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 technology is evolving fast and new fields for which applications are being developed are discovered frequently, Machine Learning is one of those fields which is evolving and redefining Image Processing and Object / Image Detection as we knew it.

In this report I will write about visual information retrieval (computer vision) by 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 final goal is to understand how the training process of these models work and have an 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 with objects highlighted through bounding boxes.

Introduction

Humans are able to recognize patterns, shapes and objects through their complex visual cortex system where they have millions of neurons and billions of connections between them. While the Human brain can do such a task effortlessly it is impossible for a computer to process such data. Therefore the difficulty of visual pattern detection becomes apparent when trying to implement the logic in an algorithm. [1]

Object detection, which is a technology related to computer vision or image processing, is a field that deals with the problem of processing images and audio visual content.

Machine Learning is a subset of Artificial Intelligence where the computer will perform tasks through algorithms and models. These work without explicit instructions, relying on patterns and models instead.

Object Detection

3.1 Introduction to Neurons and Networks

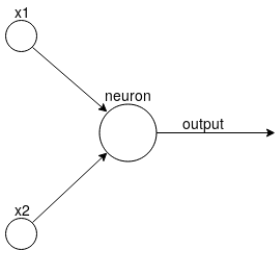
Similar to the neurons in the human brain, neurons in computer science are approaching the same concept and ideology.
The way a neuron functions is to retrieve data, possibly and ideally multiple at the same time, process and output a single result. There are many types of subsets for neurons, for the purpose of this report I will explain the two most common ones:

- Perceptrons
- Sigmoids

3.1.1 Percpetrons

Perceptrons were developed in 1957 by the scientist Frank Rosenblatt, inspired by earlier works from Waren McCulloch and Walter Pitts. [1]

Figure 3.1: Simple neuron



In this example our neuron 3.1 has 2 input variables and one output, in theory it can have more or less. Furthermore Frank Rosenblatt proposes the concept of weights for each input variable saying which variable is preferred over another one.

Figure 3.2: Activation function of a perceptron

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

A single neuron is by no means comparable to a decision making of a human being, but it can weigh variables and make decisions by defining a threshold when a true or false should be evaluated. Another way to create a conditional behavior is to use a bias instead of a threshold.

Figure 3.3: Neuron with bias

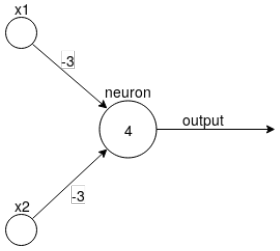
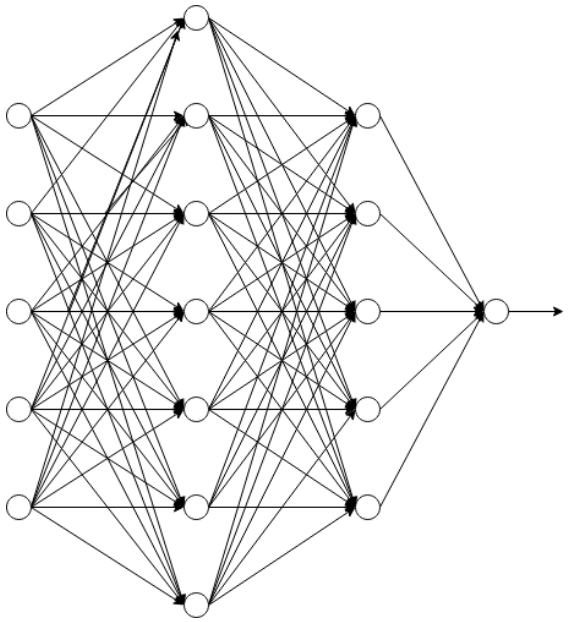


Figure 3.4: Activation function with bias

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

A whole network of these neurons is able to perform more complex decision making. The network 3.5 is a feed forward network, but there are bidirectional networks and even forms where the neuron passes its output back to itself.

Figure 3.5: Example neural network



Each column in the network 3.5 is called layer. The first column is referred to as the input layer and the last one as the output. 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 which occurs 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 why sigmoids were invented and they build on top of the concept of perceptrons.

3.1.2 Sigmoids

The sigmoid is a activation function which expands the functionality of a neuron. Its core difference is that the output can not only be 0 or 1 but anything in between those. Therefore 0.5234 could be a valid output of the sigmoid function. Likewise it can have weights and biases functioning the same as it would for the perceptrons. The core difference is the arithmetic function, called the sigmoid function or sometimes the logistic function.

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

Figure 3.6: Function for sigma

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

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

Figure 3.7: Explicit function for sigma

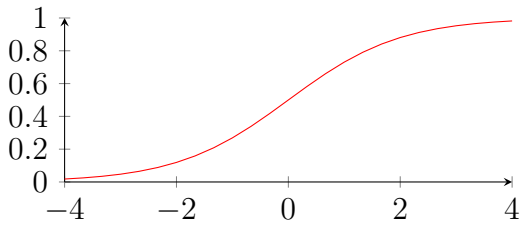


Figure 3.8: Graph for sigma

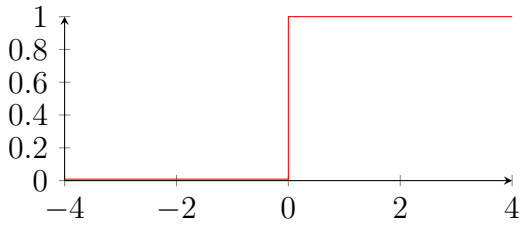


Figure 3.9: Graph for perceptrons

Since the algebraic form of σ is a smoothed and not step function, it makes a huge difference in the way how weight and bias influence the activation of a neuron.

3.2 Current state

Object detection is dealing with two major problems and therefore can be split into two processes which are object localization and object classification.
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. Models that combine functionality in one layer, where localization and classification is performed in one step, make the model smaller and therefore faster but less accurate. The balance between accuracy and speed is something that all networks deal with. Especially if the goal is to have a near real time object detection system which is still reliable

and accurate.
There are multiple algorithms covering various forms of this formula, called RCNN, F-RCNN, SSD, YOLO but first I will cover the approach of detecting interesting regions in an image because this is the key problem in object detection.

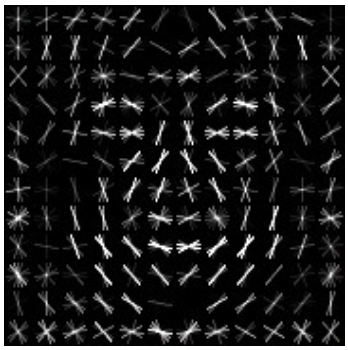
3.2.1 Object Localization & Classification

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, meaning that each object in an image has a specific place in depth, the segmentation has to be hierarchical as well. 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 for analyzing and classifying regions, I am going to cover three localization algorithms which are key for object detection.

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, have a fixed sizes and aspect ratios for the window. Further it uses weak classifiers and economic image features such as histogram of oriented gradients (HOG) to analyze each step. [2]
HOG approaches the problem of recognizing and detecting objects in an unknown image through corners. It analyzes the distribution and orientation of the corners, through which it is able to separate the image into 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. [3]

Figure 3.10: Output of a HOG algorithm

 [3]

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 an initial sub-segmentation has to be performed. [2] After the segmentation a grouping of regions is done

Figure 3.11: Segmentation of Selective Search

 [4]

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. [2]

Figure 3.12: Results of converting the segments into bounding boxes

 [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 each of these steps is a computation act, the main goal of this algorithm is to be fast and not create a bottleneck through its complexity. [2]

S_{color}:

It measures color similarities 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: [2]

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

[2]

This histogram can be efficiently propagated through the hierarchy by:

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

[2]

$S_{texture}$:

Selective Search measures texture similarity through the usage of the SIFT algorithm. SIFT stands for scale invariant feature transform and is an algorithm to detect and describe features in a region. Furthermore it takes Gaussian derivatives in eight orientations for each color channel, where a 10 bin histogram is being extracted. Similarity is measured using the same histogram intersections and the propagation function is the same as for the color. [2]

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

[2]

S_{size}:

In this step selective search encourages to merge small regions early. Doing so 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. [2]

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

[2]

S_{fill}:

It 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. [2]

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

[2]

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

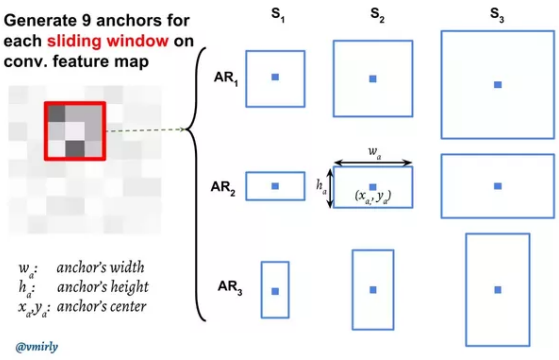
$$s(r_i, r_j) = a1s_{coulor}(r_i, r_j) + a2s_{texture}(r_i, r_j) + a3s_{size}(r_i, r_j) + a4s_{fill}(r_i, r_j)$$

[2]

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. [6]

Figure 3.13: Properties of a sliding window in RPN



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 human telling the network where, which objects are in the image. [6]

The general idea of IoU is:

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

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). The classification value is a probability indicating whether the bounding box contains a object or not. [6]

Single Shot Detector(SSD)

Compared to the Faster R-CNN where the detection happens in two stages. First having the region proposal network generating proposals and second 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." [7]

4.1 Writing Object Detection models with Tensorflow

Tensorflow is a full fledged machine learning library which supports the programming languages C++, Python. It is a multi purpose library which can be used for training neural networks for Image Recognition, object detection or other purposes. Furthermore Keras uses Tensorflow which is a high level API for building and training deep learning models. In object detection which I am using Tensorflow in this report for, it is possible to write a object detection system in multiple ways.

- Do it yourself

  It is possible to write a object detection system from scratch, starting at how features are being extracted, processed and evaluated. How layers of the 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 Networks. It is possible to compose new ideas for models with simple '.config' files without going too deep into the code and logic of Tensorflow. These configuration files contain the setup of the neural network model. Such as in which sequence the layers are going to be, what the IoU value should be for a bounding box to be valid and which features have to be extracted by

the Tensorflow API, this obviously is just a hand full of the features Tensorflow provides with this Library. It is possible to see a full configuration file used for training in the appendix.

4.2 Training models with Tensorflow

As there are multiple ways of setting up an object detection system there are a couple of ways of training the model.

- Train from scratch

  Training a model either through a configuration file or through a whole new system, starting from scratch without a basis will require:
  - large quantities of resources (images, ground truth data)
  - considerably more training time

- Train by TRANSFER learning

  Transfer learning, is taking an already trained model and builds on top of it, using a custom image set.
  This results in:
  - a reduced need for images
  - reduction of training time from several days to few hours or minutes

Training a model with the setup from Tensorflow as mentioned in 4.1 "Just write the model" is pretty straight forward. For this Tensorflow and all the required dependencies, which can be found at the page: `https://www.tensorflow.org/install/pip` have to be installed. Further more some custom scripts were required to enable converting text files to ".csv", create ".record" files, merge label files from MSCOCO and test the object detection models without an application. Lastly environmental variables, proto files and '.record' files had to be setup and compiled for Tensorflow.

4.3 Training results

After setting up the environment, my first attempt was collecting the data myself, labeling them with the Open Source software LabelImg and training a new model through a Tensorflow configuration. The results of this model were bad when using unknown images. To test results this procedure was performed in various combinations such as different data quantities, IoU, variable and extraction feature tweaks provided through Tensorflows model, without much difference in its performance.

Afterwards the transfer learning method was used. This method takes the weights and knowledge of an already existing model and applies it to a different one which should deal with similar problems. Images and bounding box data was downloaded through the API of MSCOCO which is a dataset providing more then 2.5 million label instances in 328 thousand images. [8] There are other datasets such as Pascal VOC and ImageNet but none of them provided an easy way to download data similar to MSCOCO. For the following models I downloaded 9 classes with one thousand images and labels each to test, vary and experiment with the amount of data provided to the model.

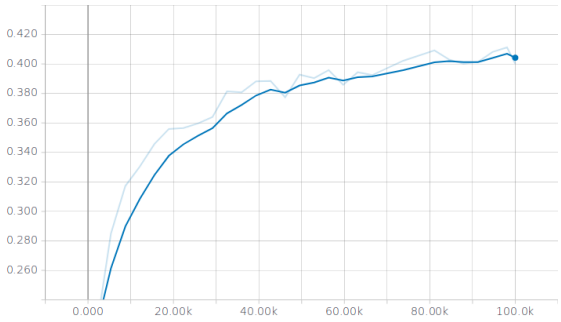
Secondly I downloaded the models from Tensorflow, one was the faster-rcnn-inception-v2-coco and the ssd-mobilnet-v2-coco trained with the dataset from MSCOCO. These are 2 models where the first one is implemented with the Faster-RCNN and the latter with SSD algorithm.

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

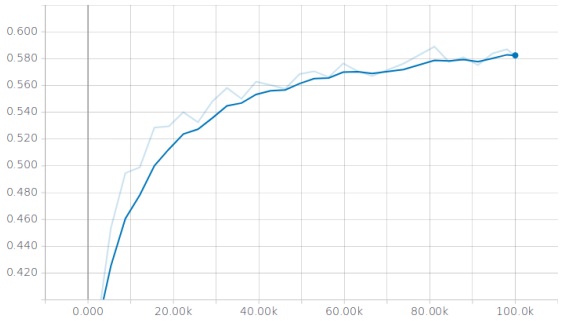
Figure 4.1: Performance of the CNN Model

For transfer learning it is noticeable that 200 images for each category was more then enough to get decent results. The results degraded compared to the original Tensorflow model which I used for transfer learning. But they were still better then the initial attempts of training. The following model through transfer learning a degradation of. But they were still better then the initial attempts of training. The following graphs are smoothed by a factor of 0.6 and showcase the performance of one of my models.

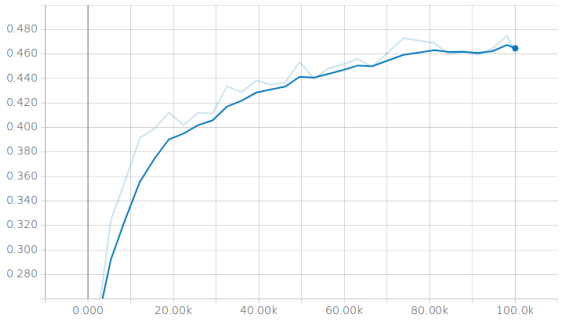
Figure 4.2: mAP of a model



(a) mAP per Stage



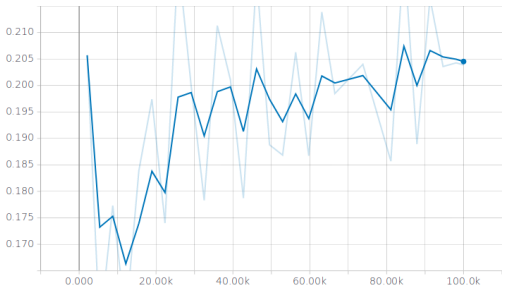
(b) mAP @0.5 IoU



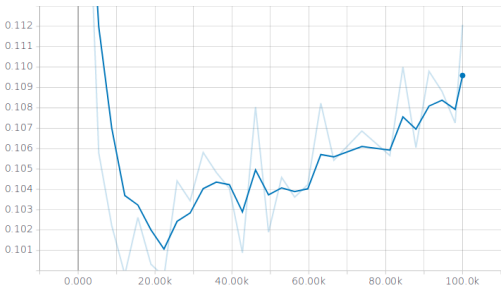
(c) mAP @0.75 IoU

Even though the mAP is increasing over the duration of the training session it is noticeable that it is not really increasing in performance but overfiting through the following graphs.

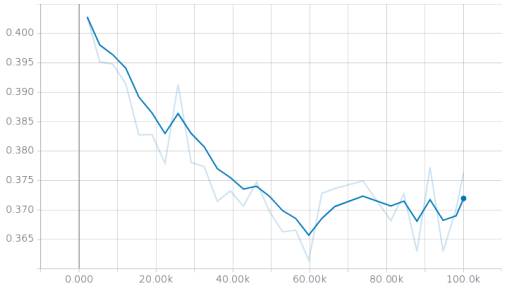
Figure 4.3: Loss of a model



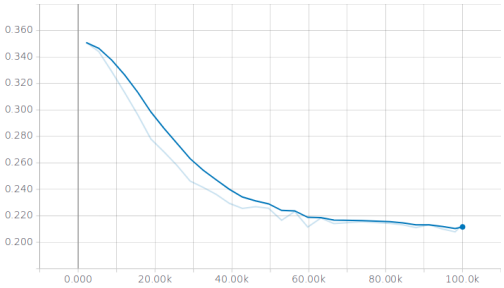
(a) Box Classifier Loss



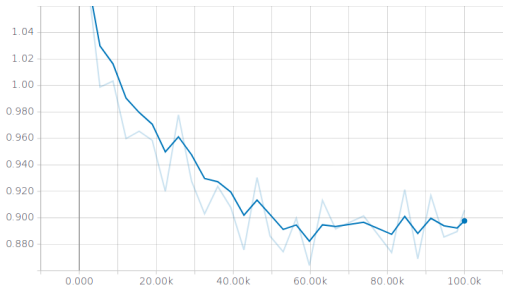
(b) Box Classifier Localization Loss



(c) RPN Localization Loss



(d) RPN Objectness Loss



(e) Total Loss

Application

The initial phase of creating the application was to have a basic script which instantiates a Tensorflow model, processed the image to Tensorflow standards and passed it on to the model for object detection. Using this script as a basis which performed the object detection on images,videos and web camera streams I started developing an application in Python with PyQt Version 5.
PyQt is a cross-platform GUI toolkit which is similar to Java Swift and is an easy library to develop an UI with. It takes ideas and concepts from different roots and implements a layer based architecture for elements.

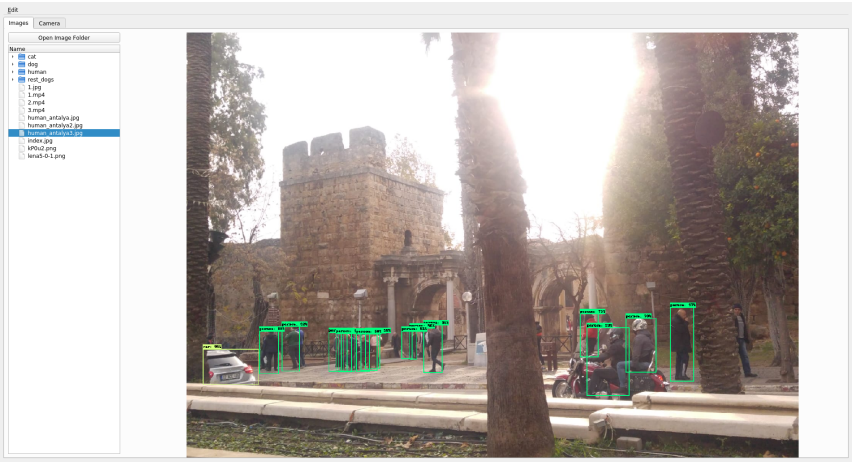


Figure 5.1: Application after object detection

There were several requirements for the application from the start:

- Use any model trained with Tensorflow

  Since the Tensorflow API is being used for the application, the problem of using different Tensorflow models is being covered to some extent 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.

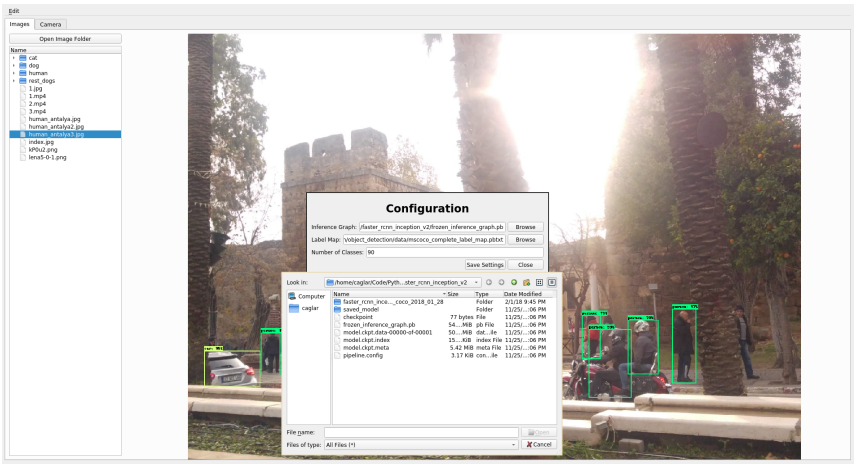


Figure 5.2: Application config section

- 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 I am recreating the Tensorflow model on the fly and 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 with custom functions implemented for each case. Since I went with the more explorer like look and feel, I implemented a TreeView as the source for the files to select from. Events and callback methods had to be written and passed over various classes to handle the selection of items, so a click is being interpreted as an 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 and then passed on to Tensorflow. The final step is to convert it back and display it by PyQt.

Conclusion

During this project I trained multiple Tensorflow models and wrote an application which utilizes a trained model through Tensorflow and is able to detect objects in an image, video or web camera stream fairly well. Comparing the object detection results of Faster RCNN and SSD, it is noticeable that the F-RCNN algorithm is performing better in regards to accuracy but lacks in speed. The reason for that as mentioned during this report, is the complex structure of the neural network where each task has its own layer resulting in a slower model.
The SSD algorithm on the other hand is performing better in regards of speed since the whole detection and classification is performed in one step but therefore losses in accuracy occur.

The results and performances of the trained models vary and further research for improving these need to be done. Possible feature extensions for the application could be having multiple models set up and active which could be used for comparing and running analyzes of the performances of different models.

To sum it up, object detection has evolved and probably will evolve further 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 which is close to real time. There are still problems such as the complexity and variety of images in the real life scenarios. But these types of problems may become less relevant as the amount of data stored in the Internet and on servers world wide, will increase and new technologies will be developed. Maybe quantum computers will bring the next breakthrough when modeling new object detection systems and technologies. There are still a lot of ways to improve and expand on the concept. How fast and how good it will evolve is uncertain.

Bibliography

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

[2] 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.

[3] Histogram of oriented gradients. https://de.wikipedia.org/wiki/Histogram_of_oriented_gradients.

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

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

[8] Tsung-Yi Lin Michael Maire Serge Belongie Lubomir Bourdev Ross Girshick-James 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
}
```