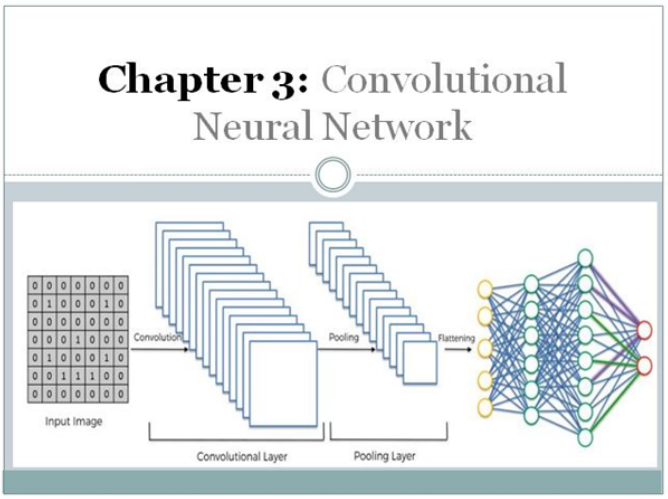
Convolutional Neural Network(CNN) with Practical Implementation

Amir Ali Follow
May 22, 2019 · 24 min read



In this Third Chapter of Deep Learning book, we will discuss the Convolutional Neural Network. It is a Supervised Deep Learning technique and we will discuss both theoretical and Practical Implementation from Scratch.

This chapter spans 4 parts:

1. What is Convolutional Neural Network?
2. Structure of Convolutional Neural Network.
3. How Convolutional Neural Network works?
4. Practical Implementation of Convolutional Neural Network.

1. What are Convolutional Neural Networks?

1.1: Introduction

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations,

Pinterest for their home feed personalization, and Instagram for their search infrastructure.

1.2: The Problem Space

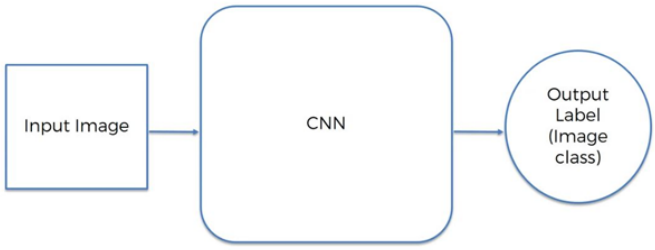
Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we can immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.

What We See

What Computers See

1.3: Input and Output

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for a cat, .15 for a dog, .05 for a bird, etc).



1.4: What we want to computer Do.

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process

that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. Similarly, the computer can perform image classification by looking for low-level features such as edges and curves and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what CNN does. Let's get into the specifics.

1.5: Biological Connection.

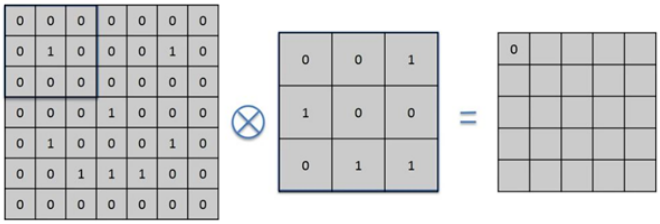
But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well and is the basis behind CNNs.

2. Structure of Convolutional Neural Network.

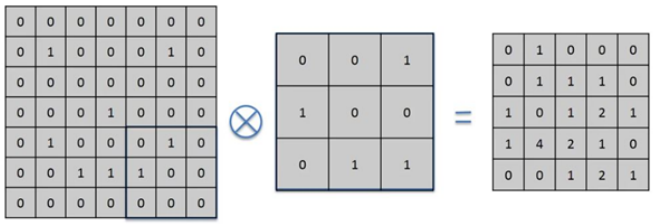
A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers does. So let's get into the most important one.

2.1: Convolutional

ConvNets derive their name from the "Convolution Operation". The Convolution in case of ConvNet is to extract features from the input images. Convolution preserves the spatial relationships between pixels by learning image features using Small Square of input data. As we discussed above, every image can be considered as a matrix of the pixel value. Consider a 5*5 image whose pixel values are only 0 & 1 (note that for a grayscale image, pixel values range from 0 to 255, where pixel values are only 0 & 1.



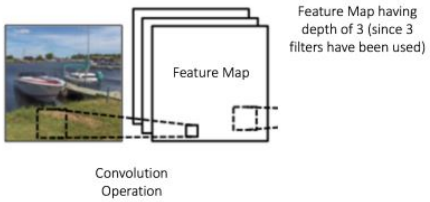
Similarly, Feature Detector Detect the Every single part of the input image and then result from the show in Feature map which base on the match of feature Detector of the input image.



CNN terminology, the 3×3 matrix is called a 'filter' or 'kernel' or 'feature detector' and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the 'Feature Map'. It is important to note that filters act as feature detectors from the original input image.

Depth:

Depth corresponds to the number of filters we use for the convolution operation. In the network shown in Figure below, we are performing the convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. You can think of these three feature maps as stacked 2d matrices, so, the 'depth' of the feature map would be three. The more numbers of filters the more accurate result.

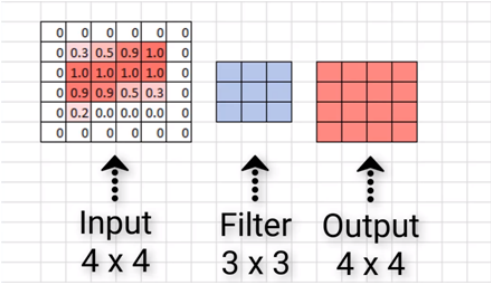


Stride:

Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.

Zero-Padding:

Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero paddings is that it allows us to control the size of the feature maps. Adding zero-padding is also called *wide convolution*, and not using zero-padding would be a *narrow convolution*.



Input = output

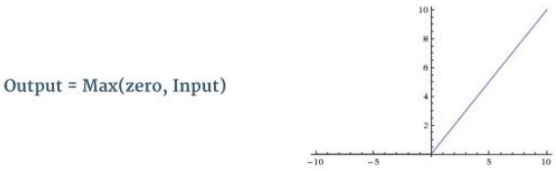
That's the benefits of padding

Padding >>> valid >>> no padding

Padding >>> same >>> padding to make output size same as input size.

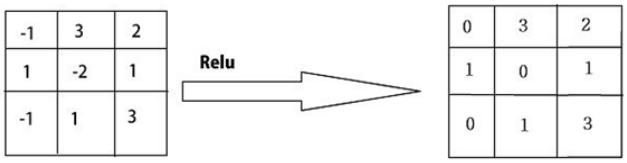
2.1a: ReLu Layer

An additional operation called ReLU has been used after every Convolution operation. ReLU stands for the Rectified Linear Unit and is a non-linear operation. Its output is given by:



ReLU is an element-wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation — element-wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

As shown below image we apply the Relu operation and he replaces all the negative numbers by 0.

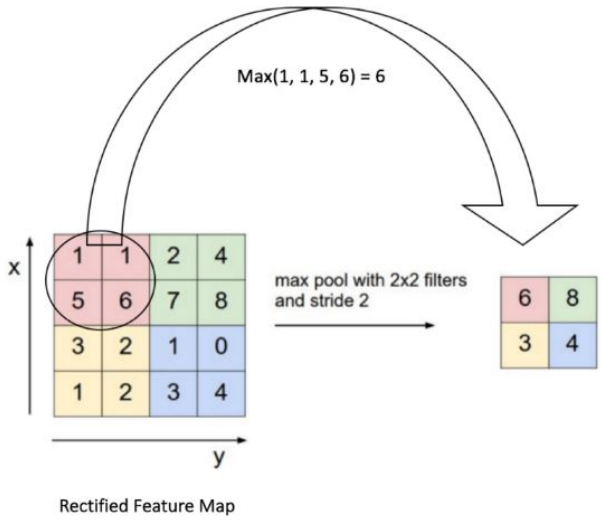


2.2: Max Pooling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum, etc.

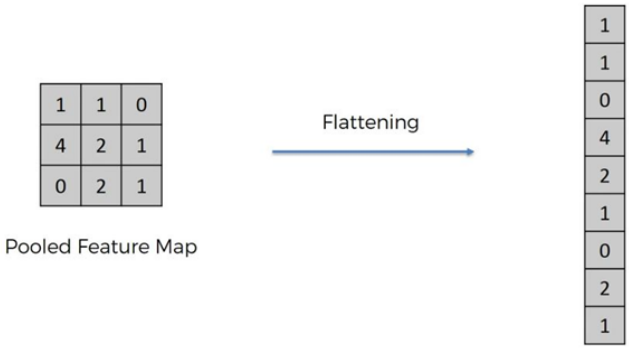
In the case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

Below shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a 2×2 window.



2.3: Flattening

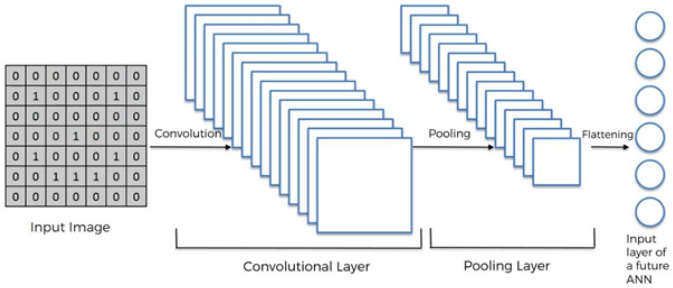
Flattening is the process of converting al the resultant 2-dimensional arrays into a single long continuous linear vector.



The process of building a CNN involves four major steps

1. Convolution
2. Pooling

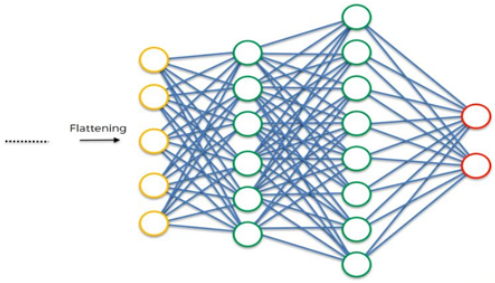
3. Flattening
4. Full Connection



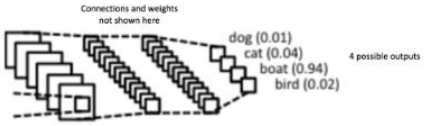
So Flattening is become the input of Artificial Neural Network which is used for the backpropagation Method.

2.4: Full Connection

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.



The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. For example, the image classification task we set out to perform has four possible outputs as shown in **Figure** below (note that Figure 14 does not show connections between the nodes in the fully connected layer)

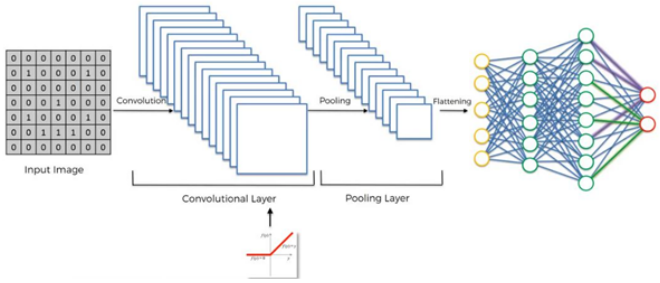


Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer

of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one.

2.5: Putting it all together Training using Backpropagation

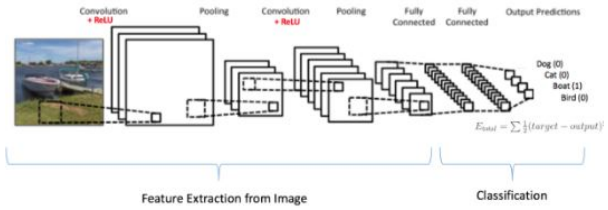
As discussed above the convolution + Pooling layers act as Features extractor from the input image while a fully connected layer acts as the classifier.



Note that in **Figure** below, since the input image is a boat, the target probability is 1 for Boat class and 0 for the other three classes, i.e.

Input Image = Boat

Target Vector = [0, 0, 1, 0]



The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters/weights with random values.

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU, and pooling operations along with forwarding propagation in the Fully Connected layer) and finds the output probabilities for each class.

Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]

Since weights are randomly assigned for the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

Total Error = ∑ ½ (target probability — output probability) ²

Step4: Use Backpropagation to calculate the *gradients* of the error concerning all weights in the network and use *gradient descent* to update all filter values/weights and parameter values to minimize the output error.

The weights are adjusted in proportion to their contribution to the total error.

When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].

This means that the network has *learn*ed to classify this particular image correctly by adjusting its weights/filters such that the output error is reduced.

Parameters like the number of filters, filter sizes, the architecture of the network, etc. have all been fixed before Step 1 and do not change during the training process — only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2–4 with all images in the training set.

3: How Convolutional Neural Network Works?

Approach

- Build a small convolutional neural network as defined in the architecture below.
- Select images to train the convolutional neural network.
- Extraction of feature filters/feature maps.
- Implementation of the convolutional layer.
- Apply the ReLu Activation function on the convolutional layer to convert all negative values to zero.
- Then apply max pooling on convolutional layers.
- Make a fully connected layer
- Then input an image into CNN to predict the image content
- Backpropagation to calculate the error rate

Selection of Image

Let's take an image of the alphabet "X" and "O".

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image X

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image O

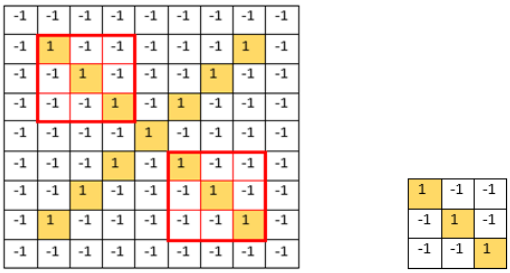
Train the Convolutional Neural Network For Image X

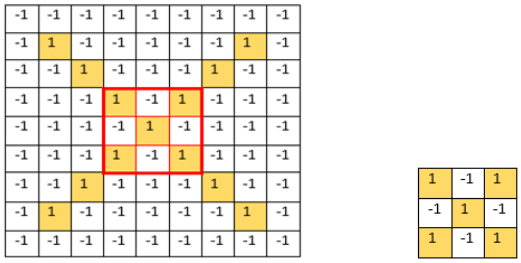
Feature Filters extraction from image X

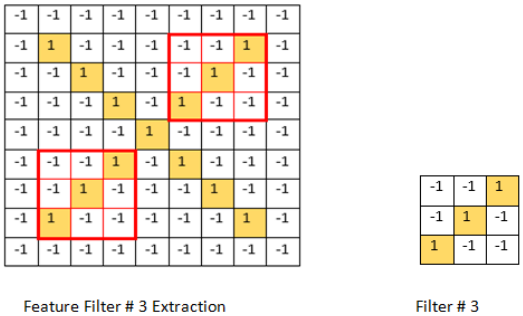
In convolutional networks, you look at an image through a smaller window and move that window to the right and down. That way you can find features in that window, for example, a horizontal line or a vertical line or a curve, etc… What exactly a convolutional neural network considers an important feature is defined while learning.

Wherever you find those features, you report that in the feature maps. A certain combination of features in a certain area can signal a larger, more complex feature exists there.

For example, your first feature map could look for curves. The next feature map could look at a combination of curves that build circles.



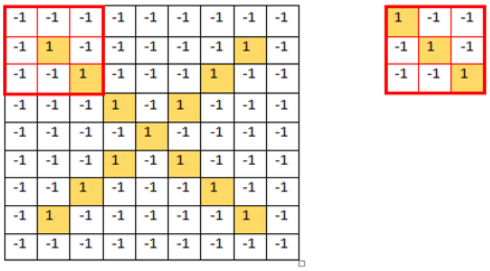




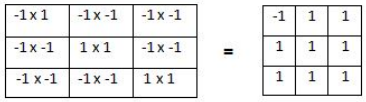
3.1 Convolutional Layers

3.1.1 Convolutional Layer 1 (Image X with filter 1)

In CNN convolutional layer, the 3×3 matrix called the '**feature filter'** or 'kernel' or 'feature detector' sliding over the image and the matrix formed will be the **convolutional layer**. It is important to note that filters act as feature detectors from the original input image. Image X matching with filter # 1 with a **stride** of 1.

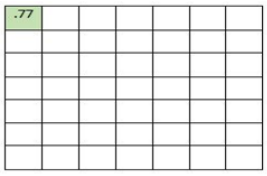


The pixel values of the highlighted matrix will be multiplied with their corresponding pixel values of the filter and then takes the average.



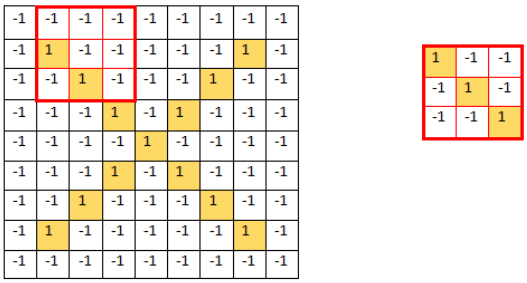
$$= \frac{-1+1+1+1+1+1+1+1+1}{9}$$

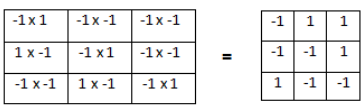
$$= 0.77$$



Convolutional layer through filter 1

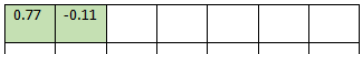
Here you will see how the filter shifts on pixels with a stride of 1.





$$= \frac{-1+1+1-1-1+1+1-1-1}{9}$$

$$= -0.11$$



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	-1	-1
-1	1	-1
-1	-1	1

-1 x 1	-1 x -1	-1 x -1	=	-1	1	1
-1 x -1	-1 x 1	-1 x -1		1	-1	1
1 x -1	-1 x -1	-1 x 1		-1	1	-1

$$= \frac{-1+1+1+1-1+1-1+1-1}{9}$$

$$= 0.11$$

0.77	-0.11	0.11				

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	-1	-1
-1	1	-1
-1	-1	1

-1 x 1	-1 x -1	-1 x -1	=	-1	1	1
-1 x -1	-1 x 1	-1 x -1		1	-1	1
-1 x -1	-1 x -1	-1 x 1		1	1	-1

$$= \frac{-1+1+1+1-1+1+1+1-1}{9}$$

$$= 0.33$$

0.77	-0.11	0.11	0.33			

similarly so on

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	-1	-1
-1	1	-1
-1	-1	1

1 x 1	-1 x -1	-1 x -1
-1 x -1	1 x 1	-1 x -1
-1 x -1	-1 x -1	-1 x 1

=

1	1	1
1	1	1
1	1	-1

$$= \frac{1+1+1+1+1+1+1+1-1}{9}$$

= 0.77

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1	-1	1
-1	1	-1
1	-1	1

3.1.2 Convolutional Layer 2 (Image X with filter 2)

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

After repeating the same steps (As we did for filter 1) of the convolutional layer on image "X" with filter 2, we get

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

-1	-1	1
-1	1	-1
1	-1	-1

3.1.3 Convolutional Layer 3 (Image X with filter 3)

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

After repeating the same steps of the convolutional layer on image "X" with filter 3, we get

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

3.2 RELU Layer:

Apply ReLu Activation Function on Convolutional layers:

Convert all negative values to zero

3.2.1 Relu layer For Convolutional Layer 1:

Apply ReLu activation Function on Convolutional Layer 1 to convert all the negative values to zero

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11

0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

3.2.2 Relu layer Convolutional Layer 2:

Apply ReLu activation Function on Convolutional Layer 2 to convert all the negative values to zero

0.33	0	0.11	0	0.11	0	0.33
0	0.55	0	0.33	0	0.55	0
0.11	0	0.55	0	0.55	0	0.11
0	0.33	0	1.00	0	0.33	0
0.11	0	0.55	0	0.55	0	0.11
0	0.55	0	0.33	0	0.55	0
0.33	0	0.11	0	0.11	0	0.33

3.2.2 Relu layer Convolutional Layer 3:

Apply ReLu activation Function on Convolutional Layer 3 to convert all the negative values to zero

0.33	0	0.55	0.33	0.11	0	0.77
0	0.11	0	0.33	0	1.00	0
0.55	0	0.11	0	1.00	0	0.11
0.33	0.33	0	0.55	0	0.33	0.33
0.11	0	1.00	0	0.11	0	0.55
0	1.00	0	0.33	0	0.11	0
0.77	0	0.11	0.33	0.55	0	0.33

3.3 Max Pooling:

After applying the Convolutional & Relu layer respectively Now we apply the Max pooling for convolutional layers 1, 2 & 3 and extract maximum feature from the image.

3.3.1 Max pooling For Convolutional Layer 1

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

1.00			

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33

1.00	0.33		

0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

1.00	0.33	0.55	

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

1.00	0.33	0.55	0.33

similarly so on

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

3.3.2 Max pooling For Convolutional Layer 2

0.33	0	0.11	0	0.11	0	0.33
0	0.55	0	0.33	0	0.55	0
0.11	0	0.55	0	0.55	0	0.11
0	0.33	0	1.00	0	0.33	0
0.11	0	0.55	0	0.55	0	0.11
0	0.55	0	0.33	0	0.55	0
0.33	0	0.11	0	0.11	0	0.33

Repeat the same steps of max pooling for convolutional layer 2, we get

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

3.3.3 Max pooling For Convolutional Layer 3

0.33	0	0.55	0.33	0.11	0	0.77
0	0.11	0	0.33	0	1.00	0
0.55	0	0.11	0	1.00	0	0.11
0.33	0.33	0	0.55	0	0.33	0.33
0.11	0	1.00	0	0.11	0	0.55
0	1.00	0	0.33	0	0.11	0
0.77	0	0.11	0.33	0.55	0	0.33

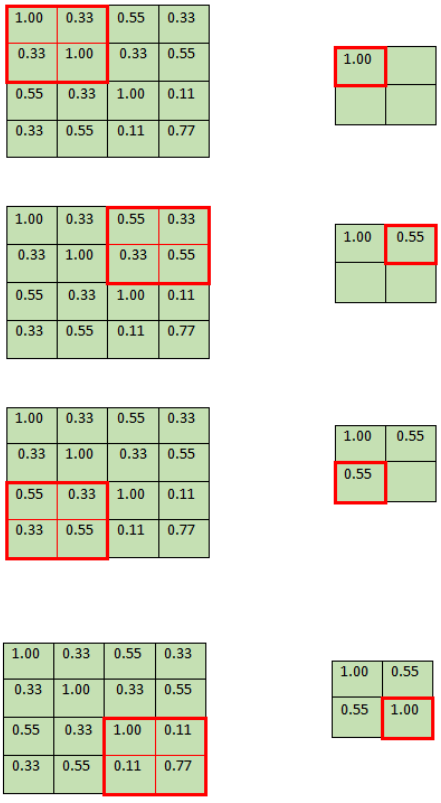
Repeat the same steps of max pooling for convolutional layer 3, we get

0.33	0.55	1.00	0.77
0.55	1.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

3.4 Further Max Pooling:

Further Max Pooling for use in fully connected layer

3.4.1 Further Max pooling for convolutional layer 1



3.4.2 Further Max pooling for convolutional layer 2

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11

0.33	0.11	0.11	0.33

Repeat the same steps of further max pooling for use in fully connected layer

1.00	0.55
0.55	0.55

3.4.3 Further Max pooling for convolutional layer 3

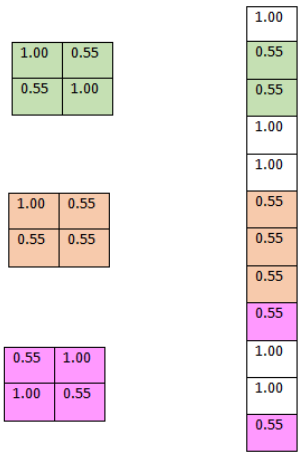
0.33	0.55	1.00	0.77
0.55	1.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Repeat the same steps of further max pooling for use in fully connected layer

0.55	1.00
1.00	0.55

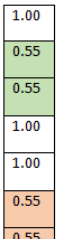
3.5 Flattening:

In this step, we converting all the resultant 2-dimensional arrays into a single long continuous linear vector.



Now, these single long continuous linear vectors are input nodes of our full connection layer.

3.6 Fully Connected Layer (for X):



0.55
0.55
1.00
1.00
0.55

White cells are votes depends on how strongly a value can predict “X”

These are the input node so after that we doing backpropagation. For backpropagation, you can see **chapter 2** which we explain backpropagation in details (we not going backpropagation here because we already have done in chapter 2)

Train the Convolutional Neural Network For Image ‘O’

Feature Filters extraction from image ‘O’

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

After repeating the procedure of training the convolutional neural network the same as we had done for training image X, we get

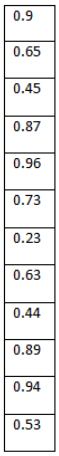
0.55
1.00
1.00
0.55
0.55
0.55
0.55
0.55
1.00
0.55
0.55
1.00

White cells are votes depends on how strongly a value can predict “O”

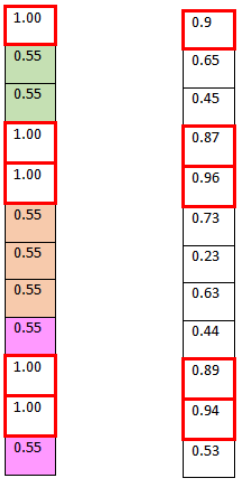
Prediction Example

We feed an image into the convolutional neural network and we don’t know whether it is X or O.

Now we get new input and we don't know what it is and want to decide the way this works is the input goes through all of our convolutional our rectified linear unit and pooling layer and at the end we get,



Now calculate predictions for X

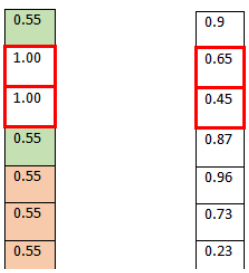


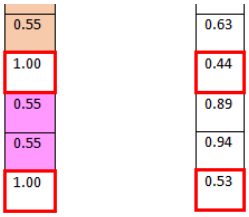
$$= \frac{0.9+0.87+0.96+0.89+0.94}{5}$$

$$= 0.912$$

So our CNN predicts the input image as X with a prediction rate of 91 percent

Now calculate a prediction for O Image





$$= \frac{0.65+0.45+0.44+0.53}{4}$$

$$= 0.5175$$

So our CNN predicts the input image as O with a prediction rate of 51 percent

As

Prediction rate for X > Prediction rate of O

91 > 51

So,

The image we input for prediction into our convolutional neural network is X

Note: If you want this article check out my **academia.edu** profile.

5: Practical Implementation of Convolutional Neural Network.

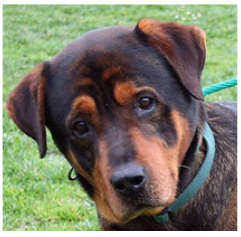
Image Recognition (Cat & Dog dataset)

In this part, we will create a Convolutional Neural Network that can detect various objects in images. We will implement this Deep Learning model to recognize a cat or a dog in a set of pictures. However, this model can be reused to detect anything else and we will show you how to do it — by simply changing the pictures in the input folder.

For example, you will be able to train the same model on a set of brain images, to detect if they contain a tumor or not. But if you want to keep it fitted to cats and dogs, then you will be able to take a picture of your cat or your dog, and your model will predict which pet you have.

Dataset sample:

The dataset contains 10000 images of cats and dogs.



Cat Dog

Part 1: Building the CNN model

1.1 Import the Libraries:

In this step, we import Keras library and packages for building the Convolutional Neural Network model. First, we import **the Sequential** module which is used for initializing our model. Second is **Convolutiona2D** is the package that we'll use for the first step of making the CNN that is the convolution step in which we add convolutional layers So here we are working with images which are only 2 dimensions there we use convolution2D(incase of video we use 3D). Then the next package is **MaxPooling2D** that's a package that we'll use to proceed to step to the pooling step that will add our pooling layers. The next package is **Flatten** this is the package that we will use in step 3 flattening in which we convert all the pooled feature maps that we created through convolution and max-pooling into the large feature vector that is then becoming the input of our fully connected layers. And then finally the last package which **Dense** package this is the package we use to add the fully connected layer.

```
# importing the keras Libraries and package
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

1.2 Initialize our model:

In this step, we initialize our Convolutional Neural Network model to do that we use sequential modules.

```
# initializing the Convolutional Neural Network
classifier = Sequential()
```

1.3 Convolution Step:

In this step, we add a convolution step and this is the first step of CNN. Here in Convolution2D, we pass several arguments in this module. First arguments which we pass here **feature detector** (32, 3, 3) that's mean we create 32 feature detectors of three by three dimensions and therefore our convolutional layers composed of 32 feature maps. Next arguments are **input shape** that's the shape of input image on which we are going to apply feature detectors through the convolution operation in this arguments we convert all images into same formats here 64 and 64 are dimension and 3 is number of channels because we are working with colored images that's why we use 3 number channels correspond to RGB (incase of black & white image we use 1 number of channel). And the last argument which we pass here is **relu** activation function the relu function is used to replace the negate pixel value by 0(As we understand in mathematical part of CNN intuition)

```
# Step 1: Convolution
classifier.add(Convolution2D(32, 3, 3, input_shape = (64, 64, 3), activation = 'relu'))
```

1.4 Pooling Step:

In this step, we add the pooling layer of our CNN model pooling layer is reducing the size of the feature map(the size of the feature map is divided by 2 when we apply the pooling layer). Here we passed only one argument which is **pool size** and define 2 by 2 dimension because we don't want to lose any information about the image there we take the minimum size of the pool.

```
# step 2: Max Pooling
classifier.add(MaxPooling2D(pool_size = (2,2)))
```

1.5 Add a second convolution layer and pooling layer:

In this step, we add a second convolution layer and pooling layer to make a mode more efficient and produce some good accuracy.

```
# adding a second convolution layer
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2,2)))
```

1.6 Flatten Step:

In this step, we convert Max pooling 2D into a single long continuous linear vector to make an input node of fully connected layers.

```
# step 3: Flattening
classifier.add(Flatten())
```

1.7 Full Connection layer:

In this step, we use the Dense model to add a different layer. The parameter which we pass here first is ouptu_dim=6 which defines hidden layer=128 and last arguments Third is activation= relu here in hidden layer we use relu activation.

```
# Step 4: Full Connection
classifier.add(Dense(output_dim = 128, activation = 'relu'))    # hidden Layer
```

After adding a Hidden layer we add an output layer in full connection layer out_dim= 1 which means one output node here we use the sigmoid function because our target attribute has a binary class which is cat or dog that's why we use sigmoid activation function.

```
classifier.add(Dense(output_dim = 1, activation = 'sigmoid'))   # output Layer
```

1.8 Compiling the CNN

In this step we compile the CNN to do that we use the compile method and add several parameters the first parameter is **optimizer = Adam** here use the optimal number of weights. So for choosing the optimal number of weights, there are various algorithms of Stochastic Gradient Descent but very efficient one which is Adam so that's why we use Adam optimizer here. The second parameter is loss this corresponds to loss function here we use **binary_crossentropy** because if we see target attribute our dataset which contains the binary value that's why we choose the binary cross-entropy. The final parameter is **metrics** its list of metrics to be evaluated by the model and here we choose the accuracy metrics.

```
# compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Part 2: Image Preprocessing

2.1 Import the library

First import a class that will allow us to use this image data Generator function. This class called Image Data Generator and we import this class from Keras image preprocessing.

```
from keras.preprocessing.image import ImageDataGenerator
```

2.2: Image Augmentation

2.2.1: Train_datagen

This step is like a data preprocessing which we did chapter 2 ANN but Now we are working with an image that's why we doing Image Augmentation. Here we pass several arguments first is **rescale** is like a feature scaling which doing in data preprocessing here our image 0 to 255-pixel value and we rescale into 0 & 1. Then next is **Shear_range**(Shear angle in the counter-clockwise direction in degrees) that to apply random transactions and we will keep this 0.2. **zoom_range** that will apply some random zoom and we keep 0.2 value. And then last is **horizontal_flip** Randomly flip inputs horizontally

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```

2.2.2: Test_datagen

Same as a train but here only rescale our images.

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

2.3 Training set & Test set

In this step, we import the images that we want to train. Remember our dataset contains 1000 images 8000 for train and 2000 for test and here first we import our images from the working directory that we want to train. Then target size here 64 by 64 that's expected CNN which we defined earlier in Building the CNN part. Then batch size that the size of batch in which some random sample of our image will be included and that contains the number of images that will go through the CNN after which the weight will be updated. And finally, the class mode that the parameter indicating if your class is dependent variable is binary or has more than two categories and therefore since two-class here cat & dog that's why class mode is binary here that we use here.

```
train_set = train_datagen.flow_from_directory('dataset/training_set',
                                              target_size=(64, 64),
                                              batch_size=32,
                                              class_mode='binary')
```

```
Found 8000 images belonging to 2 classes.
```

```
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size=(64, 64),
                                            batch_size=32,
                                            class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
```

2.4 Fit the CNN model

First arguments which we pass here is **train set** which contains 8000 images belongs to 2 class second is **stepper epoch** that is the number of the image we in our training set because remember all the observation of the training set pass through CNN during each epoch since we have 8000 images that's why 8000 here. Then the number of **epoch** you want to choose to train our CNN here we take only 1 because our purpose is just to understand we suggest to take more than 50 for good accuracy. And then **validation data** that correspond to the test set on which we want to evaluate the performance of our CNN. And last are validation steps that correspond to the number of the image in our test set and that is 2000.

```
classifier.fit_generator(train_set,
                         steps_per_epoch=8000,
                         epochs=1,
                         validation_data=test_set,
                         validation_steps=2000)
```

```
Epoch 1/1
8000/8000 [==============================] - 4038s 505ms/step - loss: 0.4053 - acc: 0.8062 - val_loss:
894 - val_acc: 0.8164
```

Here we go we obtain 80% for the training set and 81% for the test set.

Part 3: Making a new prediction

3.1 Import the library

In this step, we import libraries the first library which we import here is numpy used for multidimensional array and the second is image package from Keras used for import the image from the working directory.

```
# import libraries
import numpy as np
from keras.preprocessing import image
```

3.2 Import the Image

In this step, we import our single image we don't know either is cat image or dog image (like in mathematical part when we train images 0 and X and then make a prediction X). Here we pass two arguments first is image load and second is target size. Here we set target size 64 by 64 as expected our CNN model.

```
# load the single image from dataset
test_image = image.load_img('dataset/single_prediction/cat_or_dog_2.jpg', target_size = (64, 64))
```

```
test_image  Image  (64, 64)
```

3.3 Set the Dimension of image

In this step, we set the dimension of our image 2D (64, 64) into the 3D array(64, 64, 3) exactly as input shape that's the architecture of input shape.

```
# set the dimension
test_image = image.img_to_array(test_image)
```

```
test_image  float32  (64, 64, 3)
```

3.4 Set the Dimension of image

We are working with the single image we directly make a prediction then will raise an error which expected convolution 2d input_1 to have 4 dimensions to do that we modify the test image into 4 dimensions.

```
# set the dimension
test_image = np.expand_dims(test_image, axis = 0)
```

```
test_image  float32  (1, 64, 64, 3)
```

3.5 Make a prediction

In this step, we predict a single image

So here we go cat corresponds to 0 and dog corresponds to 1 so our image is a dog. Perfect so that's mean the prediction of our CNN model for the first image of cat & dog is correct because the image contains the dog

```
# predict the correct image
result = classifier.predict(test_image)
```

```
# on the basis of training predict img
train_set.class_indices
```

```
{'cats': 0, 'dogs': 1}
```

So here we go cat corresponds to 0 and dog corresponds to 1 so our image is a dog. Perfect so that's mean the prediction of our CNN model for the first image of cat & dog is correct because the image contains the dog

If you want dataset and code you also check my **Github** Profile.

End Notes

If you liked this article, be sure to click ❤ below to recommend it and if you have any questions, **leave a comment** and I will do my best to answer.

For being more aware of the world of machine learning, **follow me**. It's the best way to find out when I write more articles like this.

You can also follow me on **Github** for code & dataset follow on **Aacademia.edu** for this article, **Twitter** and **Email** me directly or find me on **LinkedIn**. I'd love to hear from you.

That's all folks, Have a nice day :)

Convolutional Neural Net Neural Networks Computer Vision Image Recognition Image Processing