**Training the Chess piece object detection model**

The first thing we need to do is gather data – a proposed 200 images per chess piece to ensure a moderate amount of data to train the model on. The pictures are taken with an iPhone in varying lighting conditions and among other chess pieces to ensure a varied dataset to train on.

All data to be kept in DATA folder under abbreviated name folders eg. RK for red knight.

We move the images to /object detection/images and using 

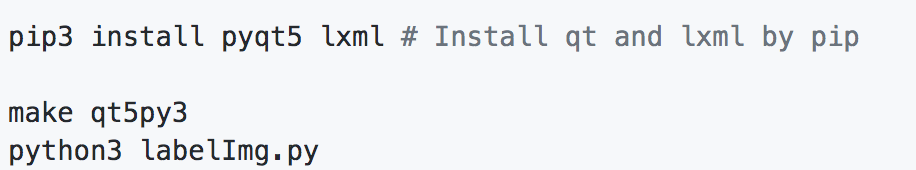
we resize the images to 1/8 of their size, significantly reducing the size of the image therefore the amount of time needed to train the model on.

**A screenshot of a cell phone

Description automatically generated**

Once the images have been resized, we then use the labelimg tool to label them, using W to select the bounding box tool, and cmd­+S to save them, then using D to go to the next image. Making sure we are in the same directory as the labelimg.py file.

git clone https://github.com/tzutalin/labelImg.git



We use pascalVOC to create bounding boxes.

Total number of images = 2(200x6) for red and green sets of chess pieces. The data will be split 50/50 to classify and label in order to reduce time taken to label data.

We then move 80% to training folder and 20% to test folder and move them to the /object detection/images folder.

We then generate .csv files using xml\_to\_csv.py for each image in the same directory.

A screenshot of a cell phone

Description automatically generated



Thiscreates two files in the images directory. One called *test\_labels.csv* and another one called *train\_labels.csv*.

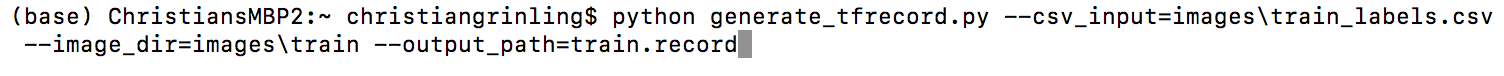
We then use the generate\_tfrecords.py file to generate the tf.records.

A close up of a screen

Description automatically generated

First we need to change the labels to our Chess piece values.

Then we generate the records using



python generate\_tfrecord.py --csv\_input=images\train\_labels.csv --image\_dir=images\train --output\_path=train.record

A close up of a device

Description automatically generated

python generate\_tfrecord.py --csv\_input=images\test\_labels.csv --image\_dir=images\test --output\_path=test.record

Which create a train.record and a test.record file.

We then create a folder in object detection called training and create a label map and save it as labelmap.pbtxt

Screen of a cell phone

Description automatically generated

We have to replace the values with the same ones defined In our tfrecords file.

We then go to <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>

To download the faster\_rcnn\_inception\_v2\_pets model. We then copy the .config file into our training folder.

We need to make a few changes to the file:

A screenshot of a cell phone

Description automatically generated

Once all of that is finished we start training the model using

python train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ faster\_rcnn\_inception\_v2\_pets.config

We can monitor the training process using

tensorboard --logdir=training

Next we need to export the inference graph to use our model.

We need to first of find out the highest saved step number. For this, we need to navigate to the training directory and look for the model.ckpt file with the biggest index.

python export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path training/faster\_rcnn\_inception\_v2\_pets.config --trained\_checkpoint\_prefix training/model.ckpt-XXXX --output\_directory inference\_graph

And replace the xxxx with the highest number.

To run the model we need to replace our object detection webcam code from

A screenshot of a cell phone

Description automatically generated