Supervisely Manual: training your own object detector (without any programming)

Toon Goedemé, EAVISE – PSI, KU Leuven

In this tutorial, we will lead you through the process of training your own custom image object detection neural network. This tutorial is especially suited for companies that do not have much programming expertise in-house. We therefore examined a programming-free method to train an object detector for a specifically trained object. In the Supervisely system, data annotations and model training can be done, all from a web-based GUI, without writing a single line of code.

1. Introduction

To get an overview of the flow and possibilities of Supervisely, we suggest you to first watch the following YouTube video: https://www.youtube.com/watch?v=w94NfyMeXRk . Here, Ned McClain explains how to train a detector for roads in images. He cheats maybe a bit, starting from pre-annotated datasets in Supervisely, but the process is very interesting to follow.

2. Create your Supervisely environment

First of all, you need to make a login for Supervisely. This is very simple, just go to www.supervise.ly and click on signup.

When your account is ready, you can log in. We suggest you to let your browser remember the login credentials, such that logging in is simple. Supervisely frequently asks you again to log in in separate browser tabs.

You also will need GPU capable computing hardware to run your training procedure on. This is called “Cluster” in Supervisely. If you click on the “Cluster” menu, you can see only the built-in Supervisely Agent, which can be used for simple calculation tasks, but not for training a neural network. You can add computation hardware with the “+ Add” button. There are two options:

- If you have a Linux PC with a decent GPU (e.g. NVIDIA RTX), it can be connected to Supervisely to use as a computation cluster. Your computer must meet the following requirements: Linux OS Kernel 3.10, Docker Version 18.0, GPU CUDA 9.0, NVIDIA-Docker Version 2.0. The instructions are very

simple and appear on your screen after pressing the “+ Add button”. You just have to run a specific command on your Linux PC, which pulls the Supervisely Docker image and starts it.

- If you do not have such a computer, you can make use of computation power in the cloud. However, this comes at a cost per computation. The cheapest solution is to use a AWS (Amazon) p2.8xlarge GPU spot instance, as explained in the video from Ned https://www.youtube.com/watch?v=w94NfyMeXRk&feature=youtu.be&t=216 or on the following help page: https://docs.supervise.ly/customization/agents/ami?_ga=2.19282362.30100371.1607602253-755545227.1607089823

After one of these steps is successful, you can see both running Agents at the “Cluster” page.

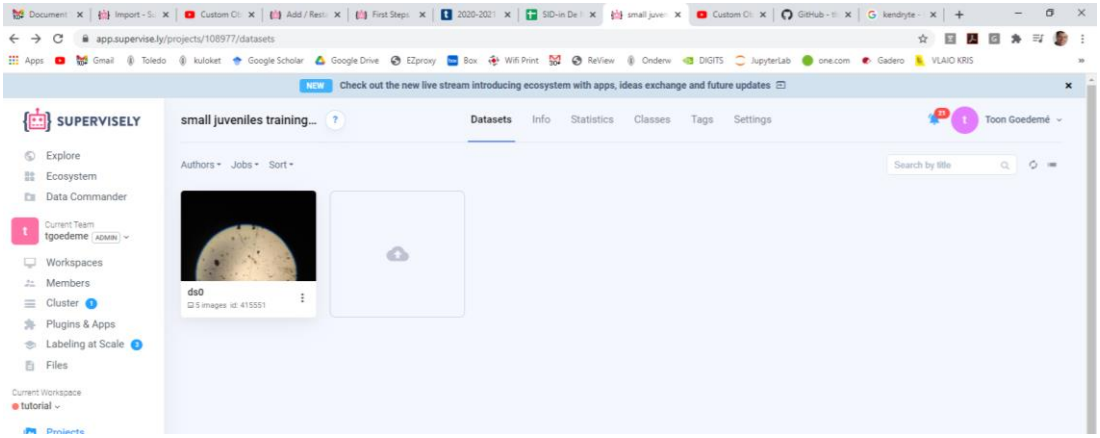
3. Adding data and preparing to annotating them

Now it is time to add images to the Supervisely system and start annotating.

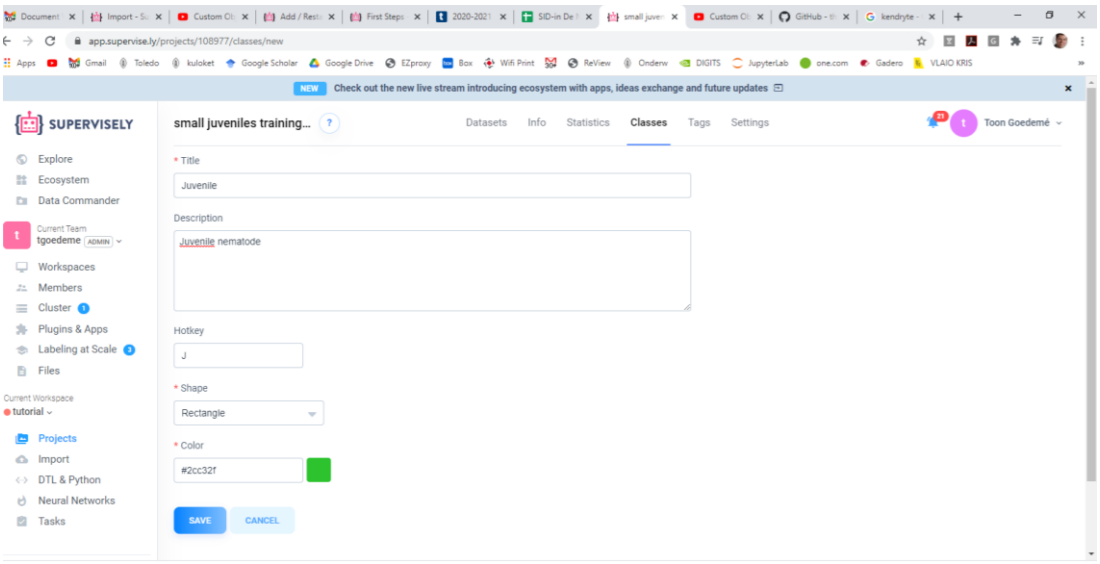
First, you will need to create a new project. Click on the “+ Add” button in the “Workspaces” page. In this tutorial, we create a workspace “Tutorial”.

Once you entered the workspace by clicking on it, you can see a welcome screen with instructions to add data. Click on the button “Import Data”. Just drag your images or image folders on the light blue field, give the dataset a name and click on “Start importing”. Important remark: these are the images that will be used during training. Later on in the process, we will need new images to test our trained model on. So it can be wise to keep some images separate now for testing in that later phase.

Now, when this task is ready, you have a dataset ready. If you go to your Workspace (via the left hand menu), you can see a project now with the imported images. You can enter the project by clicking on the image. A new dataset is now visible, normally called “ds0”:



Now, you have fist to define the object classes that you need to be detected. At the top of the page, you can see the menu-item “Classes”. There you can add object classes by clicking on the “+ Add Class” button. We can add a name and description of our object to be detected now:



Importantly, you have to choose “Rectangle” as the shape for annotation and detection, as our YOLO models work with rectangular bounding boxes. Choose a hotkey and a color and click on “Save”.

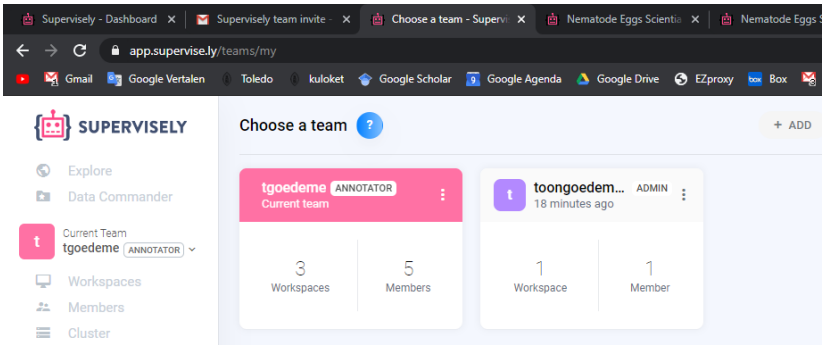
You can start annotating your dataset yourself, or even distribute these annotation tasks to co-workers. If you want to do the annotation job yourself, just go back to the “Datasets” menu-item, and then on the three vertical dots (…) icon of the dataset (normally called ds0) and chose “Start annotation”. If you want to assign a data annotation task to someone else, you can use the command “create labeling job”. (You can follow up on these labeling jobs from the left-hand menu-item “Labeling at Scale”.)

4. Annotation instructions

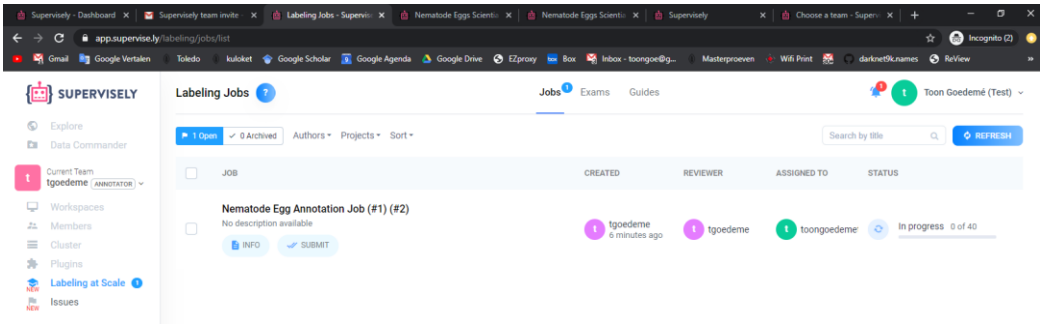
Earlier, we already created annotation instructions for the nematode egg test. Below, these instructions are repeated. These assume you have created an Egg and Shell class in the “Classes” of the project and have assigned labeling jobs to the labelers via the “Labeling at scale” menu. (But you can also follow these instructions from step 4 onwards if you do the labeling yourself.)

1. Log in to Supervisely: surf to supervise.ly

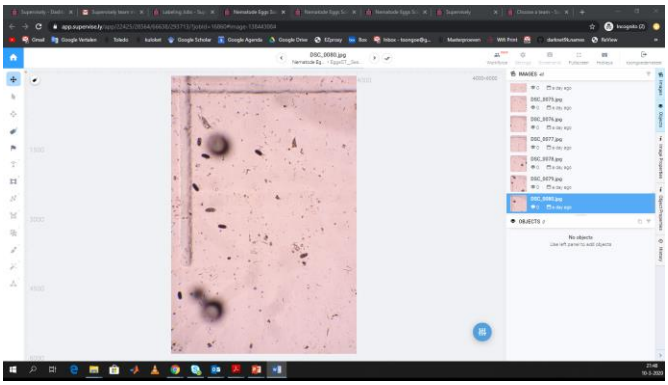
2. Choose the right team, in which you are an Annotator



3. You will see you have an open labeling job. Click on the job name to enter it



4. The first image to be annotated opens up:

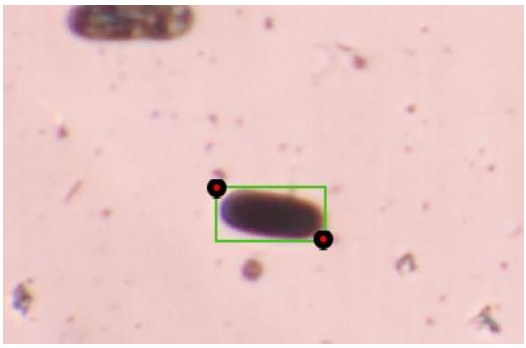


5. In this image you will have to annotate all Nematode Eggs (full ones) and Nematode Egg Shells (empty ones). Handy shortcuts to move around in the image:
   - scroll wheel: zoom in and out
   - drag with right mouse button: shift image

6. How to annotate:

   - click on the “Add rectangle” tool on the left

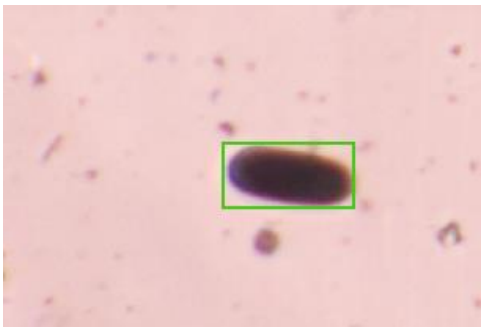
   - draw a bounding box on the egg to be annotated by clicking on the top left and bottom right corner of a box around the egg:



The bounding box should fit snuggly around the egg, not too wide, but just touching the outer contour.

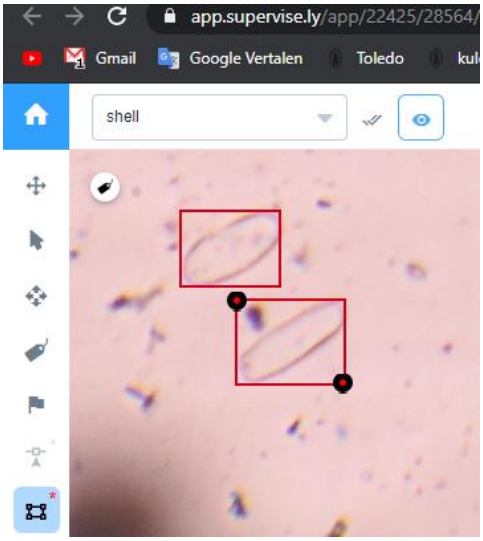
- confirm by pressing the hotkey “E” (for egg)

- the egg is now annotated! (the dots disappear, and only a green bounding box stays visible)



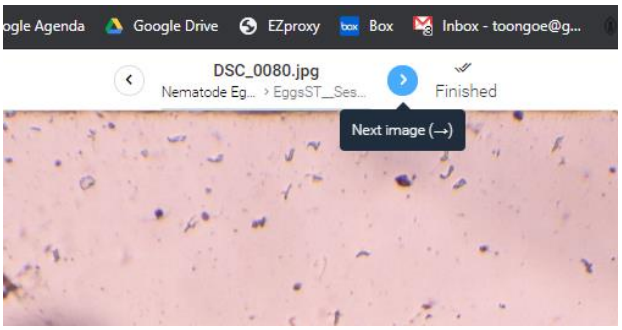
- repeat this for all eggs in the photo

7. Use the same procedure to annotate all empty nematode egg shells in the photo. First switch the object type in the top left dropdown menu to “shell” (or use hotkey “S), then click on two points of the bounding box, and then confirm with hotkey “S”:



You can always switch back to annotating the other object by means of the hotkeys E and S.

8. If your photo is ready (all eggs and shells annotated), switch to the next photo with the right arrow “>” above:



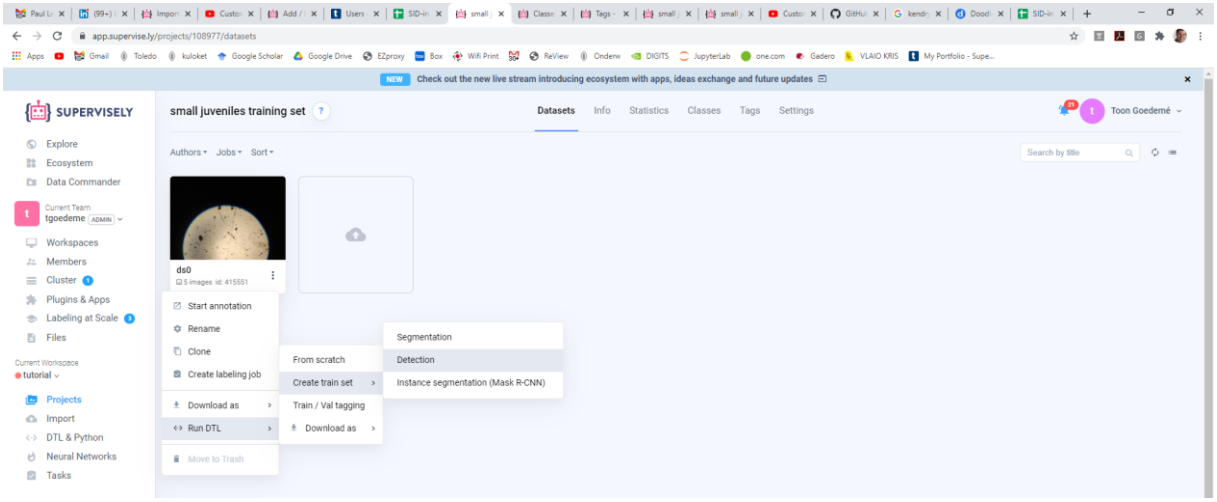
5. Training a first neural network

Now that your annotated data is ready, we can start training a network.

A first step is preparing your annotated data for training. As you might know, we have to split our dataset in two parts, one for the actual training and one for verifying if everything goes right during training, i.e. the validation set. We also know that we need as many training examples as we can, therefore the *data augmentation* tricks are invented: with simple transformations (e.g. mirror, flip, resize, crop, …) we can create additional training examples from the annotated ones and multiply the training set’s size.

These two tasks, training/validation split and data augmentation, can of course be done manually. But better, Supervisely offers an automated way to do this. There is a built-in Data Transformation Language script (DTL) to do these tasks. Here is how to execute it:

Within your workspace, navigate towards the dataset you just annotated. Now, click on the three vertical dots of the ds0 dataset and choose “Run DTL” -> “Create train set” -> “Detection”. We will indeed create a training set from our annotations for object detection.



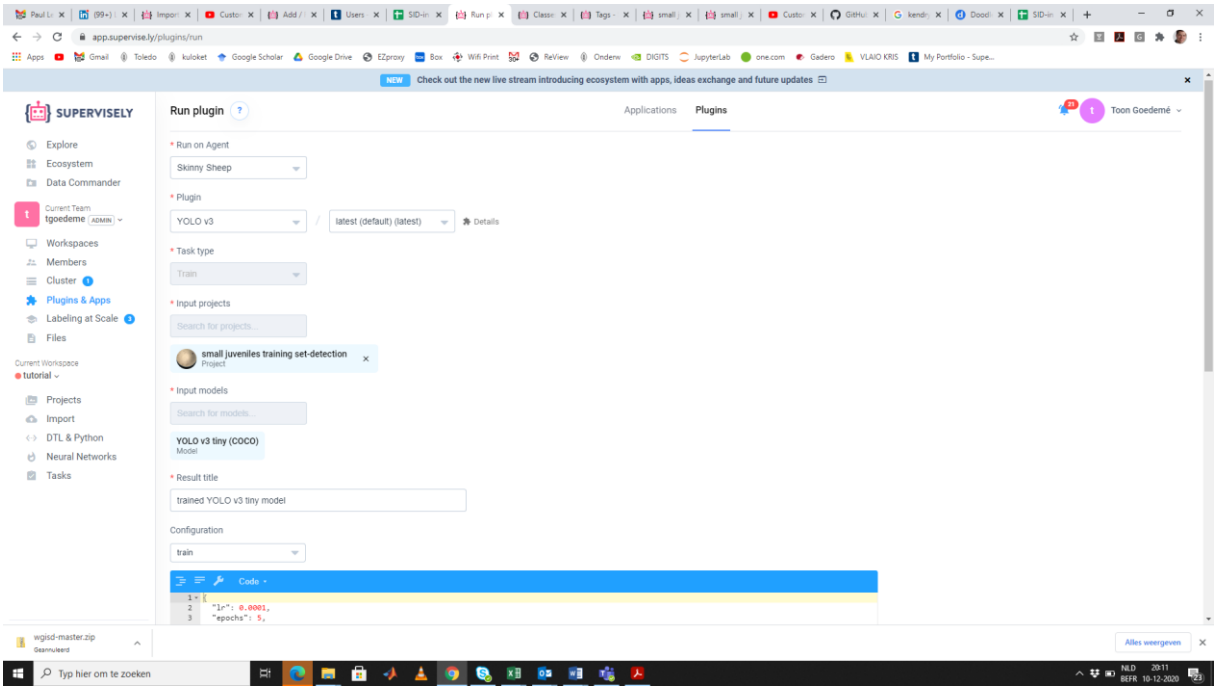
On the next screen, you can see the details of the DTL script. As you can see, also in the graphical representation, the data is first a few times randomly flipped and resized/cropped, after which each image is tagged as training (85% of the data) or validation (15% of the data). You can alter these

settings, but for these experiments the DTL is good as it is. Just click on “START” to run it. After completion, you can see in your Workspace that a second Project is added, with a much more training images as before. This project’s name ends with “-detection”.

Now we are ready to start training our first neural net! In the left hand navigation bar, choose the menu-item “Neural Networks”, near the bottom. The page that appears is now empty, but click on the “+ADD” button to add a neural network to your workspace. You will see that there is a huge list of neural network architectures available and ready to use within Supervisely. We will use the “YOLOv3 tiny” network in this tutorial because it trains so quickly, but e.g. a regular “YOLOv3” is also a good choice.

Search for this “YOLO v3 tiny (COCO)” network, and click on its name. Then you arrive at a page explaining some details about this neural network architecture. On the far right top corner, you can find the button we need: “+ Clone Model”. Click on it and click on “CLONE” such that it is copied towards your workspace.

If you go to your “Neural Networks” page, you should now see the YOLO v3 tiny model. It is pretrained on the COCO dataset, such that it only needs a limited amount of training data to learn a new object. At the right hand side, you can find a “Train” button. Clicking on it will bring you to the training settings page:



The settings on this page are important to understand.

- In the first field, you can select on which Cluster agent you will run the training calculations. Normally, the agent you created by connecting your own Linux PC or your AWS instance is selected there.

- The next fields indicate that you will be training a YOLO v3 model. That’s good.

- As input project, you must select the one the DTL script which you ran a few moments ago (doing the data augmentation and dataset train/validation split). This is normally the one that ends with “-detection”.

- The input model is of course our freshly cloned pre-trained YOLO v3 model.

- You must give a name to the neural network model you are about to train. I chose for “trained YOLO v3 tiny model”, as you can see in the screenshot.

- A predefined training script is at the ready if you select “Train” in the Configuration box.

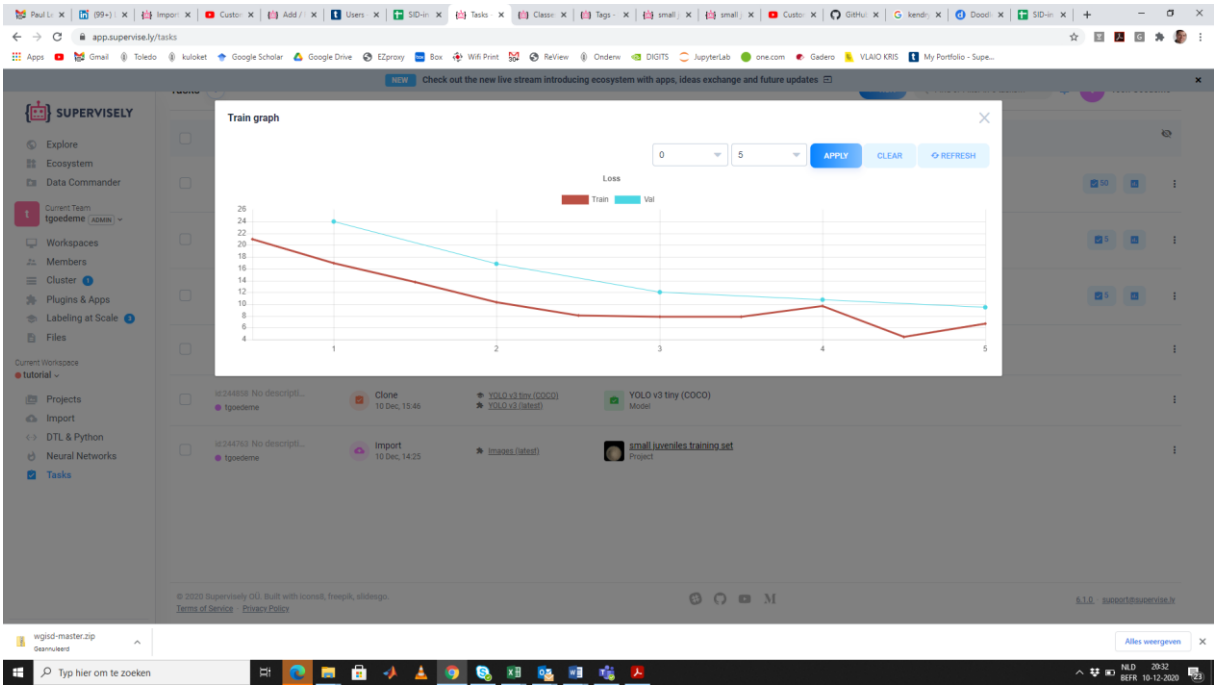
- The script itself is also very interesting. Two very important settings are the following:

* The “epochs” variable expresses how long your training will be. One epoch is a full pass over your entire training dataset. You can start with e.g. leaving this value at 5, but for better training you can surely make this number larger.

* “lr” means “learning rate”, i.e. the speed at which the neural network weights are adapted during the training phase. This value can be smaller when you train longer, resulting in a better tuned model.

Also note that at the bottom of the script, the training is set at “transfer learning”, which means that we start from pretrained weigths and tune them for our object(s).

Click on “RUN” to start the training. You will see a new task appearing, with a progress bar. Depending on the size of your training data set, this goes relatively fast. You can keep an eye on the training process by clicking on the blue “Charts” icon on the right. This gives something like this:



We see that in our case both the training loss and the validation loss goes down, which is a sign the training goes well. If you would see the training loss going down while the validation loss goes up again, it would mean that we are overfitting on the training data. If the loss curves are too jumpy, it means that you chose a too big learning rate.

The other blue icon, “View and manage checkpoints”, gives a view upon the different checkpoints that are saved during training. In the training script we used, every epoch the status of the neural network model is saved as a checkpoint. You can see the info of each checkpoint in the list. Hopefully the validation loss goes down every epoch, like it did in my case. But as there is quite some randomness in the training procedure, it can be that it jumps up and down, too.

6. Testing your trained model

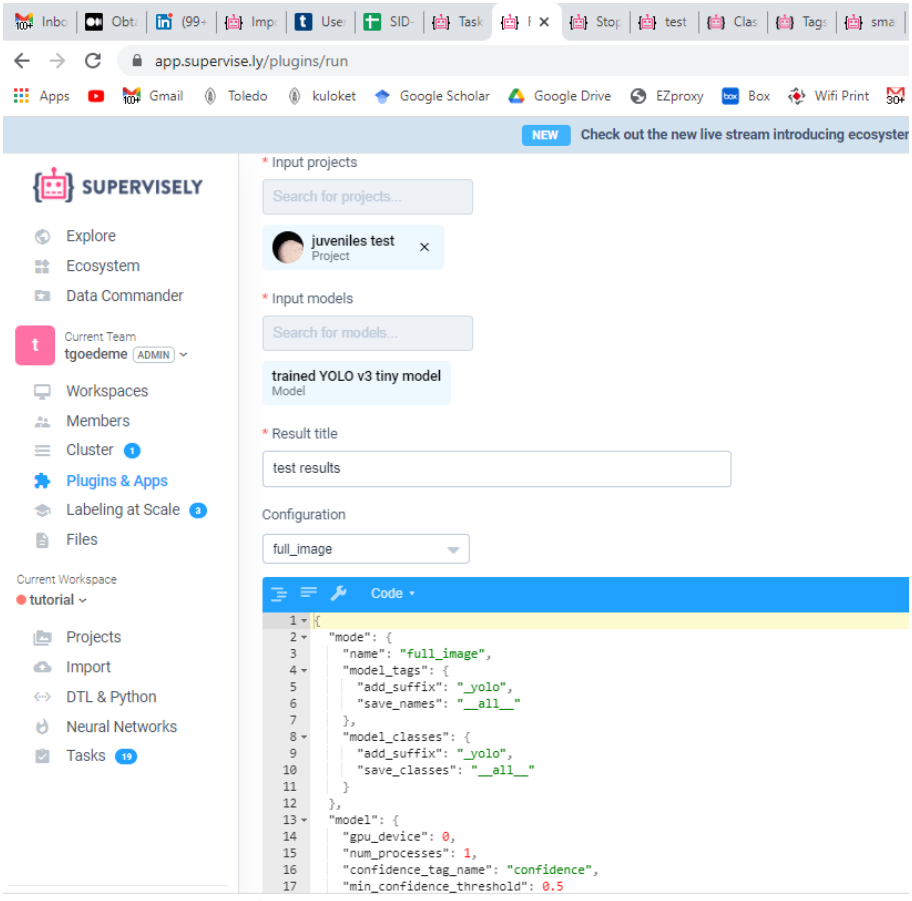
The moment of the truth! Now we will see if the freshly trained model does something useful.

Firstly, of course, we have to collect testing images. For the test to be honest, these must be images that are not used before during the training process. With new images, we can indeed see if the model really learned to recognize the object, and did not merely memorize the training images.

So, gather some new images containing the object to be detected, or find the ones you kept separate on your harddisk during step 3 of this tutorial. Navigate to your Workspace and click on the blue cloud-with-up-arrow tile. This will bring you to a page where you can simply drag and drop your images. Give it a clear name, e.g. something with “testing” in.

Now, go to your “Neural Networks” list via the menu-item on the bottom left. Here, you can see that, next to the off-the-shelf YOLO network you cloned, our newly trained network model is listed. It has a “Test” button, and that is of course the one we need. Click on it.

You will be asked to fill in some details. Most things are already filled in, luckily enough with correct values. You must manually select the “testing” dataset you just created in the “Input Projects” field. The test will execute (called “inference”) our learned neural network on all the images of this “testing” dataset. It will write the results to a new dataset, of which you can chose the name in the “Result title” box.

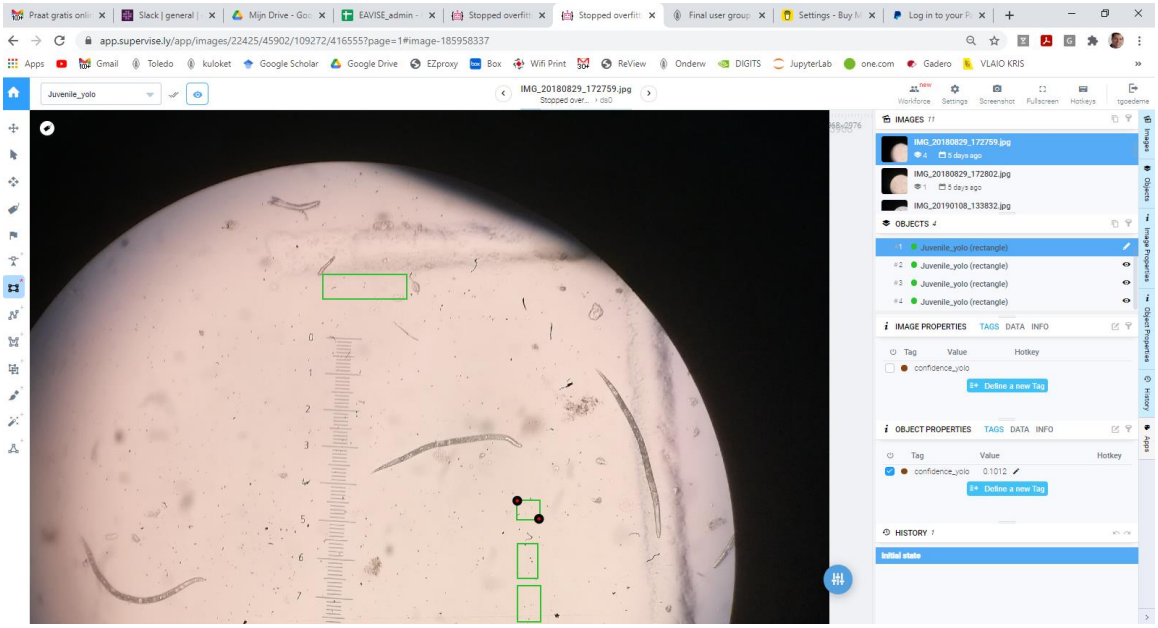


We will choose “full_image” as configuration as we would like our model to search for the objects over the entire image.

In the codeblock below that, you can see that the detected objects will have a name with the suffix “_yolo” added. Another important setting in that script is the “min_confidence_threshold”. This sets the threshold on the detection score of an object, in order for it to be shown as result. From our video (https://www.youtube.com/watch?v=i_cJkcAD8Wo) with the results of the nematode egg counting case study, you can see that we experimented with this threshold to find the value that makes the counting results the most accurate. In our experiment, for Tiny YOLO v3, the best threshold was at 0.37. However, in this phase after your first quick training, you can better set this confidence threshold a bit lower (e.g. at 0.1), such that any resulting detections are surely shown.

Click on “Run”. Now you can see the task is executed, pumping all your testing images through the model. This is called “inference”.

When the task is done, you see that a new dataset appeared in your Workspace, containing the test results. Clicking on it will bring you to the individual images, on which detections are displayed with a bounding box:



If you select one of the detected objects in the “Objects” list on the right (indeed, detections have a name that ends with “_yolo”), you can see its properties in the “Object Properties” box below it. In the example of my result in the figure above, you can see that the object detected (indicated with the thick circles on the bounding box corners) only has a detection confidence of 0.1012, only just above the threshold of 0.1 I set in the previous step. In my example, the model is clearly not yet performing well, as the correct objects are not found and erroneous objects are detected, albeit with a small confidence.

7. Making your detection model better

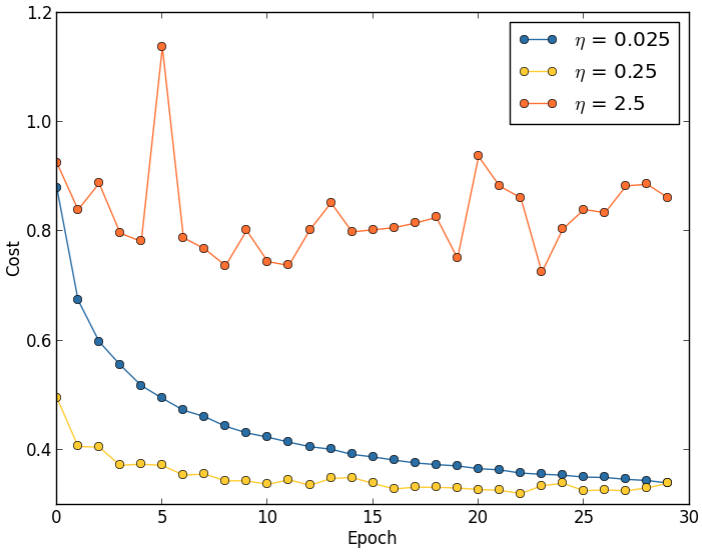
Now that you achieved the first results, you will most certainly see that there is (much) room for improvement. Here are some obvious things you can try to get better detection results:

- Train longer: Your first model was only trained for 5 epochs. You can obviously try to train longer, e.g. for 50 epochs. You can set this via the training script parameter “epochs”. To train further on a model you already trained, you can simply click on the “Train” button next to the model in the Neural Networks page:

You will see that, on the next page you got, the input model will be the first trained model.

- Add more training examples: if the number of training examples is not high enough, you will quickly see that the training is overfitting (you can see this by the fact that the validation loss goes up after a while, while the training loss keeps on going down). The solution is gathering more training examples: new images that are annotated. It is very difficult to know beforehand how many training images will be needed for a certain object detector. Adding more training data will always increase the detection accuracy of a model.

- Play with the other hyperparameters: via the training script, you have access to the learning rate (“lr”), batch size (“batch”) and momentum (“bn_momentum”). The learning rate should be a first to get experimenting with, as you will see its influence quickly on the training graph. As illustrated below, a learning rate that is too high will yield a graph that is very jumpy and that does not goes down. If the learning rate is too low, the training will only converge very slowly towards an optimum.



You can also experiment with a stepped training, e.g. in three steps where after each step the learning rate is decreased with a factor 10. In Supervisely, this can be done by training multiple times further on the same model, each time with a different learning rate.

- Try more data augmentation, to enlarge your training data set artificially. For this, you can manually adapt the “Create train set” DTL script and add more operations, like rotations and contrast changes for instance.

- Try a more complex model, such as YOLO v3 (not the Tiny version). This model is more capable to better detect difficult objects, but needs more training data (and time) to train.

8. Conclusion

In this tutorial, we guided you through the process of training an object detection model without writing any code. For this, we used the Supervisely environment. This seems indeed an interesting choice as platform to quickly prototype neural networks to do e.g. object detection in images. We covered setting up the environment, preparing data, training a model, testing a model and how to make your detection model better.

The resulting model can be exported from Supervisely to deploy it, but that part will involve some programming skills.

Good luck experimenting with deep learning models! Me and my colleagues at the KU Leuven research group EAVISE (www.eavise.be), specialized in industrial computer vision applications, are always ready to answer any questions, or help companies that want to go further with this deep learning technology.

9. Acknowledgements

This tutorial is part of the end result of the Tetra project “Start To Deep Learn”, funded by VLAIO. We have no affiliation with Supervisely.