On Synthesizing Training Data for Hand Gesture Recognition with a View to Production Deployment

Michael Hornacek
TU Vienna (IMW/CPS)

November 2020

1 Motivation

The approach to training an `AlexeyAB/darknet`[1] object detector carried out by the team thus far in the aim of recognizing hand gestures for control of a projection augmentation system can be considered an encouraging **proof of concept**. The approach taken was the conventional one: a multitude (in our case, thousands) of real-world images were acquired—making use of the production camera-projector setup in the Pilotfabrik—and each was annotated manually. However, besides the **substantial effort** this approach called for, the resulting trained model can be expected to face challenges concerning **transferability** to scenarios where the background undergoes substantial change relative to what was seen in the training imagery, or where the distance (or indeed orientation) of the camera relative to the scene varies substantially from the training setup.

The proposed study aims to fulfill two wishes: (i) to reduce the effort in producing training data, and (ii) to render feasible the process of training (or retraining) the gesture recognition system by a non-expert. The broader objective is to enable the practical use of such a gesture recognition system in a **production** setting, where change is a constant and (camera-projector) setups can vary.

2 Proposed Approach

The approach we propose is to **synthesize training data** using **segmented hand gesture templates** acquired from different viewpoints at a known fixed distance relative to the camera, overlain onto one or more **real-world target background** images. Associated annotation data is produced automatically as part of the approach.

Proposed guiding **general principle** (Ockham's razor[2]): wherever we are presented with more than one possibility of approaching something, let us stick with the simplest approach

[1] https://github.com/AlexeyAB/darknet/
[2] https://en.wikipedia.org/wiki/Occam%27s_razor

that presents itself; let us see how far we can get with the simplest possible approach before we add complexity to it (!).

2.1 Sensor Setup

All image acquisition for the experiment is to be carried out using the **same camera**, mounted in a single **fixed location** and with no change to intrinsic parameters (i.e., **fixed zoom level** and **no autofocus**, thereby implying **fixed focal length**). This is in line with our general principle, and will make it possible (as we shall see) to reason about scaling in terms of depth[3] from the camera.

2.2 Producing Hand Templates

For each target hand gesture, acquire imagery of that gesture at a **fixed distance at close range** (e.g., 1 meter) from the camera at **regular angular increments** of 45 degrees along the axis of the wrist, restricted to the 180 degrees spanned by the back (i.e., dorsal) side of the hand. Proposal (in line with our general principle): for each gesture, we use only a **single person's hand**; exploring the added value of using the hands of more than one individual could be the subject of a separate experiment.

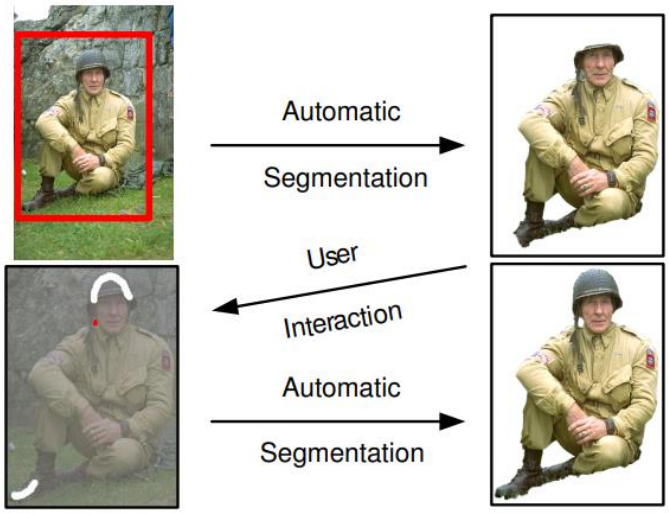

Figure 1: Foreground-background segmentation using the GrabCut algorithm, available in Microsoft PowerPoint. Note that in case GrabCut is not successful in doing a good job at carrying out the segmentation automatically, user interaction in the form of 'scribbles' can be used to help the algorithm distinguish foreground from background. (Figure lifted from the paper.)

[3] The *depth Z* of an object with respect to a camera is the distance along the ray passing from the camera's *center of projection C* through the *principal point* **p**—effectively, the center of the image plane—to the *object* in question. Cf. e.g. http://slideplayer.com/slide/5162869/16/images/2/Pinhole+Camera+Terminology.jpg

Upon acquiring each such hand gesture image, **GrabCut**[4] (cf. Figure 1)—available for immediate use in Microsoft PowerPoint via the 'Remove Background' tool—can be used to **segment** foreground (i.e., the hand) from background. Note that GrabCut is more successful in accurately carrying out segmentation for objects acquired at close range than for objects that appear small in an image raster, thereby motivating close range acquisition. Besides producing a segmentation, a further benefit of GrabCut is that the alpha channel of the segmented output is provided with object **border matting**—a form of smoothing at object boundaries—thereby rendering overlay on a novel background possible via **alpha blending**.[5] Overlay carried out in this manner appears more realistic than when using a binary (i.e., foreground vs. background) segmentation mask.

2.3 Synthesizing Training Data

Given a real-world target background image, training image data is synthesized for a given hand gesture template by randomly applying **rotation** (in plane) and **scaling** (within the bounds that would yield realistic target hand sizes[6]) and placing the resulting transformed template at a random **location**. Training annotation data is produced automatically from the bounds of the randomly transformed template and its randomly selected location.

The **parameters** of the production of synthetic training data (i.e., paths to input data, number of synthetic examples to be produced per gesture template, scaling range) are to be read in from a **configuration file** in the aim of facilitating **traceability and reproducibility**.

2.4 Comparative Evaluation

The evaluation aims to compare the performance of training using the erstwhile **manual approach** to acquiring training data vs. that of the **synthetic approach** relying on synthesized training data (produced using the same background). In order to eliminate questions of the influence of **class imbalance**, it is to be ensured that training data in all experiments be read in during training in an **interleaved** fashion, with the objective that each training **batch**[7] contain an even number of examples from each target class. Here too let us rely on **configuration files** containing experiment parameters to facilitate **traceability and reproducibility**.

Baseline experiment: cull training data from the manual training data set in order to ensure that the same number *n* of training examples for each gesture are present, and produce *n*

[4] https://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf
[5] https://www.learnopencv.com/alpha-blending-using-opencv-cpp-python/
[6] It can be shown that the scaling factor to apply to a template acquired at depth Z_{in} so as to appear as if it were acquired at depth Z_{out} is exactly Z_{in} / Z_{out}, assuming we use the same camera (with fixed focal length, i.e., with no change in zoom level and with autofocus deactivated) for training as for inference. Note that this fact enables parameterizing the scaling directly in terms of target depth.
[7] It appears this is possible by adding `track = 1` to the `.cfg` file passed to the `darknet` executable; cf. Section 3.3 for more details.

examples of each gesture using the synthetic approach. For all else fixed (i.e., hyperparameters), compare performance.

Followup experiments: incrementally halve the number *n* for both manual and synthetic. What can be say about our tradeoffs (i.e., effort vs performance, …)? Is there a 'sweet spot' with respect to tradeoffs?

2.5 Potential Future Work?

- **Background subtraction**-based approach (keep track of mean value per pixel with respect to last *m* frames; mask out all pixels whose value is close to mean value, and do gesture recognition only on remaining parts of image).
- Question of how much additional (synthetic) training data is sufficient to train **additional backgrounds**.
- Closer look at questions of **data augmentation** during training.
- Impact of Gloves? Of skin color? Of hands of several individuals?

3 Notes on `AlexeyAB/darknet` Object Detector

In previous work on training an `AlexeyAB/darknet` object detector carried out by the team, training was invoked using the following command (the paths here are abridged):

```
./darknet detector train path/to/obj.data path/to/yolo-obj.cfg
path/to/darknet53.conv.74 -dont_show -map
```

Accordingly, what we explore in the subsections below is what happens when training using `AlexeyAB/darknet`'s `detector` mode. In particular, we examine what parts of the code are responsible for reading in the `.cfg` file, how batches are read in for training (with respect to file paths provided in the `.data` file), and what forms of augmentation are applied. First, however, we outline how to set up Microsoft Visual Studio 2019 in the aim of exploring the `AlexeyAB/darknet` source code.

3.1 `AlexeyAB/darknet` in Visual Studio 2019

Visual Studio can be used not only to build (i.e., compile) the code, but also to explore the code with greater convenience than is possible directly in GitHub or using a conventional text editor. Of particular interest, Visual Studio includes a facility to jump to e.g. a function or variable definition from anywhere the function or variable is used in the code.

Obtaining Visual Studio. Order[8] Microsoft Visual Studio 2019 via TU Wien Campus Software, access[9] the installation files upon receipt of approval e-mail ("Die untenstehenden Bestellungen wurden freigegeben"…), and carry out the installation of Visual Studio.

Obtaining dependencies + source code. Install[10] the CMake, CUDA, OpenCV, cuDNN dependencies of `AlexeyAB/darknet` for Windows and download[11] a copy of the source code itself from the GitHub repository of `AlexeyAB/darknet`.

Exploring + building the code. Double click on `build/darknet/darknet.sln` (should launch Visual Studio). You can now, e.g., right click on functions and select 'Go To Definition' to be taken to location in the code where the function is implemented. To build the code, select Build Solution from the Build menu; you should find a freshly built `darknet.exe` executable in `build/darknet/x64`.

3.2 Loading the `.cfg` File

The entry point of darknet—i.e., the `main()` function—is located at `src/darknet.c`[12] at line 432. Since the first argument to the above command is `detector`, the function `main()` calls the function `run_detector()` at line 493 (implemented in `src/detector.c` at 1933). The second argument to the above command is `train`; accordingly, `run_detector()` in turn invokes `train_detector()` at line 2006 (implemented in `src/detector.c` at line 26), which reads the `.cfg` file via `parse_network_cfg()` at line 78, subsequently accessible in `train_detector()` via the struct `net` assigned on line 90. The function `train_detector()` then sets a struct `args` starting at line 131 according to what is present in `net` and, at line 146, hardcodes `args.type` to `DETECTION_DATA`.

3.3 Loading Training Data

Training data is loaded in `train_detector()` (cf. Section 3.2) via invocations of the function `load_data()`, which is implemented in `src/data.c` at line 1736. The function `load_data()` calls `load_threads()` at line 1741; `load_threads()` calls `run_thread_loop()` at line 1677, which calls `load_thread()` at line 1650. Since `args.type` is set to `DETECTION_DATA`, the function `load_thread()` finally calls `load_data_detection()` at line 1596. The function `load_data_detection()` is

[8] https://www.it.tuwien.ac.at/en/services/software/software-provision/campus-software/software-list/ (order by clicking 'online bestellen' at https://oase.it.tuwien.ac.at/438397.asHTML)
[9] https://www.it.tuwien.ac.at/en/services/software/software-provision/campus-software/connection-to-the-swd/
[10] https://github.com/AlexeyAB/darknet/#requirements
[11] https://github.com/AlexeyAB/darknet/archive/master.zip
[12] https://github.com/AlexeyAB/darknet/blob/master/src/darknet.c

implemented at line 1052, and is the **function responsible for (i) reading batches and for (ii) data augmentation**, when training using `AlexeyAB/darknet`'s `detector` mode.

Batches. Whether training data is read sequentially via `get_sequential_paths()` at line 1101 of `src/data.c` or in a random ordering via `get_random_paths_custom()` at line 1102 depends on the `track` variable in the `.cfg` file, passed on to `load_data_collection()` as part of an `args` struct. If `track` is not provided in the `.cfg` file, `args.track` defaults to a value of 0. If the variable `args.track` is assigned a value of 0, the function `get_random_paths_custom()` is called; otherwise, it is the function `get_sequential_paths()` that is invoked. **Note that the `.cfg` file used in previous work by the team did not contain a (nonzero) `track` entry, i.e., batches were read in randomly. Note also that this can be confirmed by invoking the `darknet` executable with the `-show_imgs` flag.**[13]

Data augmentation. The forms of data augmentation applied in training using the `detector` mode of `AlexeyAB/darknet` is driven by the following parameters to `load_data_detection()`:

- `use_flip`
- `use_gaussian_noise`
- `use_blur`
- `use_mixup`
- `jitter`
- `resize`
- `hue`
- `saturation`
- `exposure`

Note that this leaves out augmentation that concerns **rotations**, **affine** transformations (i.e., rotation + translation + skew), or more general **projective** transformations (also known as '**homographies**'), which could accordingly still be something to explore in the future. **Note also that the augmented training data used during training can be visualized by invoking the `darknet` executable with the `-show_imgs` flag.**

3.4 Details Concerning Prior Experiments

From Majesa via e-mail on 31.11.2020:

[13] https://github.com/AlexeyAB/Yolo_mark/issues/123

The changes that are impacting the training are mainly the downloaded convolutional darknet53.conv.74, the yolo-obj.cfg, the obj.data file which specifies the obj.names, the train set and if used the test set.

- The YOLOv3 convolutional network was only downloaded and then referenced during the beginning of the training.
- The cfg was changed according to AlexeyAB's tutorial[14] and is based on the yolov3.cfg which was already in the directory:
  - batch = 64 (line 6)
  - subdivision = 16 (line 7)
  - max_batches = 60000 (line 20 – classes * maximal number of photos per class, in our case 20000)
  - steps = 48000, 54000 (line 22 – 80% and 90% of max_batches)
  - ~~wide = 416 (line 8 – network wide, we tried to change that, but the programm failed)~~
  - ~~height = 416 (line 9 – network height, similar to above)~~
  - classes = 3 (line 610, line 696, 783 – number of classes)
  - filters = 24 (line 603, line 689, line 776 – number of filters, 3 * (classes + 5))[15]
- The obj.data[16] file only specifies the other important files and is included in the slurm[17] script:
  - classes = 3 (number of classes)
  - train = /.../train.txt (the training set for fitting)
  - valid = /.../train.txt (validation set – same as train because we use map calculations)
  - names = /…/obj.names (specifications of the class names)
  - backup = /…/backup (directory where weights are stored)

  80% of the data set were used for the train and test set.

Nummer	Datensatz	File	Qu
1	BochumGestures1998	BochumGestures1998.tar.gz	https://www.idiap.ch/resource/ges
2	Dataset ASL Hand Gestures	Dataset ASL Hand Gestures.zip	https://ieee-dataport.org/open-acc dataset
3	hands-dataset	hands-dataset.zip	https://www.kaggle.com/bigkizd/h
4	hgr1_images	hgr1_images.zip	http://sun.aei.polsl.pl/~mkawulok/
5	hgr2a_images_large	hgr2a_images_large.zip	http://sun.aei.polsl.pl/~mkawulok/
6	hgr2b_images_large	hgr2b_images_large.zip	http://sun.aei.polsl.pl/~mkawulok/
7	kinect_leap_dataset	kinect_leap_dataset.zip	https://lttm.dei.unipd.it/downloads
8	leapgestrecog	leapgestrecog.zip	https://www.kaggle.com/gti-upm/l
10	senz3d_dataset	kinect_leap_dataset.zip	https://lttm.dei.unipd.it/downloads
11	set_1	set_1.rar	https://www.gti.ssr.upm.es/data/H

[14] https://github.com/AlexeyAB/darknet#how-to-train-with-multi-gpu and https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

[15] Cf. https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

[16] Produced using BBox Label Tool, cf. https://texas-aerial-robotics.github.io/md_yoloTraining.html

[17] Used in context of Vienna Scientific Cluster

12	set_2	set_2.rar	https://www.gti.ssr.upm.es/data/H
13	Set1	Set1.zip	https://labicvl.github.io/ges_db.htr
14	Set2	Set2.zip	https://labicvl.github.io/ges_db.htr
15	Set3	Set3.zip	https://labicvl.github.io/ges_db.htr
16	Set4	Set4.zip	https://labicvl.github.io/ges_db.htr
17	Set5	Set5.zip	https://labicvl.github.io/ges_db.htr
9	Marcel-Train	shp_marcel_train.tar.gz	https://www.idiap.ch/resource/ges

QUELLEN zum verwendeten YOLO Setting:

https://pjreddie.com/darknet/
https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects
https://github.com/puzzledqs/BBox-Label-Tool
https://github.com/ayooshkathuria/pytorch-yolo-v3