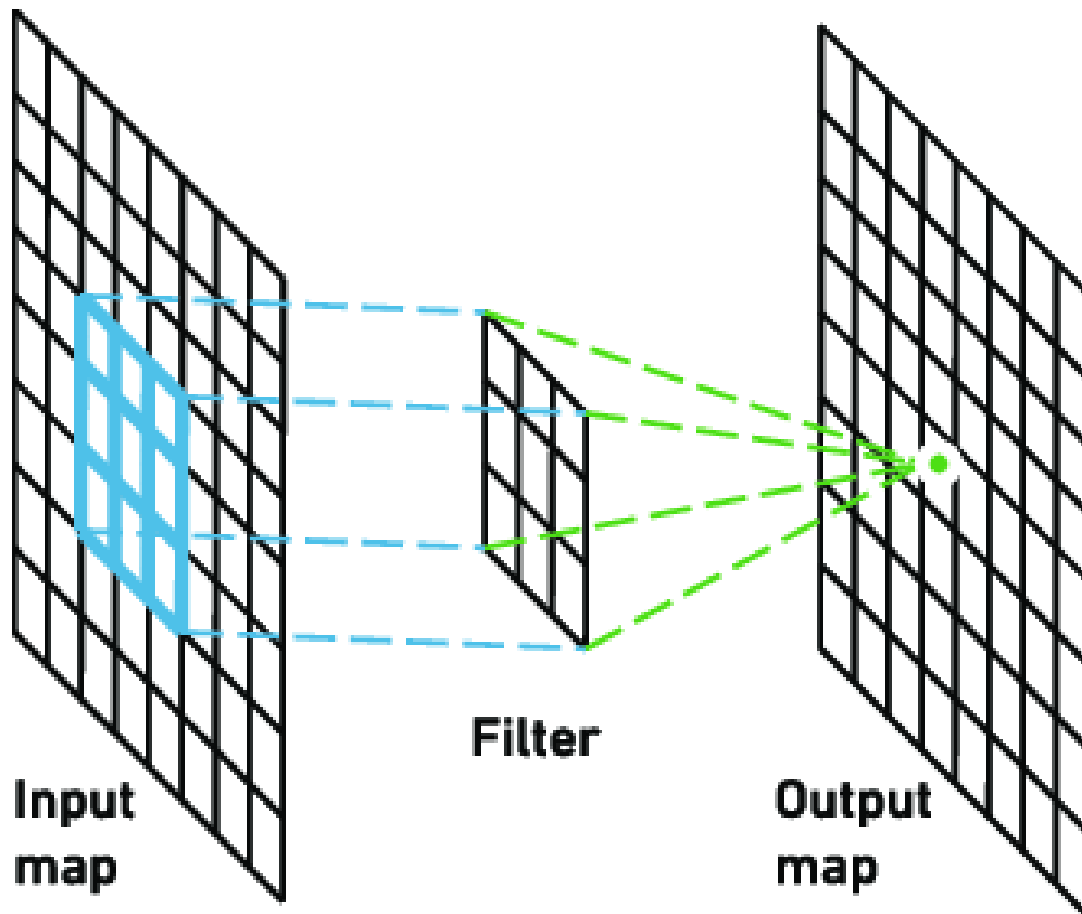
1.Convolutional layer

Đây là lớp quan trọng nhất của CNN, lớp này có nhiệm vụ thực hiện mọi tính toán. Nếu lớp này có số lượng nhiều thì chương trình chạy càng được cải thiện. Sử dụng các layer với số lượng lớn có thể dẫn đến tác động được giảm một cách đáng kể. Thường thì chỉ sau 3 đến 4 layer thôi là ta sẽ đạt được kết quả như mong muốn. Những yếu tố quan trọng của một convolutional layer là: stride, padding, filter map, feature map.

CNN sử dụng các filter để áp dụng vào vùng của hình ảnh. Những filter map này được gọi là ma trận 3 chiều, mà bên trong nó là các con số và chúng là parameter. Stride có nghĩa là khi bạn dịch chuyển filter map theo pixel dựa vào giá trị trừ trái sang phải. Và sự chuyển dịch này chính là Stride.

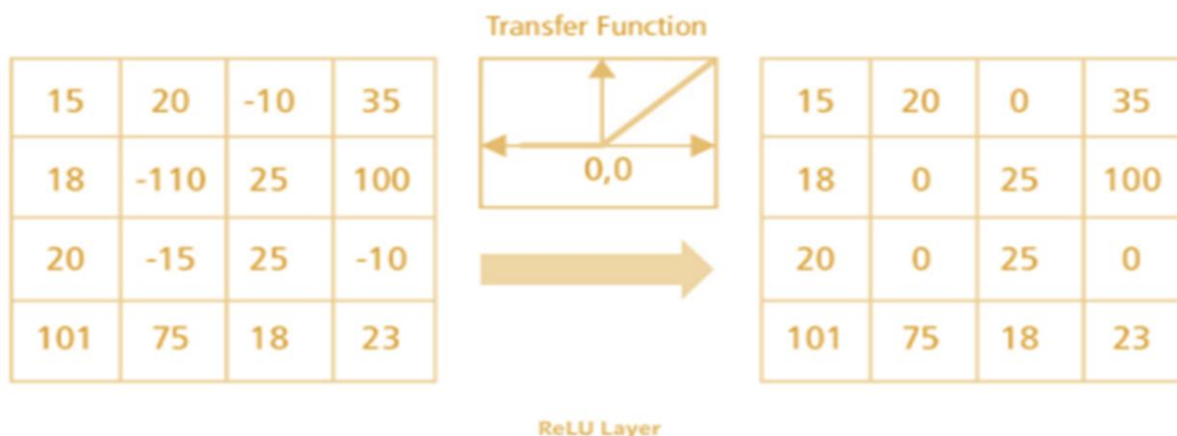
Padding: Là các giá trị 0 được thêm vào với lớp input.

Feature map: Nó thể hiện kết quả của mỗi lần filter map quét qua input. Sau mỗi lần quét sẽ xảy ra quá trình tính toán.



2. Hàm Relu

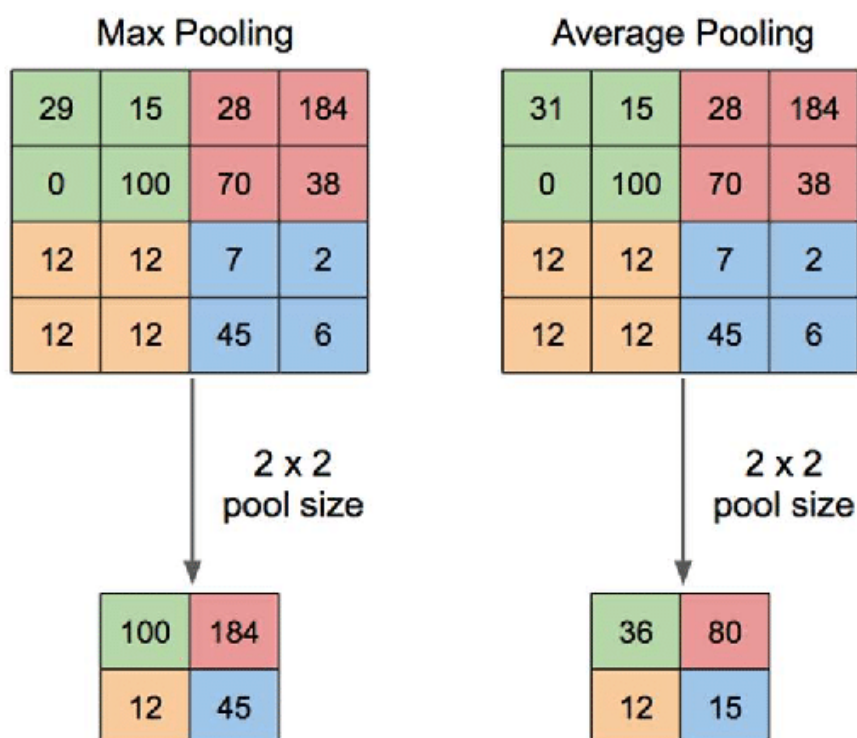
Relu layer (Rectified Linear Unit) là hàm kích hoạt trong neural network và hàm này còn được gọi là activation function. ReLU với đầu ra là: $f(x) = \max(0, x)$ thì đây là một hàm kích hoạt có tác dụng mô phỏng các neuron có tỷ lệ truyền xung qua axon. Trong activation function thì nó còn có hàm nghĩa là: Relu, Leaky, Tanh, Sigmoid, Maxout,... Hiện nay, hàm relu được dùng phổ biến và vô cùng thông dụng. Nó được sử dụng nhiều cho các nhu cầu huấn luyện mạng neuron thì relu mang lại rất nhiều ưu điểm nổi bật như: việc tính toán sẽ trở nên nhanh hơn,... Quá trình sử dụng relu, chúng ta cần lưu ý đến vấn đề tùy chỉnh các learning rate và theo dõi dead unit. Những lớp relu layer đã được sử dụng sau khi filter map được tính ra và áp dụng hàm relu lên những giá trị của filter map.



3.Pooling layer

Lớp chứa hay lớp tổng hợp (Pooling layer): là những lớp có tác dụng làm đơn giản hóa các thông tin ở đầu ra từ các lớp tích chập, giúp làm giảm parameter khi đầu vào quá lớn. Lớp pooling thường được sử dụng ngay sau lớp tích chập.

Có 2 loại pooling layer phổ biến là: max pooling và average pooling:



Nguyên lý hoạt động 2 loại pooling

Mục đích của pooling rất đơn giản, nó làm giảm số siêu tham số mà ta cần phải tính toán, từ đó giảm thời gian tính toán, tránh overfitting. Loại pooling ta thường gặp nhất là max pooling, lấy giá trị lớn nhất trong một cửa sổ pooling. Pooling hoạt động gần giống với tích chập, nó cũng có 1 cửa sổ trượt gọi là pooling window, cửa sổ này trượt qua từng giá trị của ma trận dữ liệu đầu vào, chọn ra một giá trị từ các giá trị nằm trong cửa sổ trượt (với max pooling ta sẽ lấy giá trị lớn nhất). Ví dụ như hình vẽ

trên ta chọn pooling window có kích thước là $2 * 2$, stride = 2 để đảm bảo không trùng nhau, và áp dụng max pooling / average pooling.

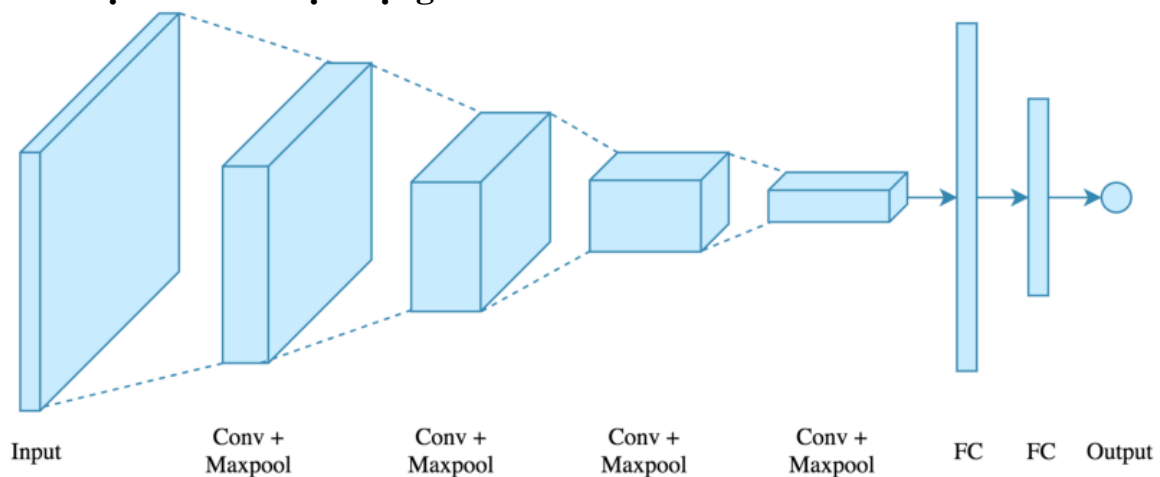
4. Fully connected layer

Lớp này có nhiệm vụ đưa ra kết quả sau khi lớp convolutional layer và pooling layer đã nhận được ảnh truyền. Lúc này, ta thu được kết quả là model đã đọc được thông tin của ảnh và để liên kết chúng cũng như cho ra nhiều output hơn thì ta sử dụng fully connected layer.

Ngoài ra, nếu như fully connected layer có được dữ liệu hình ảnh thì chúng sẽ chuyển nó thành mục chưa được phân chia chất lượng. Cái này khá giống với phiếu bầu rồi chúng sẽ đánh giá để bầu chọn ra hình ảnh có chất lượng cao nhất.

Giống trong mạng nơ ron thì mỗi lớp ẩn được gọi là kết nối đầy đủ - fully connected. Thường sau lớp kết nối đầy đủ sẽ là 2 lớp kết nối đầy đủ, 1 lớp để tập hợp các lớp đặc trưng mà ta đã tìm ra, chuyển đổi dữ liệu từ 3D, hoặc 2D thành 1D, tức chỉ còn là 1 vector. Còn 1 lớp nữa là đầu ra, số nơ ron của lớp này phụ thuộc vào số đầu ra mà ta muốn tìm ra.

Ví dụ cấu trúc một mạng CNN tiêu chuẩn:



Trên tư tưởng của một mạng CNN tiêu chuẩn cùng với sự đóng góp dữ liệu từ thử thách nhận dạng hình ảnh quy mô lớn ImageNet hoặc ILSVRC đã đem đến nhiều mô hình CNN có độ chính xác ngày được cải thiện và nhiều hơn các biến thể CNN được ra đời. Một số các mô hình CNN, biến thể CNN nổi bật có những kết quả cao trong bài toán phân loại ảnh.

Để đánh giá một mô hình thì trước hết cần có một tập dữ liệu tiêu chuẩn, một thước đo cho sự chính xác đảm bảo tính công bằng. Trong bài toán phân loại ảnh thì ImageNet được xem là một trong tập dữ liệu đáng tin cậy và trực quan nhất đánh giá các mô hình nhận dạng ảnh thông qua cuộc thi hằng năm.

5. Tóm tắt

-Đầu vào của lớp tích chập là hình ảnh

- Chọn đối số, áp dụng các bộ lọc với các bước nhảy, padding nếu cần. Thực hiện tích chập cho hình ảnh và áp dụng hàm kích hoạt ReLU cho ma trận hình ảnh.
- Thực hiện Pooling để giảm kích thước cho hình ảnh.
- Thêm nhiều lớp tích chập sao cho phù hợp
- Xây dựng đầu ra và dữ liệu đầu vào thành 1 lớp được kết nối đầy đủ (Full Connected)
- Sử dụng hàm kích hoạt để tìm đối số phù hợp và phân loại hình ảnh.

III ,Các bước giải bài toán

1 Tiền xử lý dữ liệu (Data pre-processing)

1.1 Tải xuống và đọc tập dữ liệu

```

import numpy
from keras.datasets import cifar10
import numpy as np
np.random.seed(10) #.....
(X_train,y_train),(X_test, y_test)=cifar10.load_data() # .....
print('số ảnh dùng để train:',len(X_train))
print('số ảnh dùng để test :',len(X_test))
print('kích cỡ ảnh train :',X_train.shape)
print('kích cỡ train_label :',y_train.shape)
print('kích cỡ ảnh test :',X_test.shape)
print('kích cỡ test_label :',y_test.shape)

```

```

↳ Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [=====] - 3s 0us/step
170508288/170498071 [=====] - 3s 0us/step
số ảnh dùng để train: 50000
số ảnh dùng để test : 10000
kích cỡ ảnh train : (50000, 32, 32, 3)
kích cỡ train_label : (50000, 1)
kích cỡ ảnh test : (10000, 32, 32, 3)
kích cỡ test_label : (10000, 1)

```

1.2 Hình dung một phần của tập huấn luyện

```

showImageandLabel(X_train,y_train,0,12)#bắt đầu từ 0 kết thúc tại 12

```



1.3 Chuẩn hóa hình ảnh(Image normalize)

- Các giá trị pixel có thể nằm trong khoảng từ 0 đến 256. Mỗi số đại diện cho một mã màu.
- Khi sử dụng hình ảnh như vậy và chuyển qua Mạng nơ ron sâu, việc tính toán các giá trị số cao có thể trở nên phức tạp hơn.
- Để giảm điều này, chúng ta có thể chuẩn hóa các giá trị trong phạm vi từ 0 đến 1. Bằng cách này, các con số sẽ nhỏ hơn và việc tính toán trở nên dễ dàng và nhanh chóng hơn.
- Vì các giá trị pixel nằm trong khoảng từ 0 đến 256, ngoài 0, phạm vi là 255. Vì vậy, chia tất cả các giá trị cho 255 sẽ chuyển đổi nó thành phạm vi từ 0 đến 1.

```

print(x_img_train[0][0][0]) # (50000, 32, 32, 3)
x_img_train_normalize = x_img_train.astype('float32') / 255.0
x_img_test_normalize = x_img_test.astype('float32') / 255.0
print(x_img_train_normalize[0][0][0])

```

```

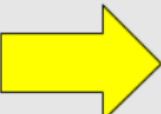
[59 62 63]
[0.23137255 0.24313726 0.24705882]

```

1.4 Mã hóa one-hot (One-Hot Encoding)

-Mã hóa one-hot là biểu diễn của các biến phân loại dưới dạng vector nhị phân. Điều này đầu tiên yêu cầu các giá trị phân loại được ánh xạ thành các giá trị số nguyên. Sau đó, mỗi giá trị số nguyên được biểu diễn dưới dạng vector nhị phân có tất cả các giá trị bằng 0 ngoại trừ chỉ số của số nguyên, được đánh dấu bằng 1.

One-hot encoding là một quá trình mà các biến phân loại (label) được chuyển đổi thành một mẫu có thể cung cấp cho các thuật toán ML để thực hiện công việc tốt hơn khi mà dự đoán.

Color		Red	Yellow	Green
Red				
Red		1	0	0
Yellow		1	0	0
Green		0	1	0
Yellow		0	0	1

-API Keras cung cấp một `to_categorical()` phương thức có thể được sử dụng để mã hóa một lần dữ liệu *số nguyên*. Nếu dữ liệu số nguyên đại diện cho tất cả các giá trị có thể có của các lớp, thì `to_categorical()` phương thức có thể được sử dụng trực tiếp; nếu không, số lớp có thể được truyền cho phương thức dưới dạng `num_classes` tham số.

```

▶ from keras.utils import np_utils
  y_train_OneHot = np_utils.to_categorical(y_train)
  y_test_OneHot = np_utils.to_categorical(y_test)
  print(y_train_OneHot.shape)
  print(y_train_OneHot[:5])

```

```

↳ (50000, 10)
  [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
   [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
   [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
   [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
   [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]

```

2. Xây dựng mạng CNN

2.1. Một số khái niệm liên quan

2.1.1 Keras là gì

Keras là một open source cho Neural Network được viết bởi ngôn ngữ Python. Nó là một library được phát triển vào năm 2015 bởi Francois Chollet, là một kỹ sư nghiên cứu Deep Learning. Keras có thể sử dụng chung với các thư viện nổi tiếng như Tensorflow, CNTK, Theano. Một số ưu điểm của Keras như:

- Dễ sử dụng, dùng đơn giản hơn Tensor, xây dựng model nhanh.

- Run được trên cả CPU và GPU.
- Hỗ trợ xây dựng CNN, RNN hoặc cả hai. Với những người mới tiếp cận đến Deep như mình thì mình chọn sử dụng Keras để build model vì nó đơn giản, dễ nắm bắt hơn các thư viện khác. Dưới đây mình xin giới thiệu một chút về API này.

2.1.2 Sequential model

Trong Keras có hỗ trợ 2 cách dựng models là Sequential model và Function API. Kiểu mô hình Sequential cho phép bạn xây dựng mô hình theo từng lớp, tuần tự.

2.1.3 Tạo Convolutional Layers

Conv2D là convolution dùng để lấy feature từ ảnh với các tham số:

- filters : số filter của convolution
- kernel_size : kích thước window search trên ảnh
- strides : số bước nhảy trên ảnh
- activation : chọn activation như linear, softmax, relu, tanh, sigmoid. Đặc điểm mỗi hàm các bạn có thể search thêm để biết cụ thể nó ntn.
- padding : có thể là "valid" hoặc "same". Với same thì có nghĩa là padding=1.

2.1.4 Pooling Layers:

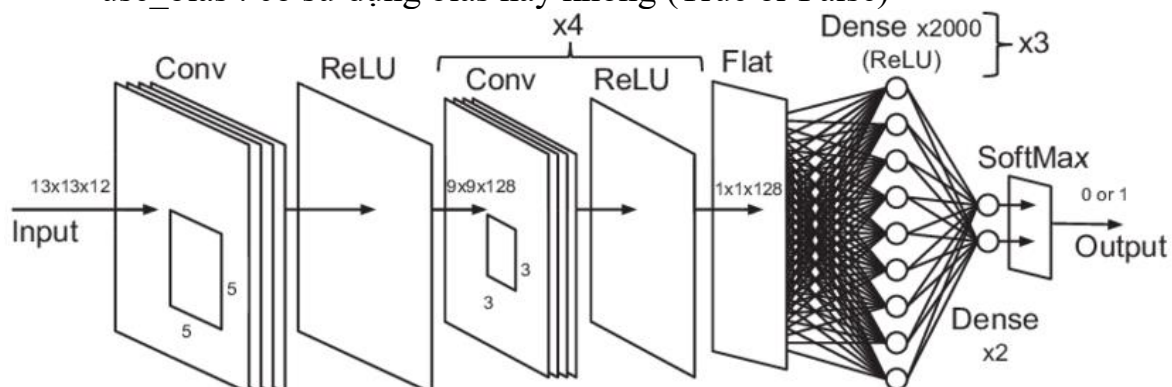
Được sử dụng để làm giảm param khi train, nhưng vẫn giữ được đặc trưng của ảnh.

- pool_size : kích thước ma trận để lấy max hay average
- Ngoài ra còn có : MaxPooling2D, AveragePooling1D, 2D (lấy max, trung bình) với từng size.

2.1.5 Dense ()

Layer này cũng như một layer neural network bình thường, với các tham số sau:

- units : số chiều output, như số class sau khi train (chó, mèo, lợn, gà).
- activation : chọn activation đơn giản với sigmoid thì output có 1 class.
- use_bias : có sử dụng bias hay không (True or False)



2.1.6 Hàm compile:

Ở hàm này chúng ta sử dụng để training models như thuật toán train qua optimizer như Adam, SGD, RMSprop,...

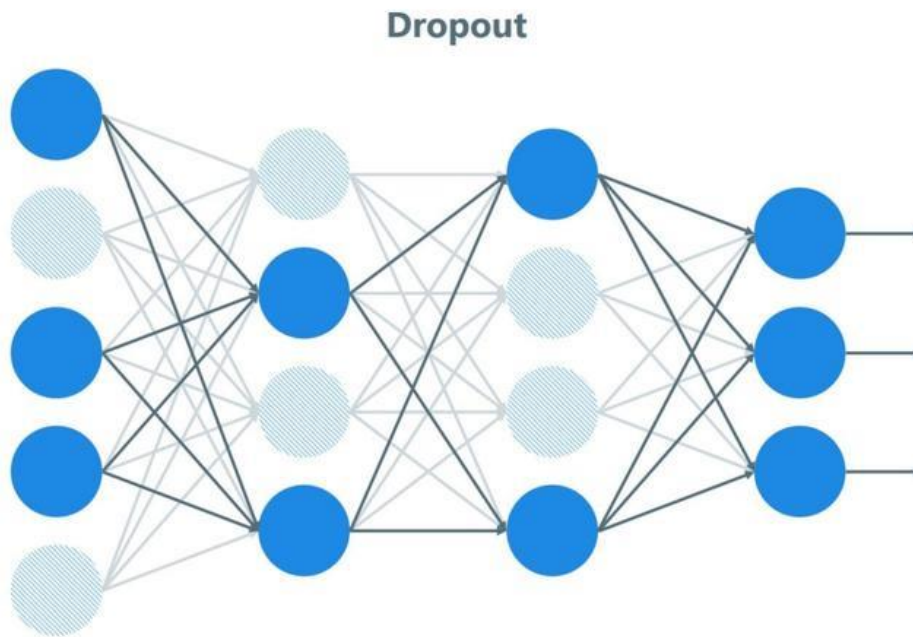
learning_rate : dạng float , tốc độ học, chọn phù hợp để hàm số hội tụ nhanh.

2.1.7 Hàm fit ():

- Bao gồm data train, test đưa vào training.
- Batch_size thể hiện số lượng mẫu mà Mini-batch GD sử dụng cho mỗi lần cập nhật trọng số .
- Epoch là số lần duyệt qua hết số lượng mẫu trong tập huấn luyện.

2.1.8 Dropout:

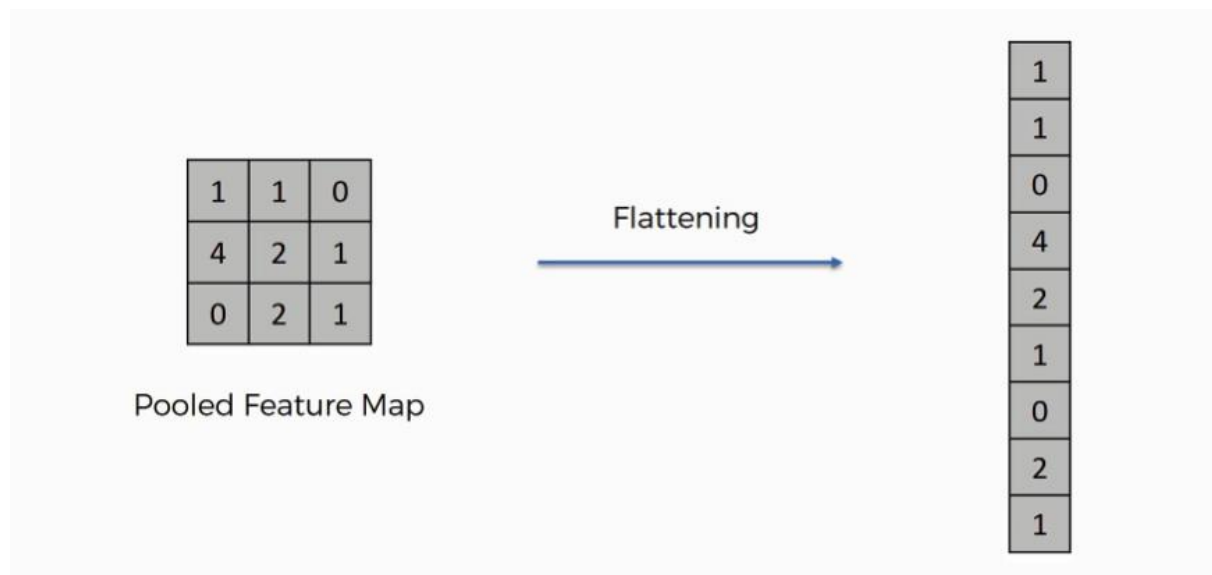
Trong mạng neural network, kỹ thuật dropout là việc chúng ta sẽ bỏ qua một vài unit trong suốt quá trình train trong mô hình, những unit bị bỏ qua được lựa chọn ngẫu nhiên. Ở đây, chúng ta hiểu “bỏ qua - ignoring” là unit đó sẽ không tham gia và đóng góp vào quá trình huấn luyện (lan truyền tiến và lan truyền ngược).



2.1.9 Flatten

Bước làm phẳng là một bước rất đơn giản liên quan đến việc xây dựng một mạng nơ-ron tích hợp. Nó liên quan đến việc lấy pooled feature map được tạo ra trong bước gộp và biến nó thành vector một chiều.

Lý do tại sao chúng ta biến đổi pooled feature map thành vector một chiều là vì vector này bây giờ sẽ được đưa vào một mạng nơ-ron nhân tạo . Nói cách khác, vector này bây giờ sẽ trở thành lớp đầu vào của mạng nơ-ron nhân tạo sẽ được xâu chuỗi vào mạng nơ-ron tích tụ mà chúng tôi đã xây dựng cho đến nay trong khóa học này.



2.2 Xây dựng mô hình

Code:

```
3 ✓
giây ▶ from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

model=Sequential()

model.add(Conv2D(filters=32,kernel_size=(3, 3),input_shape=(32, 32,3),
                activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=32, kernel_size=(3, 3),
                activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3),
                activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=64, kernel_size=(3, 3),
                activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3),
                activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=128, kernel_size=(3, 3),
                activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
|
# Xây dựng mạng nơ-ron (lớp phẳng, lớp ẩn, lớp đầu ra)
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(2500, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1500, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
print(model.summary())
```

Bản tóm tắt :

[12] Model: "sequential_1"

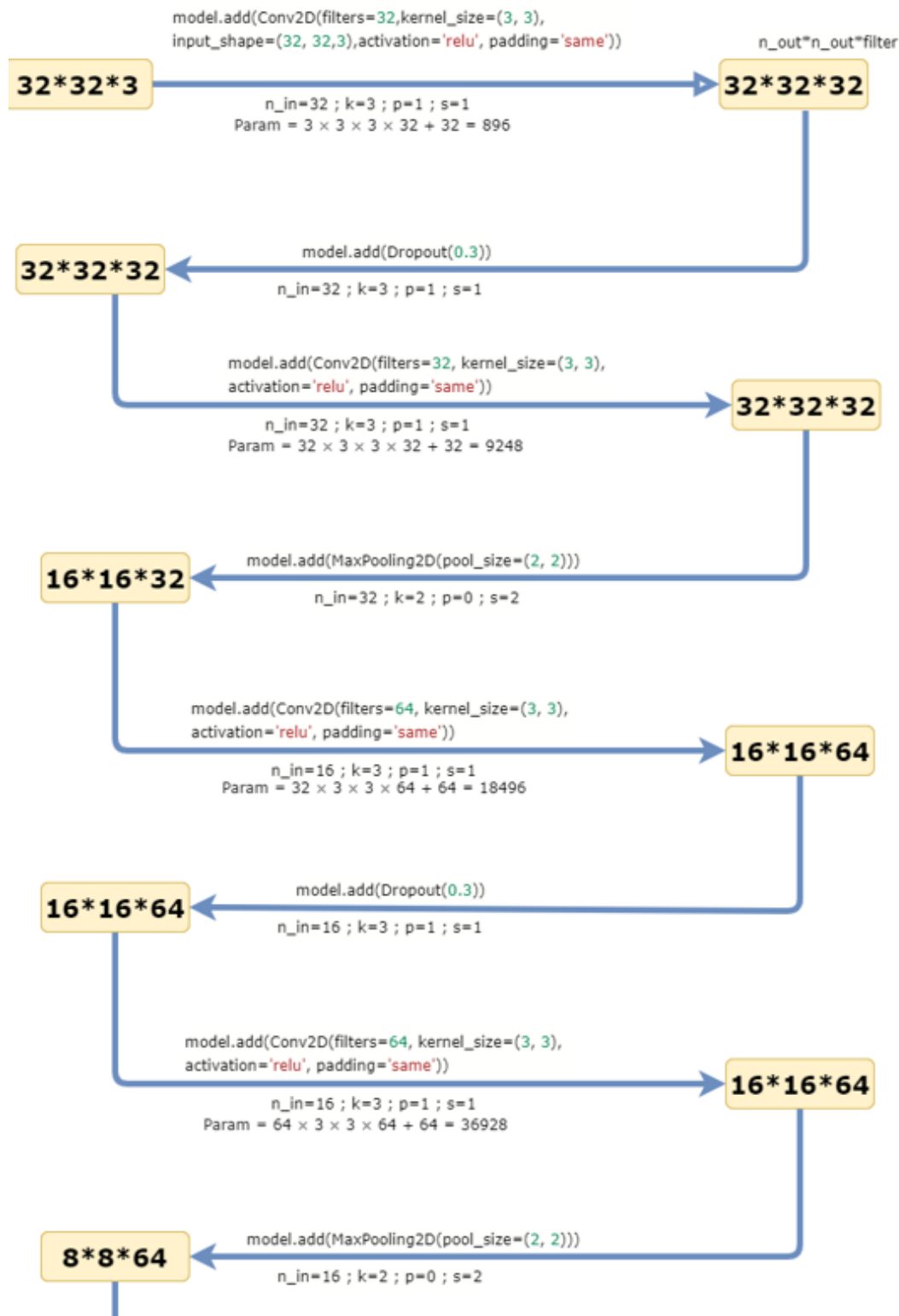
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 32)	896
dropout_6 (Dropout)	(None, 32, 32, 32)	0
conv2d_7 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_3 (MaxPooling 2D)	(None, 16, 16, 32)	0
conv2d_8 (Conv2D)	(None, 16, 16, 64)	18496
dropout_7 (Dropout)	(None, 16, 16, 64)	0
conv2d_9 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_4 (MaxPooling 2D)	(None, 8, 8, 64)	0
conv2d_10 (Conv2D)	(None, 8, 8, 128)	73856
dropout_8 (Dropout)	(None, 8, 8, 128)	0
conv2d_11 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_5 (MaxPooling 2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dropout_9 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 2500)	5122500
dropout_10 (Dropout)	(None, 2500)	0
dense_4 (Dense)	(None, 1500)	3751500
dropout_11 (Dropout)	(None, 1500)	0
dense_5 (Dense)	(None, 10)	15010
=====		
Total params: 9,176,018		
Trainable params: 9,176,018		
Non-trainable params: 0		
=====		
None		

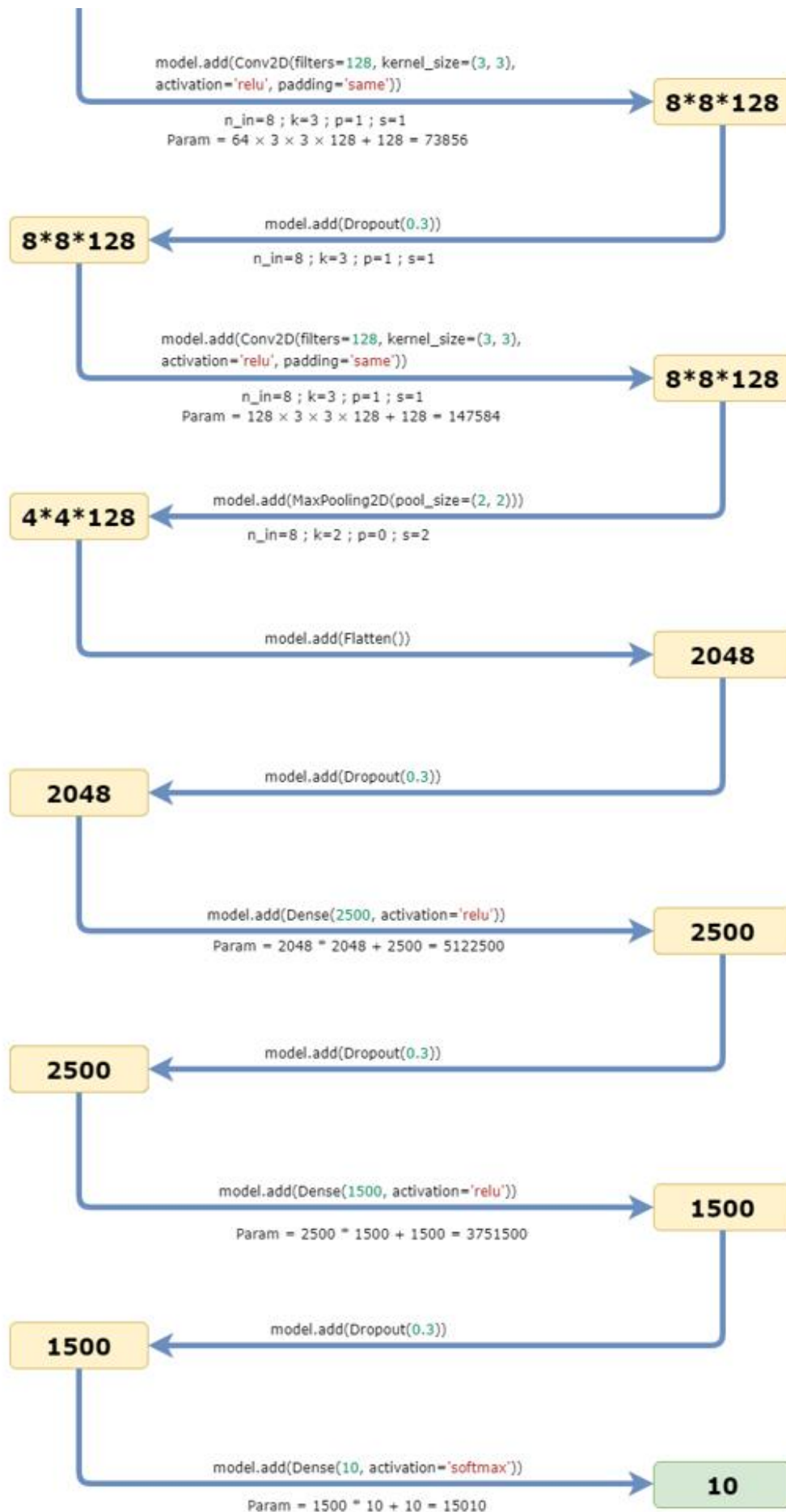
Sơ đồ hoạt động:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in} : number of input features
 n_{out} : number of output features
 k : convolution kernel size
 p : convolution padding size
 s : convolution stride size

công thức tính số tham số của
 một mạng phức hợp là:
 $\text{channel_in} * \text{kernel_width} * \text{kernel_height} * \text{channel_out} + \text{channel_out}$.





2.3 Quy trình đào tạo



```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
train_history=model.fit(X_train_normalize, y_train_OneHot,
                        validation_split=0.2,
                        epochs=40, batch_size=300, verbose=1)
```

3 phút

```
Epoch 19/40
134/134 [=====] - 5s 40ms/step - loss: 0.3964 - accuracy: 0.8602 - val_loss: 0.7135 - val_accuracy: 0.7714
Epoch 20/40
134/134 [=====] - 5s 39ms/step - loss: 0.3754 - accuracy: 0.8641 - val_loss: 0.6650 - val_accuracy: 0.7863
Epoch 21/40
134/134 [=====] - 5s 39ms/step - loss: 0.3577 - accuracy: 0.8716 - val_loss: 0.6961 - val_accuracy: 0.7804
Epoch 22/40
134/134 [=====] - 6s 41ms/step - loss: 0.3431 - accuracy: 0.8781 - val_loss: 0.6878 - val_accuracy: 0.7817
Epoch 23/40
134/134 [=====] - 5s 39ms/step - loss: 0.3269 - accuracy: 0.8827 - val_loss: 0.6454 - val_accuracy: 0.7931
Epoch 24/40
134/134 [=====] - 5s 39ms/step - loss: 0.3093 - accuracy: 0.8886 - val_loss: 0.6672 - val_accuracy: 0.7935
Epoch 25/40
134/134 [=====] - 5s 39ms/step - loss: 0.2927 - accuracy: 0.8959 - val_loss: 0.7331 - val_accuracy: 0.7813
Epoch 26/40
134/134 [=====] - 5s 40ms/step - loss: 0.2837 - accuracy: 0.8982 - val_loss: 0.6863 - val_accuracy: 0.7973
Epoch 27/40
134/134 [=====] - 5s 41ms/step - loss: 0.2715 - accuracy: 0.9046 - val_loss: 0.6925 - val_accuracy: 0.7940
Epoch 28/40
134/134 [=====] - 5s 39ms/step - loss: 0.2510 - accuracy: 0.9126 - val_loss: 0.7183 - val_accuracy: 0.7940
Epoch 29/40
134/134 [=====] - 5s 39ms/step - loss: 0.2474 - accuracy: 0.9124 - val_loss: 0.7028 - val_accuracy: 0.7910
Epoch 30/40
134/134 [=====] - 5s 39ms/step - loss: 0.2332 - accuracy: 0.9184 - val_loss: 0.7480 - val_accuracy: 0.7897
Epoch 31/40
134/134 [=====] - 5s 39ms/step - loss: 0.2287 - accuracy: 0.9208 - val_loss: 0.7141 - val_accuracy: 0.7937
Epoch 32/40
134/134 [=====] - 6s 41ms/step - loss: 0.2190 - accuracy: 0.9230 - val_loss: 0.7524 - val_accuracy: 0.7910
Epoch 33/40
134/134 [=====] - 5s 39ms/step - loss: 0.2181 - accuracy: 0.9233 - val_loss: 0.7211 - val_accuracy: 0.7975
Epoch 34/40
134/134 [=====] - 5s 40ms/step - loss: 0.2023 - accuracy: 0.9296 - val_loss: 0.7231 - val_accuracy: 0.7965
Epoch 35/40
134/134 [=====] - 5s 39ms/step - loss: 0.1994 - accuracy: 0.9296 - val_loss: 0.7618 - val_accuracy: 0.7982
Epoch 36/40
134/134 [=====] - 5s 39ms/step - loss: 0.2021 - accuracy: 0.9301 - val_loss: 0.7414 - val_accuracy: 0.7951
Epoch 37/40
134/134 [=====] - 5s 39ms/step - loss: 0.1932 - accuracy: 0.9331 - val_loss: 0.7254 - val_accuracy: 0.7959
Epoch 38/40
134/134 [=====] - 5s 39ms/step - loss: 0.1892 - accuracy: 0.9335 - val_loss: 0.7159 - val_accuracy: 0.7977
Epoch 39/40
134/134 [=====] - 5s 39ms/step - loss: 0.1809 - accuracy: 0.9366 - val_loss: 0.7762 - val_accuracy: 0.7932
Epoch 40/40
134/134 [=====] - 5s 39ms/step - loss: 0.1832 - accuracy: 0.9366 - val_loss: 0.7774 - val_accuracy: 0.7910
```

2.4 Hình dung quá trình đào tạo

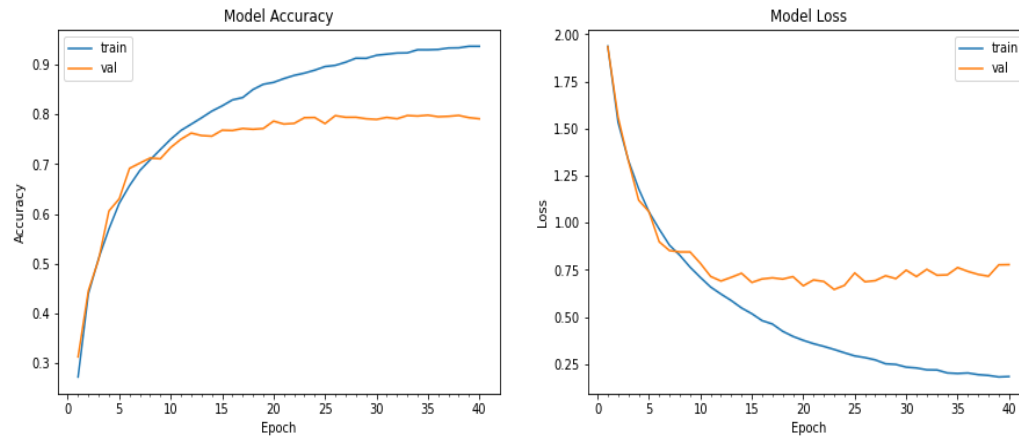


```
import matplotlib.pyplot as plt
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # vẽ đồ thị độ chính xác
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['accuracy']) #vẽ đồ thị accuracy
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.history['val_accuracy']) #vẽ đồ thị val_accuracy
    axs[0].set_title('Model Accuracy') #đặt tiêu đề đồ thị
    axs[0].set_ylabel('Accuracy') #đặt tiêu đề trục y
    axs[0].set_xlabel('Epoch') #đặt tiêu đề trục x
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_history.history['accuracy'])/10) #lấy vị trí và nhãn của trục x
    axs[0].legend(['train', 'val'], loc='best') #hiển thị ghi chú trong đồ thị
    # vẽ đồ thị độ tổn thất
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss']) #vẽ đồ thị loss
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['val_loss']) #vẽ đồ thị val_loss
    axs[1].set_title('Model Loss') #đặt tiêu đề đồ thị
    axs[1].set_ylabel('Loss') #đặt tiêu đề trục y
    axs[1].set_xlabel('Epoch') #đặt tiêu đề trục x
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.history['loss'])/10) #lấy vị trí và nhãn của trục x
    axs[1].legend(['train', 'val'], loc='best') #hiển thị ghi chú trong đồ thị
    plt.show() #xuất đồ thị
plot_model_history(model_history)
```

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()
# Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: MatplotlibDeprecationWarning: Passing the minor parameter of set_xticks()

```



2.5 Đánh giá mô hình

```

▶ scores = model.evaluate(X_test_normalize,
                           y_test_OneHot, verbose=0)
print(scores)
print(scores[0])

```

```

▶ [0.8274039030075073, 0.7788000106811523]
0.8274039030075073

```

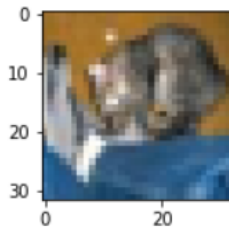
2.6 Kết quả dự đoán

```

▶ Predicted_Probability = model.predict(X_test_normalize) #.....
label_dict={0:"airplane",1:"automobile",2:"bird",3:"cat",4:"deer",
             5:"dog",6:"frog",7:"horse",8:"ship",9:"truck"}
show_Predicted_Probability(y_test,prediction,X_test,Predicted_Probability,0)
print("-----")
show_Predicted_Probability(y_test,prediction,X_test,Predicted_Probability,3)

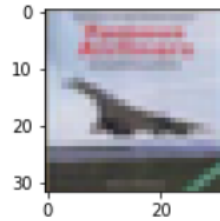
```


label: cat predict: cat



airplane Probability:0.001110370
automobile Probability:0.006932677
bird Probability:0.002242052
cat Probability:0.615930974
deer Probability:0.000447320
dog Probability:0.146729708
frog Probability:0.109238490
horse Probability:0.000453487
ship Probability:0.107873984
truck Probability:0.009040937

label: airplane predict: airplane



airplane Probability:0.934520364
automobile Probability:0.000003512
bird Probability:0.003013485
cat Probability:0.001392769
deer Probability:0.000001285
dog Probability:0.000000033
frog Probability:0.000000089
horse Probability:0.000001865
ship Probability:0.060911510
truck Probability:0.000155127

```
[ ] showImageandLabelPrediction(X_test,y_test,prediction,0, 12)
```



2.7. Hiển thị ma trận nhầm lẫn

2.7.1 Ma trận nhầm lẫn

{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}

```
import pandas as pd
print(label_dict)
pd.crosstab(y_test.reshape(-1),prediction,rownames=['label'],colnames=['predict'])
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	755	7	68	24	20	4	9	8	80	25
1	6	877	9	18	3	4	7	1	27	48
2	44	2	767	57	59	20	34	12	5	0
3	15	2	86	630	49	137	50	21	4	6
4	7	0	77	61	766	23	32	30	4	0
5	8	0	68	182	40	645	23	30	1	3
6	5	3	54	66	14	16	832	5	4	1
7	5	0	30	46	62	49	1	802	1	4
8	23	10	17	24	6	7	7	4	879	23
9	11	53	14	28	4	16	7	9	23	835

2.7.2 Đánh giá phân tích:

Automobile dự đoán chính xác nhất 909/1000 cho biết Ô TÔ ít bị nhầm

Dự đoán mèo xéo ít chính xác nhất 594/1000 cho biết nhiều khả năng bị nhầm.

{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}

predict	0	1	2	3	4	5	6	7	8	9
label										
0	812	13	43	17	17	12	19	9	36	22
1	5	909	5	7	3	8	12	1	11	39
2	46	0	734	40	63	46	54	13	3	1
3	9	5	59	594	54	175	73	19	5	7
4	11	2	69	59	751	33	36	36	2	1
5	4	1	30	156	30	734	19	23	2	1
6	5	4	32	29	18	20	887	1	2	2
7	6	2	24	46	47	62	6	797	1	9
8	44	13	16	18	8	10	12	4	862	13
9	16	68	7	32	3	16	10	7	19	822

{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}

=> dự đoán mèo là chó nhiều nhất, 175

=> dự đoán chó là mèo nhiều thứ ba, 156, mèo và chó rất dễ nhầm lẫn

predict	0	1	2	3	4	5	6	7	8	9
label										
0	812	13	43	17	17	12	19	9	36	22
1	5	909	5	7	3	8	12	1	11	39
2	46	0	734	40	63	46	54	13	3	1
3	9	5	59	594	54	175	73	19	5	7
4	11	2	69	59	751	33	36	36	2	1
5	4	1	30	156	30	734	19	23	2	1
6	5	4	32	29	18	20	887	1	2	2
7	6	2	24	46	47	62	6	797	1	9
8	44	13	16	18	8	10	12	4	862	13
9	16	68	7	32	3	16	10	7	19	822

-CIFAR-10 gần như được chia thành hai loại

+Động vật: 2, 3, 4, 5, 6, 7

+ Loại xe: 0, 1, 8, 9

- Động vật hiếm khi bị nhầm lẫn là ô tô và XE TẢI

3. Lưu lịch sử đào tạo và mô hình

3.1 Lưu lịch sử đào tạo

```
#Sử dụng Pickle để lưu các đối tượng
import pickle
filename = '/content/drive/MyDrive/Tài liệu năm 3/kỳ 2 năm 3/Nhập môn TTNT/cifar10/train_history.pickle'
pickle.dump(train_history, open(filename, 'wb'))

INFO:tensorflow:Assets written to: ram://57409956-27d8-451c-bca7-30c80dfdf58f/assets

loaded_model = pickle.load(open(filename, 'rb'))
print(loaded_model)

<keras.callbacks.History object at 0x7fb860288790>
```

Thư mục

Tên ↑

image

Tập



```
print(train_history)
print(train_history.epoch)
print(train_history.history)
print(train_history.model)
a=train_history

D <keras.callbacks.History object at 0x7fb8602ef400>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
{'loss': [1.9366275072097778, 1.5327831506729126, 1.3319953080038452, 1.180181622505180, 1.0580466985702515, 0.9658010001997375, 0.8803837895393372, 0.8270636796951294, 0.7639394998550415, 0.7096235752105713, 0.658313155174255]}
<keras.engine.sequential.Sequential object at 0x7fb860704690>
```

3.2 Lưu mô hình

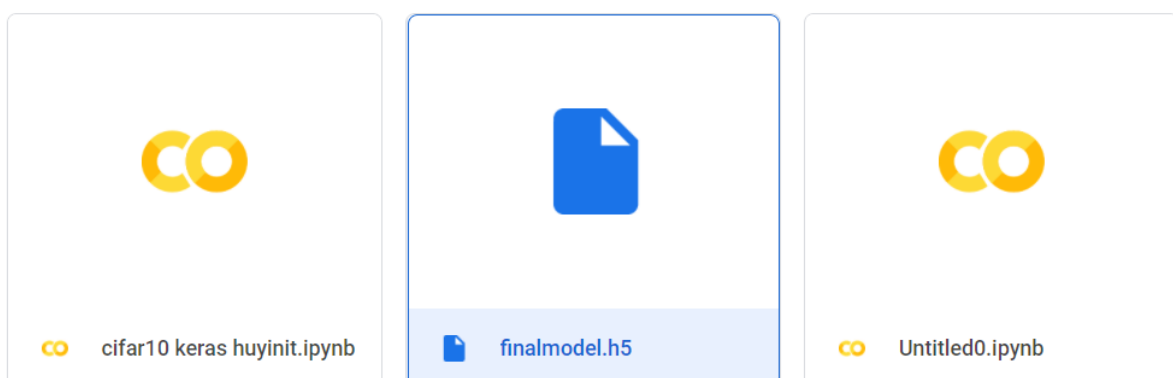
-Trong phần thứ ba, đào tạo chương trình mất nhiều thời gian, khi chạy các mô hình phức tạp hơn và tập dữ liệu hình ảnh lớn hơn, thường mất mười giờ hoặc vài ngày. Đôi khi máy tính có thể bị sập vì lý do nào đó, vì vậy phần đào tạo trước đó sẽ bị mất. Giải pháp là lưu mô hình sau mỗi lần thực hiện chương trình. Trước khi đào tạo chương trình tiếp theo, tải trọng lượng mô hình trước khi tiếp tục đào tạo.

```
path='/content/drive/MyDrive/Tài liệu năm 3/kỳ 2 năm 3/Nhập môn TTNT/cifar10/finalmodel.h5'
model.save(path) #lưu mô hình
```

Thư mục

image

Tập



4. Đẩy mô hình lên web và thử nghiệm:

4.1 Giới thiệu sơ qua về fast API:

Python FastAPI là gì?

FastAPI là nền tảng thiết kế, lập trình xây dựng API cực kỳ nhanh trên cả 2 phương diện phát triển và thực thi trên Python 3.6+.

Từ Python 3.6+ thì bạn đã có thể sử dụng cú pháp await/async để chạy code bất đồng bộ, vì lý do này các framework trên Python sẽ đạt được hiệu năng cao, FastAPI là một trong số Python framework nhanh nhất hiện nay.

Lý do chọn FastAPI

FastAPI là một micro framework khá mới, chỉ vừa được release năm 2018. Github của framework này hiện tính đến tháng 1/2021 đang đạt 25,4k star, tuy nhiên do được áp dụng khá nhiều công nghệ mới nên có FastAPI có vài điểm mạnh mà mình cảm thấy khá phù hợp để sử dụng phát triển project:

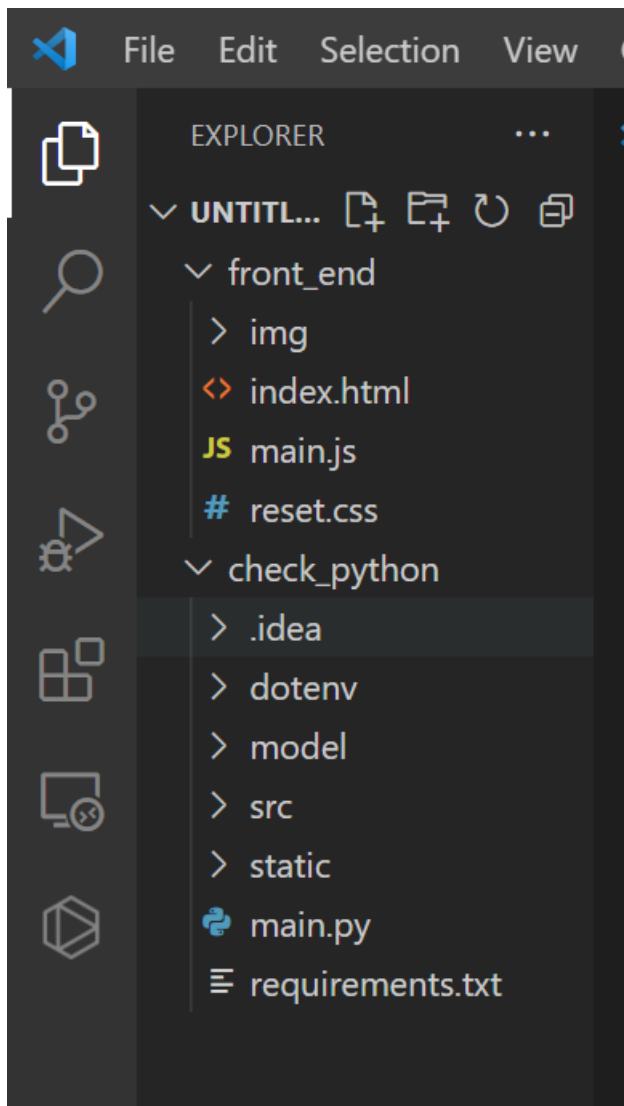
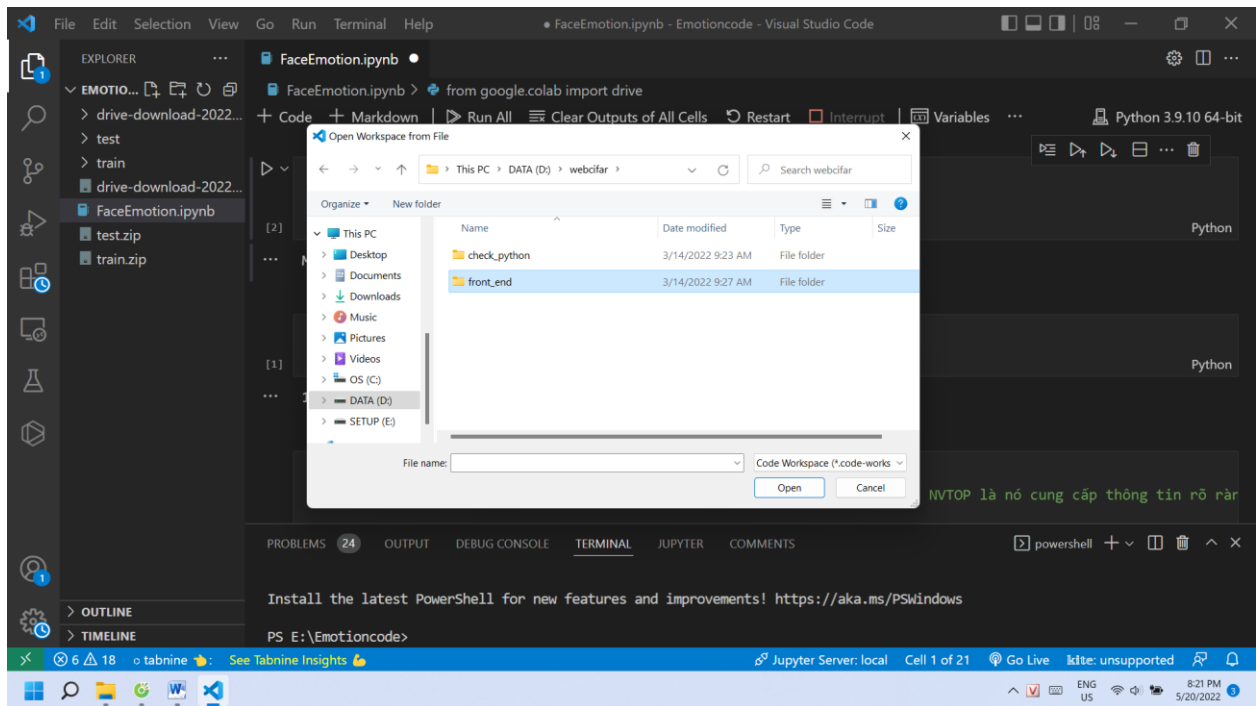
High performance Do được base trên 2 lib khá mạnh ở thời điểm hiện tại của python là Pydantic và Starlette nên FastAPI sở hữu hiệu suất cao nhất trong tất cả các framework Python hiện nay, bạn có thể tham khảo so sánh hiệu năng giữa các framework tại <https://www.techempower.com/>

Development Speed Được hỗ trợ tích hợp sẵn giao diện Swagger – OpenAPI kèm theo cách code khá đơn giản nên lập trình có thể release function rất nhanh mà vẫn có document đầy đủ, đây là lợi thế có thể nói là quan trọng nhất của FastAPI so với các Framework khác. Dưới đây là một đoạn code in ra dòng text healthcheck

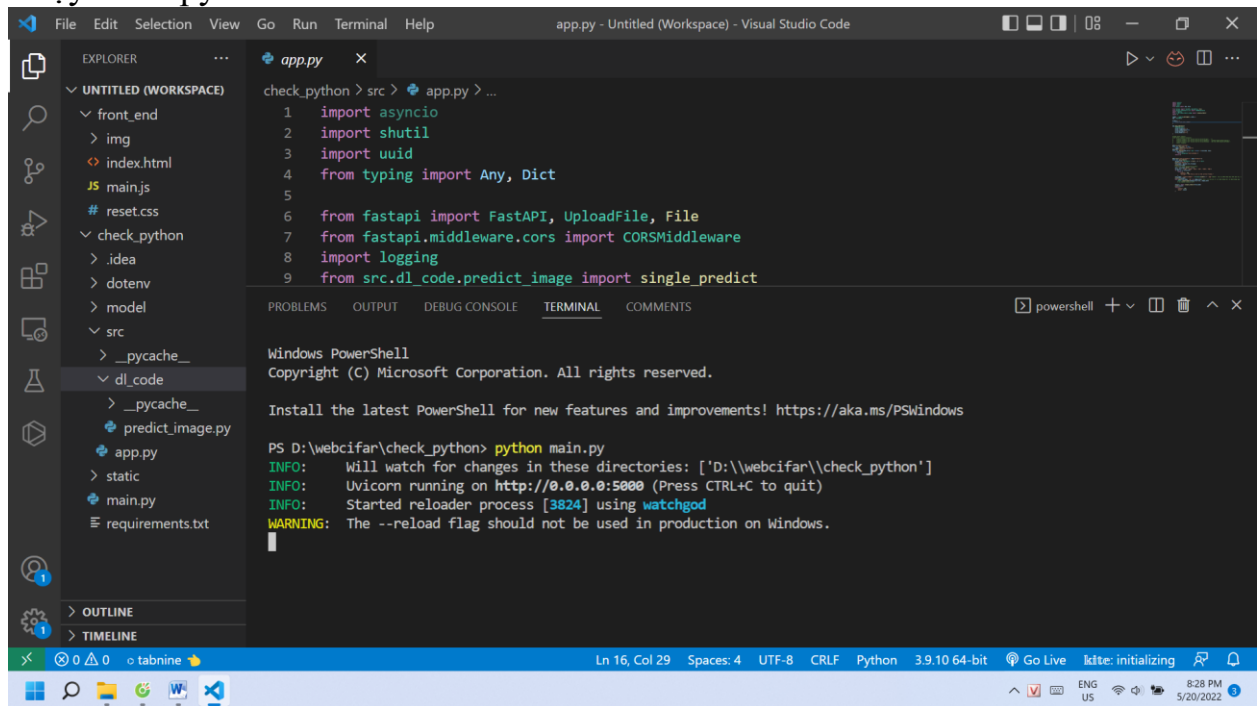
Bất đồng bộ Hiện tại bất đồng bộ đã được hỗ trợ từ phiên bản Django 3.x nhưng ngay từ khi release, FastAPI mặc định đã hỗ trợ developer Async, cũng vì vậy mà FastAPI chỉ có thể sử dụng với python 3.6 trở lên

4.2 Hướng dẫn chạy project

Mở 2 folder tại 2 workspace khác nhau:



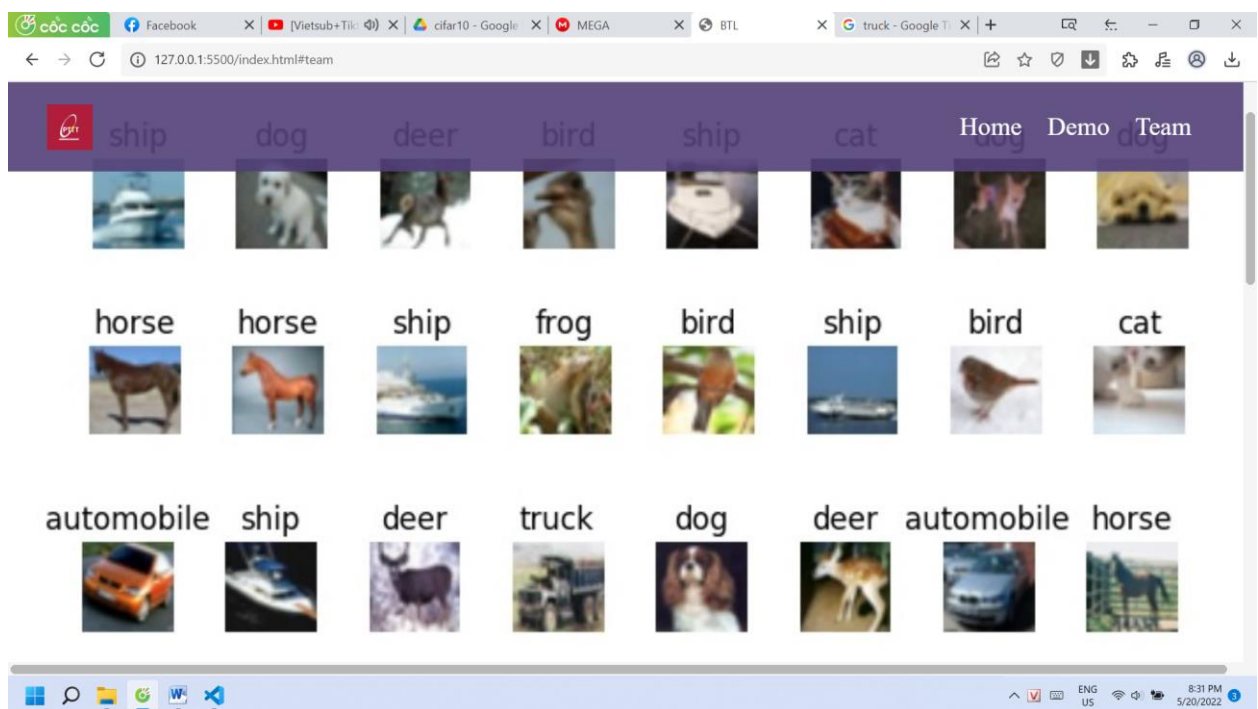
Chạy main.py bên backend:



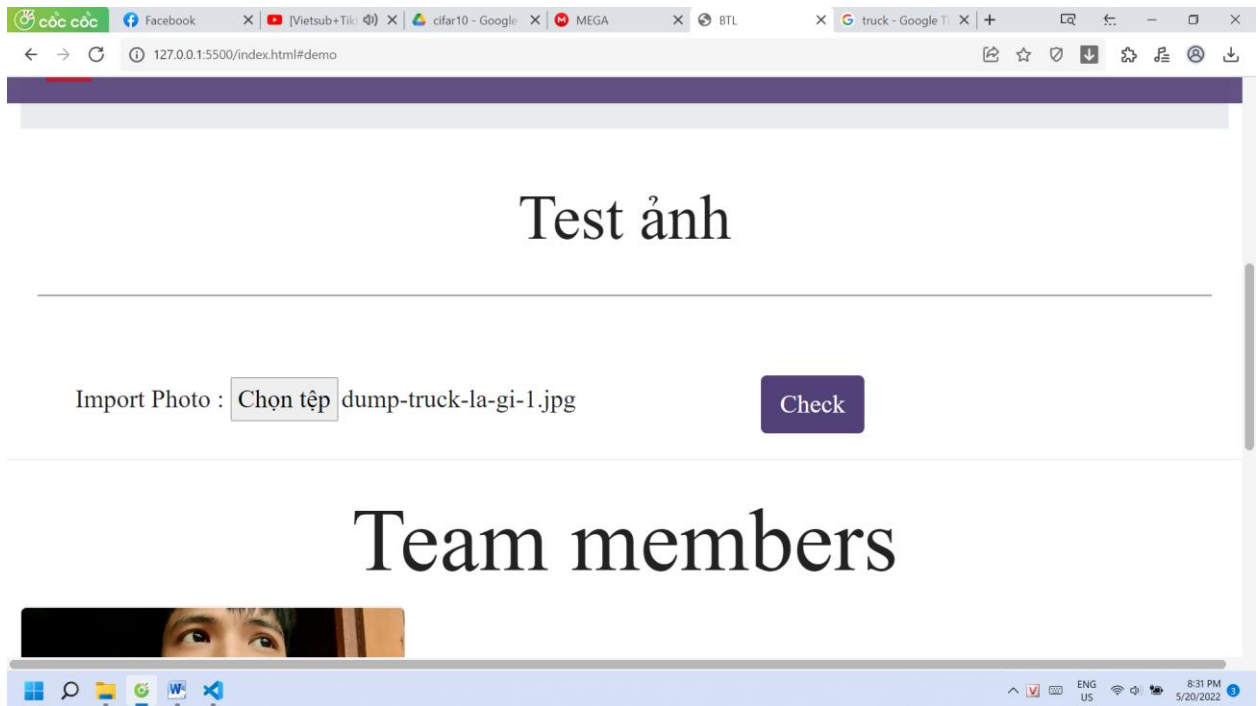
The screenshot shows the Visual Studio Code interface with a Python file named `app.py` open. The code uses `FastAPI` and `uvicorn` to run a web server. The terminal shows the command `python main.py` being executed, and the output indicates that the server is running on `http://0.0.0.0:5000` using `watchgod` for reloading. The file explorer on the left shows the project structure, including `front_end`, `check_python`, `src`, and `dl_code`.

Open in live server bên front end:

Hiện thị giao diện :

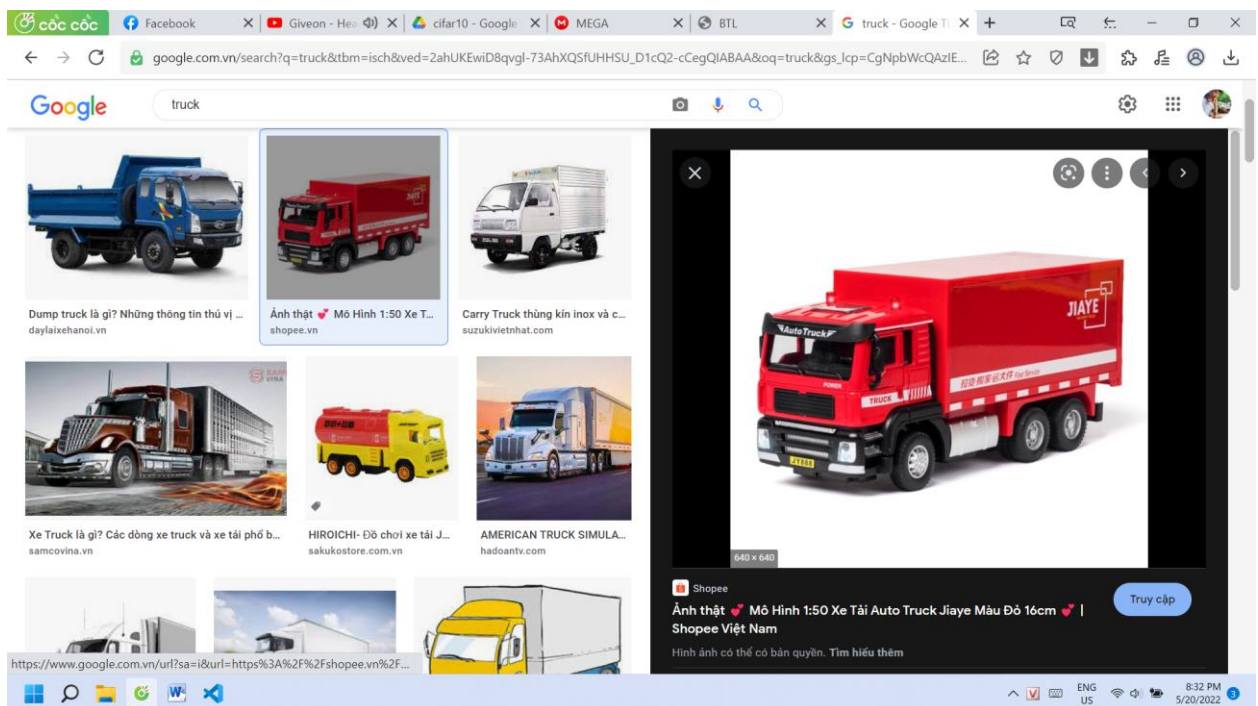


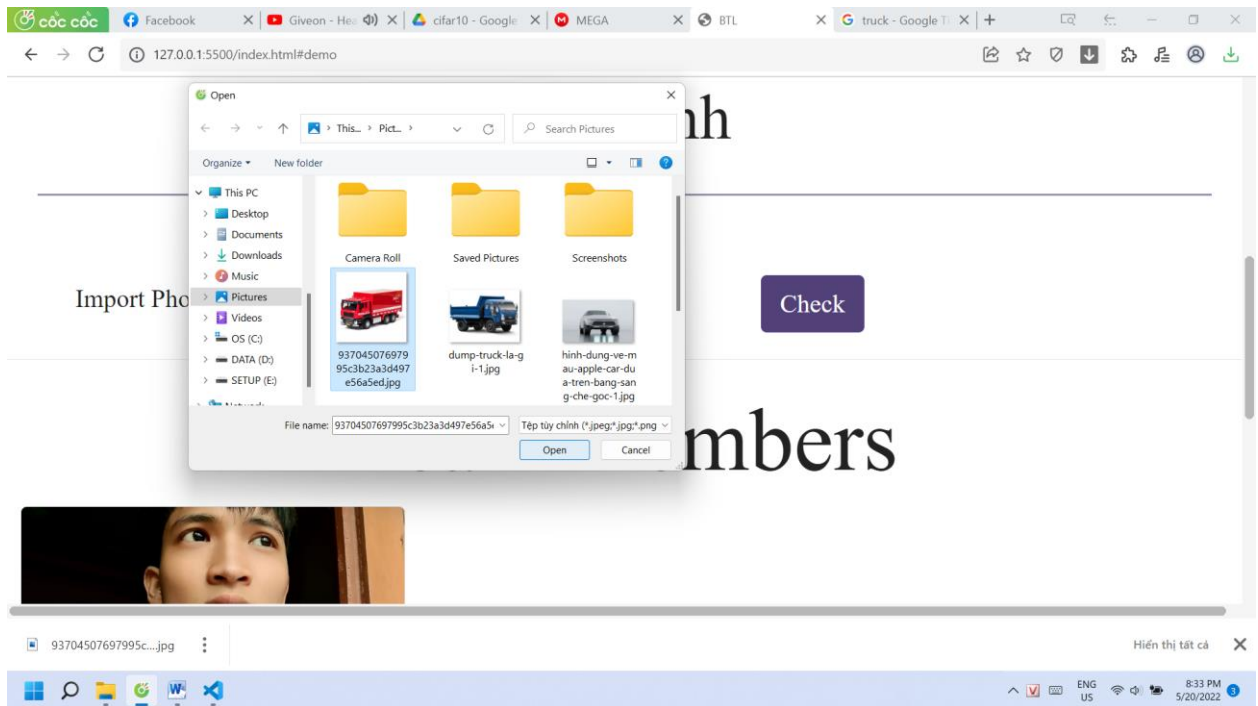
Xuống phần demo :



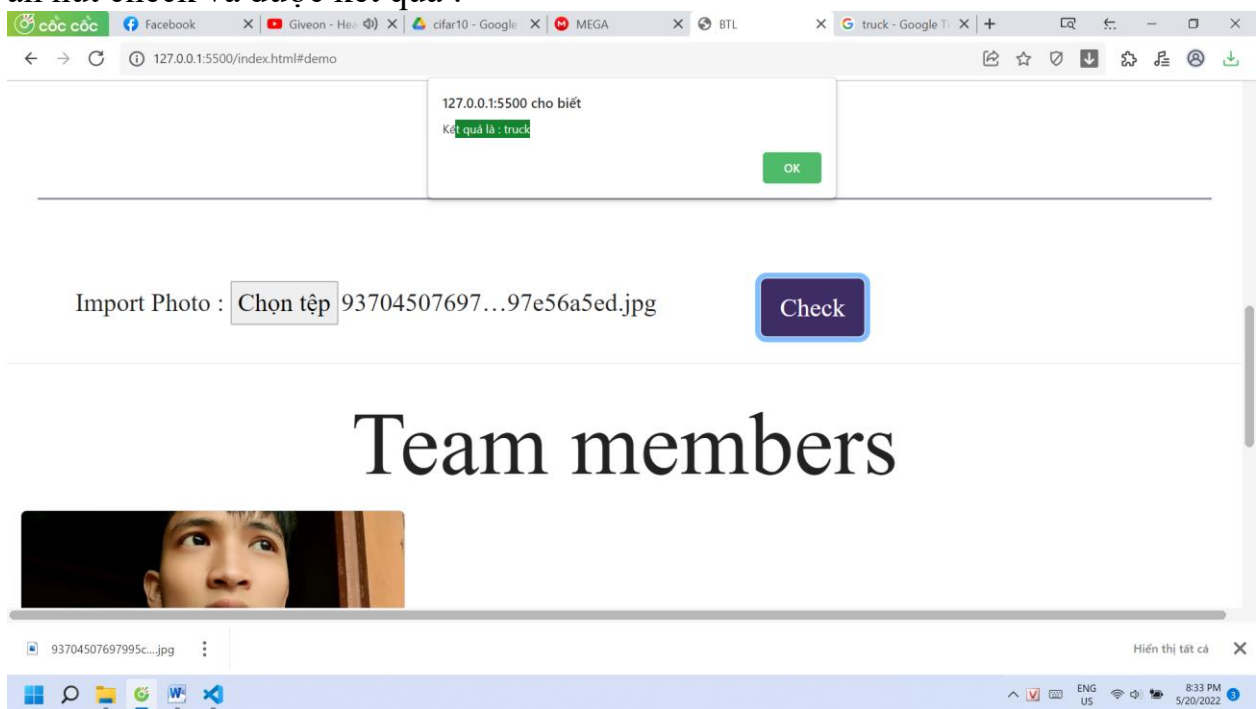
4.3 . Test thử nghiệm

Test thử 1 ảnh thứ nhất :

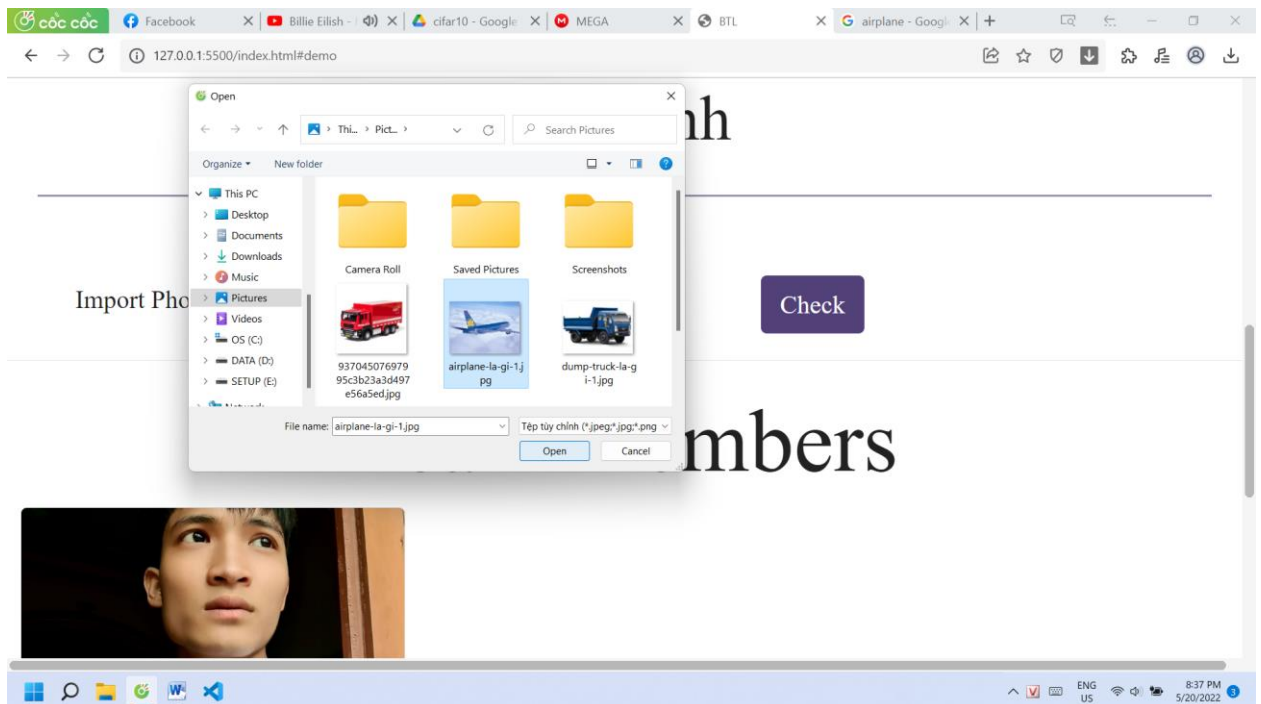
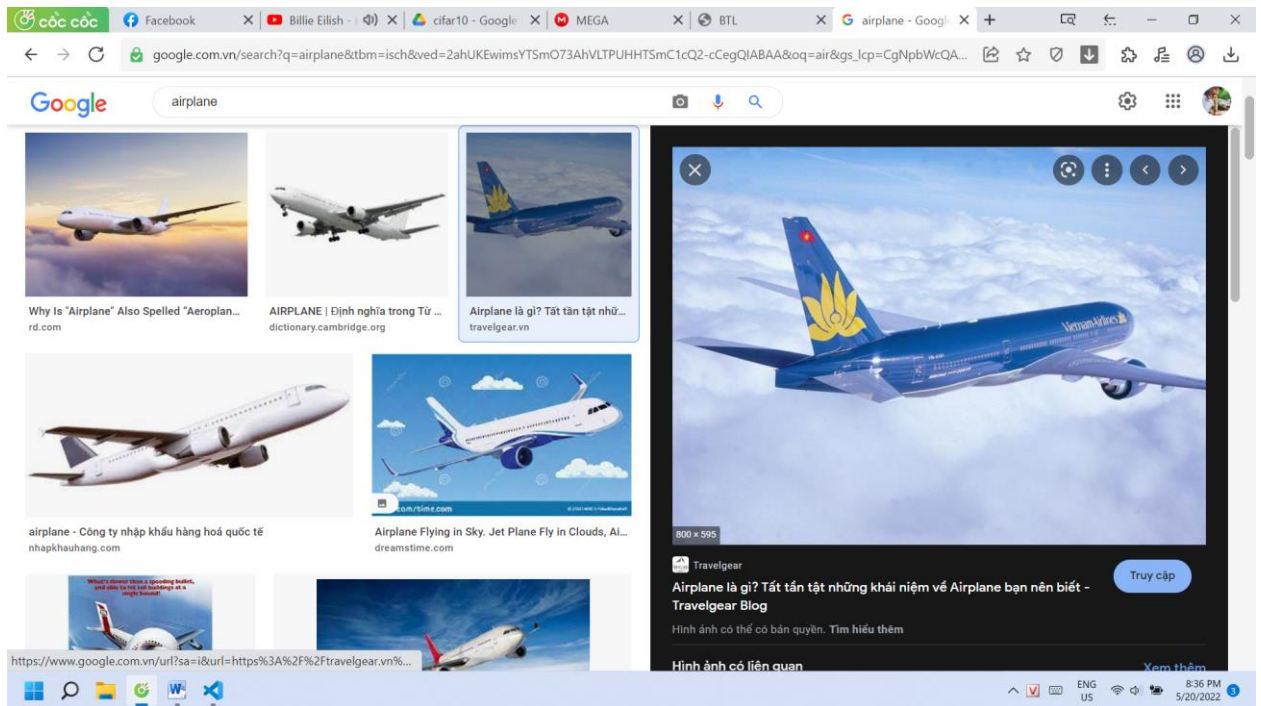


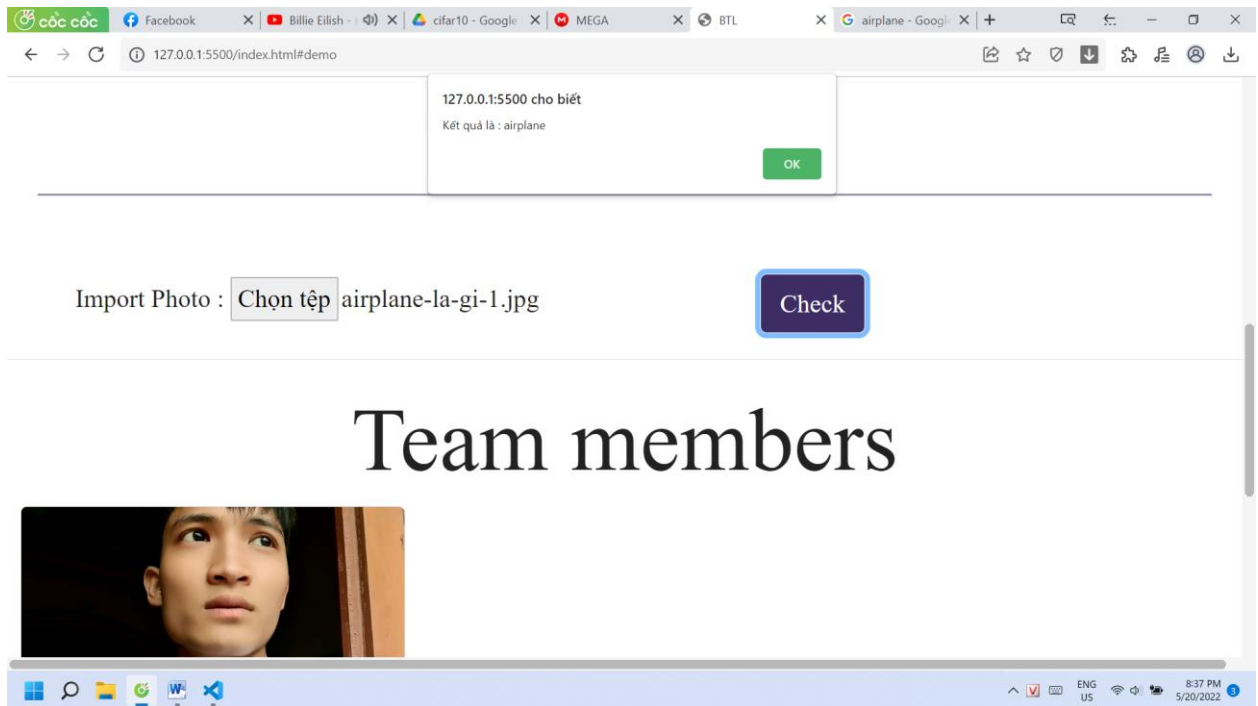


ấn nút check và được kết quả :



Test thử ảnh thứ 2 :





Test thử ảnh thứ 3:

