**Object Detection and Image Recognition using Python**

***A***

***Major Project Report***

*Submitted in partial fulfilment of the requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE & ENGINEERING**

**By**

Paras Sharma (Enrol. No. R114216017)

**Under the guidance of**

Ms. Aradhana Kumari Singh

Assistant Professor, Department of Informatics

School Of Computer Science



**SCHOOL OF COMPUTER SCIENCE**

UNIVERSITY OF PETROLEUM & ENERGY STUDIES

Bidholi Campus, Energy Acres, Dehradun – 248007.

**Year – 2019**

**Candidate’s Declaration**

I hereby certify that the project work entitled “Object Detection & Image Recognition using Python” in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING with specialization in E-Commerce, Retail and Automation submitted to the School of Computer Science, Department of Informatics, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my work carried out during a period from August,2019 to December,2019 under the supervision of Ms. Aradhana Kumari Singh, Department of Informatics. The matter presented in this project has not been submitted by me for the award of any other degree of this or any other University.

Paras Sharma

R114216017

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. T.P. Singh Ms. Aradhana Kumari Singh

HOD- Informatics Project Guide

School of Computer Science

University of Petroleum & Energy Studies

Dehradun- 248001 (Uttarakhand)

**Acknowledgement**

I wish to express our deep gratitude to my guide Ms. Aradhana Kumari Singh, for all advice, encouragement and constant support she has given me throughout my project work. This work would not have been possible without her support and valuable suggestions.

I sincerely thank to our respected HoD of the Department, Dr. T.P. Singh, for his great support in doing our project in Object Detection & Image Recognition using Python at SoCS.

I am also grateful to Dr. Manish Prateek, Director, SoCS, UPES for giving me the necessary facilities to carry out my project work successfully.

I would like to thank all my friends for their help and constructive criticism during my project work. Finally, I have no words to express my sincere gratitude to my parents who have shown me this world and for every support, they have given me.

Paras Sharma

R114216017

**Abstract**

With the tremendous increase in the amount of images that are being produced daily, there was a need for the development of a robust and efficient object detection system. Object recognition is the process by which objects are detected within images and videos.

Object detection can be used for various purposes including retrieval and surveillance. In this study, various basic concepts used in object detection are described while making use of OpenCV library of python 2.7, improving the efficiency and accuracy of object detection are presented.

**Keywords:​** *Object Detection, Python, OpenCV, NumPy, Haar Cascade*

**Introduction**

The modern world is encircled with heavy masses of digital visual information, may it be images, videos and so on. It is a need of hour to analyse and organise these plundering ocean of visual information and for these techniques are required. Particularly, it will be more useful to analyse semantic information of the images or videos. One imperative part of image content is the objects in the image.

So there is a need for object detection and image recognition techniques.

Object detection is a vital, yet difficult vision task. It is a basic part in numerous applications, for example, image search, image auto-annotation and scene understanding; be that as it may it is as yet an open issue because of the intricacy of object classes and images. It is being widely used in industries to ease user, save time and to achieve parallelism. Object detection is a part of computer vision which aims at having a human like vision, which can locate and differentiate various objects such as, numbers, location, size, position etc. The common object detection method is the colour-based approach, detecting objects based on their colour values.

Object detection is one of the most challenging applications of the image processing. It is a branch of computer vision and artificial intelligence. The aim of this project is to identify and locate the objects in the images or videos and also naming the specific objects detected.

OpenCv (Python) is used for the implementation of the project. NumPy library is also required for the same. Also, an OpenCv algorithm i.e. Haar Cascade is used and Haar like features are emphasized.

**Problem Statement**

It is not possible for machine to identify an object and classify them according to different classes.

Hence, the aim of this project is to identify and locate the objects in the images or videos and also naming the specific objects detected. For instance, we are taking 2,000 images of Cats and Dogs for our Dataset.

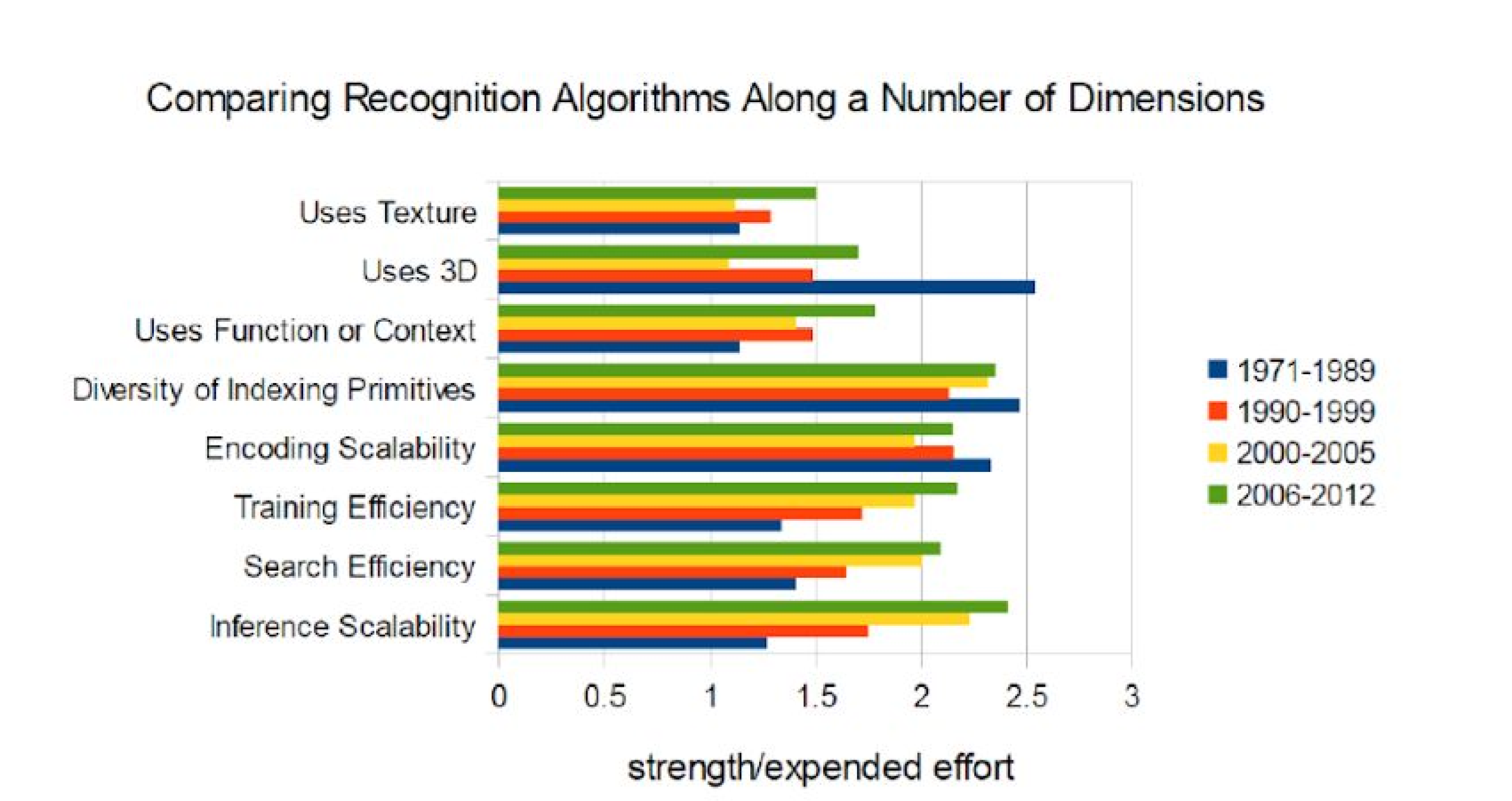
The 2,000 images used in this project are excerpted from the ["**Dogs vs. Cats**" dataset](https://www.kaggle.com/c/dogs-vs-cats/data) available on Kaggle, which contains 25,000 images. Here, we use a subset of the full dataset to decrease training time for educational purposes.

**Literature Review**

Modern computer vision approach has its starting point in early 1960s. First and foremost application was pattern recognition systems used in offices. While performing a task, some practical difficulties such as scene complexities increased as some illumination variability increased. Also time, cost and sensor noise constraints became for common.

1964 involved the automation of the wire-bonding process of transistors, with the ultimate goal of replacing human workers. However, it attained 95% accuracy in test labs but regarded too low to replace human workers. By 1973, fully automated assembly machines had been made, resulting in the world’s 1st image-based machine for the automatic assembly of semiconductor devices. It is much more evolved in these 50 years and achieved almost human accuracy.

As the experiments went on and on making the systems more efficient and reducing the need of human workers, computer vision has now become an effective and efficient technology. Working very much like human vision. It is now being used in various fields like face recognition, autonomous cars, and robots and of course object detection and recognition.



# **Computer Vision**

Humans use their eyes and brain to see and spot the objects around. Computer vision is the science giving similar functionality and capability to a machine or a computer. It aims at enabling computers to see, identify and process the images in the same way as human vision does, and then provide appropriate output. It resembles giving human intelligence and instincts to a computer.

**Object Detection**

It is the technology in the field of computer vision for finding and identifying objects in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different viewpoints, in many different sizes and scales or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer vision systems. Many approaches to the task have been implemented over multiple decades.

**Applications**

Object recognition technology has matured to a point at which exciting applications are becoming possible. Indeed, industry has created a variety of computer vision products and services from the traditional area of machine inspection to more recent applications such as video surveillance, or face recognition.

**Face detection**

Popular applications include face detection and people counting. Have you ever noticed how Facebook detects your face when you upload a photo? This is a simple application of object detection that we see in our daily life.

**People Counting**

Object detection can be also used for people counting, it is used for analysing store performance or crowd statistics during festivals. These tend to be more difficult as people move out of the frame quickly (also because people are non-rigid objects).

**Vehicle detection**

Similarly when the object is a vehicle such as a bicycle or car, object detection with tracking can prove effective in estimating the speed of the object. The type of ship entering a port can be determined by object detection (depending on shape, size etc.). This system for detecting ships are currently in development in some European countries.

**Manufacturing Industry**

Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects. Hough circle detection transform can be used for detection.

**Online images**

Apart from these object detection can be used for classifying images found online. Obscene images are usually filtered out using object detection.

**Security**

In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quadcopter).

**Content Based Image Retrieval (CBIR)**

Many retrieval methods that accept query images as input from the user represent images as vectors in the feature space and search for images based on their features and feature representations. When the user presents a sample query image, region of interest (ROI), or pattern to the system, it performs various visual query mechanisms, such as the query-by example (QBE) paradigm, and finally outputs the relevant images. In CBIR, image content is frequently represented using image features. CBIR finds applications in internet, advertising, medicine, crime detection, entertainment, and digital libraries. High retrieval efficiency and less computational complexity are the desired characteristics of CBIR system and they are the key objectives in the design of a CBIR system.

**Convolutional Neural Network (CNN)**

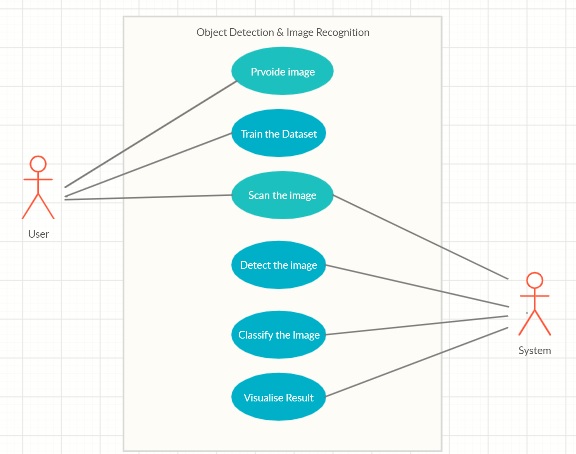
A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. **Object Detection comes under Convolutional Neural Network (CNN).**

**Objective**

The aim of this project is to identify and locate the objects in the images or videos and also naming the specific objects detected.

For this purpose, we will be using “Dogs Vs. Cats” Dataset from Kaggle from which we would be feeding over 2,000 images of Dogs and Cats to the neural network for classification.

**Use Case (Object Detection & Image Recognition)**

****

**Methodology**

**Implementation of Object Detection**

In Object Detection, a particular object will be detected from whole image. For example, from an image, consisting of several animals, we want to find particularly image of a cat, then a set of images of cat will be stored and image detector will be trained. It is a part of computer vision.

**Selecting a Model**

The default model in the notebook is the simplest (and fastest) pre-trained model offered by Tensor Flow. Looking at the table below, you can see there are many other models available. mAP stands for mean average precision, which indicates how well the model performed on the COCO dataset. Generally models that take longer to compute perform better.

To get a rough approximation for performance just try each model out on a few sample images. If the item you are trying to detect is not one of the 90 COCO classes, find a similar item (if you are trying to classify a squirrel, use images of small cats) and test each model’s performance on that.

**COCO Dataset**

COCO is a large image dataset designed for object detection, segmentation, person key points detection, stuff segmentation, and caption generation. COCO dataset contains photos of 91 objects types that would be easily recognizable by a 4 year old. With a total of 2.5 million labelled instances in 328k images, the creation of 25 our dataset drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting and instance segmentation

**Implementation of CBIR using Python**

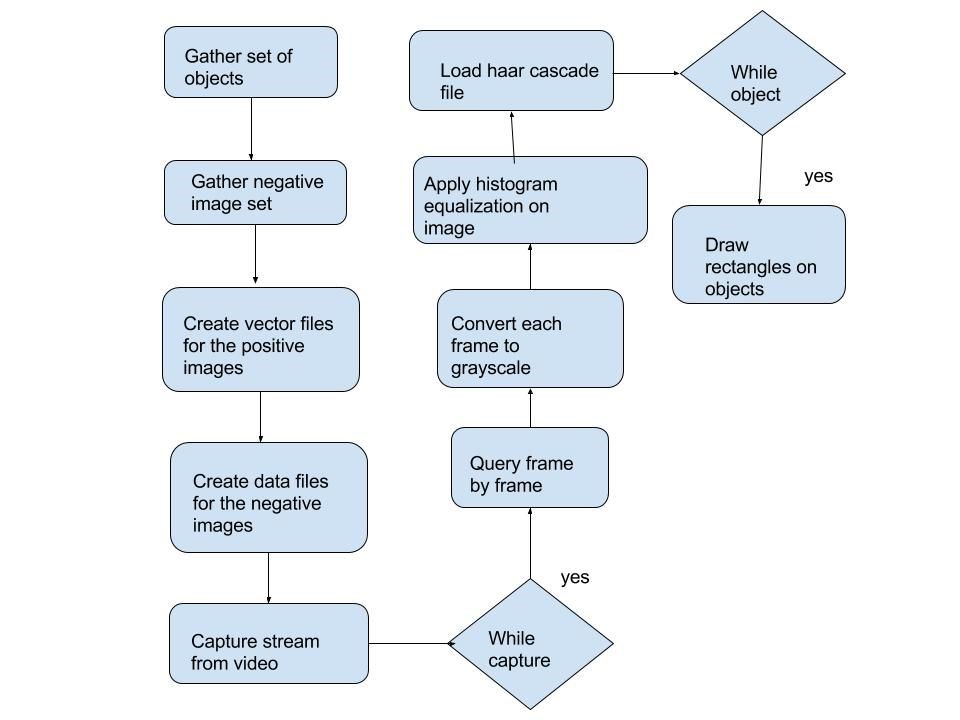
In CBIR, we had created a retrieval system that used the content of image. The user had to provide a query image, color histogram of the query image would be calculated and the closest images from the database would be retrieved.

**The Four Steps of Any CBIR System**

Any Content Based Image Retrieval System has four basic steps –

1. **Defining the Image Descriptor**: We need to decide what aspect of the image we want to describe namely - the colour of the image, the shape or the texture of the image.
2. **Indexing the Dataset :** Now that we have the image descriptor defined, we need to apply this image descriptor to each image in the dataset, extract 22 features from these images, and write the features to storage (ex. CSV file, RDBMS, Redis, etc.) so that they can be later compared for similarity.
3. **Defining the similarity metric**: After indexing the dataset, we now have a bunch of feature vectors. We need to compare these feature vectors. Popular choices include the Euclidean distance, Cosine distance, and chi-squared distance, but the actual choice is highly dependent on (1) the dataset and (2) the types of features that we have extracted.
4. **Searching:** The final step is to perform an actual search. A user will submit a query image and we have to (1) extract features from this query image and then (2) apply the similarity function to compare the query features to the features already indexed. Then, the most relevant results are returned according to the similarity function.

**Implementation of Model**



An object detection and image recognition model follows broadly two basic steps that are:

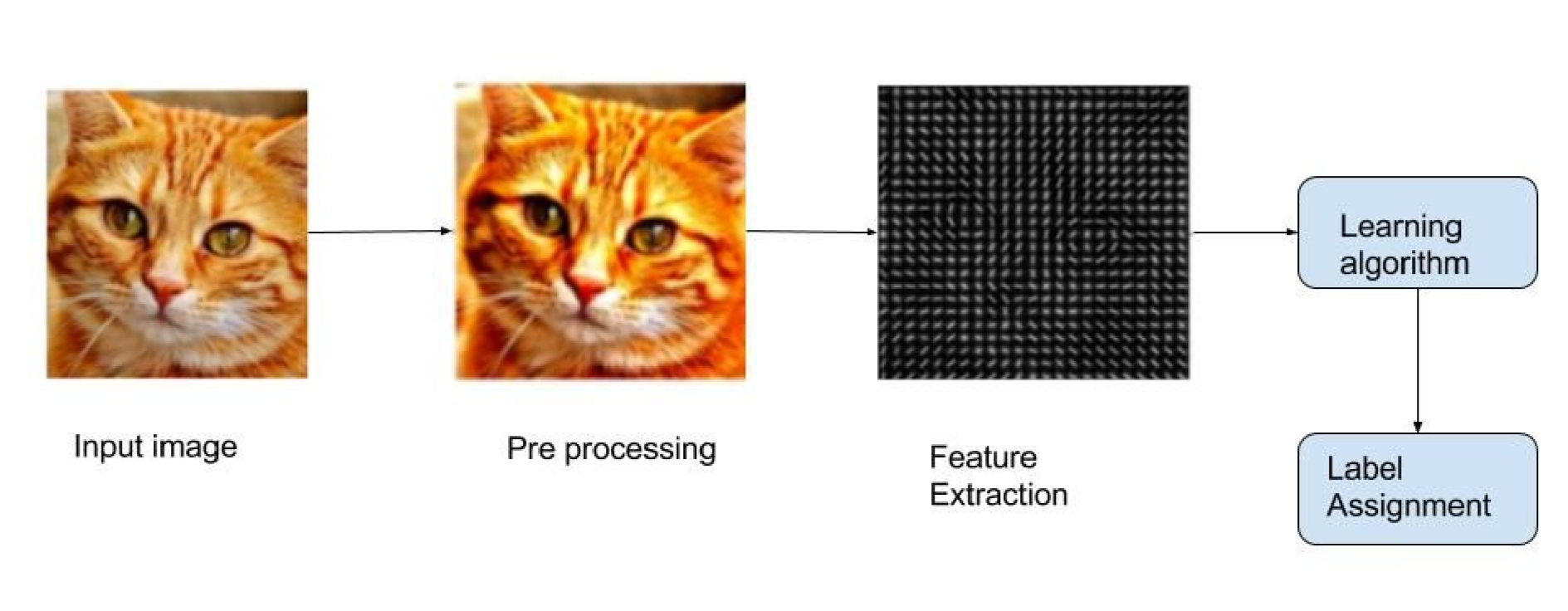
1. Image Classification
2. Detection

# **Image Classification**

An image classifier follows an algorithm which takes an image as input and outputs what the image contains. The output is the image label which tells about the class to which the image belongs to (for example, cat, elephant, chair etc.). But for recognising the content of the image, image classifier has to be trained well. It is, therefore, trained by giving different set of positive and negative images to differentiate among different features and to give the desired output.

# **Anatomy of an Image Classifier**

The following diagram shows how an image classifier works:



# **Step 1: Pre-processing**

Firstly an input picture is pre-processed to normalize the contrast and brightness effects. Also, pre-processing step involves subtracting the mean of image intensities and divide by the standard deviation. Methods like gamma correction and colour space transformation can also be used for the same and may result in better results, sometimes.

After that, the input image is cropped and resized to a fixed size image as it is integral part for the feature extraction.

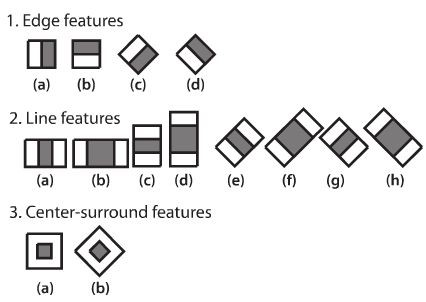
# **Step 2: Feature Extraction**

It is very important to remove the extra information contained in the image that is not necessary for the classification. Therefore, firstly simplification of the image by extracting only important information and leaving the rest is done. It can be done by running an edge detector on the image. Some​ well-known features used in computer vision are Haar-like features introduced by Viola and Jones, Histogram of Oriented Gradients ( HOG ), Scale-Invariant Feature Transform ( SIFT ), Speeded Up Robust Feature ( SURF ) etc.

We will be using Haar-like features for the implementation of our model.

# **Haar-like features**

It is an OpenCv algorithm based on machine learning approach where a cascade function is trained by a lot of positive and negative images. After input of positive and negative images, features are being extracted based on Haar features shown in the figure below.



Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. Now all possible sizes and locations of each kernel is used to calculate overflowing features. To make is less cumbersome integral images were introduced which simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. It surely makes the process super-fast but also, extract many irrelevant features. So for extracting the best and important features from those 160000+ features, *Adaboost*​works well.

Each and every feature is applied in all the training images and best threshold which will classify the object is found. There will surely be errors and misclassifications but we choose the one with minimum error rate, which means they are the best features which classifies the object best. The process is continued to achieve new error rates and weights to meet the accuracy. Final classifier is a weighted sum of these weak classifiers.

Now, the final setup has around 6000 features. By *Cascade Classifier*, these 6000 features are applied on different stages. If a window fails the first stage, it is discarded and if not it goes to the next stage. It saves a lot of time as comparing 6000 features at a time.

# **Detection**

For the detection purpose,

1. A sliding window is performed over the image, i.e., it moves through the whole image and at different scales.
2. For each windows of the sliding window, features are extracted, computed and classified.
3. A rectangle is drawn and the name is flashed on the object being detected.

**Collecting Dataset for Object Detection & Image Recognition**

* First of all, we have downloaded the example dataset, which includes a zip folder of 2,000 pictures of cats and dogs and we extracted it locally in /tmp. The dataset which we are using is being taken from Kaggle, which contains 25,000 images.



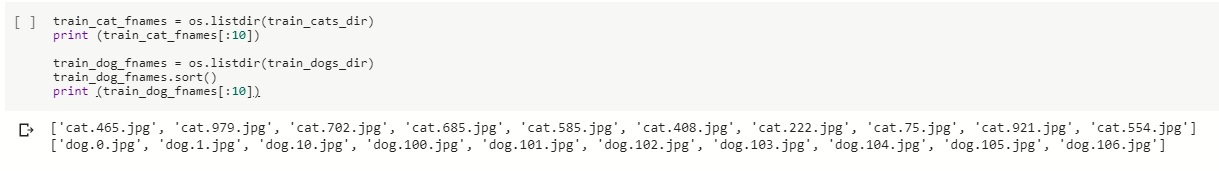
* Then, we have extracted the contents of zip file to the base directory /tmp/cats\_and\_dogs\_filtered, which contains train and validation subdirectories for the training and validation datasets which in turn each contains cats and dogs subdirectories.



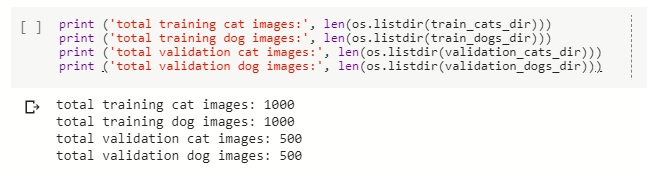
* The contents of the .zip are extracted to the base directory /tmp/cats\_and\_dogs\_filtered, which contains train and validation subdirectories for the training and validation datasets which in turn each contain cats and dogs subdirectories. Let's define each of these directories:



* Now, let's see what the filenames look like in the cats and dogs train directories (file naming conventions are the same in the validation directory):

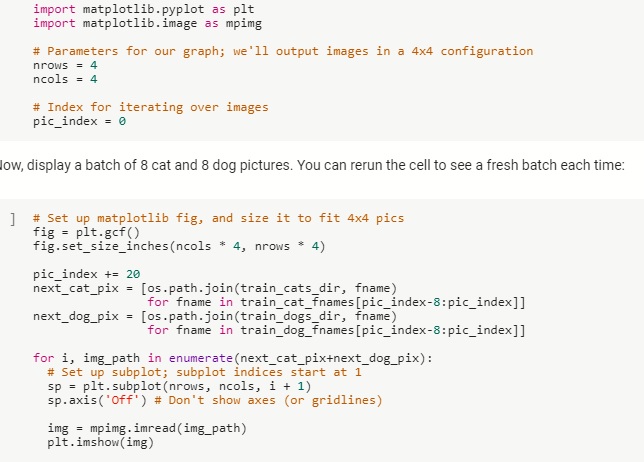


* Let's find out the total number of cat and dog images in the `train` and `validation` directories:



* For both cats and dogs, we have 1,000 training images and 500 test images.

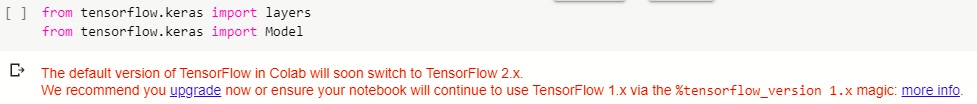
Now let's take a look at a few pictures to get a better sense of what the cat and dog datasets look like. First, configure the matplot parameters:

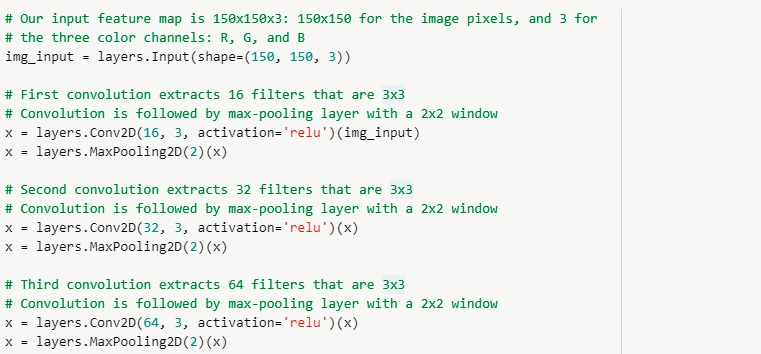


**Building a Small Convnet from Scratch to obtain some Accuracy**

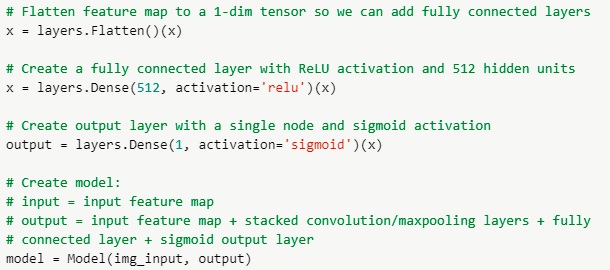
The images that will go into our convnet are 150x150 color images (We would be handling to resize the images to 150x150 before feeding them into the neural network, in Data Preprocessing).

Now we will code up the architecture. We will stack 3 {convolution + relu + maxpooling} modules. Our convolutions operate on 3x3 windows and our maxpooling layers operate on 2x2 windows. Our first convolution extracts 16 filters, the following one extracts 32 filters, and the last one extracts 64 filters.

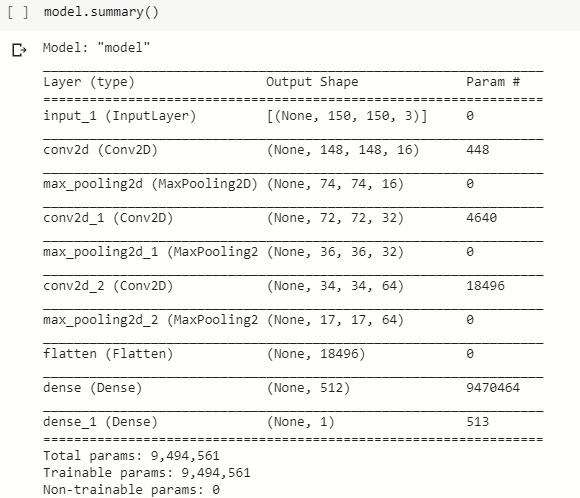




On top of it we stick two fully-connected layers. Because we are facing a two-class classification problem, i.e. a binary classification problem, we will end our network with a [**sigmoid *activation***](https://wikipedia.org/wiki/Sigmoid_function)***,*** so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0).

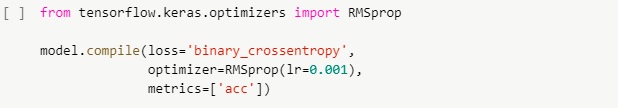
****

Now, we will summarise the whole model which is given as follows:



The "output shape" column shows how the size of your feature map evolves in each successive layer. The convolution layers reduce the size of the feature maps by a bit due to padding, and each pooling layer halves the feature map.

Next, we'll configure the specifications for model training. We will train our model with the binary\_crossentropy loss, because it's a binary classification problem and our final activation is a sigmoid. (For a refresher on loss metrics, see the Machine Learning Crash Course.) We will use the rmsprop optimizer with a learning rate of 0.001. During training, we will want to monitor classification accuracy.

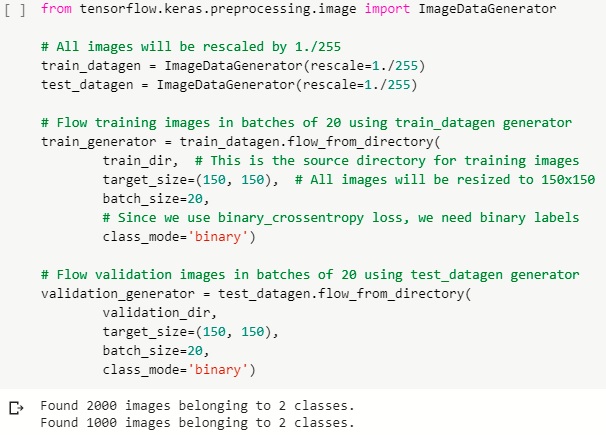
****

**Data Pre-processing of Dataset**

Let's set up data generators that will read pictures in our source folders, convert them to float32 tensors, and feed them (with their labels) to our network. We'll have one generator for the training images and one for the validation images. Our generators will yield batches of 20 images of size 150x150 and their labels (binary).

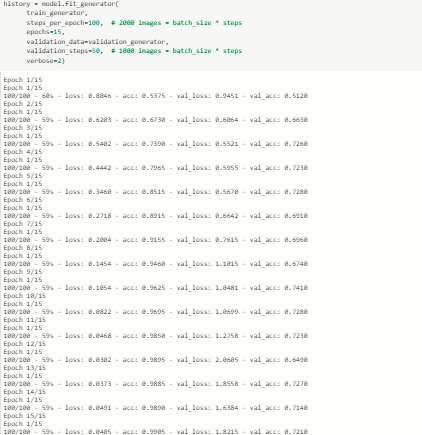
As you may already know, data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network. (It is uncommon to feed raw pixels into a convnet.) In our case, we will preprocess our images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class using the rescale parameter. This ImageDataGenerator class allows you to instantiate generators of augmented image batches (and their labels) via .flow (data, labels) or .flow\_from\_directory (directory). These generators can then be used with the Keras model methods that accept data generators as inputs: fit\_generator, evaluate\_generator, and predict\_generator.



**Training the Dataset**

Now, we will be training our given dataset and it is given as follows:

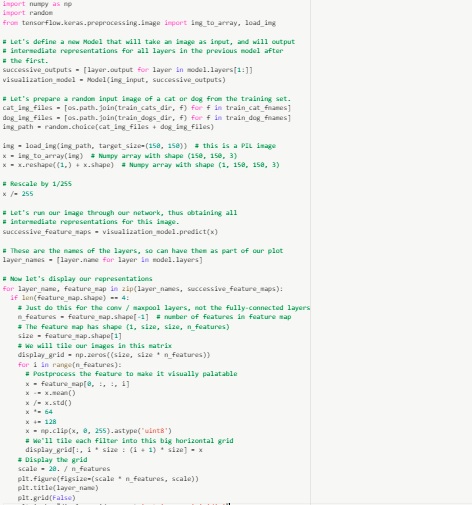


Here, we can observe that after each epoch, the accuracy of our dataset is increasing starting from 53.75% after 1st epoch to 99.05% after final epoch.

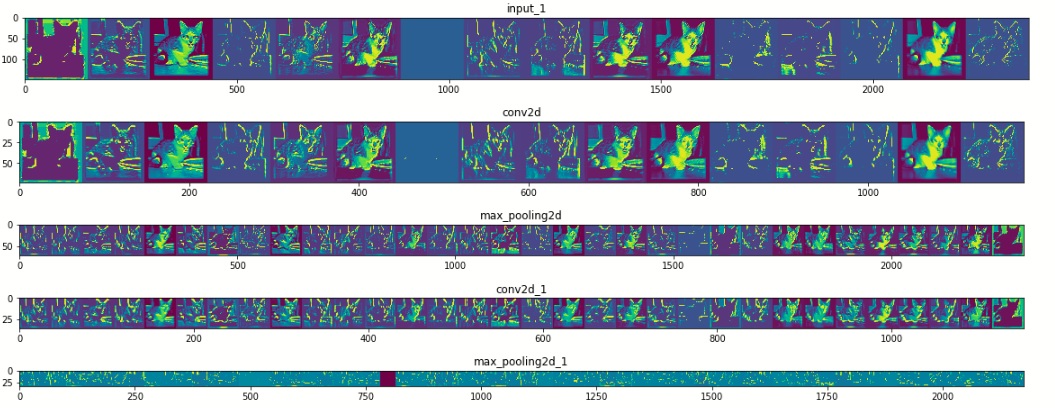
**Visualizing Intermediate Representations**

Now we will visualize how an input gets transformed as it goes through the convnet. This would help us to get feel for what kind of features our convnet has learned.

Let's pick a random cat or dog image from the training set, and then generate a figure where each row is the output of a layer, and each image in the row is a specific filter in that output feature map. Rerun this cell to generate intermediate representations for a variety of training images.



The output for the following snippet is given as follows:

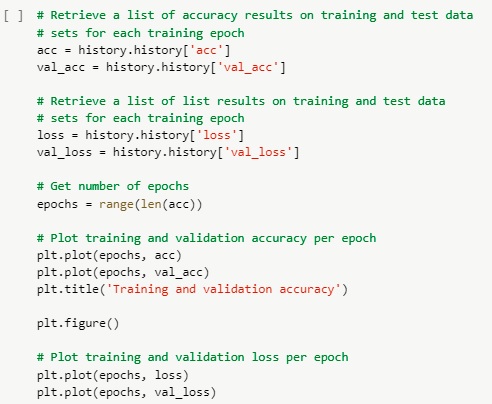


As we can see we go from the raw pixels of the images to increasingly abstract and compact representations. The representations downstream start highlighting what the network pays attention to, and they show fewer and fewer features being "activated"; most are set to zero. This is called "sparsity." Representation sparsity is a key feature of deep learning.

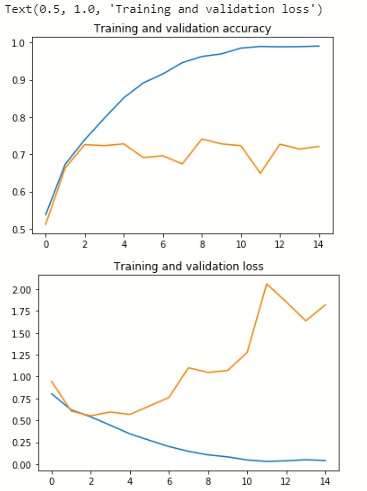
These representations carry increasingly less information about the original pixels of the image, but increasingly refined information about the class of the image. You can think of a convnet (or a deep network in general) as an information distillation pipeline.

**Evaluating Accuracy & Loss of Model**

Let’s plot the training/validation accuracy and loss as collected during training: Let's plot the training/validation accuracy and loss as collected during training:



The output for the above snippet is given as follows:



As you can see, we are **overfitting** like it's getting out of fashion. Our training accuracy (in blue) gets close to 100% (!) while our validation accuracy (in green) stalls as 70%. Our validation loss reaches its minimum after only five epochs.

Since we have a relatively small number of training examples (2000), overfitting should be our number one concern. Overfitting happens when a model exposed to too few examples learns patterns that do not generalize to new data, i.e. when the model starts using irrelevant features for making predictions. For instance, if you, as a human, only see three images of people who are lumberjacks, and three images of people who are sailors, and among them the only person wearing a cap is a lumberjack, you might start thinking that wearing a cap is a sign of being a lumberjack as opposed to a sailor. You would then make a pretty lousy lumberjack/sailor classifier.

**System Requirements**

* **Operating System:** Windows 10
* **Minimum Hardware:** 2 GB RAM, 1GB Free Hard Disk Storage, 32/64 –Bit Processor (Intel i5)
* **Environment :** Google Colaboratory (For Python)
* **Programming Language to be used:** Python 3.6
* **Libraries:** CV2, NumPy, Keras, Matplotlib
* **Dataset to be used:** COCO Dataset , Images of Cats & Dogs used from Kaggle

**Code & Demo**

import os

import zipfile

local\_zip = '/tmp/cats\_and\_dogs\_filtered.zip'

zip\_ref = zipfile.ZipFile(local\_zip, 'r')

zip\_ref.extractall('/tmp')

zip\_ref.close()

The above code is used to extract the contents of zip file to the base directory, which contains train and validation subdirectories for the training and validation datasets which in turn each contain cats and dogs subdirectories. The code given below will help us to define each of these directories:

base\_dir = '/tmp/cats\_and\_dogs\_filtered'

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

# Directory with our training cat pictures

train\_cats\_dir = os.path.join(train\_dir, 'cats')

# Directory with our training dog pictures

train\_dogs\_dir = os.path.join(train\_dir, 'dogs')

# Directory with our validation cat pictures

validation\_cats\_dir = os.path.join(validation\_dir, 'cats')

# Directory with our validation dog pictures

validation\_dogs\_dir = os.path.join(validation\_dir, 'dogs')

Now the given code below will help us to find what the filenames look like in the cats and dogs train directories:

train\_cat\_fnames = os.listdir(train\_cats\_dir)

print (train\_cat\_fnames[:10])

train\_dog\_fnames = os.listdir(train\_dogs\_dir)

train\_dog\_fnames.sort()

print (train\_dog\_fnames[:10])

Now we will find out the number of cat and dog images in the train and validation directories:

print ('total training cat images:’ len(os.listdir(train\_cats\_dir)))

print ('total training dog images:’ len(os.listdir(train\_dogs\_dir)))

print ('total validation cat images:’ len(os.listdir(validation\_cats\_dir)))

print ('total validation dog images:’ len(os.listdir(validation\_dogs\_dir)))

For both cats and dogs, we have 1,000 training images and 500 test images.

Now let's take a look at a few pictures to get a better sense of what the cat and dog datasets look like. First, configure the matplot parameters:

%matplotlib inline

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration

nrows = 4

ncols = 4

# Index for iterating over images

pic\_index = 0

Now, we will display a batch of 8 cat and 8 dog pictures which is given in the below code:

# Set up matplotlib fig, and size it to fit 4x4 pics

fig = plt.gcf()

fig.set\_size\_inches(ncols \* 4, nrows \* 4)

pic\_index += 20

next\_cat\_pix = [os.path.join(train\_cats\_dir, fname)

for fname in train\_cat\_fnames[pic\_index-8:pic\_index]]

next\_dog\_pix = [os.path.join(train\_dogs\_dir, fname)

for fname in train\_dog\_fnames[pic\_index-8:pic\_index]]

for i, img\_path in enumerate(next\_cat\_pix+next\_dog\_pix):

# Set up subplot; subplot indices start at 1

sp = plt.subplot(nrows, ncols, i + 1)

sp.axis('Off') # Don't show axes (or gridlines)

img = mpimg.imread(img\_path)

plt.imshow(img)

plt.show()

The images that will go into our convnet are 150x150 color images (We would be handling to resize the images to 150x150 before feeding them into the neural network, in Data Preprocessing).

Now we will code up the architecture. We will stack 3 {convolution + relu + maxpooling} modules. Our convolutions operate on 3x3 windows and our maxpooling layers operate on 2x2 windows. Our first convolution extracts 16 filters, the following one extracts 32 filters, and the last one extracts 64 filters.

from tensorflow.keras import layers

from tensorflow.keras import Model

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for

# the three color channels: R, G, and B

img\_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3

# Convolution is followed by max-pooling layer with a 2x2 window

x = layers.Conv2D(16, 3, activation='relu')(img\_input)

x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 32 filters that are 3x3

# Convolution is followed by max-pooling layer with a 2x2 window

x = layers.Conv2D(32, 3, activation='relu')(x)

x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3

# Convolution is followed by max-pooling layer with a 2x2 window

x = layers.Conv2D(64, 3, activation='relu')(x)

x = layers.MaxPooling2D(2)(x)

# Flatten feature map to a 1-dim tensor so we can add fully connected layers

x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units

x = layers.Dense(512, activation='relu')(x)

# Create output layer with a single node and sigmoid activation

output = layers.Dense(1, activation='sigmoid')(x)

# Create model:

# input = input feature map

# output = input feature map + stacked convolution/maxpooling layers + fully

# connected layer + sigmoid output layer

model = Model(img\_input, output)

from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary\_crossentropy',

              optimizer=RMSprop(lr=0.001),

              metrics=['acc'])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

train\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(

        train\_dir,  # This is the source directory for training images

        target\_size=(150, 150),  # All images will be resized to 150x150

        batch\_size=20,

        # Since we use binary\_crossentropy loss, we need binary labels

        class\_mode='binary')

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = test\_datagen.flow\_from\_directory(

        validation\_dir,

        target\_size=(150, 150),

        batch\_size=20,

        class\_mode='binary')

history = model.fit\_generator(

      train\_generator,

      steps\_per\_epoch=100,  # 2000 images = batch\_size \* steps

      epochs=15,

      validation\_data=validation\_generator,

      validation\_steps=50,  # 1000 images = batch\_size \* steps

      verbose=2)

import numpy as np

import random

from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

# Let's define a new Model that will take an image as input, and will output

# intermediate representations for all layers in the previous model after

# the first.

successive\_outputs = [layer.output for layer in model.layers[1:]]

visualization\_model = Model(img\_input, successive\_outputs)

# Let's prepare a random input image of a cat or dog from the training set.

cat\_img\_files = [os.path.join(train\_cats\_dir, f) for f in train\_cat\_fnames]

dog\_img\_files = [os.path.join(train\_dogs\_dir, f) for f in train\_dog\_fnames]

img\_path = random.choice(cat\_img\_files + dog\_img\_files)

img = load\_img(img\_path, target\_size=(150, 150))  # this is a PIL image

x = img\_to\_array(img)  # Numpy array with shape (150, 150, 3)

x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255

x /= 255

# Let's run our image through our network, thus obtaining all

# intermediate representations for this image.

successive\_feature\_maps = visualization\_model.predict(x)

# These are the names of the layers, so can have them as part of our plot

layer\_names = [layer.name for layer in model.layers]

# Now let's display our representations

for layer\_name, feature\_map in zip(layer\_names, successive\_feature\_maps):

  if len(feature\_map.shape) == 4:

    # Just do this for the conv / maxpool layers, not the fully-connected layers

    n\_features = feature\_map.shape[-1]  # number of features in feature map

    # The feature map has shape (1, size, size, n\_features)

    size = feature\_map.shape[1]

    # We will tile our images in this matrix

    display\_grid = np.zeros((size, size \* n\_features))

    for i in range(n\_features):

      # Postprocess the feature to make it visually palatable

      x = feature\_map[0, :, :, i]

      x -= x.mean()

      x /= x.std()

      x \*= 64

      x += 128

      x = np.clip(x, 0, 255).astype('uint8')

      # We'll tile each filter into this big horizontal grid

      display\_grid[:, i \* size : (i + 1) \* size] = x

    # Display the grid

    scale = 20. / n\_features

    plt.figure(figsize=(scale \* n\_features, scale))

    plt.title(layer\_name)

    plt.grid(False)

    plt.imshow(display\_grid, aspect='auto', cmap='viridis')

# Retrieve a list of accuracy results on training and test data

# sets for each training epoch

acc = history.history['acc']

val\_acc = history.history['val\_acc']

# Retrieve a list of list results on training and test data

# sets for each training epoch

loss = history.history['loss']

val\_loss = history.history['val\_loss']

# Get number of epochs

epochs = range(len(acc))

# Plot training and validation accuracy per epoch

plt.plot(epochs, acc)

plt.plot(epochs, val\_acc)

plt.title('Training and validation accuracy')

plt.figure()

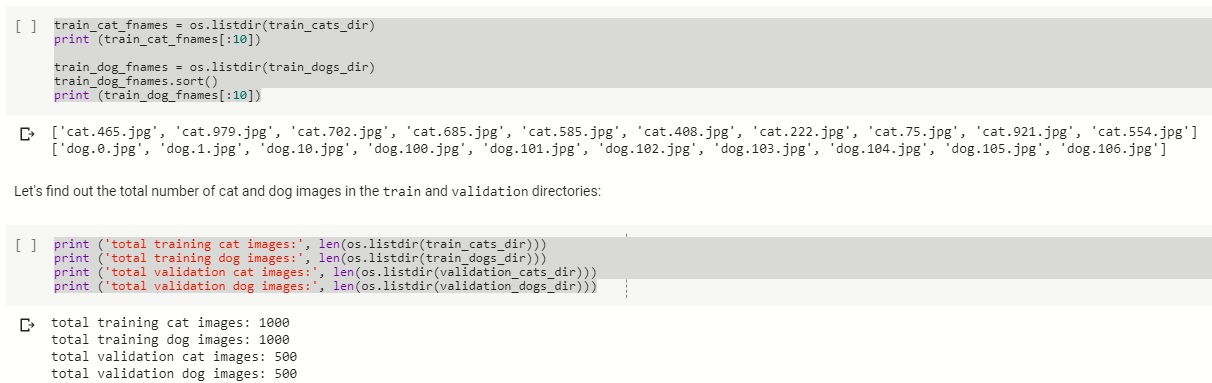
# Plot training and validation loss per epoch

plt.plot(epochs, loss)

plt.plot(epochs, val\_loss)

plt.title('Training and validation loss')

**Output**



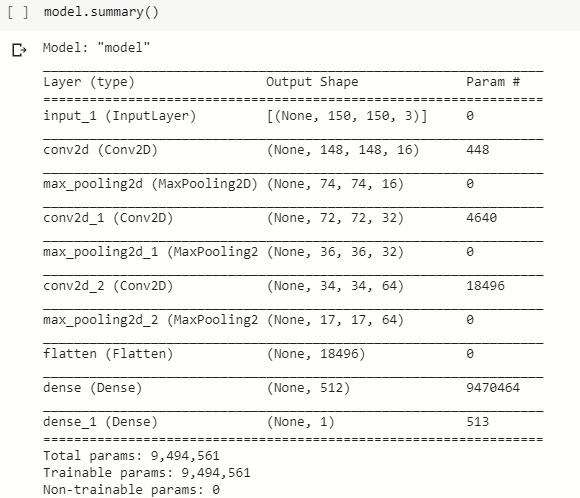
The snippet given above is the no of cat and dog images in jpg format and given below it is the no of training images and validation images of cats and dogs which are to be trained and validated further by CNN which we would be going to work upon.



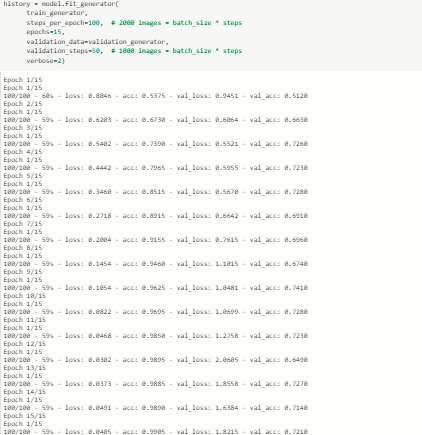
These are 8 cat images which we have extracted from the zip folder and we have actually obtained these images with the help of Matplotlib. Given below are the images of dogs which we have extracted along with these images as well.



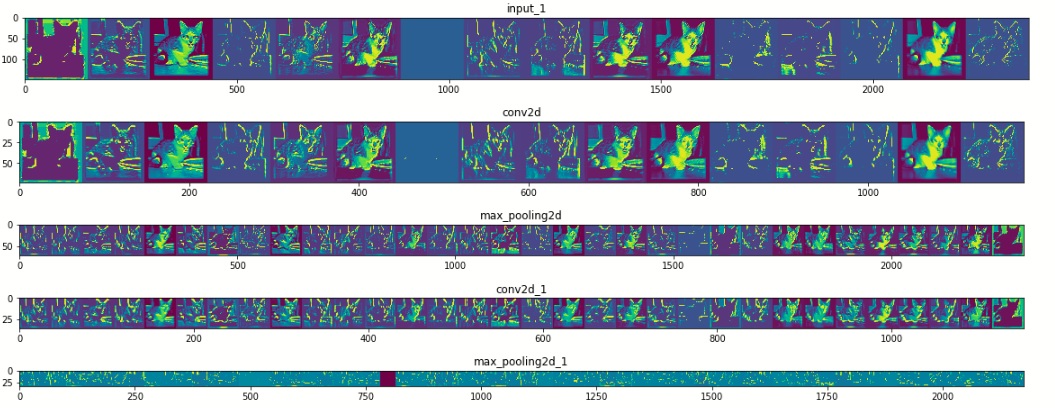
The summary of our model is given as follows as we have built a CNN model

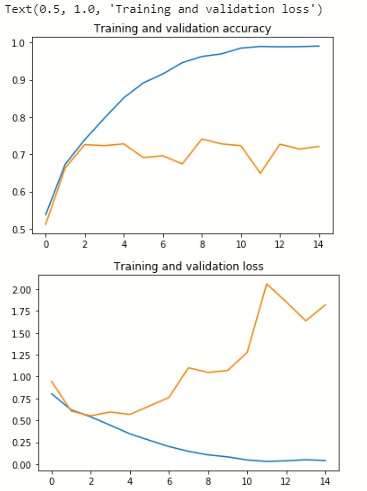


After we have done pre-processing of our dataset, the following output shows the training process of our dataset



The following given is the visualisation of our model and visualisation of accuracy and loss for our model





**Schedule**

**Study Period**

**Duration 1 weeks**

**Requirement gathering**

**Duration 1 week**

**Pseudo code**

**Duration 1 week**

**Design**

**Duration 2 week**

**Coding and implementation**

**Duration 4 weeks**

**Debugging**

**Duration 1 week**

**Testing**

**Duration 2 week**

**Publish Report**

**Duration 1 week**

**References**

*[1]R. Priyatharshini and S. Chitrakala, "Association based image retrieval: A survey, Springer-Verlag Berlin Heidelberge, pp. 17-26, 2013.*

*[2]Y. Rui and T. S. Huang, “Image retrieval: Current techniques, promising directions and open issues," Journal of Visual Communication and Image Representation, vol. 10, pp. 39{62, 1999.*

*[3] A. Ponomarev et al., "Content-Based Image Retrieval Using Color, Texture and Shape Features", Key Engineering Materials, Vol. 685, pp. 872-876, 2016*

*[4, 1]Yushi Jing, David Liu , Dmitry Kislyuk , Andrew Zhai , Jiajing Xu , Jeff Donahue, Sarah Tavel “Visual Search at Pinterest “ 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015*

*[5] U.S.N. Raju, Shibin George, V. Sairam Praneeth, Ranjeet Deo,Priyanka Jain Department of Computer Science and Engineering, National Institute of Technology, Warangal, India -“Content Based Image Retrieval on Hadoop Framework”*

*[6] Nima Razavi, Juergen Gall and Luc Van Gool, “Scalable Multi-class Object Detection”, IEEE Conference on Computer Vision and Pattern Recognition, pp. 1505- 1512, 2011.*

*[7]Swati V. Sakhare & Vrushali G. Nasre,” Design of Feature Extraction in Content Based Image Retrieval (CBIR) using Color and Texture.*

**Synopsis Draft verified by**

**Project Guide HOD**

**(Name & Sign) (Dept. of Informatics)**