

# Dog vs Cat Image Classification

CS 337

Team:

Piyush Onkar (160050012)

Mohit Korabu (160050065)

Instructor:

Prof. Ganesh Ramakrishnan

## Overview

The Dogs vs. Cats dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or cat.

Although the problem sounds simple, it was only effectively addressed in the last few years using deep learning convolutional neural networks. While the dataset is effectively solved, it can be used as a basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch.

This includes how to develop a robust test harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.

## Description of the Problem

The dataset for this problem is comprised of photos of dogs and cats provided as a subset of photos from a much larger dataset of 3 million manually annotated photos. The dataset is taken from [Kaggle](https://www.kaggle.com/c/dogs-vs-cats) website. The photos are named either starting with 'cat' or 'dog' respectively for training purposes. The *splitTestandTrain.py* file is used to separate the photos into 'cat' and 'dog' directories respectively in the 'dataset' directory which will be used for training and testing respectively.

The photos will have to be reshaped prior to modeling so that all images have the same shape. This is often a small square image(224,224).

There are many ways to achieve this, although the most common is a simple resize operation that will stretch and deform the aspect ratio of each image and force it into the new shape. We could load all photos and look at the distribution of the photo widths and heights, then design a new photo size that best reflects what we are most likely to see in practice.

Smaller inputs mean a model that is faster to train, and typically this concern dominates the choice of image size.

### Model for our problem:

We are using a VGG16 model for our project. VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". It was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. We have used tensorflow==2.0 for creating model.

The summary of the model can be seen below:


Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080

block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3211392
dense_1 (Dense)	(None, 1)	129

=====  
Total params: 17,926,209

Trainable params: 3,211,521

Non-trainable params: 14,714,688



The 'include\_top' keras parameter being false means our model can be used for feature extraction, for example to build an autoencoder or stack any other model on top of it. Also, 'input\_shape' parameter can be only used when 'include\_top' is false. 'Input\_shape' is a shape tuple (a tuple of integers or None entries, where None indicates that any positive integer may be expected). In input\_shape, the batch dimension is not included.

We have used Stochastic Gradient Descent Algorithm in this project. The word 'stochastic' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge.

In this project we have kept the batch size of 64 for the epoch. We have only 1 iteration which has a Training Accuracy of around 97% and Testing Accuracy of around 96%. Since this is a pretty deep convolutional network consisting of more than 16 layers. It takes a very long time for training.

The code for training model is mentioned in model.py and code for predicting the output of unknown image whether is present in predict.py. Pass the path of the image as command line argument to predict.py and it will predict the the image belongs to dog or cat.