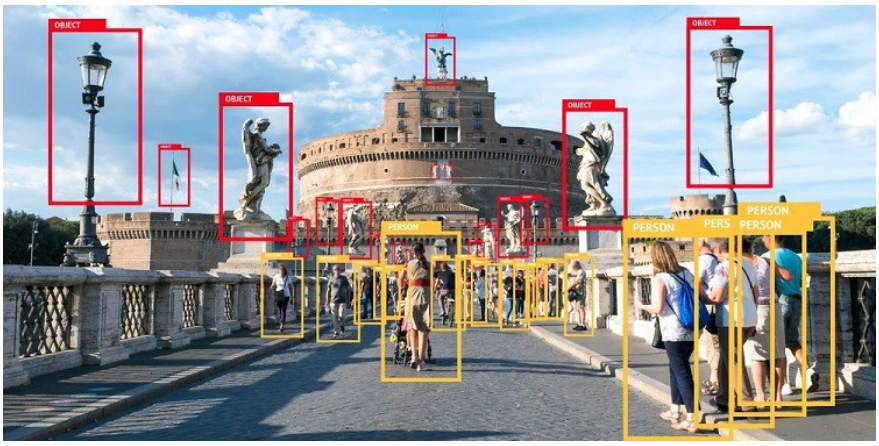
Create your first Image Recognition Classifier using CNN, Keras and Tensorflow backend

Yash Agarwal Follow
Jul 8, 2018 · 7 min read

With the dawn of a new era of A.I., machine learning, and robotics, its time for the machines to perform tasks characteristic of human intelligence. Machines use their own senses to do things like planning, pattern recognizing, understanding natural language, learning and solving problems. And Image Recognition is one of its senses!!!

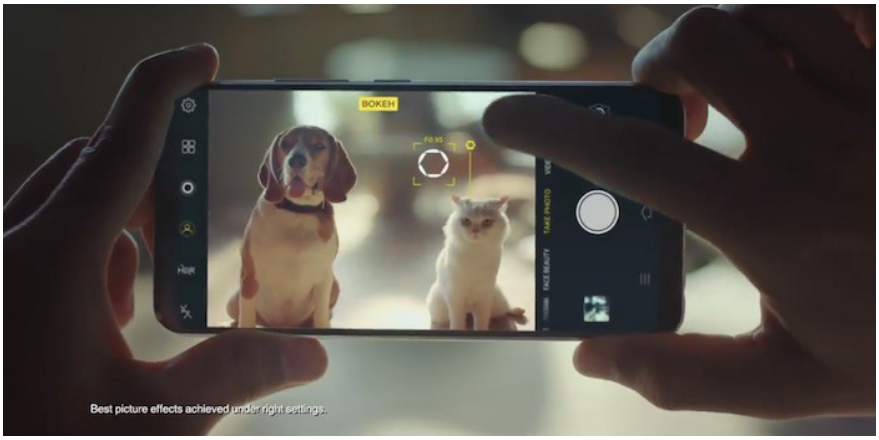
From Automated self-driven cars to Boosting augmented reality applications and gaming, from Image and Face Recognition on Social Networks to Its application in various Medical fields, Image Recognition has emerged as a powerful tool and has become a vital for many upcoming inventions.



So, why not create our own Image Recognition Classifier, and that too with a few lines of code, thanks to the modern day machine learning libraries. Let's get started !!

Getting Started—Dog or Cat

Well, not asking what you like more. Lets first create a simple image recognition tool that classifies whether the image is of a dog or a cat. The idea is to create a simple Dog/Cat Image classifier and then applying the concepts on a bigger scale.



Tools And Technologies

1. **Anaconda**—Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications, that aims to simplify package management and deployment. You can download it from the link below according to your system
https://www.anaconda.com/download/

2. **Spyder**—Spyder is an open source cross-platform IDE for scientific programming in the Python language. It comes installed with anaconda. If not, install it using anaconda navigator.

3. **Tensorflow**—TensorFlow is an open-source software library for dataflow programming across a range of tasks. Download link—
https://www.tensorflow.org/install/install_windows

4. **Keras**—Keras is an open source neural network library written in Python. Activate Tensorflow env and install keras using 'pip install keras'.

5. **CNN**—Convolution Neural network , a class of deep, feed-forward artificial **neural networks**, most commonly applied to analyzing visual imagery. Here is a very good explanation to what it actually

is -*https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/*

Plan of Attack

Its time to get into action and here is the plan —

- Collecting the Dataset
- Importing Libraries and Splitting the Dataset
- Building the CNN
- Full Connection
- Data Augmentation
- Training our Network
- Testing

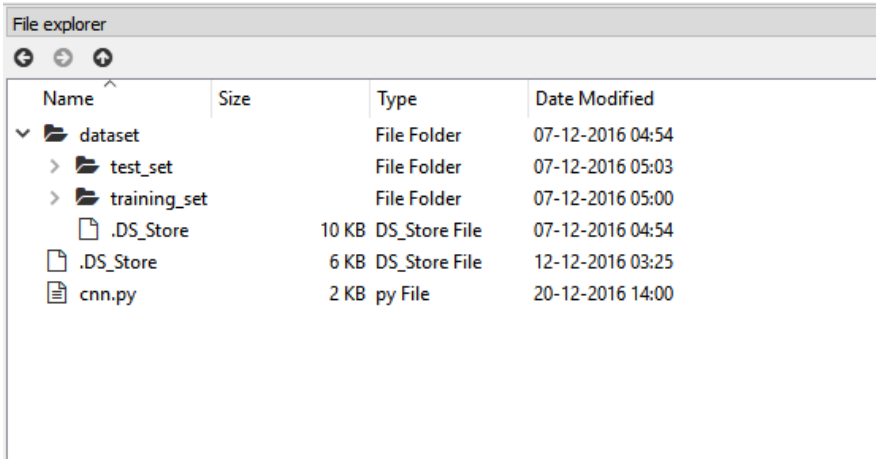
Step 1—Collecting the Dataset

In order to train our machine, we need a huuuuggge amount of data so that our model can learn from them by identifying out certain relations and common features related to the objects.

Sample of cats & dogs images from Kaggle Dataset

Fortunately many such datasets are available on internet. Here is a link for the cats and dogs dataset which consist of 10000 images—5000 of each. This will help in training as well testing our classifier.

http://www.superdatascience.com/wp-content/uploads/2017/03/Convolutional-Neural-Networks.zip



Step 2—Importing Libraries and Splitting the Dataset

To use the powers of the libraries, we first need to import them.

```
#Convolutional Neural network

#Importing the Keras libraries and packages
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

```

After importing the libraries, we need to split our data into two parts- taining_set and test_set.

In our case, the dataset is already split into two parts. The training set has 4000 image each of dogs and cats while the test set has 1000 images of each.

Step 3—Buliding the CNN

This is most important step for our network. It consists of three parts -

1. Convolution
2. Polling
3. Flattening

The primary purpose of Convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

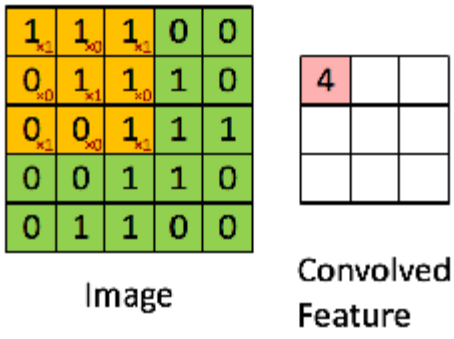
Since every image can be considered as a matrix of pixel values. Consider a 5 x 5 image whose pixel values are only 0 and 1 (note that for a grayscale image, pixel values range from 0 to 255, the green matrix below is a special case where pixel values are only 0 and 1):

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Also, consider another 3 x 3 matrix as shown below:

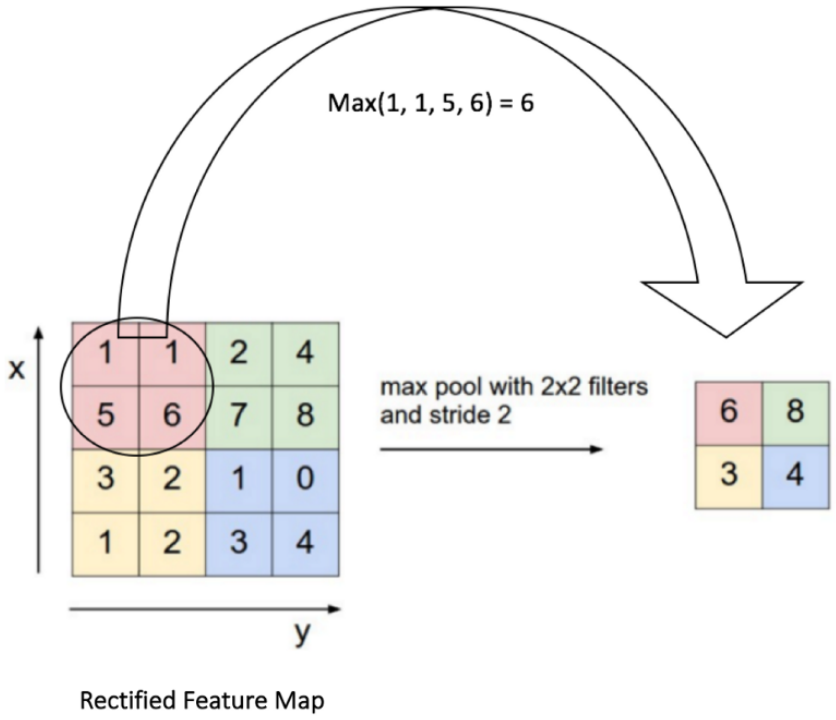
1	0	1
0	1	0
1	0	1

Then, the Convolution of the 5 x 5 image and the 3 x 3 matrix can be computed as shown in the animation in **Figure 5** below:



The obtained matrix is also known as the feature map. An additional operation called ReLU is used after every Convolution operation. The next step is of pooling.

Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. In 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.

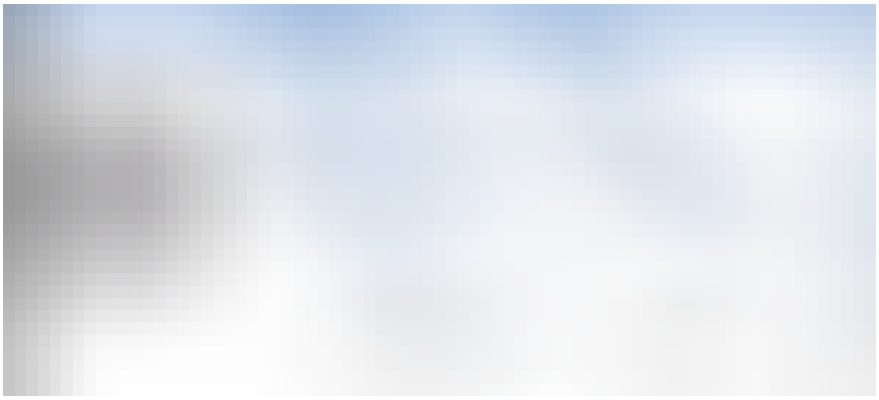


After pooling comes flattening. Here the matrix is converted into a linear array so that to input it into the nodes of our neural network.

Let's come to the code.

```
#Convolutional Neural network

#Importing the Keras libraries and packages
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

#Initialize the CNN
classifier = Sequential()

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

#Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

#Step 3 - Flattening
classifier.add(Flatten())

```

So now our CNN network looks like this



Step 4—Full Connection

Full connection is connecting our convolutional network to a neural network and then compiling our network.

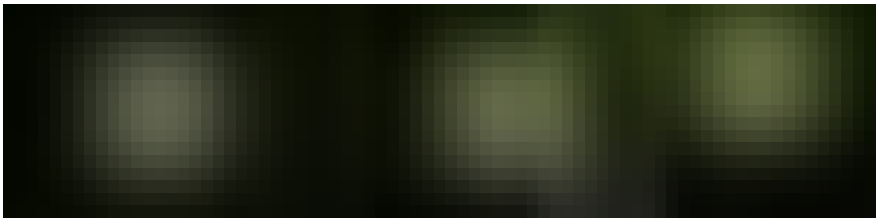
Here we have made 2 layer neural network with a sigmoid function as an activation function for the last layer as we need to find the probability of the object being a cat or a dog.

So now the final network looks something like this -

Step 5—Data Augmentation

While training your data, you need a lot of data to train upon. Suppose we have a limited number of images for our network. What to do now??

You don't need to hunt for novel new images that can be added to your dataset. Why? Because, neural networks aren't smart to begin with. For instance, a poorly trained neural network would think that these three tennis balls shown below, are distinct, unique images.



The same tennis ball, but translated.

So, to get more data, we just need to make minor alterations to our existing dataset. Minor changes such as flips or translations or rotations. Our neural network would think these are distinct images anyway.

Data augmentation is a way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data. The field of data augmentation is not new, and in fact, various data augmentation techniques have been applied to specific problems.

And here goes the code

Now we have a huge amount of data and its time for the training.

Step 6—Training our Network

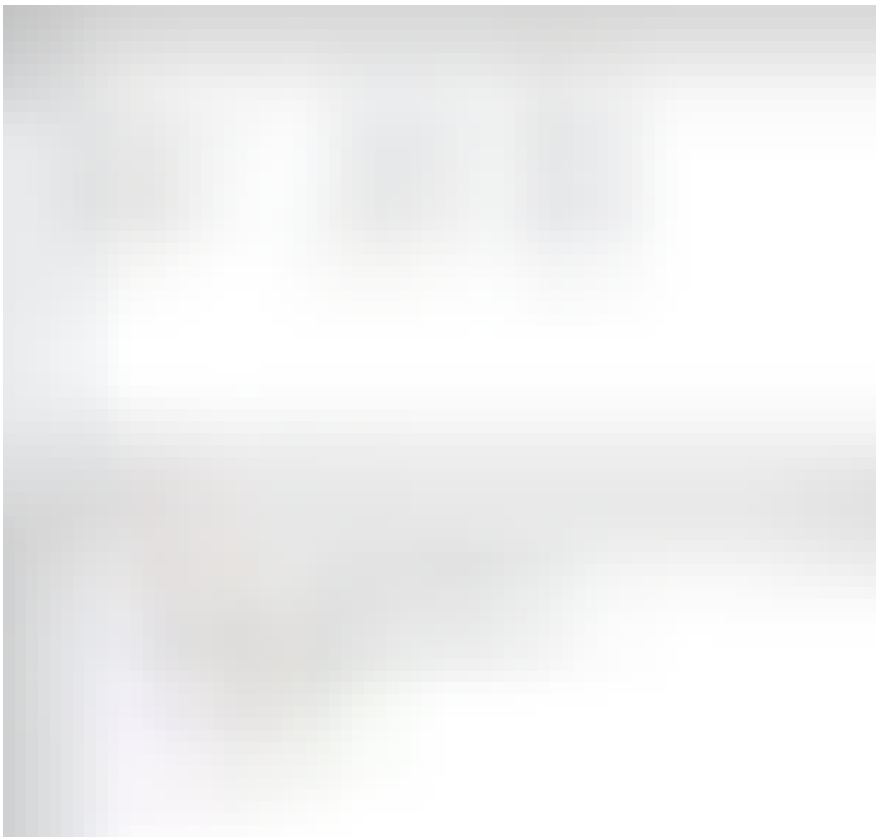
So, we completed all the steps of construction and its time to train our model.

If you are training with a good video card with enough RAM (like an Nvidia GeForce GTX 980 Ti or better), this will be done in less than an hour. If you are training with a normal cpu, it might take a lot longer.

With increasing number of epochs, the accuracy will increase.

Step 7—Testing

Now lets test a random image.



And, yes !! our network correctly predicted the image of the dog!! Though it is not 100% accurate but it will give correct predictions most of the times. Try adding more convolutional and pooling layers, play with the number of nodes and epochs, and you might get high accuracy result.

You can even try it with your own image and see what it predicts. Whether you look close to a dog or a cat.

Summary

So, we created a simple Image Recognition Classifier. The same concept can applied to a diverse range of objects with a lot of training data and appropriate network. You can change the dataset with the images of your friends and relatives and work upon the network to make a Face Recognition Classifier.

So, now you know how to build it,

> "Be quiet, darling. Let pattern recognition have its way."
> — ***William Gibson***, ***The Peripheral***

However there are many APIs available which can be automatically embedded into our application. They have been trained on a large dataset and powerful machines. If you don't want to go deep in machine learning, you can use them.

But for machine learning enthusiasts, this is a very good start for them to learn the basics and dive to an ocean of infinite possibilities!!