



Camnets

CS504 Programming Languages for Data Analysis
Final Project

Author: Daniel Correa Tucunduva

Professor: Dr. Rami Yared

March 31, 2019

Table of Contents

Introduction.....	3
High level architecture diagram of <i>Camnets</i>	5
Technology stack.....	6
JavaScript.....	6
Server-side stack.....	6
Node.js.....	6
Express.js.....	7
Client-side stack.....	7
Angular 7.....	7
Tensorflow.js.....	8
Google Chrome.....	8
Neural network high level overviews	9
Yolo.....	9
Mobilenet.....	11
Implementation details of image processing in <i>Camnets</i>	14
Face tracking pipeline	14
Keypoints estimation pipeline	15
User interface	16
About.....	16
Face tracking.....	17
Keypoints estimation	18
Source code and live application	19

Introduction

The contemporary application of artificial neural networks (ANN) has been shifting the landscape of data analysis.

Among the many problems that ANNs can solve, an area of emphasis is image analysis.

For several image analysis tasks, such as object detection and image interpretation, ANNs are the current state-of-the-art standard ¹.

Support for the use of ANNs in browser-based, client-side applications is in its very infancy: Google just released in 2018 ² tensorflow.js, the JavaScript implementation of the highly popular TensorFlow machine learning library.

The use of ANNs in *browser-based* applications offers the benefit of very high portability, since web browsers are ubiquitous.

The use of ANNs in *client-side* applications offers four compelling advantages:

- consumption of hardware resources on the client side, instead of centralizing the burden on the server side
- minimized network traffic, since once loaded, the application runs independently from the server
- zero latency during execution
- complete privacy, since the user data never leaves the client device

¹ <http://cs231n.stanford.edu/>

² <https://medium.com/tensorflow/introducing-tensorflow-js-machine-learning-in-javascript-bf3eab376db>

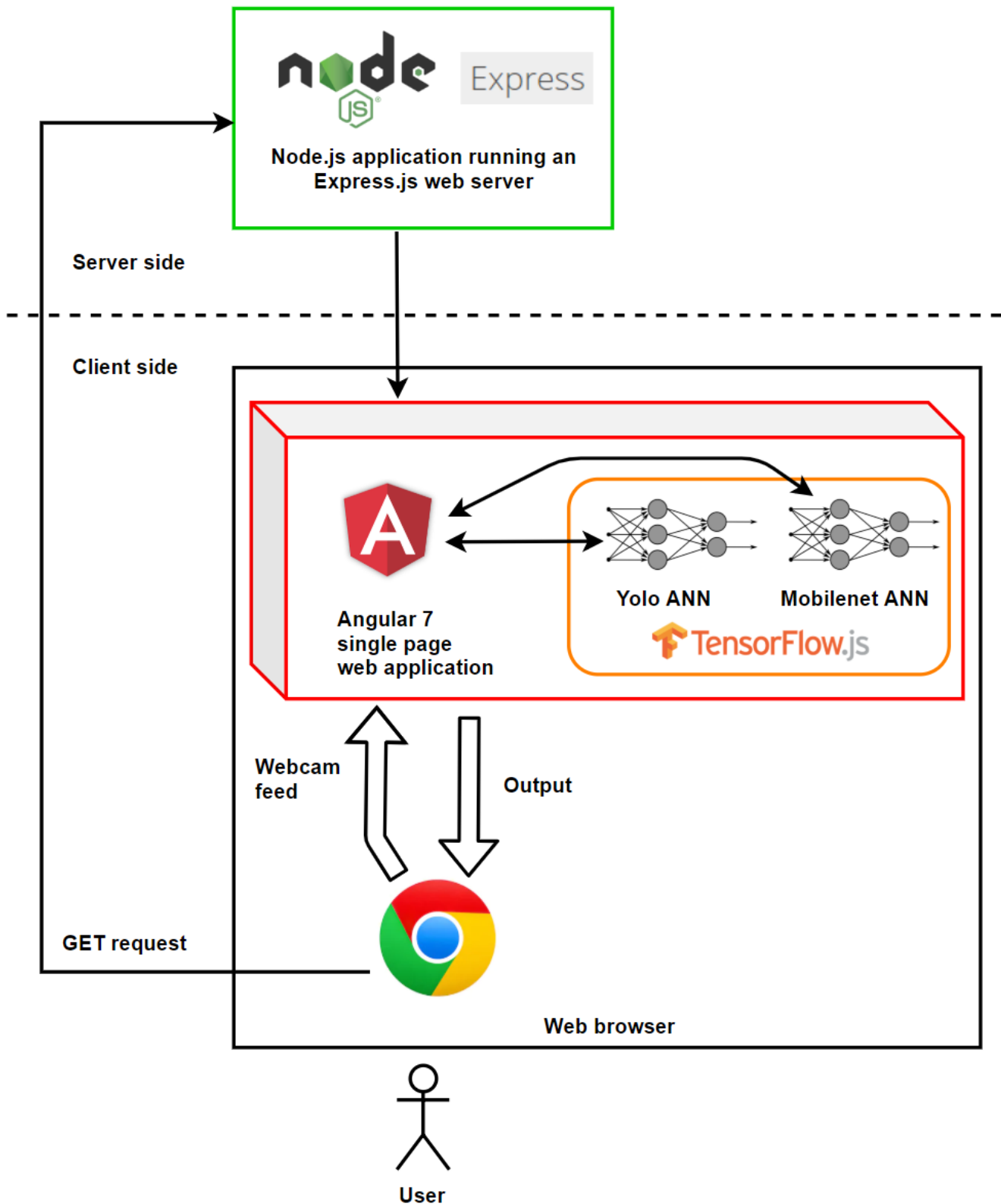
This project provides an implementation of a client-side, browser-based single page web application that utilizes a live webcam feed as input. The application can perform two different tasks:

- **face tracking**
- **human pose estimation**

The core of the image processing computations is accomplished by utilizing two specialized, extensively trained ANN instances.

Since the runtime environment is the client's web browser, the selected ANNs have a lightweight architecture, allowing for real-time or near real-time results on consumer level hardware, including on mobile platforms.

High level architecture diagram of *Camnets*



Technology stack

JavaScript

JavaScript ³ is a high-level, multi-paradigm, dynamically typed programming language that conforms to the ECMAScript ⁴ international standard. It has the unique quality of being the only programming language with a long history of native support in web browsers, and it continues to be the exclusive standard for frontend web development. With the creation of Node.js, JavaScript has been quickly gaining popularity as a backend language as well.

Camnets is entirely based on JavaScript, both in the server side and client side.

Server-side stack

Node.js

Node.js ⁵ is an open-source, cross-platform, lightweight runtime that executes JavaScript outside of a web browser, with a single-thread, event-drive, non-blocking I/O paradigm. It uses Chrome's V8 JavaScript engine ⁶, the same engine used by Google Chrome.

Camnets uses Node.js in the server side to run an Express.js web server, responsible for serving the client-side Angular application.

³ <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

⁴ <https://en.wikipedia.org/wiki/ECMAScript>

⁵ <https://nodejs.org/en/>

⁶ <https://v8.dev/>

Express.js

Express.js ⁷ is a free open-source web application framework for Node.js that is fast, flexible, minimalist and unopinionated. It is the current industry standard for Node.js web servers.

Camnets uses an Express.js web server that receives HTTP GET requests and responds by serving the Angular 7 application.

Client-side stack

Angular 7

Angular 7 ⁸ is an open-source web application framework that provides support for building multiplatform applications, using a modular structure and a single-page application ⁹ paradigm. It is the latest iteration of the Angular 2+ family (usually called simply Angular) that started with Angular 2, a complete rewrite of the AngularJS framework. Both AngularJS and the Angular 2+ family were developed and by the Angular Team at Google.

Angular utilizes TypeScript ¹⁰, an open-source statically typed programming language developed and maintained by Microsoft, that is a strict syntactical superset of JavaScript (in rough terms, it is “JavaScript with static typing”). TypeScript is used for development only and is transcompiled ¹¹ into JavaScript at build time. Static typing facilitates the development of large-scale applications by moving most type related errors from runtime (a common issue in JavaScript applications) to build time.

⁷ <https://expressjs.com/>

⁸ <https://angular.io/>

⁹ https://en.wikipedia.org/wiki/Single-page_application

¹⁰ <https://www.typescriptlang.org/>

¹¹ https://en.wikipedia.org/wiki/Source-to-source_compiler

On the client side, *Camnets* is a pure Angular 7 application.

Tensorflow.js

TensorFlow.js ¹² is the JavaScript porting of TensorFlow ¹³, a free and open-source symbolic math library developed by the Google Brain team for training and deploying machine learning models, that has become an industry standard. TensorFlow.js is the first machine learning library to provide support for hardware acceleration in the browser, by using WebGL ¹⁴ for computations.

Camnets makes use of TensorFlow.js to support the two neural networks that perform the core of the image processing computations.

Google Chrome

Google Chrome is a proprietary, freeware, cross-platform web browser developed by Google. It is by far the most popular web browser, with a market share of over 70% for desktops in 2018 ¹⁵ and over 60% ¹⁶ for all platforms in 2019, and offers a robust built-in developer tools console, making it the current standard for development and testing of web applications.

Camnets is optimized and tested for use with Google Chrome.

¹² <https://github.com/tensorflow/tfjs>

¹³ <https://www.tensorflow.org/>

¹⁴ https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

¹⁵ <https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/>

¹⁶ <https://www.w3counter.com/globalstats.php?year=2019&month=2>

Neural network high level overviews

Yolo

YOLO ¹⁷ (You Only Look Once) is a neural network architecture that performs object detection by using a single convolutional network to predict bounding boxes and class probabilities simultaneously. This requires only a single pass of the image over the network, giving the architecture its name.

Object detection is framed as a single regression problem, from image pixels to bounding boxes and class probabilities.

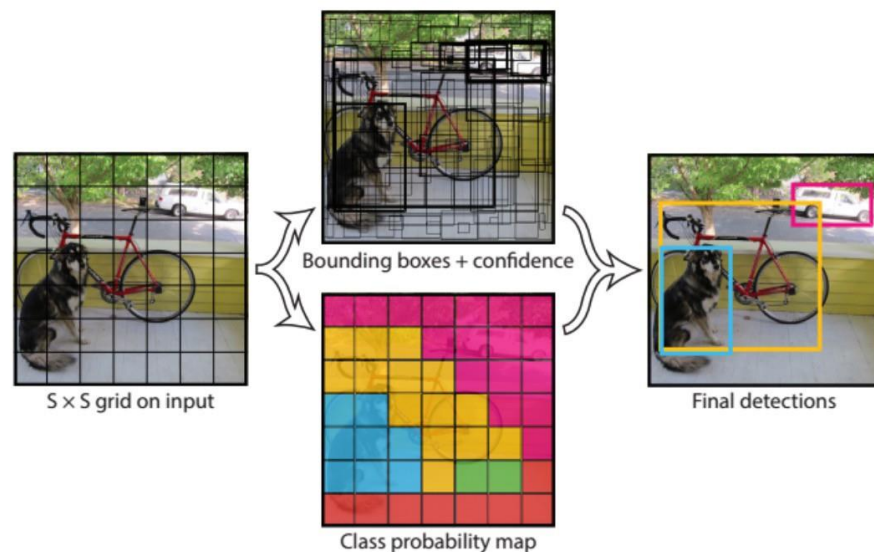
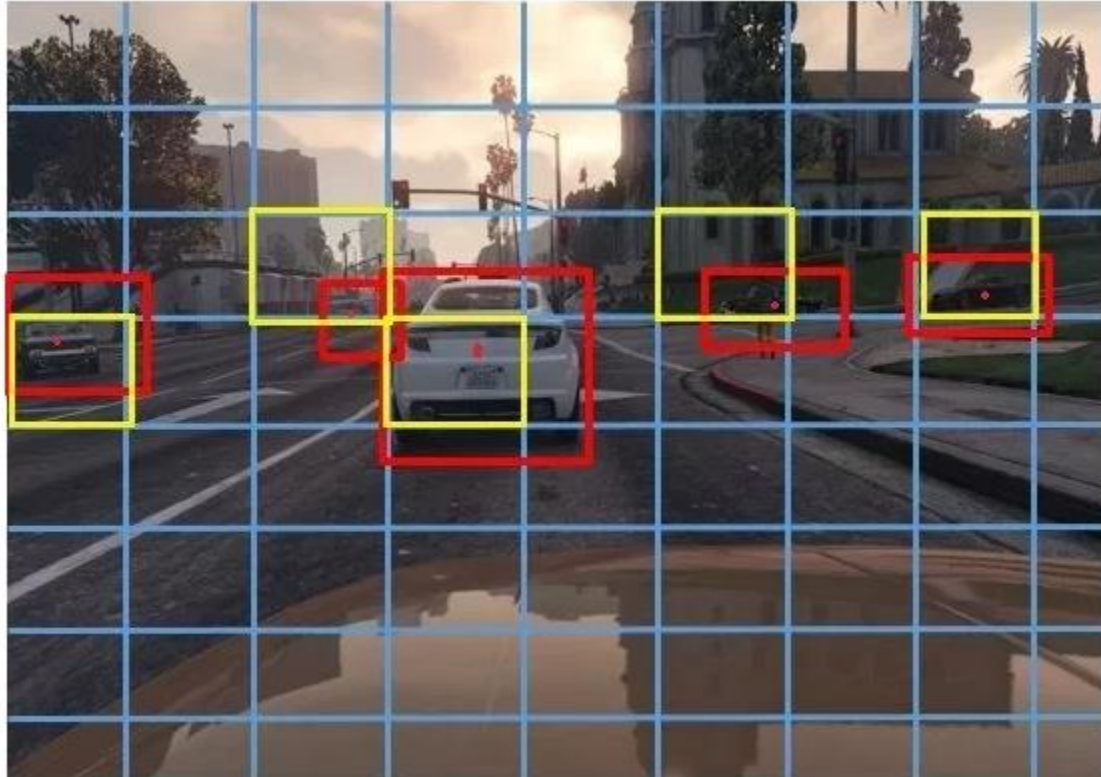


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Source: <https://arxiv.org/pdf/1506.02640.pdf>

¹⁷ <https://arxiv.org/abs/1506.02640>

At training time, bounding boxes (red rectangles, in the image below) are placed on the objects for detection in the training images. Each bounding box has four features: the x and y coordinates for the center of the object (red dots, in the image below) and the bounding rectangle height and width. A grid is then placed on the whole image, and each object is assigned to the grid box that contains its center (yellow boxes, in the image below).



Source: <http://ramok.tech/tag/tiny-yolo/>

At prediction time, input images are divided into a grid, and the grid cell that contains the center of each object is responsible for its detection. The predicted bounding box is then defined in relation to the grid box that contains the center of the object.

Camnets computes bounding boxes for face tracking using an instance of a specialized lightweight YOLO (fewer hidden layers than the original) trained to detect human faces ¹⁸.

¹⁸ <https://github.com/justadudewhohacks/tfjs-tiny-yolov2>

Mobilenet

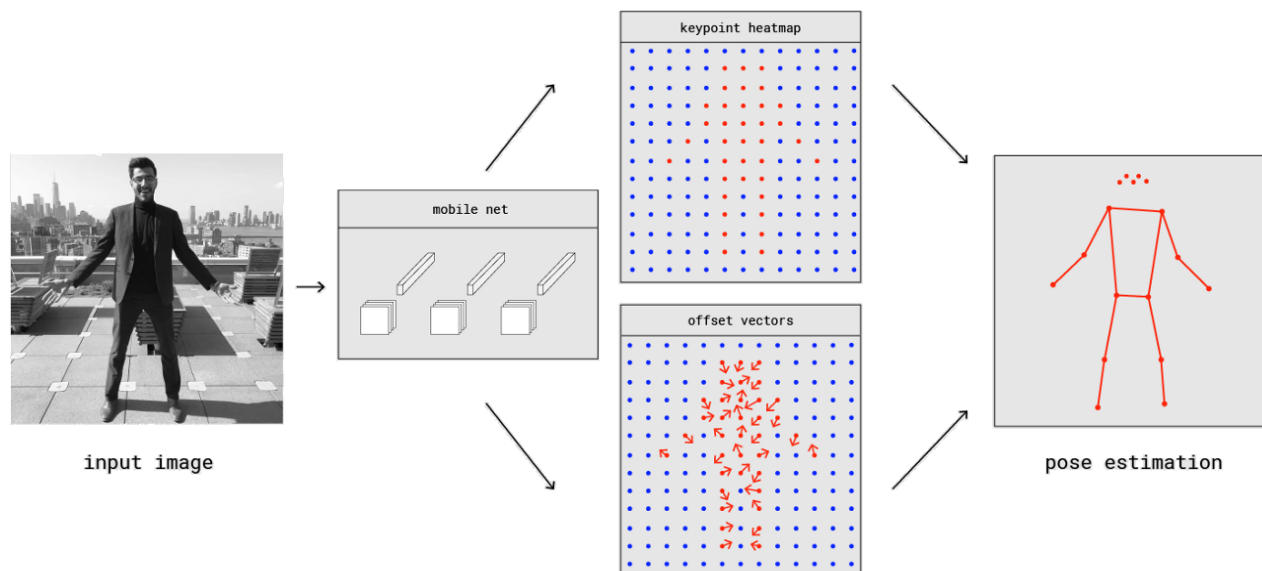
PoseNet¹⁹ is an instance of the Mobilenet²⁰ architecture trained to identify 17 *keypoints*:

- nose
- left eye
- right eye
- left ear
- right ear
- left shoulder
- right shoulder
- left elbow
- right elbow
- left wrist
- right wrist
- left hip
- right hip
- left knee
- right knee
- left ankle
- right ankle

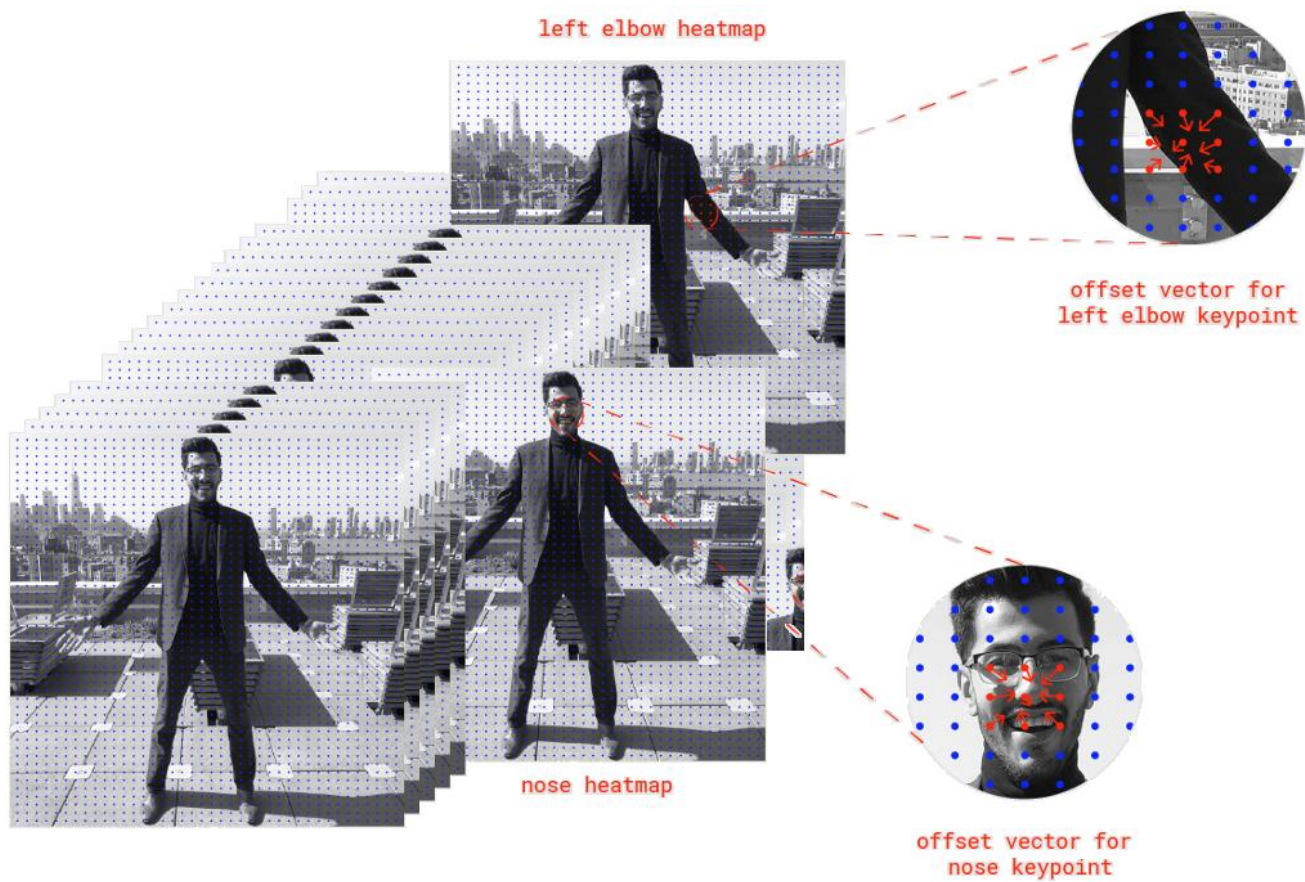
The neural network produces a heatmap and two offset vector maps (for the x and y coordinates) for each *keypoint* (17 heatmaps + 34 offset maps total), which are then used to determine the coordinates of the high confidence *keypoints* in the image.

¹⁹ <https://github.com/tensorflow/tfjs-models/tree/master/posenet>

²⁰ <https://arxiv.org/abs/1704.04861>

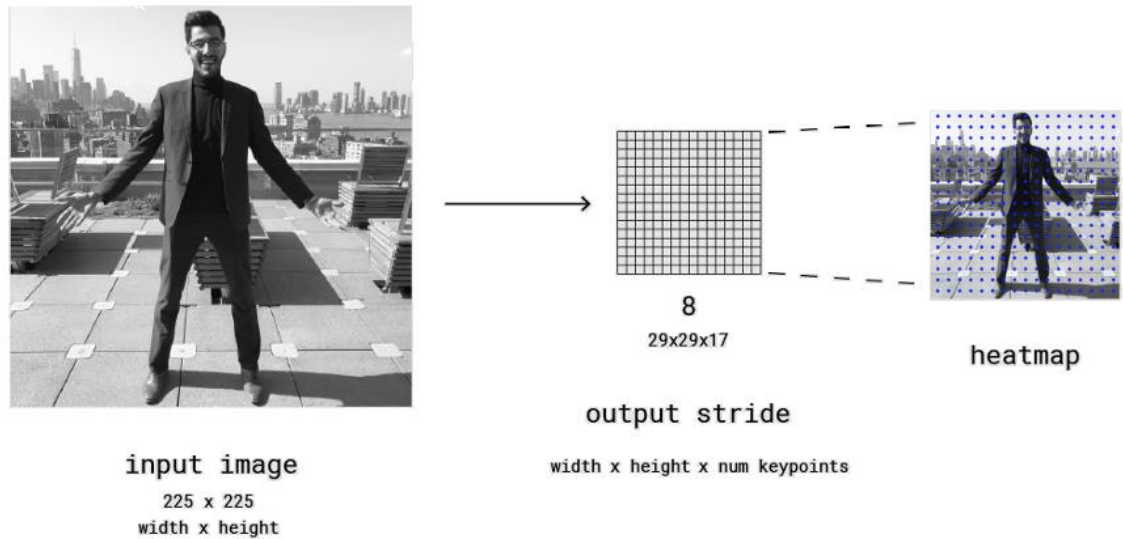


Adapted from: https://cdn-images-1.medium.com/max/1400/1*ey139jykinBzUqcknAjHGQ.png



Adapted from: https://cdn-images-1.medium.com/max/1400/1*mcaovEoLBt_Aj0lwv1-xtA.png

MobileNet uses convolution with striding ²¹ to reduce the resolution of the input image, so the internal layers of the network have a scaled down size compared to the input, reducing computational cost at the expense of accuracy.



Adapted from: https://cdn-images-1.medium.com/max/800/1*zXXwR16kprAWLPIOKCrXLw.png

To determine the *keypoints* in the final output, each heatmap coordinate is multiplied by the stride and then added to the corresponding offset vector.

The final output is a set of poses. Each pose contains an overall confidence score (mean of the scores of its *keypoints*) and an array of 17 elements. Each element contains the label, coordinates and confidence score of one *keypoint*.

Camnets uses a PoseNet instance to compute keypoint coordinates.

²¹ For an explanation of convolution and striding, see: <https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/> and <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>

Implementation details of image processing in *Camnets*

Face tracking pipeline

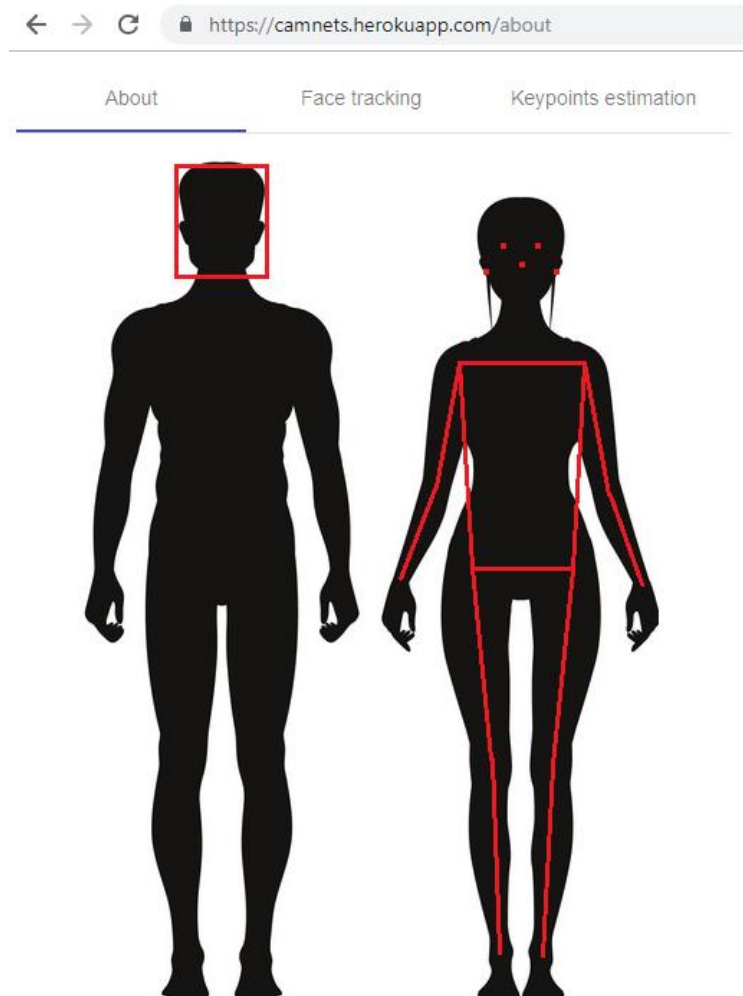
- the webcam feed is started, and captured in a hidden `<video>` HTML Element:
 - the video input is transposed to a **visible `<canvas>`** HTML Element
 - meanwhile, the Yolo ANN is loaded in the background
 - when the ANN finishes loading the **Start face tracking** button is enabled
- once the Start button is clicked:
 - the intended **FPS for tracking** is captured
 - given the intended FPS, the execution interval for frame processing is calculated
- frame capture for processing is intermittent, according to the calculated interval
- at the end of each interval:
 - the execution lock is acquired, and an execution cycle starts
 - each frame captured undergoes a single pass over the ANN
 - the ANN results are filtered according to the user-selected **Face confidence** threshold (faces with a score lower than the threshold are removed)
 - a hidden sketch `<canvas>` is drawn with the original input
 - red bounding boxes are drawn around faces in the sketch `<canvas>`
 - the sketch `<canvas>` is transposed to the visible `<canvas>`
 - the processed frame count is incremented by one
 - the execution lock is released
- if at the end of an interval the previous execution cycle is not finished, the current cycle is skipped
- once per second, the processed frame count is transposed as the **Effective FPS** output, and then reset to zero
- when the Stop button is clicked, the capture for processing is terminated, and the visible `<canvas>` is reset to receive the transposed `<video>` input

Keypoints estimation pipeline

- the webcam feed is started, and captured in a hidden <video> HTML element:
 - the video input is transposed to a **visible <canvas>** HTML element
 - meanwhile, the PoseNet ANN is loaded in the background
 - when the ANN finishes loading the **Start pose estimation** button is enabled
- once the Start button is clicked:
 - the intended **FPS for estimation** is captured
 - given the intended FPS, the execution interval for frame processing is calculated
- frame capture for processing is intermittent, according to the calculated interval
- at the end of each interval:
 - the execution lock is acquired, and an execution cycle starts
 - each frame captured undergoes a single pass over the ANN
 - the ANN results are filtered according to the user-selected **Pose confidence** threshold (poses with an overall score lower than the threshold are removed)
 - for each remaining pose, the keypoints are filtered according to the user-selected **Keypoint confidence** threshold (keypoints with a score lower than the threshold are removed)
 - a hidden sketch <canvas> is drawn with the original input
 - red dots are drawn on keypoint coordinates in the sketch <canvas>
 - the applicable keypoint-to-keypoint lines (12 possible in total, for all adjacent keypoint pairs that are not in the face) are drawn in the sketch <canvas>
 - the sketch <canvas> is transposed to the visible <canvas>
 - the processed frame count is incremented by one
 - the execution lock is released
- if at the end of an interval the previous execution cycle is not finished, the current cycle is skipped
- once per second, the processed frame count is transposed as the **Effective FPS** output, and then reset to zero
- when the Stop button is clicked, the capture for processing is terminated, and the visible <canvas> is reset to receive the transposed <video> input

User interface

About



Camnets v 1.0

Author: Daniel Correa Tucunduva

Developed as the final project for CS504

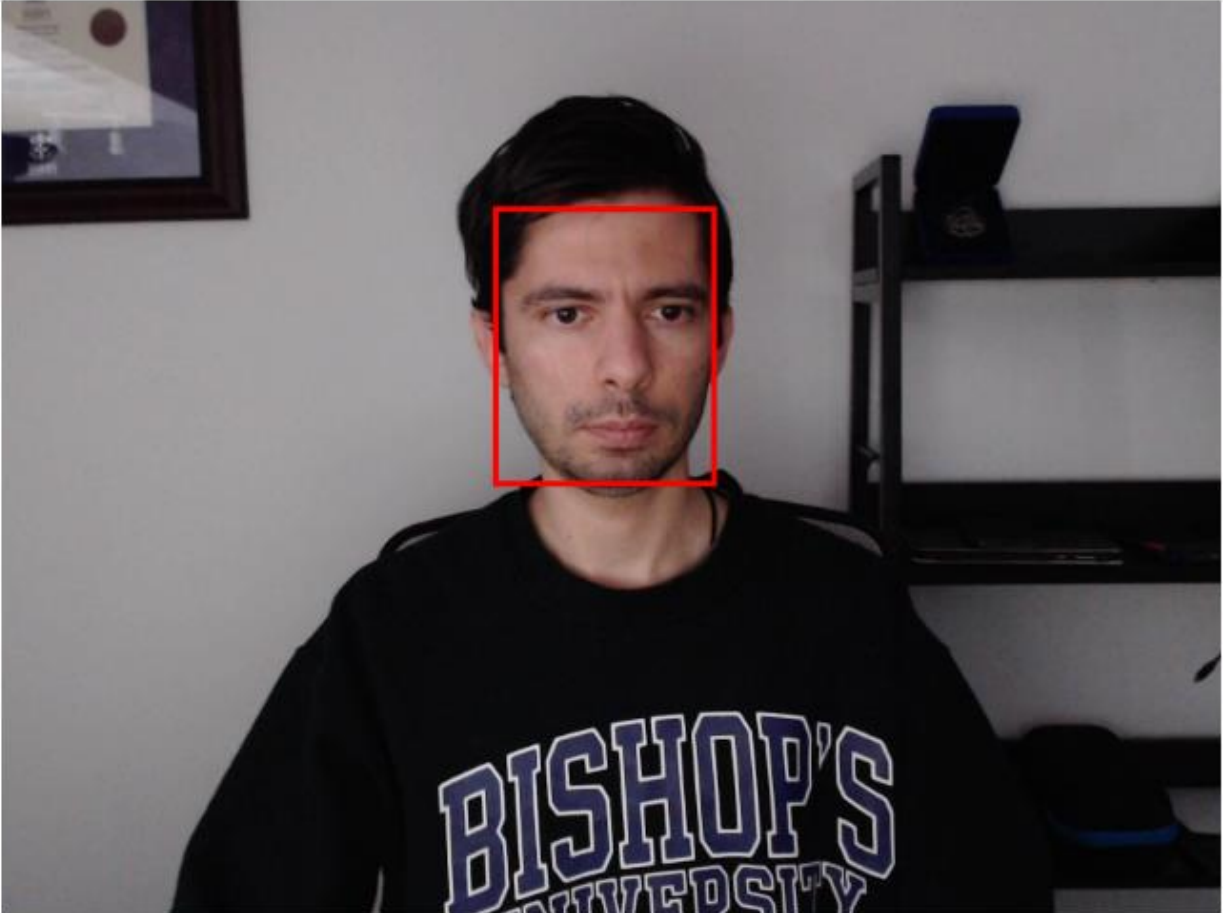
Professor: Dr. Rami Yared

Bishop's University, Winter 2019

Face tracking

[←](#) [→](#) [↻](#) <https://camnets.herokuapp.com/face-tracking>

[About](#) [Face tracking](#) [Keypoints estimation](#)



FPS for tracking

10

1 to 30

Face confidence

0.4

0.1 to 1.0

Stop face tracking

Effective FPS: 9

Keypoints estimation

← → ↻ 🔒 https://camnets.herokuapp.com/keypoints

About

Face tracking

Keypoints estimation



FPS for estimation

5

1 to 30

Pose confidence

0.4

0.1 to 1.0

Keypoint confidence

0.5

0.1 to 1.0

Stop pose estimation

Effective FPS: 3

Source code and live application

The complete source code is publicly available on Github, along with a readme file with instructions to run the project locally, at:

<https://github.com/danieltucunduva/camnets.git>

The application has been deployed and made available live at:

<https://camnets.herokuapp.com/>