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

如何在Tensorflow上使用神经网络（SSD）构建实时手检测器

[原文地址](https://towardsdatascience.com/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce)：  
https://towardsdatascience.com/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce



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.

这篇文章记录了使用Tensorflow（Object Detection API）训练手检测器的步骤和脚本。 我主要感兴趣的是在桌子上检测手。且是实时的。希望这篇文章能够展示如何将神经网络应用于追踪手的问题（以自我为中心和其他观点），并取得良好的结果。

All of the code ([including the frozen model](https://github.com/victordibia/handtracking/tree/master/hand_inference_graph)) are now available on [Github](https://github.com/victordibia/handtracking" \t "_blank).

所有的代码（[包括冻结的模型](https://github.com/victordibia/handtracking/tree/master/hand_inference_graph)）现在都可以在[Github](https://github.com/victordibia/handtracking/tree/master/hand_inference_graph)上找到。

And here is the detector in action.

这里是探测器的行动。

【图片：手检测器的动态展示】

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](http://www.robots.ox.ac.uk/~vgg/data/hands/) (the results were not good). I then tried the [Egohands Dataset](http://vision.soic.indiana.edu/projects/egohands/" \t "_blank)which was a much better fit to my requirements (egocentric view, high quality images, hand annotations).

与任何基于DNN / CNN的任务一样，流程中最昂贵的（也是最危险的）部分与查找或创建正确的（带注释的）数据集有关。 我首先用牛津数据集进行了实验（结果不好）。 然后我尝试了Egohands Dataset，这更符合我的要求（以自我为中心的观点，高质量的图像，手注释）。

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).

一些fps数字：

* 使用320 \* 240图像的**21 FPS**，运行时不会显示结果
* 使用320 \* 240图像的**16 FPS**，运行时显示结果
* 使用640 \* 480图像的**11 FPS**，运行时显示结果（如上图所示）

以上数字是基于使用MacBook Pro CPU（i7,2.5GHz，16GB）的测试。

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

动机 - 为什么使用神经网络跟踪/检测手？

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](https://www.cse.unr.edu/~bebis/handposerev.pdf) on hand pose estimation from the HCI perspective).

在计算机视觉领域有几种现有的跟踪手的方法。顺便提一下，这些方法中的许多方法是基于规则的（例如，基于纹理和边界特征来提取背景，使用颜色直方图和HOG分类器来区分手和背景等）使得它们不是非常健壮的。 例如，如果背景不寻常，或者照明条件的急剧变化导致肤色急剧变化或被追踪的物体被遮挡，则这些算法可能会变得混乱。 （请参阅[这里](https://www.cse.unr.edu/~bebis/handposerev.pdf)查看来自HCI角度的手部姿态估计的评论文章）。

【图片】

Detection on live video from a webcam. There were some misses when motion was fast and hands were from an unlikely egocentric viewpoint.

来自网络摄像头的实时视频检测。 当运动很快，手是从一个不太可能的以自我为中心的观点来看时，有一些失误。

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.

有了足够大的数据集，神经网络提供了训练模型性能良好的机会，并解决了现有对象跟踪/检测算法的挑战-多光照，或者光照不足，多视角甚至遮挡带来的问题。用于实时跟踪/检测的主要缺点是它们可能是复杂的，与只有跟踪算法相比相对较慢，并且组装好数据集可能是相当昂贵的。 但随着更快神经网络的发展，情况正在改变。

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](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models-coco-models) ) 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](https://pythonprogramming.net/training-custom-objects-tensorflow-object-detection-api-tutorial/) and [here](https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9)). 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](http://vision.soic.indiana.edu/projects/egohands/" \t "_blank)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.

— **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`.

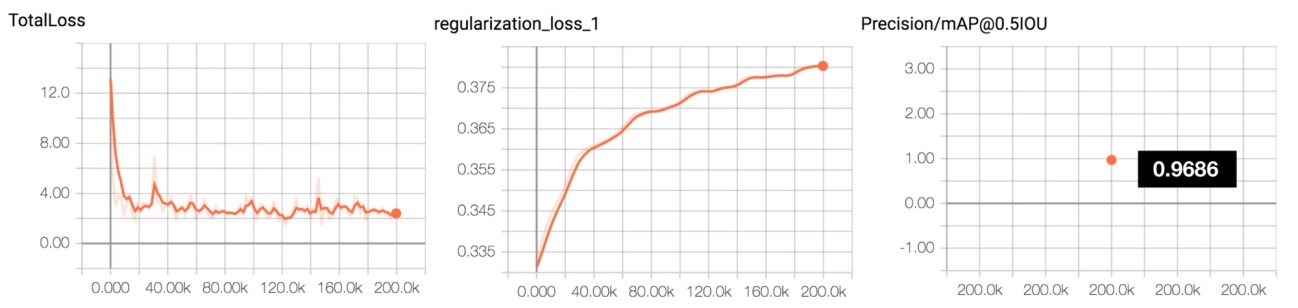
Next: convert your dataset + csv files to tfrecords. Please use the [guide](https://pythonprogramming.net/creating-tfrecord-files-tensorflow-object-detection-api-tutorial/" \t "_blank)provided by Harrison from pythonprogramming on how to generate tfrecords given your label csv files and your images. 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](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_cloud.md).

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](https://www.tensorflow.org/tutorials/image_retraining) 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](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models-coco-models)) 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](https://arxiv.org/pdf/1512.02325.pdf)). 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, the mAP value was **0.9686@0.5IOU**. **mAP** values range from 0–1, the higher the better.



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**

If you have not done this yet, please follow the guide on installing [Tensorflow and the Tensorflow object detection api](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md" \t "_blank). 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](https://github.com/victordibia/handtracking/tree/master/hand_inference_graph)` 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](https://github.com/victordibia/handtracking) 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.

**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](https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/) 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](https://www.learnopencv.com/why-does-opencv-use-bgr-color-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](https://pythonprogramming.net/training-custom-objects-tensorflow-object-detection-api-tutorial/" \t "_blank) and [Dat Tran](https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9" \t "_blank) 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](http://twitter.com/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 T[witter](https://twitter.com/vykthur), [Github](https://github.com/victordibia/" \t "_blank) or [Linkedin.](https://www.linkedin.com/in/dibiavictor" \t "_blank)

**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)

* [Machine Learning](https://towardsdatascience.com/tagged/machine-learning?source=post)
* [TensorFlow](https://towardsdatascience.com/tagged/tensorflow?source=post)
* [Object Detection](https://towardsdatascience.com/tagged/object-detection?source=post)
* [Data Science](https://towardsdatascience.com/tagged/data-science?source=post)
* [Towards Data Science](https://towardsdatascience.com/tagged/towards-data-science?source=post)