**HO CHI MINH CITY NATIONAL UNIVERSITY**

BACH KHOA UNIVERSITY

---------------o0o---------------

****

-------------------------------------------------------------------------------------------------------

**CAPSTONE PROJECT 2**

-------------------------------------------------------------------------------------------------------

**REPORT**

**CROP LICENSE PLATE AND REGCONIZE DIGITS**

**Instructor: Assoc.Prof, Dr. Hà Hoàng Kha**

**Members: Lê Hoàng Giang 1551032**

**CONTENT**

[CHAPTER 1 THEORETICAL BASIS](#_Toc369509822)

[ABSTRACT 3](#_Toc369509823)

[1.1 INTRODUCTION 4](#_Toc369509825)

[1.2 COMPUTER VISION 5](#_Toc369509825)

[1.3 DATASET AND FEATURES](#_Toc369509825) 16

[CHAPTER 2 METHODOLOGY](#_Toc369509827)

[2.1 ANACONDA](#_Toc369509829) 18

[2.2 JUPYTER NOTEBOOK](#_Toc369509829) 20

[2.3 DATA PROCESSING](#_Toc369509829) 27

[2.4 TESSERACT](#_Toc369509829) 29

[2.5 OPENCV](#_Toc369509829) 31

[2.6 MODEL HAARCASCADE](#_Toc369509829) 32

[CHAPTER 3 RESULTS AND APPLICATIONS](#_Toc369509830) 34

[REFERENCE 36](#_Toc369509834)

[TOOLS 36](#_Toc369509835)

# CHAPTER 1: THEORETICAL BASIS

## ABSTRACT

## Traffic control and vehicle owner identification has become major problem in every country. Sometimes it becomes difficult to identify vehicle owner who violates traffic rules and drives too fast. Therefore, it is not possible to catch and punish those kinds of people because the traffic personal might not be able to retrieve vehicle number from the moving vehicle because of the speed of the vehicle. Therefore, there is a need to develop Automatic Number Plate Recognition (ANPR) system as a one of the solutions to this problem. Therefore, I build an system to do that job.

## 1.1 INTRODUCTION

## In last few years, ANPR or license plate recognition (LPR) has been one of the useful approaches for vehicle surveillance. It is can be applied at number of public places for fulfilling some of the purposes like traffic safety enforcement, automatic toll text collection , car park system and Automatic vehicle parking system . ANPR algorithms are generally divided in four steps: (1) Vehicle image capture (2) Number plate detection (3) Character segmentation and (4) Character recognition. As it is shown in Fig, the first step i.e. to capture image of vehicle looks very easy but it is quite exigent task as it is very difficult to capture image of moving vehicle in real time in such a manner that none of the component of vehicle especially the vehicle number plate should be missed. Presently number plate detection and recognition processing time is less than 50 ms [4] in many systems. The success of fourth step depends on how second and third step are able to locate vehicle number plate and separate each character. These systems follow different approaches to locate vehicle number plate from vehicle and then to extract vehicle number from that image. Most of the ANPR systems are based on common approaches like artificial neural network (ANN), Probabilistic neural network (PNN) , Optical Character Recognition (OCR). Feature salient , MATLAB, Configurable method, Sliding concentrating window (SCW), BP neural network, support vector machine(SVM), inductive learning , region based , color segmentation , fuzzy based algorithm , scale invariant feature transform (SIFT) , trichromatic imaging, Least Square Method(LSM) , online license plate matching based on weighted edit distance and color-discrete characteristics. A case study of license plate reader (LPR) is well explained in. Some authors focus on improving resolution of the low-resolution then to extract vehicle number from that image. Most of the ANPR systems are based on common approaches like artificial neural network (ANN), Probabilistic neural network (PNN) [11], Optical Character Recognition (OCR), Feature salient , MATLAB, Configurable method, Sliding concentrating window (SCW), BP neural network, support vector machine(SVM), inductive learning, region based , color segmentation, fuzzy based algorithm, scale invariant feature transform (SIFT) , trichromatic imaging, Least Square Method(LSM), online license plate matching based on weighted edit distance and color-discrete characteristics . A case study of license plate reader (LPR) is well explained in. Some authors focus on improving resolution of the low-resolution

## 1.2 COMPUTER VISION

**Definition**

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. "Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding." As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

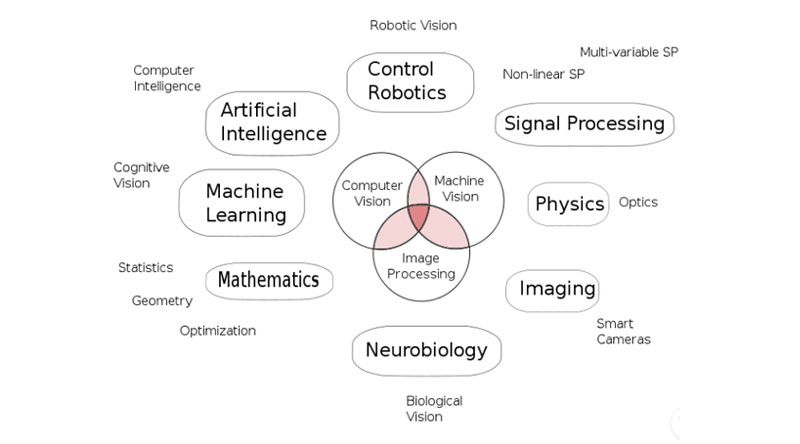


Figure 1: Place of computer vison

The goal of Computer Vision is to emulate human vision using digital images through three main processing components, executed one after the other:  
1. Image acquisition  
2. Image processing  
3. Image analysis and understanding

As our human visual understanding of world is reflected in our ability to make decisions through what we see, providing such a visual understanding to computers would allow them the same power :

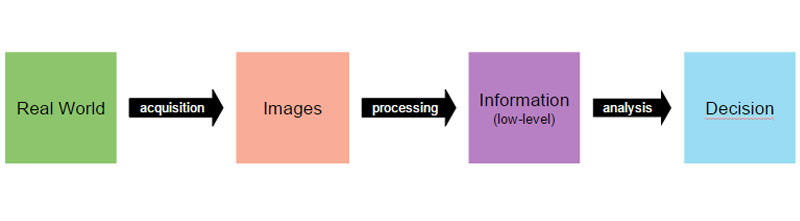


Figure 2: Each step of computer vison

**Image preprocessing**

The second component of Computer Vision is the low-level processing of images. Algorithms are applied to the binary data acquired in the first step to infer low-level information on parts of the image. This type of information is characterized by image edges, point features or segments, for example. They are all the basic geometric elements that build objects in images.

**Edge detection**

Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. The same problem of finding discontinuities in one-dimensional signals is known as step detection and the problem of finding signal discontinuities over time is known as change detection. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction.

The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. It can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to: discontinuities in depth, discontinuities in surface orientation, changes in material properties and variations in scene illumination. In the ideal case, the result of applying an edge detector to an image may lead to a set of connected curves that indicate the boundaries of objects, the boundaries of surface markings as well as curves that correspond to discontinuities in surface orientation. Thus, applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image. If the edge detection step is successful, the subsequent task of interpreting the information contents in the original image may therefore be substantially simplified. However, it is not always possible to obtain such ideal edges from real life images of moderate complexity. Edges extracted from non-trivial images are often hampered by fragmentation, meaning that the edge curves are not connected, missing edge segments as well as false edges not corresponding to interesting phenomena in the image – thus complicating the subsequent task of interpreting the image data. Edge detection is one of the fundamental steps in image processing, image analysis, image pattern recognition, and computer vision techniques.

**Thresholding and linking**

Once we have computed a measure of edge strength (typically the gradient magnitude), the next stage is to apply a threshold, to decide whether edges are present or not at an image point. The lower the threshold, the more edges will be detected, and the result will be increasingly susceptible to noise and detecting edges of irrelevant features in the image. Conversely a high threshold may miss subtle edges, or result in fragmented edges. If the edge is applied to just the gradient magnitude image, the resulting edges will in general be thick and some type of edge thinning post-processing is necessary. For edges detected with non-maximum suppression however, the edge curves are thin by definition and the edge pixels can be linked into edge polygon by an edge linking (edge tracking) procedure. On a discrete grid, the non-maximum suppression stage can be implemented by estimating the gradient direction using first-order derivatives, then rounding off the gradient direction to multiples of 45 degrees, and finally comparing the values of the gradient magnitude in the estimated gradient direction. A commonly used approach to handle the problem of appropriate thresholds for thresholding is by using thresholding with hysteresis. This method uses multiple thresholds to find edges. We begin by using the upper threshold to find the start of an edge. Once we have a start point, we then trace the path of the edge through the image pixel by pixel, marking an edge whenever we are above the lower threshold. We stop marking our edge only when the value falls below our lower threshold. This approach makes the assumption that edges are likely to be in continuous curves, and allows us to follow a faint section of an edge we have previously seen, without meaning that every noisy pixel in the image is marked down as an edge. Still, however, we have the problem of choosing appropriate thresholding parameters, and suitable thresholding values may vary over the image.

**Edge thinning**

Edge thinning is a technique used to remove the unwanted spurious points on the edges in an image. This technique is employed after the image has been filtered for noise (using median, Gaussian filter etc.), the edge operator has been applied (like the ones described above) to detect the edges and after the edges have been smoothed using an appropriate threshold value. This removes all the unwanted points and if applied carefully, results in one pixel thick edge elements.

Advantages:

1. Sharp and thin edges lead to greater efficiency in object recognition.
2. If [Hough transforms](https://en.wikipedia.org/wiki/Hough_transform) are used to detect lines and ellipses, then thinning could give much better results.
3. If the edge happens to be the boundary of a region, then thinning could easily give the image parameters like perimeter without much algebra.

There are many popular algorithms used to do this, one such is described below:

1. Choose a type of connectivity, like 8, 6 or 4.
2. [8 connectivity](https://en.wikipedia.org/wiki/Moore_neighborhood) is preferred, where all the immediate pixels surrounding a particular pixel are considered.
3. Remove points from North, south, east and west.
4. Do this in multiple passes, i.e. after the north pass, use the same semi processed image in the other passes and so on.
5. Remove a point if:  
   The point has no neighbors in the North (if you are in the north pass, and respective directions for other passes).  
   The point is not the end of a line.  
   The point is isolated.  
   Removing the points will not cause to disconnect its neighbors in any way.
6. Else keep the point.

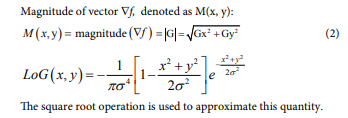
The number of passes across direction should be chosen according to the level of accuracy desired.

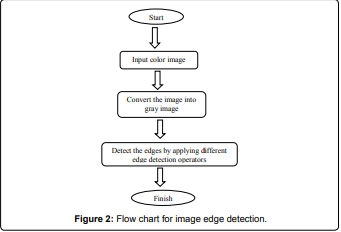
**Gradient based edge detection**

The gradient based approach is also called as mask in digital images. Differential approximations in the either horizontal or vertical direction of the image are calculated with the digital mask . The edge part of the image declared where the gray value rapidly changes. The first order derivative of the image is used to find out the maximum and minimum values in the gradient based operator [4,7]. In image processing using the magnitude of the gradient first order derivatives is achieved . For a function f (x, y), as the dimensional column vector the differential of ‘f’ at coordinates (x, y) is denoted as given below:

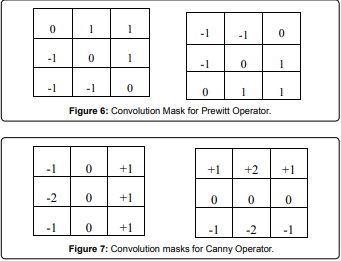
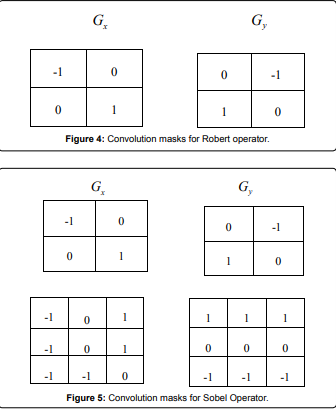


Where ∇f is the gradient of vector. The (x, y) coordinates have the maximum rate of change, normally the gradient direction . This can be observed with the vector evaluation. Here the angle is represents with the gradient angle (Figure 2).





Some famous gradient filter:



**Edge Detection Method Based on Laplacian**

The Laplacian based edge detector found image edges by compute the second order derivative expression of the image. The second order expression has zero crossing where the image edges are found.

However, Laplacian of Gaussian is very useful because the 2nd derivative is very sentient of noise and this is helpful in the filtering of noise from the image.

**Laplacian of gaussian (Log)**

The Laplacian of Gaussian edge detector is also referred as LoG. It is also known as Marr-Hildreth edge detector. This approach defines the image is convoluting with Gaussian filter is to reduce the noise after that smoothing is used to filter out the isolated noise points and the related small structures (Figure 7). Those pixels, that have locally maximum gradient, are examined as edges in which the zero crossing is found. The zero crossing of second order derivative expression avoid the unimportant edges whose correlate with first order derivative selected image edges of those pixel have value above some threshold. Edge direction can obtain where the zero crossing occurs. The 2D LOG function which is cantered on zero-crossing and the σ standard Gaussian deviation has given in the form:



Here, we show the LoG function with the three dimensional plot. Its view looks like the Mexican hat as shown in Figure 8. The Laplacian edge detector uses only one mask to compute the second order derivative mask unlike the Sobel operator. The mask used for it is shown in Figure 9. During the run time on the image from the above three used only one mask because the mask in LoG operator can be calculated in advance. We can find the correct location of edges in all directions by testing the area around the pixels for those have fixed attributes. The orientation of edges and their detection are possible in LoG operator. The variations in the gray level pixels intensity are the indication of the correct edge places, but sometimes there is the damage at the corners and curves. It does not respond to some existing edges because it is also a sensation of noise.

**Edge detection based on fuzzy logic**

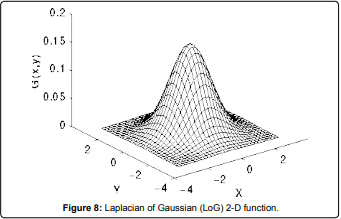
The decision making is to be full filled by the use of fuzzy logic based approach. Since the concept of fuzzy logic was developed in 1965 by Zadesh. In the image processing areas, many analysis have been carried out some of them image edge detection, segmentation and quality assessment. In the past, researchers suggested many techniques for edge detection with the use of fuzzy logic [34]. Davesh Nawgaje [34] presented a Fuzzy Inference System (FIS) approach to detect the edges of the microscopic images within color, which is robust and has stability degrees. They proposed the logic based technique which is a set of three pixels and also used the smallest mask of 2\*2 window image consists of a set of fuzzy rules which highlight all the edges that are correlated with an image (Figure 10). Image processing based on fuzzy logic: Fuzzy logic based image processing is helpful in many fields of processing an image such as feature extraction, segmentation, enhancement and edge detection. This is classified into following three stages:

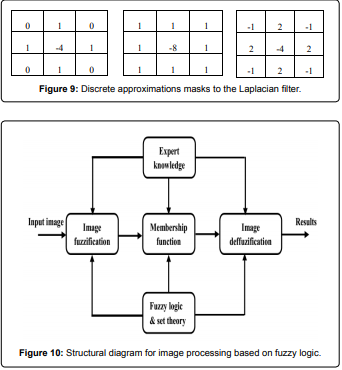
1. Fuzzification of image

2. Membership values alteration

3. Deffuzification of image

For the image fuzzification and deffuzification there is no need of the hardware because the coding and decoding of the image data is possible during the processing of images. In the image processing fuzzy logic helps in the various fields of applications. It supports the if-then rules which are based on human knowledge in the form of coding and to overcome the difficulties use of expert knowledge is possible. It consists of fuzzy set theory which is a powerful tool. There are the different approaches for edge detection is used, some of them are: Yasar Beceriklil and Tayfun Karan proposed that we can apply the heuristic rules to the system and results were checked for different images. Moreover system is consists of dynamic structure adopted by the variation in rules easily. Dharampal et al., proposed an improved type-1 fuzzy logic method for edge detection. This paper applies the Sobel operator combined with a type-1 fuzzy inference system (T1FIS) where the authors conclude that T1FIS is better than the standard methods of edge detection. In the classical techniques for edge detection with the certain parameters such as threshold and σ is used and have fixed edge thickness. But the fuzzy rules base offers the more advantages such as by adding the new rules or by changing parameters we can change the edge thickness so that’s why it is a flexible structure and can be adopted any time to implement edge detection process. The fuzzy based edge detection method provides the flexible structure and less complexity, so that in future, we will detect the edges using the fuzzy rule base.





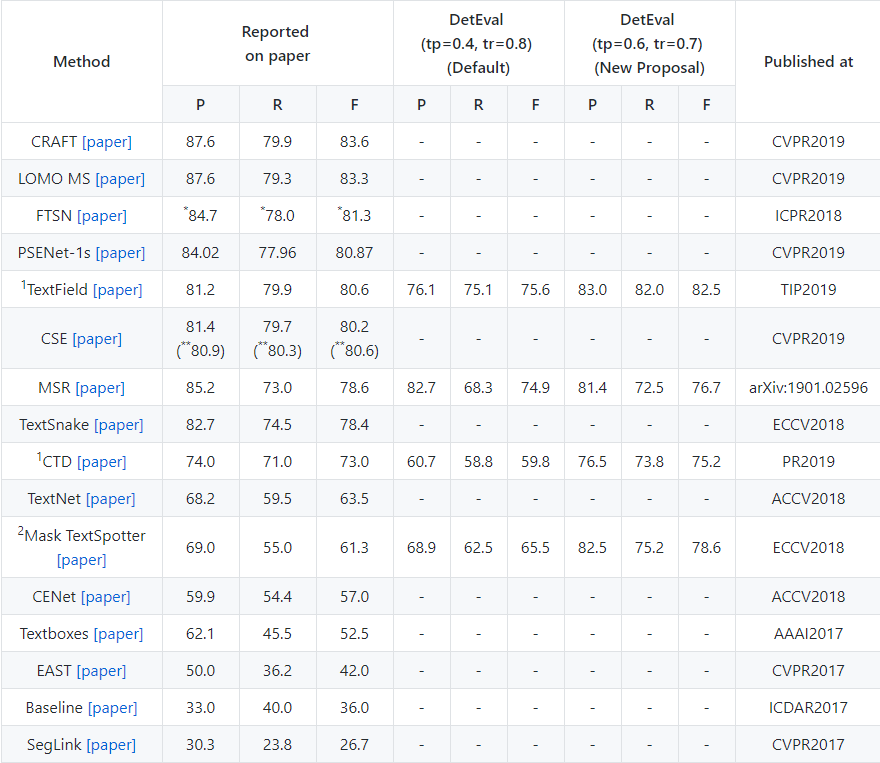
## 1.3 DATASET AND FEATURES

In this word, we use TOTAL-TEXT data. TOTAL-TEXT is a word-level based English curve text dataset. If you are interested in text-line based dataset with both English and Chinese instances, we highly recommend you to refer SCUT-CTW1500. In addition, a Robust Reading Challenge on Arbitrary-Shaped Text (ArT), which is extended from Total-Text and SCUT-CTW1500, was held at ICDAR2019 to stimulate more innovative ideas on the arbitrary-shaped text reading task. Congratulations to all winners and challengers.

Total-Text and SCUT-CTW1500 are now part of the training set of the largest curved text dataset - ArT (Arbitrary-Shaped Text dataset). In order to retain the validity of future benchmarking on Total-Text datasets, we will soon provide a list of ID for users to remove the test-set images of Total-Text from the ArT dataset shall one intend to leverage the extra training data from the ArT dataset. We count on the trust of the research community to perform such removal operation to attain the fairness of the benchmarking.

**Table ranking**

* The results from recent papers on Total-Text dataset are listed below where P=Precision, R=Recall & F=F-score.
* If your result is missing or incorrect, please do not hesisate to contact us.
* \*Pascal VOC IoU metric; [\*\*Polygon Regression](https://arxiv.org/abs/1712.02170)



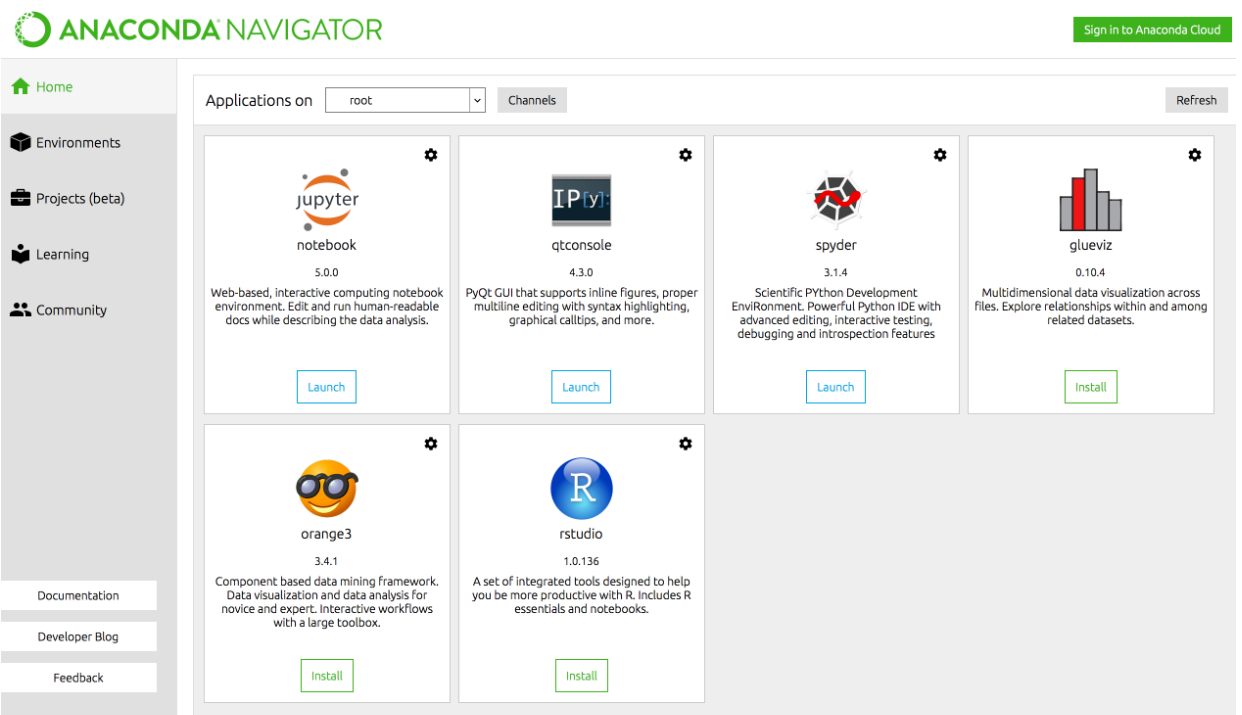
## **CHAPTER 2: METHODOLOGY**

## 2.1 ANACONDA

## Káº¿t quáº£ hÃ¬nh áº£nh cho anaconda navigator

**What is Anaconda Navigator?**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.



**Why use Navigator?**

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages, and use multiple environments to separate these different versions.

The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

**What applications can I access using Navigator?**

The following applications are available by default in Navigator:

* [JupyterLab](https://jupyterlab.readthedocs.io/en/stable/)
* [Jupyter Notebook](https://jupyter.readthedocs.io/en/latest/)
* [QTConsole](https://qtconsole.readthedocs.io/en/latest/)
* [Spyder](https://www.spyder-ide.org/)
* [VSCode](https://code.visualstudio.com/docs)
* [Glueviz](http://glueviz.org/en/stable/)
* [Orange 3 App](http://orange.biolab.si/docs/)
* [Rodeo](http://rodeo.yhat.com/docs/#requirements)
* [RStudio](http://docs.rstudio.com/)

Advanced conda users can also [build your own Navigator applications](https://conda.io/docs/build_tutorials/app.html).

**Conda**

****

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language.

Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

In its default configuration, conda can install and manage the thousand packages at repo.continuum.io that are built, reviewed and maintained by Anaconda®.

Conda can be combined with continuous integration systems such as Travis CI and AppVeyor to provide frequent, automated testing of your code.

The conda package and environment manager is included in all versions of Anaconda and Miniconda. Anaconda Repository. Conda is also included in Anaconda Enterprise, which provides on-site enterprise package and environment management for Python, R, Node.js, Java and other application stacks. Conda is also available on PyPI, although that approach may not be as up to date.

## 2.2 JUPYTER NOTEBOOK



The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

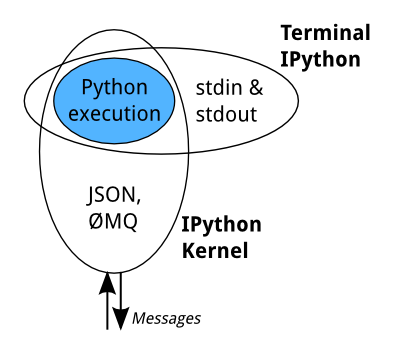
This section focuses on IPython and Jupyter notebook and how they interact. When we discuss IPython, we talk about two fundamental roles:

* Terminal IPython as the familiar REPL
* The IPython kernel that provides computation and communication with the frontend interfaces, like the notebook

Jupyter Notebook and its flexible interface extends the notebook beyond code to visualization, multimedia, collaboration, and more.

All the other interfaces —- the Notebook, the Qt console, ipython console in the terminal, and third party interfaces —- use the IPython Kernel. The IPython Kernel is a separate process which is responsible for running user code, and things like computing possible completions. Frontends, like the notebook or the Qt console, communicate with the IPython Kernel using JSON messages sent over [ZeroMQ](http://zeromq.org/) sockets; the protocol used between the frontends and the IPython Kernel is described in [Messaging in Jupyter](https://jupyter-client.readthedocs.io/en/latest/messaging.html#messaging).

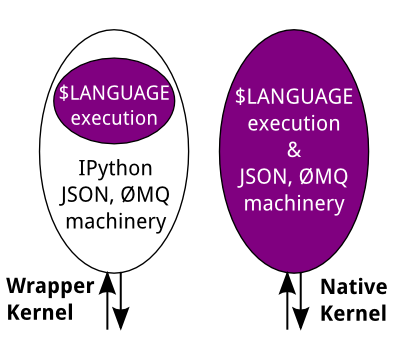
The core execution machinery for the kernel is shared with terminal IPython:



A kernel process can be connected to more than one frontend simultaneously. In this case, the different frontends will have access to the same variables.

This design was intended to allow easy development of different frontends based on the same kernel, but it also made it possible to support new languages in the same frontends, by developing kernels in those languages, and we are refining IPython to make that more practical.

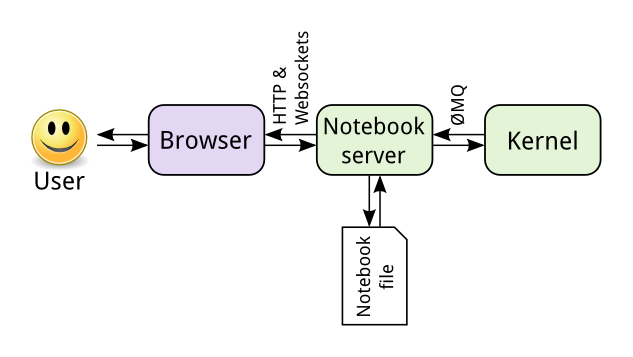
Today, there are two ways to develop a kernel for another language. Wrapper kernels reuse the communications machinery from IPython, and implement only the core execution part. Native kernels implement execution and communications in the target language:



Wrapper kernels are easier to write quickly for languages that have good Python wrappers, like [octave\_kernel](https://pypi.python.org/pypi/octave_kernel), or languages where it’s impractical to implement the communications machinery, like [bash\_kernel](https://pypi.python.org/pypi/bash_kernel). Native kernels are likely to be better maintained by the community using them, like [IJulia](https://github.com/JuliaLang/IJulia.jl) or [IHaskell](https://github.com/gibiansky/IHaskell).

**Notebook**

The Notebook frontend does something extra. In addition to running your code, it stores code and output, together with markdown notes, in an editable document called a notebook. When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with a .ipynb extension.

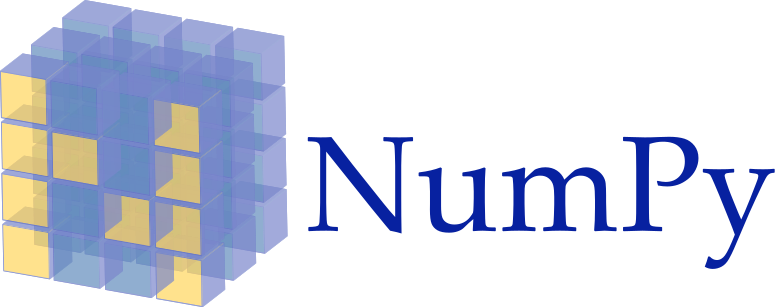


The notebook server, not the kernel, is responsible for saving and loading notebooks, so you can edit notebooks even if you don’t have the kernel for that language—you just won’t be able to run code. The kernel doesn’t know anything about the notebook document: it just gets sent cells of code to execute when the user runs them.

## 2.3 DATA PROCESSING

We used numpy library, scikit-learn to process the Fer2013 data because this library support lots of functions to handle the data. Due to the lack of data preprocessing skills we use only the raw data from this dataset. In the future, we may use other techniques to pre-process the datas before training such as : data augmentation(eg rotation, shifting, skew, scaling, noise, contrast and color jittering.), normalization, Principal component analysis (PCA).

**Numpy**



NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* 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. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the [BSD license](http://www.numpy.org/license.html#license), enabling reuse with few restrictions.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

**Installation**

* **Mac**and **Linux**users can install NumPy via pip command:

pip3 install numpy

* **Windows** does not have any package manager analogous to that in linux or mac.Please download the pre-built windows installer for NumPy from [here](http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy) (according to your system configuration and Python version).  
  And then install the packages manually.

**Scikit-learn**



We used sklearn in order to split the data into training, validation dataset.

Scikit-learn is a [free software](https://en.wikipedia.org/wiki/Free_software) [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [library](https://en.wikipedia.org/wiki/Library_(computing)) for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) programming language. It features various [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and [clustering](https://en.wikipedia.org/wiki/Cluster_analysis) algorithms including [support vector machines](https://en.wikipedia.org/wiki/Support_vector_machine), [random forests](https://en.wikipedia.org/wiki/Random_forests), [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting), [*k*-means](https://en.wikipedia.org/wiki/K-means_clustering) and [DBSCAN](https://en.wikipedia.org/wiki/DBSCAN), and is designed to interoperate with the Python numerical and scientific libraries [NumPy](https://en.wikipedia.org/wiki/NumPy) and [SciPy](https://en.wikipedia.org/wiki/SciPy).

The scikit-learn project started as scikits.learn, a [Google Summer of Code](https://en.wikipedia.org/wiki/Google_Summer_of_Code) project by [David Cournapeau](https://en.wikipedia.org/wiki/David_Cournapeau). Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from [INRIA](https://en.wikipedia.org/wiki/INRIA) took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as [scikit-image](https://en.wikipedia.org/wiki/Scikit-image" \o "Scikit-image) were described as "well-maintained and popular" in November 2012.

Scikit-learn is largely written in Python, with some core algorithms written in [Cython](https://en.wikipedia.org/wiki/Cython) to achieve performance. Support vector machines are implemented by a Cython wrapper around [LIBSVM](https://en.wikipedia.org/wiki/LIBSVM); logistic regression and linear support vector machines by a similar wrapper around [LIBLINEAR](https://en.wikipedia.org/wiki/LIBLINEAR).

## 2.4 TESSERACT

**PyCharm** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) used in [computer programming](https://en.wikipedia.org/wiki/Computer_programming), specifically for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language))language. It is developed by the Czech company [JetBrains](https://en.wikipedia.org/wiki/JetBrains). It provides code analysis, a graphical debugger, an integrated unit tester, integration with [version control systems](https://en.wikipedia.org/wiki/Revision_control) (VCSes), and supports web development with [Django](https://en.wikipedia.org/wiki/Django_(web_framework)).

PyCharm is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform), with [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS) and [Linux](https://en.wikipedia.org/wiki/Linux) versions. The Community Edition is released under the [Apache License](https://en.wikipedia.org/wiki/Apache_License), and there is also Professional Edition with extra features, released under a [proprietary license](https://en.wikipedia.org/wiki/Proprietary_software). Therefore, we use Pycharm community version.

**History**

The beta version was released in July 2010, with the 1.0 arriving 3 months later. Version 2.0 was released on 13 December 2011, version 3.0 on 24 September 2013, and version 4.0 on November 19, 2014.

PyCharm Community Edition, the open source version of PyCharm, became available on 22 October 2013.

**Feature**

* Coding assistance and [analysis](https://en.wikipedia.org/wiki/Code_analysis), with [code completion](https://en.wikipedia.org/wiki/Autocomplete), syntax and error highlighting, linter integration, and quick fixes
* Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages
* Python [refactoring](https://en.wikipedia.org/wiki/Refactoring): including rename, extract method, introduce variable, introduce constant, pull up, push down and others
* Support for web frameworks: [Django](https://en.wikipedia.org/wiki/Django_(web_framework)), [web2py](https://en.wikipedia.org/wiki/Web2py) and [Flask](https://en.wikipedia.org/wiki/Flask_(web_framework))
* Integrated Python [debugger](https://en.wikipedia.org/wiki/Debugger)
* Integrated [unit testing](https://en.wikipedia.org/wiki/Unit_testing), with line-by-line [code coverage](https://en.wikipedia.org/wiki/Code_coverage)
* [Google App Engine](https://en.wikipedia.org/wiki/Google_App_Engine) Python development
* Version control integration: unified user interface for [Mercurial](https://en.wikipedia.org/wiki/Mercurial), [Git](https://en.wikipedia.org/wiki/Git_(software)), [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion), [Perforce](https://en.wikipedia.org/wiki/Perforce) and [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System) with changelists and merge

It competes mainly with a number of other Python-oriented IDEs, including [Eclipse](https://en.wikipedia.org/wiki/Eclipse_(software))'s [PyDev](https://en.wikipedia.org/wiki/PyDev), and the more broadly focused [Komodo IDE](https://en.wikipedia.org/wiki/Komodo_IDE).

**Language written in, library and its environtment**

We can be written in Java, Python and we choose Python for coding because it seems like C that make us easy to understand.

We have to install some library package to run the model in Pycharm. The following figure will these packages.

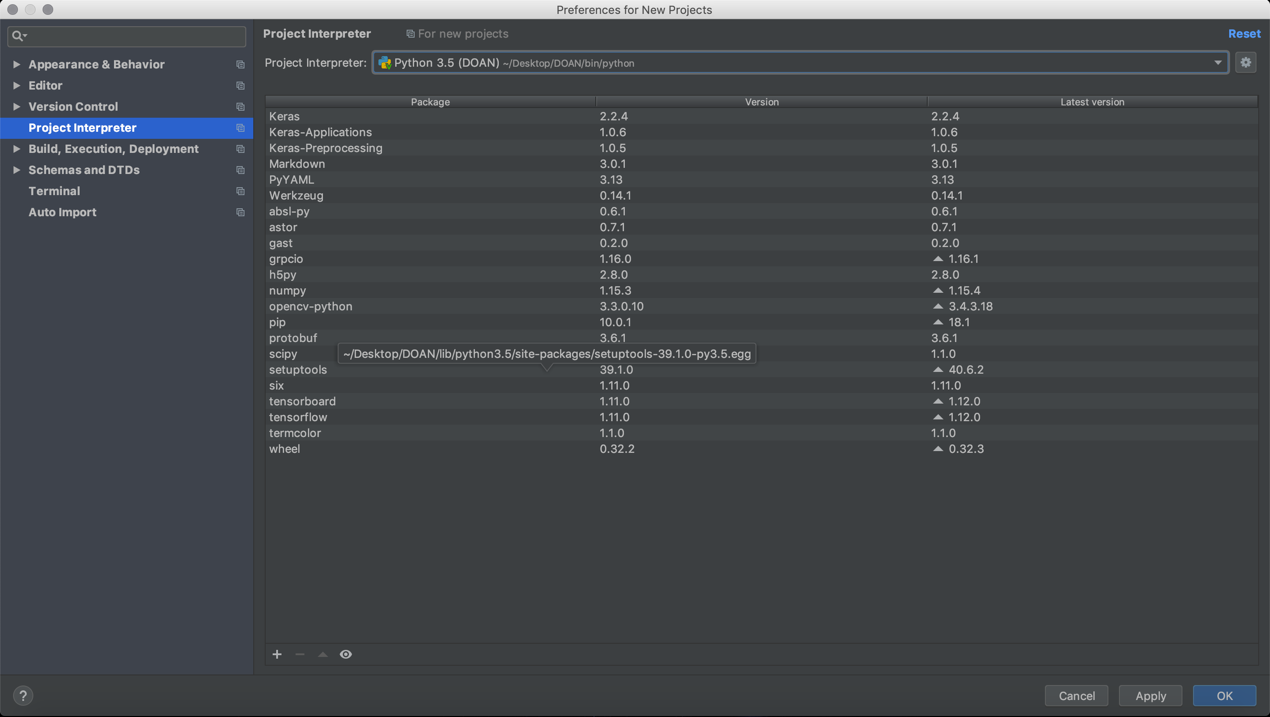


Figure 9: PyCharm Interface

**Python (Programming language)**

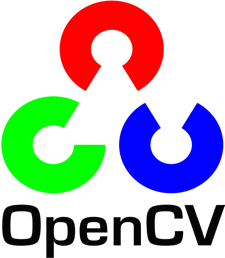


**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level programming language](https://en.wikipedia.org/wiki/High-level_programming_language) for [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum)and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community after 30 years.

Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [Cpython](https://en.wikipedia.org/wiki/CPython" \o "CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open-source_software)softwareand has a community-based development model, as do nearly all of Python’s other implementations. Python and Cpython are managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation).

## 2.5 OPENCV

**OpenCV** (*Open source computer vision*) is a [library of programming functions](https://en.wikipedia.org/wiki/Library_(computing)) mainly aimed at real-time [computer vision](https://en.wikipedia.org/wiki/Computer_vision). Originally developed by [Intel](https://en.wikipedia.org/wiki/Intel_Corporation), it was later supported by [Willow Garage](https://en.wikipedia.org/wiki/Willow_Garage) then Itseez (which was later acquired by Intel). The library is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and free for use under the [open-source](https://en.wikipedia.org/wiki/Open-source) [BSD license](https://en.wikipedia.org/wiki/BSD_license).

OpenCV supports the [deep learning](https://en.wikipedia.org/wiki/Deep_learning) frameworks [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), PyCharm and [Caffe](https://en.wikipedia.org/wiki/Caffe_(software)).

Officially launched in 1999, the OpenCV project was initially an [Intel Research](https://en.wikipedia.org/wiki/Intel_Research_Lablets) initiative to advance [CPU](https://en.wikipedia.org/wiki/Central_processing_unit)-intensive applications, part of a series of projects including [real-time](https://en.wikipedia.org/wiki/Real-time_computing) [ray tracing](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)) and [3D display](https://en.wikipedia.org/wiki/3D_Display) walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel’s Performance Library Team. In the early days of OpenCV, the goals of the project were describedas:

* Advance vision research by providing not only open but also [optimized code](https://en.wikipedia.org/wiki/Code_optimization) for basic vision infrastructure. No more [reinventing the wheel](https://en.wikipedia.org/wiki/Reinventing_the_wheel).
* Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
* Advance vision-based commercial applications by making [portable](https://en.wikipedia.org/wiki/Portability_(computer_science)), performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the [IEEE Conference on Computer Vision and Pattern Recognition](https://en.wikipedia.org/wiki/Conference_on_Computer_Vision_and_Pattern_Recognition) in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 “pre-release” was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the [C++](https://en.wikipedia.org/wiki/C%2B%2B) interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months[[6]](https://en.wikipedia.org/wiki/OpenCV" \l "cite_note-6) and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site. On May 2016, Intel signed an agreement to acquire Itseez, the leading developer of OpenCV.

***2.6 MODEL HAARCASCADE***

Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of **​​** features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.  
  
**It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images.**It is then used to detect objects in other images.

The algorithm has four stages:

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

It is well known for being able to detect faces and body parts in an image, but can be trained to identify almost any object.

Lets take face detection as an example. Initially, the **algorithm needs a lot of positive images of faces and negative images** without faces to train the classifier.   Then we need to extract features from it.

First step is to collect the Haar Features.  A Haar​ feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.

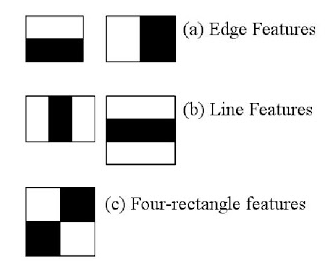
In order to recognize facial expression, firstly we need to detect the face then draw a bouncing box and crop that box. After that we convert the image to gray scale, resize it to 48x48x1 so as to fit our model. Notice that 1 represent for the gray channel, orginally the image is in red, green, blue channels.

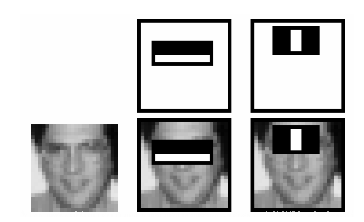
For dectect face, they used haarcascade file. This file was implemented by Intel and OpenCV.

Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then they need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn’t it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. It is achieved by **Adaboost**.





# 

Figure 10: Haarcascade detect

For this, they apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features

You’ll also need OpenCV on your system. The preferred installation methods differ between operating systems so I won’t go into them, but be sure to get at least OpenCV 2.4.5, since that’s the version this post is based on and OpenCV



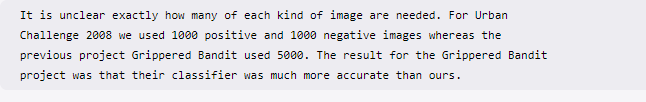
Another thing we need is the OpenCV source code corresponding to our installed version. So if your preferred installation method doesn’t provide you with access to it go and download it. If you get a compiler error further down, be sure to try it with OpenCV 2.4.5.

**Sample**

In order to train our own classifier we need samples, which means we need a lot of images that show the object we want to detect (positive sample) and even more images without the object (negative sample).

How many images do we need? The numbers depend on a variety of factors, including the quality of the images, the object you want to recognize, the method to generate the samples, the CPU power you have and probably some magic.

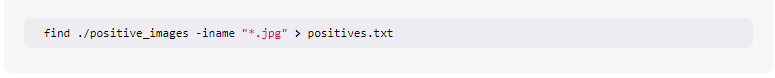
Training a highly accurate classifier takes a lot of time and a huge number of samples. The classifiers made for face recognition are great examples: they were created by researchers with thousands of good images. The TechnoLabsz blog has a [great post](http://www.technolabsz.com/2011/08/how-to-do-opencv-haar-training.html) that provides some information based on their experience:



This post here is just an introduction and getting a large number of good samples is harder than you might think, so we’ll just settle on the right amount that gives us decent results and is not too hard to come by: I’ve had success with the following numbers for small experiments: 40 positive samples and 600 negative samples. So let’s use those!

**POSITIVE IMAGE**

Now we need to either take photos of the object we want to detect, look for them on the internet, extract them from a video or take some Polaroid pictures and then scan them: whatever it takes! We need 40 of them, which we can then use to generate positive samples OpenCV can work with. It’s also important that they should differ in lighting and background. Once we have the pictures, we need to crop them so that only our desired object is visible. Keep an eye on the ratios of the cropped images, they shouldn’t differ that much. The best results come from positive images that look exactly like the ones you’d want to detect the object in, except that they are cropped so only the object is visible. 

Take the positive, cropped images and put them in the ./positive\_images directory of the cloned repository. Then, from the root of the repository, run this command in your shell: 

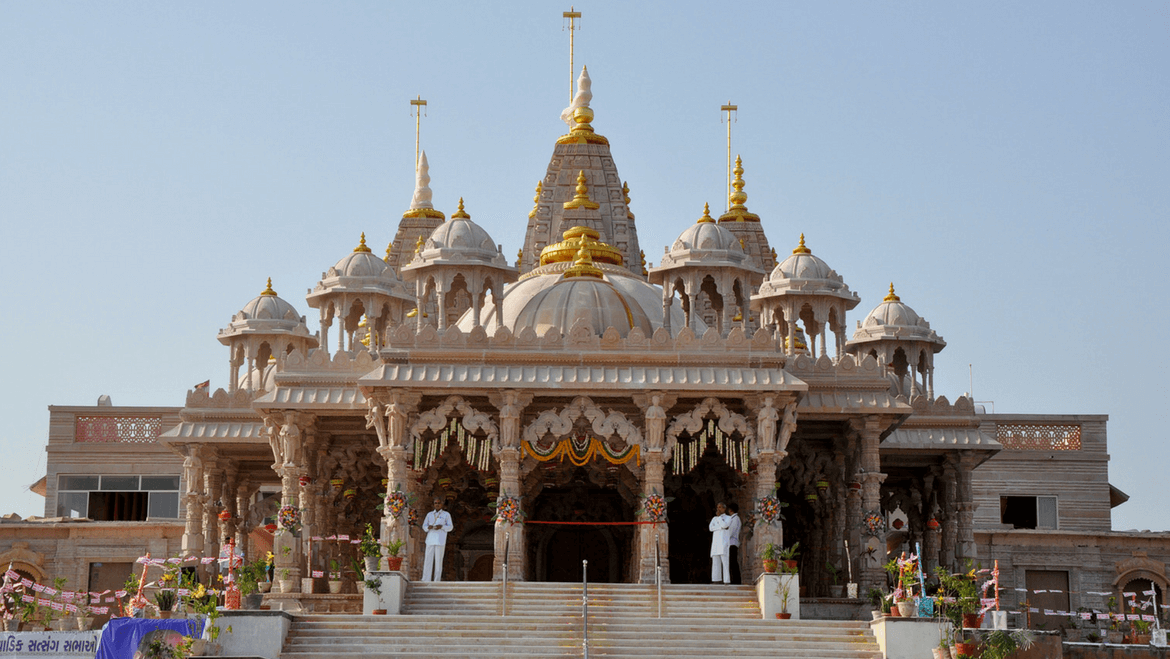
**Negative images**

Now we need the negative images, the ones that don’t show a banana. In the best case, if we were to train a highly accurate classifier, we would have a lot of negative images that look exactly like the positive ones, except that they don’t contain the object we want to recognize. If you want to detect stop signs on walls, the negative images would ideally be a lot of pictures of walls. Maybe even with other signs. We need at least 600 of them. And yes, getting them manually by hand takes a long time. I know, I’ve been there. But again: you could take a video file and extract the frames as images. That way you’d get 600 pictures pretty fast. For the banana classifier I used random photos from my iPhoto library and some photos of the background where I photographed the banana earlier, since the classifier should be able to tell OpenCV about a banana in pretty much any picture. Once we have the images, we put all of them in the negative\_images folder of the repository and use find to save the list of relative paths to a file:



Some example in negative images: (anything but license plate)





# CHAPTER 3: APPLICATION AND RESULT

**Use Opencv for training model**

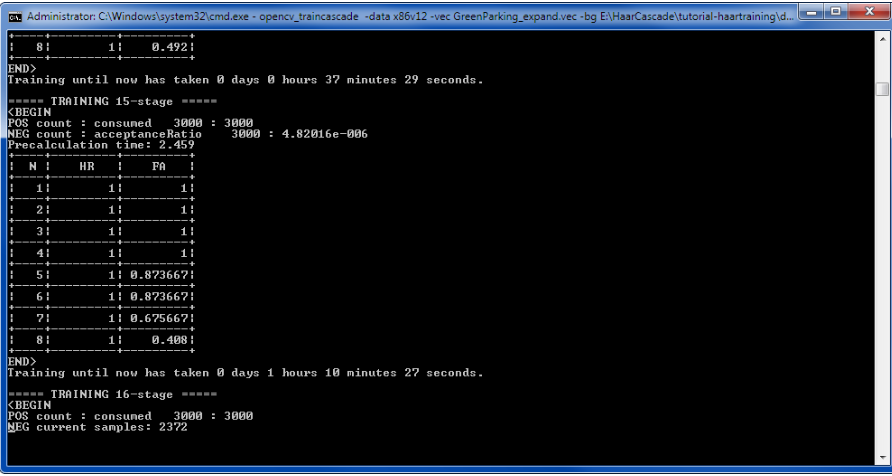
Use this sentence on Linux terminal for create vector named GreenParking.vec which has 1749 positive image with size 30x20 from dataset.

|  |
| --- |
| opencv\_createsamples -info E:\GreenParking\location.txt -vec GreenParking.vec -w 30 -h 20 -num 1749 |

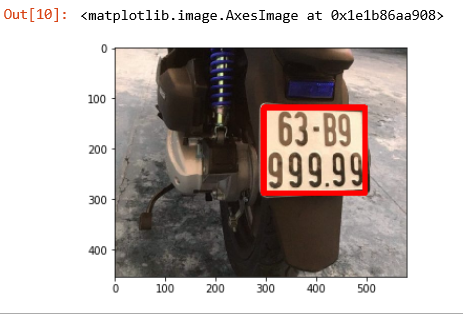
Training by

|  |
| --- |
| opencv\_traincascade -data E:\train -vec GreenParking.vec -bg E:\negatives\bg.txt -numPos 1749 -numNeg 3000 -w 30 -h 20 -featureType LBP -mode ALL |

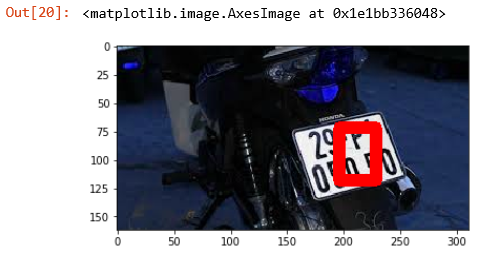
Training will run like this:



The output model then helps us to find license plate on image:



However, sometime the result is not what we want, because the accuracy of haarcascade model is not too high so we will have result like this.



By testing, we rely that license plate will have the biggest area in those result, so we write a basic function:

|  |
| --- |
| def findLargestArea(all\_result):     biggest\_A = 0     for contour in all\_result:         x,y,w,h = contour         A = w\*h         if (A > biggest\_A):             license\_plate = np.array([[x,y,w,h]])             biggest\_A = A     return license\_plate |

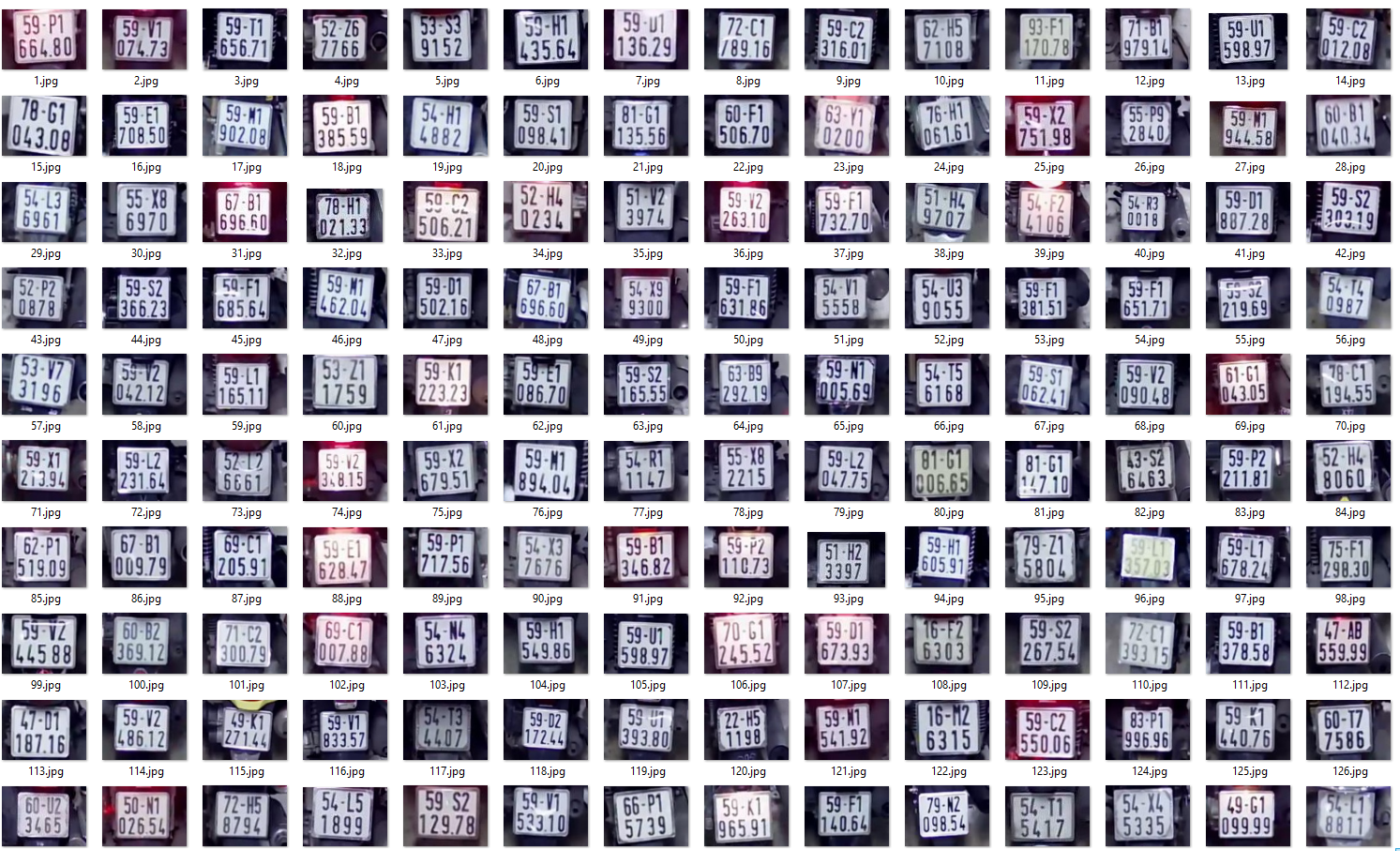
## With that function, we can easy find the biggest area which is license plate in picture



Then by using this code for crop all data in my dataset

|  |
| --- |
| import cv2 import numpy as np from matplotlib import pyplot as plt import os  license\_cascade = cv2.CascadeClassifier('vn2.xml')  img\_path = 'M:\\Project2\_data\\GreenParking\\'  count = '1' count\_str = 1 for i in os.listdir(img\_path):     image = cv2.imread(img\_path+i)     gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)     license = license\_cascade.detectMultiScale(gray, 1.3,5)     if(license == ()):         continue     license = findLargestArea(license)     img = image[license[0,1]:(license[0,1]+license[0,3]),  license[0,0]:(license[0,0]+license[0,2])]     cv2.imwrite(('M:\\Project2\_data\\crop\_by\_cascade\\'+count+'.jpg'), img)      count\_str += 1     count = str(count\_str) |
|  |

I will have an folder full of license:



Then I read them by tesseract. Before put them into tesseract, I need some preprocess these licenses, I decide to use blur and thresholding method then compare their result:

|  |  |  |
| --- | --- | --- |
| |  | | --- | | import pytesseract def OCR\_scan(img\_path = '', pre = 'thresh', save\_path = '', number\_pics = ''):     image = cv2.imread(img\_path)     gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)     if pre == 'thresh':         ret, gray = cv2.threshold(gray, 0, 255,cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)     elif pre =='blur':         gray = cv2.medianBlur(gray, 3)     save\_into = save\_path+number\_pics+'.jpg'     cv2.imwrite(save\_into, gray)  img\_list = 'crop\_image\\' pre1 = 'thresh' pre2 = 'blur' save\_path1 = 'preprocess\_thresh\\' save\_path2 = 'preprocess\_blur\\' count = '1' count\_str = 1 for i in os.listdir(img\_list):     img\_path = img\_list+i     OCR\_scan(img\_path = img\_path, pre = pre1, save\_path = save\_path1, number\_pics = count)     OCR\_scan(img\_path = img\_path, pre = pre2, save\_path = save\_path2, number\_pics = count)     count\_str += 1     count = str(count\_str) | |  | |
|  |

After that code, I had a blur folder:



And a thresholding folder by OTSU method:

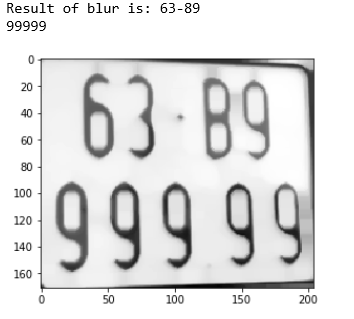


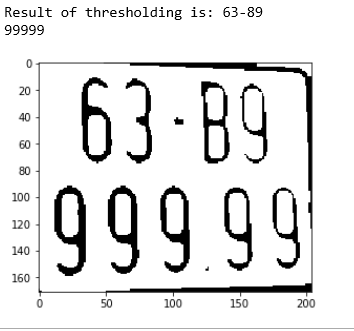
Then put all of them into tesseract predict

|  |
| --- |
| filename = 'preprocess\_thresh\\' list\_thresh = [] list\_blur = [] list\_o = [] list\_test = [] for i in range(28):     name = i+1     thresh = filename + str(name) + '.jpg'     blur = 'preprocess\_blur\\'+ str(name) + '.jpg'     original = 'crop\_image\\'+ str(name) + '.jpg'     test = 'preprocess\_test\\'+ str(name) + '.jpg'     list\_thresh.append(pytesseract.image\_to\_string(Image.open(thresh)))     list\_blur.append(pytesseract.image\_to\_string(Image.open(blur)))     list\_o.append(pytesseract.image\_to\_string(Image.open(original)))     list\_test.append(pytesseract.image\_to\_string(Image.open(test))) |
|  |

**RESULT**

## This is result of blur picture:





As we can see that they both give the good result.

**TOOLS**

## Apps.

* Google Colaboratory (<https://colab.research.google.com/notebooks/welcome.ipynb>)
* Anaconda

**REFERENCE**

<https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d>

<https://arxiv.org/pdf/1804.08348.pdf>

<https://arxiv.org/pdf/1711.04598.pdf>

<https://www.researchgate.net/publication/307571567_Combining_Deep_Facial_and_Ambient_Features_for_First_Impression_Estimation>

<https://thigiacmaytinh.com/tai-nguyen-xu-ly-anh/>

<https://www.quora.com/What-is-haar-cascade>

<https://answers.opencv.org/question/198652/how-to-create-custom-cascade-for-unshapped-objects/>

<https://github.com/cs-chan/Total-Text-Dataset>

<https://sefiks.com/2018/01/10/real-time-facial-expression-recognition-on-streaming-data/>

<https://en.wikipedia.org/wiki/OpenCV>

<https://github.com/opencv/opencv/wiki>

<https://keras.io/>

<https://www.python.org/>

<https://www.tensorflow.org/>

<https://opencv.org/>