

# Grad-CAM visual explanation for face recognition

Design document

Benny Brazawoski, Itamar Kaminski, Raz Tamir

# Table of Contents

Introduction .....	3
About the project.....	3
1. The Core .....	4
1.1. Face Detection Algorithm with Python using OpenCV .....	4
1.1.1. Python .....	4
1.1.2. OpenCV .....	4
1.1.3. Algorithm Outline .....	5
1.2. VGG Face Descriptor with Torch using Caffe .....	6
1.2.1. Torch .....	6
1.2.2. Caffe .....	6
1.2.3. Network Details .....	7
1.2.4. Dataset Collection .....	8
1.2.5. Algorithm Outline .....	8
1.3. Gradient-weighted Class Activation Mapping with Torch .....	8
1.3.1. Algorithm Outline .....	8
1.4. Feature Extraction with Python using OpenCV .....	9
1.4.1. Algorithm Outline .....	9
1.5. Python and Torch Integration .....	10
1.6. Support Functions .....	11
1.7. Errors .....	11
2. Web Application .....	12
2.1. General module design and user flow .....	12
2.2. Django Framework .....	13
2.2.1. Introduction .....	13
2.2.2. System Architecture .....	13
2.2.3. Installation .....	14
2.2.4. Running the sever .....	14
2.2.5. Modules .....	15
2.3. Hosting .....	15

## Introduction

This document is the technical design document of the 'Moshiko's Eye-Brows' project. 'Moshiko's Eye-Brows' is a project of the Computer Science Workshop of Tel-Aviv's University.

The goal of this document is mainly to create general guidelines for the development of the project but also to help us, as a team of 3 people, to work on it separately and merge the layers that compose it.

## About the project

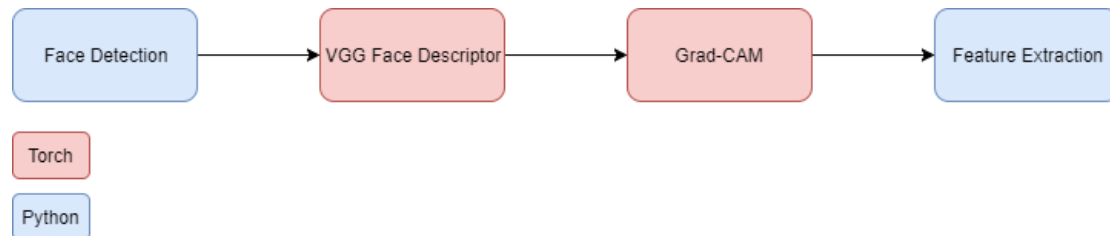
The BRAIN system is a web-based application providing visual explanation for face identity recognition.

Querying the application with an image with face will result the following:

- A name of a celebrity identity that is the most like the person in the query image.
- Visual explanation - the query image blended with a layer of the visualization, highlighting the features that were most significant for the identification.
- Verbal justification – a short textual explanation using facial landmarks.

# 1. The Core

In this chapter, we will describe the core algorithms of our project: face detection, VGG-Face convolutional neural network, Gradient-weighted Class Activation Mapping (Grad-CAM) and feature extraction. We will also explain how we integrated Python with Torch.



## 1.1. Face Detection Algorithm with Python using OpenCV

### 1.1.1. Python

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

In our project, we used Python version 2.7.

Libraries used in the Face detection algorithm:

- cv2 (OpenCV)
- math, OS (both part of The Python Standard Library)

### 1.1.2. OpenCV

OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python. We used face detection using Haar Feature-based Cascade Classifiers.

#### 1.1.2.1. Haar Feature-based Cascade Classifiers

Object Detection using Haar feature-based cascade classifiers is an effective object detection method 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.

To get around this, OpenCV uses cascades. What's a cascade? The best answer can be found from the dictionary: A waterfall or series of waterfalls. Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. The advantage is that the majority of the pictures will return negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time.

Though the theory may sound complicated, in practice it is quite easy. The cascades themselves are just a bunch of XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you.

Cascades XML files used for face detection are:

- haarcascade\_frontalface\_alt2.xml
- haarcascade\_frontalface\_alt.xml
- haarcascade\_frontalface\_default.xml
- haarcascade\_profileface.xml

### 1.1.3. Algorithm Outline

We used an Object-Oriented approach for this algorithm. We created A class 'FaceDetect' in file face\_detect.py. Initializing an instance of this class is by providing it with OpenCV data directory (contains the cascades XML files) and a path to a file (should be an image with face).

We begin by loading the cascades to our program and then finding a face in the query image. At the end, if everything goes as expected, the Class should contain a rectangle where it believes it found a face. We use these 4 rectangle values to crop it and save it as an image to our servers – 'cropped.jpg'



**Figure 1:** The query image before and after ('cropped.jpg') detecting and cropping face

## 1.2. VGG Face Descriptor with Torch using Caffe

The VGG Face Descriptor is a Convolutional Neural Network based on the VGG-Very-Deep-16 architecture, which competing with that of internet giants like Google and Facebook and is evaluated on the Labeled Faces in the Wild and the YouTube Faces dataset.

### 1.2.1. Torch

Torch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep machine learning, and uses the scripting language LuaJIT, and an underlying C implementation.

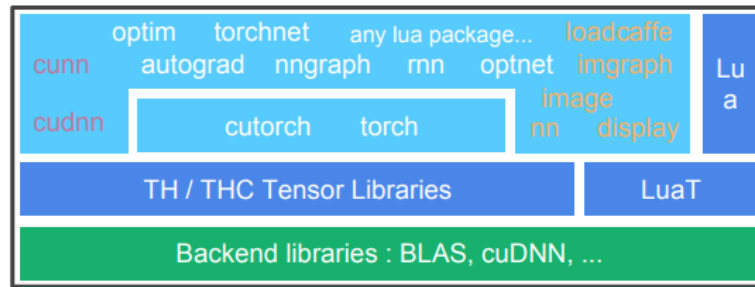
Why Torch?

- Flexibility
- Readability
- Modularity
- Easy to pull someone's code
- Use LuaRocks to install required packages
- Speed

All of this makes Torch very convenient for research.

### 1.2.2. Caffe

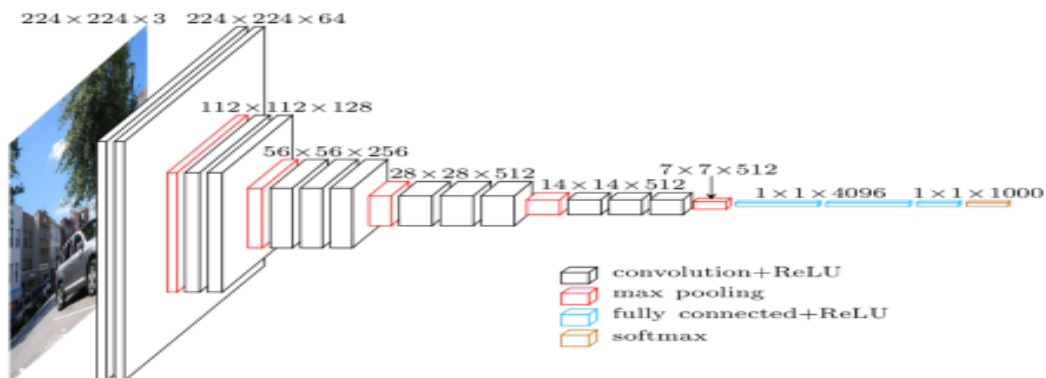
Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. We used the 'loadcaffe' library.



**Figure 3:** Architecture of the Torch framework

### 1.2.3. Network Details:

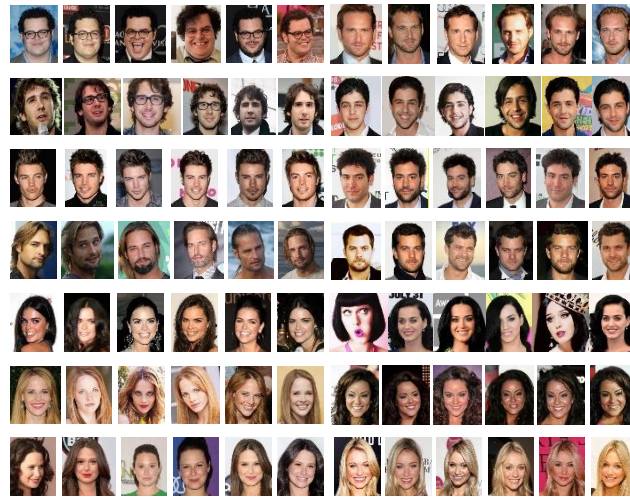
- 3 x 3 Convolution Kernels (Very small)
- Conv. Stride 1 pixel.
- RELU non-linearity
- No local contrast normalization
- 3 Fully connected layers



**Figure 4:** An Illustration of VGG-16 Architecture

#### 1.2.4. Dataset Collection:

2622 celebrities in the final dataset.



**Figure 5:** Example images from our database

#### 1.2.5. Algorithm Outline:

- Load a pre-trained VGG model from file – 'cropped.jpg'.
- Set to evaluate and remove SoftMax layer.
- Load query (cropped) image from file and preprocess it – scale and subtract mean.
- Forward pass.
- Take argmax – 'predlabel'

### 1.3. Gradient-weighted Class Activation Mapping with Torch

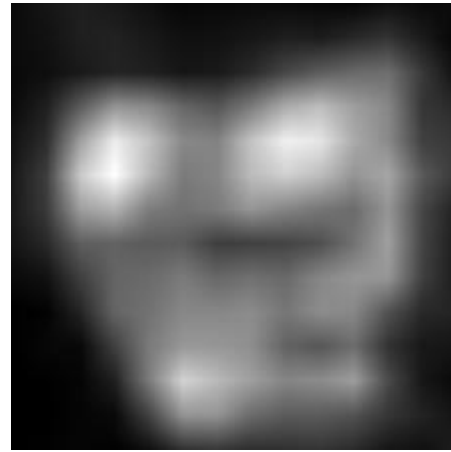
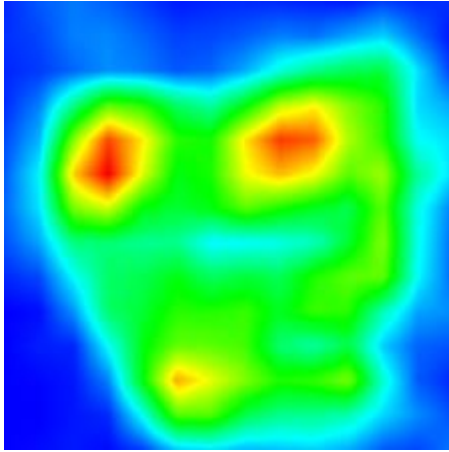
A technique for producing "visual explanations" for decisions from a large class of CNN-based models, making them more transparent. Gradient-weighted Class Activation Mapping (GradCAM), uses the gradients of any target concept, flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept. Unlike previous approaches, GradCAM is applicable to a wide variety of CNN model-families.

#### 1.3.1. Algorithm Outline:

- Receiving integer output: 'predlabel' from VGG-Face Descriptor



- Run GradCAM algorithm as described here:  
<https://arxiv.org/pdf/1611.07450.pdf>
- Save heat map as: heat\_map\_predlabel.png
- Save gc\_map (which is a grayscale map of important pixels) as: gc\_map\_predlabel.png



Figures 6, 7: heat\_map\_683.png and gc\_map\_683.png respectively

#### 1.4. Feature Extraction with Python using OpenCV

Here again we were using the Haar Feature-based Cascade Classifiers of OpenCV and an Object-Oriented approach. We return an object 'Results' that contains all the results from the process: output filename, the features extracted and more.

We used the following XML files:

- "haarcascade\_eye.xml"
- "haarcascade\_smile.xml"
- "haarcascade\_mcs\_mouth.xml"
- "haarcascade\_mcs\_nose.xml"

Python libraries used in this part:

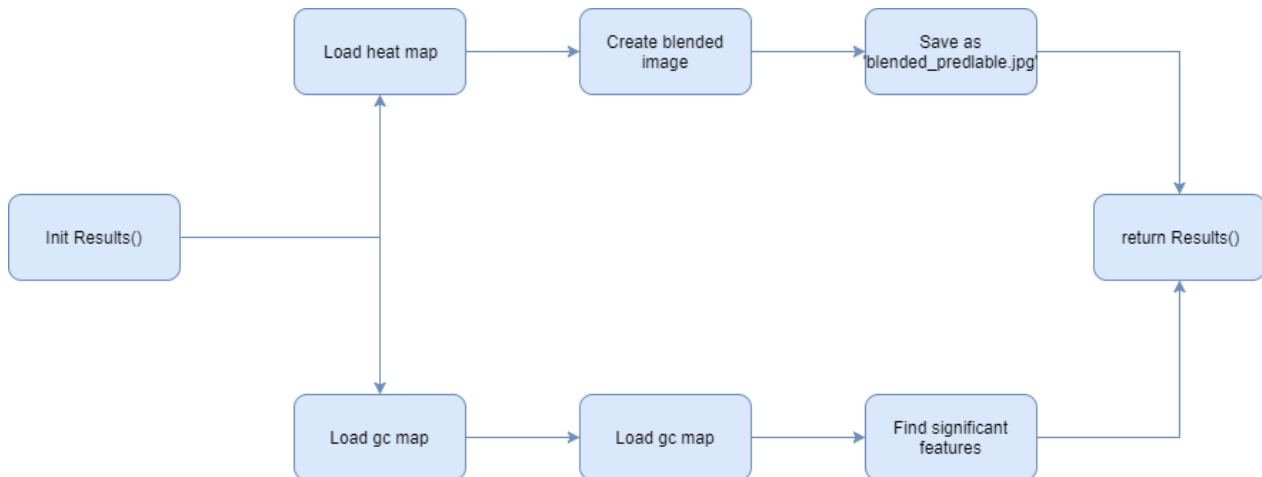
- Numpy
- cv2 (OpenCV)
- OS (Standard library)

##### 1.4.1. Algorithm Outline

- Load gc\_map\_predlabel.png
- Load heat\_map\_predlabel.png
- Set score threshold
- For each feature in cascades:
  - o Find feature in original image
  - o Compute score using the gc map:

Mean pixels value of the area detected divided by 255 (pixels' values are in greyscale)

- Save all features that passed the threshold
- Create 'blended\_predlabel.jpg' which is the heat map blended with the original image.



## 1.5. Python and Torch Integration

Face detection and feature extraction both implemented in Python. On the other hand, VGG16-Face descriptor and GradCAM are implemented in Torch7.

To bridge this gap, we used the 'subprocess' module in Python to run a command from the command line. The 'subprocess' allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

```

5  def run_torch_from_cmd():
6      try:
7          rv = subprocess.check_output(["th", "classification.lua"])
8          return rv
9      except subprocess.CalledProcessError as e:
10         message(e)
11         return None
12

```

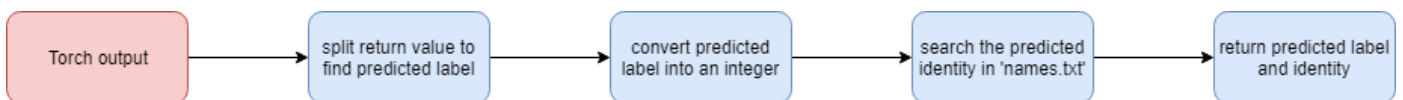
In this code, 'rv' is a string that contains the predicted label of the VGG classification.

The GradCAM output (i.e. gc map and head map) are saved as explained in 1.3.1 section and can be restored using only the predicted label of the VGG classification.

## 1.6. Support Functions

We used several support function to ease our core algorithms flow.

One method worth mentioning is `get_prediction_from_names(torch_output)`. It receives the output from `classification.lua` (VGG classification) as explained in 1.5 section. The method is splitting the output to find the predicted label. Then it converts the predicted label into an integer and pulling the identity name from the 'names.txt' file (out of 2622 celebrities). Finally it returns the predicated label and predicted identity.



## 1.7. Errors

As part of the 'Results' object returned from our core, we included error code and error message that are changed in case of an error along the way. The messages are printed to 'stderr' (Standard Error, stderr is another output stream typically used by programs to output error messages or diagnostics)

Error Code	Error Message	Reason
1	"ERROR: cannot load input image ... "	Could not load image from server
2	"ERROR: cannot load cascades from: ... "	Could not load cascades from OpenCV data directory
3	"ERROR: sorry, frontal face wasn't detected"	Image does not contain any face / could not find face in image
4	"Error in forward pass / GRAD-CAM, check torch file"	Torch file run from command line failed. Check stderr for more details.
5	"ERROR: could not load names.txt file"	Failed to open names.txt / predicted label has no identity in names.txt
6	"ERROR: No significant features found"	Failed to load features cascades / None was significant, possibly threshold is too high

**Table 1:** Error codes and error messages

## 2. Web Application

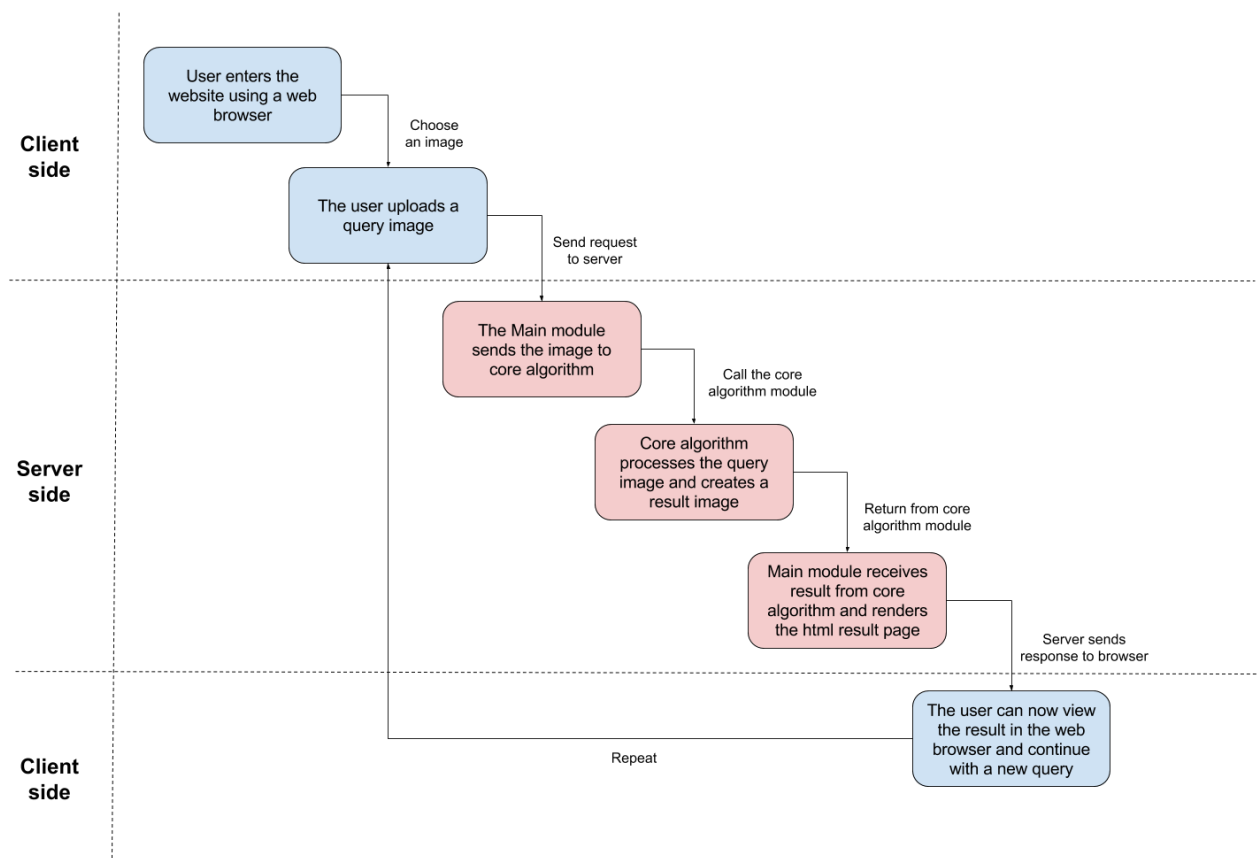
In this chapter, we will describe the web application, which provides the users interface that wraps the core functions of the project and allows sending queries to the system.

The web based application is available both for desktop and mobile devices.

### 2.1. General module design and user flow

The web application was created using the Django framework (Python-based development framework), which provides both client-side and server-side functions.

A user uploads an image and submit it using a designated form in the web page. A request is then sent to the server, which process the query image and sends it as input to the core algorithm of the program. The result is then returned to the Django server which renders a result web page that is sent as a response to the user.



## 2.2. Django Framework

### 2.2.1. Introduction

Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. The Django Software Foundation maintains it, an independent organization established as a 501 non-profit.

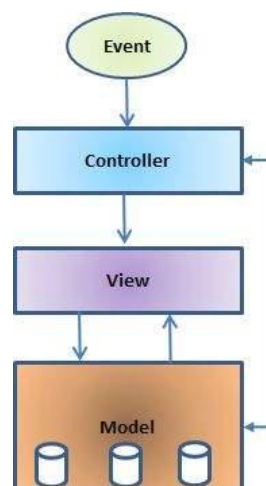
### 2.2.2. System Architecture

The Django frameworks follows the MVT pattern, which is adapted from the more common Model-View-Controller pattern.

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

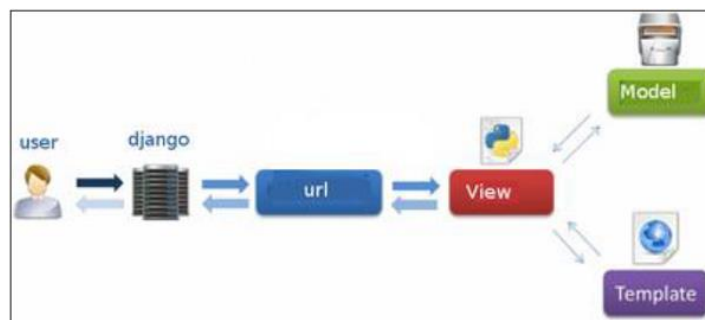
- Model - The lowest level of the pattern which is responsible for maintaining data.
- View - This is responsible for displaying all or a portion of the data to the user.
- Controller - Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.



The Model-View-Template (MVT) is slightly different from MVC (Model-View-Controller, a common pattern for. In fact the main difference between the two patterns is that Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View), leaving us with the template. The template is a HTML file mixed with Django Template Language (DTL).

The following diagram illustrates how each of the components of the MVT pattern interacts with each other to serve a user request. The developer provides the Model, the view, the template then just maps it to a URL, and Django does the magic to serve it to the user.



### 2.2.3. Installation

- On a Unix based platform, open a terminal and run the command:  
`>> pip install Django`  
Note: if *pip* is not installed on your machine, refer to its documentation and install the latest version
- Copy the entire "*brain*" folder to your server.

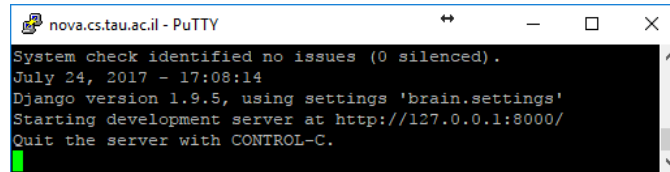
### 2.2.4. Running the sever

- On the server, open a terminal and navigate to the "*brain*" directory.
- Run the following command:  
`>> python manage.py runserver <server name>:<port number>`

#### Notes:

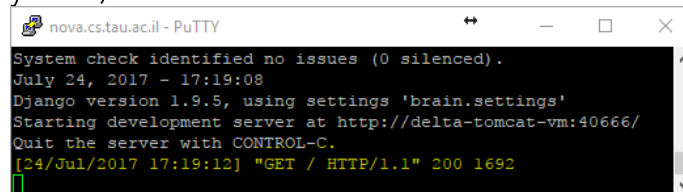
- o The server name and port number are optional and can be obtained by contacting your server manager.
- o If you run a server on a local machine, this can be omitted you can reach the web app by navigating on your local browser to <http://127.0.0.1:8000/>

- After running the command, you should see the following status:



```
nova.cs.tau.ac.il - PuTTY
System check identified no issues (0 silenced).
July 24, 2017 - 17:08:14
Django version 1.9.5, using settings 'brain.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

- From this point, any http request from a web browser is sent the server and handled by Django (highlighted in yellow):



```
nova.cs.tau.ac.il - PuTTY
System check identified no issues (0 silenced).
July 24, 2017 - 17:19:08
Django version 1.9.5, using settings 'brain.settings'
Starting development server at http://delta-tomcat-vm:40666/
Quit the server with CONTROL-C.
[24/Jul/2017 17:19:12] "GET / HTTP/1.1" 200 1692
```

- Stopping the server is done by the command Ctrl+C in the terminal.

### 2.2.5. Modules

#### "Main"

- The "Main" module consists of the functions that renders the main homepage to an html file and handles query request:
- Forms.py – defines the form fields that are used in the query
- Urls.py – translates the path from the browser to the path in the server filesystem.
- Views.py – collection of functions that renders the web page and handles the form requests. From this module, the core algorithm is called and the results are returned to.

### 2.3. Hosting

As previously mentioned, the application can be hosted on any Unix based server (or local machine).

It is currently hosted on CS-TAU TOMCAT server and can be reached by navigating to the link: <http://delta-tomcat-vm.cs.tau.ac.il:40666>

#### Note:

Port number 40666 is the default port for the Django server on TOMCAT. Please contact us before accessing the application to make sure the server is up and that the port number is up to date.