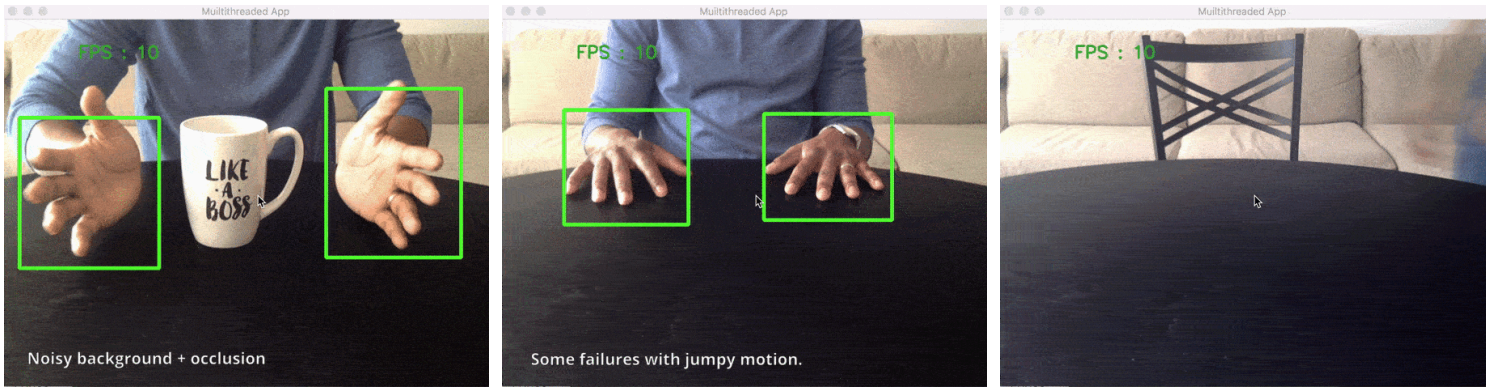
# How to Build a Real-time Hand-Detector using Neural Networks (SSD) on Tensorflow

Victor Dibia · Follow
11 min read · Dec 1, 2017



TLDR: We train a model to detect hands in real-time (21fps) using the Tensorflow Object Detection API.




Detection on live video from a webcam. (a) Detection works well even with partial occlusions. (b) There were some misses when motion was fast and hands were from an unlikely egocentric viewpoint. (c ) Detection worked fairly well even with object overlap.

This post documents steps and scripts used to train a hand detector using Tensorflow (Object Detection API). I was interested mainly in detecting hands on a table. And in real time. Hopefully this post demonstrates how neural networks can be applied to the problem of tracking hands (egocentric and other views) with good results.

All of the code (including the frozen model) are now available on Github.

**victordibia/handtracking**

Building a Real-time Hand-Detector using Neural Networks (SSD) on Tensorflow - victordibia/handtracking

github.com

As with any DNN/CNN based task, the most expensive (and riskiest) part of the process has to do with finding or creating the right (annotated) dataset. I experimented first with the Oxford Hands Dataset (the results were not good). I then tried the Egohands Dataset which was a much better fit to my requirements (egocentric view, high quality images, hand annotations).

Some fps numbers:

- **21 FPS** using a 320 * 240 image, run without visualizing results
- **16 FPS** using a 320 * 240 image run while visualizing results
- **11 FPS** using a 640 * 480 image run while visualizing results (image above)

Above numbers are based on tests using a macbook pro **CPU** (i7, 2.5GHz, 16GB).

## Motivation — Why Track/Detect hands with Neural Networks?

There are several existing approaches to tracking hands in the computer vision domain. Incidentally, many of these approaches are rule based (e.g. extracting background based on texture and boundary features, distinguishing between hands and background using color histograms and HOG classifiers, etc) making them not very robust. For example, these algorithms might get confused if the background is unusual or where sharp changes in lighting conditions cause sharp changes in skin color or the tracked object becomes occluded. (see here for a review paper on hand pose estimation from the HCI perspective).

With sufficiently large datasets, neural networks provide opportunity to train models that perform well and address challenges of existing object tracking/detection algorithms — varied/poor lighting, diverse viewpoints and even occlusion. The main drawbacks to usage for real-time tracking/detection is that they can be complex, are relatively slow compared to tracking-only algorithms and it can be quite expensive to assemble a good dataset. But things are changing with advances in fast neural networks.

Furthermore, this entire area of work has been made more approachable by deep learning frameworks (such as the tensorflow object detection api) that simplify the process of training a model for custom object detection. More

importantly, the advent of fast neural network models like ssd, faster r-cnn, rfcn (see here ) etc make neural networks an attractive candidate for real-time detection (and tracking) applications. There are multiple applications for robust hand tracking like this across HCI areas (as an input device etc.).

> *If you are **not** interested in the process of training the detector, you can skip straight to the section on applying the model to detect hands.*

Training a model is a multi-stage process (assembling dataset, cleaning, splitting into training/test partitions and generating an inference graph). While I lightly touch on the details of these parts, there are a few other tutorials which cover training a custom object detector using the tensorflow object detection api in more detail (see here and here). I recommend you walk through those if interested in training a custom detector from scratch.

## Data preparation and network training in Tensorflow

### The Egohands Dataset

The hand detector model is built using data from the Egohands Dataset dataset. This dataset works well for several reasons. It contains high quality, pixel level annotations (>15000 ground truth labels) where hands are located across 4800 images. All images are captured from an egocentric view (Google glass) across 48 different environments (indoor, outdoor) and activities (playing cards, chess, jenga, solving puzzles etc). If you will be using the Egohands dataset, you can cite them as follows:

> *Bambach, Sven, et al. "Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions." Proceedings of the IEEE International Conference on Computer Vision. 2015.*


The Egohands dataset provide a polygon (the white dots) around each hand. We need to generate bounding boxes from the polygons, and generate tfrecords to train a tensorflow model.

The Egohands dataset (zip file with labelled data) contains 48 folders of locations where video data was collected (100 images per folder).

```
– LOCATION_X
  – frame_1.jpg
  – frame_2.jpg
  …
  – frame_100.jpg
  – polygons.mat // contains annotations
– LOCATION_Y
  – frame_1.jpg
  – frame_2.jpg
  …
  – frame_100.jpg
  – polygons.mat // contains annotations
```

**Converting data to Tensorflow Format**

Some initial work needs to be done to the Egohands dataset to transform it into the format (*tfrecord*) which Tensorflow needs to train a model. The Github repo contains *egohands_dataset_clean.py* a script that will help you generate these csv files.

- Downloads the egohands datasets
- Renames all files to include their directory names to ensure each filename is unique
- Splits the dataset into train (80%), test (20%) folders.
- Reads in `*polygons.mat*` for each folder, generates bounding boxes and visualizes them to ensure correctness.
- Once the script is done running, you should have an images folder containing two folders - train, and test. Each of these folders should also contain a csv label document each - `train_labels.csv`, `test_labels.csv` that can be used to generate `tfrecords`.

Next: convert your dataset + csv files to tfrecords. Please use the guide provided by Harrison from pythonprogramming on how to generate tfrecords given your label csv files and your images. The guide also covers how to start the training process if training locally. If training in the cloud using a service like GCP, see the guide here.

Note: While the egohands dataset provides four separate labels for hands (own left, own right, other left, and other right), for my purpose, I am only

interested in the general `hand` class and label all training data as `hand`. You can modify the train script to generate `tfrecords` that support 4 labels.

**Training the hand detection Model**

Now that the dataset has been assembled, the next task is to train a model based on this. With neural networks, it is possible to use a process called transfer learning to shorten the amount of time needed to train the entire model. This means we can take an existing model (that has been trained well on a related domain (here image classification) and retrain its final layer(s) to detect hands for us. Sweet!. Given that neural networks sometimes have thousands or millions of parameters that can take weeks or months to train, transfer learning helps shorten training time to possibly hours. Tensorflow does offer a few models (in the tensorflow model zoo) and I chose to use the `ssd_mobilenet_v1_coco` model as my start point given it is currently (one of) the fastest models (see the research paper on SSD here, see a comparison between SSD and others e.g Yolo here.). The training process can be done locally on your CPU machine which may take a while, or better on a (cloud) GPU machine (which is what I did). For reference, training on my macbook pro (tensorflow compiled from source to take advantage of the mac's cpu architecture) the maximum speed I got was 5 seconds per step as opposed to the ~0.5 seconds per step I got with a GPU. For reference it would take about 12 days to run 200k steps on my mac (i7, 2.5GHz, 16GB) compared to ~5hrs on a GPU.

As the training process progresses, the expectation is that total loss (errors) gets reduced to its possible minimum (about a value of 1 or lower). By observing the tensorboard graphs for total loss(see image below), it should be possible to get an idea of when the training process is complete (total loss does not decrease with further iterations/steps). I ran my training job for 200k steps (took about 5 hours) and stopped at a total Loss (errors) value of **2.575**.(In retrospect, I could have stopped the training at about 50k steps and gotten a similar total loss value). With tensorflow, you can also run an evaluation concurrently that assesses your model to see how well it performs on the test data. A commonly used metric for performance is mean average precision (mAP) which is single number used to summarize the area under the precision-recall curve. **mAP** is a measure of how well the model generates a bounding box that has at least a 50% overlap with the ground truth bounding box in our test dataset. For the hand detector trained here,


Final total loss value of **2.575** and mAP of **0.9686.**

Once training is completed, the trained inference graph (`frozen_inference_graph.pb`) is then exported (see the earlier referenced guides for how to do this) and saved in the `hand_inference_graph` folder. Now its time to do some interesting detection.

## Using the Detector to Detect/Track hands


Applying the detector to a Youtube video.

If you have not done this yet, please follow the guide on installing Tensorflow and the Tensorflow object detection api. This will walk you through setting up the tensorflow framework, cloning the tensorflow object detection repo and installing it.

The general steps to detect hands are as follows:

- Load the `frozen_inference_graph.pb` trained on the hands dataset as well as the corresponding label map.

- Read in your input image (this may be captured from a live video stream, a video file or an image).
- Detect hands and visualize detected bounding detection_boxes.

On GitHub, the provided repo contains two scripts that tie all these steps together.

- detect_multi_threaded.py : A threaded implementation for reading camera video input detection and detecting. Takes a set of command line flags to set parameters such as ` — display` (visualize detections), image parameters ` — width` and ` — height`, video ` — source` (0 for camera) etc.
- detect_single_threaded.py : Same as above, but single threaded. This script works for video files by setting the video source parameter videe ` — source` (path to a video file).

Most importantly, the repo contains a frozen_inference_graph.pb that contains a trained model based on SSD which you can easily import to your tensorflow applications to detect hands.

**Update**: Here's a link to a web based game where a player controls a paddle using hand tracking.



## Thoughts on Optimization.

A few things that led to noticeable performance increases.

- Threading: Turns out that reading images from a webcam is a heavy I/O event and if run on the main application thread, can slow down the program. I implemented some good ideas from Adrian Rosebuck on parallelizing image capture across multiple worker threads. This mostly led to an FPS increase of about 5 points.
- For those new to OpenCV, the `cv2.read()` method return images in [BGR format]. Ensure you convert to RGB before detection (accuracy will be

much reduced if you don't).

- Keeping your input image small will increase fps without any significant accuracy drop.(I used about 320 x 240 compared to the default1280 x 720 which my webcam provides).

Performance can also be increased by a clever combination of tracking algorithms with the already decent detection and this is something I am still experimenting with. Have ideas for optimizing , please share!


Performance on random images with hands show some limitations of the detector.

Note: The detector does reflect some limitations associated with the training set. This includes non-egocentric viewpoints, very noisy backgrounds (e.g in a sea of hands) and sometimes skin tone. There is opportunity to improve these with additional data.

**Integrating Multiple DNNs.**

One way to make things more interesting is to integrate our new knowledge of where "hands" are with other detectors trained to recognize other objects. Unfortunately, while our hand detector can in fact detect hands, it cannot detect other objects (a factor or how it is trained). To create a detector that classifies multiple different objects would mean a long involved process of assembling datasets for each class and a lengthy training process.

> *Given the above, a potential alternate strategy is to explore structures that allow us* ***efficiently*** *interleave output form multiple pretrained models for various object classes and have them detect multiple objects on a single image.*

An example of this is with my primary use case where I am interested in understanding the position of objects on a table with respect to hands on same table. I am currently doing some work on a threaded application that loads multiple detectors and outputs bounding boxes on a single image. More on this soon.

## Acknowledgements

This work also served as an intense weekend crash course for me to learn Python and Tensorflow. It would be impossible without the Egohands Dataset, many thanks to the authors! The tensorflow custom object detection guides by Harrison from pythonprogramming and Dat Tran were immensely helpful to this learning process. And ofcourse, many thanks to the Tensorflow authors! Its a great framework!

**Citing this tutorial**

If you'd like to cite this tutorial, use the below.

Victor Dibia, Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow, (2017), GitHub repository, https://github.com/victordibia/handtracking

@misc{Dibia2017,
author = {Victor, Dibia},
title = {Real-time Hand Tracking Using SSD on Tensorflow },
year = {2017},
publisher = {GitHub},
journal = {GitHub repository},
howpublished = {\url{https://github.com/victordibia/handtracking}},
commit = {b523a27393ea1ee34f31451fad656849915c8f42}
}

If you would like to discuss this in more detail, feel free to reach out on Twitter, Github or Linkedin.

**References**

Some related and referenced papers.

Bambach, S., Lee, S., Crandall, D. J., and Yu, C. 2015. "**Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions**," in *ICCV*, pp. 1949–1957 (available at https://www.cv-

foundation.org/openaccess/content_iccv_2015/html/Bambach_Lending_A_Hand_ICCV_2015_paper.html).

Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., and Twombly, X. 2007. "**Vision-based hand pose estimation: A review**," *Computer Vision and Image Understanding* (108:1–2), pp. 52–73 (doi: 10.1016/j.cviu.2006.10.012).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., and Berg, A. C. 2016. "**SSD: Single shot multibox detector**," in *European conference on computer vision* (Vol. 9905 LNCS), Springer Cham, pp. 21–37 (doi: 10.1007/978-3-319-46448-0_2).

Betancourt, A., Morerio, P., Regazzoni, C. S., and Rauterberg, M. 2015. "**The Evolution of First Person Vision Methods: A Survey**," *IEEE Transactions on Circuits and Systems for Video Technology* (25:5), pp. 744–760 (doi: 10.1109/TCSVT.2015.2409731)

J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE CVPR*, 2017, pp. 1–21.

Machine Learning | TensorFlow | Object Detection | Data Science | Towards Data Science

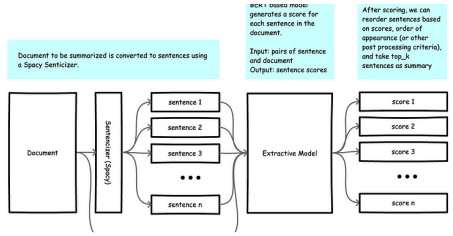
**Written by Victor Dibia**

1.1K Followers

HCI/Applied AI Researcher with interests in usable AI. Principal RSDE at Microsoft Research, Google Dev Expert for ML. Additional posts on victordibia.com

Follow

**More from Victor Dibia**

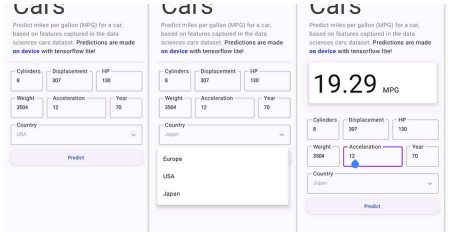



Victor Dibia

## Implementing Extractive Text Summarization with BERT- Issue...

3 min read · Aug 30, 2021

58



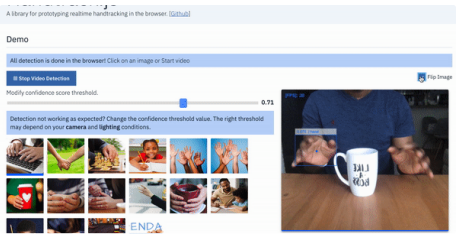

Victor Dibia

## Running Machine Learning Models on Android Devices—Issue #9

In this issue we will discuss how to run Tensorflow machine learning models locally...

5 min read · Jun 16, 2022

187 1



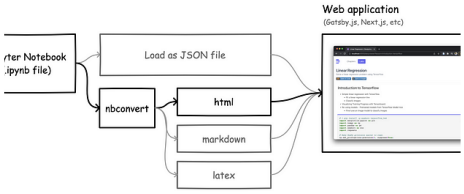

Victor Dibia in Towards Data Science

## Handtrack.js: Hand Tracking Interactions in the Browser using...

Handtrack.js library allows you track a user's hand (bounding box) from an image in any...

9 min read · Mar 5, 2019

1.96K 7




Victor Dibia

## Integrating Jupyter Notebooks in Web Applications—Issue #7

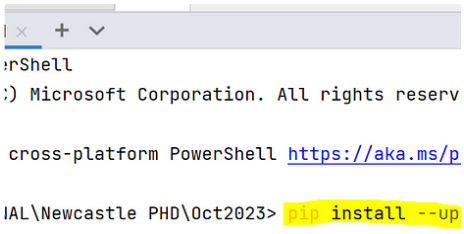
Hi everyone! Hope you are all doing well! In this newsletter issue, I'll discuss some of my...

4 min read · Mar 28, 2022

105

See all from Victor Dibia

## Recommended from Medium



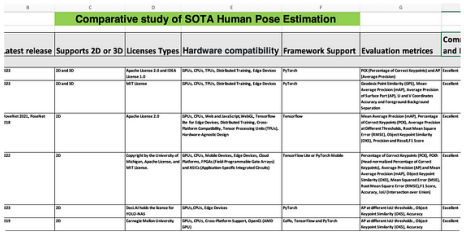

Elven Kim

## How to use Mediapipe Model Maker on local PC

With TensorFlow Model Maker kept crashing, I am sure many of us cannot help by groan in...

2 min read · Oct 26, 2023

51




Pallawi

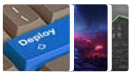
## Which Human pose estimation model should you pick to realise...

The SOTA models may look lucrative and exciting to a business but I do not want your...

8 min read · Dec 19, 2023

83

### Lists

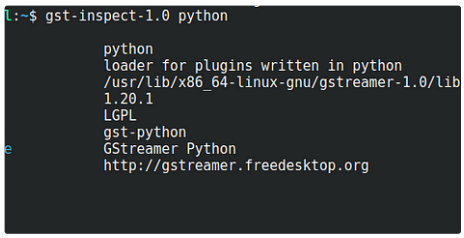



**Predictive Modeling w/ Python**
20 stories · 1069 saves

**Practical Guides to Machine Learning**
10 stories · 1282 saves

**Natural Language Processing**
1354 stories · 837 saves

**data science and AI**
40 stories · 122 saves



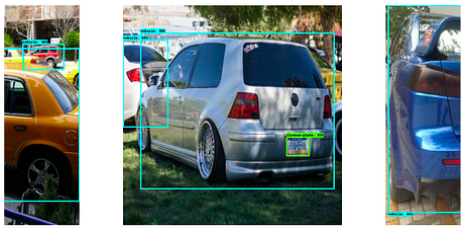

Jason Kim

## Writing GStreamer Plugin with Python

This post shows how to write GStreamer plugins in Python with examples.

3 min read · Jan 14, 2024

4

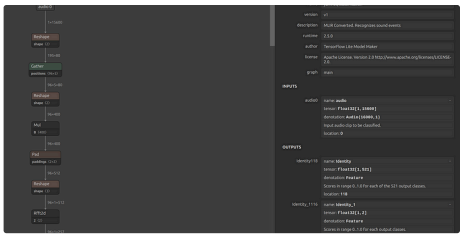


Sravan Neeli

## Car Number Plate Detection using Tensorflow Model Garden

A Step-by-Step Guide to Building and Deploying a Robust Object Detection Model...

6 min read · Nov 13, 2023



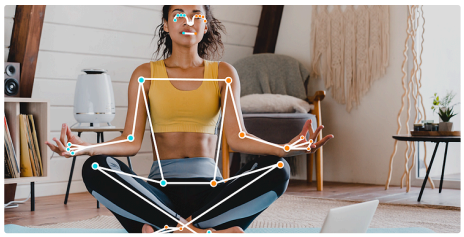
George Soloupis

## Use TensorFlow Lite Model Maker with a custom dataset

Written by George Soloupis ML and Android GDE.

6 min read · Nov 5, 2023

56



Arumugam

## Comparative study of SOTA Human pose estimation deep learning...

Deep Learning-based pose estimation algorithms have come a long way since the...

5 min read · Oct 13, 2023

50 1

See more recommendations