Image Key-Point Detection & Tagging

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science and Engineering

Submitted By:

N.Rahul (14071A05L9)

Reetish Chand (14071A05M7)

B. Amar Koni(14075A05I4)

A. Anoop Reddy (14071A05I5)

Under the Guidance of:

Dr. C.Kiranmai

Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS & ACCREDITED BY NAAC WITH "A" GRADE)

VIGNANA JYOTHI NAGAR, BACHUPALLY, NIZAMPET (SO) HYDERABAD – 500 090

April 2018.

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS & ACCREDITED BY NAAC WITH "A" GRADE)
VIGNANA JYOTHI NAGAR, BACHUPALLY, NIZAMPET (SO) HYDERABAD – 500 090

CERTIFICATE

This is to certify that Mr. G.P. Reetish Chand (14071A05M7), Mr.A.Anoop Reddy (14071A05I5) , Mr. N. Rahul (14071A05L9) and Mr. B. Amar Koni (14075A05I4) have successfully completed their major project work at CSE Department of VNR VJIET, Hyderabad entitled **"Image Key-Point Detection & Tagging"** in partial fulfilment of the requirements for the award of B.Tech degree during the academic year 2017-2018.

This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

Dr. C. Kiranmai
Professor
DEPT. CSE
VNRVJIET

Mrs. B. V. Kiranmayee
Associate Professor and Head
DEPT. CSE
VNRVJIET

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS & ACCREDITED BY NAAC WITH "A" GRADE)
VIGNANA JYOTHI NAGAR, BACHUPALLY, NIZAMPET (SO) HYDERABAD – 500 090

DECLARATION BY THE CANDIDATE

We hereby declare that the major project entitled **"Image Key-Point Detection & Tagging"** submitted in partial fulfilment of the requirements for award of the degree of Bachelor of Technology in Computer Science and Engineering at **VNR Vignana Jyothi Institute of Engineering and Technology,** An Autonomous Institute, Hyderabad. This is a bona fide record carried out by us. This is an authentic work and has not been submitted to any other University/Institute for award of any degree/diploma.

N. Rahul	**Reetish Chand**	**B. Amar Koni**	**A. Anoop Reddy**
14071A05L9	**14071A05M7**	**14071A05I4**	**14071A05I5**

ACKNOWLEDGEMENT

Over a span of three and a half years, VNRVJIET has helped us transform ourselves from mere amateurs in the field of computer science into skilled Engineers capable of handling every given situation in real life.

We are highly indebted to the Institute for everything that it has given us. We would like to express our gratitude to our Principal, **Dr. Challa Dhanunjaya Naidu** and the Head of the Computer Science & Engineering Department, Mrs. **B. V. Kiranmayee** for their kind co-operation and encouragement which helped us complete the project in the stipulated time.

Although we have spent a lot of time and put in a lot effort into this project, it would not have been possible without the motivating support and help of our project guide **Dr. C. Kiranmai.** We thank her for her guidance, constant supervision and for providing necessary information to complete this project.

Our thanks and appreciations also go to all the faculty members, staff members of VNRVJIET, to our parents who have supported and helped us at every stage of our lives and all our friends who have helped us put this project together.

N. Rahul (14071A05L9)
Reetish Chand (14071A05M7)
Amar Koni (14071A05I4)
A. Anoop Reddy (14071A05I5)

ABSTRACT

Image tagging and key point detection finds applications in multiple domains such as security, medicine, social media, search engines etc. We are restricting images to faces of people in our application. Any image tagging & key point detection methodology involves face detection, key feature extraction and feature comparison to tag the correct person. Our project proposes usage of the state of the art method of using a combination of Histogram of Oriented Gradients (used to detect faces) and a deep Convolutional Neural Network architecture named Inception that improved the utilization of computing resources over pre-existing neural network models. This was achieved by a carefully crafted design that allows for increasing the depth and width of the network while keeping the computational budget constant. Then it uses K- Nearest Neighbours mechanism which uses a “ball tree” algorithm to tag respective person as the features of a person remain similar to a large extent. We also use few shearing and rotation operations to augment face detection when they are oriented in any angle. We do so by making use of Face Landmark Estimation method and affine transformations.

INDEX

Contents Page. No

CHAPTER 5

CHAPTER 6

Contents Page. No

LIST OF FIGURES

1. Introduction

In the last few years, technology has revolutionized every aspect of our lives. Major breakthroughs were made in multiple domains. One such innovation is Computer vision. The field of computer vision started in the early 1950's. The first major step forward for this domain was by a psychologist and computer vision pioneer Frank Rosenblatt when he invented a '"perceptron machine", created to study the functionality of neuron cells. It was able to differentiate between simple shapes like triangles and squares. It was a rudimentary example of the artificial neural networks that we know today. The next milestone to this journey of development was a project at MIT where Marvin Minsky, who was a professor assigned one of his students to connect a camera to a computer system and make it explain what the camera sees. Though the project was not a complete success, the system was still able to predict edges in the image. The next advancement which we all know today as Optical Character Recognition, was brought about in 1970's when Kurzweil Computer Products released a system that helps blind people to read via OCR computer programs. The field took a quantum leap in 1980's when artificial neural networks were sophisticated by the introduction of multiple hidden layers. And it was in 1990's that statistical methods were delegated the job of recognizing faces in an image.

The field of computer vision as we know today is completely different from then. This is due to the integration of Computer graphics and Computer vision. 21st century has seen enormous growth in two fronts - research and applications. Some of the commercial applications of it are - face detection and tagging(Facebook), image retrieval(Google images), 2D/3D Mapping, Photogrammetry(Apple), stereo vision (Microsoft Kinect), live surveillance, biometrics etc. We are going to majorly concentrate on face detection and tagging.

Any Image(faces of people in our example) detection algorithm can be divided into three sub processes namely, image detection(localizing a face), feature extraction(retrieval of rich features from the face) and finally tagging(finding a person whose features are the most like the current one). We are going to use the state of the

art method of face recognition process that involves a composition of Histogram of Oriented Gradients(HOG's) and Deep Learning(Convolutional neural networks).

The process also involves Face Landmark estimation and a few geometric transformations based on it. HOG's are found to be more efficient than the pre-existing Weak classifier cascade method, also nicknamed Viola-Jones algorithm, when it comes to image detection, in that, HOG was immune to object rotations and illumination conditions which the latter failed to deal with. For the feature extraction part, we made use of a deep convolutional neural network architecture codenamed Inception, which was responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). For the tagging part we used K- Nearest Neighbours algorithm simply because it reduces the training cost and still provides good results.

2. Existing System & Proposed System

Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to run on cheap cameras.

2.1 Viola Jones Algorithm – Existing System

The Viola–Jones object detection framework is the first object detection framework to provide competitive object detection rates in real-time proposed in 2001 by Paul Viola and Michael Jones. Although it can be trained to detect a variety of object classes, it was motivated primarily by the problem of face detection. The problem to be solved is detection of faces in an image. A human can do this easily, but a computer needs precise instructions and constraints. To make the task more manageable, Viola–Jones requires full view frontal upright faces. This in order to be detected, the entire face must point towards the camera and should not be tilted to either side. While it seems like these constraints could diminish the algorithm's utility somewhat, because the detection step is most often followed by a recognition step, in practice these limits on pose are quite acceptable. The Viola-Jones algorithm is a widely used mechanism for object detection. The main property of this algorithm is that training is slow, but detection is fast. This algorithm uses Haar basis feature filters, so it does not use multiplications.
The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
- Real time – For practical applications at least 2 frames per second must be processed.
- Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

The algorithm has four stages:

1. Haar Feature Selection
2. Creating an Integral Image
3. Adaboost Training
4. Cascading Classifiers

The features sought by the detection framework universally involve the sums of image pixels within rectangular areas. As such, they bear some resemblance to Haar basis functions, which have been used previously in the realm of image-based object detection. However, since the features used by Viola and Jones all rely on more than one rectangular area, they are generally more complex. The figure on the right illustrates the four different types of features used in the framework. The value of any given feature is the sum of the pixels within clear rectangles subtracted from the sum of the pixels within shaded rectangles. Rectangular features of this sort are primitive when compared to alternatives such as steerable filters. Although they are sensitive to vertical and horizontal features, their feedback is considerably coarser.

Features:

The simple features used are reminiscent of Haar basis functions:

• two-rectangle feature: difference between the sum of the pixels within two rectangular regions

• three-rectangle feature: sum within two outside rectangles subtracted from the sum in a centre rectangle

• four-rectangle feature: difference between diagonal pairs of rectangles.

The efficiency of the Viola-Jones algorithm can be significantly increased by first generating the integral image.

$$II(y,x) = \sum_{p=0}^{y} \square \sum_{q=0}^{x} Y(p,q)$$

Drawbacks:

- Viola Jones algorithm was a break through which was fast enough to detect faces in real time and this was embedded into point and shoot cameras.
- The algorithm looks for vertical bright bands(Noses), then horizontal dark bands(Eyes) and other general properties of a face.
- The application of the Viola Jones algorithm was limited to face detection only and not recognition. Another disadvantage being the algorithm couldn't accurately detect faces from other angles.
- Metadata is hugely relied upon when doing image search/retrieval in the existing systems.

2.2. Proposed System

CNN is indeed a faster and more accurate method of deep learning, when applied to facial key point recognition. It gives results far too better than simple networks. Adjusting filter sizes, keeping larger filters for data input layers and decreasing the size in subsequent layers, produces better results. With further tuning of networks and large data size, the network is bound to improve.

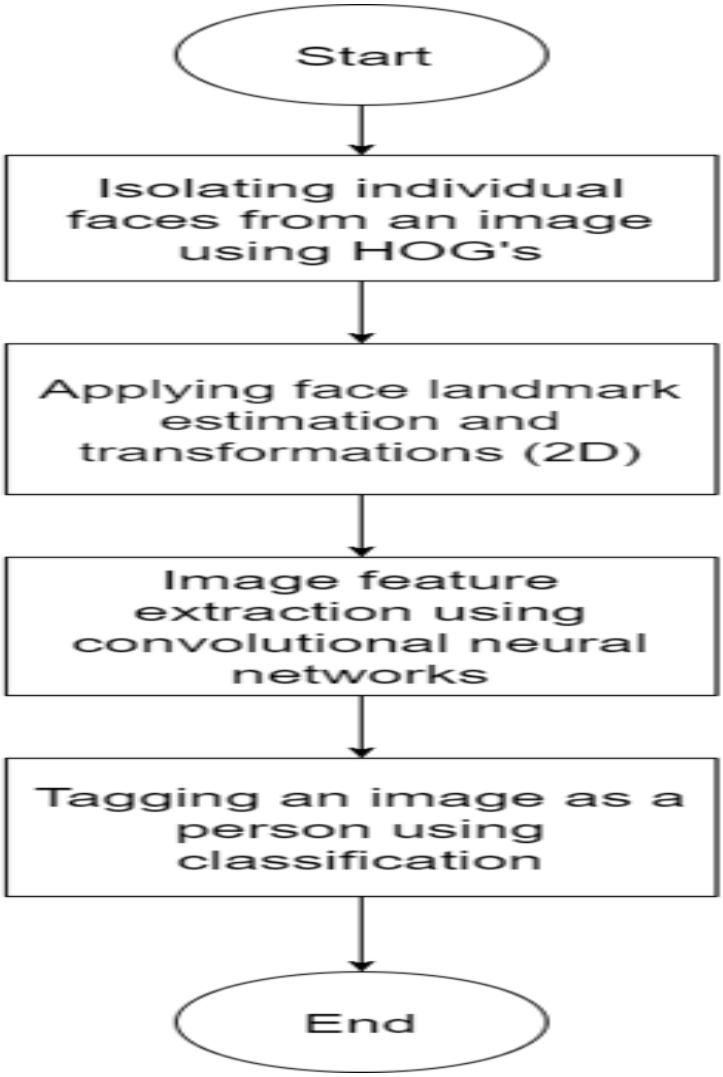


Figure 2.1: Flowchart of the proposed system

The proposed system presents a unified front for face verification (*is this the same person*), recognition (*who is this person*) and clustering (*find common people among these faces*). This method is based on learning a Euclidean embedding per image using a deep convolutional network. The network is trained such that the squared L2 distances in the embedding space directly correspond to face similarity: faces of the same person have small distances and faces of distinct people have large distances.

Once this embedding has been produced, then the above-mentioned tasks become straight-forward: face verification involves setting a threshold for the distances between the two Euclidean embeddings while recognition becomes a k-NN classification problem.

Previous face recognition approaches based on deep networks use a classification layer trained over a set of known face identities and then take an intermediate bottle-neck layer as a representation used to generalize recognition beyond the set of identities used in training. The downsides of this approach are its indirectness and its inefficiency: one has to hope that the bottleneck representation generalizes well to new faces; and by using a bottleneck layer the representation size per face is usually very large since the bottleneck has more number of nodes compared to the final output layer.

In contrast to these approaches, the output of the proposed system will be a compact 128-D embedding which will be produced after training using a triplet-based loss function based on LMNN (large margin nearest neighbour). The triplets consist of two matching face thumbnails and a non-matching face thumbnail and the loss aims to separate the positive pair from the negative by a distance margin. The thumbnails are tight crops of the face area, no 2D or 3D alignment, other than scale and translation has been performed. (DLIB is used for scaling and transformation and Keras for constructing/training the model)

The architecture of the proposed system is based on the Inception model of Szegedy et al. which was used as the winning approach for ImageNet 2014. These networks use mixed layers that run several different convolutional and pooling layers in parallel and concatenate their responses.

Advantages:

- Current methods being used in opencv are Fisherfaces & Eignefaces. PCA gives you the Eigenfaces algorithm while LDA gives you Fisherfaces (both are in OpenCV, hence the claim that they are widely used). Both fail badly when one trains using faces in one scene and try to recognize the same person in other images. But that is a well-known hard problem for face recognition: recognition under illumination, pose, angle, facial expression etc. variations. Eigenfaces and Fisherfaces work well only under controlled illumination, pose, etc. conditions, because they depend heavily on an almost pixel-by-pixel correspondence

between the face images to be matched.

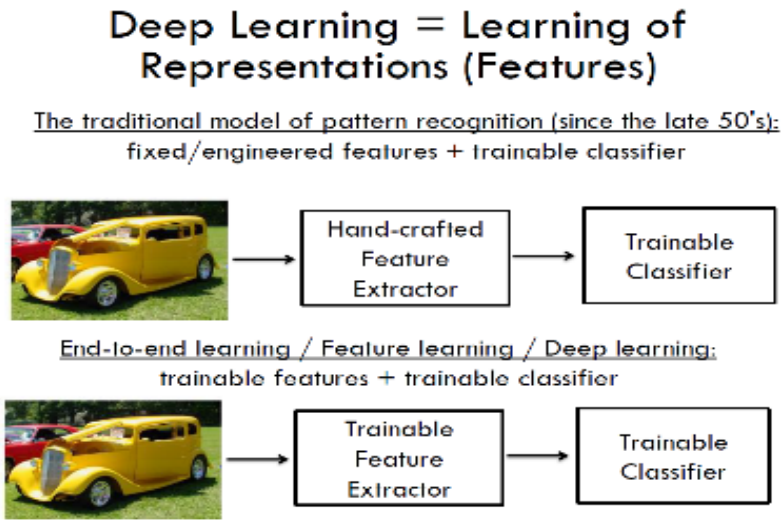


Figure 2.2: The differences between the methods of pattern recognition and classification in 50's and now

- Sensitivity to Image Quality(Less Sensitive)

The model is robust across varying levels of JPEG compression.

3. Feasibility Study

A feasibility study involves taking a judgment call on whether a project is doable. The two criteria to judge feasibility are cost required and value to be delivered. A well-designed study should offer a historical background of the business or project, a description of the product or service, accounting statements, details of operations and management, marketing research and policies, financial data, legal requirements and tax obligations. Generally, such studies precede technical development and project implementation. A feasibility study evaluates the project's potential for success; therefore, perceived objectivity is an important factor in the credibility of the study for potential investors and lending institutions.

Preliminary investigation examines project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operation Feasibility
- Economic Feasibility
- Scheduling Feasibility

3.1 Technical Feasibility:

There is no interactive GUI (*graphical user interface*) for this proposed system, the system can be interacted using CLI (*command line interface*) with a python shell. The libraries used in this project are Dlib, Keras, sklearn, PIL, sciPy, numpy which are open sourced and being constantly revised by contribution from a large community of enthusiastic developers/researchers.

3.2 Operational Feasibility:

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. The application involves design-dependent parameters such as reliability, maintainability, supportability, usability, disposability, sustainability, affordability, and others. It minimises the drawbacks of the current system by building an application that involves less amount training data and is less sensitive to wide range of image qualities comparatively. It is a clear showcase of the incredible invariance to occlusion, lighting, pose etc.

3.3 Economic Feasibility:

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economic feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible because the hardware used to train the model is required only once. Since the interface for this system is developed using the existing resources and technologies available, there is nominal expenditure and economic feasibility for certain. To make this project more economical, the training data set and the model can be hosted using a server and a database so that prediction is available at any place and any time without using a separate hardware. This change can make the system more economically feasible.

3.4 Scheduling Feasibility:

The project development took place in timely process by understanding time schedules of the project and maintaining timelines for project development.

4.System Analysis

4.1 Python :

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is **Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is **Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is **Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games. Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Features :

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read − Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain − Python's source code is fairly easy-to-maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases − Python provides interfaces to all major commercial databases.
- GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable − Python provides a better structure and support for large programs than shell scripting.
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is available on a wide variety of platforms including Linux, Mac OS X, Windows etc.

4.2 Python libraries :

4.2.1 Dlib:

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high-performance computing environments. Dlib's is open source toolkit thus free of any charge.

Overview:

Classes:

- *class* dlib.fhog_object_detector

This object represents a sliding window histogram-of-oriented-gradients based object detector.

run(*self: dlib.fhog_object_detector*, *image: object*, *upsample_num_times: int=0L*, *adjust_threshold: float=0.0*) → tuple

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- upsample_num_times >= 0

- This function runs the object detector on the input image and returns a tuple of (list of detections, list of scores, list of weight_indices).
- Upsamples the image upsample_num_times before running the basic detector.

save(*self: dlib.fhog_object_detector*, *detector_output_filename: unicode*) → None

Save a simple_object_detector to the provided path

- *class* dlib.shape_predictor

This object is a tool that takes in an image region containing some object and outputs a set of point locations that define the pose of the object. The classic example of this is human face pose prediction, where you take an image of a human face as input and are expected to identify the locations of important facial landmarks such as the corners of the mouth and eyes, tip of the nose, and so forth.

save(*self: dlib.shape_predictor*, *predictor_output_filename: unicode*) → None

Save a shape_predictor to the provided path.

- *class* dlib.cnn_face_detection_model_v1

This object detects human faces in an image. The constructor loads the face detection model from a file.

- *class* dlib.face_recognition_model_v1

This object maps human faces into 128D vectors where pictures of the same person are mapped near to each other and pictures of different people are mapped far apart. The constructor loads the face recognition model from a file.

4.2.2 PIL (Python Imaging Library)

Overview:

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Use case of PIL:

- Image Archives: PIL is ideal for image archival and batch processing applications. You can use the library to create thumbnails, convert between file formats, print images, etc.
- Image Display: For debugging, there's also a show() method which saves an image to disk and calls an external display utility.

- Image Processing: The library contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and colour space conversions. The library also supports image resizing, rotation and arbitrary affine transforms.
- The Python Imaging Library handles raster images; that is, rectangles of pixel data.

Bands

- An image can consist of one or more bands of data. The Python Imaging Library allows you to store several bands in a single image, provided they all have the same dimensions and depth. For example, a PNG image might have 'R', 'G', 'B', and 'A' bands for the red, green, blue, and alpha transparency values. Many operations act on each band separately, e.g., histograms. It is often useful to think of each pixel as having one value per band.
- To get the number and names of bands in an image, use the get bands() method.

Modes

The mode of an image defines the type and depth of a pixel in the image. The current release supports the following standard modes:

- 1 (1-bit pixels, black and white, stored with one pixel per byte)
- L (8-bit pixels, black and white)
- P (8-bit pixels, mapped to any other mode using a colour palette)
- RGB (3x8-bit pixels, true colour)
- RGBA (4x8-bit pixels, true colour with transparency mask)
- CMYK (4x8-bit pixels, colour separation)
- YCbCr (3x8-bit pixels, colour video format)
- Note that this refers to the JPEG, and not the ITU-R BT.2020, standard
- LAB (3x8-bit pixels, the L*a*b colour space)
- HSV (3x8-bit pixels, Hue, Saturation, Value colour space)
- I (32-bit signed integer pixels)
- F (32-bit floating point pixels)

Using the Image class : The most important class in the Python Imaging Library is the Image class, defined in the module with the same name. Instances of this class can be created by either loading images from files, processing other images, or creating images from scratch.

To load an image from a file, use the open() function in the Image module:

```
>>> from PIL import Image
>>> im = Image.open("hopper.ppm")
```

If the file cannot be opened, an IOError exception is raised. Once you have an instance of the Image class, you can use the methods defined by this class to process and manipulate the image. For example, let's display the image we just loaded:

```
>>> im.show()
```

The Python Imaging Library supports a wide variety of image file formats. To read files from disk, use the open() function in the Image module. The library automatically determines the format based on the contents of the file.
To save a file, use the save() method of the Image class. When saving files, the name becomes important. Unless you specify the format, the library uses the filename extension to discover which file storage format to use.

4.2.3 numPy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Installation:

- Mac and Linux users can install NumPy via pip command:
- pip install numpy
- Windows does not have any package manager analogous to that in linux or mac.
- And then install the packages manually.

Note: All the examples discussed below will not run on an online IDE.

1. Arrays in NumPy: NumPy's main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called *axes*. The number of axes is *rank*.
- NumPy's array class is called ndarray. It is also known by the alias array.

2. Array creation: There are various ways to create arrays in NumPy.

- For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation.
- For example: np.zeros, np.ones, np.full, np.empty, etc.
- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
- arange: returns evenly spaced values within a given interval. step size is specified.
- linspace: returns evenly spaced values within a given interval. num no. of elements are returned.
- Reshaping array: We can use reshape method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is:

- a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
- Flatten array: We can use flatten method to get a copy of array collapsed into one dimension. It accepts *order* argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

3. Array Indexing: Knowing the basics of array indexing is important for analysing and manipulating the array object. NumPy offers many ways to do array indexing.

- Slicing: Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- Integer array indexing: In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.

- Boolean array indexing: This method is used when we want to pick elements from array which satisfy some condition.

4.2.4 sciPy:

Python, a general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.

NumPy, the fundamental package for numerical computation. It defines the numerical array and matrix types and basic operations on them.

The SciPy library, a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics and much more.

Matplotlib, a mature and popular plotting package, that provides publication-quality 2D plotting as well as rudimentary 3D plotting

On this base, the SciPy ecosystem includes general and specialised tools for data management and computation, productive experimentation and high-

performance computing. Below we overview some key packages, though there are many more relevant packages.

Data and computation tools scipy can be used:

pandas, providing high-performance, easy to use data structures. scikit-image is a collection of algorithms for image processing.scikit-learn is a collection of algorithms and tools for machine learning. h5py and PyTables can both access data stored in the HDF5 format.

4.2.5 Math:

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

Some of the functions provided by this module are mentioned below, except when explicitly noted otherwise, all return values are floats.

Number-theoretic and representation functions :

math.**ceil**(*x*)
Return the ceiling of *x* as a float, the smallest integer value greater than or equal to *x*.

math.**copysign**(*x*, *y*)

Return *x* with the sign of *y*. On a platform that supports signed zeros, copysign(1.0, -0.0) returns *-1.0*.

math.**fabs**(*x*)
Return the absolute value of *x*.

math.**factorial**(*x*)
Return *x* factorial. Raises ValueError if *x* is not integral or is negative.

Power and logarithmic functions :

math.**exp**(*x*)

Return e**x.

math.**log**(*x*[, *base*])

With one argument, return the natural logarithm of *x*(to base *e*).

Trigonometric functions :

math.**acos**(*x*)

Return the arc cosine of *x*, in radians.

math.**asin**(*x*)

Return the arc sine of *x*, in radians.

math.**atan**(*x*)

Return the arc tangent of *x*, in radians.

Angular conversion :

math.**degrees**(*x*)

Convert angle *x* from radians to degrees.

math.**radians**(*x*)

Convert angle *x* from degrees to radians.

4.2.6 sklearn:

Scikit-learn requires:

- Python (>= 2.7 or >= 3.3),
- NumPy (>= 1.8.2),

- SciPy (>= 0.13.3).

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features. We can separate learning problems in a few large categories:

- supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:
    - classification: samples belong to two or more classes and we want to learn from already labelled data how to predict the class of unlabelled data.
    - regression: if the desired output consists of one or more continuous variables, then the task is called *regression*. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

- unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization*.

Training set and testing set

Machine learning is about learning some properties of a data set and applying them to new data. This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the training set on which we learn data properties and one that we call the testing set on which we test these properties.

Loading an example dataset: scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the Boston house prices dataset for regression.It is possible to save a model in the scikit by using Python's built-in persistence model, namely pickle:

```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC()
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0:1])
array([0])
>>> y[0]
0
```

4.3 HISTOGRAM OF ORIENTED GRADIENTS

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. Robert K. McConnell of Wayland Research Inc. first described the concepts behind HOG without using the term HOG in a patent application in 1986. In 1994 the concepts were used by Mitsubishi Electric Research Laboratories. However, usage only became widespread in 2005 when Navneet Dalal and Bill Triggs, researchers for the French National Institute for Research in Computer Science and Automation (INRIA), presented their supplementary work on HOG descriptors at the Conference on Computer Vision and Pattern Recognition (CVPR). In this work they focused on pedestrian detection in static images, although since then they expanded their tests to include human detection in videos, as well as to a variety of common animals and vehicles in static imagery.The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for human detection in images.

Implementation of the HOG descriptor algorithm is as follows:

- Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.
- Discretize each cell into angular bins according to the gradient orientation.
- Each cell's pixel contributes weighted gradient to its corresponding angular bin.

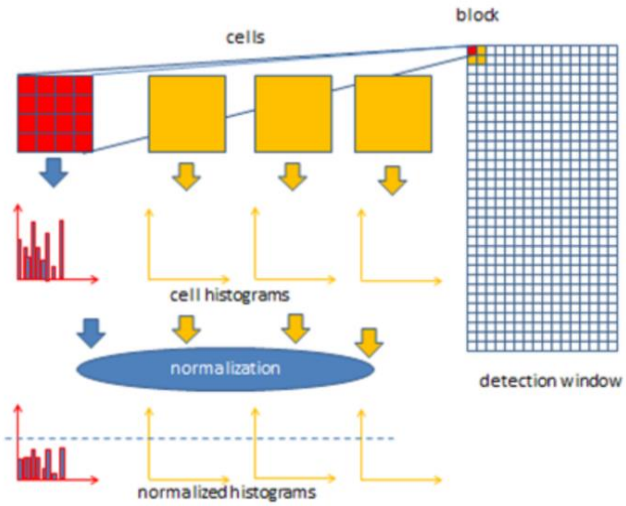


Figure 4.1: Demonstration of Histogram of oriented gradients

- Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.
- Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

Computation of the HOG descriptor requires the following basic configuration parameters:

- Masks to compute derivatives and gradients
- Geometry of splitting an image into cells and grouping cells into a block
- Block overlapping
- Normalization parameters

The recommended values for the HOG parameters are:

- 1D centred derivative mask [-1, 0, +1]

- Detection window size is 64x128
- Cell size is 8x8
- Block size is 16x16 (2x2 cells)

PROCESS:

Step 1: Pre-processing

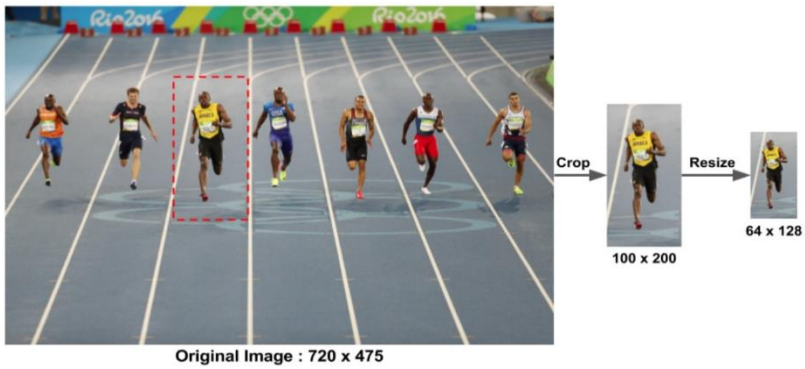


Figure 4.2 Cropping and resizing for fitting window size

Step 2: Calculate the Gradient Images

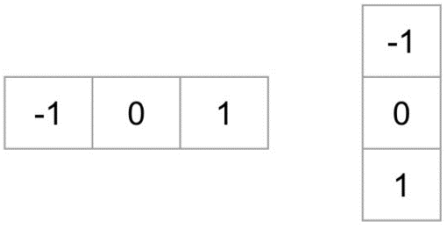


Figure 4.3: An example of an HOG descriptor

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

Step 3: Calculate Histogram of Gradients in 8×8 cells

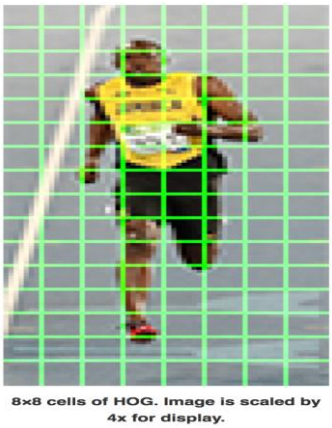


Figure 4.4: Division of image into blocks of 8X8 cells

In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells. An 8×8 image patch contains 8x8x3 = 192 pixel values. The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = 128 numbers. By the end of this section we will see how these 128 numbers are represented using a 9-bin histogram which can be stored as an array of 9 numbers. Not only is the representation more compact, calculating a histogram over a patch makes this representation more robust to noise. Individual gradients may have noise, but a histogram over 8×8 patch makes the representation much less sensitive to noise.

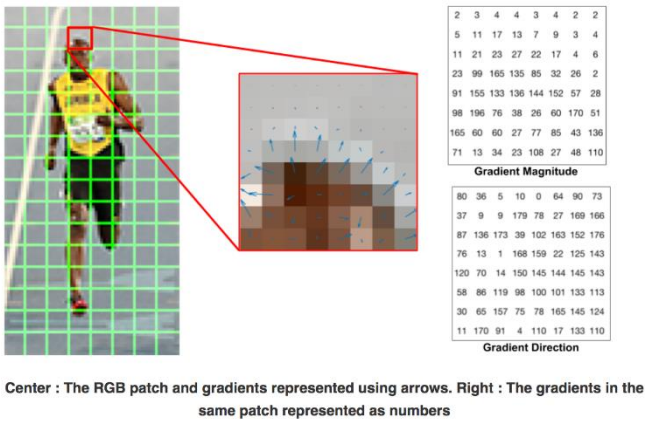


Figure 4.5: Representation of Gradients within a block

The histogram is essentially a vector (or an array) of 9 bins (numbers) corresponding to angles 0, 20, 40, 60 … 160.

Let us look at one 8×8 patch in the image and see how the gradients look.

A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude. Let's first focus on the pixel encircled in blue. It has an angle (direction) of 80 degrees and magnitude of 2. So, it adds 2 to the 5th bin. The gradient at the pixel encircled using red has an angle of 10 degrees and magnitude of 4. Since 10 degrees is half way between 0 and 20, the vote by the pixel splits evenly into the two bins.

If the angle is greater than 160 degrees, it is between 160 and 180, and we know the angle wraps around making 0 and 180 equivalents.

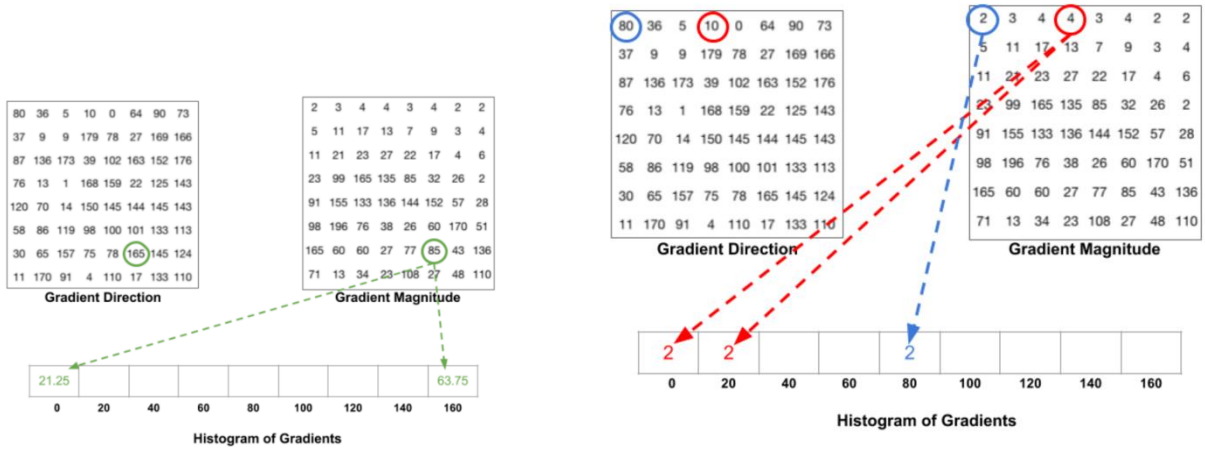


Figure 4.6 & 4.7: Demonstration of the method by which the magnitude is divided into bins with respect to the direction

Step 4: 16×16 Block Normalization

Say we have an RGB colour vector [128, 64, 32]. The length of this vector is $\sqrt{128^2 + 64^2 + 32^2} = 146.64$. This is also called the L2 norm of the vector. Dividing each element of this vector by 146.64 gives us a normalized vector [0.87, 0.43, 0.22]. Now consider another vector in which the elements are twice the value of the first vector 2 x [128, 64, 32] = [256, 128, 64]. You can work it out yourself to see that normalizing [256, 128, 64] will result in [0.87, 0.43, 0.22], which is the same as the normalized version of the original RGB vector. You can see that normalizing a vector removes the scale.

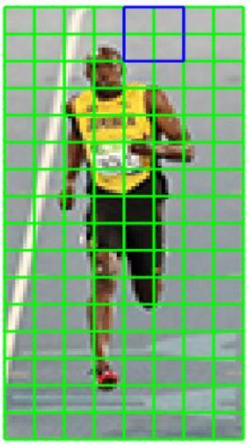
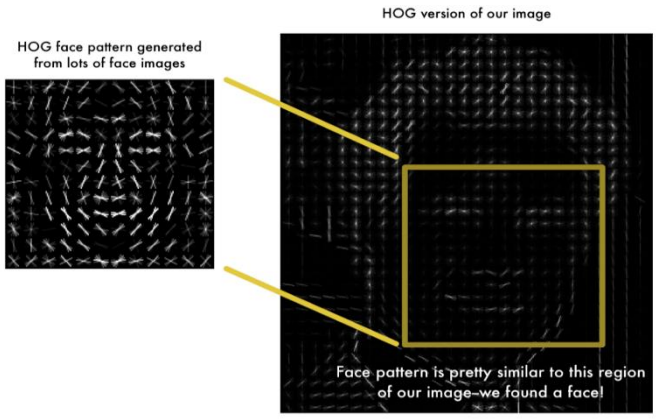


Figure 4.8: 16X16 cell block selected to normalize

Step 5: Calculate the HOG feature vector

To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate

1. How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
2. Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one giant vector we obtain a 36×105 = 3780 dimensional vectors



4.4 Face Landmark Estimation:

Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape. In the context of facial landmarks, our goal is to detect important facial structures on the face using shape prediction methods.

Detecting facial landmarks is therefore a twostep process:

- Step #1: Localize the face in the image.
- Step #2: Detect the key facial structures on the face ROI.

Face detection (Step #1) can be achieved in a number of ways. We could use OpenCV's built-in Haar cascades. We might apply a pre-trained HOG + Linear SVM object

detector specifically for the task of face detection or we might even use deep learning-based algorithms for face localization.

In either case, the actual algorithm used to detect the face in the image doesn't matter. Instead, what's important is that through some method we obtain the face bounding box (i.e., the *(x, y)*-coordinates of the face in the image). Given the face region we can then apply Step #2: detecting key facial structures in the face region.

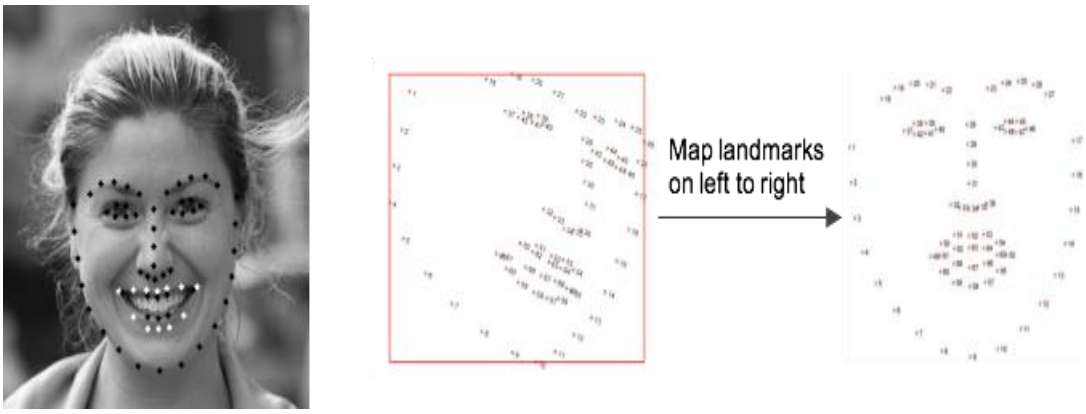


Figure 4.10: Demonstration of how landmarks are plotted against a face and performing affine transformations on them to match a figure taken as basis(right)

There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

- Mouth
- Right eyebrow
- Left eyebrow
- Right eye
- Left eye
- Nose
- Jaw

This method starts by using:

1. A training set of labelled facial landmarks on an image. These images are *manually labelled*, specifying specific *(x, y)*-coordinates of regions surrounding each facial structure.
2. *Priors*, of more specifically, the *probability on distance* between pairs of input pixels.

Given this training data, an ensemble of regression trees is trained to estimate the facial landmark positions directly from the *pixel intensities themselves* (i.e., no "feature extraction" is taking place).

The end result is a facial landmark detector that can be used to detect facial landmarks in *real-time* with high quality predictions.

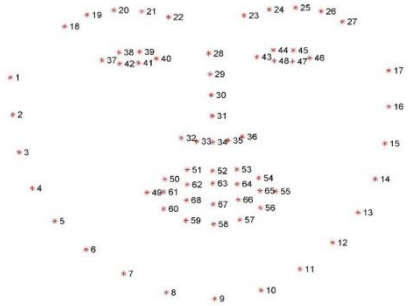


Figure 4.11: The indices of all the 68 landmarks on a face

Understanding dlib's facial landmark detector.

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of *68 (x, y)-coordinates* that map to facial structures on the face. The indexes of the 68 coordinates can be visualized on the image on the next page:

4.5 KNN Classification:

In pattern recognition, the k-nearest neighbour's algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression: In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbour's, with the object being assigned to the class most common among its k nearest neighbour's (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbour's. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, a useful technique can be to assign weight to the contributions of the neighbour's, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of 1/d, where d is the distance to the neighbour.

The neighbours are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm is not to be confused with k-means, another popular machine learning technique.

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output.
2. Calculation time.
3. Predictive Power.

Let us take a few examples to place KNN in the scale:

	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

Figure 4.12: A table comparing multiple classification techniques with respect to performance aspects

KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

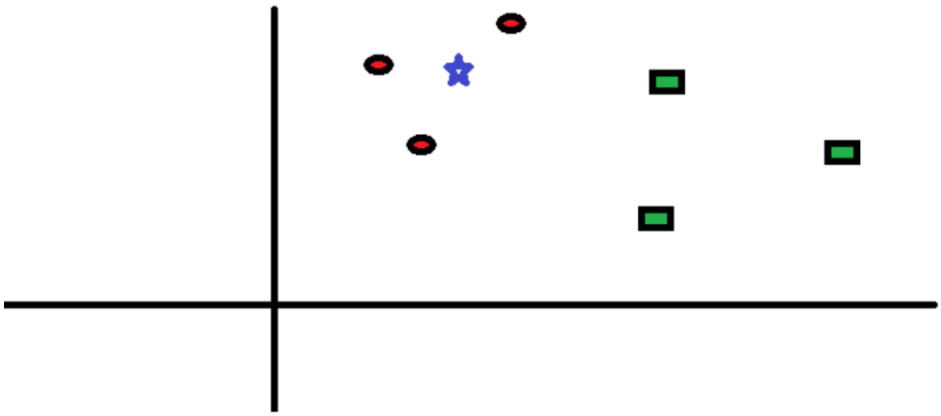


Figure 4.13: A graph showing a classification problem between Red Circles and Green Rectangles and star is the individual to be predicted

How does the KNN algorithm work?

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):

You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The "K" is KNN algorithm is the nearest neighbours we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as centre just as big as to enclose only three data points on the plane.

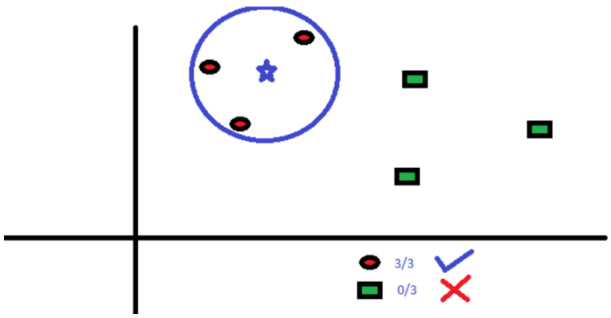
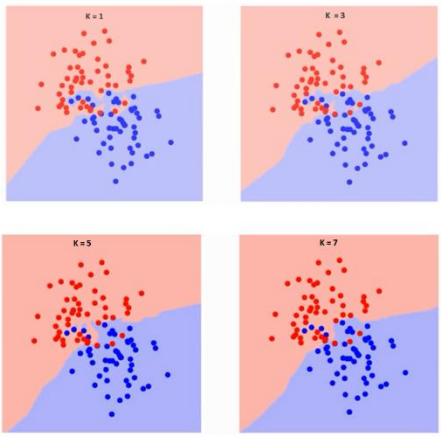


Figure 4.14: Finding the nearest neighbour for the blue star

The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became obvious as all three votes from the closest neighbour went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand what the factors are to be considered to conclude the best K.

How do we choose the factor K?

First let us try to understand what exactly does K influence in the algorithm. If we see the last example, given that all the 6-training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K. If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority. The training error rate and the validation error rate are two parameters we need to access on different K-value. Following is the curve for the training error rate with varying value of K:



Fiure 4.15: The boundary between red and blue with increasing K values - 1,3,5,7

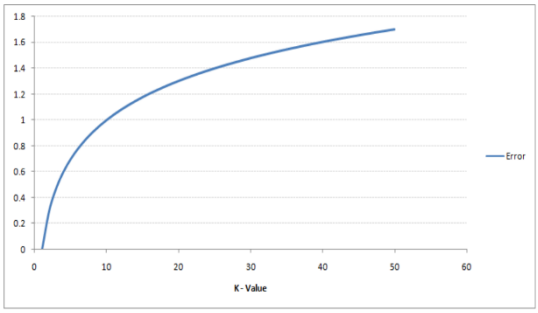


Figure 4.16: Error vs K-value graph

As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself. Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:

This makes the story clearer. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minimal. After the minima point, it then increases with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the

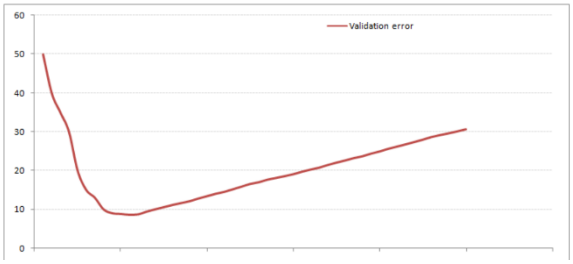


Figure 4.17: Validation error vs K-value graph

validation error curve to get the optimal value of K. This value of K should be used for all predictions.

A few Applications and Examples of KNN:

1. Credit ratings — collecting financial characteristics vs. comparing people with similar financial features to a database. By the very nature of a credit rating, people who have similar financial details would be given similar credit ratings. Therefore, they would like to be able to use this existing database to predict a new customer's credit rating, without having to perform all the calculations.
2. Should the bank give a loan to an individual? Would an individual default on his or her loan? Is that person closer in characteristics to people who defaulted or did not default on their loans?
3. In political science — classing a potential voter to a "will vote" or "will not vote", or to "vote Democrat" or "vote Republican".
4. More advance examples could include handwriting detection (like OCR), image recognition and even video recognition.

Pros:

- No assumptions about data—useful, for example, for nonlinear data
- Simple algorithm—to explain and understand/interpret
- High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
- Versatile—useful for classification or regression.
- Insensitive to outliers—accuracy can be affected from noise or irrelevant features

Cons:

- High memory requirement
- Computationally expensive—because the algorithm stores all of the training data
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

The algorithm can be summarized as:

1. A positive integer k is specified, along with a new sample
2. We select the k entries in our database which are closest to the new sample
3. We find the most common classification of these entries
4. This is the classification we give to the new sample

A few other features of KNN:

- KNN stores the entire training dataset which it uses as its representation.
- KNN does not learn any model.
- KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.

K-Nearest Neighbours for Machine Learning

- The model representation used by KNN.
- How a model is learned using KNN (hint, it's not).
- How to make predictions using KNN
- The many names for KNN including how different fields refer to it.
- How to prepare your data to get the most from KNN.
- Where to look to learn more about the KNN algorithm.

Making Predictions with KNN

KNN makes predictions using the training dataset directly.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbours) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.

Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))

The value for K can be found by algorithm tuning. It is a good idea to try many different values for K (e.g. values from 1 to 21) and see what works best for your problem.

The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.

KNN can be used for regression and classification problems.

KNN for Regression

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

KNN for Classification

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance. For example, in a binary classification problem (class is 0 or 1):

p(class=0) = count(class=0) / (count(class=0)+count(class=1))

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset.

Curse of Dimensionality

KNN works well with a small number of input variables(p), but struggles when the number of inputs is very large. Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables x1 and x2, the input space would be 2-dimensional. As the number of dimensions increases the volume of the input space increases at an exponential rate.

In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the **"Curse of Dimensionality".**

4.6 Ball Tree algorithm:

A ball tree is a binary tree in which every node defines a D-dimensional hypersphere, or ball, containing a subset of the points to be searched. Each internal node of the tree partitions the data points into two disjoint sets which are associated with different balls. While the balls themselves may intersect, each point is assigned to one or the other ball in the partition according to its distance from the ball's centre. Each leaf node in the tree defines a ball and enumerates all data points inside that ball.

Each node in the tree defines the smallest ball that contains all data points in its subtree. This gives rise to the useful property that, for a given test point *t*, the distance to any point in a ball *B* in the tree is greater than or equal to the distance from *t* to the ball.

An important application of ball trees is expediting nearest neighbour search queries, in which the objective is to find the k points in the tree that are closest to a given test point by some distance metric (e.g. Euclidean distance). A simple search algorithm, sometimes called KNS1, exploits the distance property of the ball tree. In particular, if the algorithm is searching the data structure with a test point *t*, and has already seen

some point *p* that is closest to *t* among the points encountered so far, then any subtree whose ball is further from *t* than *p* can be ignored for the rest of the search.

4.6.1 Description

The ball tree nearest-neighbour algorithm examines nodes in depth-first order, starting at the root. During the search, the algorithm maintains a max-first priority queue (often implemented with a heap), denoted *Q* here, of the k nearest points encountered so far. At each node *B*, it may perform one of three operations, before finally returning an updated version of the priority queue:

1. If the distance from the test point *t* to the current node *B* is greater than the furthest point in *Q*, ignore *B* and return *Q*.

2. If *B* is a leaf node, scan through every point enumerated in *B* and update the nearest-neighbour queue appropriately. Return the updated queue.

3. If *B* is an internal node, call the algorithm recursively on *B*'s two children, searching the child whose center is closer to *t* first. Return the queue after each of these calls has updated it in turn.

Performing the recursive search in the order described in point 3 above increases likelihood that the further child will be pruned entirely during the search.

4.6.2 Pseudocode

```
function knn_search is
    input:
        t, the target point for the query
        k, the number of nearest neighbors of t to search for
        Q, max-first priority queue containing at most k points
        B, a node, or ball, in the tree
    output:
        Q, containing the k nearest neighbors from within B
    if distance(t, B.pivot) - B.radius ≥ distance(t, Q.first) then
        return Q unchanged
    else if B is a leaf node then
        for each point p in B do
            if distance(t, p) < distance(t, Q.first) then
```

```
            add p to Q
            if size(Q) > k then
               remove the furthest neighbour from Q
            end if
         end if
      repeat
   else
      let child1 be the child node closest to t
      let child2 be the child node furthest from t
      knn_search(t, k, Q, child1)
      knn_search(t, k, Q, child2)
   end if
end function
```

4.7 Convolutional Neural Networks (CNNs):

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery. CNNs use a variation of multilayer perceptron's designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems and natural language processing.

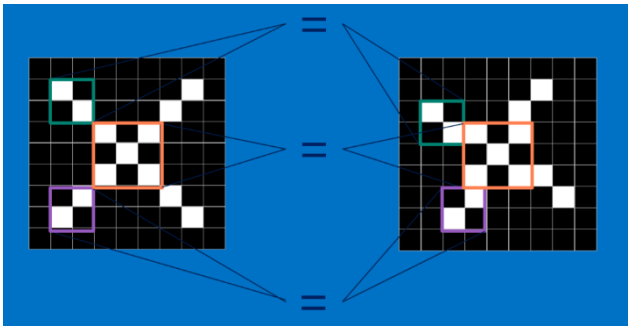


Figure 4.18: CNN's divide an image and then extract features

CNNs compare images piece by piece. The pieces that it looks for are called features. By finding rough feature matches in roughly the same positions in two images, CNNs get a lot better at seeing similarity than whole-image matching schemes. Each feature is like a mini-image—a small two-dimensional array of values. Features match common aspects of the images. In the case of X images, features consisting of diagonal lines and a crossing capture all the important characteristics of most X's. These features will probably match up to the arms and centre of any image of an X.

When presented with a new image, the CNN doesn't know exactly where these features will match so it tries them everywhere, in every possible position. In calculating the match to a feature across the whole image, a filter is needed. The math used to do this is called convolution, from which Convolutional Neural Networks take their name. To calculate the match of a feature to a patch of the image, simply multiply each pixel in the feature by the value of the corresponding pixel in the image.

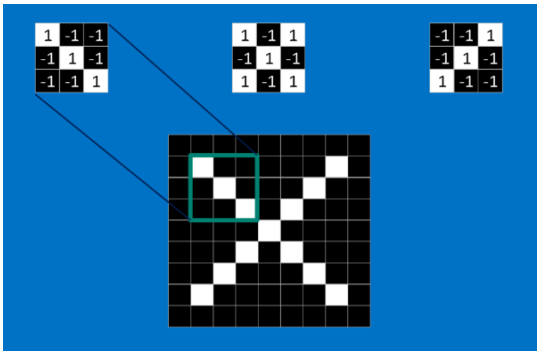


Figure 4.19: Representation of individual pieces of a black and white image

Then add up the answers and divide by the total number of pixels in the feature. If both pixels are white (a value of 1) then 1 * 1 = 1. If both are black, then (-1) * (-1) = 1. Either way, every matching pixel results in a 1. Similarly, any mismatch is -1. If all the pixels in a feature match, then adding them up and dividing by the total number of pixels gives 1. Similarly, if none of the pixels in a feature match the image patch, then the answer is a -1).

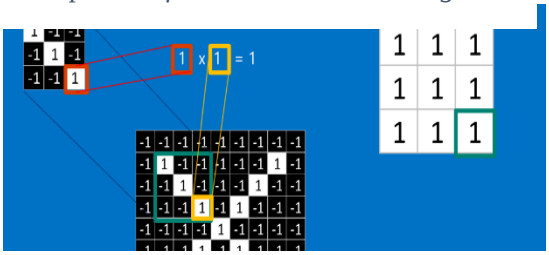


Figure 4.20: Calculation of dot products resulting in the degree of similarity of pieces.

To complete our convolution, we repeat this process, lining up the feature with every possible image patch.

Answer from each convolution can be taken and a new two-dimensional array can be constructed from it, based on where in the image each patch is located. This map of matches is also a filtered version of the original image. It's a map of where in the image the feature is found. Values close to 1 show strong matches, values close to -1 show strong matches for the photographic negative of a feature, and values near zero show no match of any sort.

The result is a set of filtered images, one for each of the filters. It's convenient to think of this whole collection of convolution operations as a single processing step. In CNNs this is referred to as a convolution layer, hinting at the fact that it will soon have other layers added to it.

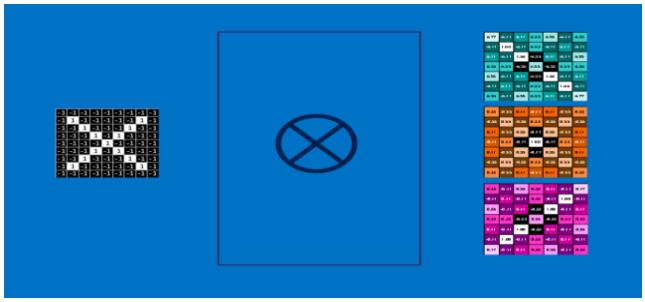


Figure 4.21: Filtered images for each filter

POOLING:

Another power tool that CNNs use is called pooling. Pooling is a way to take large images and shrink them down while preserving the most important information in them. It consists of stepping a small window across an image and taking the maximum value from the window at each step.

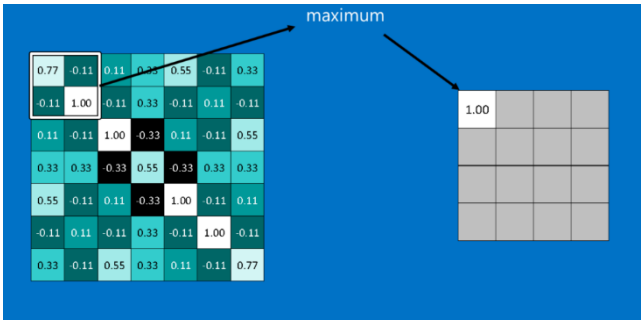


Figure 4.22: Demonstration of Poolling operation where maximum value is propagated to the shrunken dimension

In practice, a window 2 or 3 pixels on a side and steps of 2 pixels work well. After pooling, an image has about a quarter as many pixels as it started with. Because it keeps the maximum value from each window, it preserves the best fits of each feature within the window. This means that it doesn't care so much exactly where the feature fit if it fits somewhere within the window. The result of this is that CNNs can find whether a feature is in an image without worrying about where it is. This helps solve the problem of computers being hyper-literal.

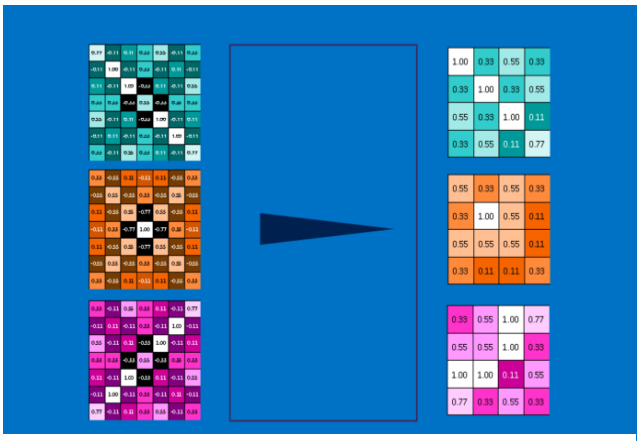


Figure 4.23: Following the pooling process for all the filters

A pooling layer is just the operation of performing pooling on an image or a collection of images. The output will have the same number of images, but they will each have fewer pixels. This is also helpful in managing the computational load. Taking an 8-megapixel image down to a 2-megapixel image makes life a lot easier for everything downstream.

Rectified Linear Units

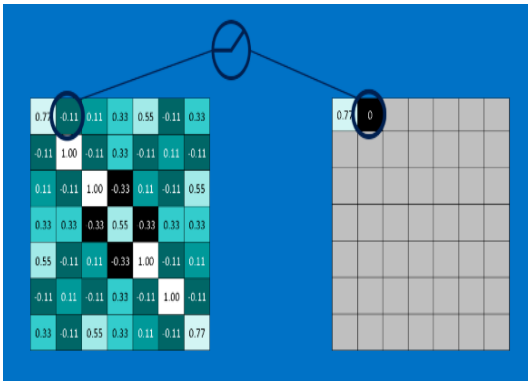
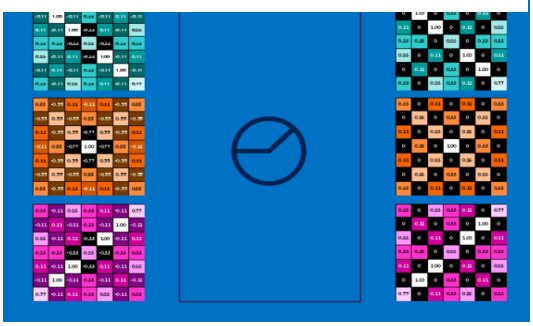


Figure 4.24 & 4.25: The functionality of the ReLU activation function

A small but important player in this process is the Rectified Linear Unit or ReLU. Its math is also very simple—wherever a negative number occurs, swap it out for a 0. This helps the CNN stay mathematically healthy by keeping learned values from getting stuck near 0 or blowing up toward infinity. It's the axle grease of CNNs, but without it they don't get very far. The output of a ReLU layer is the same size as whatever is put into it, just with all the negative values removed.



DEEP LEARNING:

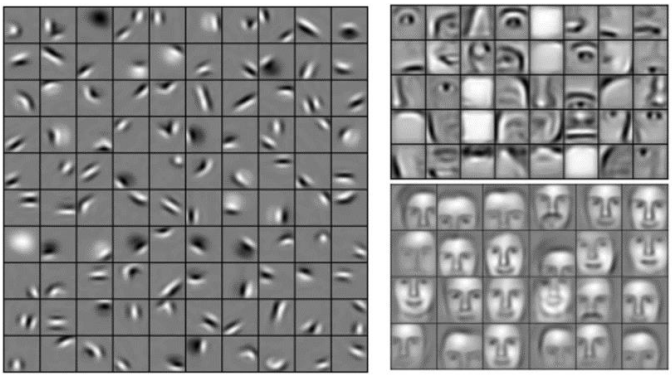


Figure 4.26: Final feature set

The input to each layer (two-dimensional arrays) looks a lot like the output (two-dimensional arrays). Raw images get filtered, rectified and pooled to create a set of shrunken, feature-filtered images. These can be filtered and shrunken again and again. Each time, the features become larger and more complex, and the images become more compact. This lets lower layers represent simple aspects of the image, such as edges and bright spots. Higher layers can represent increasingly sophisticated aspects of the image, such as shapes and patterns. These tend to be readily recognizable. For instance, in a CNN trained on human faces, the highest layers represent patterns that are clearly face-like. Fully connected layers CNNs have one more arrow in their quiver. Fully connected layers take the high-level filtered images and translate them into votes. Fully connected layers are the primary building block of traditional neural networks. Instead of treating inputs as a two-dimensional array, they are treated as a single list and all treated identically. Every value gets its own vote depending on the current image.

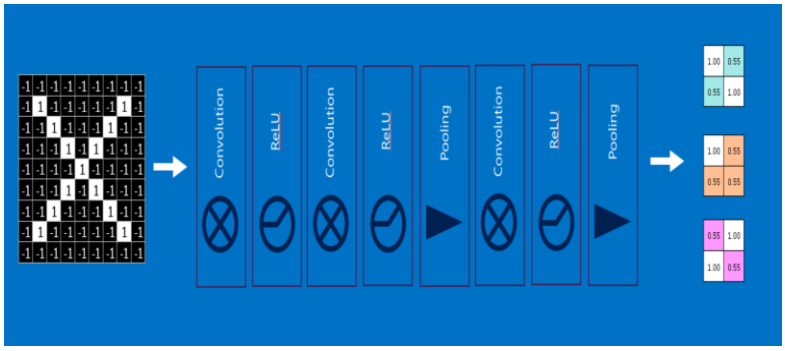


Figure 4.27: The entire process of deep learning

However, the process isn't entirely democratic. Some values are much better than others at knowing when the image contains a particular feature. These get larger votes than the others. These votes are expressed as weights, or connection strengths, between each value and each category. When a new image is presented to the CNN, it percolates through the lower layers until it reaches the fully connected layer at the end. Then an election is held. The answer with the most votes wins and is declared the category of the input. Fully connected layers, like the rest, can be stacked because their outputs (a list of votes) look a whole lot like their inputs (a list of values). In practice, several fully connected layers are often stacked together, with each intermediate layer voting on phantom "hidden" categories. In effect, each additional layer lets the network learn ever more sophisticated combinations of features that help it make better decisions.

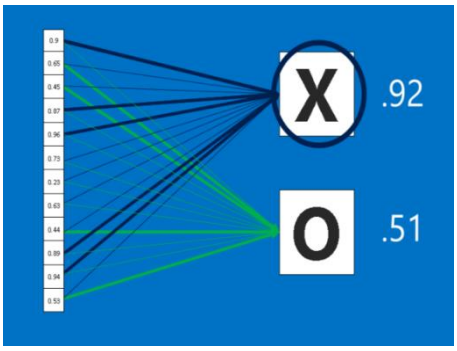


Figure 4.28: The final probabilities that an image is X or O

Backpropagation

Backpropagation is used to identify features and find the weights in fully connected layers. To make use of backpropagation, a collection of images whose answers are known are required.

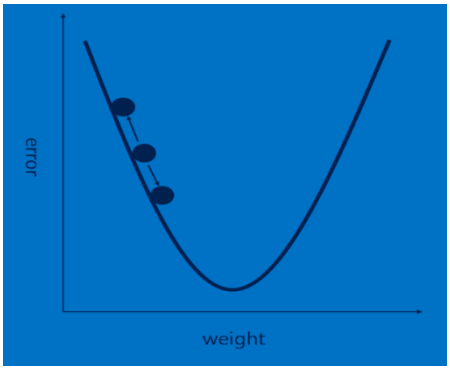


Figure 4.29: The Error vs weight graph for a neuron in CNN's

These are used with an untrained CNN, which means that every pixel of every feature and every weight in every fully connected layer is set to a random value. Images are passed, one after other. Each image the CNN processes results in a vote. The amount of wrongness in the vote, the error, tells how good the features and current weights are. The features and weights can then be adjusted to make the error less. Each value is adjusted a little higher and a little lower, and the new error computed each time. Whichever adjustment makes the error less is kept. After doing this for every feature pixel in every convolutional layer and every weight in every fully connected layer, the new weights give an answer that works slightly better for that image. This is then

repeated with each subsequent image in the set of labelled images. Quirks that occur in a single image are quickly forgotten, but patterns that occur in lots of images get baked into the features and connection weights. With enough labelled images, these values stabilize to a set that works well across a wide variety of cases.

As is probably apparent, backpropagation is another expensive computing step, and another motivator for specialized computing hardware. The largest bottleneck to be aware of when constructing ConvNet architectures is the memory bottleneck. A rule of thumb: If your data is just as useful after swapping any of your columns with each other, then you can't use Convolutional Neural Networks. However if you can make your problem look like finding patterns in an image, then CNNs may be exactly what you need.

5. Use Cases

The current system finds the application in the following:

5.1 Social media tagging:

This application can be used to identify the faces of various users present on social media platforms. Initially a model will be trained with the profile picture or cover photo of the user. Later we can identify the various pictures in which the person is present in. We train on new images to improve the accuracy of the algorithm.

5.2 Access and security:

We train a model for various users of an organization. The trained model can be applied to identify an existing stakeholder. For instance, in a software organization we need access cards to open the gates/doors instead we can use face-detection to serve the purpose. In another case we can apply the model to gated communities where we can identify if a person is a resident if the community or not. An admin can track the frequency of the persons moving in and out and perform analytics. Unlocking smartphones with face recognition technology has become the latest trend.

5.3 Real time face tracking in images and video for fraud detection:

We can divide a footage into frames where we can identify the face of the person performing fraud. For instance, in online examinations if the eyeballs in every frame are captured we can detect if a user is cheating in an exam without any manual supervision. Various shoplifting cases can also be identified.

5.4 Analysing facial expressions:

We extract the features from the face we apply the landmark estimation algorithm. We can measure the width of the lips, position of eyeballs, alignment of eyebrows ,etc. using which we can determine the mood of a person.

5.5 Detecting dysmorphic facial signs for medical diagnosis:

Using face recognition, we can detect and find the solution to treat dysmorphic facial signs in humans. Using the training data, the system gets to know how an average human face structure is designed. Phenotypic combinations are typical for distinct dysmorphic syndromes. Compare the dysmorphic images with the model. Therefore, the detection of particular key point positions in dysmorphic face images is of a high diagnostic value.

5.6 Criminal Identification:

The models can be exclusively trained to store various faces of con-men, criminals, etc. While the model is being trained, their faces need to be positioned in various angles, with hair/beard, without hair/beard etc.
Even if they modify their faces later, a naked human eye may not be able to identify it but an algorithm can identify the face using feature extraction principles.

5.7 Image Search:

Recent developments introduced the ability to search for images by comparing them to others. By uploading an image or using an image URL with search engines, will show us where that image is used on the Web, and display similar images too. The ability to search for matching images is a boon for photographers looking to check where their images have been used. It’s also great for checking if an image is what it said it was.

6. Implementation

The entire problem of face detection involves a series of steps:

1. The first step involves finding all the faces in a given image.
2. The second step is devising a method that is indifferent to faces inclined in weird angles and varying lighting conditions.
3. Create a method that can extract rich features from the image that are integral to tell faces apart.
4. Finally, compare the features of the face with the other samples and tag with the most similar one.

It is hard for computers to perform high level generalizations, so each step should be trained individually.

Step 1: Finding all the Faces

Let's see the first step, finding all possible faces in an image. We need to find each one of them before we can start working on who it is.

Since we only care about where the faces are instead of which colour it is, the image is converted into black and white format.

Then we consider each pixel at a time and compare it with the pixels immediately surrounding it.

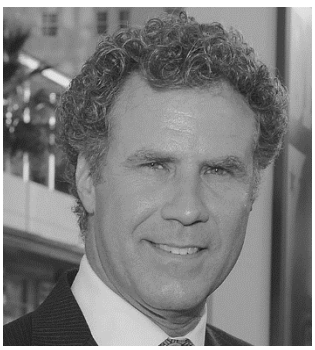


Figure 6.1: The sample image that we took as example converted into a black and white image

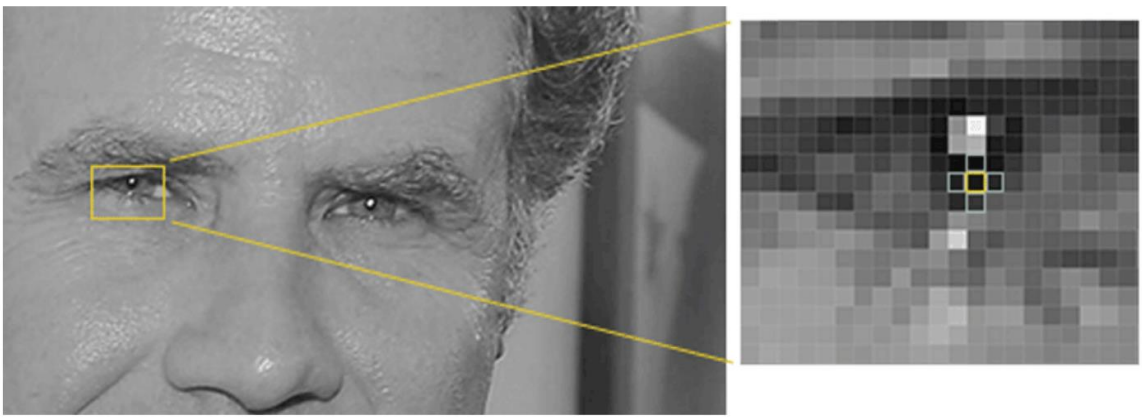


Figure 6.2: Pixel representation of the image and comparison of a pixel intensity with the neighbouring pixels

The process involves finding out how dark the current pixel is compared to the ones around it. We will then draw an arrow in the direction towards the darker side of the image. We will also note down the magnitude of darkness.

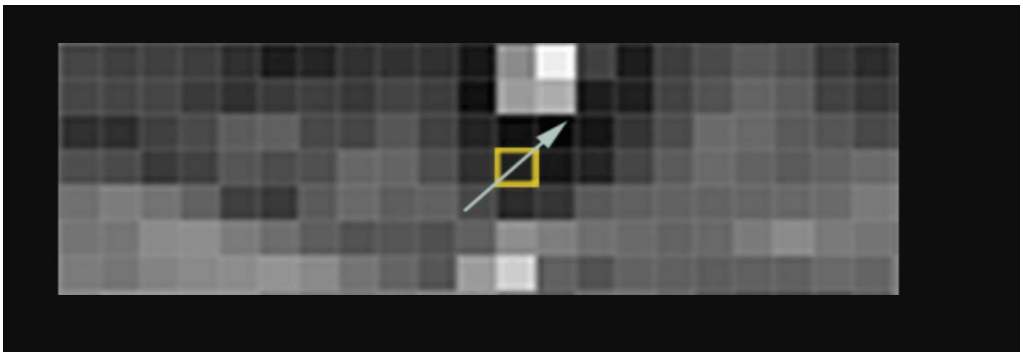


Figure 6.3: Representation of a histogram for a 3X3 block. We can see how the pixels get darker towards the direction of histogram

If we repeat this process for every individual pixel of the image, we end up with a lot of arrows pointing in multiple directions. These arrows are called *gradients* and they show the flow from light to dark across the entire image. The reason for doing this is that, if we consider the pixel values they might vary a lot. But no matter how light or dark the image is the direction of brightness or darkness remains the same. That makes the problem a lot easier to solve. But doing this at a pixel level leaves us with a lot of information. We end up missing the forest for the trees. It is better if we can abstract this to a certain level so that we don't overfit as well as miss important features.

So, we will instead consider 16X16 celled blocks at a time. In each block, we'll count up how many gradients point in each major direction (how many point up, down, right, left, up-right, down-left etc.). Then we'll replace that square in the image with the arrow directions that were the strongest.

We end up with an image representation that is standard, simple and ignores unwanted jargon (features that overfit).

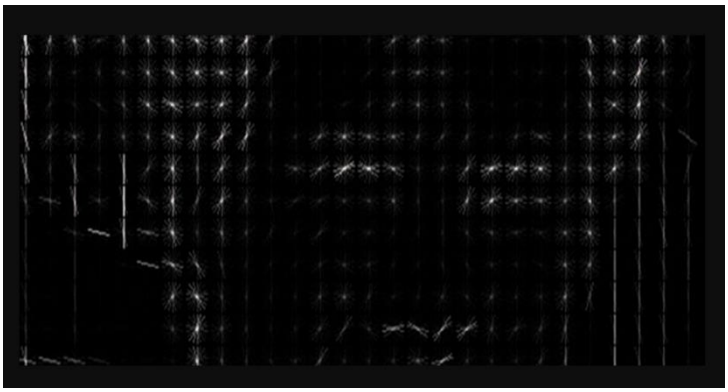


Figure 6.4: HOG representation of the sample image taken

The step following it is to find out which pattern is similar to a generalized HOG pattern generated by training on a lot of faces. We then slide a window over our image to find the region that looks pretty similar.

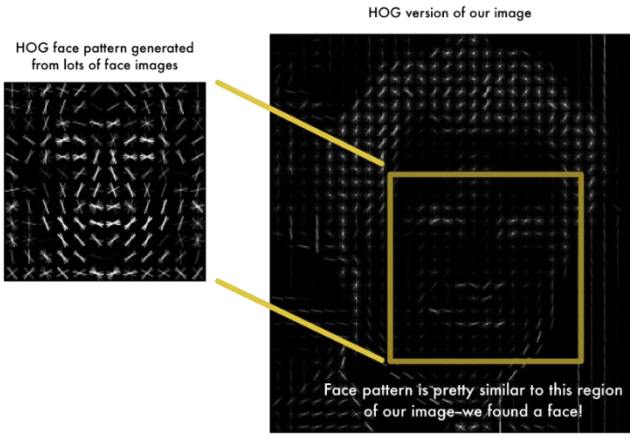


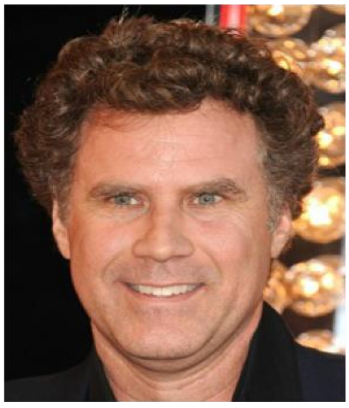
Figure 6.5: Comparison of the HOG's of our sample image with respect to a generalised HOG.

Using this technique, we can now easily find faces in any image:

Figure 6.6: The bounding box drawn around the face for our sample image

Step 2: Posing and Projecting Faces

Now we have an isolated face to work with. But an inclined or faces turned in an angle look different to a computer than a straight looking face.



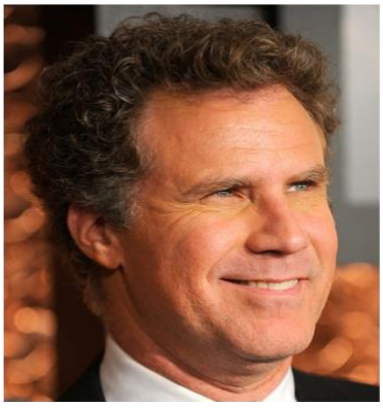


Figure 6.7: Image of the same person in multiple angles

Human brain is complex enough to tell that both the images above are of Will Farrell. To enable a computer to do the same, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps. We use Face Landmark Estimation algorithm to facilitate this. The idea is to come up with 68 unique points (called *landmarks*) on every

face—22 points on the outline of face, 6 points on each eye 9 points on nose and 20 on mouth and lips. Then we train a machine learning algorithm that finds these 68 specific points:

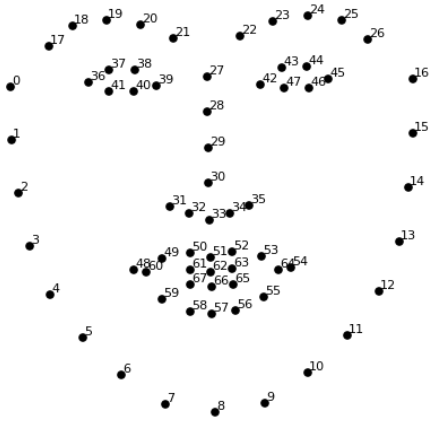


Figure 6.8: the generalized landmarks drawn on any face and their corresponding indices.

The resulting image with 68 points plotted is as follows:

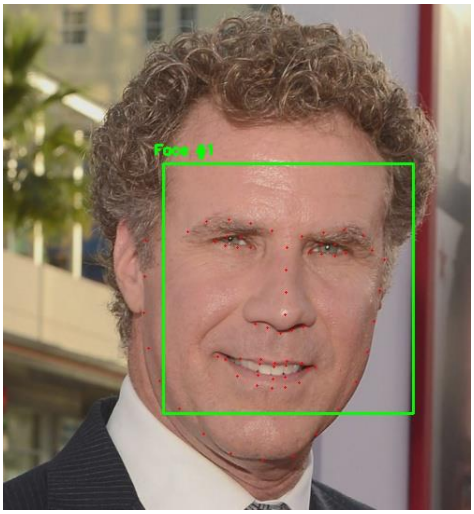


Figure 6.8: All 68 landmarks plotted on our sample image in red.

As we now know where the eyes, nose etc., are we can convert this image into a standard straight looking face. For this we use simple transformation operations like rotation and scaling. We shouldn't perform any steps that distorts the image.The parallel lines should remain parallel. These are called affine transformations):

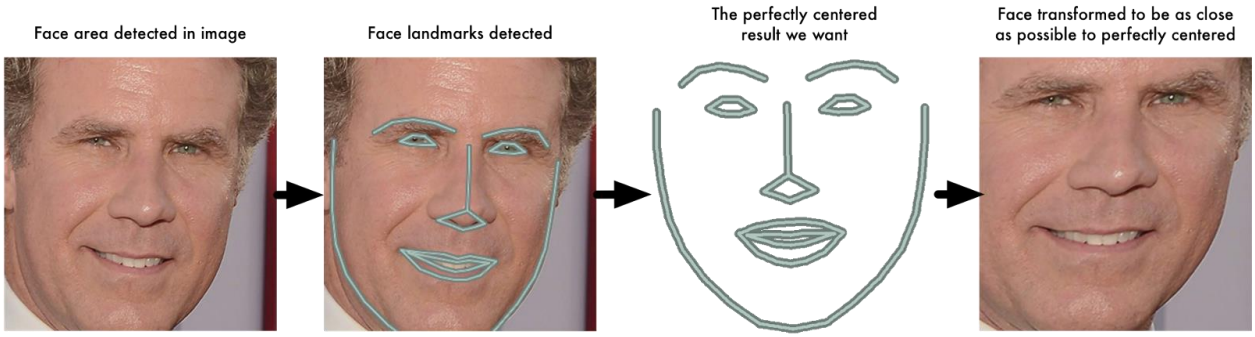


Figure 6.9: The above figure shows the process of operating on an extracted face to give better results.

Now no matter how the face is turned, we can centre the eyes and mouth are in roughly the same position in the image. This will make our next step a lot more accurate.

Step 3: Encoding Faces

The simplest approach to face recognition is to directly compare the unknown faces we found in Step 2 with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person. If the data size is very big the process becomes very slow and redundant if we compare an image with every image in train dataset. So, we extract a few basic measurements from each face in training set. Then we could measure our unknown face the same way and find the known face with the closest measurements. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a Deep Convolutional Neural Network to generate 128 measurements for each face.

The model trains itself by processing over 3 images at a time:

1. Image of a known person
2. Another image of the same person
3. Image of a totally different person

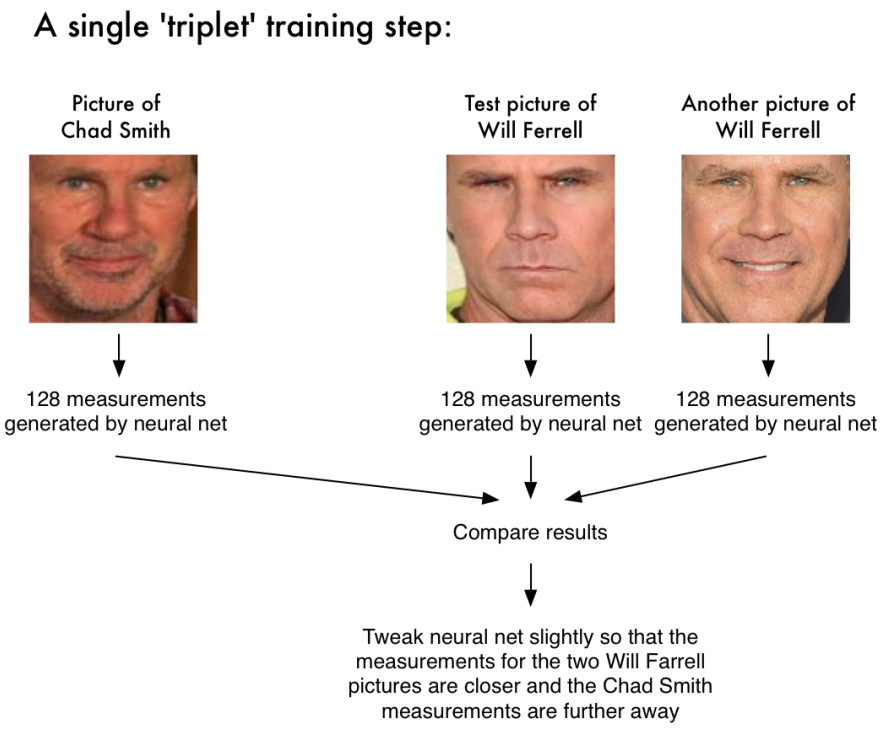


Figure 6.10: The process of Triplet training

After repeating this step for all the images in training set, the neural network learns to reliably generate 128 measurements for each person. Any ten different pictures of the same person should give roughly the same measurements. This process can also be called embedding. The main idea is to reduce complicated raw data (picture) into a list of computer-generated numbers.

Encoding our face image:

Generally, training a neural network consumes a lot of time and computational power. It takes around 24 to 48 hours of continuous training on the best graphic processor to come up with a model that offers great accuracies over benchmark datasets. But then, this process needs to be done only once. And it can also be used to predict new faces correctly. The feature measurements for our test image area as follows

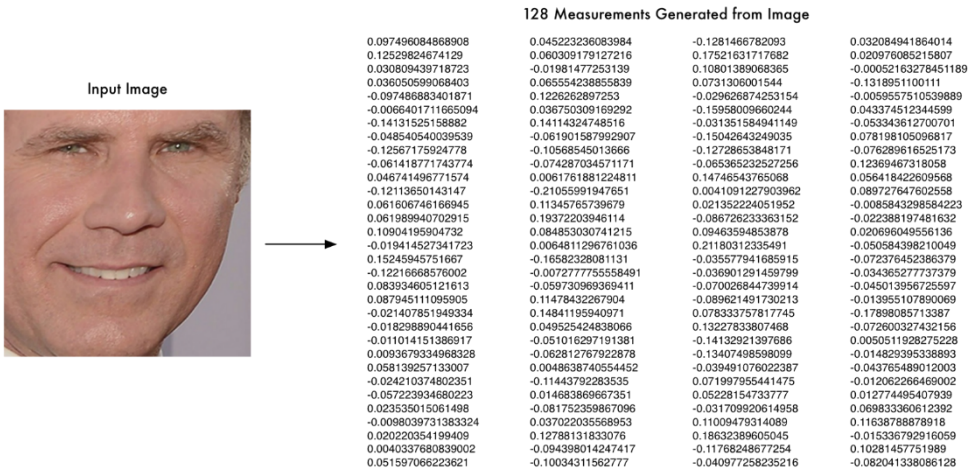


Figure 6.11: The feature measurements generated after feeding the image to trained CNN

Step 4: Finding the person's name from the encoding

This marks the last and the easiest step of the entire process. We compare the features of the face extracted in the previous process with our own training set of images and find the most similar one. This can be done by using any common classification algorithm. We will make use of KNN just because it is simple and supports iterative learning and easy to train.

A bird's eye view of the process is to extract features from the test image and find the closest person. The execution of our system involves training the model over training set. Then we provide path to the image to be tested. The test image is processed incrementally over HOG, Face Landmark estimation, trained deep CNN resulting in

128 features which are then sent to KNN to predict the person. The result is shown below.

Figure 6.12 & 6.13: the image that we started with and the result of the project. The convention that we used shows a red coloured bounding box around the face and denotes it with the name of the person tagged. If no one is tagged, it shows N/A.

7. Testing

7.1 Testing Methodologies

Testing is the process of finding differences between the expected behaviour specified by system models and the observed behaviour implemented system. From modelling point of view, testing is the attempt of falsification of the system with respect to the system models. The goal of testing is to design tests that exercise defects in the system and to reveal problems. The process of executing a program with intent of finding errors is called testing. During testing, the program to be tested is executed with a set of test cases, and the output of the program for the test cases is evaluated to determine if the program is performing as expected. Testing forms the first step in determining the errors in the program. The success of testing in revealing errors in program depends critically on test cases.

7.2 Strategic Approach to Software Testing

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirements analysis where the information domain, functions, behaviour, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral in along streamlines that decreases the level of abstraction on each item. A Strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing will progress by moving outward along the spiral to integration testing, where the focus on the design and the concentration of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against

the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested. Testing is the process of finding difference between the expected behaviour specified by system models and the observed behaviour of the implemented system. Testing is the process of finding difference between the expected behaviour specified by system models and the observed behaviour of the implemented system.

7.3 TESTING ACTIVITIES

Different levels of testing are used in the testing process, each level of testing aims to test different aspects of the system. The basic levels are:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

7.3.1 Unit Testing

Unit testing focuses on the building blocks of the software system, that is, objects and sub system. There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall tests activities, allowing us to focus on smaller units of the system. Second, unit testing makes it easier to pinpoint and correct faults given that few components are involved in this test. Third, Unit testing allows parallelism in the testing activities that is each component can be tested independently of one another. Hence the goal is to test the internal logic of the module.

7.3.2 Integration Testing

In the integration testing, many test modules are combined into sub systems, which are then tested. The goal here is to see if the modules can be integrated

properly, the emphasis being on testing module interaction. After structural testing and functional testing, we get error free modules. These modules are to be integrated to get the required results of the system. After checking a module, another module is tested and is integrated with the previous module. After the integration, the test cases are generated and the results are tested.

7.3.3 System Testing

In system testing the entire software is tested. The reference document for this process is the requirement document and the goal is to see whether the software meets its requirements. The system was tested for various test cases with various inputs.

7.3.4 Acceptance Testing

Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactory. Testing here focus on the external behaviour of the system, the internal logic of the program is not emphasized. In acceptance testing the system is tested for various inputs.

7.4 TYPES OF TESTING

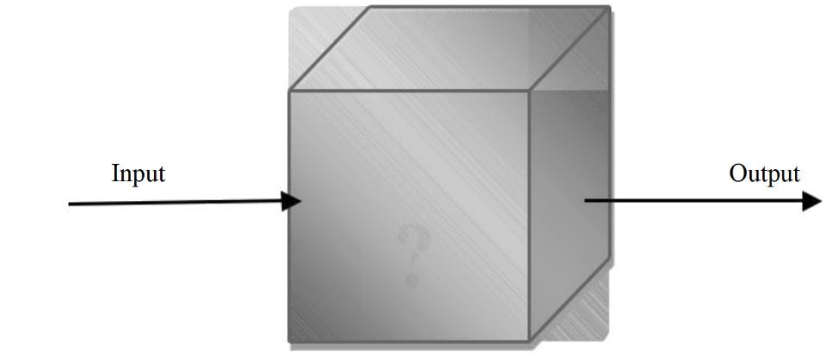
There are 2 types of testing. They are given below:

- Black Box or Functional testing :

  This method is used when knowledge of the specified function that a product has been designed to perform is known. The concept of black box is used to represent a system whose inside workings are not available to inspection. In a black box the test item is a "Black", since its logic is unknown, all that is known is what goes in and what comes out, or the input and output. Black box testing attempts to find errors in the following categories:

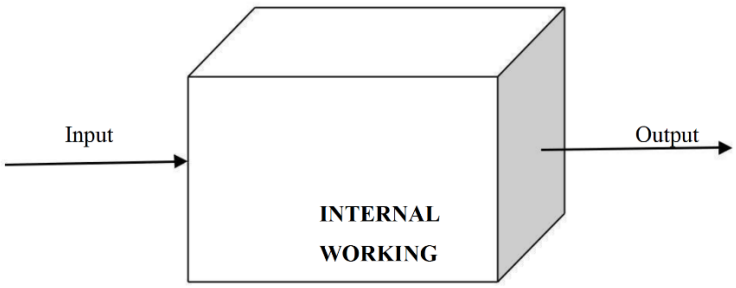
  - Incorrect or missing functions
  - Errors in data structure

- Initialization and termination errors
- Interface errors
- Performance errors



- White Box or Structural testing :

  White box testing is concerned with testing the implementation of the program. The intent of structural is not to exercise all the inputs or outputs but to exercise the different programming and data structure used in the program. Thus, structural testing aims to achieve test cases that will force the desire coverage of different structures. Two types of path testing are statement testing coverage and branch testing coverage.



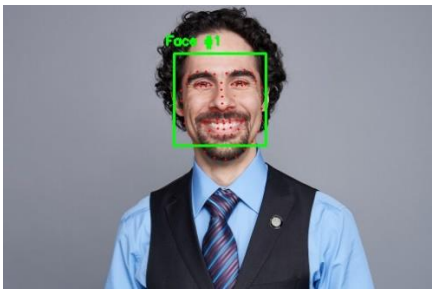
7.5 TESTCASES

7.5.1 Face landmark estimation test cases:

CASE 1: single face

Input:

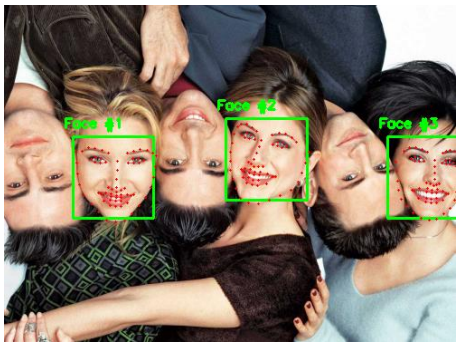
Output:



CASE 2: multiple faces

Input:

Output:



Face Tagging Test cases:

CASE 1 : Two Faces:

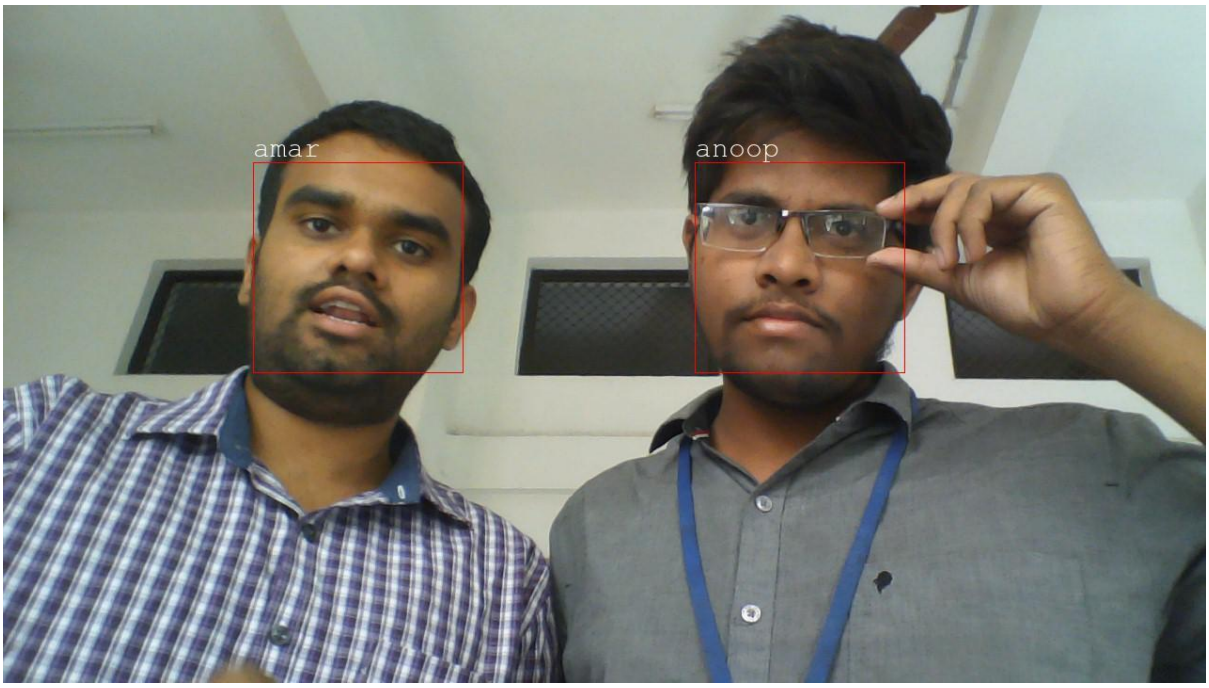


Figure 7.1 Two faces

CASE 2: Low brightness

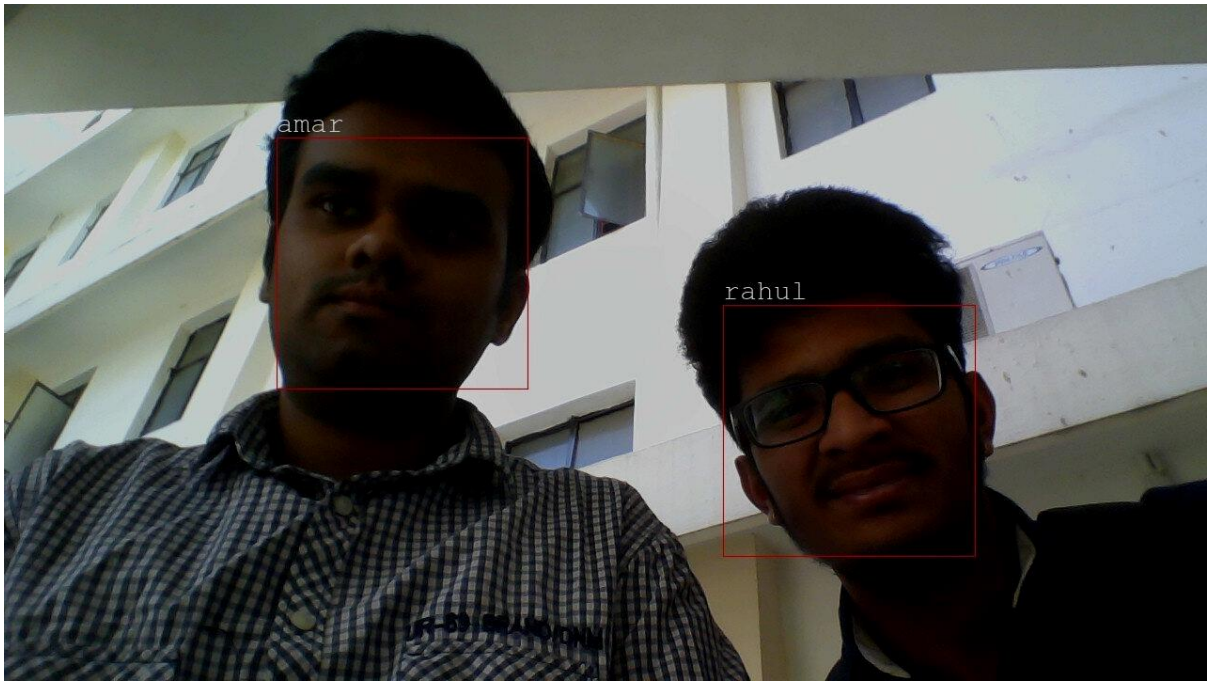


Figure 7.2 dark faces

CASE 3 : Distance of 6 meters away from the camera

Figure 7.3 6 meters distance

CASE 4 : Distance of 8 meters away from the camera

Figure 7.4 8 meters distance

CASE 5 : Grainy image, unable to detect face

Figure 7.5 grainy image

CASE 6 : Multiple faces detection and recognition

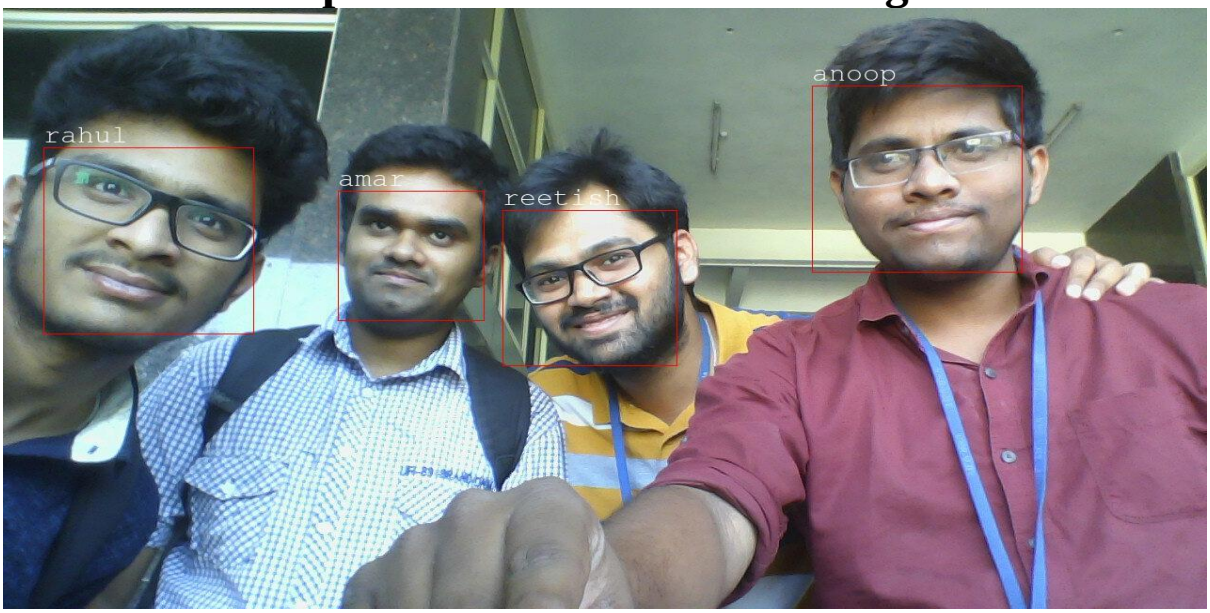


Figure 7.6 multiple faces

CASE 7: Trained and untrained pictures of persons

Figure 7.7 two faces

CASE 8:Decreasing contrast from one end to other of face

Figure 7.8decreasing contrast

CASE 9: Group of 13 people

Figure 7.9 group of 13 people

CASE 10: Group of 10 people in harsh lighting

Figure 7.9 group of 10 people in harsh lighting

CASE 11:Tilted face

Figure 7.11 Tilted face

CASE 12:Faces at different locations in a single image

Figure 7.12 faces at different locations

CASE 13: ID CARD

Figure 7.13 ID CARD

CASE 14 :

Face#1 and Face#2 have similar facial features, yet model is able to distinguish between both but Model distinguishes between similar faces

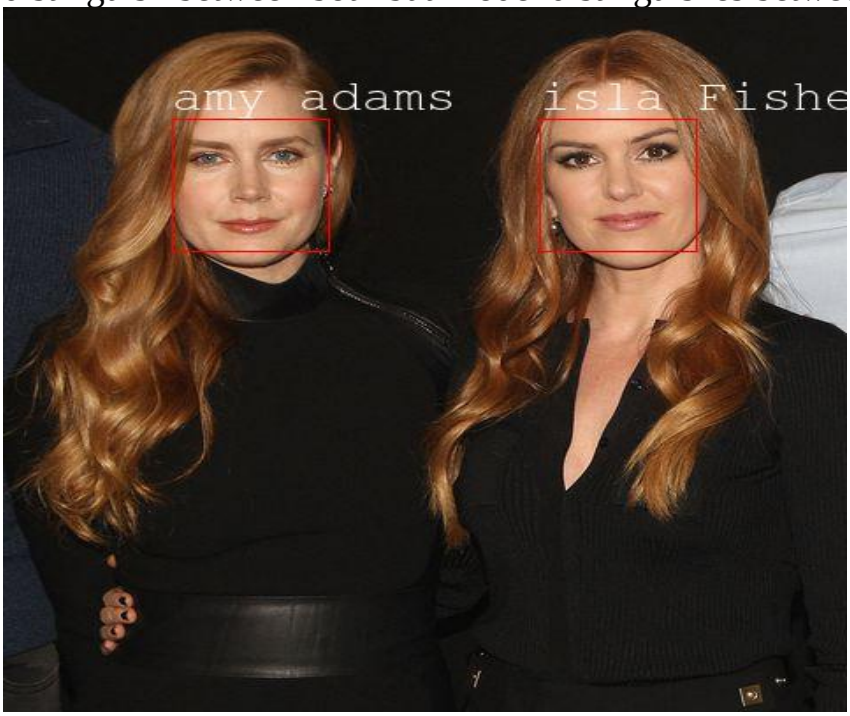


Figure 7.14 similar faces

8. Future improvements

1. Improved face recognition capabilities can also lead to better tools for law enforcement agencies. The model can be trained with increasing resolution of images and more training data to do away with current problems of face recognition software where parts of the face are hidden by wearables.

2. Another promising area of research is maintaining the accuracy of the model for a given individual/subject over a period of time i.e., decrease the effects of aging on making an incorrect prediction.

3. The model can be further modified to be able to train on simulated 3D representation of the images rather than 2D images. This model can then be at the crux of a pay-by-face authentication with payment being made as easy as a "meaningful nod" to the scanner. Alternative use case of this can be face-unlock for smartphones.

4. Facial recognition is an active area of research in the spheres of advertisement too since it can help influence decisions specific to an individual or a group of individuals with specific traits obtained after clustering., Ex: gender identification to show relevant ads

9. Conclusion

Thus, we have used a culmination of multiple state of art algorithms to get the final result. To summarize, first we started off with an image containing single/ multiple faces. Then in order to compensate for the angular inclination of faces and multiple illumination conditions, we have made use of Histogram of Oriented Gradients method, that is proved to be better than the previous methods, to extract faces from the image. This was followed by providing the extracted face as input to a deep convolutional neural network that provides us with 128 rich facial features. All that is left to do now is use a KNN mechanism to identify the correct person.

The results from the previous column clearly state that if cameras with sufficient resolution can be included there is no limit to the amount of advancement that can be fostered with respect to accuracy. The general example of surveillance systems demands a maximum distance of up to 15 metres. Our system was able to detect a person from a maximum distance of about 11 meters. When it comes to the accuracy of recognition systems of companies like Google, Facebook, Baidu etc., they have already achieved an average accuracy of 97%. So, one advancement of face recognition can be increasing the resolution of the images taken. This can be achieved in two ways. First, by the advancements in hardware at feasible prices. The second is "Super-Resolution" (The process of enhancing the resolution of a less clear image). Several techniques and algorithms exist for the above concept. But it consumes more computational power. If this bottleneck can be mastered, the system can make do with less sophisticated imaging hardware.

Face recognition is a technology just reaching sufficient maturity for it to experience a rapid growth in its practical applications. Much research effort around the world is being applied to expanding the accuracy and capabilities of this biometric domain, with a consequent broadening of its application in the near future. Verification systems for physical and electronic access security are available today, but the future holds the promise and the threat of passive customization and automated surveillance systems enabled by face recognition. Face recognition in the form of Apple's Face ID has made tremors in the tech world recently.

10. References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In Proc. of ICML, New York, NY, USA, 2009. 2

D. Chen, S. Ren, Y. Wei, X. Cao, and J. Sun. Joint cascade face detection and alignment. In Proc. ECCV, 2014. 7

K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbour classification. In NIPS. MIT Press, 2006. 2

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013. 2, 3, 4, 6

https://arxiv.org/abs/1503.03832

http://brohrer.github.io/how_convolutional_neural_networks_work.html

https://github.com/davisking/dlib

https://docs.scipy.org/doc/

http://scikit-learn.org/stable/documentation.html

http://download.tensorflow.org/paper/whitepaper2015.pdf


SHOW AND TELL
ANDA PUNNAIAH SQU

PLAGARISM REPORT

Image Key-Point Detection & Tagging

ORIGINALITY REPORT

29%	20%	5%	21%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS