

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
KHOA CÔNG NGHỆ THÔNG TIN I**



**BÁO CÁO**

**NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI: SỬ DỤNG CNN ÁP DỤNG CHO BÀI  
TOÁN PHÂN LOẠI ẢNH CHÓ MÈO**

**Nhóm học phần:** 03

**Giảng viên:** Đào Thị Thúy Quỳnh

**Thành viên nhóm:**

Nguyễn Văn Khanh B21DCCN449

Dương Văn Dự B21DCCN233

Lê Minh Phụng B21DCCN596

**Hà Nội, 2024**

# MỤC LỤC

LỜI MỞ ĐẦU .....	3
I. Tìm hiểu về mạng tích chập CNN (Convolutional Neural Network).....	4
1. Mạng CNN(Convolutional Neural Network) là gì ?.....	4
1.1 Giới thiệu .....	4
2. Các thành phần chính của một mạng CNN (Convolutional Neural Network).....	5
2.1 Lớp đầu vào – Input Layer .....	5
2.2 Lớp tích chập - Convolution Layer .....	5
2.3 Lớp Relu.....	8
2.4 Lớp Pooling.....	9
2.6 Lớp đầu ra – Output Layer .....	12
3. Cấu trúc của mạng CNN là gì? .....	13
4. Một số cách lựa chọn tham số CNN .....	15
II. Nhận dạng hình ảnh chó mèo sử dụng CNN .....	16
1. Vấn đề bài toán .....	16
2. Hướng giải quyết .....	16
3. Một số thư viện được dùng trong bài toán .....	20
3.1. TensorFlow .....	20
3.2 Scikit-learn .....	20
3.3 Keras .....	20
3.4 Numpy .....	21
3.5 Pandas.....	21
III. Code và chạy chương trình .....	21
IV. Tổng kết .....	30
Tài liệu tham khảo: .....	32

## LỜI MỞ ĐẦU

Trí tuệ nhân tạo (Artificial Intelligence - AI) là một lĩnh vực rộng lớn trong khoa học máy tính và có vai trò quan trọng trong cuộc cách mạng công nghiệp 4.0. Trong những năm gần đây, các mô hình học máy và học sâu đã đạt được những thành tựu đáng kể trong việc giải quyết các bài toán phức tạp trong nhiều lĩnh vực, từ y học, tài chính, giáo dục cho đến sản xuất. AI đã thay đổi cách thức chúng ta tương tác với công nghệ, từ những trợ lý ảo đến hệ thống giám sát tự động và tự động hóa quy trình sản xuất. Tuy nhiên, để phát triển các ứng dụng AI đòi hỏi sự kết hợp giữa kiến thức về lý thuyết và các kỹ năng thực tiễn, đặc biệt là trong việc xử lý dữ liệu lớn và tạo ra các mô hình học máy chính xác và hiệu quả.

Xử lý hình ảnh là một lĩnh vực rộng lớn trong trí tuệ nhân tạo và được ứng dụng trong nhiều lĩnh vực, từ y học cho đến công nghiệp sản xuất. Với sự phát triển của deep learning, các mô hình học sâu trên dữ liệu hình ảnh đã đạt được những kết quả ấn tượng trong việc xử lý và phân tích hình ảnh. Một số ứng dụng của xử lý hình ảnh bao gồm nhận dạng đối tượng, phân loại hình ảnh, phát hiện khuôn mặt và xử lý ảnh y học. Để bắt đầu làm việc với xử lý hình ảnh, ta cần có kiến thức về lý thuyết cơ bản và các công cụ phần mềm để thực hiện các phép tính và xử lý trên hình ảnh.

Nhận thấy việc quan trọng của vấn đề này nhóm chúng em dưới sự giảng dạy của cô Đào Thị Thúy Quỳnh đã quyết định chọn đề tài: “Hệ thống nhận dạng hình ảnh chó mèo” để làm báo cáo môn học. Mặc dù đã có nhiều cố gắng nhưng với khả năng và thời gian còn hạn chế nên đề tài của nhóm em không tránh khỏi sai sót cũng như còn nhiều vấn đề chưa được đề cập tới. Em mong cô và các bạn chúng ta góp ý để báo cáo của nhóm em được hoàn thiện hơn.

**Hà Nội, ngày 4 tháng 5 năm 2024**

# I. Tìm hiểu về mạng tích chập CNN (Convolutional Neural Network)

1. Mạng CNN(Convolutional Neural Network) là gì ?

## 1.1 Giới thiệu

- Mạng CNN (Convolutional Neural Network) là một kiến trúc mạng nơ-ron sâu được sử dụng phổ biến trong các bài toán liên quan đến xử lý ảnh, âm thanh và video. Mạng CNN được thiết kế để tự động học các đặc trưng đặc biệt của dữ liệu thông qua các lớp tích chập và pooling.

- CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định (Ví dụ: Chó, Mèo, Hổ, ...). Máy tính coi hình ảnh đầu vào là 1 mảng pixel và nó phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy  $H \times W \times D$  (H: Chiều cao, W: Chiều rộng, D: Độ dày(tùy tài liệu có thể ghi là C: Channel)). Ví dụ:

color image is 3rd-order tensor

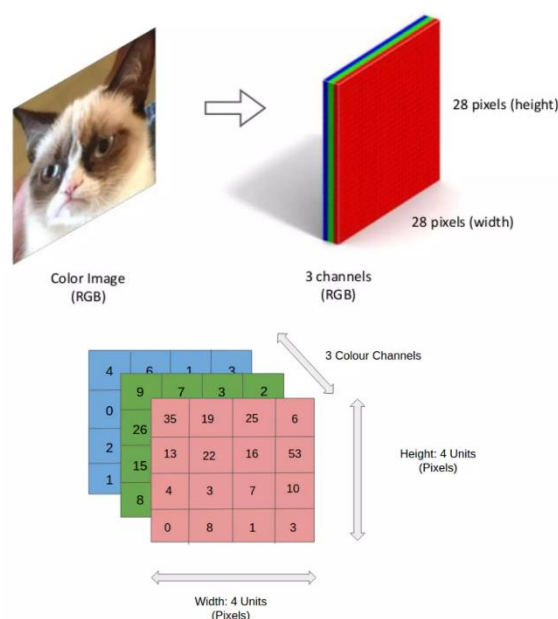
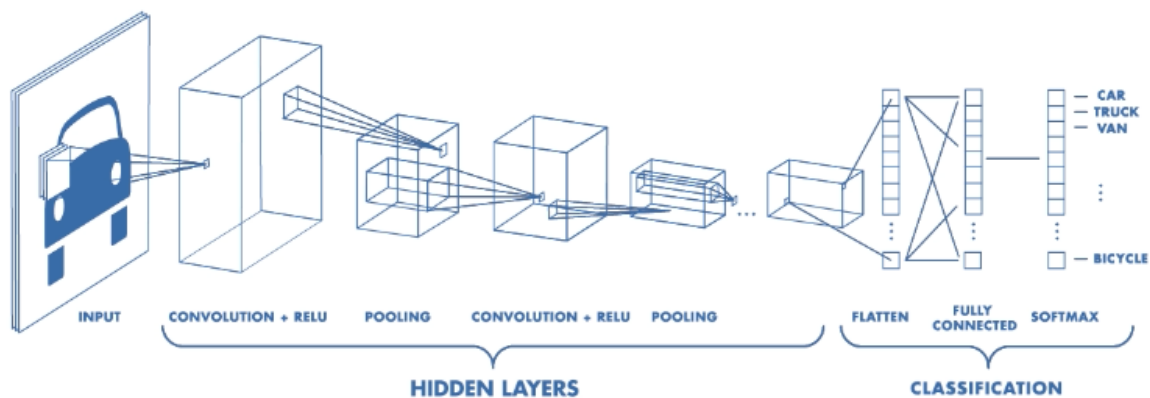


Figure 1. Hình ảnh minh họa

- Về kỹ thuật, mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernels), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1. Hình dưới đây là toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị:



## 2. Các thành phần chính của một mạng CNN (Convolutional Neural Network)

### 2.1 Lớp đầu vào – Input Layer

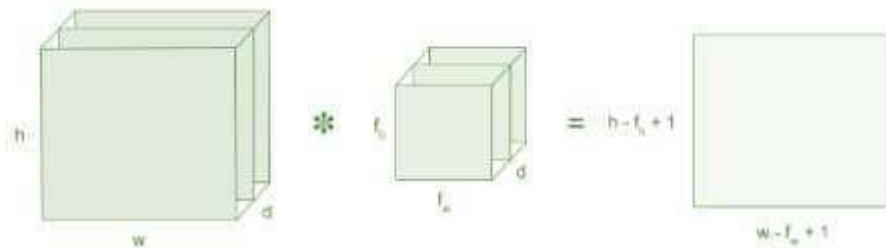
- Input Layer (lớp đầu vào) là lớp đầu tiên trong một mạng neural, nó chứa dữ liệu đầu vào của mạng, ví dụ như ảnh, âm thanh hoặc văn bản. Input layer định nghĩa kích thước và định dạng của dữ liệu đầu vào và truyền nó đến các lớp tiếp theo trong mạng. Trong mạng CNN (Convolutional Neural Network), input layer là ảnh đầu vào hoặc một tập hợp các ảnh đầu vào có kích thước nhất định. Mỗi ảnh được biểu diễn dưới dạng một ma trận các pixel, với mỗi pixel có giá trị tương ứng cho mức độ sáng tại vị trí đó trên ảnh.
- Khi ảnh được truyền qua input layer, nó được xử lý bởi các lớp tiếp theo trong mạng, bao gồm các lớp tích chập (convolutional layer), lớp kích hoạt (activation layer), lớp giảm mẫu (pooling layer) và các lớp fully connected layer. Các tham số và trọng số của mạng sẽ được học và tối ưu hóa trong quá trình huấn luyện mạng.
- Input layer cũng có thể được sử dụng trong các mạng neural khác, ví dụ như mạng neural thần kinh tiêu chuẩn (feedforward neural network) hoặc mạng neural hồi quy (recurrent neural network). Các loại dữ liệu đầu vào khác nhau như âm thanh hoặc văn bản có thể có định dạng và xử lý khác nhau trong input layer tùy thuộc vào bài toán mà mạng được sử dụng để giải quyết.

### 2.2 Lớp tích chập - Convolution Layer

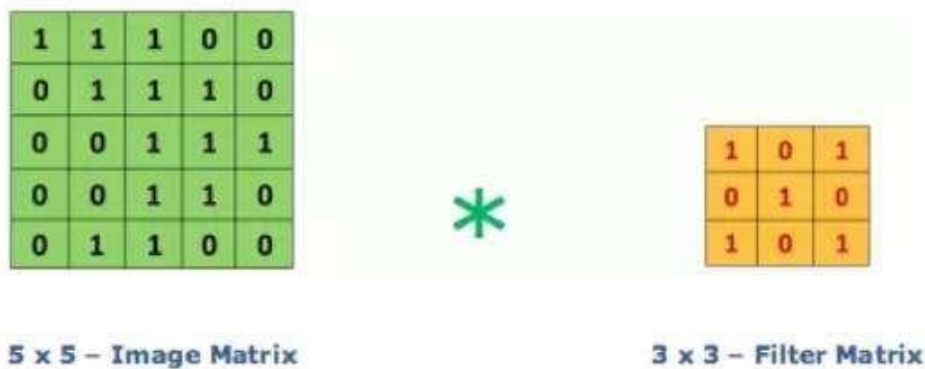
- Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vuông nhỏ của dữ liệu đầu vào.

Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân.

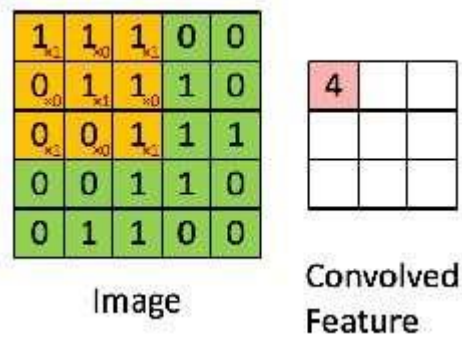
- An image matrix (volume) of dimension  **$(h \times w \times d)$**
- A filter  **$(f_h \times f_w \times d)$**
- Outputs a volume dimension  **$(h - f_h + 1) \times (w - f_w + 1) \times 1$**







- Xem xét 1 ma trận  $5 \times 5$  có giá trị pixel là 0 và 1. Ma trận bộ lọc  $3 \times 3$  như hình bên dưới.



- Sau đó, lớp tích chập của ma trận hình ảnh  $5 \times 5$  nhân với ma trận bộ lọc  $3 \times 3$  gọi là 'Feature Map' như hình bên dưới.



Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

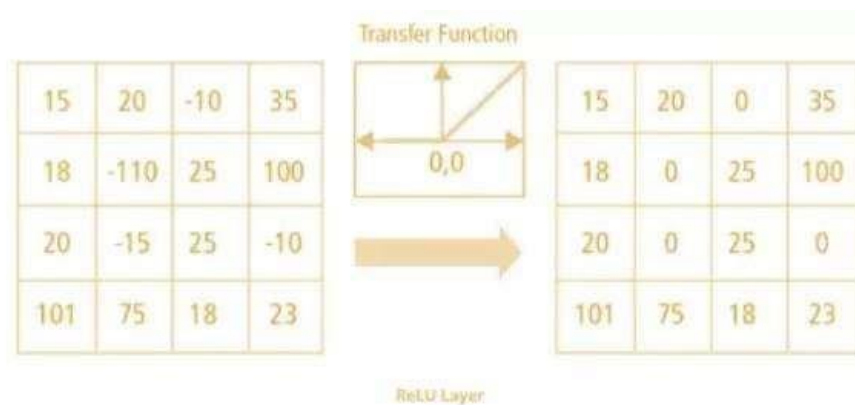
Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

### 2.3 Lớp Relu

- Lớp ReLU (Rectified Linear Unit) là một trong những lớp kích hoạt phổ biến được sử dụng trong các mạng neural, bao gồm cả mạng CNN (Convolutional Neural Network). Lớp ReLU có tác dụng chuyển đổi giá trị đầu vào thành giá trị đầu ra mới bằng cách áp dụng hàm ReLU cho mỗi phần tử trong ma trận đầu vào.
- Hàm ReLU được định nghĩa như sau:  $f(x) = \max(0, x)$



- Trong đó,  $x$  là giá trị đầu vào và  $\max(0, x)$  là hàm trả về giá trị lớn nhất giữa 0 và  $x$ . Điều này có nghĩa là nếu giá trị đầu vào là âm, thì giá trị đầu ra của hàm ReLU sẽ bằng 0, còn nếu giá trị đầu vào là dương, thì giá trị đầu ra của hàm ReLU sẽ bằng chính giá trị đầu vào đó.
- Lớp ReLU được sử dụng trong mạng CNN để kích hoạt các tính năng quan trọng trong ảnh như cạnh, góc, đường cong, v.v... Các giá trị âm trong đầu vào sẽ được chuyển thành 0, giúp giảm thiểu hiện tượng quá khớp (overfitting) trong quá trình huấn luyện mạng. Ngoài ra, lớp ReLU còn giúp tăng tốc độ huấn luyện và làm giảm độ phức tạp tính toán của mạng.
- Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.



- Ví dụ, lớp ReLU được áp dụng sau lớp tích chập trong mạng CNN để tạo ra các bản đồ đặc trưng (feature maps) và kích hoạt các tính năng quan trọng trong ảnh. Các giá trị âm trong bản đồ đặc trưng sẽ được chuyển thành 0, trong khi các giá trị dương sẽ được giữ nguyên, tạo ra các kích hoạt đáng chú ý cho các lớp tiếp theo trong mạng. Có 1 số hàm phi tuyến khác như *tanh*, *sigmoid* cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

## 2.4 Lớp Pooling

- Lớp Pooling là một trong những lớp quan trọng trong kiến trúc của mạng CNN (Convolutional Neural Network). Lớp này có chức năng giảm kích thước của bản đồ đặc trưng (feature map) được tạo ra bởi lớp tích chập (convolution layer) trong quá trình trích xuất đặc trưng của ảnh.
- Các phương pháp pooling phổ biến bao gồm Max Pooling và Average

Pooling. Trong Max Pooling, một cửa sổ trượt (sliding window) được áp dụng lên bản đồ đặc trưng và chỉ giữ lại giá trị lớn nhất trong từng vùng. Trong Average Pooling, giá trị trung bình của từng vùng được tính toán và giữ lại làm giá trị đại diện cho vùng đó.

- Mục đích của lớp pooling là giảm kích thước của bản đồ đặc trưng, giúp giảm độ phức tạp tính toán của mạng và tránh hiện tượng quá khớp (overfitting) trong quá trình huấn luyện. Ngoài ra, lớp pooling còn giúp tăng tính bất biến với việc dịch chuyển (translation invariance) của ảnh, tức là với một ảnh được dịch chuyển một chút, các giá trị pooling vẫn giữ nguyên.

- Ví dụ, sau khi áp dụng lớp tích chập cho ảnh đầu vào, ta sẽ được các bản đồ đặc trưng có kích thước lớn. Khi đó, ta có thể áp dụng lớp Max Pooling để giảm kích thước của bản đồ đặc trưng này và lấy ra các giá trị đại diện quan trọng nhất của vùng đó. Điều này giúp giảm độ phức tạp tính toán và giúp mạng CNN học được các đặc trưng chung của ảnh.

- Đây là một ví dụ về việc xây dựng một mô hình CNN đơn giản bằng thư viện Keras trên Python để phân loại bức ảnh thành 10 lớp sử dụng lớp tích chập và lớp pooling:

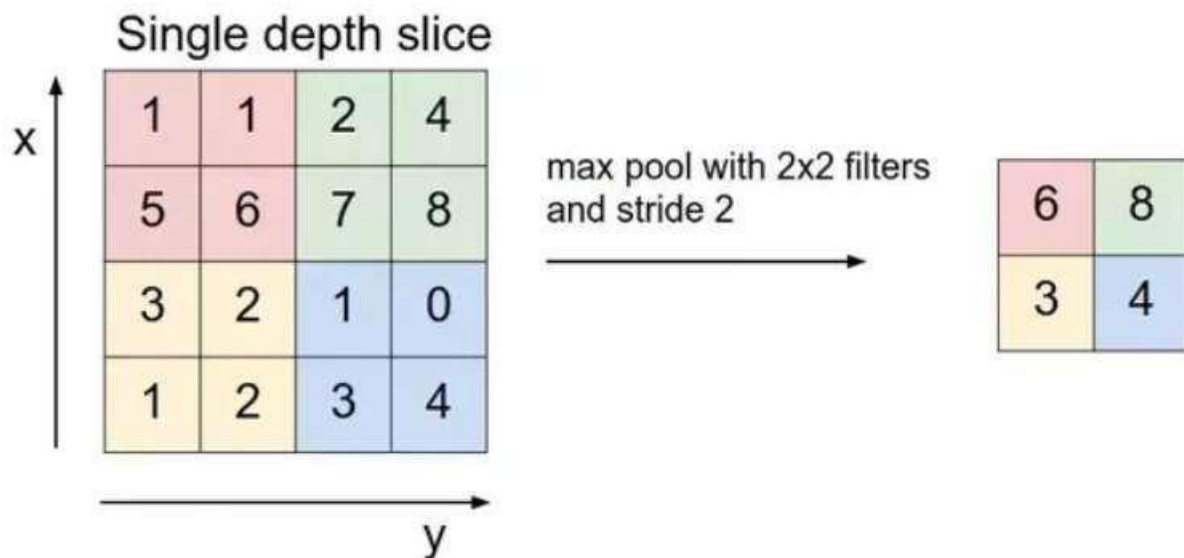
```
# Khởi tạo thư viện
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Xây dựng mô hình CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Biên dịch mô hình
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
# Huấn luyện mô hình
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

- Ở đây, chúng ta sử dụng lớp Conv2D để thêm các lớp tích chập với số

lượng bộ lọc và kích thước bộ lọc được chỉ định. Chúng ta cũng sử dụng lớp MaxPooling2D để thêm các lớp pooling để giảm kích thước của các bản đồ đặc trưng. Sau đó, chúng ta sử dụng lớp Flatten để chuyển đổi các bản đồ đặc trưng thành một vector và sử dụng các lớp kết nối đầy đủ (Dense) để phân loại ảnh. Cuối cùng, chúng ta sử dụng hàm compile để biên dịch mô hình và hàm fit để huấn luyện mô hình trên dữ liệu đào tạo và kiểm tra.

Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:

- Max Pooling
- Average Pooling
- Sum Pooling
- Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling.



## 2.5 Lớp kết nối đầy đủ - Fully connected layer

- Lớp kết nối đầy đủ (fully connected layer) là một lớp trong mạng nơ-ron

nhân tạo, trong đó mỗi nơ-ron ở lớp trước sẽ được kết nối đến tất cả các nơ-ron ở lớp sau. Lớp này được sử dụng để kết nối các đặc trưng đầu vào và trích xuất được từ lớp trước đó với đầu ra mong muốn.

- Ví dụ, nếu chúng ta sử dụng mạng nơ-ron để phân loại ảnh, thì lớp đầu tiên của mạng sẽ là một lớp tích chập (convolutional layer) để trích xuất các đặc trưng của ảnh. Sau đó, các đặc trưng này sẽ được kết nối đến lớp kết nối đầy đủ để giảm kích thước và trích xuất các đặc trưng chung của ảnh. Cuối cùng, lớp đầu ra sẽ thực hiện phân loại dựa trên đầu vào từ lớp kết nối đầy đủ. Lớp kết nối đầy đủ thường được sử dụng trong các mô hình mạng nơ-ron sâu (deep neural networks) và là một phần quan trọng của các mô hình CNN (Convolutional Neural Networks). Lớp này giúp kết nối tất cả các đặc trưng của đầu vào và tạo ra một đầu ra đầy đủ thông tin để thực hiện tác vụ phân loại, dự đoán hoặc nhận dạng đối tượng.
- Dưới đây là một ví dụ về cách sử dụng lớp kết nối đầy đủ trong thư viện Keras trên Python:

```
#Khởi tạo thư viện
from keras.models import Sequential
from keras.layers import Dense
# Xây dựng mô hình mạng nơ-ron
model = Sequential()
model.add(Dense(64, input_dim=100, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Biên dịch mô hình
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

- Ở đây, chúng ta sử dụng hai lớp kết nối đầy đủ với 64 và 10 nơ-ron tương ứng để phân loại dữ liệu đầu vào vào 10 lớp khác nhau. Lớp đầu tiên trong mô hình được kết nối đầy đủ với đầu vào có kích thước 100 và được kích hoạt bởi hàm ReLU. Cuối cùng, chúng ta sử dụng hàm compile để biên dịch mô hình

## 2.6 Lớp đầu ra – Output Layer

- Lớp đầu ra (output layer) là lớp cuối cùng trong kiến trúc mạng nơ-ron và có trách nhiệm cho phân loại hoặc dự đoán đối tượng của đầu vào dựa trên kết

quả tính toán ở các lớp trước đó. Các loại lớp đầu ra khác nhau được sử dụng cho các mô hình khác nhau, tùy thuộc vào bài toán cụ thể. Ví dụ, trong bài toán phân loại hình ảnh, lớp đầu ra sẽ là một lớp kết nối đầy đủ hoặc lớp tích chập với số lượng nơ-ron hoặc kênh đầu ra tương ứng với số lớp phân loại.

- Lớp đầu ra (output layer) là lớp cuối cùng trong kiến trúc mạng nơ-ron và có trách nhiệm cho phân loại hoặc dự đoán đối tượng của đầu vào dựa trên kết quả tính toán ở các lớp trước đó. Các loại lớp đầu ra khác nhau được sử dụng cho các mô hình khác nhau, tùy thuộc vào bài toán cụ thể. Ví dụ, trong bài toán phân loại hình ảnh, lớp đầu ra sẽ là một lớp kết nối đầy đủ hoặc lớp tích chập với số lượng nơ-ron hoặc kênh đầu ra tương ứng với số lớp phân loại.

- Dưới đây là một ví dụ về cách sử dụng lớp đầu ra trong mô hình mạng nơ-ron sâu (deep neural network) trong thư viện Keras trên Python:

```
# Khởi tạo thư viện
from keras.models import Sequential
from keras.layers import Dense
# Xây dựng mô hình mạng nơ-ron
model = Sequential()
model.add(Dense(64, input_dim=100, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Biên dịch mô hình
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Huấn luyện mô hình
model.fit(X_train, y_train, epochs=10, batch_size=32)
# Đánh giá mô hình
loss, accuracy = model.evaluate(X_test, y_test)
```

- Ở đây, chúng ta sử dụng hai lớp kết nối đầy đủ với 64 và 10 nơ-ron tương ứng để phân loại dữ liệu đầu vào vào 10 lớp khác nhau. Lớp đầu tiên trong mô hình được kết nối đầy đủ với đầu vào có kích thước 100 và được kích hoạt bởi hàm ReLU. Cuối cùng, chúng ta sử dụng hàm compile để biên dịch mô hình.

### 3. Cấu trúc của mạng CNN là gì?

- Mạng CNN là một trong những tập hợp của lớp Convolution bị chồng lên nhau cũng như sử dụng hàm nonlinear activation như ReLU và tanh để kích hoạt trọng số trong node. Lớp này sau khi thông qua hàm thì sẽ được trọng số trong các node. Những lớp này sau khi đã thông qua hàm kích hoạt thì có thể tạo ra những thông tin trừu tượng hơn cho những lớp tiếp theo.

- Trong mô hình CNN có tính bất biến và tích kết hợp. Nếu như chúng ta có cùng một đối tượng mà lại chiếu theo nhiều góc độ khác nhau thì độ chính xác có thể sẽ bị ảnh hưởng. Với chuyển dịch, quay và co giãn thì pooling layer sẽ được sử dụng để giúp làm bất biến những tính chất này. Vì vậy, CNN sẽ đưa ra kết quả có độ chính xác tương ứng ở từng mô hình.

- Trong đó, pooling layer sẽ cho chúng ta tính bất biến đối với phép dịch chuyển, phép co giãn và phép quay. Còn tính kết hợp cục bộ sẽ cho chúng ta thấy những cấp độ biểu diễn, thông tin từ thấp đến mức độ cao với độ trừu tượng thông qua convolution từ các filter. Mô hình CNN có các layer liên kết được với nhau dựa vào cơ chế convolution.

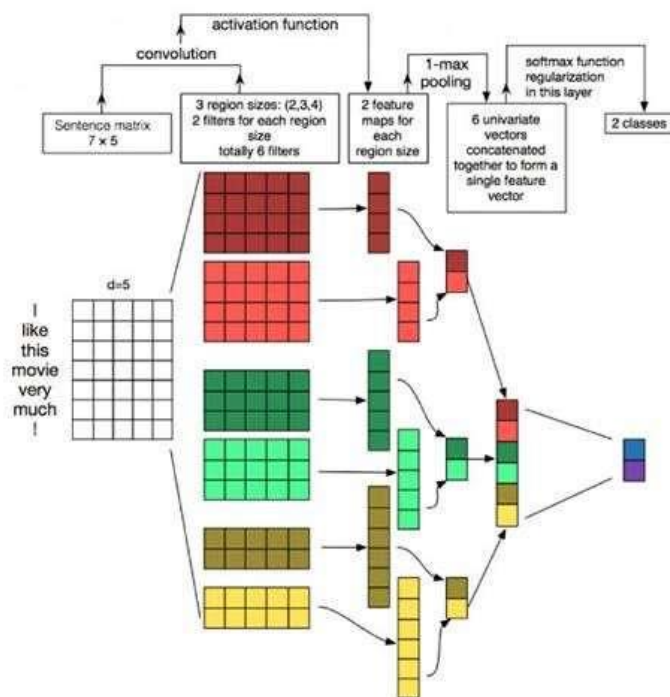
- Những layer tiếp theo sẽ là kết quả từ những convolution từ layer trước đó, vì thế mà chúng ta sẽ có các kết nối cục bộ phù hợp nhất. Vậy, mỗi neuron ở lớp sinh ra tiếp theo từ kết quả filter sẽ áp đặt lên vùng ảnh cục bộ của một neuron có trước đó. Trong khi huấn luyện mạng, CNN sẽ tự động học hỏi các giá trị thông qua lớp filter dựa vào cách thức mà người dùng thực hiện.

Trong đó, cấu trúc cơ bản của CNN thường bao gồm 3 phần chính là:

- **Local receptive field** (trường cục bộ): Lớp này có nhiệm vụ tách lọc dữ liệu, thông tin ảnh và lựa chọn các vùng ảnh có giá trị sử dụng cao nhất.

- **Shared weights and bias** (trọng số chia sẻ): Lớp này giúp làm giảm tối đa lượng tham số có tác dụng chính của yếu tố này trong mạng CNN. Trong mỗi convolution sẽ có các feature map khác nhau và mỗi feature lại có khả năng giúp detect một vài feature trong ảnh.

- **Pooling layer** (lớp tổng hợp): Pooling layer là lớp cuối cùng và có tác dụng làm đơn giản các thông tin đầu ra. Có nghĩa là, sau khi đã hoàn tất tính toán và quét qua các lớp thì đến pooling layer để lược bớt các thông tin không cần thiết. Từ đó, cho ra kết quả theo như ý mà người dùng mong muốn.



*CNN được ứng dụng vô cùng rộng rãi*

#### 4. Một số cách lựa chọn tham số CNN

\*Để lựa chọn được tham số phù hợp nhất cho CNN thì chúng ta cần lưu ý đến các số lượng: filter size, pooling size, số convolution và việc train test

- **Convolution layer:** Nếu lớp này có số lượng nhiều thì chương trình chạy của chúng ta lại 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 chúng ta sẽ đạt được kết quả như mong muốn.
- **Filter size:** Thông thường, các filter size đều có kích thước là 3x3 hoặc 5x5.
- **Pooling size:** Với các hình ảnh thông thường thì chúng ta cần sử dụng kích thước 2x2. Tuy nhiên, nếu đầu vào có hình ảnh lớn hơn thì chúng ta nên sử dụng 4x4.
- **Train test:** Cần thực hiện train test nhiều lần, như vậy thì mới cho ra được các parameter tốt nhất

Thuật toán Convolutional neural network đem đến cho người dùng một mô hình chất lượng. Dù bản chất nó không phải là thuật toán quá đơn giản nhưng lại mang đến kết quả hài lòng. Mặc dù vậy, đây là thuật toán khá khó hiểu và cần trải qua tiếp xúc lâu dài thì người dùng mới có thể ứng dụng nó một cách chính xác nhất. Bởi vì, rất khó để biết và hiểu rõ CNN nếu như là người mới tiếp xúc. Chính vì vậy, muốn áp dụng hiệu quả CNN, nên học hỏi và tìm tòi cũng như bổ sung

nhiều kiến thức hơn cho bản thân.

## II. Nhận dạng hình ảnh chó mèo sử dụng CNN

### 1. Vấn đề bài toán

Thị giác máy tính có nhiều công dụng. Nó có thể nhận dạng khuôn mặt, nó có thể được sử dụng trong kiểm soát chất lượng và bảo mật và nó cũng có thể nhận dạng rất thành công các đối tượng khác nhau trên hình ảnh. Chúng ta sẽ xây dựng một mô hình học máy có giám sát để nhận ra mèo và chó trên hình ảnh bằng cách sử dụng **Mạng thần kinh**.

### 2. Hướng giải quyết

Bước 1: Chuẩn bị dữ liệu:

- Tải xuống tập dữ liệu hình ảnh chó mèo từ các nguồn như ImageNet, Kaggle, Microsoft hoặc tập dữ liệu của chính chúng ta.
- Chia tập dữ liệu thành hai phần, một phần chứa ảnh chó và phần còn lại chứa ảnh mèo. Trong trường hợp này sử dụng DataFrame từ thư viện numpy của python để lưu trữ.
- Chia tập dữ liệu thành tập huấn luyện(70%) và tập kiểm tra(30%).

Bước 2: Xây dựng mô hình CNN:

- Import thư viện TensorFlow và Keras.
- Khởi tạo model Sequential.

\*Trong thư viện Keras của Python, lớp **Sequential** được sử dụng để xây dựng mô hình mạng neural theo kiểu tuyến tính (sequential). Một mô hình sequential được xây dựng từ các lớp (layers) liên tiếp nhau, với đầu ra của mỗi lớp là đầu vào của lớp tiếp theo. Mô hình sequential là phù hợp cho các bài toán mà đầu ra của một lớp chỉ đưa ra một giá trị và đầu vào của lớp tiếp theo là một vector.

- Thêm các lớp Convolutional, MaxPooling, Flatten và Dense dùng để phân loại ảnh

- Convolutional layer: Lớp tích chập giúp tìm các đặc trưng (features) của ảnh bằng cách thực hiện việc trượt một bộ lọc (filter) qua các vùng của ảnh. Mỗi bộ lọc sẽ tìm kiếm các đặc trưng khác nhau của ảnh, ví dụ như cạnh, góc, vết nứt, vân tay, v.v.

- MaxPooling layer: Lớp giảm mẫu (subsampling) giúp giảm kích thước của các feature maps được trích xuất từ lớp Convolutional trước đó bằng cách chọn ra giá trị lớn nhất trong mỗi vùng của feature maps. Flatten layer: Lớp làm phẳng chuyển đổi các feature maps thành một vector duy nhất để đưa vào lớp Dense kết nối đầy đủ (fully connected) sau đó.

- Dense layer: Lớp kết nối đầy đủ giúp phân loại các ảnh bằng cách tính toán trọng số (weights) và độ lệch (biases) của các nơ-ron để dự đoán nhãn của ảnh. Tất cả các lớp này đóng vai trò quan trọng trong việc trích xuất đặc trưng của



ảnh và dự đoán nhãn của nó. Các mô hình CNN có thể có nhiều lớp Convolutional và MaxPooling để tăng độ phức tạp và độ chính xác của mô hình.

- Cấu hình thông số cho mô hình: hàm loss, optimizer và metrics. loss: hàm mất mát được sử dụng để đánh giá độ lỗi của mô hình trong quá trình huấn luyện. Ở đây, ta sử dụng hàm `binary_crossentropy` vì đây là bài toán phân loại nhị phân (chó hoặc mèo).

optimizer: là thuật toán tối ưu hóa được sử dụng để tối thiểu hóa hàm mất mát. Ở đây, ta sử dụng thuật toán adam.

Adam (Adaptive Moment Estimation) là một trong những thuật toán tối ưu hóa phổ biến được sử dụng trong quá trình huấn luyện mô hình Deep Learning, bao gồm cả mô hình CNN. Thuật toán này được coi là kết hợp của hai thuật toán khác là RMSprop và MomentumAdam cải thiện hiệu suất huấn luyện của mô hình bằng cách điều chỉnh tỷ lệ học tự động (adaptive learning rate), tức là tỷ lệ học sẽ được điều chỉnh theo từng tham số riêng biệt của mô hình. Nó cũng sử dụng moment giống như Momentum để tăng tốc độ hội tụ của mô hình. Adam được coi là một trong những thuật toán tối ưu hóa hiệu quả và ổn định cho quá trình huấn luyện mô hình Deep Learning.

metrics: là các độ đo được sử dụng để đánh giá hiệu suất của mô hình. Ở đây, ta sử dụng độ đo accuracy (độ chính xác) để đánh giá hiệu suất của mô hình.

"Accuracy" là một độ đo được sử dụng để đánh giá hiệu suất của thuật toán phân loại trong machine learning. Nó được tính bằng tỉ lệ giữa số lượng các mẫu được phân loại đúng và tổng số lượng các mẫu. Với bài toán phân loại chó mèo, accuracy được tính bằng tỉ lệ giữa số lượng ảnh chó và mèo được phân loại đúng và tổng số lượng ảnh trong tập dữ liệu.

Ví dụ, nếu ta có một mô hình phân loại chó mèo và ta đánh giá nó trên một tập dữ liệu kiểm tra gồm 100 ảnh, trong đó có 60 ảnh là chó và 40 ảnh là mèo. Nếu mô hình phân loại đúng 55 ảnh chó và 35 ảnh mèo, thì accuracy của mô hình là:

$$\text{accuracy} = (55 + 35) / 100 = 0.9 = 90\%$$

Từ kết quả trên, ta có thể hiểu rằng mô hình phân loại được 90% các ảnh trong tập dữ liệu kiểm tra đúng. Accuracy là một độ đo đơn giản và phổ biến được sử dụng để đánh giá hiệu suất của các thuật toán phân loại. Tuy nhiên, nó cũng có một số hạn chế, đặc biệt khi các lớp trong tập dữ liệu không cân bằng (một lớp có số lượng mẫu lớn hơn lớp khác), trong trường hợp này, ta cần sử dụng các độ

đo khác như precision, recall và F1-score để đánh giá hiệu suất của mô hình.

Bước 3: Huấn luyện mô hình:

- Đưa dữ liệu huấn luyện vào mô hình.

Để đưa dữ liệu huấn luyện vào mô hình, ta sử dụng method **fit()** của model. Ví dụ:

```
history = model.fit(train_iterator, epochs=10, validation_data=val_iterator)
```

Trong đó:

- **train\_iterator**: tập dữ liệu huấn luyện
- **val\_iterator = validation\_data**: để đánh độ chính xác của mô hình trên tập dữ liệu trong quá trình huấn luyện. Cụ thể, mỗi lần hoàn thành một epoch của quá trình huấn luyện, mô hình sẽ được đánh giá trên tập dữ liệu kiểm tra (validation data) và các chỉ số đánh giá hiệu suất như accuracy, loss,.. sẽ được trả về để phân tích hiệu suất của mô hình.
- **Epochs**: là số lần lặp lại toàn bộ dữ liệu huấn luyện để cập nhật trọng số (tức là chạy qua toàn bộ tập dữ liệu 10 lần)
- Chọn số epoch và batch size.

Ví dụ:

```
Epoch 1/10
40/40 [=====] - 151s 3s/step - loss: 0.7398 - accuracy: 0.5390 - val_loss: 0.6604 - val_accuracy: 0.6162
Epoch 2/10
40/40 [=====] - 133s 3s/step - loss: 0.6271 - accuracy: 0.6520 - val_loss: 0.5745 - val_accuracy: 0.7124
Epoch 3/10
40/40 [=====] - 141s 4s/step - loss: 0.5844 - accuracy: 0.6894 - val_loss: 0.5694 - val_accuracy: 0.7108
Epoch 4/10
40/40 [=====] - 134s 3s/step - loss: 0.5631 - accuracy: 0.7110 - val_loss: 0.5147 - val_accuracy: 0.7404
Epoch 5/10
40/40 [=====] - 134s 3s/step - loss: 0.5485 - accuracy: 0.7219 - val_loss: 0.5120 - val_accuracy: 0.7466
Epoch 6/10
40/40 [=====] - 133s 3s/step - loss: 0.5280 - accuracy: 0.7356 - val_loss: 0.4933 - val_accuracy: 0.7662
Epoch 7/10
40/40 [=====] - 134s 3s/step - loss: 0.5041 - accuracy: 0.7518 - val_loss: 0.4785 - val_accuracy: 0.7708
Epoch 8/10
40/40 [=====] - 133s 3s/step - loss: 0.4984 - accuracy: 0.7529 - val_loss: 0.4854 - val_accuracy: 0.7646
Epoch 9/10
40/40 [=====] - 134s 3s/step - loss: 0.4985 - accuracy: 0.7598 - val_loss: 0.4384 - val_accuracy: 0.7956
Epoch 10/10
40/40 [=====] - 133s 3s/step - loss: 0.4691 - accuracy: 0.7750 - val_loss: 0.4332 - val_accuracy: 0.7944
```

**Epoch = 10**

**Batch size = 40.**

- Gọi hàm fit để bắt đầu quá trình huấn luyện.

Bước 4: Đánh giá và kiểm tra mô hình

- Sử dụng tập kiểm tra để đánh giá mô hình.
- Tính toán độ chính xác và mất mát của mô hình trên tập kiểm tra.

```

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Accuracy Graph')
plt.legend()
plt.figure()

loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()

```

- Dòng đầu tiên lấy giá trị accuracy của training set trong từng epoch từ biến **history.history** và gán vào biến **acc**. Tương tự, lấy giá trị accuracy của validation set trong từng epoch và gán vào biến **val\_acc**.
  - Biến **epochs** lưu trữ các epoch.
  - Dòng tiếp theo dùng thư viện **matplotlib** để vẽ biểu đồ đánh giá độ chính xác của mô hình. Cụ thể, vẽ đường màu xanh biểu thị accuracy của training set và đường màu đỏ biểu thị accuracy của validation set. Tiêu đề của biểu đồ là "Accuracy Graph" và legend của biểu đồ sẽ hiển thị tên của 2 đường (Training Accuracy và Validation Accuracy).
  - Dòng thứ 6 và 7 tương tự như dòng 1 và 2, nhưng lần này lấy giá trị mất mát (loss) thay vì accuracy. Dùng thư viện **matplotlib** để vẽ biểu đồ đánh giá mất mát của mô hình. Cụ thể, vẽ đường màu xanh biểu thị loss của training set và đường màu đỏ biểu thị loss của validation set. Tiêu đề của biểu đồ là "Loss Graph" và legend của biểu đồ sẽ hiển thị tên của 2 đường (Training Loss và Validation Loss).
  - Cuối cùng, dùng **plt.show()** để hiển thị 2 biểu đồ.
  - Kiểm tra mô hình bằng cách dùng hình ảnh chó hoặc mèo bất kỳ.
- Bước 5: Sử dụng mô hình để phân loại hình ảnh mới
- Sử dụng hình ảnh mới để kiểm tra mô hình.

- Tiền xử lý hình ảnh theo cùng cách như khi huấn luyện mô hình.
- Sử dụng hàm predict để phân loại hình ảnh.

### 3. Một số thư viện được dùng trong bài toán

#### 3.1. *TensorFlow*

TF là một nền tảng Học máy mã nguồn mở, được thiết kế bởi đội ngũ Google Brain và tổ chức nghiên cứu trí tuệ máy của Google nhằm triển khai các ứng dụng của Học máy và Học sâu theo cách đơn giản. Nó là kết hợp giữa Đại số tính toán của các kỹ thuật tối ưu hoá để dễ dàng tính toán các biểu thức toán học. TF có một hệ sinh thái toàn diện, linh hoạt bao gồm các công cụ, thư viện và tài nguyên cộng đồng cho phép các nhà nghiên cứu xây dựng và triển khai các ứng dụng Học máy. Đây cũng là một trong những thư viện máy học lâu đời nhất.

Trang chủ của TF tại địa chỉ sau: <https://www.tensorflow.org/>.

Trang mã nguồn Github của TF nằm tại:

<https://github.com/tensorflow/tensorflow>

#### 3.2 *Scikit-learn*

Sklearn là một thư viện Học máy mã nguồn mở hữu ích và mạnh mẽ trong Python. Dự án của David Cournapeau bắt đầu vào năm 2007 với tư cách là một dự án của Google Summer of Code. Hiện tại, Sklearn đang được duy trì bởi một đội ngũ các tình nguyện viên. Sklearn cung cấp một sự lựa chọn các công cụ hiệu quả cho Học máy và mô hình thống kê, bao gồm phân loại, hồi quy, phân cụm và giảm chiều dữ liệu với giao diện nhất quán trong Python. Thư viện này phần lớn được viết bằng Python, được xây dựng dựa trên NumPy, SciPy và Matplotlib. Trang chủ của Sklearn tại địa chỉ sau: <https://scikit-learn.org/>.

Trang Github chứa mã nguồn Sklearn nằm tại:

<https://github.com/scikitlearn/scikit-learn>.

Một số sản phẩm thương mại sử dụng Sklearn như Spotify, Evernote, Booking.com, J.P.Morgan, Hugging Face, Télécom ParisTech, Aweber,...

#### 3.3 *Keras*

Keras là một thư viện Học sâu mã nguồn mở dành cho Python. Nó được phát triển bởi một nhà nghiên cứu trí tuệ nhân tạo của Google là Francois Chollet. Keras có thể chạy trên các thư viện mã nguồn mở như TensorFlow, Theano, R hay CognitiveToolkit (CNTK). Mục tiêu thiết kế của Keras là cho phép thử nghiệm các mạng Học sâu nhanh chóng. Các tổ chức hàng đầu như Google,

Square, Netflix, Huawei và Uber hiện đang sử dụng Keras. Trang chủ của Keras tại địa chỉ sau: <https://keras.io/>. Trang Github chứa mã nguồn của Keras nằm tại: <https://github.com/keras-team/keras>

### 3.4 Numpy

Numpy (Numeric Python): là một thư viện toán học phổ biến và mạnh mẽ của Python. Cho phép làm việc hiệu quả với ma trận và mảng, đặc biệt là dữ liệu ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều lần khi chỉ sử dụng “core Python” đơn thuần. Trang chủ của Sklearn tại địa chỉ sau:

<https://www.numpy.org>

### 3.5 Pandas

Pandas là một thư viện Python cung cấp các cấu trúc dữ liệu nhanh, mạnh mẽ, linh hoạt và mang hàm ý. Tên thư viện được bắt nguồn từ panel data (bảng dữ liệu). Pandas được thiết kế để làm việc dễ dàng và trực quan với dữ liệu có cấu trúc (dạng bảng, đa chiều, có tiềm năng không đồng nhất) và dữ liệu chuỗi thời gian. Trang chủ của Sklearn tại địa chỉ sau: <https://www.pandas.pydata.org>

## III. Code và chạy chương trình

### Thông tin tập dữ liệu

- Kho lưu trữ đào tạo chứa 25.000 hình ảnh về chó và mèo đã được gán nhãn (0 = Cat, 1 = Dog).
- Chạy chương trình trên **Google Colab**
- Bộ dữ liệu gồm 25000 ảnh đã được gán nhãn ‘dog’ và ‘cat’.
- Tỉ lệ test:train là 20 : 80

### Kiểm tra kết nối GPU – nhằm lấy ưu thế trong việc xử lý đa luồng làm tăng tốc xử lý

```
import tensorflow as tf
if tf.test.gpu_device_name():
    print('GPU found')
else:
    print("No GPU found")
```

kết quả:

 GPU found

### Import các module cần thiết

```
# Import các thư viện cần thiết
```

```

# import model và các thư viện liên quan training
from keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras import layers
from tensorflow.keras import models
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense,
Activation, BatchNormalization

# xử lý tệp
import zipfile
import os
from google.colab import drive

# xử lý toán học
import pandas as pd
import numpy as np
import cv2

# biểu đồ
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

```

## Kết nối vào google drive và giải nén thư mục zip vào workspace của Colab

```

drive.mount('/content/drive')

# Đường dẫn đến thư mục chứa các file zip
zip_train_path = '/content/drive/MyDrive/TrainingDogCatData/train.zip'
zip_test_path = '/content/drive/MyDrive/TrainingDogCatData/test.zip'


# Giải nén các file zip
with zipfile.ZipFile(zip_train_path, 'r') as zip_ref:
    zip_ref.extractall('/content/train')

with zipfile.ZipFile(zip_test_path, 'r') as zip_ref:
    zip_ref.extractall('/content/test')

# Đường dẫn đến các thư mục sau khi giải nén
train_dir = '/content/train/train'
test_dir = '/content/test/test1'

```

## kết quả

 Mounted at /content/drive

## Tạo khung dữ liệu chứa ảnh và nhãn của dataset

```
# Tạo danh sách chứa tên tệp và nhãn của từng tệp (0 cho mèo, 1 cho chó)
categories = []
filenames = []

dog = 0
cat = 0

train_dir_lst = os.listdir(train_dir)

for filename in train_dir_lst:
    if filename.startswith('cat'):
        categories.append(0)
        filenames.append(filename)
        cat += 1
    elif filename.startswith('dog'):
        categories.append(1)
        filenames.append(filename)
        dog += 1

# Tạo DataFrame từ danh sách tên tệp và nhãn tương ứng
df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})

print('cat image number = ', cat)
print('dog image number = ', dog)
print(df)
```

## kết quả

```
⇒ cat image number = 12500
   dog image number = 12500
   filename  category
0    cat.6397.jpg      0
1    dog.6764.jpg      1
2    cat.11213.jpg     0
3    cat.9603.jpg      0
4    dog.9389.jpg      1
...         ...      ...
24995  cat.4777.jpg     0
24996  dog.12181.jpg     1
24997  dog.1575.jpg     1
24998  dog.5952.jpg     1
24999  cat.4337.jpg     0

[25000 rows x 2 columns]
```



## Khởi tạo model gồm nhiều lớp cho CNN

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax')) # 2 because we have cat and
dog classes
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',
metrics=['accuracy'])
model.summary()
```



Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (Batch Normalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_1 (Batch Normalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_1 (Dropout)	(None, 30, 30, 64)	0



## Chuẩn bị dữ liệu và huấn luyện mô hình học máy bằng Keras

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLRonPlateau
earlystop = EarlyStopping(patience=10)
learning_rate_reduction = ReduceLRonPlateau(monitor="val_accuracy",
                                             patience=2,
                                             verbose=1,
                                             factor=0.5,
                                             min_lr=0.00001)
callbacks = [earlystop, learning_rate_reduction]
from sklearn.model_selection import train_test_split
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
# chia dữ liệu train val

train_df, validate_df = train_test_split(df, test_size=0.20,
random_state=42)
train_df= train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
batch_size=15
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

# Thay đổi kích thước ảnh
val_datagen = ImageDataGenerator(rescale=1./255)

# Luồng train hình ảnh theo lô 20 bằng train_datagen
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

# Flow from dataframe for training data
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    directory=train_dir,
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode="categorical",
    batch_size=batch_size
)
```

```
# Flow from dataframe for validation data
validation_generator = val_datagen.flow_from_dataframe(
    dataframe=validate_df,
    directory=train_dir,
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    batch_size=batch_size
)

epochs=3 if FAST_RUN else 20

# Bắt đầu training
history = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
Epoch 1/20
<ipython-input-7-1f7b1ae1c9f5>:70: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which
  history = model.fit_generator(
1333/1333 [=====] - 150s 106ms/step - loss: 0.7768 - accuracy: 0.6152 - val_loss: 0.7398 - val_accuracy: 0.6132 - lr: 0.0010
Epoch 2/20
1333/1333 [=====] - 141s 106ms/step - loss: 0.5891 - accuracy: 0.6957 - val_loss: 0.6059 - val_accuracy: 0.7137 - lr: 0.0010
Epoch 3/20
1333/1333 [=====] - 136s 102ms/step - loss: 0.5340 - accuracy: 0.7375 - val_loss: 0.4337 - val_accuracy: 0.7980 - lr: 0.0010
Epoch 4/20
1333/1333 [=====] - 139s 104ms/step - loss: 0.4795 - accuracy: 0.7756 - val_loss: 0.5647 - val_accuracy: 0.7405 - lr: 0.0010
Epoch 5/20
1333/1333 [=====] - 139s 104ms/step - loss: 0.4471 - accuracy: 0.7985 - val_loss: 0.3980 - val_accuracy: 0.8206 - lr: 0.0010
Epoch 6/20
1333/1333 [=====] - 139s 104ms/step - loss: 0.4176 - accuracy: 0.8120 - val_loss: 0.3617 - val_accuracy: 0.8434 - lr: 0.0010
Epoch 7/20
1333/1333 [=====] - 141s 106ms/step - loss: 0.3943 - accuracy: 0.8252 - val_loss: 0.3225 - val_accuracy: 0.8675 - lr: 0.0010
Epoch 8/20
1333/1333 [=====] - 140s 105ms/step - loss: 0.3778 - accuracy: 0.8360 - val_loss: 0.3821 - val_accuracy: 0.8242 - lr: 0.0010
Epoch 9/20
1333/1333 [=====] - ETA: 0s - loss: 0.3674 - accuracy: 0.8404
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.00050000000237487257.
1333/1333 [=====] - 139s 104ms/step - loss: 0.3674 - accuracy: 0.8404 - val_loss: 0.4153 - val_accuracy: 0.8232 - lr: 0.0010
Epoch 10/20
1333/1333 [=====] - 139s 104ms/step - loss: 0.3243 - accuracy: 0.8600 - val_loss: 0.2602 - val_accuracy: 0.8911 - lr: 5.0000e-04
Epoch 11/20
1333/1333 [=====] - 143s 107ms/step - loss: 0.3093 - accuracy: 0.8658 - val_loss: 0.2320 - val_accuracy: 0.9017 - lr: 5.0000e-04
```

## Lưu model dưới dạng file h5 và tải về máy

```
# lưu model
model.save("model.h5")
```

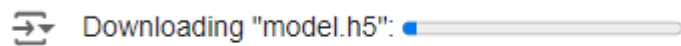
kết quả:

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:
  saving_api.save_model(
```

```
# lưu về máy tính
import numpy as np
from google.colab import files

# Tải tệp .h5 về máy tính
files.download("model.h5")
```

kết quả:

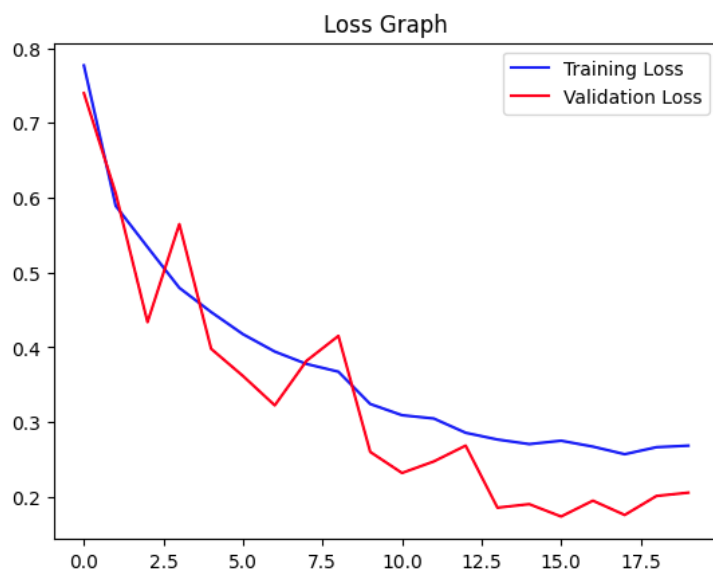
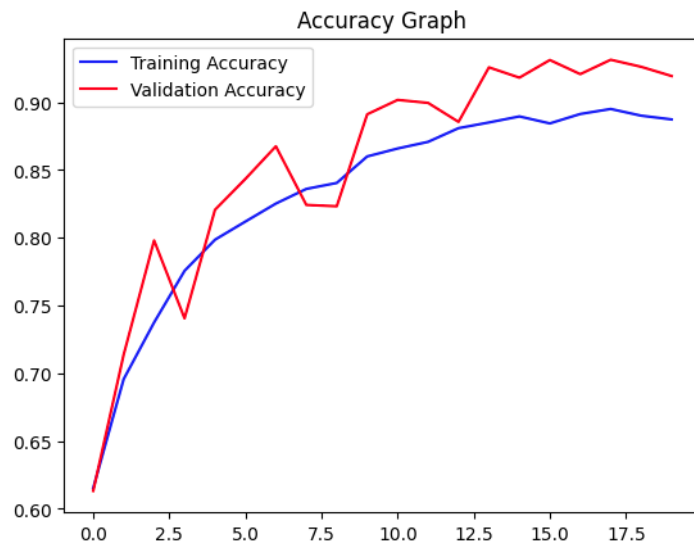


Tệp .h5 là một định dạng tệp được sử dụng để lưu trữ dữ liệu. .h5 là phần mở rộng của tệp sử dụng định dạng Hierarchical Data Format version 5 (HDF5). HDF5 là một định dạng tệp mạnh mẽ và linh hoạt được thiết kế để lưu trữ và quản lý dữ liệu phức tạp. HDF5 hỗ trợ lưu trữ dữ liệu lớn và có cấu trúc, cung cấp các tính năng như nén dữ liệu, quản lý siêu dữ liệu, và truy cập dữ liệu hiệu quả. Việc lưu model dưới file .h5 là một lựa chọn thường xuyên đối với các công việc liên quan đến học máy và khoa học dữ liệu.

## Trực quan hóa kết quả

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Accuracy Graph')
plt.legend()
plt.figure()
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(epochs, loss, 'b',
label='Training Loss')
plt.plot(epochs, val_loss, 'r',
label='Validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()
```

Kết quả:



Nhận xét:

- Độ chính xác đào tạo cao nhất là 0.88
- Độ chính xác xác thực cao nhất là 0.92
- Mất đào tạo thấp nhất (Training Loss) là 0.23
- Mất xác nhận thấp nhất (Validation Loss) là 0.18

### Kiểm tra bằng hình ảnh thật

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Lấy danh sách tất cả các file ảnh trong thư mục chưa gán nhãn
all_files = os.listdir(test_dir)

# Chọn ngẫu nhiên 10 ảnh
```

```

random_files = random.sample(all_files, 10)

# Hàm để hiển thị ảnh và dự đoán
def display_images_with_predictions(files, model, target_size):
    plt.figure(figsize=(15, 10))

    for i, file in enumerate(files):
        img_path = os.path.join(test_dir, file)

        # Load và chuẩn bị ảnh
        img = load_img(img_path, target_size=target_size)
        img_array = img_to_array(img) / 255.0 # Chuẩn hóa ảnh
        img_array = np.expand_dims(img_array, axis=0)

        # Dự đoán
        prediction = model.predict(img_array)
        if prediction[0][0] > prediction[0][1]:
            label = 'Cat'
        else:
            label = 'Dog'

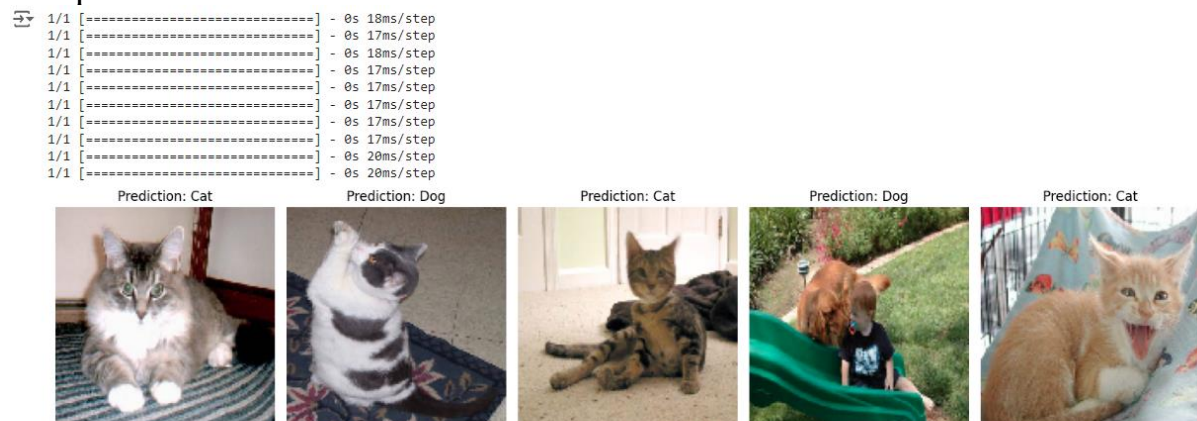
        # Hiển thị ảnh và dự đoán
        plt.subplot(2, 5, i + 1)
        plt.imshow(img)
        plt.title(f"Prediction: {label}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Gọi hàm để hiển thị ảnh và dự đoán
IMAGE_SIZE = (128, 128) # Kích thước ảnh đã dùng để train model
display_images_with_predictions(random_files, model, IMAGE_SIZE)

```

## kết quả





- Hàm predict của mô hình nhận vào `img_array`, là mảng số liệu đại diện cho hình ảnh đã được chuẩn hóa và định dạng phù hợp. Kết quả trả về là một mảng dự đoán (prediction), trong đó mỗi phần tử của mảng là xác suất thuộc về mỗi lớp (ở đây là hai lớp: "Cat" và "Dog").
- `prediction[0]`: Đại diện cho mảng xác suất của hình ảnh đầu vào, vì predict trả về một mảng hai chiều (batch size, số lớp), và ở đây batch size là 1.
- `prediction[0][0]`: Xác suất dự đoán cho lớp đầu tiên (giả sử là "Cat").
- `prediction[0][1]`: Xác suất dự đoán cho lớp thứ hai (giả sử là "Dog").
- Điều kiện `if prediction[0][0] > prediction[0][1]` kiểm tra xem xác suất dự đoán cho lớp "Cat" có lớn hơn xác suất dự đoán cho lớp "Dog" hay không.

## IV. Tổng kết

Nhận dạng hình ảnh chó mèo là một lĩnh vực rộng và đang được nghiên cứu và phát triển liên tục. Dưới đây là một số hướng phát triển chính của nhận dạng hình ảnh chó mèo mà chúng em đã tìm hiểu và áp dụng trong bài:

- *Sử dụng deep learning*: Deep learning đã trở thành một công cụ quan trọng trong nhận dạng hình ảnh chó mèo. Các mô hình deep learning như Convolutional Neural Networks đã được sử dụng để nhận dạng chó mèo với độ chính xác cao.
- *Sử dụng các kỹ thuật xử lý ảnh*: Những kỹ thuật xử lý ảnh như tiền xử lý ảnh, phân đoạn ảnh, đặc trưng hóa ảnh và đánh giá ảnh có thể được sử dụng để cải thiện độ chính xác của hệ thống nhận dạng.
- *Sử dụng các phương pháp kết hợp*: Các phương pháp kết hợp giữa deep learning và các kỹ thuật xử lý ảnh có thể được sử dụng để tăng cường độ chính xác của hệ thống nhận dạng hình ảnh chó mèo.
- *Sử dụng dữ liệu đa dạng*: Nhận dạng hình ảnh chó mèo có thể được cải thiện bằng cách sử dụng dữ liệu mới và đa dạng. Các dữ liệu được chúng em sử dụng trong bài thu thập từ các nguồn khác nhau như các bài đăng trên mạng xã hội hoặc các hình ảnh chó mèo chụp trong các môi trường khác nhau, dẫn đến việc độ chính xác được cải thiện rất nhiều.

Các kết quả có được trong chương trình cho thấy mô hình có hiệu suất tốt trên cả tập xác thực và huấn luyện, với độ chính xác cao và giá trị mất mát thấp. Điều này chứng tỏ mô hình không chỉ học tốt trên tập dữ liệu huấn luyện mà còn có khả năng tổng quát hóa tốt trên tập dữ liệu mới.

Tuy nhiên, mặc dù độ chính xác của mô hình đã ở mức cao nhưng thực tế có rất nhiều mô hình máy học, hoặc một số phương pháp tối ưu có thể thực hiện với mức chính xác tốt hơn và với thời gian huấn luyện thấp hơn thế nữa. Do đó, chúng em tự đánh giá bản thân cần phải chủ động tiếp thu nhiều hơn nữa để có thể tối ưu hóa model hiện tại.

Qua bài tập này, chúng em đã hiểu rõ hơn về quy trình huấn luyện mô hình học máy và các yếu tố ảnh hưởng đến hiệu suất của mô hình. Đây là nền tảng quan trọng cho việc nghiên cứu và phát triển trong lĩnh vực trí tuệ nhân tạo của chúng em sau này.

Chúng em xin gửi lời cảm ơn đến cô Đào Thị Thúy Quỳnh, giảng viên bộ môn Nhập môn trí tuệ nhân tạo, vì đã luôn mang đến những buổi học đầy năng lượng và thiết thực, tạo tiền đề để chúng em phát triển và nghiên cứu nội dung trong bài tập này. Mong rằng ở tương lai gần chúng em sẽ lại tiếp tục đồng hành cùng cô ở các môn học sắp tới.

Chúng em xin chân thành cảm ơn!

## Tài liệu tham khảo:

- [1]: MathWorks, What is a Convolution Network?,  
<https://www.mathworks.com/discovery/convolutional-neural-network.html>
- [2]: Adit Deshpande,  
<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-ToUnderstanding-Convolutional-Neural-Networks/>
- [3]: Ujjwal Karn,  
<https://ujjwalkarn.me/2016/08/11/intuitiveexplanationconvnets/>
- [4]: datawow.io, <https://datawow.io/blogs/interns-explain-cnn-8a669d053f8b>
- [5] EasyTensorFlow, *Convolutional Neural Networks (CNNs)*, <https://www.easy-tensorflow.com/tf-tutorials/convolutionalneuralnets-cnns>