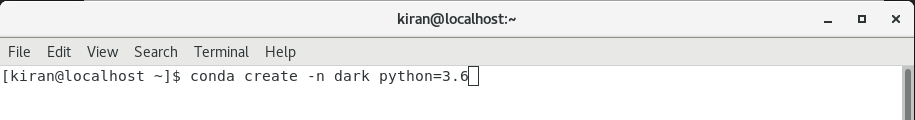
**Watermark Detection Using YOLO Model**

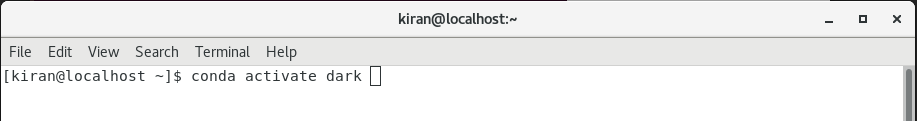
**Overview**

1. Installation
   1. Create a new Conda virtual environment (Optional)

conda create -n dark python=3.6

Screenshot:

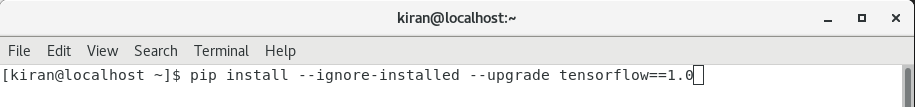




* 1. Install TensorFlow GPU for Python

pip install --ignore-installed --upgrade tensorflow==1.0

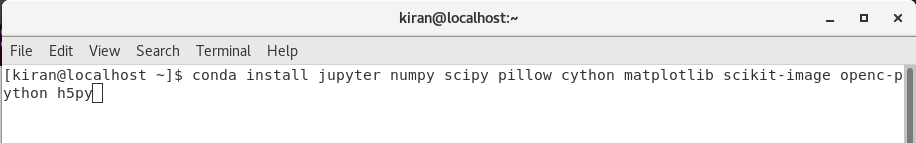
Screenshot:



* 1. Install Prerequisites

conda install jupyter, numpy, scipy, pillow, cython, matplotlib, scikit-image, opencv-python, h5py

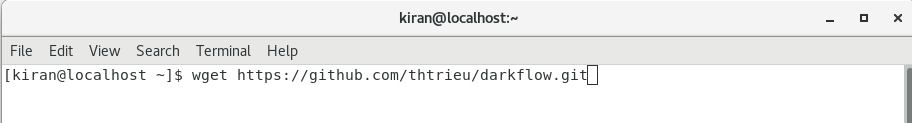
Screenshot:



1. Downloading the DarkFlow Repository

wget https://github.com/thtrieu/darkflow.git

Screenshot:



1. DarkFlow Installations
2. Just build the Cython extensions in place. NOTE: If installing this way you will have to use ./flow in the cloned darkflow directory instead of flow as darkflow is not installed globally.

* python3 setup.py build\_ext --inplace

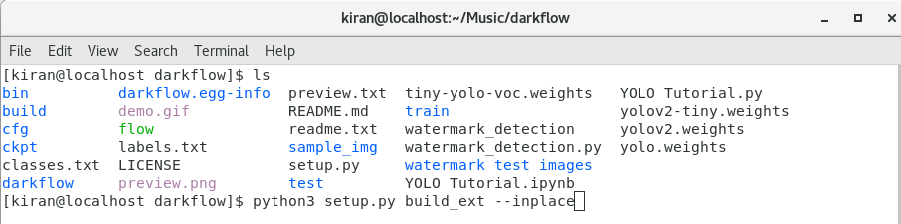
1. Let pip install darkflow globally in dev mode (still globally accessible, but changes to the code immediately take effect)

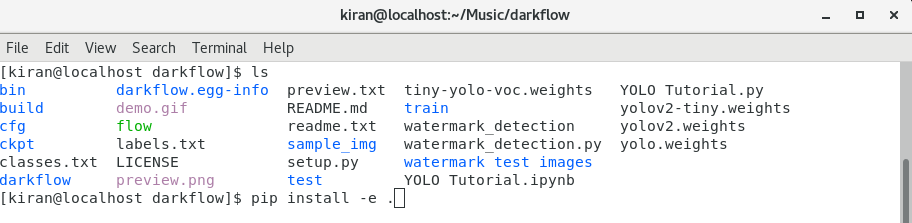
* pip install -e .

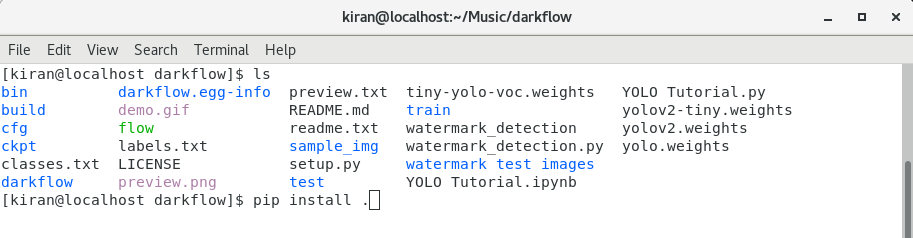
1. Install with pip globally

* pip install .

Screenshots:







1. Modifying configuration Files (Configuring the network)

There is two possible network configurations that can be used to train, yolo or tiny-yolo. As the name suggest tiny-yolo is a smaller network, that obviously will be faster to process but will suffer from lower accuracy. Under cfg/ there are configuration files for both of these versions:

$ ls -1 cfg/ | grep yolo.cfg

tiny-yolo.cfg

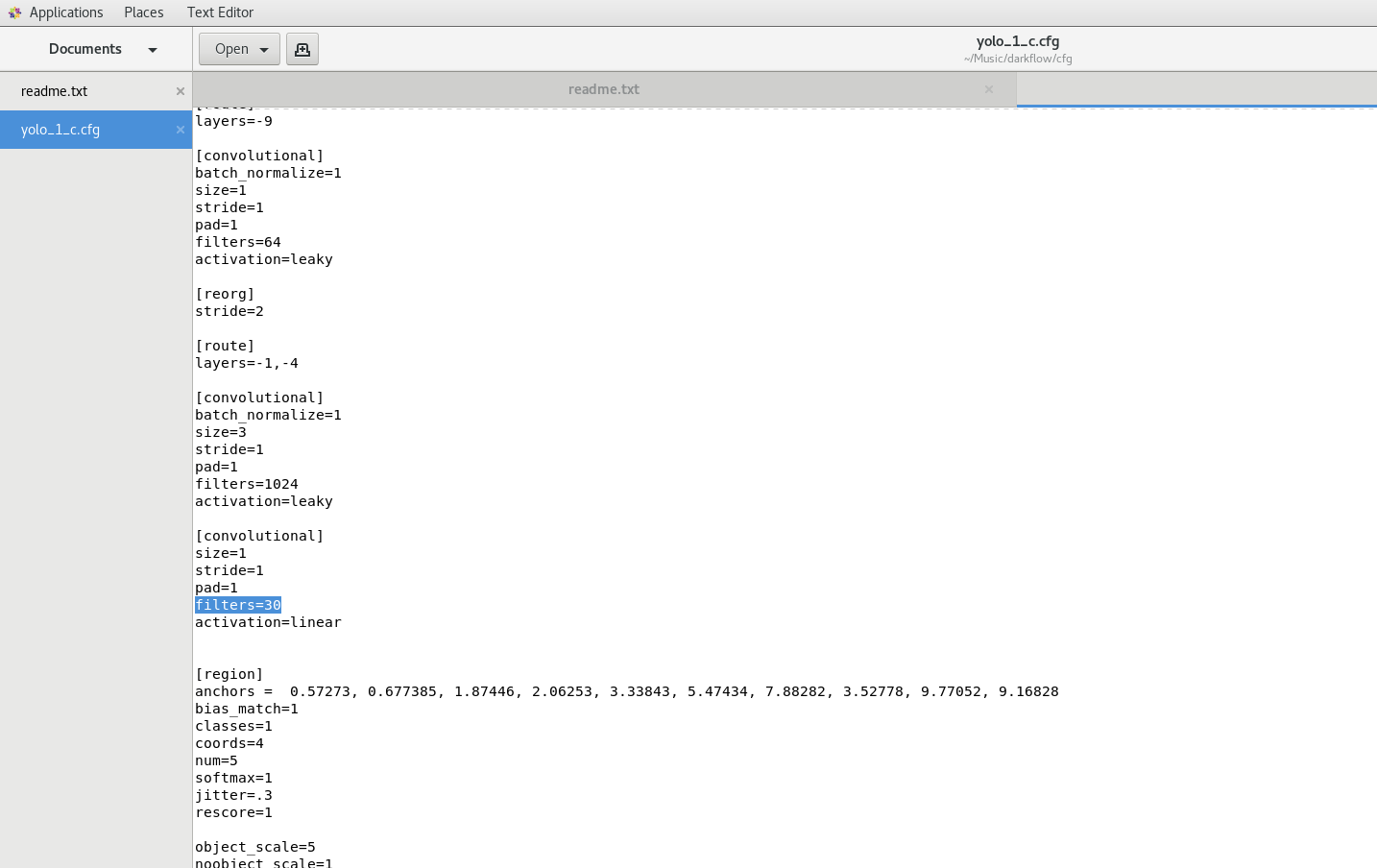
yolo.cfg

For this example, we will use the full yolo configuration, for that we need to create a copy of this file yolo.cfg, that we will need to modify for our problem

We need to modify two lines:

* In the last [convolutional] section, we need to change the number of filters , the formula is filters=(number of classes + 5)\*5 , since we have only one class, we set filters=30 .
* Under the section [region] there is a line to specify the number of classes (around line 244), change it to classes=1 or the number of classes you have.

Screenshots:

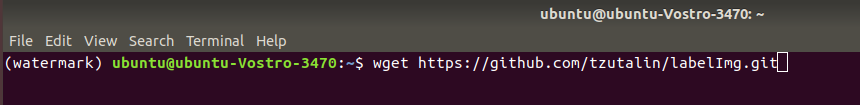


1. LabelImg Installation

* To annotate images we will be using the [labelImg](https://github.com/tzutalin/labelImg) package. If you haven’t installed the package yet, then download from below link.

wget <https://github.com/tzutalin/labelImg.git>

Screenshot:



Once you have collected all the images to be used to test your model (ideally more than 100 per class).

Open a new *Anaconda/Command Prompt* window and cd into labelImg.

Next go ahead and start labelImg, pointing it to your images folder.

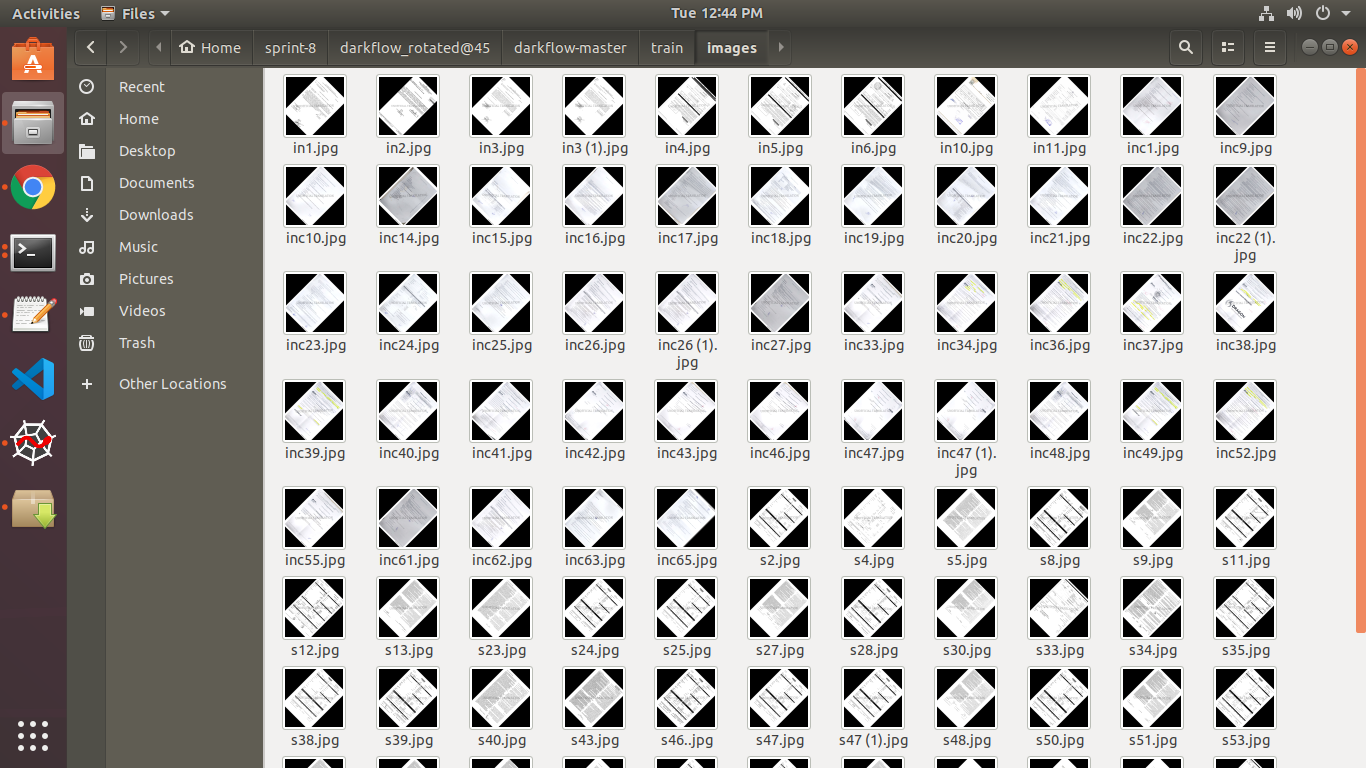
python labelImg.py ..\..\images

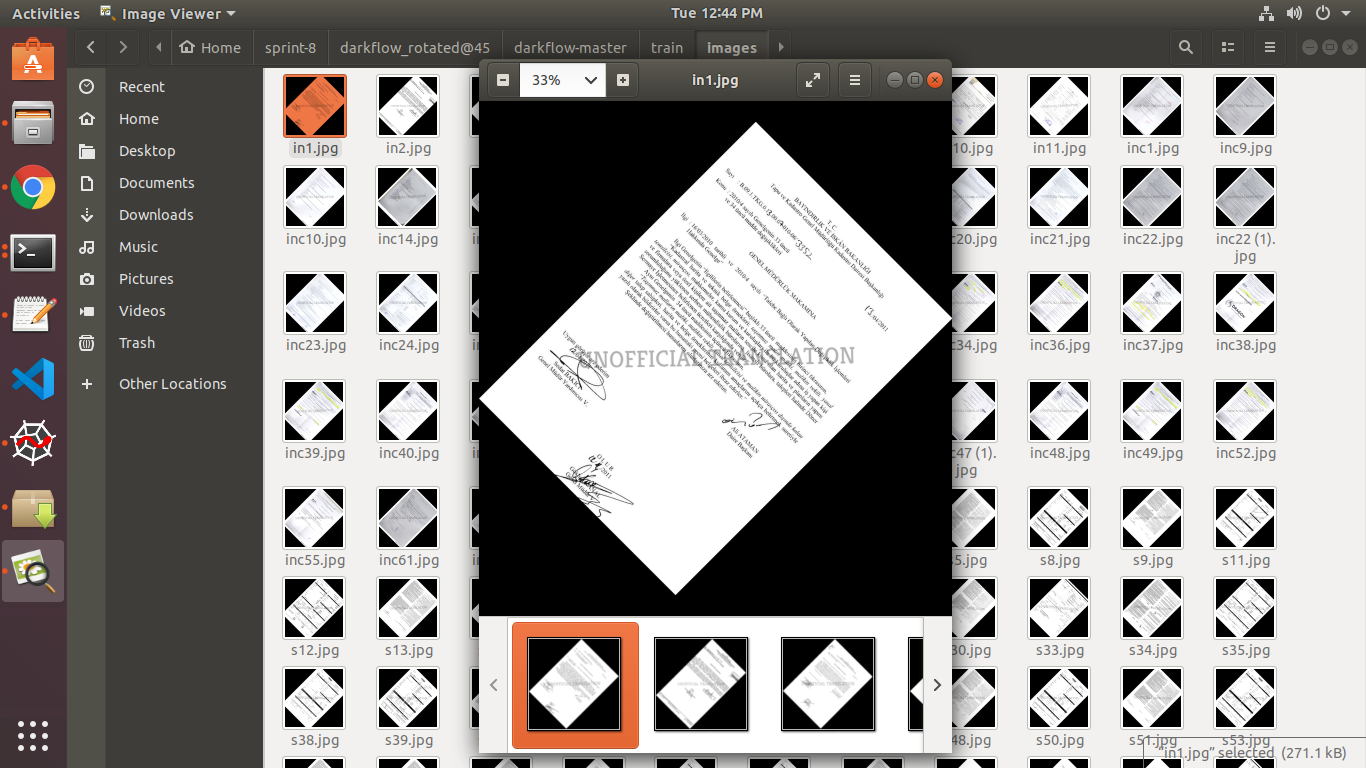
A File Explorer Dialog windows should open, which points to the images folder.

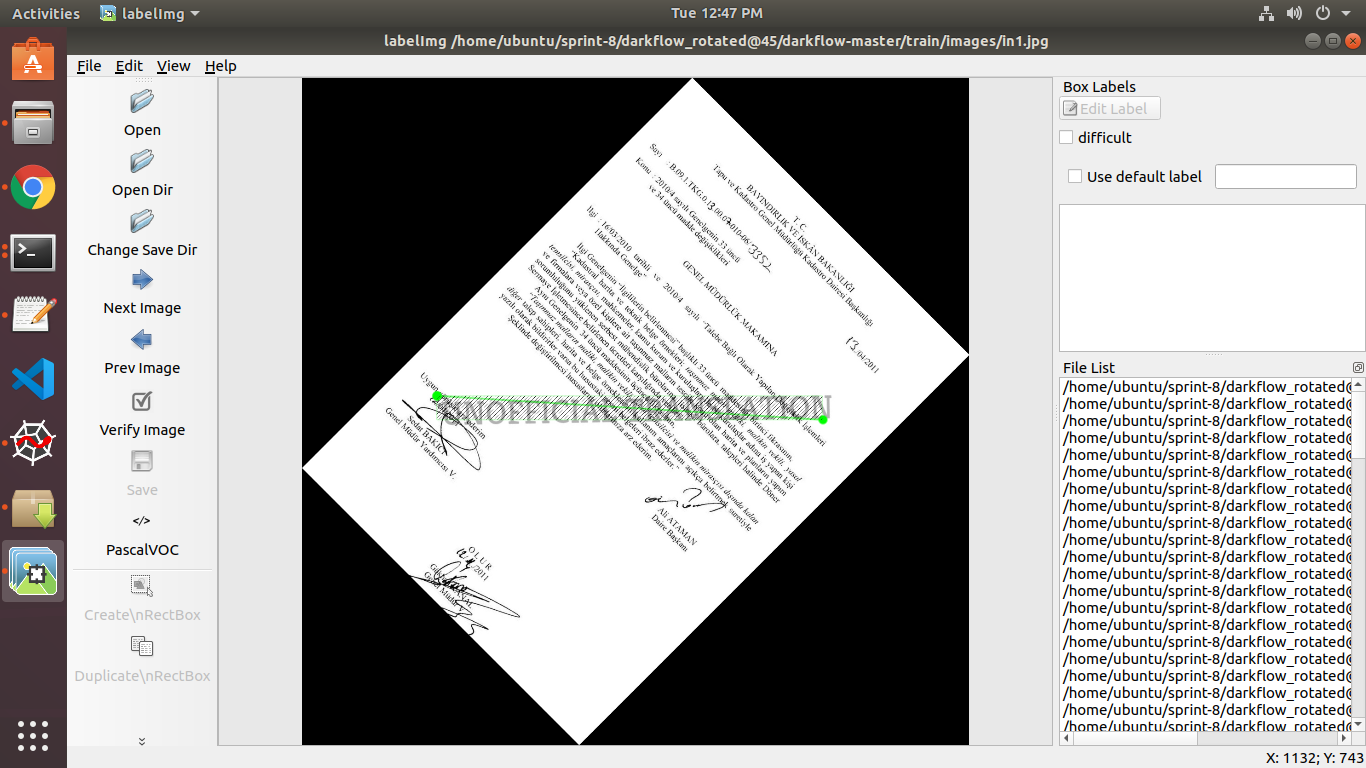
Press the “Select Folder” button, to start annotating your images.

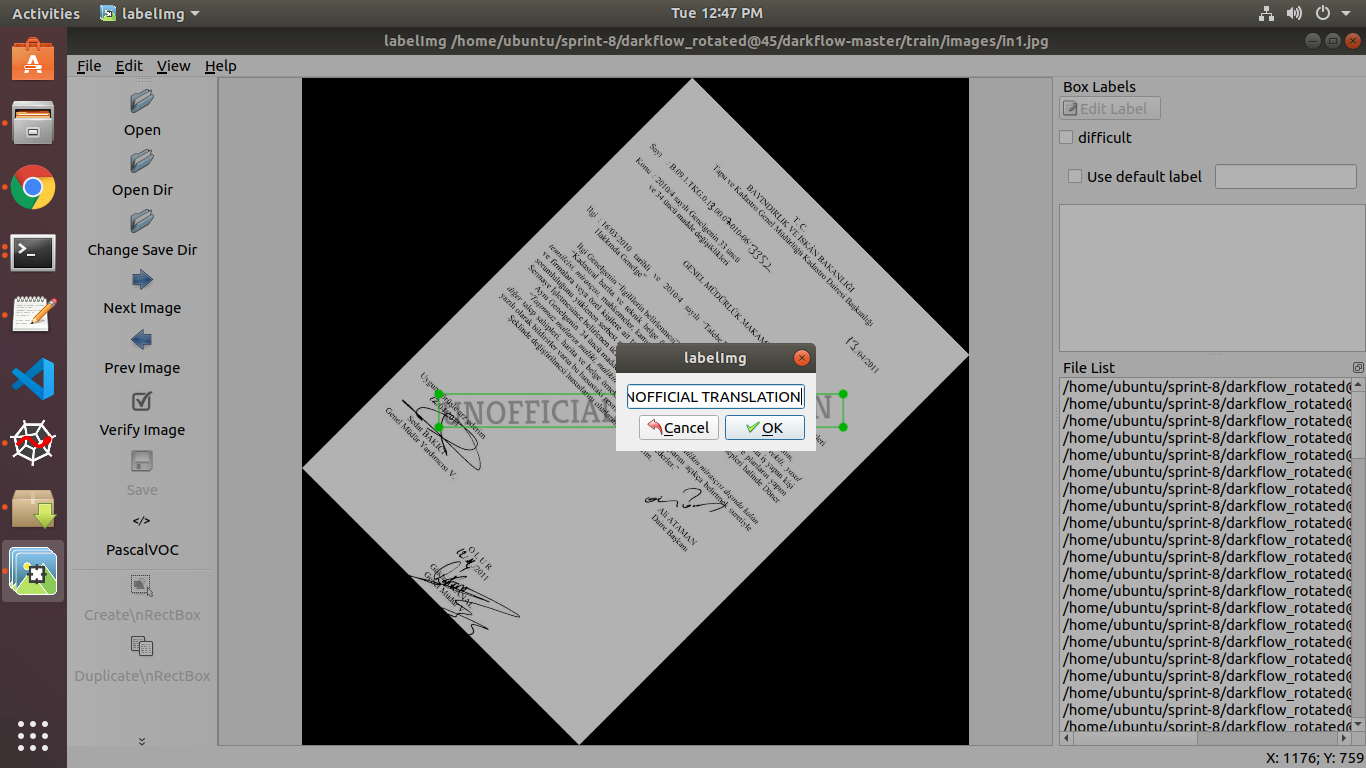
After splitting your dataset, copy all training images, together with their corresponding \*.xml files, and place them inside the images\train folder. Similarly, copy all testing images, with their \*.xml files, and paste them inside images\train.

Screenshots:









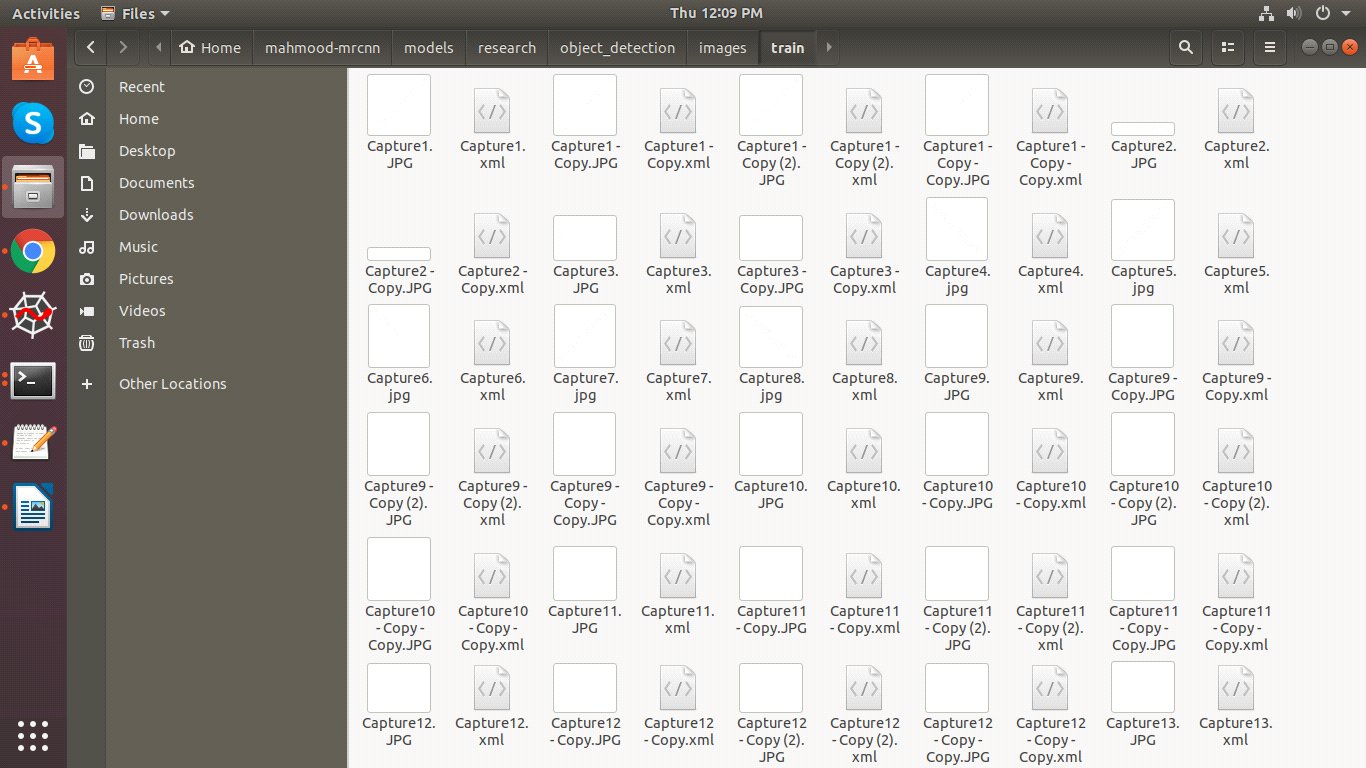
1. Gathering data

Before we can get started creating the object detector we need data, which we can use for training.

To train a robust classifier, we need a lot of pictures which should differ a lot from each other. So they should have different backgrounds, random object, and varying lighting conditions.

You can either take the pictures yourself or you can download them from the internet. For my microcontroller detector, I took about 25 pictures of each individual microcontroller and 25 pictures containing multiple microcontrollers.

Screenshots:



1. Starting the Training

Training is simple as you only have to add option --train. Training set and annotation will be parsed if this is the first time a new configuration is trained. To point to training set and annotations, use option --dataset and --annotation. A few examples:

# Initialize yolo-new from yolo-tiny, then train the net on 100% GPU:

flow --model cfg/yolo-new.cfg --load bin/tiny-yolo.weights --train --gpu 1.0

# Completely initialize yolo-new and train it with ADAM optimizer

flow --model cfg/yolo-new.cfg --train --trainer adam

During training, the script will occasionally save intermediate results into Tensorflow checkpoints, stored in ckpt/. To resume to any checkpoint before performing training/testing, use --load [checkpoint\_num] option, if checkpoint\_num < 0, darkflow will load the most recent save by parsing ckpt/checkpoint.

# Resume the most recent checkpoint for training

flow --train --model cfg/yolo-new.cfg --load -1

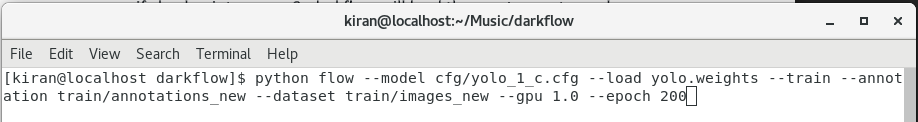
# Test with checkpoint at step 1500

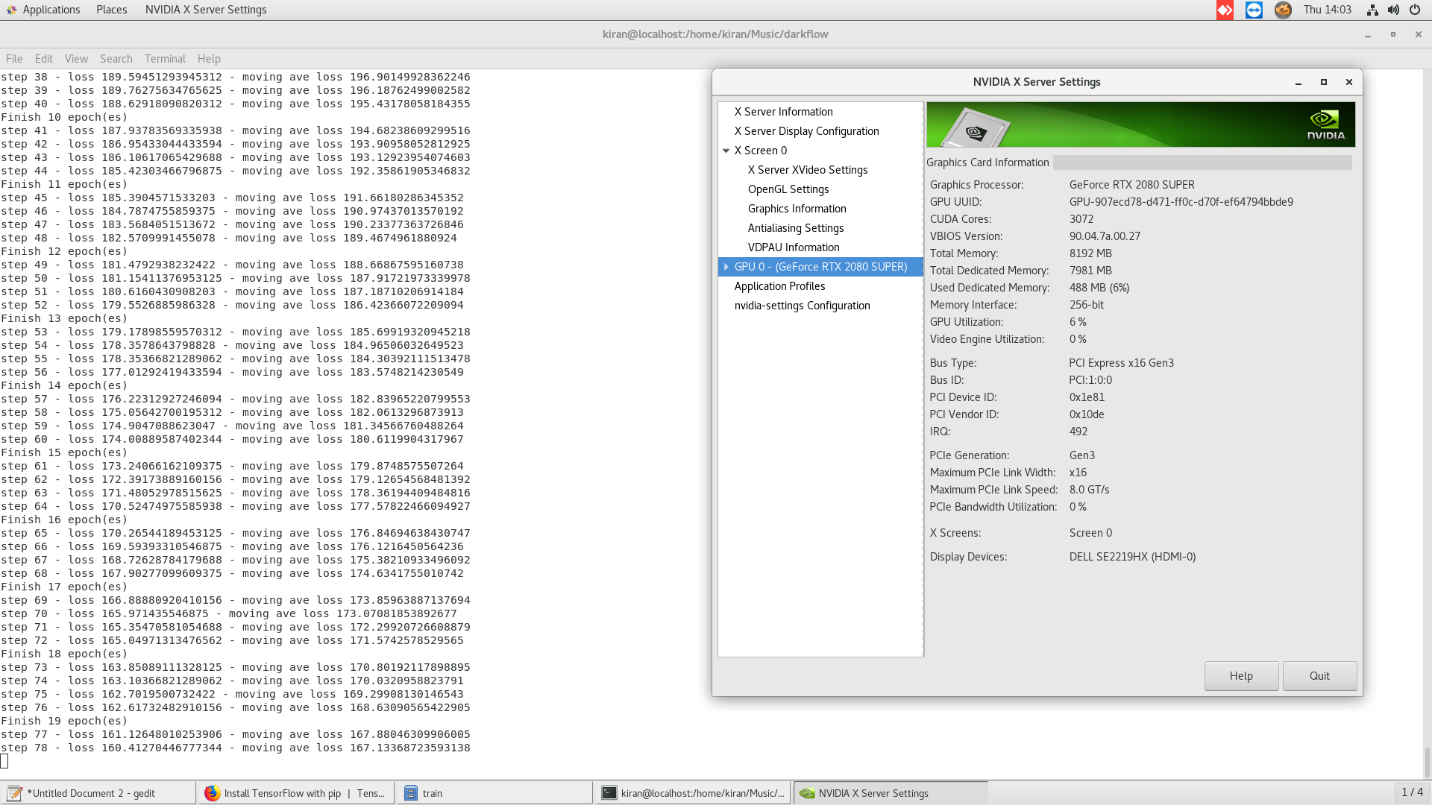
flow --model cfg/yolo-new.cfg --load 1500

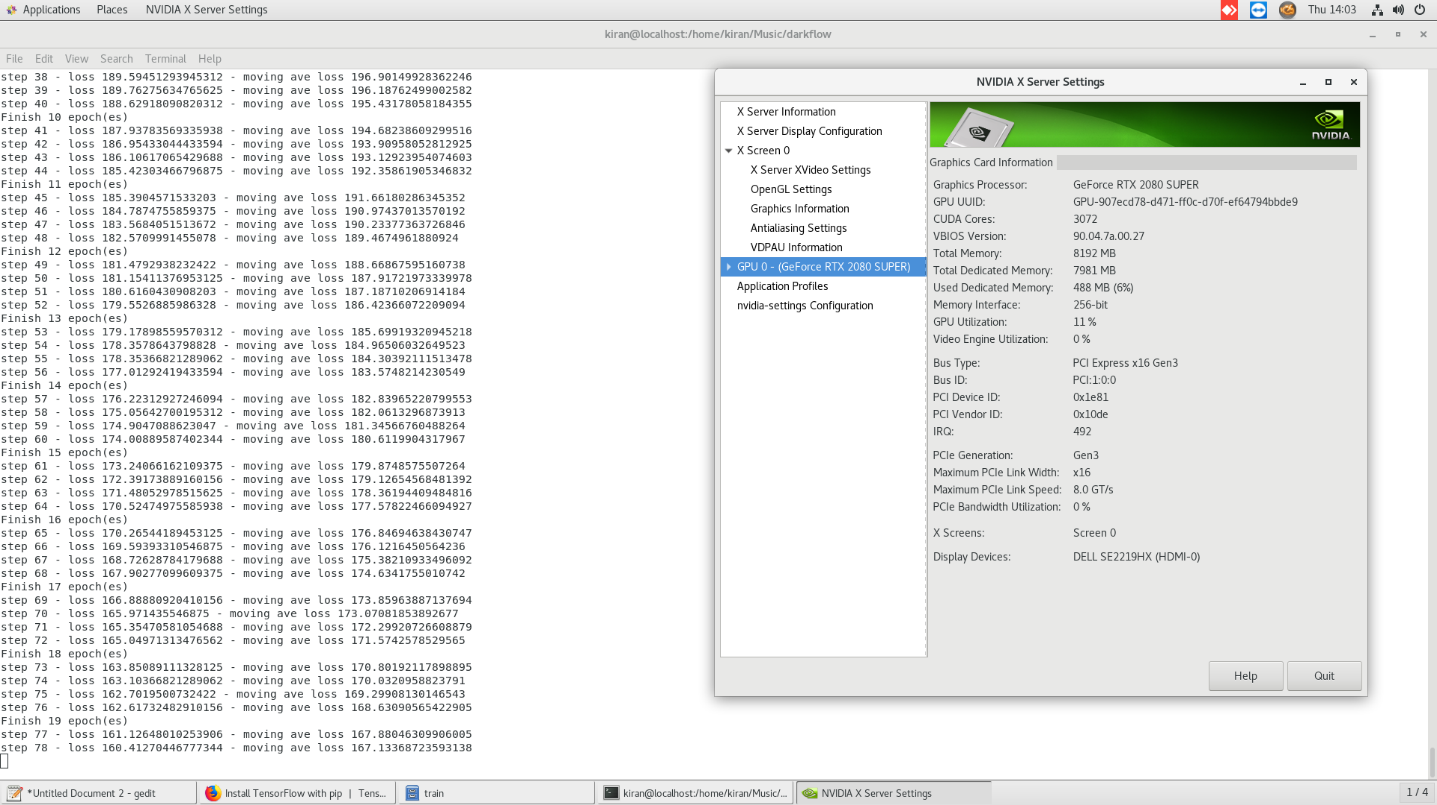
# Fine tuning yolo-tiny from the original one

flow --train --model cfg/tiny-yolo.cfg --load bin/tiny-yolo.weights

Screenshot:







1. Exporting inference graph to protobuf file (.pb)

## Saving the lastest checkpoint to protobuf file

flow --model cfg/yolo-new.cfg --load -1 --savepb

## Saving graph and weights to protobuf file

flow --model cfg/yolo.cfg --load bin/yolo.weights --savepb

When saving the .pb file, a .meta file will also be generated alongside it. This .meta file is a JSON dump of everything in the meta dictionary that contains information nessecary for post-processing such as anchors and labels. This way, everything you need to make predictions from the graph and do post processing is contained in those two files - no need to have the .cfg or any labels file tagging along.

Also, darkflow supports loading from a .pb and .meta file for generating predictions (instead of loading from a .cfg and checkpoint or .weights).

## Forward images in sample\_img for predictions based on protobuf file

flow --pbLoad built\_graph/yolo.pb --metaLoad built\_graph/yolo.meta --imgdir sample\_img/

If you'd like to load a .pb and .meta file when using return\_predict() you can set the "pbLoad" and "metaLoad" options in place of the "model" and "load" options you would normally set.

1. Testing YOLO Watermark detector

In order to test our newly created object detector, we need to run our object detection code with our input image. Now we can run all the cells and we will see a new prediction.

Screenshots:

