**BÁO CÁO THÁNG 4 – ĐỒ ÁN 2:**

**MỞ CỬA TỰ ĐỘNG BẰNG NHẬN DIỆN KHUÔN MẶT**

**Giảng viên hướng dẫn:**

* PGS.TS.Vũ Đức Lung

**Sinh viên thực hiện:**

* Nguyễn Tấn Phát - 16521619
* Lã Tuấn Vinh – 16521439

**Nội dung báo cáo:**

* Neural network là gì?
* Cấu trúc mạng CNN
* Cách lập trình mạng CNN

1. **NEURAL NETWORK LÀ GÌ ?**

Neural Network là một trong những thuật toán machine learning phổ biến nhất hiện tại. Điều đó đã được chứng minh bởi Neural Networks vượt qua các giải thuật khác về hiệu quả cũng như tốc độ. Với một số biến thể như CNN, RNN, Autoencoders, DeepLearning, …

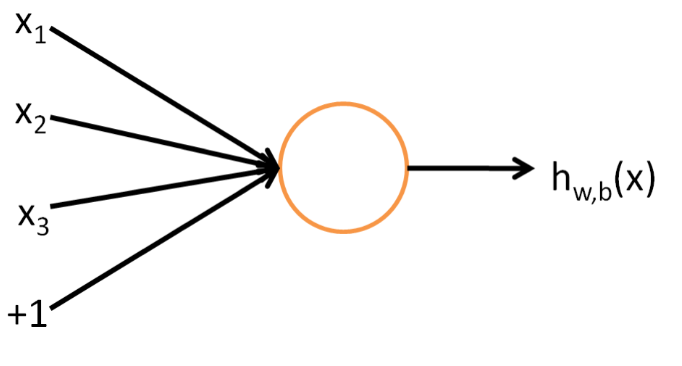
Neural Network dần được các nhà nghiên cứu dữ liệu hoặc các người tìm hiểu về machine learning … !!!!!!!!!!

Vì vậy chúng ta cần có một cái nhìn tổng quan về Neural Network là gì, chúng cấu thành từ những thứ gì, ưu điểm và khuyết điểm của chúng.

**Neuron là gì?**

Cái tên đã cho thấy, cụm từ ***Neural Network*** được lấy cảm hứng bởi hệ thống cấu trúc neural tại bộ não của con người, và cũng như bộ não người các khối cấu tạo nên chúng được gọi là ***Neuron***. Chức năng của chúng cũng tương tự như các neuron của con người: Chúng nhận vào nhiều input và đưa ra một output.

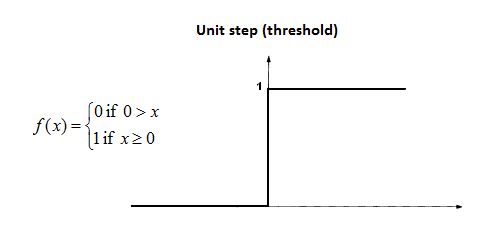
Đối với lĩnh vực toán học, một ***Neuron*** trong thế giới Machine Learning là một ***placeholder*** cho một phép toán, và công việc duy nhất của chúng là đưa ra một output bằng cách áp dụng phép toán trên các input cho sẵn.



Các function sử dụng trong một neuron thường được gọi là một ***Activation Funtion***. Hiện nay có rất nhiều các ***Activation Function*** như: step, sigmoid, tanh, ReLU, leaky ReLU, …

1. **Activation Function**
2. ***Step function***

Một step function được định nghĩa bởi:



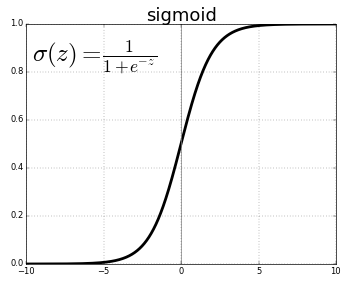
Output sẽ bằng 1 nếu giá trị của x lớn hơn hoặc bằng không và bằng 0 nếu x bé hơn không. Có thể thấy *step function* có giá trị không đổi tại 0.

Hiện tại, một ***Neural Network*** sử dụng phương thức *Back Propagation* cùng với *Gradient Descent* để tính các weights cho các layer khác nhau. Vì *step function* có điểm không đổi tại 0 nên không còn phù hợp để dự đoán với *gradient descent* và không thể cập nhật các weights.

Để khắc phục điều này, sigmoid function đã được giới thiệu để khắc phục điểu này của step function.

1. ***Sigmoid function***

Một sigmoid function hay logistic funtion được định nghĩa:



Giá trị của function hướng về 0 khi z hoặc biến độc lập hướng về âm vô cực, hướng về 1 khi z hướng về dương vô cực. Function này biểu diễn sự hội tụ sự hoạt động của các biến độc lập.

Vậy chúng ta dùng sigmoid function khi nào?

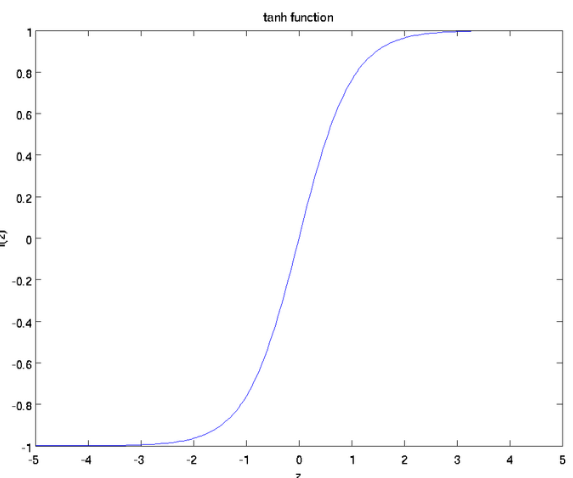
* Sử dụng cho tập dữ liệu ***không*** tuyến tính.
* Sigmoid function khác nhau tại mọi điểm và do đó có thể sử dụng với GD và backpropagation.

Việc đảm nhiệm các biến độc lập để dựa vào sigmoid function vốn có cho thấy(3. The assumption of a dependent variable to follow a sigmoid function inherently assumes a Gaussian distribution for the independent variable which is a general distribution we see for a lot of randomly occurring events and this is a good generic distribution to start with.)

Tuy nhiên, sigmoid function cũng gặp phải vấn đề về “Vanishing Gradient”. Có thể thấy từ hình trên rằng sigmoid funciton nén input của nó thành một phần rất nhỏ [0, 1] và có đạo hàm rất dốc. Vì vậy, những mảng input lớn còn sót lại, nơi nà đến những thay đổi lớn chỉ đưa ra những thay đổi rất nhỏ tại output.

1. ***Tanh Function***

The tanh(z) function là phiên bản cải tiến của sigmoid fucntion và output của nó nằm trong khoảng [-1, 1] thay vì [0, 1]

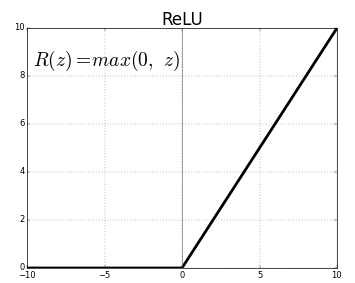


Lý do thông thường để dùng Tanh Function ở một số trường hợp thay cho Sigmoid funciton là vì khi dữ liệu nằm vòng quanh điểm 0, đạo hàm cao hơn. Và với độ dốc cao hơn sẽ giúp cải thiện learning rate.

Tuy nhiên Vanishing gradients vẫn tồn tại tại Tanh function.

1. ***ReLU Function***

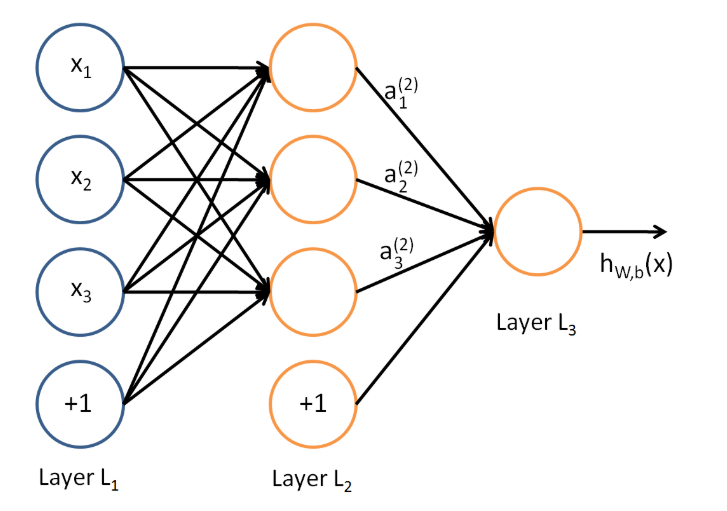
Rectified Linear Unit được sử dụng rất nhiều trong các deep learning model. Function này trả về 0 nếu nó nhận bất cứ giá trị input âm nào. Nhưng đối với các input dương nó sẽ trả về chính giá trị dương đó. Vì thế có thể viết f(x) = max(0, x).

****

1. **Neural Network**

Chúng ta đã được biết qua về neuron và activation fucntion . Bây giờ chũng ta sẽ đi sâu hơn về Neural network và các biến thể của nó.

Trước khi hiểu về neural network, chúng ta cần biết được layer bên trong chúng là gì. Một layer is nothing nhưng là một tập hợp các neuron thứ mà lấy input và đưa ra output. Inputs của mỗi neuron được xử lí thông qua activation function.định sẵn của mỗi neuron đó.



Layer bên trái ngoài cùng được gọi là **input layer**, và layer bên phải ngoài cùng là **output layer**. Các layer ở giữa được gọi là các **hidden layer** bởi vì giá trị của chúng không quan sát được bên trong training set.

Cũng có thể nói ví dụ trên bao gồm 3 **input units** (*không tính node bias*), **3 hidden uint** và 1 **output unit.**

Mỗi neural network đều phải có 1 input và 1 output layer và một số hidden layer.

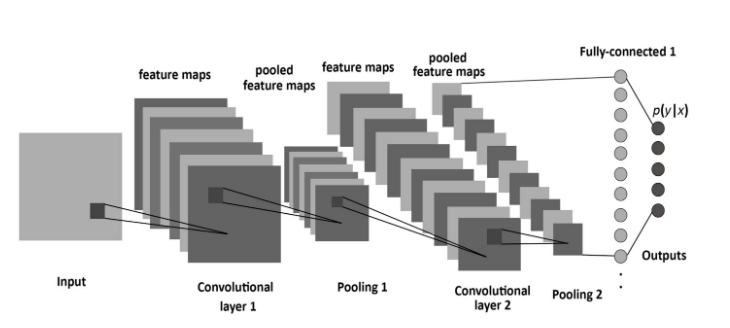
Một điều đáng chú ý ở đây là mỗi hidden layer có thể có activation fuction khác nhau. Ví dụ: hidden layer 1 có thể dùng sigmoid fuction, hidden layer 2 có thể dùng ReLU và Tanh fucntion tại lớp thứ 3. Chúng ta sẽ chọn activation function sao cho phù hợp với từng trường hợp dựa trên các loại dữ liệu được đưa ra.

Tiếp đến neural network cần phải đưa ra dự đoán chính xác. Với mỗi neuron ở mỗi layer đều có một giá trị weight. Đó chính là giá trị cần được cải thiện qua mỗi lần học từ dữ liệu của neural network. Và giải thuật học đó được gọi là back propagration.

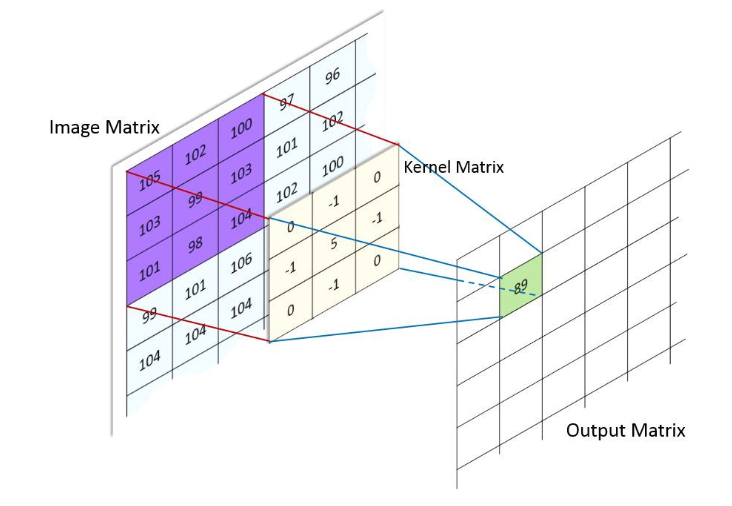
1. **Convolutional Neural Network(Cnn)**

CNN là một biến thể của NN được sử dụng rộng rãi trong lĩnh vực Computer Vision. Tên gọi của CNN được đặt theo loại của hidden layer nằm bên trong nó.

Các hidden layer của CNN thông thường bao gồm convolution layers, pooling layers, fully connected layers và normalzition layers. Thay vì sử dụng các activation function ở bên trên thì activation fucntion mà CNN sử dụng là convolution và pooling fucntion.

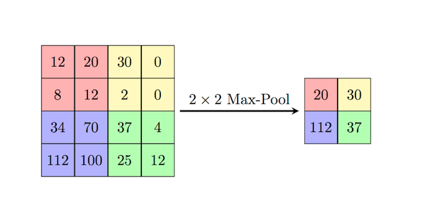


1. **Covolution**: Convolution hoạt động dựa trên 2 tín hiệu (ở 1D) hoặc 2 tín hiệu (ở 2D): 1 “input: signal (1 bức ảnh) và signal còn lại (được gọi là kernel) là 1 “filter” của bức ảnh đó. sẽ tạo ra 1 bức ảnh ouput. (vì thế covolution lấy 2 bức ảnh làm input và tạo ra 1 bức thứ 3 là output)



1. **Pooling**: Pooling là quá trình phân tách lấy mẫu. Đối tượng được thay đổi cách biểu diễn (image, hidden layer output, matrix, …) giảm số chiều và cho phép để giả định để tạo ra các feature chứa trong các sub-regions đã được giảm chiều.

Có 2 loại pooling chính: max và min pooling



Vì vậy có thể xem CNN đơn giản là deep neural network chứa các hidden layer bao gồm covolution và pooling function.

1. **CẤU TRÚC MẠNG CNN**

Mạng CNN sử dụng 3 có ý tưởng cơ bản:

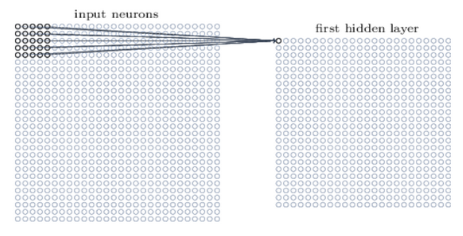
* Local receptive field
* Shared weights
* Pooling

**Local receptive field (Trường tiếp nhận cục bộ):**

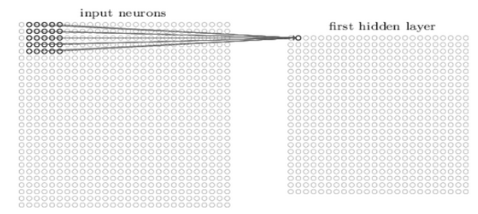
Đầu vào của mạng CNN là một ảnh. Vd ảnh có kích thước 28x28 và giá trị mỗi điểm ảnh là một ô trong ma trận. Trong mô hình NN truyền thống thì chúng ta sẽ kết nối các neuron thành 1 chuỗi với nhau.

Tuy nhiên trong CNN chúng ta không làm như vậy mà chúng ta chỉ kết nối trong vùng nhỏ của các đầu neuron đầu vào với các filter. Ta có thể tạo ra 1 hidden layer bằng cách sau:

Tạo ra neuron ẩn đầu tiên trong lớp ẩn 1



Dịch filter qua bên phải 1 cột sẽ tạo được neuron ẩn thứ 2



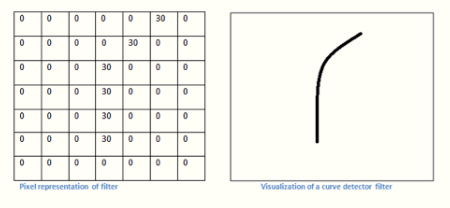
Với ảnh 28x28 và filter 5x5 ta sẽ tạo được ma trận ẩn (28 - 5 + 1) = 24x24

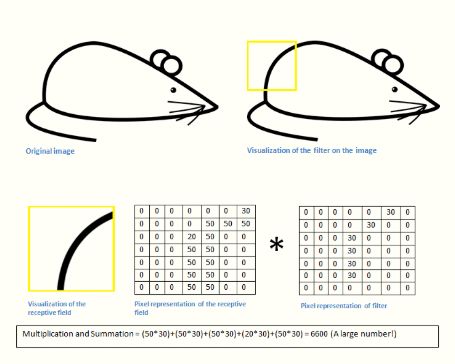
Như vậy, local receptive field thích hợp cho việc phân tách dữ liệu ảnh, giúp chọn ra những vùng ảnh có giá trị nhất cho việc đánh giá phân lớp.

**Shared weights and biases (Trọng số chia sẻ):**

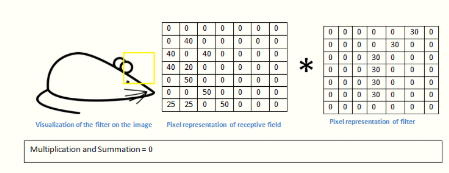
Trọng số chia sẻ giúp chúng ta có thể phát hiện được 1 feature ở các vị trí khác nhau trên ảnh.

Cho filter như bên dưới:



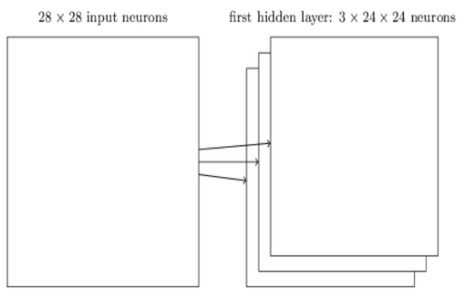
Apply filter vào ảnh trong trường tiếp nhận cục bộ: 

Cơ bản, bên trong ảnh đầu vào nếu có một feature tương tự với filter sẽ cho ra một kết quả lớn. Nếu chuyến filter sang khu vực khác không trùng với filter thì sẽ cho kết quả bằng 0:



Vì thế, convolution sẽ tìm ra được đúng đường cong trong filter dù nó nằm ở bất cứ đâu trên ảnh.

Với cấu trúc trên ta sẽ có thể phát hiện được chỉ một feature. Vì vậy nếu dùng nó cho các bài toán lớn hơn như phát hiện khuôn mặt thì ta sẽ cần nhiều hơn một feature map. Và 1 convolutional layer hoàn chỉnh sẽ bao gồm nhiều feature maps.



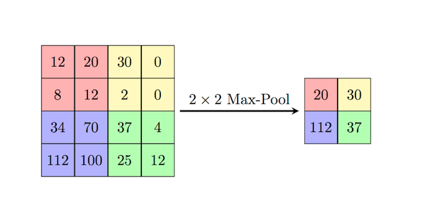
Điểm mạnh lớn nhất của sharing weights and biases là giảm mạnh số các thông số bên trong convolutional network.

Với mỗi feature map ta cần 25 (5 x 5) shared wieght, cộng thêm 1 shared bias. Vì vậy mỗi feature map sẽ có 26 thông số. Nếu có 20 feature map thì ta sẽ có 20x26 = 520 thông số cho 1 CNN bé hơn rất nhiều so với NN thông thường với lớp đầu tiên 28x28 = 784 với 30 hidden neuron thì sẽ có 784x30 +30 = 23550 thông số.

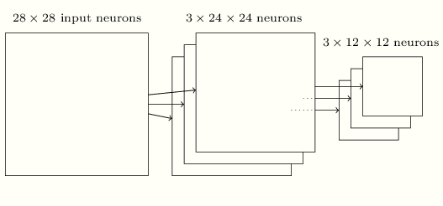
**Pooling layers (Lớp tổng hợp):**

Ngoài các lớp trên, CNN còn có các pooling layers. Pooling layer thường được dùng ngay sau các lớp convolutional. Chức năng của lớp này là đơn giản hóa giảm bớt số lượng neuron đầu ra của convolutional layer.

Chi tiết hơn, mỗi pooling layer lấy output từ convolutional layer và tạo ra các feature map thu gọn. VD: mỗi neuron bên trong pooling layer tóm tắt một vùng 2x2 các neuron ở lớp trước đó.



Nếu áp dùng pooling cho map 24x24 thì sau khi pooling sẽ chỉ còn lại 12x12



Ta có thể cho rằng max-pooling như một cách tìm feature được phát hiện ở vùng nào trên ảnh. Sau đó nó sẽ đưa ra thông tin vị trí chính xác. Một lợi ích lớn là có ít hơn nhiều các pooled feature vì vậy chúng giúp giảm số lượng lớn các thông số cho các lớp sau này.

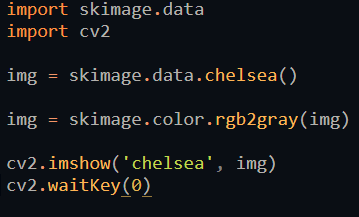
1. **CÁCH LẬP TRÌNH CONVOLUTION NEURAL NETWORK**

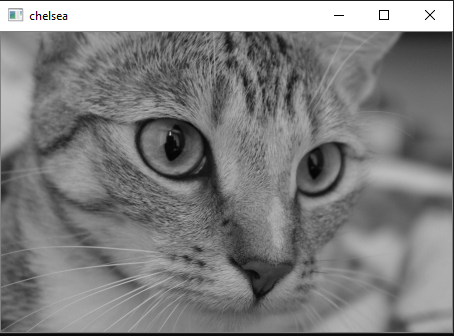
***Các bước để tạo ra CNN:***

* Đọc input image
* Chuẩn bị kernel
* Convolution layer: tích chập từng kernel với input image => feature map
* ReLU layer: Áp dụng ReLU Activation funciton vào feature map
* Max pooling layer: Áp dụng pooling vào output của ReLU layer

**Bước 1: Đọc ảnh đầu vào**

Ta sẽ dùng thư viện skimage Python và chuyển ảnh về gray

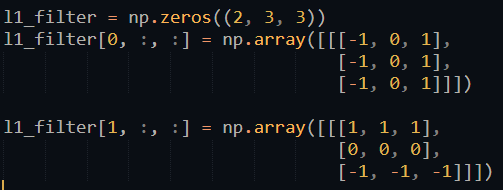




*Hình: ảnh được lấy từ thư viện skimage và chuyển từ rbg sang gray*

**Bước 2: Chuẩn bị kernel(filter)**

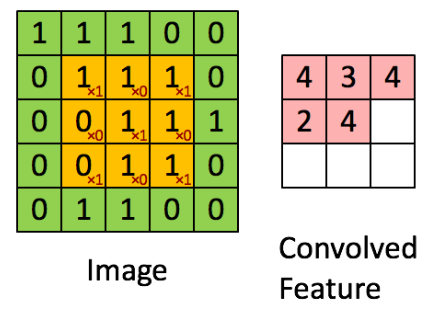
Convulution hay tích chập là nhân từng phần tử trong ma trận 3. Sliding windows hay còn gọi là kernel, filter hoặc feature detect là một ma trận vuông có kich thước nhỏ 3x3 hoặc 5x5



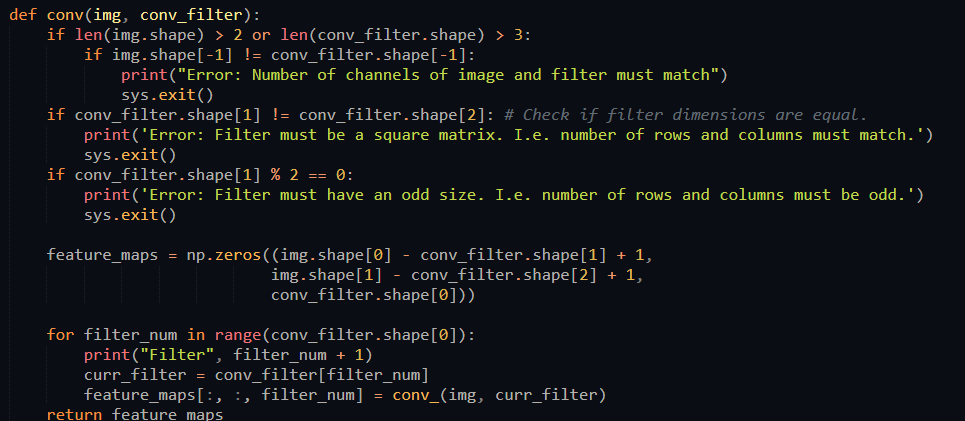
*Hình: Một mảng được tạo bao gồm 2 filter với kích thước 3x3:*

**Bước 3: Convolutional Layer**

Conolution là nhân từng phần tử của ma trận ***kernel*** với ảnh. Kết quả nhận được một ma trận được gọi là feature maps hay convolved feature



*Hình: Cách tạo ra feature map*



*Hình: Code convolution*

Function bắt đầu từ việc kiểm tra điều kiện của img và filter để có thể tiến hành covolution:

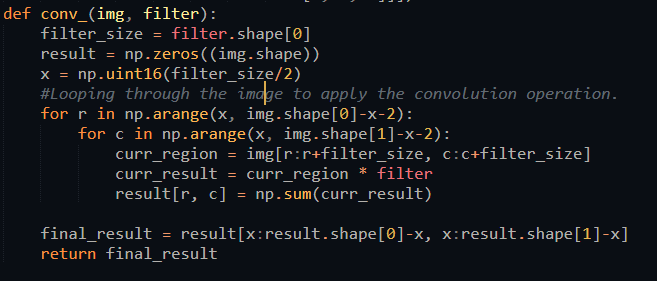
* Nếu là ảnh màu thì ***depth*** của ảnh và filter phải bằng nhau
* Filter phải là ma trận vuông và có số chiều là lẻ (3x3, 5x5, 7x7, …)

Tiếp đến sẽ tạo ra một ma trận rỗng để chứa feature image:

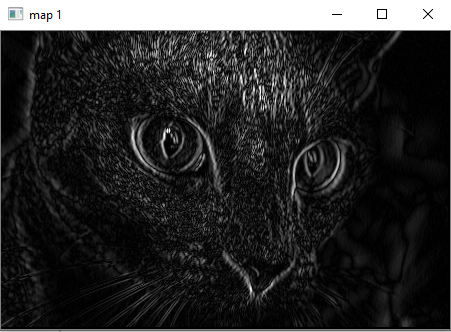
* Dimesion của feature image khi không có stride và padding

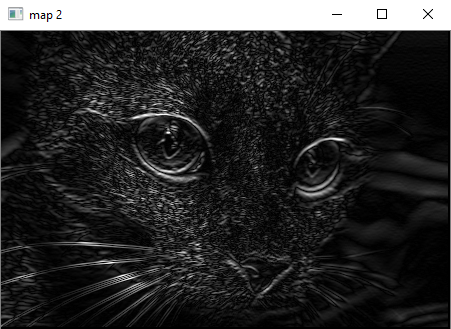
[image.x – filter.x + 1, image.y – filter.y + 1, số filter]

Sau đó sẽ convolution từng filter vào ảnh đầu vào và lưu vào feature image rỗng



*Hình: Thuật toán Convolution:*

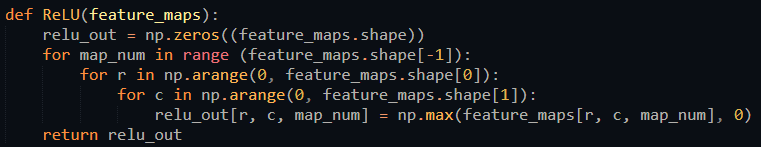




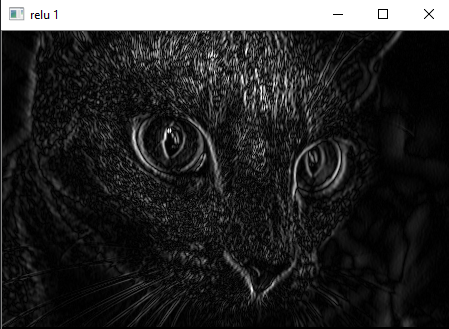
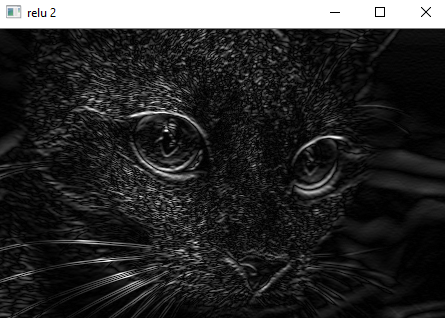
*Hình: 2 feature image từ 2 filter khác nhau*

Từng output của mỗi filter sau đó sẽ được đưa vào ReLU layer

**Bước 4: Relu Activation Function**

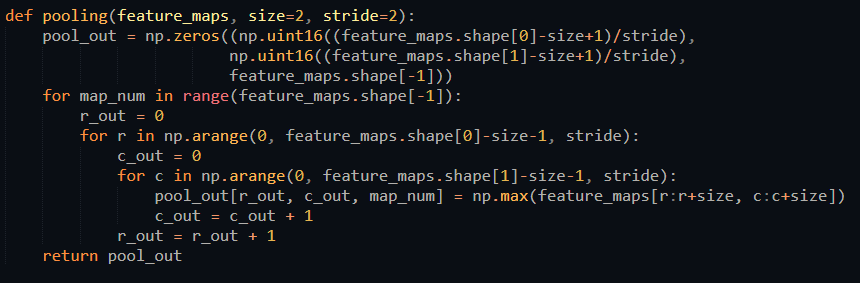


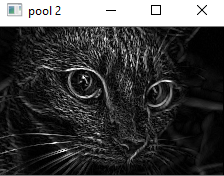
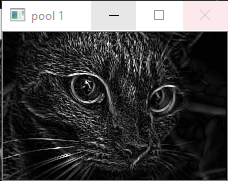
Thuật toán relu là chỉ cần lặp qua từng phần tử của feature image nếu phần từ đó lớn hơn bằng 0 thì vẫn là chính nó, nếu nó bé hơn 0 thì sẽ bằng 0



*Hình: Output của relu layer*

**Bước 5: Max Pooling Layer**



*Hình: Output của pooling layer*

*Source code:* <https://github.com/CuteBoiz/neuralnetwork/blob/master/NN1.py>

1. **NGUỒN THAM KHẢO**

* <https://www.kdnuggets.com/2018/04/building-convolutional-neural-network-numpy-scratch.html>
* <https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529>
* <https://chsasank.github.io/deep-learning-crash-course-2.html>
* <https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90>