Phân Biệt Chó Và Mèo

CS114.K23.KHCL

**Nguyễn Tiến Bảo**  
Trường đại học Công Nghệ Thông Tin – ĐHQG Tp. Hồ Chí Minh  
18520492@gm.uit.edu.vn**Vũ Duy Di Đan**  
Trường đại học Công Nghệ Thông Tin – ĐHQG Tp. Hồ Chí Minh  
18520554@gm.uit.edu.vn



*Sơ lược*

*Bản báo cáo mô tả phương pháp máy học trong bài toán phân lớp nhị phân, phân biệt hai loại động vật thân thiết nhất với con người đó là chó và mèo. Đồ án sử dụng ba phương pháp máy học để thực hiện bài toán bao gồm: Logistic Regression, Neural Network with One Hidden Layer, Convolutional Neural Network.*

# GIỚI THIỆU ĐỒ ÁN

Để phân biệt chó mèo đối với con người là một điều khá dễ dàng, tuy nhiên máy tính sẽ gặp đôi chút khó khăn với vấn đề này. Nhóm thấy rằng bài toán này tương đối dễ để có thể áp dụng được các kiến thức đã học để tạo ra một mô hình máy học thực sự. Chính vì vậy nhóm quyết định chọn đề tài này để làm nền tảng cho việc nghiên cứu về sau.

## Mục tiêu

Vận dụng được thành công những kiến thức về máy học cơ bản mà thầy cô đã dạy.

Mô hình hóa được một bài toán máy học thực sự.

Biết cách sử dụng và tinh chỉnh cũng như hiểu được nguyên lý hoạt động của những mô hình.

## Kết quả mong muốn

Mô hình có thể phân biệt thành công chó và mèo với độ chính xác cao nhất.

## Hướng phát triển tiếp theo

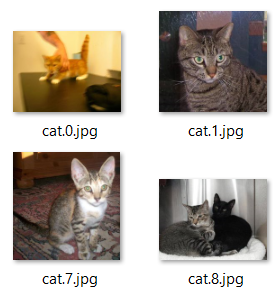
Ứng dụng cho phần mềm quản lý thú cưng, quản lý bán hàng cho các shop chó, mèo.

# DỮ LIỆU

## Giới thiệu bộ dữ liệu

Bài toán sử dụng tập dữ liệu có sẵn trên Kaggle.

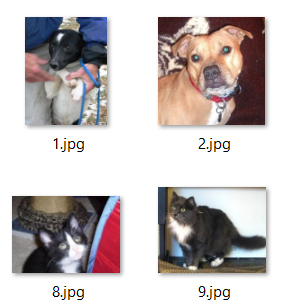
Bộ dữ liệu gồm có: 25000 hình ảnh của chó và mèo trong tập tin *“train”* dùng để huấn luyện mô hình đã được gắn nhãn:

* 12500 ảnh bé mèo với định dạng *“cat.0.jpg, cat.1.jpg, ...v.v”.*

*H.1. Hình ảnh mèo trong bộ dữ liệu train*

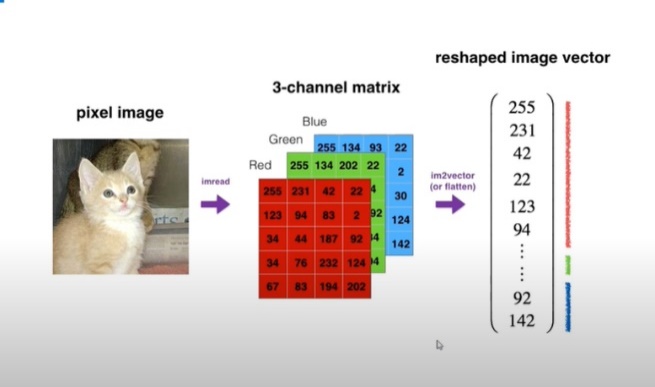
* 12500 ảnh chú chó với định dạng *“dog.0.jpg, dog.1.jpg, ...v.v”*.

*H.2. Hình ảnh chó trong bộ dữ liệu train*

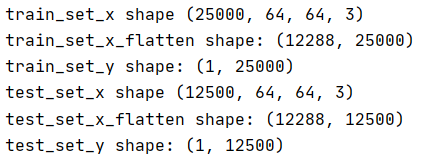
Bộ dữ liệu bao gồm: 12500 hình ảnh của chó và mèo trong tập tin *“test”* dùng để thử nghiệm mô hình đã được đánh số thứ tự từ 0 đến 12499 với định dạng *“0.jpg, 1.jpg, ...v.v”.*

*H.3. Hình ảnh chó mèo trong bộ dữ liệu test*

## Khai thác và xử lý bộ dữ liệu

* Những hình ảnh trong tập dữ liệu có kích thước ngẫu nhiên và hình theo hệ màu RGB.
* Để tối ưu tài nguyên của hệ thống, mô hình có thể huấn luyện nhanh hơn, chúng tascale mỗi tấm hình xuống kích thước 64x64x3 (Height=64, Width=64, “RGB”), khi scale xuống kích cỡ này mỗi tấm hình sẽ có số lượng pixel là 12288 pixels x 25000 hình với hệ màu 32bits RGB. Máy tính sẽ phải nạp vào ram và xử lý xấp xỉ 300 triệu pixels x 4bytes, xấp xỉ 1.3GB ram. Điều này sẽ làm cho mô hình huấn luyện trong thời gian ngắn hơn và tránh việc mô hình bị overfitting.
* Sau khi scale ta sử dụng thư viện OpenCV để đọc bức hình theo hệ màu RGB sau đó sử dụng thư viện numpy để reshape tấm hình về một vector có kích thước (12288, 1).

*H.4. Hình ảnh sau khi được xử lý*

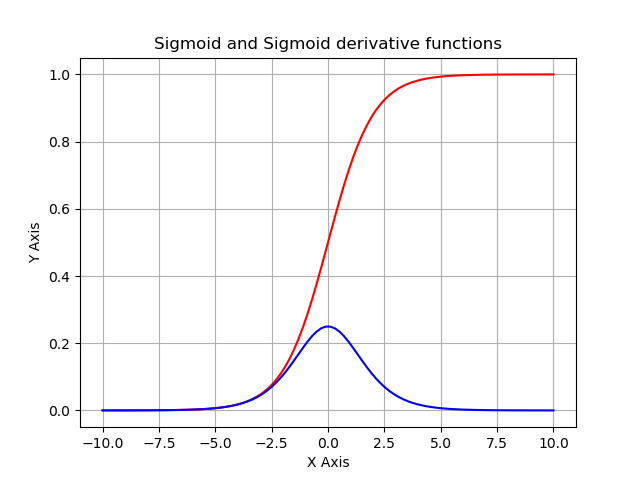
* Sau đó ta đưa dữ liệu lần lượt vào *“train\_set”*, *“test\_set”* và reshape phù hợp cho việc huấn luyện mô hình.
* *Train\_set\_x\_flatten* có số hàng là số pixel của một tấm ảnh, có số cột là số tấm ảnh trong tệp *“train”.*
* *Train\_set\_y\_flatten* có số hàng là 1 và số cột là số tấm ảnh có trong tệp “*train”,* ta quy định *“dog” = 1*, *“cat” = 0.*
* Sau đó ta chuẩn hóa các vector bằng cách chia từng dòng của *“train\_set”* với 255 (max value pixel channel), đây là cách chuẩn hóa vector vô cùng đơn giản khi làm việc với hình ảnh. Việc này giúp chúng tránh được việc làm cho hàm sigmoid bị bão hòa và xảy ra hiện tượng gradiant vanishing gây khó khăn trong việc tinh chỉnh tham số.

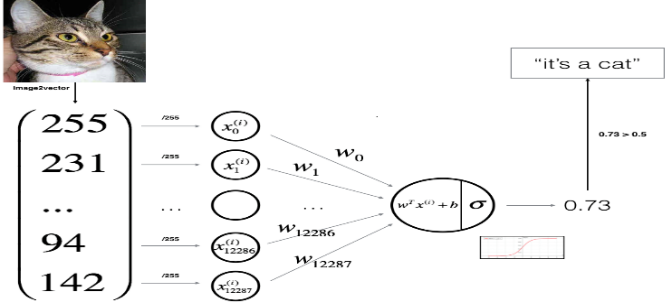
# CÁC PHƯƠNG PHÁP MÁY HỌC

## Logistic Regreesion

### Giới thiệu thuật toán

Kĩ thuật phân lớp là một phần rất quan trọng của máy học. Vì sự đặc trưng của *Logistic Regression* nên thuật toán này rất hiệu quả trong các bài phân lớp *“two-variables”.* *Logistic Regression* rất dễ để áp dụng và cơ sở của bất kì bài toán phân lớp nhị phân nào.

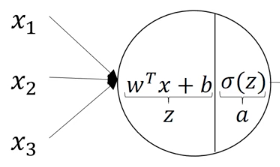
 Nói đến *Logistics Regression* là nói đến hàm *sigmoid,* nó là một *logistics function* có đường cong *“S”* đặc trưng. Nếu đường cong có giá trị dương thì y predicted = 1, nếu đường cong có giá trị âm thì y prediected = 0. Nếu giá trị đầu ra của hàm sigmoid lớn hơn 0.5 ta có thể phân lớp là 1 hoặc Yes, nếu giá trị đầu ra của hàm sigmoid bé hơn 0.5 ta có thế phân lớp là 0 hoặc No. Ví dụ {1: “dog”, 0: “cat”} nếu giá trị đầu ra bằng 0.75, ta có thể nói trong tấm ảnh đã cho có 75% chứa hình ảnh một chú chó.

Nhìn vào hình dưới ta có thế thấy *Logistics Regression* thật ra là một mô hình *Neural Network* cơ bản với một *neuron.*

*H.5. Mô hình Logistic Regreesion*

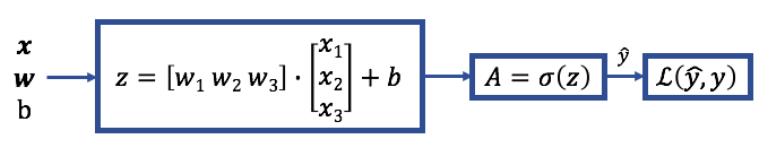
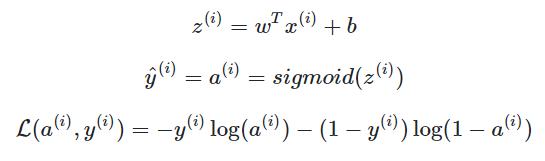
### Xử lý các tham số và xử lý lỗi

#### Forward Propagation

 Từ input, ta sẽ tính toán giá trị các neural của từng lớp một, đến cuối cùng sẽ tính ra giá trị của lớp output.

Trong đó z gọi là hàm pre-activation, còn a gọi là hàm activation, w và b là các parameter của neural network. Mục đích cuối cùng của chúng ta là tối ưu hóa các parameter này.

Trong quá trình này *weight* và *bias* đã được đẩy vào từ input, quá trình này được gọi là *forward propagation.* Sau khi đã tính được output từ dữ liệu trên ta tiến hành tính toán hàm mất mát cho từng tấm hình.

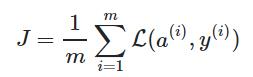


*H.6. Công thức để tính hàm mất mát*

*Tham số:*  
w - weights, một vector 0 có số hang là số tấm hình của tập dữ liệu.  
b - bias, vector vô hướng 0.  
x – là một ma trận có số dòng là số pixel của một tấm ảnh và số cột là số hình ảnh trong tập *“train”* hoặc tập *“test”* (12288, 25000), (12288, 12500).

y – là tập, vector huấn luyện đã được quy định label *“dog = 1, cat = 0”* (1, 25000).

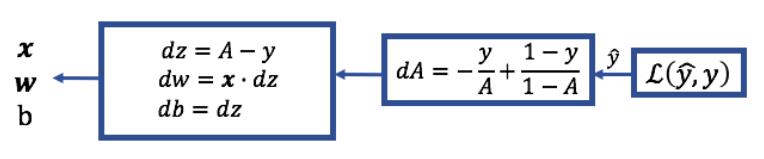
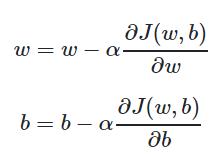
Sau khi đã tính được giá trị hàm mất mát của từng vòng lặp, ta sẽ tính trung bình cộng tất cả giá trị của hàm mất mát để được giá trị hàm lỗi.

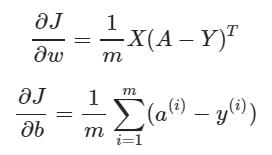


#### Backward Propagation

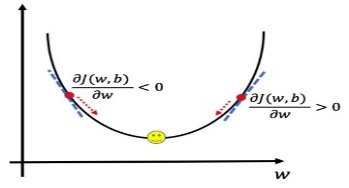
Sau khi đã tính toán được hàm lỗi ta dựa vào hàm lỗi trước đó để cập nhật các giá trị weight và bias sao cho hàm lỗi có giá trị là nhỏ nhất. Ta sử dụng đạo hàm để tìm được giá trị hiện tại là âm hay dương và hướng đi tiếp theo của hàm số.

Các công thức toán học để tính toán *backward propagation* (tính đạo hàm).

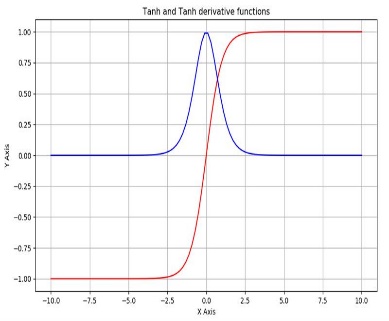




Sau khi đã có các giá trị đạo hàm, ta viết hàm tối ưu dựa trên thuật toán Gradient Descent, ta cập nhật giá trị của weight và bias sau mỗi lần lặp, tốc độ và giá trị của sự thay đổi này phụ thuộc vào giá trị của hằng số *(learning rate)* và số lần lặp để lấy mẫu.

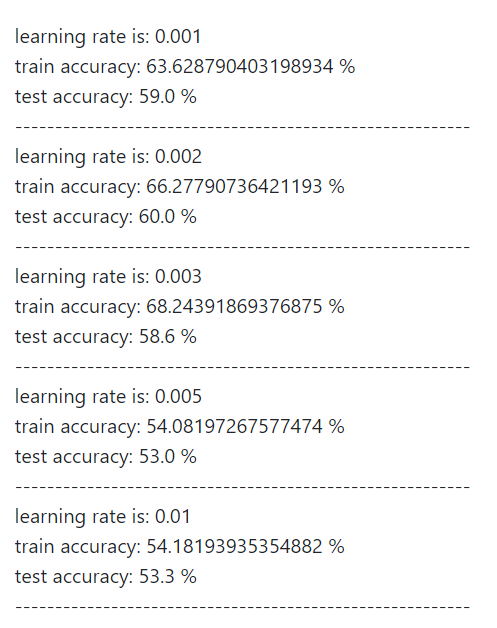
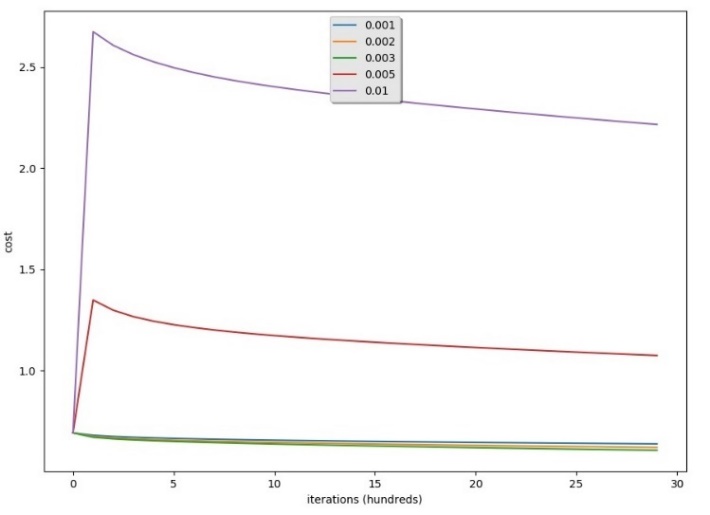


Nhìn vào đồ thị trên sẽ giúp ta miêu tả hướng đi của weight. Khi đạo hàm dương ta sẽ di chuyển về hướng ngược lại, giảm giá trị của weight. Khi đạo hàm âm ta di chuyển về hướng ngược lại, tăng giá trị của weight. Việc này luôn đảm bảo rằng ta sẽ luôn tiến tới vị trí *“mặt cười”* trong đồ thị.

 Việc lựa chọn *learning rate* rất quan trọng. Nếu ta lựa chọn *learning rate* quá thấp, mô hình sẽ tốn rất nhiều thời gian để hàm lỗi tiến tới giá trị nhỏ nhất. Nếu chọn *learning rate* quá cao, hàm số sẽ nhảy qua nhảy lại xung quanh điểm *“mặt cười”.*

Gradient Descent là bản chất của việc học, nó thể hiện cách thức mà máy tính có thể tìm được hai giá trị weight và bias sao cho hàm lỗi có giá trị nhỏ nhất bằng cách so sánh lặp đi lặp lại giá trị dự đoán đầu ra với chuỗi dữ liệu *“true output”* của tập *“train”,* điển hình ra tập *“train\_set\_y\_flatten”* được quy định *“dog =1”, “cat = 0”.*

#### Thực nghiệm mô hình

 Ta thí nghiệm mô hình với 4 giá trị *learning rate* khác nhau [0.001, 0.002, 0.003, 0.005, 0.01], số lần lặp để lấy mẫu là 500 lần.

*H.7. Biểu đồ thể hiện kết quả*

*Nhận xét:*

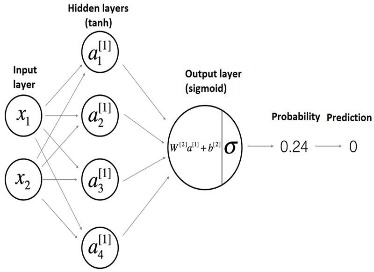
Ta thấy *learning rate* khác nhau sẽ cho ra kết quả khác nhau. Accuracy của mô hình chưa cao, tất cả mô hình đều có giá trị của hàm *cost* giảm tuyến tính, đặc biệt với *learning rate* 0.005, 0.01 hàm *cost* dao động lên xuống rồi lại giảm xuống tuyến tính.

Mô hình có giá trị của hàm *cost* thấp không có ý nghĩa đó là một mô hình hiệu quả, chúng ta phải kiểm tra xem mô hình có bị overfitting hay không.

## Neural Network With One Hidden Layer

### Giới thiệu thuật toán

Đây là một hệ thống mạng *neuron* đơn giản nhất với một *“hidden layer”,* thuật toán này cũng tương tự *logistic regression* nhưng thay vì sử dụng hàm activation là hàm thì mỗi *neuron* của *hidden layer* sử dụng hàm activation là hàm .



Vì mô hình của chúng ta có một hidden layer nên chúng ta cần khởi tạo các tham số cho input và hidden layer chúng ta không thể khởi tạo các tham số bằng 0 vì các tham số của hidden layer phụ thuộc vào các tham số của input nên chúng ta khởi tạo một tham số random có giá trị nhỏ gần bằng 0, bias = 0.

### Xử lý các tham số và xử lý lỗi

#### Forward Propagation

Chúng ta tính toán các tham số *z1, a1, z2, a2,* theo các công thức ở hình dưới.

*Tham số:*

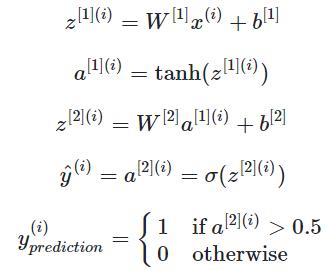
x – input kích cỡ của dữ liệu (input\_layer, số lượng hình sample).

a2 – đầu ra ouput của hàm activation thứ hai (sigmoid).

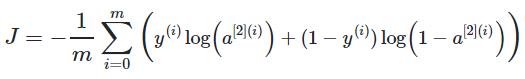
y - là tập, vector huấn luyện đã được quy định label *“dog = 1, cat = 0”,* (1, 25000).

w1, w2 = rand (0.01, 0.1)

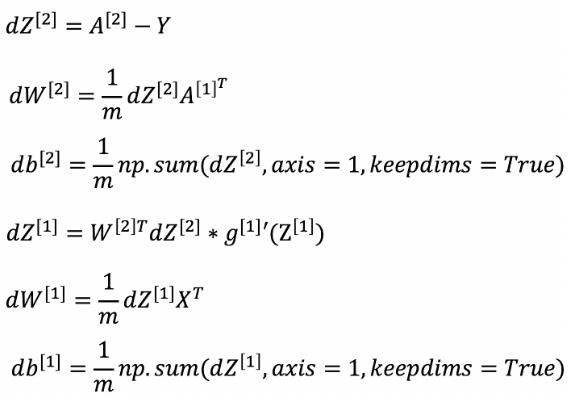
b1, b2 = 0



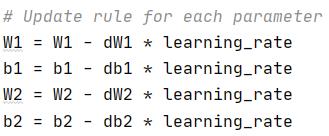
Sau khi tính toán đầy đủ các tham số trên ta có thể tính hàm lỗi bằng công thức sau đây:



#### Backward Propagation

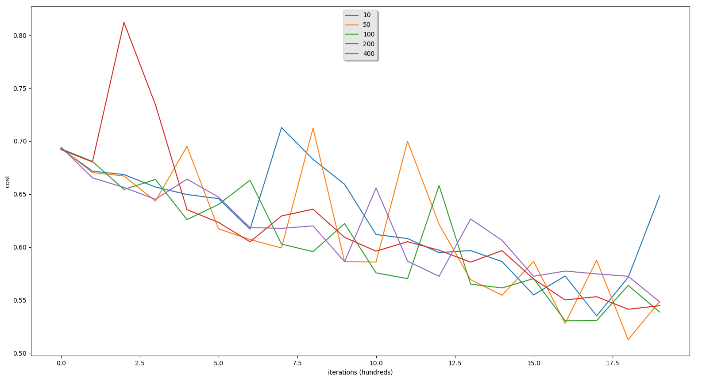
 Các công thức để tính đạo hàm các tham số z1, z2, w1, w2, b1, b2:

Chúng ta cập nhật các tham số w1, b1, w2, b2 sao cho hàm lỗi giảm tuyến tính giống như phương pháp *logistic regression* sử dụng thuật toán *gradient descent*

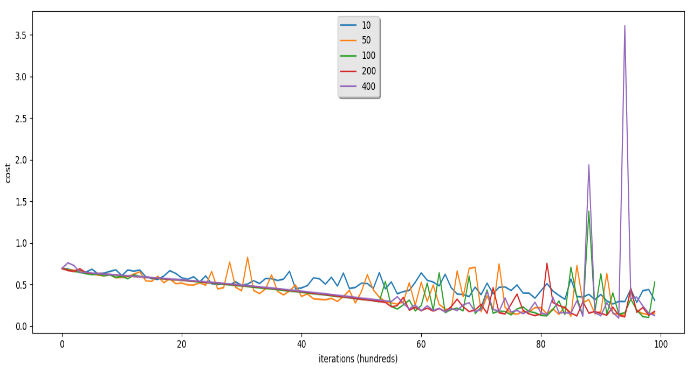


Việc tinh chỉnh *learning rate* và số lần lặp để lấy mẫu cũng như phương pháp logistic regression.

#### Thực nghiệm mô hình



*H.8.* Biểu đồ thể hiện kết quả cột bên trái



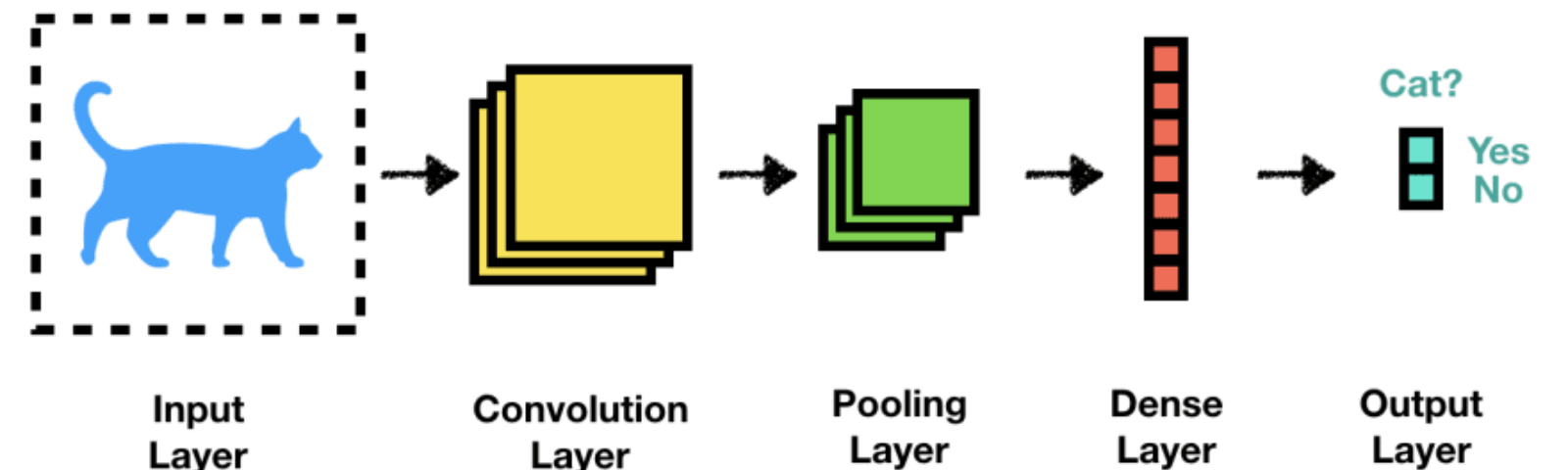
*H.9.* Biểu đồ thể hiện kết quả cột bên phải

## Convolutional Neural Network (CNN)

### Giới thiệu thuật toán

Sau khi đã trải qua hai phương pháp trên, nhóm nhận thấy độ chính xác là không cao. Nhóm đã làm qua phương pháp Neural Network with One Hidden Layer, nếu ta thêm một số *hidden layer* nữa, mỗi layer có vài chục *neuron* thì ta sẽ tạo ra *deep neural* *network*. Việc này gây tốn thời gian và có rất nhiều tham số cần được xử lý nên nhóm đã quyết định chọn phương pháp Convolutional Neural Network hiện thực bằng thư viện *keras* dưới nền *tensorflow.*

Thay vì các layer đều kết nối với nhau theo thứ tự (input layer, hidden layer, output layer) thì CNN kết nối các *layer* con của từng phần rồi sau đó kết nối từng phần với nhau và có một lớp đặc biệt *convolutional layer.*

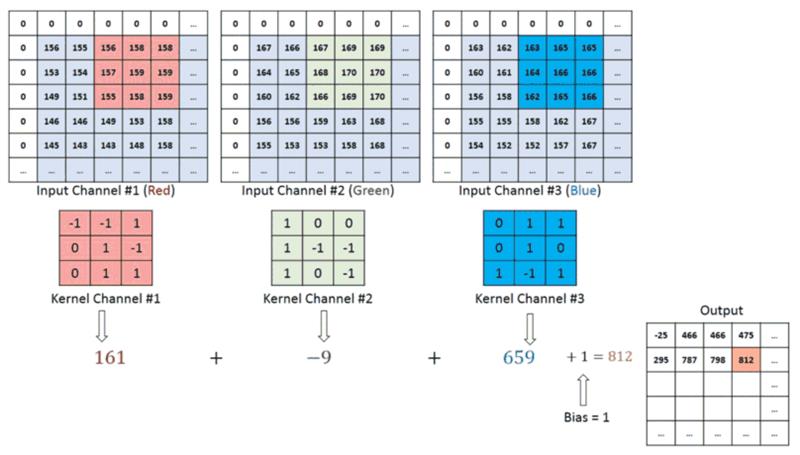


CNN bao gồm hai phần chính:

* *Feature extraction:* trong giai đoạn này mạng neuron sẽ liên tục thực hiện các bước *“convolution”* và *“pooling”* để phát hiện những đặc điểm của tấm hình. Nếu trong bức hình có một con mèo thì nó sẽ trích xuất ra những đặc điểm của mèo như: tai, mũi, miệng đuôi, ...
* *Classification:* sau khi đã trích xuất những đặc điểm thì chúng ta sẽ cho qua lớp *fully connected layer* hoặc *dense layer* để dự đoán khả năng tấm hình con mèo hay chó để đưa ra tấm hình output layer.

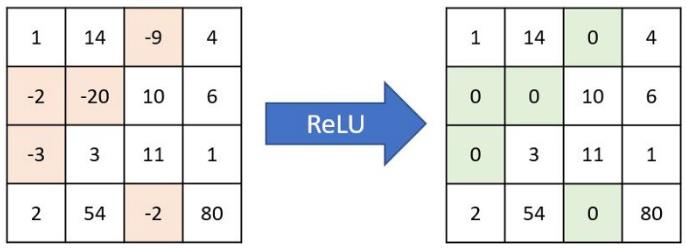
#### C:\Users\vuduy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\3148E655.tmpC:\Users\vuduy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4C9D9B6F.tmpConvolution layer

Trên tấm hình trên ta sử dụng 1 filter 3x3 để quét một tấm hình 5x5 ta lần lượt lấy filter bao phủ góc trái của tấm hình và dịch qua phải theo tham số stride, ở đây ta cho stride = 1. Tiếp tục dịch dần xuống đến khi quét hết tấm hình, ta sẽ được một tấm hình 3x3 mới với giá trị của từng pixel là tích chập của các pixel trong các tấm hình cũ khi ta phủ filter lên.



*H.10. Convolution layer thực hiện với hệ màu RGB*

Sau đó ta cho tấm hình đi qua hàm *activation reLU* để biến tất cả các pixel có giá trị âm về 0 và các pixel có giá trị dương giữ nguyên để loại bỏ các pixel dư thừa và làm nổi bật lên *feature* của tấm hình.

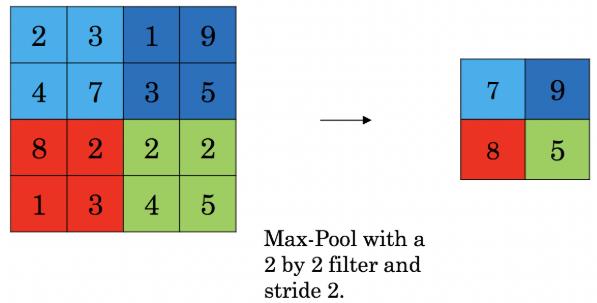


Một mô hình có thể có nhiều *convolution layer*, layer đầu tiên trích xuất *“low level features”,* những layer sau sẽ trích xuất *“medium level features”* và *“high level features”*.

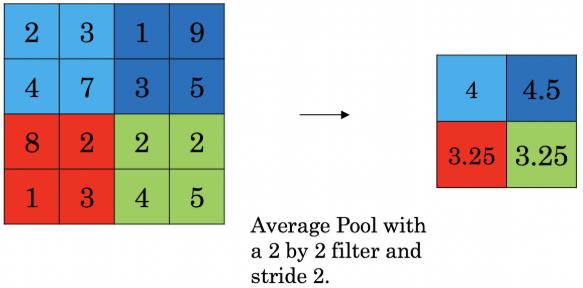
#### Pooling layer

Sau khi tấm hình đi qua lớp *“convolutional layer”,* chúng ta sẽ có một *“feature map”*, thêm vào model một *“pooling layer”* để giảm kích thước của tấm hình cũng như giảm đi những tham số cần tính toán của mô hình mà vẫn giữ nguyên được đặc điểm của tấm hình, điều này giúp rút ngắn thời gian huấn luyện và tránh hiện tượng *“overfitting”*. Có hai loại *pooling:*

* *Max pooling*: Trả về giá trị cao nhất của phần hình ảnh bị filter bao phủ.



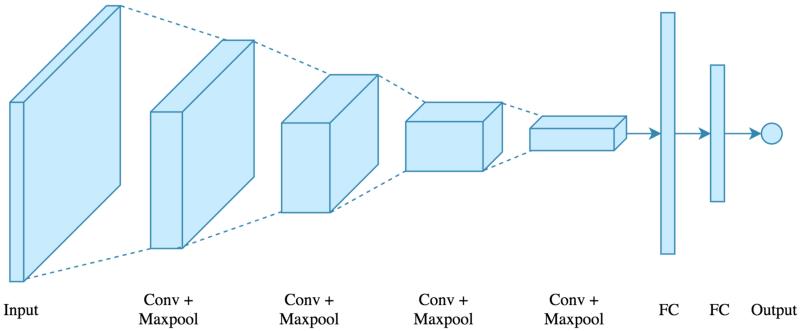
* *Average pooling:* Trả về giá trị trung bình cộng của phần hình ảnh bị filter bao phủ.



Trên thực tế *Max pooling* cho ra kết quả tốt hơn *Average pooling.*

#### Fully connected layer

Fully connected, 1-layer để tập hợp các feature layer mà ta đã tìm ra, chuyển đổi dữ liệu từ 3D, hoặc 2D thành 1D, tức chỉ còn là 1-vector. Còn 1-layer nữa là output, số neuron của layer này phụ thuộc vào số output mà ta muốn tìm ra. Thường thì sau các lớp Conv+Pooling thì sẽ là 2 lớp Fully connected.



Ngoài ra mô hình còn sử dụng thêm một lớp nữa là *dropout layer,* lớp này có nhiệm vụ loại bỏ ngẫu nhiên các tham số trong mô hình tránh mô hình bị overfitting. Ba lớp Conv2D+Pooling2D+Dropout còn được gọi là *convolution block.* Ta có thể áp dụng nhiều *convolution block* cho mô hình để tăng độ sâu.

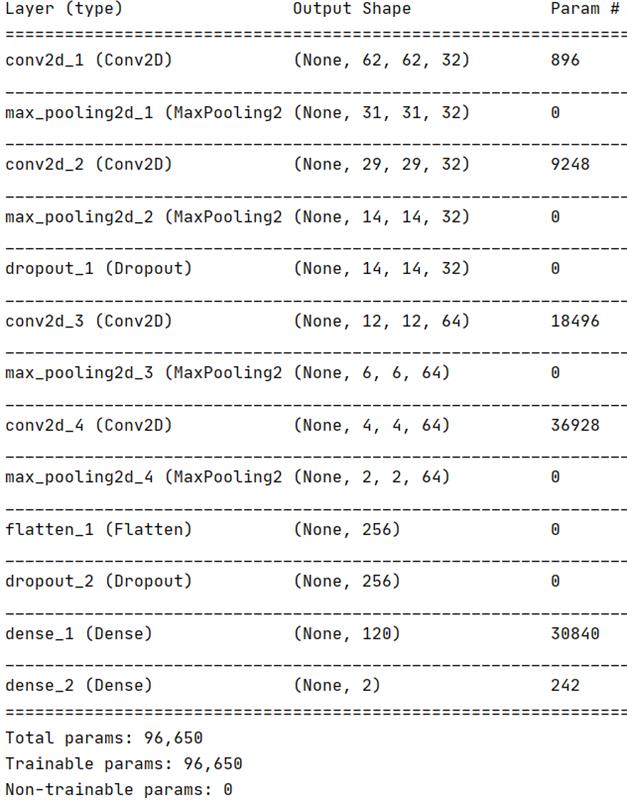
### Thực nghiệm mô hình và tinh chỉnh tham số

Ta chia tập *“train”* thành hai phần: 80% để huấn luyện, 20% để thực nghiệm (20000 train and 5000 validation samples).

Ta training model với epochs = 20 và batch\_size = 64.

* A screenshot of a social media post

  Description automatically generatedModel 1: Ta xây dựng model 1 đơn giản theo mô hình đã nói ở phần trên.



A close up of a map

Description automatically generated



A close up of a map

Description automatically generated

*Nhận xét:*

Ta thấy mô hình của nhóm đã đạt được *accuracy* với tập train là 83.5%, với tập *validation* là 85.6%. Mô hình này không gặp phải hiện tượng overfitting hay underfitting. Tuy nhiên, ta vẫn có thể cải thiện *accuracy* cho mô hình. Bảng *summary* của mô hình cho ta biết *trainable params* chỉ là 96.650. Chúng ta có thể cải thiện điều này bằng hai cách:

Tăng độ sâu của mô hình bằng cách thêm vào mô hình một số *convolution block*, mỗi block ta sẽ tăng số lượng filter lên gấp đôi.

Giảm số lượng dropout sau mỗi *convolution block* xuống 40% để tăng số lượng *trainable params* nhưng không quá lớn tránh hiện tượng overfitting.

* A screenshot of a social media post

  Description automatically generatedModel 2: Ta cải thiện model 1 thông qua model 2.

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A close up of a device

Description automatically generated*Nhận xét:*

Ta thấy sau khi thêm 1 lớp *convolution layer*, tăng số lượng filter của mỗi lớp lên gấp đôi, đồng thời *dropout* hợp lý là 40% parameters sau mỗi lớp *convolution layer* ta đã tránh được hiện tượng *overfitting*. Accuracy của *train* và *validation* gần như bằng nhau ở hình trên.

# KẾT LUẬN VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN

Sau khi thực hiện bài toán qua ba thuật toán ta thấy sự phát triển của lĩnh vực Machine Learning thật thần kì đặc biệt là trong lĩnh vực *computer vision*. Bài toán này chỉ là bước khởi đầu lĩnh vực *computer vision* với muôn ngàn ứng dụng thực tế. Mô hình CNN với tốc độ train nhanh nhờ vào việc vận dụng được nhân GPU trong lúc huấn luyện cũng như độ chính xác cực kì cao với đặc trưng trích xuất đặc điểm của ảnh.. Neural Network with One Hidden layer tuy cho ra kết quả tốt hơn LR tuy nhiên thời gian huấn luyện là quá lâu để đổi lại chỉ là 3%.

Dựa vào mô hình trên chúng ta có thể mở rộng lên nhận diện multiclass như nhiều loại động vật khác ngoài mèo và chó như: bò, gà…Nhóm có thể phát triển theo hướng nhận diện khuôn mặt và định danh cũng như phát hiện vật thể và phân loại.

bảng accuracy của ba thuật toán

| ***Thuật Toán*** | Accuracy | |
| --- | --- | --- |
| Train | Test |
| Logistic Regreesion | 66.2% | 60% |
| Neural Network with One Hidden Layer | 70% | 61.1% |
| Convolutional Neural Network | 89.8% | 89.2% |

# KẾ HOẠCH

|  |  |  |  |
| --- | --- | --- | --- |
| ***Thành viên*** | Thu thập dữ liệu | Tìm hiểu thuật toán | Tổng hợp báo cáo |
| Nguyễn Tiến Bảo | ✓ | ✓ | ✓ |
| Vũ Duy Di Đan | ✓ | ✓ | ✓ |

Thời gian thực hiện:

* 15/4/2020 Cả nhóm tìm hiểu, đưa ra ý tưởng và tổng hợp dữ liệu cũng như phương pháp.
* 27/4/2020 Báo cáo tiến độ và giải quyết khó khăn mà mọi người gặp phải.
* 11/5/2020 Cả nhóm báo cáo tiến độ.
* 23/5/2020 Bảo hướng dẫn Đan hiểu được những chỗ khó của thuật toán
* 28/5/2020 - 15/6/2020 Chạy thử nghiệm các phương pháp – tổng hợp src code.
* 18/6/2020 - 30/6/2020 Viết báo cáo và chuẩn bị.
* 9/7/2020 Báo cáo hoàn thành đồ án.

*Nhận xét:* Cả hai đều nhất quán trong quan điểm cũng như về thời gian thực hiện đồ án, cùng với việc hỗ trợ lẫn nhau trong suốt quá trình thực hiện nên tiến độ công việc diễn ra đúng tiến độ và hiệu quả.

# TÀI LIỆU THAM KHẢO

1. Greg Surma, “Image Classifier - Cats🐱 vs Dogs🐶”, Leveraging Convolutional Neural Networks (CNNs) and Google Colab’s Free GPU, Nov 19, 2018
2. Jason Brownlee, “Overfitting and Underfitting With Machine Learning Algorithms,” Machine Learning Algorithms, Mar 21, 2016, Last Updated on Aug 12, 2019.
3. Ha Nguyen, “Định nghĩa và lý giải tổng quan về Backpropagation,” forum.machinelearningcoban.com, comment 6/18, Oct, 2018.
4. Cuong Tran (Vicohub), “Thuật Toán Tối Ưu Gradient Descent,” Aug 25, 2019.
5. Sanjaya Subedi, “Convolutional Neural Networks in Keras (Cat vs. Dog example),” Learn how to design and train a convolutional neural network for binary classification in Keras, Categories: DeepLearning, Sanjaya’s Blog, Jan 01, 2018.
6. nguyen, Nguyen Manh Nguyen, “Mối quan hệ đánh đổi giữa bias và variance,” forum.machinelearningcoban.com, comment 1/13, Feb, 2019.