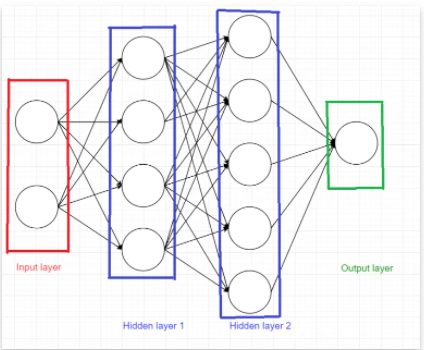
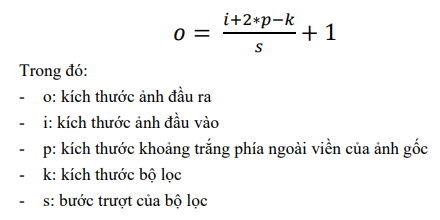
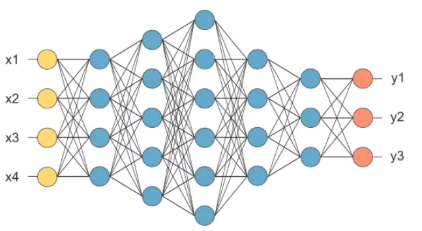
**Câu 1: Cấu trúc của một mạng CNN**

* Định nghĩa: Mạng cnn (convolutional neural network): hiểu đơn giản nó là tập hợp của các lớp Convolution chồng lên nhau và sử dụng các hàm kích hoạt phi tuyến tính (VD: Hàm ReLU, hàm tank) để kích hoạt trọng số cho các node, thông tin của node này là đầu vào của node kế tiếp (mô hình này được gọi là fully connected layer)
* Ý tưởng:
  1. Áp dụng các bộ lọc lên ảnh trước khi training mạng neural
  2. Sau khi đi qua các bộ lọc thì các đặc trưng của tấm ảnh sẽ trở lên nổi bật hơn
  3. Bộ lọc ở đây thực chất là tổ hợp của các phép nhân tích chập
  4. Kết hợp các bộ lọc lại với nhau bằng lớp tổng hợp pooling
  5. Lớp pooling sẽ gom một tập các điểm ảnh lại với nhau và cho ra một tập nhỏ hơn
* Cấu trúc của CNN:
  1. Lớp tích chập (Convolutional): Có nhiệm vụ trích xuất các đặc trưng của ảnh, output của convolutional layer đầu tiên là input của convolutional layer tiếp theo, convolutional layer bao gồm: input(dữ liệu đầu vào) và bộ lọc tích chập:
     + Bộ lọc: ví dụ một bộ lọc kích thước 3x3, di chuyển lần lượt qua từng vùng ảnh cho đến khi quét xong toàn bộ ảnh, tạo ra ảnh có kích thước nhỏ hơn hoặc bằng ảnh ban đầu. Kích thước này tùy theo khoảng trắng đc thêm ở viền
     + Tổng quan: Gỉa sử input của một convolutional layer là tensor kích thước H \* W \* D, Kernel kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.
     + => Output của layer là tensor 3 chiều có kích thước: () \*  () \* K
     + Output của convolutional layer sẽ đi qua các hàm kích hoạt trước khi trở thành input của convolutional layer kia
  2. Lớp kích hoạt phi tuyến tính (Rectified Linear Unit)
     + Lớp này được xây dựng với ý nghĩa đảm bảo tính phi tuyến của mô hình huấn luyện sau khi đã thực hiện một loạt các phép tính toán tuyến tính qua các lớp Tích chập. Lớp Kích hoạt phi tuyến nói chung sử dụng các hàm kích hoạt phi tuyến như ReLU hoặc sigmoid, tanh… để giới hạn phạm vi biên độ cho phép của giá trị đầu ra.
     + Phi tuyến tính có nghĩa là đầu ra không thể được sao chép từ tổ hợp tuyến tính của các đầu vào, nếu không có các hàm kích hoạt phi tuyến, thì mạng nơ-ron của chúng ta dù có nhiều lớp vẫn sẽ có hiệu quả như một lớp tuyến tính mà thôi
  3. Lớp tổng hợp (Pooling):
     + có chức năng tổng hợp các bộ lọc lại với nhau, pooling sẽ gom 1 tập các điểm ảnh và cho ra kích thước nhỏ hơn, làm giảm dần kích thước không gian để giảm số lượng tham số và tính toán của ảnh. Ví dụ maxpooling 2x2 thì sẽ chọn ra từ ảnh đầu vào giá trị điểm ảnh lớn nhất trong các nhóm điểm ảnh với kích thước 2x2.
     + Lớp lấy mẫu cũng sử dụng một cửa sổ trượt để quét toàn bộ các vùng trong ảnh tương tự như lớp Tích chập, và thực hiện phép lấy mẫu thay vì phép tích chập – tức là ta sẽ chọn lưu lại một giá trị duy nhất đại diện cho toàn bộ thông tin của vùng ảnh đó.
  4. Lớp kết nối đầy đủ (Fully-connected):
     + tất cả các điểm ảnh được kết nối đầy đủ với node trong lớp tiếp theo
     + output của layer cuối cùng sẽ được là phẳng thành vector và đưa vào một lớp được kết nối như một mạng nơ-ron.
     + Với FC layer được kết hợp với các tính năng lại với nhau để tạo ra một mô hình. Cuối cùng sử dụng softmax hoặc sigmoid để phân loại đầu ra.

**Câu 2: Lớp pooling để làm gì, maxpooling, average pooling là để làm gì**

* Ý nghĩa đầu tiên khi mà nói đến lớp pooling đó là: tổng hợp các điểm ảnh lại với nhau và cho ra tập nhỏ hơn tăng hiệu suất khi training
* Ý nghĩa quan trọng hơn là: làm giảm kích thước của ảnh những vẫn giữ lại được những đặc trưng nổi bật của ảnh, càng thu nhỏ thì những chi tiết quan trọng càng mất đi, càng hiển thị được ra cái chi tiết nổi bật mà có trong ảnh (VD: Khi hỏi trong ảnh cái gì là đặc trưng – thì chính là thằng nào là nổi bật nhất, khi giảm kích thước ảnh thì cái nào bé cái nào mờ nó sẽ bị xóa và giữ lại cái nổi bật từ đó nhận biết được cái nào nổi bật nhất)
* Max pooling là tính giá trị lớn nhất trong vùng đầu vào 2x2, như vậy qua lớp này sẽ giảm đi 1 nửa
  + Filter size: thường là ma trận 3x3 hoặc 5x5
  + Epochs: số lượng chu kì lặp đi lặp lại của CNN
  + Batch: để cải thiện việc tính toán trong CNN, ta sẽ dùng phương pháp chia nhỏ các hình ảnh ban đầu thành nhóm nhỏ hơn

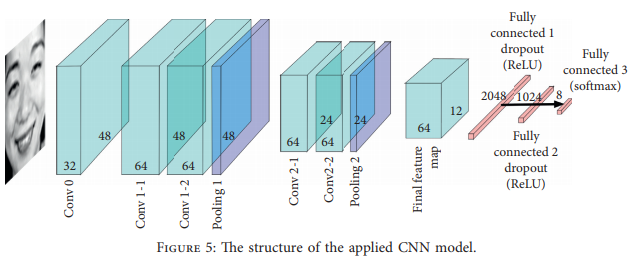
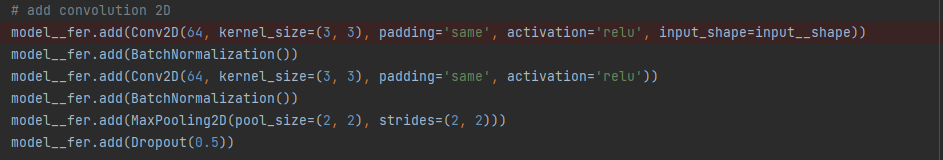
**Câu 3: Tại sao lại phải trải ảnh ra thành mảng 1 chiều ví dụ ảnh 4x4 -> 1x16**

* Đây chỉ là một cách lưu trữ để cho dễ biểu diễn và xử lý hơn
* Phải trải ảnh từ 2 chiều sang 1 chiều vì: ví dụ nếu có 1 ảnh thì chỉ cần 4x4 thôi, nếu có 3 ảnh thì cần đến 3 cái ma trận 2 chiều thì n ảnh thì n mảng 2 chiều thì gom lại thành 3 chiều mỗi chiều 1 ảnh mỗi ảnh 2 chiều => khá phức tạp cho vc biểu diễn, còn 1 ảnh kéo thành 1 dòng thì chỉ cần 1 ma trận 2 chiều
* Nhờ đi khi training cho 1 mô hình chỉ nhận mảng 1 chiều thì ta hoàn toàn có thể chuyển ảnh 2 chiều về 1 chiều để nhét được vào trong model

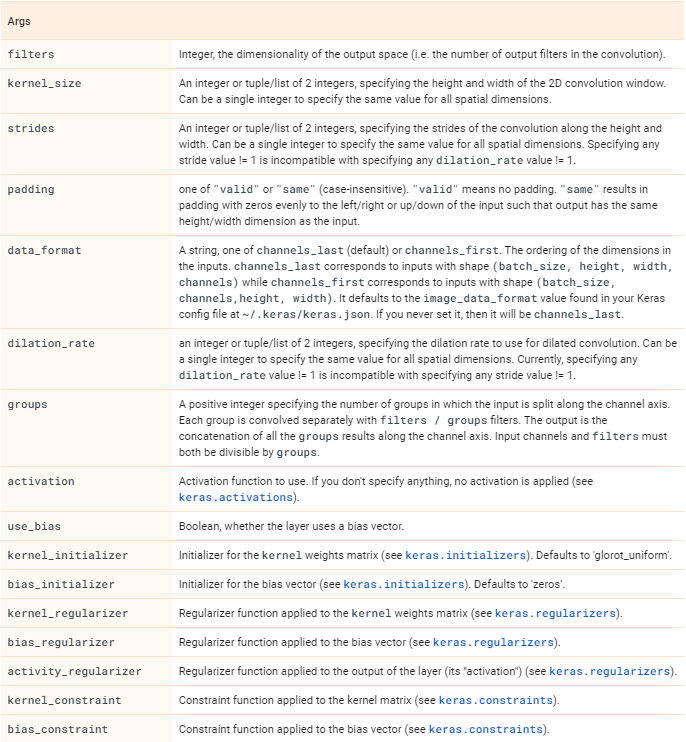
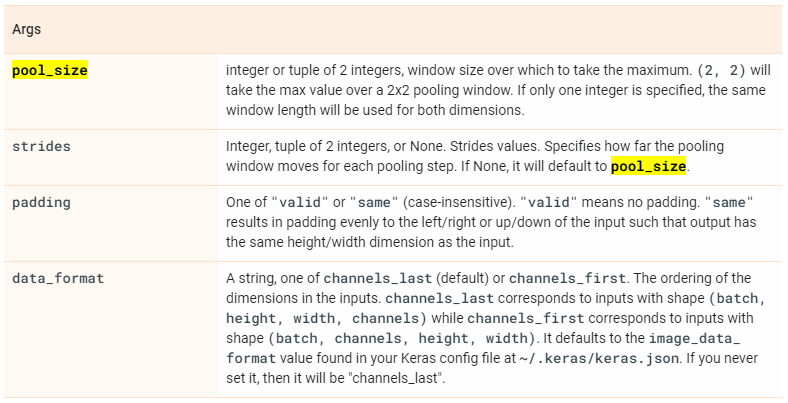
**Câu 4: Tại sao khi nhận ảnh vào thì và xử lý ảnh thì lại có 1 bước là chia 255**

* Về nguyên tắc mỗi một điểm ảnh n là 1 gtri nguyên nằm trong khoảng từ 0-255 và khi xử lý thì n phải dùng các phép toán để xly chứ ko xl số nguyên
* Các kỹ thuật xử lý bắt buộc ảnh phải là số thực nằm trong đoạn 0-1
* Ví dụ phép xử lý bình phương điểm ảnh, nếu dùng số nguyên là 200 ^2 lên là 4k nằm ngoài khoảng 255
* Do đó chuyển 200 về đoạn 0-1 thì 200/255 -> 0.78 do đó nếu muốn chuyển lại cũng chỉ nhân ngược lại -> 200 ^2 = 155
* Khi số nằm trong khoảng 0 1 thì n vẫn ở trong khoảng đó (biểu diễn thực), nếu 255 nó sẽ vượt qua 255 (biểu diễn nguyên)
* Dẫn đến sự tương đồng giữa các data point nhiều hơn

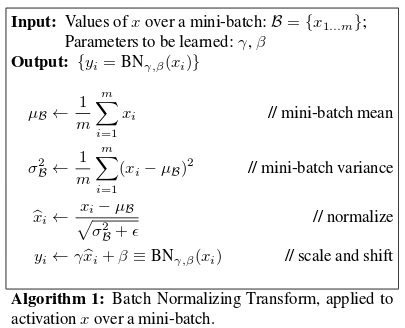
**Câu 5: Tại sao lại có 4 lớp?, sao không phải là 5 là 6?, ý nghĩa từng block?**

* Để trả lời cho câu hỏi bao nhiêu lớp thì nó còn phụ thuộc vào độ lớn vào số lượng của data, ví dụ có 1024 **params** thì nó sẽ phải tương đồng với **data point**, nếu ko tg đồng sẽ xảy ra trường hợp overfit (quá thông minh params >>> data point) hoặc underfit (params <<< data point ngu dốt mà lại học cái quá khó)
* Data point ở đây vi dụ như là các điểm ảnh
* Sau mỗi một lần đi qua các lớp conv thì kích thước của ảnh giảm nhưng số lượng channel sẽ tăng lên theo ý tưởng của VGG16 (5 lớp)
* Thường thì lớp đầu tiên sẽ là giữ lại hầu hết thông tin của ảnh, lớp đầu thường sẽ trích xuất viền và cạnh của ảnh
* Càng đi sâu các lớp về sau thì khó nhận ra đc là ảnh gì bởi vì: càng về sâu, các feature map biểu diễn những hình trừu tượng hơn
* Các thông tin về sau sẽ càng thưa thớt hơn. Điều này là hợp lý vì ban đầu phải giữ lại các cạnh của ảnh, ảnh nào mà chả phải có cạnh, càng về sâu thì phải biểu diễn những cái nó phức tạp hơn thứ mà không phải hình ảnh nào cũng c
* 1. Xây dựng bài toán 2. Chuẩn bị dữ liệu (dataset) 3. Xây dựng model 4. Định nghĩa loss function 5. Thực hiện backpropagation và áp dụng gradient descent để tìm các parameter gồm weight và bias để tối ưu loss function. 6. Dự đoán dữ liệu mới bằng model với các hệ số tìm được ở trên

**Câu 6: Lớp conv2d để làm gì?, các tham số trong đó là gì?, activation trong conv2d là gì?**

* Các tham số trong conv2d: ví dụ
* model\_\_fer.add(Conv2D(64, kernel\_size=(3, 3), padding='same', activation='relu', input\_shape=input\_\_shape))
* Ở đây có 64 kernel và mỗi kernel có kích thước là 3x3, giữ lại cạnh của anh, hàm kích hoạt phi tuyến relu
* Các thông số trong maxpooling2d:

**Câu 7: Lớp batchnormalization là để làm gì**

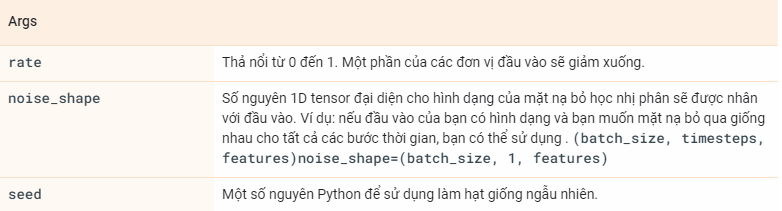
* ****Chuẩn hóa lại data để nó đồng nhất với input ban đầu
* **Batch Normalization** là một phương pháp hiệu quả khi training một mô hình mạng nơ ron. Mục tiêu của phương pháp này chính là việc muốn chuẩn hóa các feature (đầu ra của mỗi layer sau khi đi qua các activation) về trạng thái **zero-mean** với độ lệch chuẩn 1. Vậy hiện tượng ngược lại đó là **non-zero mean** có ảnh hưởng như thế nào đến việc training mô hình:
  + Thứ nhất: **Non zero mean** là hiện tượng dữ liệu không phân bố quanh giá trị 0, mà dữ liệu có phần nhiều giá trị lớn hơn không, hoặc nhỏ hơn không. Kết hợp với vấn đề high variance khiến dữ liệu trở nên có nhiều thành phần rất lớn hoặc rất nhỏ. Vấn đề này rất phổ biến khi training các mạng nơ ron với số layer sâu. Việc feature không phân phối trong những khoảng ổn định (giá trị to nhỏ thất thường) sẽ có ảnh hưởng đến quá trình tối ưu của mạng. Vì như chúng ta đã biết việc tối ưu một mạng nơ ron sẽ cần phải sử dụng đến tính toán đạo hàm. Giả sử như một công thức tính layer đơn giản là y = (Wx + b)*y*=(*Wx*+*b*) thì đạo hàm của y*y* theo w*w* có dạng: dy = dWx*dy*=*dWx*. Như vậy giá trị x*x* ảnh hưởng trực tiếp đến giá trị của đạo hàm. Do đó nếu x*x* mang các giá trị thay đổi không ổn định dẫn đến đạo hàm sẽ có thể bị quá lớn, hoặc quá nhỏ dẫn đến việc learning model không được ổn định. Và điều đó cũng đồng nghĩa với việc chúng ta có thể sử dụng các learning rate cao hơn trong quá trình training khi sử dụng Batch Normalization.
  + Thứ hai: **Batch normalization** có thể giúp chúng ta tránh được hiện tượng giá trị của x*x* rơi vào **khoảng bão hòa** sau khi đi qua các hàm kích hoạt phi tuyển. Vậy nên nó đảm bảo rằng không có sự kích hoạt nào bị vượt quá cao hoặc quá thấp. Điều này giúp cho các weights mà khi không dùng **BN** có thể sẽ không bao giờ được học thì nay lại được học bình thường. Điều này giúp chúng ta làm giảm đi sự phụ thuộc vào giá trị khởi tạo của các tham số.
  + Thứ ba: **Batch Normalization** còn có vai trò như một dạng của **regularization** giúp cho việc giảm thiểu overfiting. Sử dụng batch normalization, chúng ta sẽ không cần phải sử dụng quá nhiều dropput và điều này rất có ý nghĩa vì chúng ta sẽ không cần phải lo lắng vì bị mất quá nhiều thông tin khi dropout weigths của mạng. Tuy nhiên vẫn nên sử dụng kết hợp cả hai kĩ thuật này

**Câu 8: Vẽ và mô tả lại dữ liệu sau khi đi qua từng lớp từng câu lệnh**

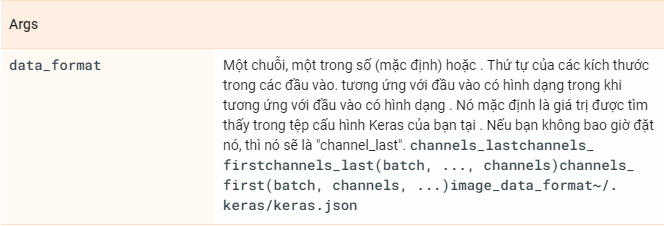
**Câu 9: Tại sao khi training model người ta lại phải tráo ảnh lộn xộn lên chứ ko để thành các phần riêng biệt như ban đầu?**

* Thường thì các bộ dữ liệu chia ra thành từng phần ví dụ: tập hợp những ảnh có cảm xúc buồn bã, … thì khi training model theo từng phần một mà các ảnh giống nhau => điều này làm cho ảnh hưởng đến tính tổng quát của ảnh thậm chí tráo hết nhãn

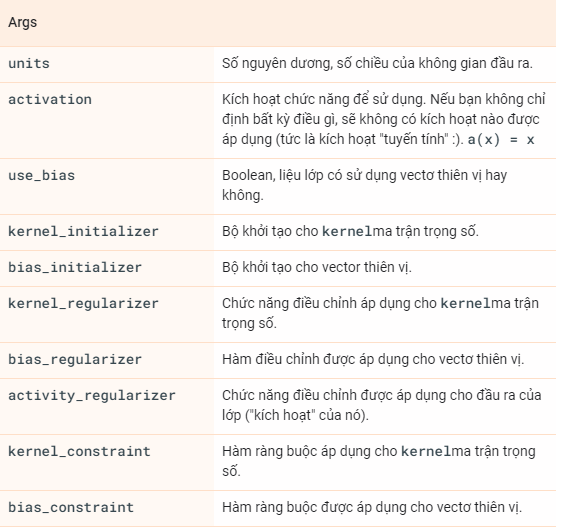
**Câu 10: Tìm hiểu về Dropout?**

* Hiểu đơn giản là, 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).
* Về mặt kỹ thuật, tại mỗi giai đoạn huấn luyện, mỗi node có xác suất bị bỏ qua là 1-p và xác suất được chọn là p
* Vậy tại sao cần dropout tại sao cần loại đi một vài unit
* => Câu trả lời cho câu hỏi này là **để chống over-fitting**
* Trước đây trong ML, người ta thường sử dụng regularization để ngăn chặn overfiting, regularization làm giảm overfiting bằng cách thêm vào những loss function yếu tố **Laplacian** để features weights giảm đi sự phụ thuộc lẫn nhau

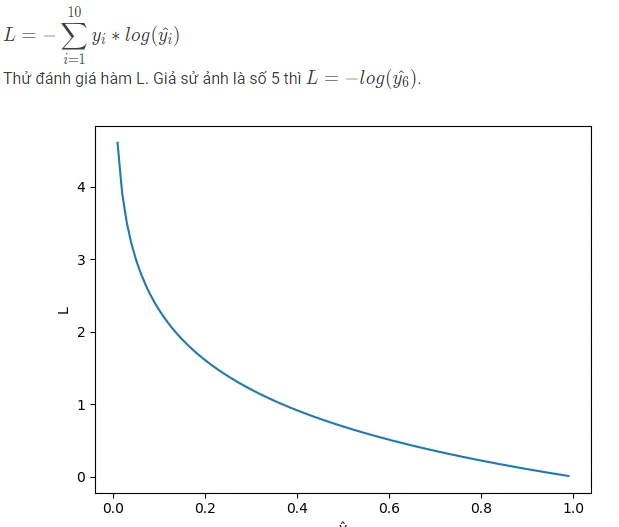
**Câu 11: Tìm hiểu về Flatter layer**

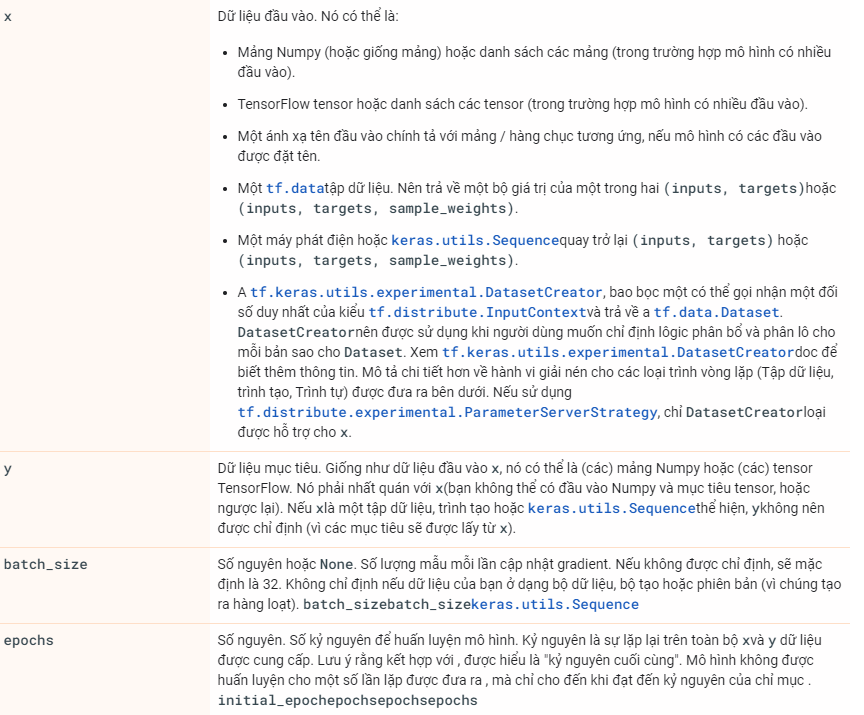
* Flatter đơn giản chỉ là một lớp dùng để làm phẳng dữ liệu rồi cung cấp cho mạng neural để xử lý

**Câu 12: Tìm hiểu về lớp Dense**

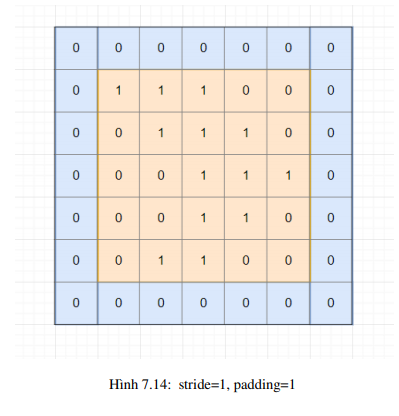
* Khái niệm: dense là một lớp trong cnn được kết nối sâu, có nghĩa là: mỗi một neuron trong lớp dense sẽ nhận input đầu vào từ tất cả các lớp trước đó
  + lớp này sẽ thực hiện nhân các vector ma trận, các giá trị sử dụng trong ma trận thật ra chỉ là các tham số truyền vào có thể training và cập nhật được với sự trợ giúp của lan truyền ngược
  + đầu ra của dense layer là một vecto m chiều. Do đó, về cơ bản thì dense làm thay đổi kích thước của một vector. Các lớp dense cũng hoạt động theo các cách như xoay, chia tỷ lệ, dịch trên vector.
* Thông số truyền vào trong tensorflow

**Câu 13: Compile**

* Ý nghĩa: cấu hình mô hình để tiến hành đào tạo nhóm các lớp thành một đối tượng với các tính năng huấn luyện và suy luận. Để đào tạo một mô hình với fit() , bạn cần chỉ định một hàm mất mát, một trình tối ưu hóa và tùy chọn, một số chỉ số cần theo dõi. Bạn chuyển chúng vào mô hình dưới dạng đối số cho phương thức compile() :
* Các tham số truyền vào:
  + Loss: loss function, ví dụ trong bài đang sử dụng là categorical\_crossentropy, giải thích về categorical\_crossentropy: Loss function giống như một hình thức để bắt model đóng phạt mỗi lần nó dự đoán sai, và số mức phạt tỉ lệ thuận với độ trầm trọng của sai sót, thì có rất nhiều cách xây dựng hàm loss function ví dụ như categorical\_crossentropy thì sẽ có cách xây dựng sau hàm L nhỏ khi giá trị model dự đoán gần đúng với giá trị thật và rất lớn khi model dự đoán sai, hay nói cách khác L càng nhỏ thì model dự đoán càng gần với giá trị thật => bài toán tìm L nhỏ nhất
  + Optimizer: (trả lời riêng ở câu hơi bên dưới)
  + Metrics: là một hàm chức năng đánh giá hiệu suất của model, các tham số tương tự như loss function, nó tính tần suất dự đoán các nhãn bằng nhau. một phép toán tổng hợp đơn giản chia total cho count.: ví dụ điền hình như accuracy: Chỉ số này tạo ra hai biến cục bộ total và count được sử dụng để tính toán tần suất y\_pred phù hợp với nó y\_true. Tần số này cuối cùng được trả về dưới dạng binary accuracy:



**Câu 14: Stride là gì**

* Khi thực hiện padding ta thu đc ma trận mới có cùng kích thước, ta gọi stride là 1
* Tuy nhiên nếu stride=k (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử x1+i\*k, 1+j\*k. Ví dụ k = 2. => hiểu đơn giản là từ bắt đầu vị trí x11 thì nó sẽ nhảy k bước theo chiều dọc và theo chiều ngang cho đến hết ma trận
* Hiểu đơn giản là bắt đầu từ vị trí x11 sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X. Kích thước của ma trận Y là 3\*3 đã giảm đi đáng kể so với ma trận X.

**Câu 15: chuyển ảnh rgb về ảnh đa mức xám (đen trắng)**

* Mỗi pixel trong ảnh màu được biểu diễn bằng 3 giá trị (r,g,b) còn trong ảnh xám chỉ cần 1 giá trị x để biểu diễn. Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng công thức: x = r \* 0.299 + g \* 0.587 + b \* 0.114.

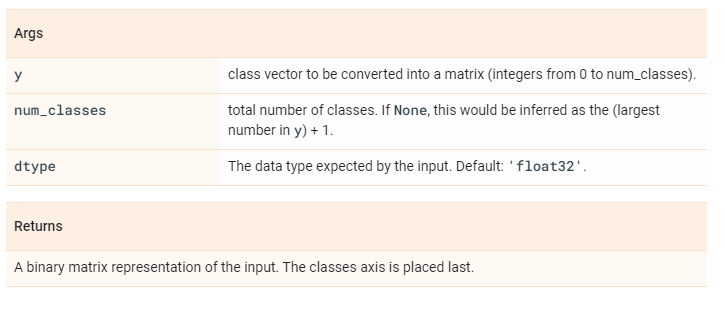
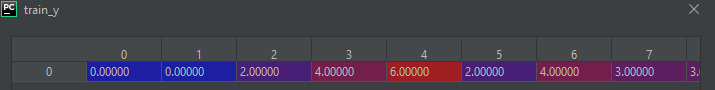
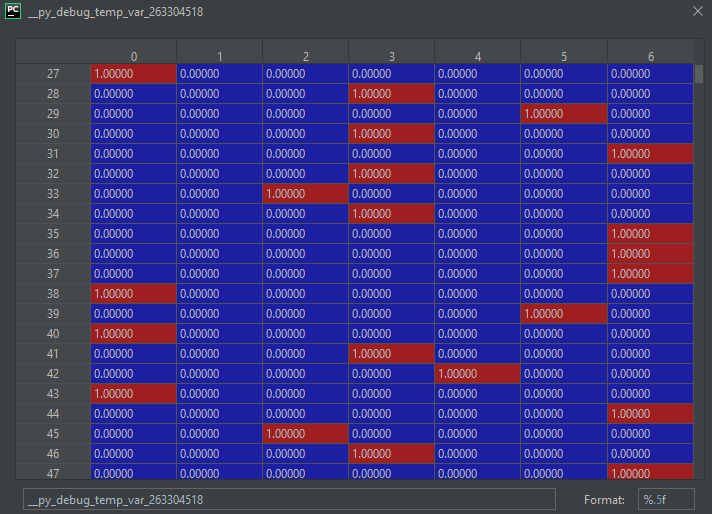
**Câu 16: Loss function và activaction là gì**

* Loss function hay còn gọi là hàm mất mát, thể hiện một mối quan hệ giữa y\* (là kết quả dự đoán của model) và y (là giá trị thực tế). Ví dụ ta có hàm loss như sau: f(y) = (y\* - y)^2. Khi đó người ta đưa vào hàm loss function này mục đích là để tối ưu model của mình sao cho tốt nhất, hay cũng dùng để đánh giá độ tốt của model , y\* (là kết quả dự đoán của model) càng gần y (là giá trị thực tế) thì càng tốt. Tức là dựa vào loss function, khi đó chúng ta có thể tính ra gradient descent để tối ưu loss function càng về gần 0 càng tốt
* Hàm kích hoạt (activation function) mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh, là những hàm phi tuyến được áp dụng vào đầu ra của các nơ-ron trong tầng ẩn của một mô hình mạng ( có nhiệm vụ là chuẩn hoá output của neura) và được sử dụng làm input data cho tầng tiếp theo
* **Chuyện gì sẽ xảy ra nếu không có các hàm phi tuyến này ?** Hãy tưởng tượng rằng thay vì áp dụng 1 hàm phi tuyến, ta chỉ áp dụng 1 hàm tuyến tính vào đầu ra của mỗi neuron. Vì phép biến đổi không có tính chất phi tuyến, việc này không khác gì chúng ta thêm một tầng ẩn nữa vì phép biến đổi cũng chỉ đơn thuần là nhân đầu ra với các weights. Với chỉ những phép tính đơn thuần như vậy, trên thực tế mạng neural sẽ không thể phát hiện ra những quan hệ phức tạp của dữ liệu (ví dụ như: dự đoán chứng khoán, các bài toán xử lý ảnh hay các bài toán phát hiện ngữ nghĩa của các câu trong văn bản). **Nói cách khác nếu không có các activation functions, khả năng dự đoán của mạng neural sẽ bị giới hạn và giảm đi rất nhiều, sự kết hợp của các activation functions giữa các tầng ẩn là để giúp mô hình học được các quan hệ phi tuyến phức tạp tiềm ẩn trong dữ liệu.**
* Tại sao hàm activation phải non-linear? Điều gì xảy ra nếu hàm linear activation được sử dụng? Các activation function phải là nonlinear (phi tuyến), vì nếu không, nhiều layer hay một layer cũng là như nhau. Ví dụ với hai layer trong Hình 2, nếu activation function là một hàm linear (giả sử hàm f(s) = s) thì cả hai layer có thể được thay bằng một layer với ma trận hệ số W = W1xW2 (tạm bỏ qua hệ số bias)

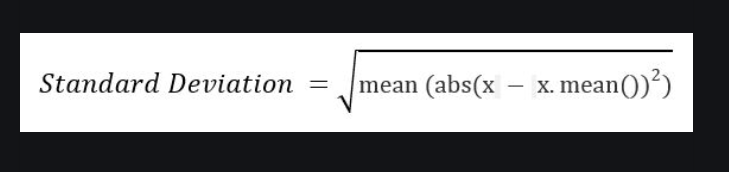
**Câu 17: Tìm hiểu về optimizer**

* Trước khi đi sâu vào vấn đề thì cần hiểu thế nào là **thuật toán tối ưu (optimizers)**. Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features ( hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model. Nhưng vấn đề là "học" như thế nào? Cụ thể là weights và bias được tìm như thế nào! Đâu phải chỉ cần random (weights, bias) 1 số lần hữu hạn và hy vọng ở 1 bước nào đó ta có thể tìm được lời giải. Rõ ràng là không khả thi và lãng phí tài nguyên! Chúng ta phải tìm 1 thuật toán để cải thiện weight và bias theo từng bước, và đó là lý do các thuật toán optimizer ra đời.
* Các thuật toán tối ưu: Gradient Descent(GD), Adam: Adam là sự kết hợp của Momentum và RMSprop . Nếu giải thích theo hiện tượng vật lí thì Momentum giống như 1 quả cầu lao xuống dốc, còn Adam như 1 quả cầu rất nặng có ma sát, vì vậy nó dễ dàng vượt qua local minimum tới global minimum và khi tới global minimum nó không mất nhiều thời gian dao động qua lại quanh đích vì nó có ma sát nên dễ dừng lại hơn.

**Câu 18: to\_categorical**

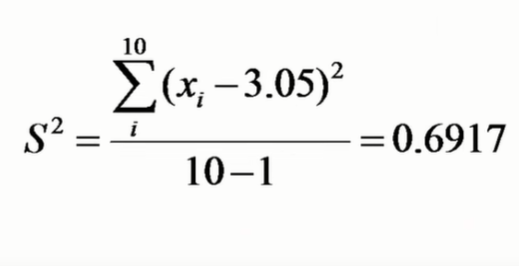
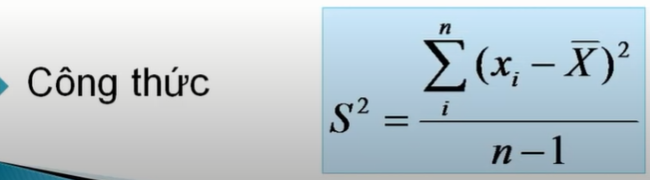
* Mục đích: chuyển một vector lớp số nguyên thành ma trận lớp nhị phân
* tf.keras.utils.to\_categorical(  
      y, num\_classes=None, dtype='float32'  
  )
* Các tham số truyền vào: vectơ lớp được chuyển đổi thành một ma trận (các số nguyên từ 0 đến num\_classes).
* Vậy làm sao để chuyển từ 1 số thành số nhị phân:
  + Như kiểu chuyển 1 sang nhị phân là 0001 nhưng ở đây n sẽ phụ thuộc vào numlable ví dụ num lable là 7 thì sẽ là 0000001
  + Và khi kết hợp lại ta được một ma trận nhịn phân
  + Ví dụ:
  + Ví dụ: 0 -> 1 0 0 0 0 0
  + 2-> 0 0 1 0 0 0
  + Cái lượng số kia phụ thuộc vào numlable

**Câu 19: Câu lệnh np.mean và np.std**

* numpy.mean(arr, axis = None): Tính giá trị trung bình cộng (trung bình) của dữ liệu đã cho (phần tử mảng) dọc theo trục xác định. Giống phương sai
* Các tham số truyền vào: mean(a, axis=None, dtype=None, out=None, keepdims=np.\_NoValue)
  + a:  array\_like: Mảng chứa các số có giá trị trung bình được mong muốn. Nếu a không phải là một mảng, một chuyển đổi sẽ được thử.
  + Axis: Trục hoặc các trục dọc theo đó các phương tiện được tính toán. Mặc định là tính giá trị trung bình của mảng phẳng.*Mới trong phiên bản 1.7.0.* Nếu đây là một bộ int, một giá trị trung bình được thực hiện trên nhiều trục, thay vì một trục hoặc tất cả các trục như trước đây.
  + Type: kiểu dữ liệu mặc định là float64 dấu phẩy động
  + Out: mảng đầu ra, mặc định là none,
  + Keepdims: nếu là true các trúc đc giảm xuống sẽ đc giữ lại trong kq là các kích thước có kích thước là 1
* numpy.std (arr, axis = None): Tính độ lệch chuẩn của dữ liệu đã cho (các phần tử mảng) dọc theo trục được chỉ định (nếu có) ..
* Độ lệch chuẩn (SD) được đo lường như là mức độ lan truyền của phân phối dữ liệu trong tập dữ liệu nhất định.
  + x = 1 1 1 1 1
  + Standard Deviation = 0 .
  + y = 9, 2, 5, 4, 12, 7, 8, 11, 9, 3, 7, 4, 12, 5, 4, 10, 9, 6, 9, 4
  + **Step 1 :** Mean of distribution 4 = 7
  + **Step 2 :** Summation of (x - x.mean())\*\*2 = 178
  + **Step 3 :** Finding Mean = 178 /20 = 8.9
  + This Result is **Variance.**
  + **Step 4 :** Standard Deviation = sqrt(Variance) = sqrt(8.9) = 2.983..
  + x = 1 1 1 1 1

**Câu 20: Bổ trợ cho câu 19 tính phương sai và độ lệch chuẩn:**

* Phương sai là gì: ps của 1 bảng số liệu đặc trưng cho độ phân tán của các số liệu trong tập dữ liệu so với giá trị trung bình, bộ số có phương sai càng nhỏ thì đó là bộ số có các giá trị gần với giá trị trung bình,
* Giống như việc ta có 1 bộ số và có gia trị trung bình của bộ số đó và đặt ra câu hỏi từng số liệu quan sát trong bộ số liệu biến thiên ntn xung quanh gtri trung bình
* Lấy cột 1 trừ đi gt trung bình -> ra cột 2, tại sao ko tính trung bình của cột t2 như đã làm vs cột 1 vì gt tb ở cột 2 luôn luôn = 0 -> bình phương lên mới đc -> tính gtri ở cột đỏ vs mẫu là 10-1=9 và kết quả ra đc 0.6917 -> cuối cùng đưa ra đc công thức như trong hình



* Độ lệch chuẩn = căn bậc 2 của phương sai nhưng lấy giá trị dương

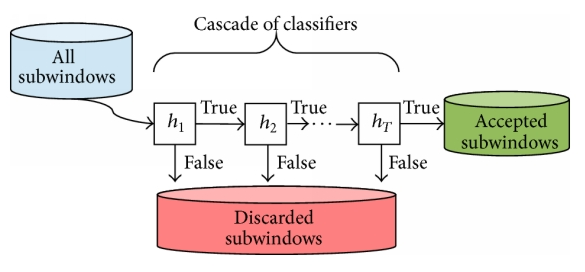
### **Reshape:** Khi làm việc với ma trận, chúng ta sẽ phải thường xuyên làm việc với các phép biến đổi kích thước của ma trận. Phép biến đổi kích thước có thể coi là việc sắp xếp lại các phần tử của một ma trận vào một ma trận khác có tổng số phần tử như nhau, sử dụng reshape để định hình lại theo kích thước mong muốn. ví dụ có mảng 3 chiều kích thước 2x10x8 muốn định hình lại thành 5x5x8 thì np.reshape(r, shape=(5,5,8)) hoặc chỉ cần np.reshape(r, shape=(5,5,-1)) chiều còn lại n tự xác định

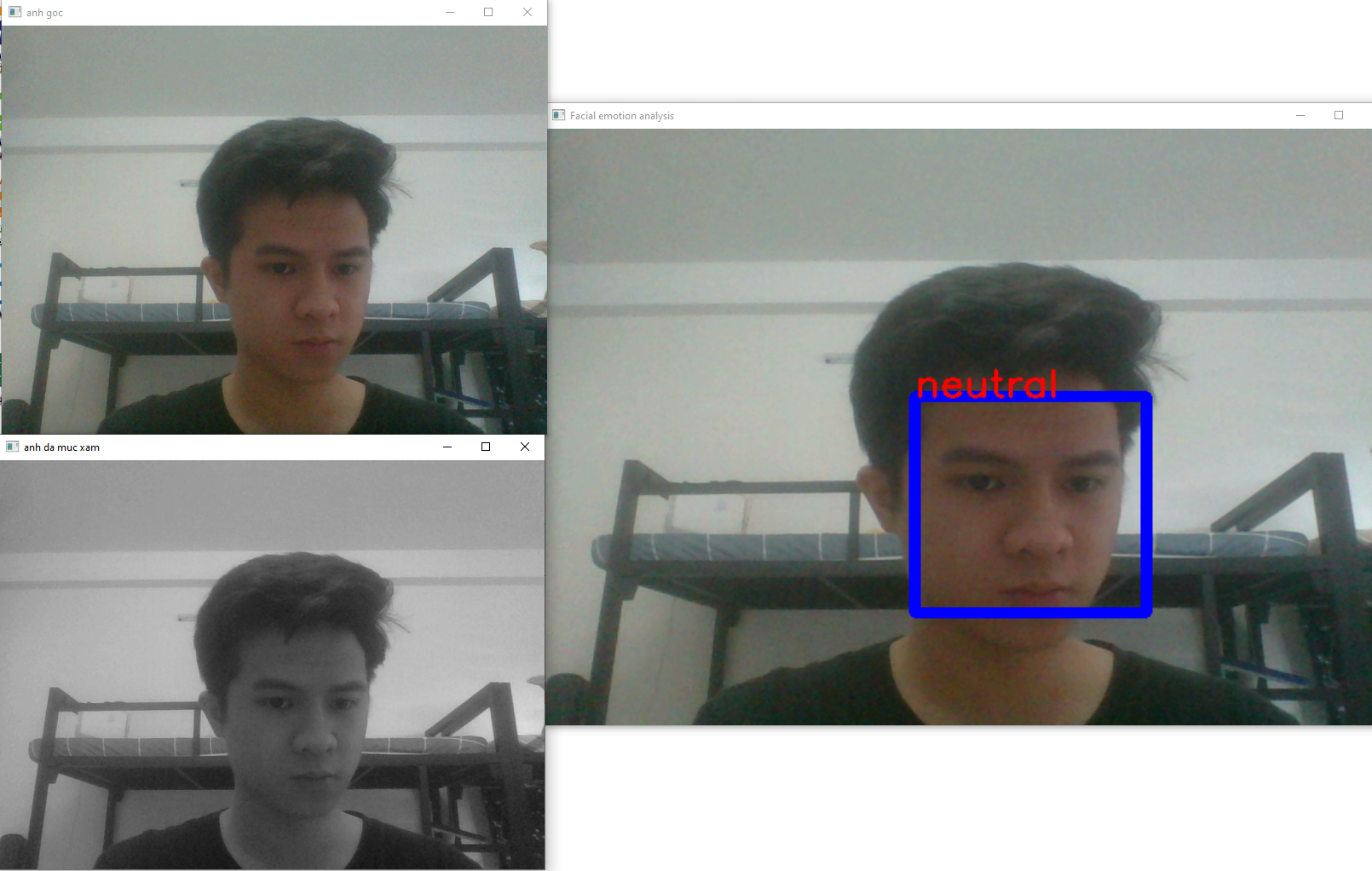
* Shape: đưa ra kích thước của mảng

**Câu 21: Các câu lệnh sử dụng trong project**

* Model\_from\_json: đưa vào một tệp model json và nó sẽ phân tích cú pháp mô hình này và trả về một ví dụ của mô hình

**Câu 22: Series về nhận diện khuân mặt**

* 1. **Haarlike:** trong 1 bức ảnh đầu vào thì khuân mặt đc đặc trưng bởi tập các pixel trong vùng khuôn mặt tạo nên sự khác biệt với vùng pixel khác, vì sử dụng các pixel riêng lẻ ko hiệu quả mà phải hết hợp lại nên các nhà nghiên cứu đưa ra ý tưởng kết hợp các pixel để tạo nên đặc trưng có khả năng phân loại các vùng của khuân mặt => 1 trong số đó là đặc trưng haarlike,
* Đặc trưng haarlike được tạo thành bằng việc kết hợp các hình chữ nhật đen, trắng với nhau theo một trật tự, một kích thước nào đó.
* Có 4 loại đặc trưng haarlike cơ bản đc chia làm 3 tập: đặc trưng cạnh, đường, xung quanh tâm.
* ta tính sự chênh lệch giữa tổng của các pixel của các vùng đen và các vùng trắng theo công thức sau: vùng đen(pixel) – Tổngvùng trắng(pixel)
* Kernel là các bộ lọc, bây h hình dung mỗi 1 kernal dùng để tính toán nhiều features thì ảnh 24x24 thôi cũng phải lên đến hơn 160000 feature, sau đó họ giảm xuống còn 6000 feature, chúng ta có tầm 160k+ các bộ lọc như vậy cơ! Tuy nhiên, chúng ta có thể sử dụng Adaboost (adaptive boosting) để kết hợp các bộ lọc trên khi bạn có các classifier yếu khác nhau, kết hợp chúng để tạo thành một classifier mạnh các bộ lọc Haar kể cả sau Adaboost như trên vẫn chỉ bắt được những đặc trưng rất cơ bản, và để nhận ra một khuôn mặt thì chúng ta cần tầm 6000 các đặc trưng như vậy! Vậy chúng ta cần có một cách để vote xem cửa sổ đó có chứa mặt không, mà vẫn phải xử lý đủ nhanh cho cả 6000 đặc trưng đó: câu trả lời là [Cascade of Classifiers](https://viblo.asia/p/en.wikipedia.org/wiki/Cascading_classifiers),
* Việc cascade đó được thiết kế như sau: trong 6000+ đặc trưng đó, chia chúng ra thành rất nhiều bước. Trong đó, mỗi lần cửa sổ trượt qua một vùng bước ảnh, từng bước một sẽ được xử lý: nếu bước 1 nhận đó là mặt, chúng ta chuyển qua bước 2; và nếu không thì chúng ta bỏ qua vùng đó và trượt cửa sổ đi chỗ khác. Nếu một vùng pass toàn bộ các bước test mặt đó thì cửa sổ đó có chứa mặt người. Các vùng không chứa mặt sẽ bị vứt vào hộp đỏ kia và không bao giờ được nhớ tới nữa, và các vùng có mặt sẽ được đưa vào hộp xanh để xử lý tiếp.
* Cách sử dụng:
  + - Bước 1: Cv2.data.haarcasades là đường dẫn đến file package cv2 data
    - Thu nhỏ kích thước của ảnh nếu cần thiết
    - Bước 2: Chuyển đổi ảnh thành ảnh đa mức xám, vì ảnh mà chúng ta nhìn thấy là RGB khi openCV đọc ảnh RGB thì nó lưu hình ảnh trong kênh BRG, với mục đích nhận dang hình ảnh chúng ta cần chuyển kênh BRG này sang kênh xám để dễ dàng xử lý và ít chuyên sâu hơn về mặt tính toán vì nó chỉ chứa 1 kênh đen và trắng bằng câu lệnh cv2.cvtColor(<tham số đọc ảnh>, cv2.COLOR\_BGR2GRAY)



* Bước 3: Sau khi chuyển sang ảnh đa mức xác, bây h sẽ xác định các đặc điểm chính trên khuân mặt sử dụng câu lệnh: detectMultiScale(self, image, scaleFactor=None, minNeighbors=None, flags=None, minSize=None, maxSize=None)
* Tham số truyền vào:
  + - image: ma trận kieru CV\_8U chứa ảnh đa mức xám
    - scaleFactor: tham số xác định kích thước ảnh được giảm theo tỉ lệ bao nhiêu
    - minNeighbors: tham số chỉ có bao nhiêu lân cận mỗi ứng viên hình chữ nhật có thể giữ lại nó (sẽ có rất nhiều khuân mặt do thay đổi kích thước cửa sổ trượt vì vậy cần có phương pháp tiếp cận vùng lân cận, kiểu như nó nằm trong vùng lân cận của cửa số trượt khác với nó thì ok, tham số này xác định có bao nhiêu vùng lân cận của các hình chữ nhật khác được yêu cầu chuyển thành 1 hcn)
* DetectMultiscale tìm ra khuân mặt và sẽ trả về 4 giá trị tọa độ x, y, chiều rộng, chiều cao, qua đó chúng ta sẽ vẽ một hình chữ nhật để thể hiện ra khuân mặt được phát hiện
* Bước 4: Vẽ HCN nhận diện khuân mặt:
  + - Syntax: cv2.rectangle(image, start\_point, end\_point, color, thickness)
    - Parameters:
    - image: It is the image on which rectangle is to be drawn.
    - start\_point: It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
    - end\_point: It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
    - color: It is the color of border line of rectangle to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue color.
    - thickness: It is the thickness of the rectangle border line in px. Thickness of -1 px will fill the rectangle shape by the specified color.
    - Return Value: It returns an image.

**Câu 22: Hàm dự đoán predict**

* Hàm dự đoán predict: thực hiện tính toán theo lô, phương pháp này được thiết kế để thực hiện trong các đầu vào quy mô lớn
* predict(  
      x, batch\_size=None, verbose=0, steps=None, callbacks=None, max\_queue\_size=10,  
      workers=1, use\_multiprocessing=False  
  )

**Câu 23: Mô tải lại luồng dữ liệu quá trình nhận diện ảnh**

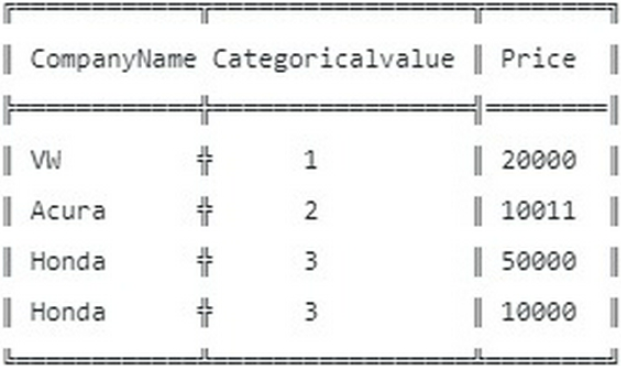
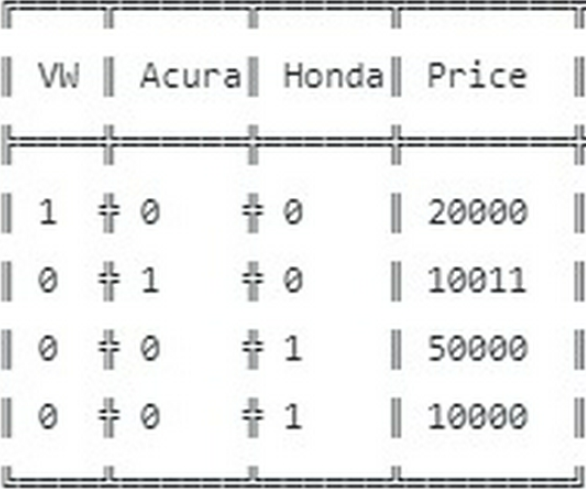
* Đầu tiên chúng ta cần 1 ảnh đầu vào:
  + - Thông tin ảnh đầu vào là một ma trận 2 chiều
* Chuyển ảnh đầu vào thành một ảnh đa mức xám
* Khai báo faces\_deatected để nhận diện khuân mặt
* Để có thể vẽ được một hình chữ nhật biểu thị vùng khuân mặt ta khai báo cv2.rectangle
* Sau khi detected được rồi thì sẽ for ra 4 chỉ số x, y, w, h là tọa độ chiều rộng và chiều cao và vẽ lên ảnh thật với các thông số đã lấy được
* roi\_gray = gray\_img[y:y + h, x:x + w] có lẽ đây là đoạn code khó hiểu nhất và ko biết nó để làm gì thì roi\_gay thực hiện cắt vừa in cái khuân mặt từ ảnh đa mức xám ra theo tọa độ x y h w đã được detech => như vậy chúng ta đã lấy ra được cái mặt của mình để đưa vào nhận diện cảm xúc
* Ơ nhưng mà khoan, khi dùng cái roi\_gray kia thì kích thước nó thay đổi vừa in theo mặt detech được => kích thước thay đổi liên tục và ko đúng kích cỡ ảnh model nhận diện được theo lúc bạn đầu training (ban đầu training theo ảnh 48x48) -> như vậy mình resize về 48x48 thôi ez 1 dòng code
* Tiếp theo chúng ta chuyển cái ảnh này về thành 1 mảng numpy
* Sử dụng hàm expand\_dims để mở rộng hình dạng của mảng bằng cách chèn một trục mới (Ví dụ mảng [2,4] thì khi shape nó chỉ nhận là 2 ko biết kích thước còn lại, khi expand\_dims thì sẽ là (1, 2)) axis=0 thì là chiều ngang còn 1 là dọc

**Câu 24: Mô tả lại luồng training model**

* B1: Định nghĩa các thông số: numlable, batch\_size, epochs, width, height của ảnh và import 1 số thư viện cần thiết
* B2: Khởi tạo model
  1. Tùy thuộc vào số lượng params truyền vào để tạo ra các lớp
  2. Thường lớp đầu tiên để trích xuất ra viền và cạnh
  3. Các lớp sau trích xuất ra các thông tin trừu tượng hơn
  4. Ở đây chúng ta dùng 4 lớp theo vgg16 là 5 lớp
* B3: Đọc file ảnh (có thể là ảnh thô chưa qua xử lý->cái này phải xử lý nhiều trước khi training hoặc ảnh đã qua xl dưới dạng ma trận thì training đc luôn)
* B4: Bóc tách các thông tin ban đầu ra các mảng dữ liệu dùng để training và mảng dữ liệu dùng để validate, thường thi ở dữ liệu training hay validate đều có 2 mảng con là dữ liệu và nhãn đã đc đánh dấu sẵn
* B5: Chuyển mảng đã thu được từ input sang float 32 và khi này thường chúng ta sẽ thu được một cái vector với các lable đầu vào thì khi đó phải chuyển nó sang một ma trận 2 chiều sử dụng to\_categorical để one hot encode(giải thích cái này bên dưới)
* B6: Chúng ta cần thêm 1 bước này để giảm nhiễu và chuẩn hóa lại ảnh bằng cách sử dụng np.mean và np.std và lấy mảng ban đầu trừ và chia lần lượt cho 2 thằng này là đc

**Câu 25: kỹ thuật one hot encode là gì**

* one hot encode là gì: one-hot là "Mã hóa các tính năng của một số nguyên sử dụng one-hot hay còn gọi là một chương trình one-hot",
* 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.



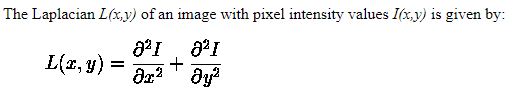
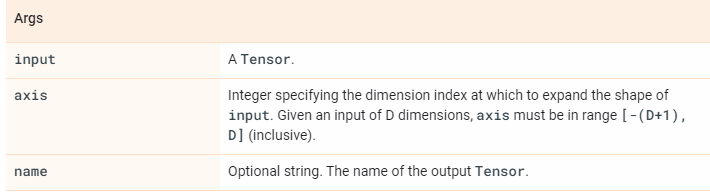
* Trước khi tiếp tục, liệu bạn có thắc mắc rằng tại sao không dùng trực tiếp nhãn phân loại lại phải sử dụng đến one-hot? Tôi cũng từng ngồi nghĩ mãi như vậy, tại sao??
* Hãy để tôi giải thích: Từ dữ liệu ban đầu, đặt VW = 1, Acura = 2, Honda = 3, từ số liệu ta nhận thấy3-1 = 2 chứng tỏ Acura=Honda-VW? wtf? Đây thực sự là công thức thảm họa. Chắc chắn sẽ rất nhiều lỗi.
* Đây là lý do tại sao chúng ta sử dung one-hot encoding để thực hiện chuyển đổi nhãn và góp phần không nhỏ vào đào tạo mô hình.

**Câu 26: std và mean có ý nghĩa như thế nào**

Gỉa sử ảnh H có 2 điểm ảnh là 3 và 7 => mean là 5

* 2 điểm ảnh so với trung bình là -2 và 2
* Tiến hành tăng sáng điểm ảnh lên 4 và 8 => mean là 6
* 2 điểm ảnh so với trung bình vẫn là -2 và 2

**Một số khái niệm liên quan**

* **Gradient** nghĩa là độ dốc. Trong xử lý ảnh, độ dốc (tức gradient) đang nói đến ở đây chính là độ dốc về mức sáng. Hay nói cách khác chính là sự thay đổi các giá trị pixel trong ảnh.
* **Laplacian** là thước đo [đẳng hướng](https://homepages.inf.ed.ac.uk/rbf/HIPR2/isotrop.htm) 2-D của [đạo hàm không gian](https://homepages.inf.ed.ac.uk/rbf/HIPR2/spatdom.htm) thứ 2 của một hình ảnh. **Laplacian** của một hình ảnh làm nổi bật các vùng thay đổi cường độ nhanh chóng và do đó thường được sử dụng để https://homepages.inf.ed.ac.uk/rbf/HIPR2/mote.gifphát hiện cạnh (xem bộ [phát hiện cạnh cắt bằng không](https://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm) ). **Laplacian** thường được áp dụng cho một hình ảnh lần đầu tiên được làm mịn bằng một cái gì đó xấp xỉ với [bộ lọc làm mịn **Gauss**](https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm) để giảm độ nhạy của nó với nhiễu, và do đó hai biến thể sẽ được mô tả cùng nhau ở đây. Toán tử thường lấy một hình ảnh graylevel duy nhất làm đầu vào và tạo ra một hình ảnh graylevel khác làm đầu ra.
* Dropout là một kỹ thuật khác, một cách tiếp cận khác để regularization trong mạng neural netwoks.
* Kỹ thuật dropout được thực hiện như sau:
* **Trong pha train**: với mỗi hidden layer, với mỗi trainning sample, với mỗi lần lặp, chọn ngẫu nhiên p phần trăm số node và bỏ qua nó (bỏ qua luôn hàm kích hoạt cho các node bị bỏ qua).
* Trong pha test: Sử dụng toàn bộ activations, nhưng giảm chúng với tỷ lệ p (do chúng ta bị miss p% hàm activation trong quá trình train).
* Dropout ép mạng neural phải tìm ra nhiều robust features hơn, với đặc điểm là chúng phải hữu ích hơn, tốt hơn, ngon hơn khi kết hợp với nhiều neuron khác.
* Dropout đòi hỏi phải gấp đôi quá trình huấn luyện để đạt được sự hội tụ. Tuy nhiên, thời gian huấn luyện cho mỗi epoch sẽ ít hơn.
* Với H unit trong mô hình, mỗi unit đều có xác xuất bị bỏ qua hoặc được chọn, chúng ta sẽ có 2^H mô hình có thể có. Trong pha test, toàn bộ network được sử dụng và mỗi hàm activation được giảm đi với hệ số p
* Theo nghiên cứu và thực nghiệm chỉ ra rằng Giá trị dropout tốt nhất là 0.2 và cần epochs ít nhất là trên 20
* **Iterrows** được sử dụng để duyệt qua các hàng của 1 data frame
* expand\_dims: Trả về một tensor có chiều dài 1 trục được chèn vào chỉ mục axis., input trong hàm ở đây là 1 tensor, axis là một số nguyên xác định chỉ số kích thước để mở rộng hình dạng input
* Tensor hay tiếng Việt gọi là Ten-xơ là đối tượng hình học miêu tả quan hệ tuyến tính giữa các đại lượng vectơ, vô hướng, và các tenxơ với nhau. Những ví dụ cơ bản về liên hệ này bao gồm tích vô hướng, tích vectơ, và ánh xạ tuyến tính. Đại lượng vectơ và vô hướng theo định nghĩa cũng là tenxơ
* cap.release() giúp giải phóng tài nguyên phần cứng
* cv2.destroyAllWindows
* Sequential nhóm một chồng lớp tuyến tính thành một tf.keras.Model cung cấp các tính năng đào tạo và suy luận.