Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Independent Course work Report

Object Detection with Bounding Boxes through Machine Learning

Caglar Özel

supervised by
Prof. Dr. Klaus Jung

Contents

Abstract

Today where computer software and technologies are developing fast and new usages are being discovered frequently, Machine Learning is evolving and redefining Image Processing and Object / Image Detection as we knew it.

In this report I will talk about visual information retrieval (computer vision) through machine learning where I will use state of the art object detection systems depending on region proposal algorithms such as faster region-based convolutional neural networks (Faster R-CNN) and single shot detection (SSD) networks.

The end goal is to understand how these models work, train one and have a application which will utilize the Tensorflow framework with the Neural Network Models it uses to perform Object Detection on visual media. The result of this Object Detection will be an image, video or live footage through a webcam with objects highlighted through bounding boxes.

Introduction

While humans effortlessly recognize shapes and objects through our complex visual cortex system where we have millions of neurons and billions of connections between them, which were fine tuned through generations. The difficulty of visual pattern detection becomes apparent when you try implemented the logic in a algorithm. [1]

Object detection which is a technology related to computer vision or image processing is a field that deals with the detection of objects in a digital image or video.
Machine Learning on the other hand is a subset of Artificial Intelligence where the computer will perform tasks through algorithms and models without explicit instructions, relying on patterns and models instead.

Combining these two technologies with the recent advances in object detection through region proposal methods and region-based convolutional neural networks. Opens up new possibilities which were impossible to achieve before through low level feature analysis only such as mean color comparison, shape, filter analysis and other methods.

Region proposal methods typically rely on inexpensive features and economical inference schemes. Selective Search, one of the most popular methods, greedily merges superpixels based on engineered low-level features. Yet when comparing to efficient detection networks, Selective Search takes 2 seconds per Image on a CPU implementation, where as fast region-based CNN's take advantage of the GPU and therefore are not applicable for comparison. [2]

Object Detection

3.1 Brief introduction to Neurons and Networks

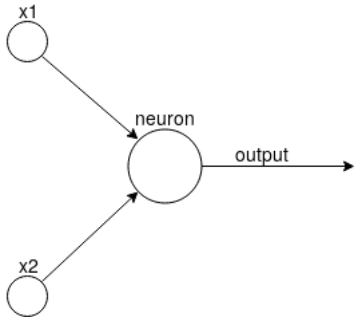
Similar to the neurons in the human brain, neurons in computer science are approaching the same concept and same ideology.
The functionality of a neuron is to retrieve data, possibly and ideally multiple at the same time, process and output a single result. There are many types of neurons, for introduction purposes I am just gonna mention two here.

- Perceptrons
- Sigmoids

The more commonly used artificial neuron in Neural Network models is the Sigmoid. The reasons for that will be mentioned in the chapters belowe.

3.1.1 Percpetrons

Perceptrons were developed in 1950 to 1960 by the scientist Frank Rosenblatt, inspired by earlier works from Waren McCulloch and Walter Pitts.

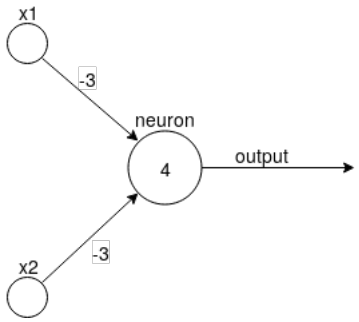


In this example our neuron has 2 input variables and one output, in theory it can have more or less. Further more Frank Rosenblatt proposes the concept of weights for each input variable defining how important a variable is compared to another.

$$output = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq threshold \\ 1 \text{ if } \sum_j w_j x_j > threshold \end{cases}$$

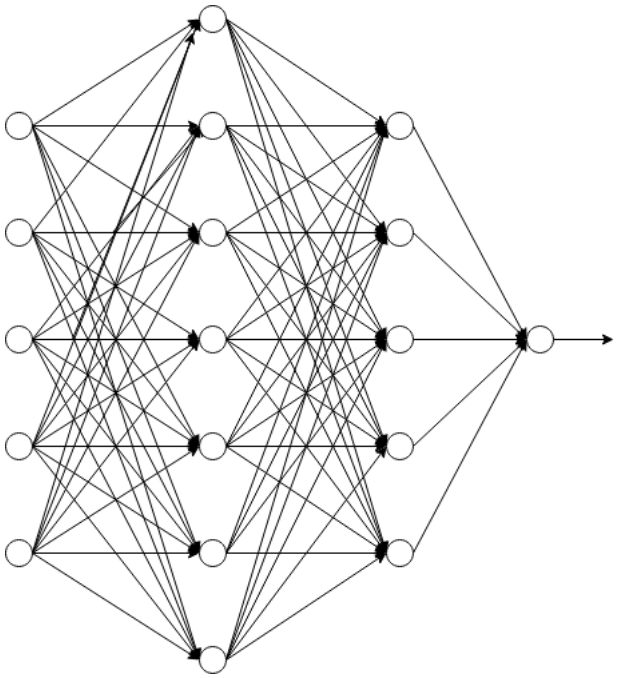
A single neuron is by no means comparable to a decision making possibilities of a human being, but

it can weigh variables and make decisions accordingly by defining a threshold for when a true or false gets output. Another way to create a conditional behavior is to use a bias instead of a threshold.



$$output = \begin{cases} 0 \text{ if } \sum_j w_j x_j + b \leq 0 \\ 1 \text{ if } \sum_j w_j x_j + b > 0 \end{cases}$$

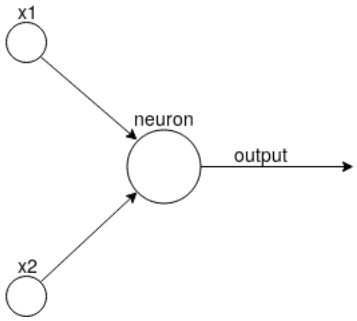
Making it obvious that they are able to perform more complex decisions in layers.



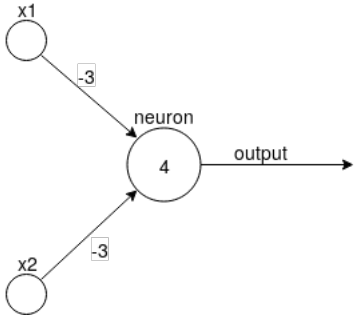
The columns in this network are called layers, the first column is refered to as the first layers and the last one as the output one all the ones in between are called hidden layers.
Since the result of a perceptron can only be one or zero (true or false), one major problem someone faces when using perceptrons in more complex neural networks is that biases and weights can have weird and unwanted effects. Making perceptrons trigger where they should not, thats where sigmoids come in handy and build on top of the concept of perceptrons.

3.1.2 Sigmoids

The core principle of a sigmoid neuron is the same as the perceptrons. Just like perceptrons, sigmoids have input and output variables but instead of just being 0 or 1 it can be anything in between 0 and 1 so 0.5234 is a valid output of a sigmoid.



It can also have weights and biases similiar to the perceptron:



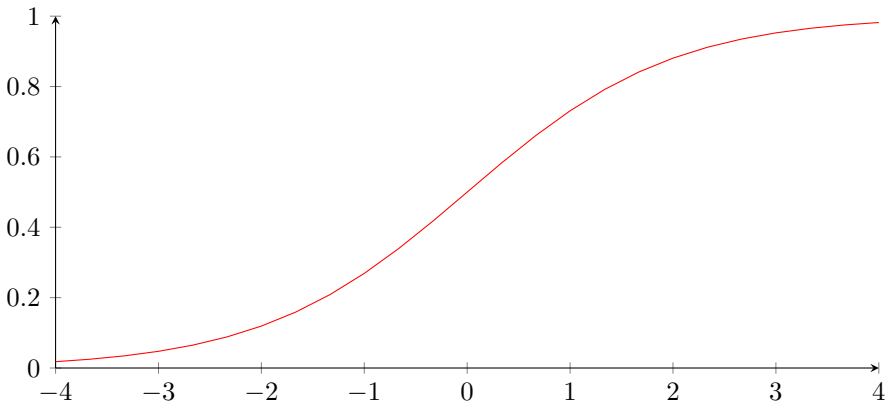
The arithmetic function on the other looks different and is called the sigmoid function or sometimes the logistic function.

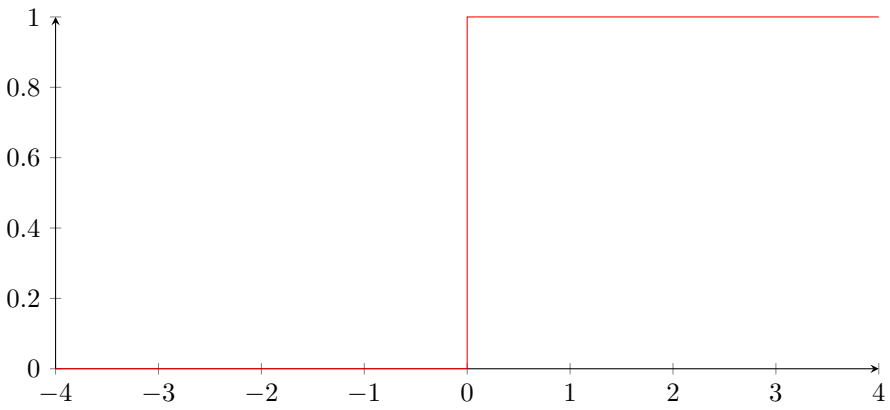
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

To put it more explicit with input variables, weights and bias:

$$\frac{1}{1 + exp(-\sum_j w_j x_j - b)}$$

Since the algebraic form of σ a is smoothed and not a step function this difference is minimal but makes a huge difference in the way how weights and bias influence the result of a sigmoid compared to a percpetron.


1
0.8
0.6
0.4
0.2
0
−4
−3
−2
−1
0
1
2
3
4


1
0.8
0.6
0.4
0.2
0
−4
−3
−2
−1
0
1
2
3
4

3.2 Current state

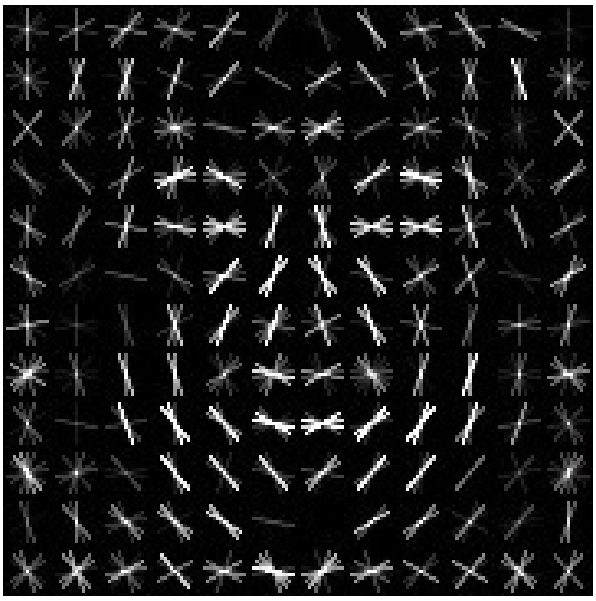
Object Detection in dealing with two major problems and therefore can be split into two processes, these are object localization and object classification of multiple objects in one image. There are many approaches in current Neural Network models to perform these actions. There are models which have a multi layer architecture, each performing complex tasks such as analyzing, localizing and classifying each sections individually, making the cost high and the network in general slow but accurate. There models that combine functionality in one layer so that one layer for instance localizes and classifies at the same time, is making the model more flexible, smaller and therefore faster but less accurate. The trade off between accuracy and speed is something that all networks deal with nowadays especially if the goal is to have a near real life object detection system which is still reliable and accurate when doing so.
There are multiple algorithms covering various forms of this formula, called RCNN, F-RCNN, SSD, YOLO but first we will cover the approach of detecting interesting regions in an image because this is a key problem in object detection.

3.2.1 Object Localization & Classification

The segmentation of an image is the key approach to Object Localization and at the same time for Object Classification. Since images are always hierarchical the segmentation as well has to be hierarchical. It has to be able to cover different criteria and forms, where maybe objects are part of a bigger total or just smaller objects which are inside one another. Since is not feasible to compute every possibility inside an image there has to be some kind of separation through grids and scales.
As there are many approaches to algorithms as how to analyze and classify regions I am going to cover three localization algorithms.

Exhaustive Search

Since an object can be located anywhere in an image the scope to look through is enormous. Making the computational cost for such a search very expensive. To compensate for this cost most exhaustive search approaches such as the sliding window approach, constraint the size and aspect ratio of the window over the grid, using weak classifiers and economic image features such as histogram of oriented gradients (HOG). [3]
HOG approaches the problem of recognizing and detecting objects in an unknown image through corners. It analyzes the distribution and their orientation, through which he is able to separate the image in multiple partitions or detect similar regions in different images. Robert K. McConnell described the concept in a patent in 1986 but it became known in 2005 through a publication of Navneet Dalal and Bill Triggs. [**?**]

 [?]

Selective Search

A selective search algorithm is subjected into 3 design ideas:

- Capture All Scales

  The idea is similar to the one of the Exhaustive Search that all objects have to be found. Therefore all scales have to be considered as a potential region. The best approach is a hierarchical grouping, where a initial sub segmentation has to be performed. [3]

 [4]

  After the segmentation a grouping of regions is done by similarity. After each grouping a similarity between resulting regions and its neighbors are being calculated. This process is done until the whole image becomes a single region. [3]

[5]

- Diversification

  There is no single strategy for grouping regions together therefore multiple aspects have to be considered. Selective search diversifies by using a variety of color spaces with different invariance properties, by using different similarity measures and by varying the starting regions. By using different color spaces, selective search is able to account for different scene and light conditions. To account for these variances selective search is performing the hierarchical grouping algorithm with different color spaces which have different invariance properties.

- Fast to Compute

  Since this step is a computation step, the main goal of this algorithm is to be fast and not create a bottleneck through is complexity. [3]

S*color*:

Measures color similarity specifically for each region where a color histogram of 25 bins is being created and normalized with the L1 norm. Similarity is measured using the histogram intersection: [3]

$$S_{color}(r_i, r_j) = \sum_{n=1}^{n} min(c_i^k, c_j^k)$$

[3]

The histogram can be efficienlty propagated through the hierarchy by:

$$C_t = \frac{size(r_j) * C_i + size(r_j) * C_j}{size(r_i) + size(r_j)}$$

[3]

S*texture*:

Selective Search measures texture similarity through the usage of the SIFT algorithm. SIFT stands for scale invariant feature transform and is a algorithm to detect and describe features in a region. Further more it takes Gaussian derivatives in eight orientations for each color channel, where a 10 bin histogram is being extracted. Similarity is being measured again using the histogram intersections and the propagation is function is the same as for the color. [3]

$$S_{texture}(r_i, r_j) = \sum_{n=1}^{n} min(t_i^k, t_j^k)$$

[3]

S*size*:

With this step selective search encourages to merge small regions early. By doing so it forces the algorithm to focus on smaller regions which have not been merged yet first. This is encouraged because it hinders the algorithm from one region gobbling up the rest of them one by one. [3]

$$S_{size}(r_i, r_j) = 1 - \frac{size(r_i) + size(r_j)}{size(image)}$$

[3]

S*fill*:

Measures how well region r_i and r_j fit into each other. The idea is to fill gaps if r_i is contained in r_j, therefore it is only logical that these regions should be merged first in order to avoid holes. On the other hand if r_i and r_j hardly touch each other it may form a strange region and therefore should not be merged. To keep the process fast only the size of the regions and their contained bounding boxes are being used. Specifically the defined BB_{ij} has to be a tight bounding box around r_i and r_j. Now S_{fill} is the fraction of the image contained in BB_{ij} which is not by the regions of r_i and r_j. [3]

$$fill(r_i, r_j) = 1 - \frac{size(BB_{ij}) - size(r_i) - size(r_j)}{size(image)}$$

[3]

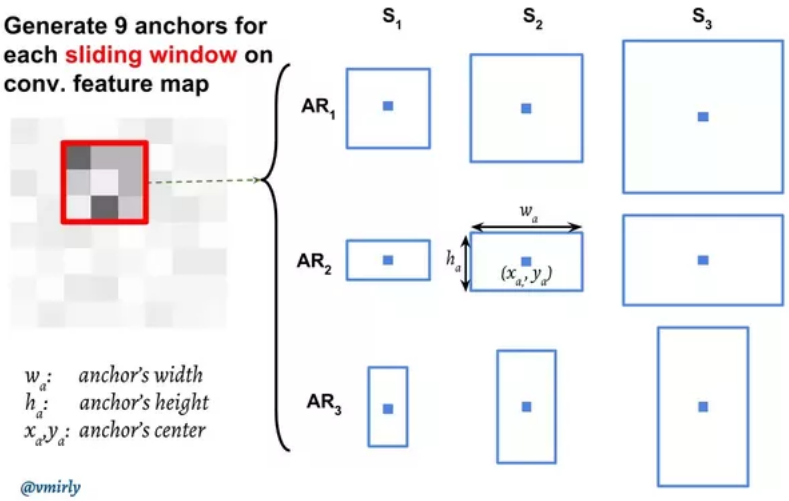
The final formula which measures the similarity is a combination of the above 4:

$$s(r_i, r_j) = a1s_{coulor}(r_i, r_j) + a2s_{texture}(r_i, r_j) + a3s_{size}(r_i, r_j) + a4s_{fill}(r_i, r_j)$$

[3]

Region Proposal Network in Faster R-CNN

A Region Proposal Network (RPN) takes an image of any size as an input and outputs a set of rectangular object proposals, each with an objectness score. After the initial proposals, a sliding window is being run over the feature map. The sliding window is of size n x n. In this example it will be of size 3 x 3. So for each sliding window a set of 9 anchors are being generated, 3 scales and 3 aspect ratios. [2]



Furthermore for each sliding window anchor a value called Intersection of Union short IoU is being calculated. IoU represents the value of how much the predicted bounding box overlaps with the ground truth. The ground truth is the basis for teaching the localization network and is a set of data predefined by a person telling the network where, which objects are in the image.
The general idea of IoU is: [2]

$$p = \begin{cases} 1 & \text{if IoU} > threshold \\ -1 & \text{if IoU} < threshold \\ 0 & otherwise \end{cases}$$

In the final step the results of the spatial sliding window is being fed into a smaller network which has two tasks. Classification (cls) and regression (reg). The output of the regression represents a bounding box with (x, y, w, h) on the other hand the cls value is a probability indicating whether the bounding box contains a object or not. [2]

Single Shot Detector(SSD)

Compared to the Faster R-CNN where the detection happens in two stages. Firstly having the region proposal network generating proposals and the second one then classifying them. SSD is a single feed forwards convolutional network to directly predict classes and anchor offsets without requiring a second stage per-proposal.

Tensorflow

TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. [6]

4.1 Writing Object Detection Models with Tensorflow

Tensorflow is a full fledged machine learning framework which support the programming languages C++, Python. It is multi purpose framework which can be used for training Neural Networks for Image Recognition, Object Detection or other purposes. Furthermore tensorflow uses Keras which is a high level API for building and training deep learning models. For Object Detection for which I am using Tensorflow in this report, you are able to write a Object Detection Neural Network in multiple ways.

- Do it Yourself

  You are able to write your object detection logic from scratch, starting at how features are being extracted processed and evaluated to how layers of your neural networks are connected, sequenced, communicate and how the object localization and object classification is being used and implemented.

- Just write the Model

  Tensorflows object detection model which they provide at `https://github.com/tensorflow/models` is a simple yet effective way of training Object Detection Neural Netowrks. You are able to compose new ideas for models with simple '.config' files without going too deep into the code and logic of tensorflow. These config files contain the setup of the neural network model. For instance in which sequence the layers are going to be, what the IoU value has to be for a bounding box to be valid and which features have to bextracted by the tensorflow api. See appendix7 for a full '.config' file used for training.

4.2 Training Models with Tensorflow

As there are multiple ways of setting up your neural network there are also a couple of ways training it. It start with the choice of your dataset, it can bias orientated meaning that it will recognize one type of image very good but fail on others, in machine learning terms it overfitts. Or it can be

- Train from Scratch

  Which ever way you are training your model, starting from scratch without a basis will require:
  - alot of resources (images, ground truth data)
  - alot of training time

- Train by transfer Learning

  When you are doing transfer learning, you are taking an already trained model and build ontop of it with your own image set.
  This results in:
  - far fewer images need
  - less training time (comparing days and weeks to hours and minutes)

Training is pretty straight forward when using the setup from tensorflow as mentioned in4.1 "Just write the Model". For this you have to install Tensorflow and all the required dependencies. They can be found here: `https://www.tensorflow.org/install/pip`. Futhermore some custom scripts were written to enable convertig txt files to .csv files to enable the generation of .record files and to merge entries of multiple files with the same naming (there were multiple files with the same name and content but labeled differently for each category). Lastly environmental variables, proto files and '.record' files had to be setup, compiled for tensorflow.

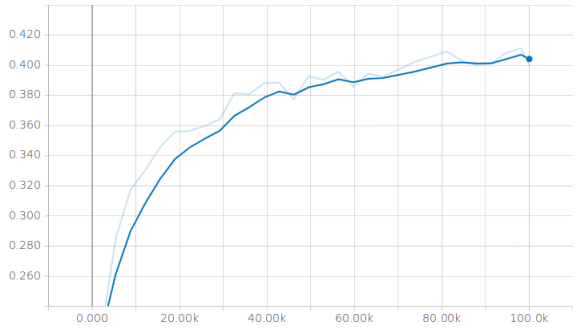
Model	Speed(ms)	mAP[^1]
faster-rcnn-inception-v2-cooc	58	28
ssd-mobilenet-v2-coco	32	22

4.3 Training results

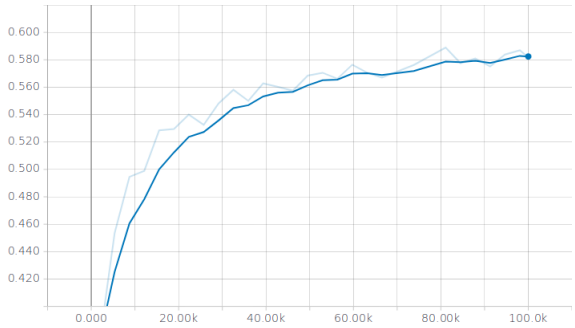
After setting up the environment my first attempt was getting the data myself, labeling them with the Open Source software LabelImg and training a new Model through a tensorflow config. This was a tedious effort and resulted in a bad model which detected images pretty well as long as they were from the train set or similiar but failed on real world data. This was done with various combinations of data quantities which was not enough at all for this, IoU, variable and feature tweaks provided through Tensorflow without much difference in its performance. To note is that these were my first attempts training models with tensorflow and I did not understand the whole procedure too well, meaning I didn't consult tensorboard which is User Interface from Tensorflow to view the results of my Data and Model.
After this I switched to transfer learning which takes the weights and knowledge of a already existing models from tensorflow and applies it to a different one with related problems.
Firstly I downloaded images and bounding box data through the api of ms coco which is a dataset providing more then 2.5 million labele instances in 328k images. [7] There are other datasets such as Pascal VOC and ImageNet but none of them provided a easy way to download data similar to MSCOCO. For the following models I downloaded 9 Classes with 1k images and labels each to test, vary and experiment with the amount of data provided to training my model.
Secondly I donwloaded two models from tensorflow, one was the faster-rcnn-inception-v2-coco and the ssd-mobilnet-v2-coco trained with the dataset from MSCOCO. These are 2 models where the first one is implemented with the Faster-RCNN and the later with SSD algorithm.
When transfer learning I noticed that 200 images for each category was more then enough to get decent results, the results degraded compared to the original Tensorflow models I used for transfer learning

but it was still better then the intial attempts of training. The following graphs are smoothed by a factor of 0.6.

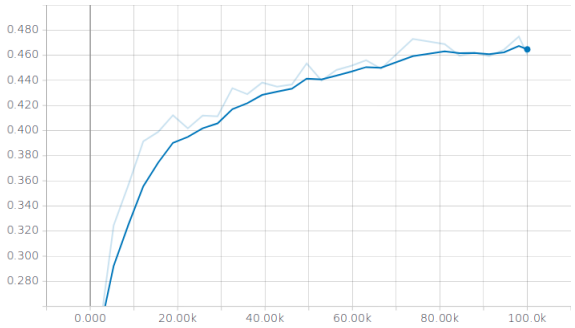
Figure 4.1: mAP



(a) mAP per Stage



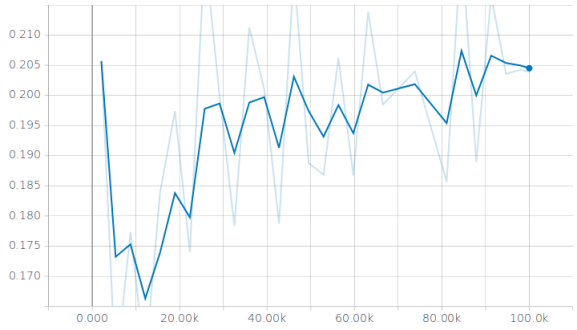
(b) mAP @0.5 IoU



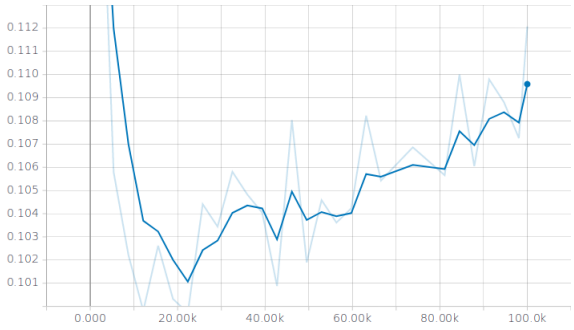
(c) mAP @0.75 IoU

Even though the mAP is increasing over the duration of the training session you can see that it is not really increasing in performance but overfitting through the following grpahs:

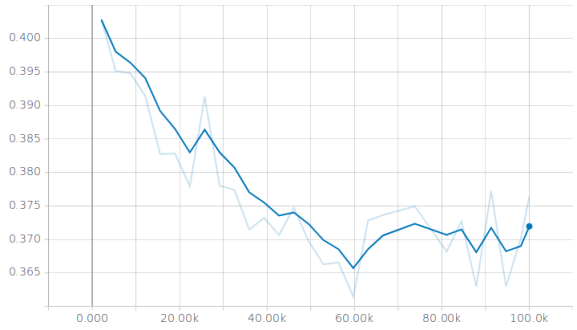
Figure 4.2: Loss of Models



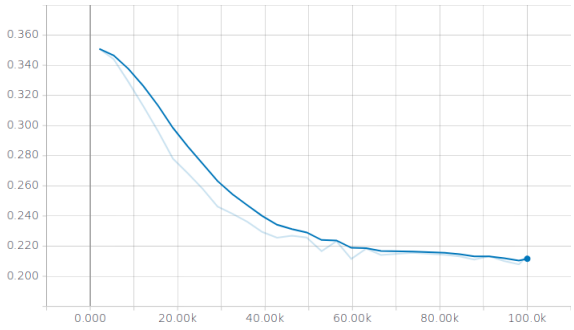
(a) Box Classifier Loss



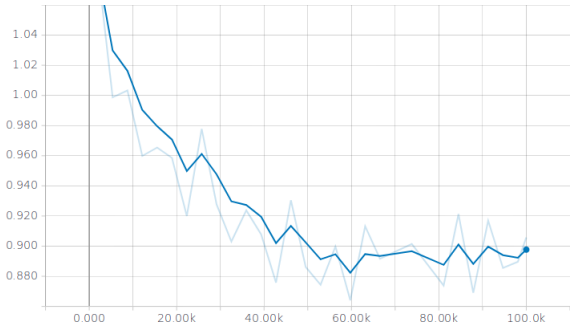
(b) Box Classifier Localization Loss



(c) RPN Localization Loss



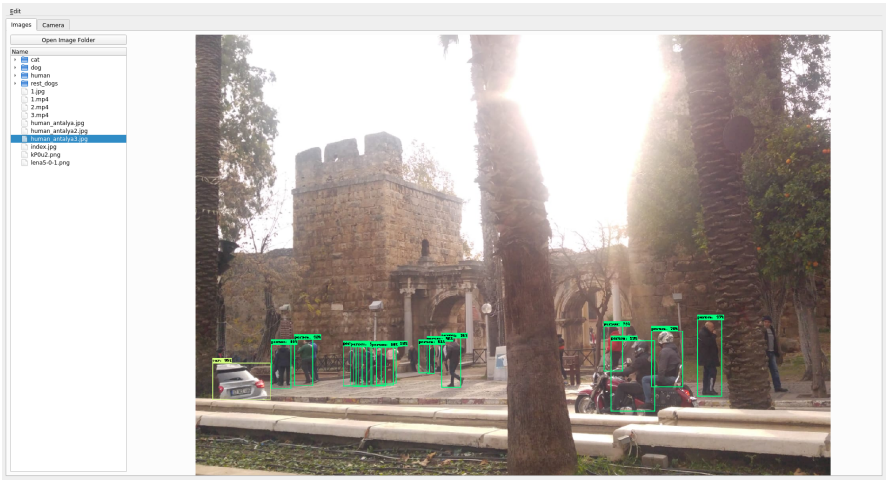
(d) RPN Objectness Loss



(e) Total Loss

Application

The initial phase to create the Application was having a basic script which instantiates a Tensorflow model, processes the image to Tensorflow standards and passes it into the Model for object detection. Using this script as a basis I started developing a application with PyQt version 5. PyQt is a cross-platform GUI toolkit which is similar to Java Swift and is a easy framework to develop a UI with, it takes ideas and concepts from different roots and implements a layer based architecture for elements.

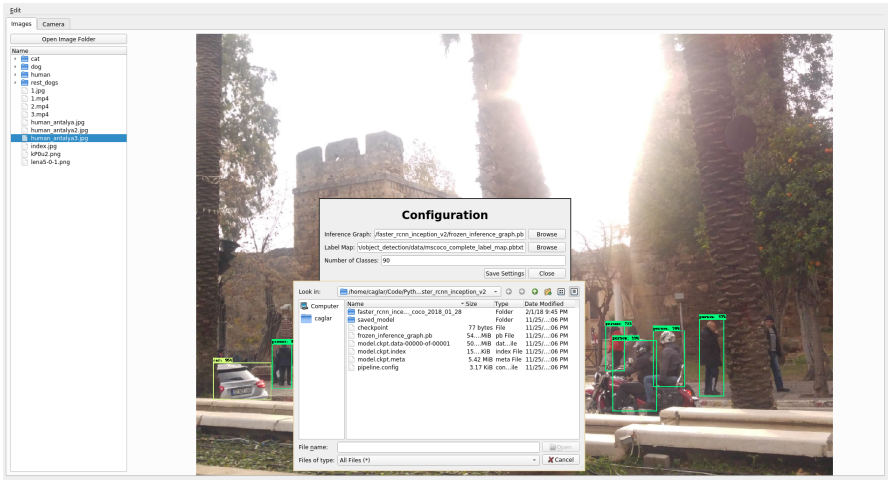


There were several requirements for the Application from the start:

- Use any model trained with Tensorflow

  Since I am using the Tensorflos API for my application this point is being handled by Tensorflow itself.

- Change used model, label map and class number on runtime
  For this I implemented a config class where I will store the path's of the selected inference graph, the label map and the number of classes set by the User. These if existent will be queried for on startup and with them a Tensorflow model will be instantiated.



- Initialize Tensorflow only when required (Application start, Configuration change)

  To cover for the changes of the configuration I am checking if there are any changes in the configuration file and am recreating the Tensorflow Model on the Fly if not the old one will be used.

- Support Image, Video and Webcam detection

  Once Tensorflow is instantiated the events of the application have to handled accordingly, with functionality implemented for each case. Since I went with the more explorer like feeling and implemented a TreeView as the source for the Files to select from a explorer was Implemented where you can select a folder as the root. Events and callback methods had to be written and passed over various classes to handle the selection of items, so an click is being interpreted as a object detection task.
  Images are being handled and rendered through PyQt however Video and Webcam streaming had to be implemented manually since PyQt had no option to access the data of the streams.
  To cover for this lack of functionality I used the package python-opencv and incorporated it into the work flow of PyQt. So Videos and Camera streams are analyzed frame by frame by opencv, converted then passed Tensorflow and finally converted back and displayed by PyQt.

Conclusion

As object detection has evolved and probably will in the future, we can see at its current state that it is able to perform complex object detection tasks in a pretty decent way close to real time. There are still problems such as the complexity and variety of image in visual data from real life scenario's but these types of problem may become less obvious as the amount of data stored in the net and servers world wide is increasing and new technology are being developed. So maybe quantum computers will bring the next breakthrough when modeling new object detection frameworks and systems. There are still a lot of ways to improve and expand on the concept so how fast and how good it will evolve is uncertain.

Bibliography

[1] Michael Nielson. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com, 2018.

[2] Ross Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster r-cnn: Towards real-time object-detection with region proposal networks. https://arxiv.org/pdf/1506.01497.pdf, 2015.

[3] T. Gevers J.R.R. Uijlings, K.E.A. van de Sande and A.W.M. Smeulders. Selective search for object recognition. http://www.huppelen.nl/publications/selectiveSearchDraft.pdf, 2012.

[4] Image with segmentation. https://www.learnopencv.com/wp-content/uploads/2017/09/breakfast_fnh.jpg.

[5] Image with bounding boxes. https://www.learnopencv.com/wp-content/uploads/2017/09/breakfast-top-200-proposals.jpg.

[6] Tensorflow. https://www.tensorflow.org/.

[7] Tsung-Yi Lin Michael Maire Serge Belongie Lubomir Bourdev Ross GirshickJames Hays Pietro Perona Deva Ramanan C. Lawrence Zitnick Piotr Doll'ar. Microsoft coco: Common objects in context. https://arxiv.org/pdf/1405.0312.pdf.

Appendix

Tensorflow .config

```
# Faster R-CNN with Inception v2, configuration for MSCOCO Dataset.
# Users should configure the fine_tune_checkpoint field in the train
    ↪ config as
# well as the label_map_path and input_path fields in the
    ↪ train_input_reader and
# eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the
    ↪ fields that
# should be configured.


model {
  faster_rcnn {
    num_classes: 8
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
```

```
  op: CONV
  regularizer {
    l2_regularizer {
      weight: 0.0
    }
  }
  initializer {
    truncated_normal_initializer {
      stddev: 0.01
    }
  }
}
first_stage_nms_score_threshold: 0.0
first_stage_nms_iou_threshold: 0.7
first_stage_max_proposals: 300
first_stage_localization_loss_weight: 2.0
first_stage_objectness_loss_weight: 1.0
initial_crop_size: 14
maxpool_kernel_size: 2
maxpool_stride: 2
second_stage_box_predictor {
  mask_rcnn_box_predictor {
    use_dropout: false
    dropout_keep_probability: 1.0
    fc_hyperparams {
      op: FC
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        variance_scaling_initializer {
          factor: 1.0
          uniform: true
          mode: FAN_AVG
        }
      }
    }
  }
}
second_stage_post_processing {
  batch_non_max_suppression {
    score_threshold: 0.0
```

```
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "/code/image_retrieval/tensorflow/models/
     ↪ research/object_detection/trained_models/
     ↪ faster_rcnn_inception_v2/model.ckpt"
  from_detection_checkpoint: true
  num_steps: 10000
}
```

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "/code/image_retrieval/tensorflow/models/research/
       ↪ object_detection/data/train.record"
  }
  label_map_path: "/code/image_retrieval/tensorflow/models/research/
     ↪ object_detection/data/cust_label_map.pbtxt"
}

eval_config: {
  num_examples: 640
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/code/image_retrieval/tensorflow/models/research/
       ↪ object_detection/data/test.record"
  }
  label_map_path: "/code/image_retrieval/tensorflow/models/research/
     ↪ object_detection/data/cust_label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```