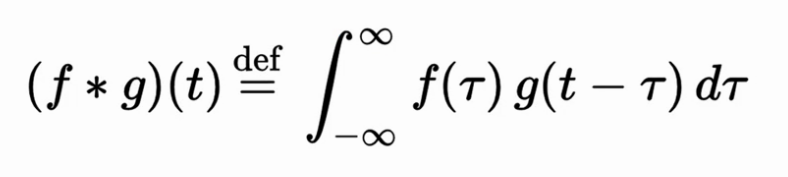
The convolutional neural networks have 4 stages, convolution, max pooling, flattening and full connection.

They work by representing images as arrays of pixels and putting these through a CNN, to match to output classes.

**Convolution:**



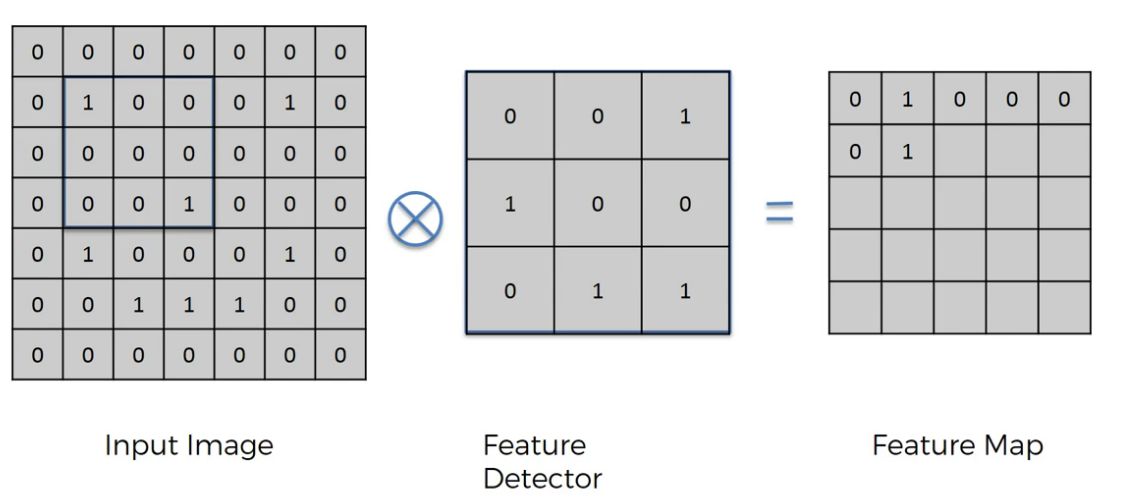
**Input image**, matrix representing the image

**Feature detector:**

3x3 matrix, (can be other dimensions).

\*also known kernel, or filter

The way to operate the feature detector is to place it at the top left corner of the image and run an “AND” (multiplication) function on the numbers the feature detector lines up with on the input image:



In the end we SUM all the line ups.

So in the above case we multiply the top left corner box of both the feature detector and the selected box in the input image.

1x0 = 0. Then we do the next one and so on and sum al of them. The end result is only one of them actually match up and the result of the AND is 1+(0+0+0+0+0+0+0+0).

The feature map is the result.

We start the feature map at the top left corner, and move slide it one ‘box’ to the right until the end, Then move to the next row down. The amount you move Is called the ‘stride’. Common strides = 2.

Feature maps can also be called activation maps or ‘convolved’ maps; ie; subject to a convolution operation.

One of the advantages is that it works like a compression. Algorithm that makes the image smaller. We do lose some value.

The way the feature map works as a filter that looks at each part of the image, and matches it against the feature detector and gives a score for whether the feature is present.

Several of thses feature maps are used to identify different features, each generating a score for whether they exist in the input image.

**Relu Layer**

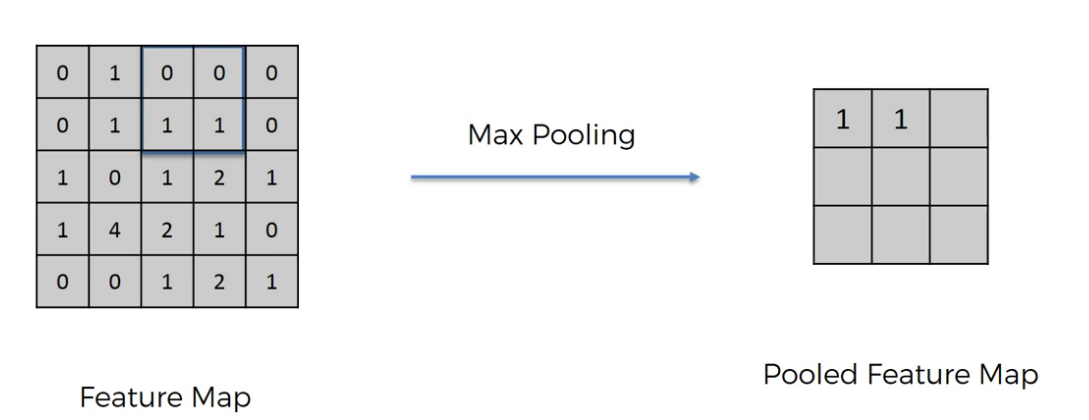
A rectifier activation function is placed here in order to promote nonlinearity of the system. The rectifier functions off then on behaviour turns parts of the image into almost binary states 0-1 with no fading in or out, making it easier to recognise images.

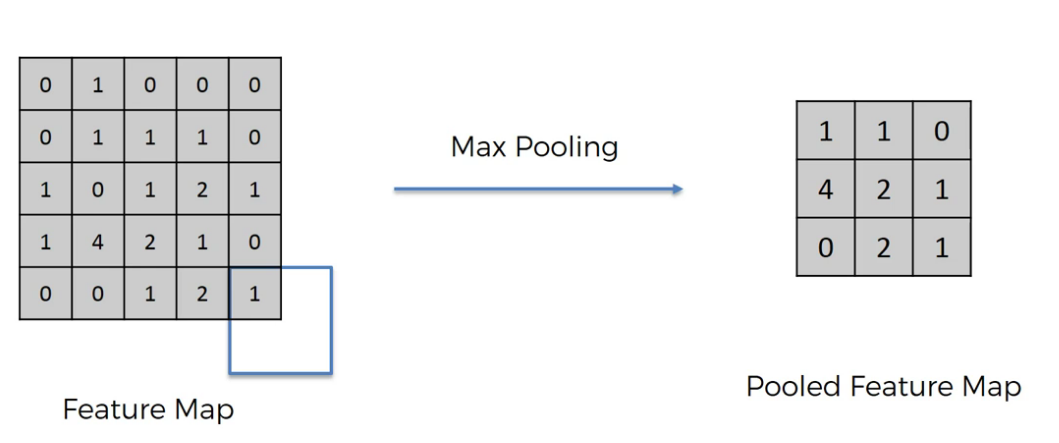
**Max Pooling**

To do with dealing with rotated images, or features which may be in-exact.

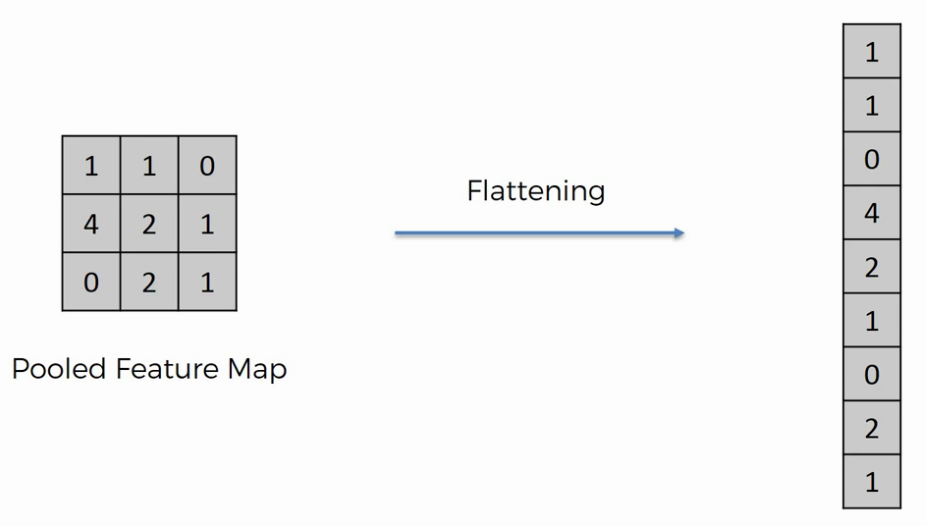
It starts with the feature map, (which is the output of step 1, including the relu layer). That is; the input and feature detector convolution. On this feature map we select a 2x2 ‘cursor’ of 4 boxes, and slide in by <stride> units left to right then down. (remember the strides ‘stride’ downwards aswell as across.) For every cursor resting spot we take the highest value number and put it into a ‘pooled’ feature map.

By doing this we eliminate 75% of the data. What actually happens is by taking the highest number in every 4 square, the ‘detection’ is the same wherever in those 4 squares the feature appears. This means that features can be slightly distorted, rotated etc and still be detected.





**Flattening:**



**Full connection:**

The flattened layer is inserted as inputs into a neural network. The neural network trains itself to recognise these features against a taught output of what it’s looking at. The back propogatation not only mutates the weights, it also mutates the feature detector so that what it’s looking for in each image slowly changes to what differentiates the two images.

An interesting question is asked: Why don’t you just take an image. Pixel by pixel and feed it into a neaural network to crunch? Wouldn’t that work?

There are two answers: 1) it would be very computer intentsive, but more importantly. The pixels would represent pixels by themselves with no information on how they relate to other pixels around them. With convolution networks though (the convolution steps) pixels, and their immediate surroundings are taken together, so each input node actually doesn’t just represent a pixel it almost represents a ‘cluster’ of pixels around a feature.

**ImageDataGenerator**

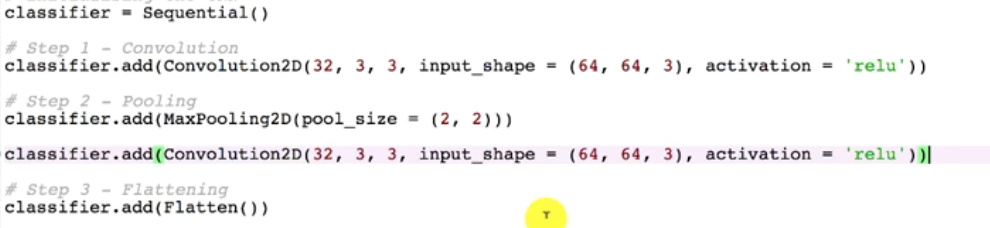
This is a very interesting preprocessing step for images. It’s to try to prevent overfitting by creating many more images than is actually present.

It does this by slightly shifting, and transforming the images in the training set to create many variations (simulating that there are many more images than there actually are).

This is called ‘image augmentation’

**Improving performance:**

Performance can be achieved by adding more convolutional layers or more fully connected layer:



Copy and paste the convolutional layers under MAX pooling layer.

We also need to add a new max pool layer.

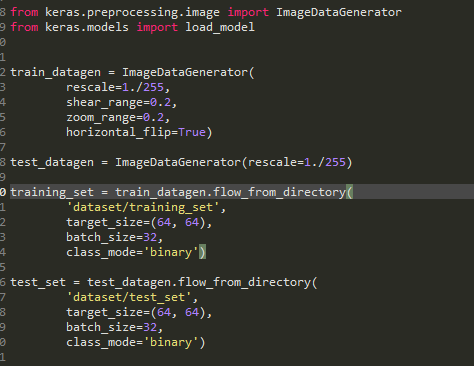
NOTE: the input shape parameter MUST be left blank. (because the input shape layer defines it as the ‘first’ layer. Leaving it empty allows it to take the last layers outputs as inputs.

**Setting up training sets.**

To set up training sets we use ImageDataGenerator class. This class generates Image data for use in the neural network including scaling operations.

The class when instantiated takes in a rescale factor, a scale range, a zoom range and horizontal flip.

After the object is generated it uses it’s ‘flow\_from\_directory’ method to define image flows. It can have a test set flow and a training set flow:

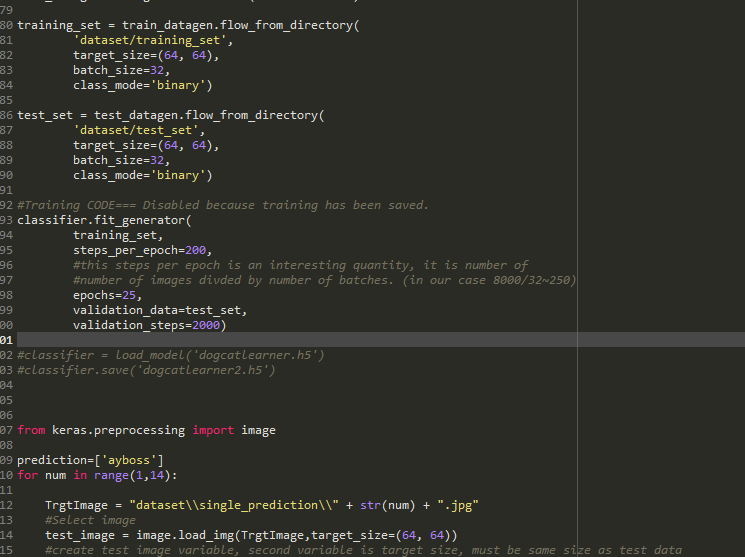


Use fit-generator as opposed to regular fit. This is a batch by batch fitting operation. It allows train\_generator to be executed, between each batch, some images are morphed and inserted in parralell.

**To test single prediction:**

Use image from keras.processing to import image

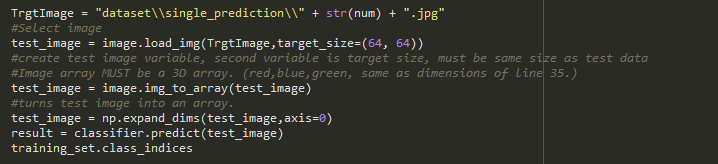
Image.load\_img function takes in image path (from the working folder) and target\_size=(64, 64). Remember target size has to be same as that defined in the Imagegenerator object:



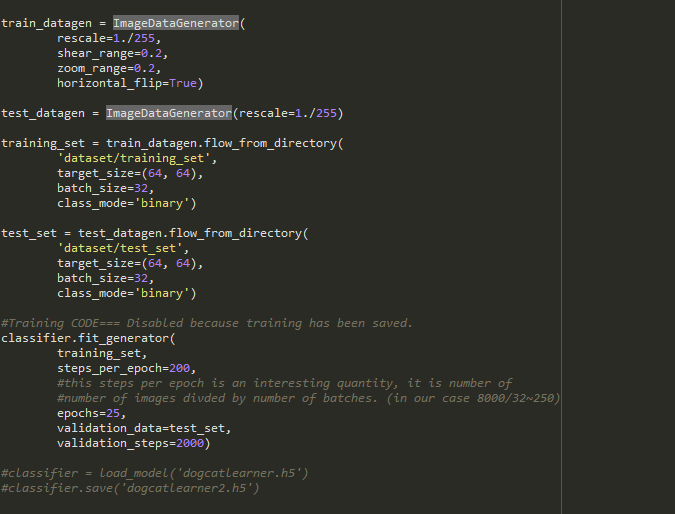
Use image.img\_to\_array to put test image into an array. This is required because

Classifier.predict expects an array. Not a single image.

It actually expects a 4D array! So use np.expand\_dims(test\_image,axis=0) to expand 3d array to 4d.



With regard to the 4 blocks of code for set generation:



The first (train\_datagen), generates an object that rescales training images.

The second (test\_datagen) generates an object that rescales test images.

The third (training\_set) declares the training set and defines it’s batches.

The fourth (test\_set) defines the test set, and defines it’s batches.

The fifth, fits network to the training data.