Refining your DeepLabCut model – How to learn from errors

Overfitting or generalizing, that is the question. What do you want to use your DLC model for? If you want to label many videos from a similar setup, an overfitted model will do very well (until you change the camera perspective, lighting conditions or maybe even the subject). On the other hand, if you wanted to train a model that detects human facial landmarks in different subjects, from different genders, ethnicities, facial hair styles and glasses and specially from different angles, then you need a model that generalizes well to all these conditions. And for that, you will need different training examples, quite a few actually.

If you still need to install DeepLabCut, check out this guide [Installing DeepLabCut](<https://guillermohidalgogadea.com/openlabnotebook/installing-deeplabcut/>) and this short example on [Training your first DeepLabCut Model](<https://guillermohidalgogadea.com/openlabnotebook/training-your-first-dlc-model-/>). As always, check the ever-expanding DeepLabCut documentation on [Github](<https://github.com/DeepLabCut/DeepLabCut>) for further details.

In this example we will start with a trained DLC model to detect facial landmarks, overfitted to a single subject (i.e., me). We will then try to analyze new videos from different subjects of different genders and ethnicities. We will extract outliers from these misdetections, refine them manually and re-train the model. Last, just for fun, I will use anipose to triangulate multiple camera perspectives into a final 3D facial expression for each subject.

The original model was trained on 120 labeled frames of one single subject from 6 different camera angles, using resnet50 with 1.030.000 iterations

Figure 1: Video of parallel tracked face

<https://youtu.be/RoaPwEWcHF0>

It was to be expected, that this model wouldn’t generalize well to new subjects, some wearing glasses and different hair styles coving parts of the face. But we can make use of this and check when exactly the model struggled the most. Then, we can refine a few of those outlier frames and re-train the model with the information that was apparently missing before.

A step by step guide for this example is provided as jupyter notebook [here](), check it out and tag along. \*\*Note\*\*: In windows machines you may experience some permission issues trying to extract frames and adding new videos to your project. Just to be sure, try starting anaconda terminal with elevated privileges before starting the notebook, to avoid some small issues later.

First, start deeplabcut in your favorite mode, either as graphic user interface or command line. I will use a jupyter notebook myself, but the same steps should work over the gui as well. Initialize the previous project by setting the path to the corresponding config.yaml file and starting a list with new videos to be analyzed, in this case 30 new videos from 5 different subjects:

path\_config\_file =

new\_videos =

We will then use the following commands to analyze the new videos with our previously overfitted model:

deeplabcut.analyze\_videos(path\_config\_file, new\_videos, shuffle=1, save\_as\_csv=True, videotype='mp4' )

deeplabcut.create\_labeled\_video(path\_config\_file, new\_videos, videotype = 'mp4', save\_frames=False)

And of course, we won’t be too sad that DeepLabCut struggles to find my nose and my eyebrows in others’ faces

Fig 2. outlier tracking from DeepLabCut (Mengmeng, Sevim and Julian)

We will extract this poorly tracked outlier frames, refine them manually and feed them back to the training dataset for re-training. To keep it (relatively) simple, I will only be extracting 5 frames from each video, because a total of 150 manually labeled frames from 30 new videos seems enough for my purpose. Open your config.yaml file and set the number of frames to pick to whatever you want, in this case (numframes2pick: 5). Manually labeling frames is your work, while training the model and analyzing videos is the machine’s work. I would recommend refining the model labeling only few outlier frames and see what happens. If the model still struggles, you can refine it over and over again, extracting few new outliers, instead of extracting many outliers at once.

deeplabcut.extract\_outlier\_frames(automatic=True) ## Test again

The function above will look for outliers in the tracked coordinates and ask for user feedback to start extracting these outlier frames from the video file. If you don’t want to monitor every single outlier extraction, make sure to pass ‘automatic=True’ to the function above. The extracted frames are stored in the subdirectory ‘labeled-data’ in your DLC project folder, and the path to the respective video should be included in the config.yaml file. At this point you could get an error message like [this](<https://github.com/DeepLabCut/DeepLabCut/issues/232>) telling you that saving the video path failed. In this case, you need to add new video paths manually for DLC to include these in the new training set. You can either add them by hand, writing in the config.yaml file in the same format as the first video paths, or you can run the following command to add the list of videos to your config file:

deeplabcut.add\_new\_videos(path\_config\_file, new\_videos, copy\_videos=False)

If the permission error persists, try starting a new anaconda terminal as administrator (right click > run as administrator) and then starting jupyter notebook with elevated privileges.

After extracting the outlier frames your actual work can begin. The function below starts a graphical interface to refine labels manually:

deeplabcut.refine\_labels()

First, you need to load the directory with frames you want to refine, and then load the corresponding .h5 file with the outlier predictions from the previous model.

You are asked to define a likelihood threshold, but don’t worry too much about it, this is only to show you which labels have an especially low prediction. Labels with a likelihood higher than your threshold will appear as opaque colored circles, while labels below the threshold will appear as transparent colored circles. As I am going to refine all labels, and I recommend you do the same, I like setting the threshold to 1 to display all markers as transparent circles. This makes it somewhat easier to place the center of the marker when you use the zoom.

The refining process is overall very similar to labeling, other than the labels are already placed somewhere in the frame. You drag them holding the left mouse click and drop them in place. Delete labels that are not supposed to be in the frame by clicking the mouse wheel. Make sure to delete labels that are not visible and suppress that “knowing” where the eyebrow is behind that strand of hair, if you don’t see it, don’t label it. Minimally different from the original labeling process, instead of saving and quitting to go on labeling other files, you will be prompted to refine other files directly after clicking save. The Quit button will close the GUI without redirecting you to new files … and that may cause your notebook to crash. In that case try restarting the kernel.

After refining all the outlier frames extracted, merge the dataset to combine these new labels with the labeled data from your previous model, and create a new training data set:

deeplabcut.merge\_datasets(path\_config\_file)

deeplabcut.create\_training\_dataset(path\_config\_file, net\_type='resnet\_50', augmenter\_type='imgaug')

Note that this step will only work if the link to the videos these new frames come from are included in the config.yaml file (see permission issues [here](https://github.com/DeepLabCut/DeepLabCut/issues/1181) and [here](<https://stackoverflow.com/a/65504258>)). Otherwise, you will just duplicate the previous training dataset, and spend several days training a new model that turns out to be exactly as the old one, as it happened to a friend of mine.

To start training from the last iteration from the previous model, we would go in the dlc-models directory, go to the iteration of the previous model, find the train subdirectory, and look for the latest snapshot, e.g., 'snapshot-1030000'. Then, go to the newest iteration in the dlc-models directory, go to the train directory and open the pose\_cfg.yaml file. Edit the parameter init\_weights to add the last snapshot without any filetype ending. It should look something like this, applied to your directory or course:

# instead of general resnet\_v1\_50

init\_weights: C:\Users\hidalggc\Anaconda3\envs\DLC-GPU\lib\site-packages\deeplabcut\pose\_estimation\_tensorflow\models\pretrained\resnet\_v1\_50.ckpt

# use pre-trained model

init\_weights: 'D:\cool-stuff \DLC-Projectname\dlc-models\iteration-0\DLCApr14-trainset95shuffle1\train\snapshot-1030000'

Finally, you can re-train the model with your new extended labels as follows:

deeplabcut.train\_network(path\_config\_file, shuffle=1, displayiters=100, saveiters=1000)

Note that I use a GPU on my local machine. If you are working with a CPU and outsource training to google colab, now would be the time to do so. Check the last Blogpost on [Training your first DeepLabCut Model](<https://guillermohidalgogadea.com/openlabnotebook/training-your-first-dlc-model-/>) for an example on how to migrate you project for cloud computing.

The result is a DLC model trained over two iterations on 270 labeled frames from 6 different camera angles and 6 different subjects. Three females and three males, representing nationalities from Germany, Spain, Turkey, Iran, and China.

Run the functions below once again to analyze the novel data with the newly refined model to see how it now generalizes to different subjects:

deeplabcut.analyze\_videos(path\_config\_file, test\_videos, shuffle=1, save\_as\_csv=True, videotype='mp4' )

deeplabcut.create\_labeled\_video(path\_config\_file, test\_videos, videotype = 'mp4', save\_frames=False)

Figure 2: Collage of tracked facial expressions from different subjects

A little Extra

Recording the same scene from multiple camera angles has several benefits. First, different angles provide multiple training examples from the same recording, thus increasing your training set in number and in perspectives. Second, multiple angles avoid occlusion, e.g., when subjects roll their eyes behind your back, at least one camera will catch that. And last but not least, the original reason to record from multiple camera angles was to allow 3D triangulation from the different 2D images. This next process will be covered in detail in a separate post, but for now you can lay back, start reading about [Anipose](https://anipose.readthedocs.io/en/latest/) and enjoy the 3D tracked facial expressions below.

Figuer 3: 3D video collage

*Let me know on*[*Twitter*](https://twitter.com/G_HidalgoGadea)*if you found this guide useful or would like to have a more detailed discussion on any of the methods used above.*