CROP AND WEED DETECTION USING TENSORFLOW OBJECT DETECTION API

A thesis submitted in fulfillment of the requirements for the degree of MSc in Data Science in the Department of Informatics

In collaboration with

Centaur Analytics

Author: Konstantinos Alexis

Academic supervisor: Assoc.Prof. Yannis Kotidis

Company supervisor: Pavlos Kallis

December 25, 2018
Abstract

Department of Informatics
Msc in Data Science

Crop and weed detection using Tensorflow Object Detection API
by Konstantinos Alexis

The purpose of this thesis is to study a specific object detection task, namely to develop an end-to-end pipeline, able to detect crop and weed instances in a field imagery dataset. Tensorflow Object Detection API was utilized, which has democratized the object detection field by making research code developed at Google available. Object detection key concepts along with two state-of-art approaches, commonly referred in the relevant bibliography, the Single Shot Detector and the Faster R-CNN, are presented. Afterwards, we demonstrate the steps to create an image dataset ready to be used in the API. We then present the experimental results, providing some remarks about the whole task. We find out that the SSD models seem to be faster, while Faster R-CNN tend to achieve a higher performance, making the choice of the best approach to be a matter of each specific application needs.

Academic supervisor: Assoc.Prof. Yannis Kotidis
Company supervisor: Pavlos Kallis
This thesis was conducted in collaboration with Centaur Analytics.
# Contents

Abstract iii  

1 Introduction 1  
1.1 Foreword 1  
1.2 Problem Statement 1  
1.3 Object Detection 1  
1.4 Deep Learning 1  
1.5 Tensorflow Object Detection API 2  

2 Theoretical Background 5  
2.1 Convolutional Neural Network Architecture 5  
2.1.1 The convolutional layer 5  
2.1.2 The pooling layer 6  
2.1.3 The output layer 7  
2.2 Feature Extractor 7  
2.3 Evaluation Metrics 8  
2.3.1 Intersection Over Union (IoU) 8  
2.3.2 True Positives, False Positives, False Negatives and True Negatives 8  
2.3.3 Precision 9  
2.3.4 Recall 9  
2.3.5 Precision - Recall Relationship 9  
2.3.6 Average Precision (AP) 9  

3 Related Work 13  
3.1 Object Detection Model Map 13  
3.2 Object Detection Framework 13  
3.3 Single Shot Multibox Detector (SSD) 14  
3.3.1 Key Concepts 14  
3.3.2 Model 14  
3.3.3 Default Bounding Boxes Framework 15  
3.3.4 Additional Notes 16  
3.4 Faster R-CNN 16  
3.4.1 Key Concepts 16  
3.4.2 Model 16  
3.4.3 Region Proposal Network (RPN) 17  
3.4.4 Regions of Interest (RoI) Pooling 17  
3.4.5 Additional Notes 18  

4 The Dataset 19  
4.1 Data Source 19  
4.2 Bounding Box Annotation 19  
4.3 Train and Test Split 20
4.4 TFRecord conversion ........................................ 20
4.5 Data Augmentation ........................................ 21

5 Experiments .................................................. 23
  5.1 Setup .................................................... 23
  5.2 Working with the API .................................... 23
  5.3 Selected Models .......................................... 24
  5.4 Experiments Results ...................................... 24
    5.4.1 Training with pretrained weights ................... 24
    5.4.2 Training time ........................................ 26
    5.4.3 Detection of large, medium and small sized objects 27
    5.4.4 Input size impact .................................... 28
    5.4.5 Performance and Inference speed .................... 28
    5.4.6 Detections ........................................... 29

6 Conclusions and Future Work ............................. 31
  6.1 Conclusions ............................................... 31
  6.2 Future Work ............................................. 32

Bibliography .................................................. 33
# List of Figures

1.1 Deep Learning era ................................................. 2  
1.2 Machine Learning vs Deep Learning frameworks ............... 2  
1.3 Tensorflow Object Detection API logo .......................... 3  
2.1 Convolutional layer example .................................... 5  
2.2 Trained filters example .......................................... 6  
2.3 Pooling layer example .......................................... 7  
2.4 IoU between predicted and ground truth bounding boxes .... 8  
2.5 Precision x Recall plot ......................................... 11  
2.6 Precision x Recall (interpolated) plot .......................... 12  
3.1 Object detection model map ...................................... 13  
3.2 SSD model architecture .......................................... 15  
3.3 SSD default bounding boxes framework ........................ 15  
3.4 Faster R-CNN model architecture .............................. 17  
3.5 The Region Proposal Network workflow ....................... 18  
4.1 Sample of images used ........................................... 19  
4.2 Annotating an image using the LabelImg GUI ................. 20  
5.1 mAP@.5, Train loss and Validation loss for the pretrained models .... 26  
5.2 Train loss against absolute training time (in hours) plot .... 27  
5.3 Performance of models on detecting large, medium and small sized objects ............................................. 28  
5.4 mAP@.5 for faster_rcnn_inception_v2 with respect to input size ..................................................................... 29  
5.5 Predictions by faster_rcnn_resnet101 ........................... 30  
5.6 Predictions by ssd_inception_v2 .................................. 30
List of Tables

2.1  Some well known image classification models. .............................. 7
2.2  Example detections (1/2). ............................................................. 10
2.3  Example detections (2/2). ............................................................. 11

5.1  Some well known object detection datasets. Records refer to the train sets of the datasets. ................................................................. 25
5.2  Performance and Speed of models. .................................................. 29
Chapter 1

Introduction

1.1 Foreword

Over the last years, Internet of Things and artificial intelligence applications have transformed the way we live and perceive the world around us. Our comforts have enhanced through automation. Various industry sectors have started leveraging the benefits of such technologies improving their production in both quantity and quality terms. In the case of this thesis, the task to be studied originates from the agriculture sector, which is a sector making its first steps into being “smart”.

1.2 Problem Statement

The purpose of this thesis is to create a system and experiment with various neural network architectures that are able to detect instances of crop (sugar beets) and weed in field images. The system will be trained with images that have been previously tagged with the coordinates of each crop and weed appearing in the image. It will be able to locate and draw bounding boxes around them with the exact coordinates of each instance. A full end-to-end pipeline which includes image preprocessing/augmentation, training and evaluation of accuracy of each neural network architecture will be implemented.

1.3 Object Detection

Object detection is a general term for computer techniques closely related to computer vision and image recognition. These techniques come to deal with a twofold problem, identifying the location of an object in an image (localization) and labelling this object into a certain category (classification). In general, the aim is to correctly locate and label every object that appears in a given image.

Object detection applications appear in a variety of domains, including security, statistics, robotics and others. Face detection [1], crowd statistics [2], self driving cars [3], object tracking and video surveillance [4] can be indicatively mentioned as popular among them.

1.4 Deep Learning

With recent research and advancements many difficulties have been tackled leading to efficient and accurate techniques. The increasing volume of available data has helped Deep Learning based computer vision models make the next step, starring in the winning submissions of most image analysis challenges. The beginning
was made by *AlexNet* [5] in 2012, when a deep convolutional neural network managed to drop the ImageNet Challenge classification error record from 26% to 15% (Figure 1.1).

![Deep Learning era](image1)

**Figure 1.1:** Deep Learning era [6].

Deep learning models are more effective and faster than previous approaches. They also minimize the prior knowledge and human effort involved in feature design, as they can learn features themselves by looking into data, while in a traditional image analysis algorithm this would be hand-engineered (Figure 1.2). Besides significant performance improvements, these techniques have also been leveraging massive image datasets to reduce the need for huge case-specific datasets.

![Machine Learning vs Deep Learning frameworks](image2)

**Figure 1.2:** Machine Learning vs Deep Learning frameworks.

### 1.5 Tensorflow Object Detection API

The Tensorflow Object Detection API [7], [8] consists of the object detection research code developed at Google. It is an open source framework built on top of Tensorflow capable of developing, training and deploying object detection models. By making this code public, they democratized object detection making its applications available to everyone and a great place for further research. This API was chosen over
other available alternatives as it provides a common ground in order to fairly compare and experiment with a number of different models.

The API comes with a bundle of ready to use but also highly customizable object detection models. These models can be modified in terms of both their architecture as well as their training and evaluation configuration and tools to easily perform such operations are provided. Moreover, the API takes advantage of all the default Tensorflow utilities like GPU-support, Tensorboard etc. These specifications make the Tensorflow Object Detection API a great framework in order to develop, train, evaluate and finally compare the pros and cons of a list of state-of-art object detection models.

![Tensorflow Object Detection API logo](image-url)

**Figure 1.3:** Tensorflow Object Detection API logo.
Chapter 2

Theoretical Background

2.1 Convolutional Neural Network Architecture

Convolutional Neural Networks are a category of deep neural networks, proven to be very efficient in computer vision tasks such as image classification [9] and object recognition. They are especially good at identifying visual patterns in an image. Starting from their first layers, basic lines, color and texture are recognized composing more complex patterns as moving through deeper layers. A typical convolutional neural network consists of the following basic components:

1. The convolutional layer
2. The pooling layer
3. The output layer

2.1.1 The convolutional layer

The convolutional layer is the main building block of a convolutional neural network. This layer is about passing the input image through a set of learnable filters. A filter starts from the top left of the input and in a sliding window manner, it “convolves” through the whole input. In each position, element wise multiplications are computed between the filter weights and the corresponding input values. The products of these multiplications are the summed, resulting in a single number for each possible filter position. This process is illustrated in Figure 2.1.

\[
\begin{array}{ccc}
1_{x1} & 1_{x0} & 1_{x1} \\
0_{x0} & 1_{x1} & 1_{x0} \\
0_{x1} & 0_{x0} & 1_{x1} \\
0 & 0 & 1 \\
0 & 1 & 1 \\
\end{array}
\rightarrow
\begin{array}{c}
0 \\
1 \\
1 \\
0 \\
0 \\
\end{array}
\begin{array}{c}
\text{FIGURE 2.1: Convolutional layer example.}
\end{array}
\]

\[
\begin{array}{ccc}
4 & 3 & 4 \\
2 & 4 & 3 \\
2 & 3 & 4 \\
\end{array}
\]

The result of this process, shown as the red array above, is referred to as feature map. In a real case scenario, each convolutional layer involves applying a number
of such filters. Each filter outputs its own feature map and by stacking these features maps, we get the layer’s output. As a note, both input and filter are actually 3-dimensional matrices (speaking of RGB images) but there are shown here as 2-dimensional for simplicity.

The distance the filter moves each time it convolves is called the *stride*. As shown in Figure 2.1, passing an image through a convolutional layer, its spatial dimensions (height and width) decrease and for this reason the input array may sometimes be padded by zeros around the border in order to maintain its dimensions after the convolution. This is called the *padding*.

The weights of each filter are actually learned during the training phase of the network. Visualizations of such real case filters, taken from the first convolutional layer of a trained network are shown in Figure 2.2.

![Figure 2.2: Trained filters example [9].](image)

### 2.1.2 The pooling layer

Pooling layers are commonly used between successive convolutional layers in the convolutional network architecture. The main purpose of this layer is to reduce the dimension of its input. By doing so, the number of parameters are also reduced and thus overfitting is prevented.

Like convolutional layers, pooling layers operate as a sliding window, applying a fixed function on the corresponding sub-array of the input. The most common functions are MAX and AVG. Since pooling layer performs a fixed function, it does not add any parameters to the whole architecture. An example of a MAX pooling layer using a window of size 2 and stride 2 is illustrated in Figure 2.3.
2.1.3 The output layer

After applying a series of convolutional and pooling layers combination, the architecture has hopefully detect some high level visual features. This is where the output layer shines. It is actually a fully connected layer as those appearing in traditional neural networks. This layer takes as input the output of the previous layer and transforms it into a N-sized vector, where N is the number of the classes that the model should recognize. The fully connected layer processes the high level features in its input, as a classifier, trying to find out to which class they are most correlated. Then, each value $i$ of the N-sized output would refer to the probability that the initial input image actually shows an instance of class $i$.

2.2 Feature Extractor

Object detection models start by utilizing such convolutional neural networks, namely image classification models, cut just before their fully connected layers, in order to extract visual features. In this context, this first part is referred to as the feature extractor of the whole object detector. Typically, the goal of the feature extractor is to transform the input image into a set of fixed sized features.

Table 2.1 shows some of the well known image classification models that can be utilized as feature extractors in a object detection system.

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 accuracy</th>
<th>Num. Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>71.0</td>
<td>14 714 688</td>
</tr>
<tr>
<td>MobileNet</td>
<td>71.1</td>
<td>3 191 072</td>
</tr>
<tr>
<td>Inception_v2</td>
<td>73.9</td>
<td>10 173 112</td>
</tr>
<tr>
<td>ResNet101</td>
<td>76.4</td>
<td>42 605 504</td>
</tr>
<tr>
<td>Inception_v3</td>
<td>78.0</td>
<td>21 802 784</td>
</tr>
<tr>
<td>Inception Resnet_v2</td>
<td>80.4</td>
<td>54 336 736</td>
</tr>
</tbody>
</table>

ResNet and Inception are designed for high performance when inference speed is not the main goal. On the contrary, the lightweight Mobilenet is preferred for real time applications. The choice of feature extractor is of a great importance as its
architecture and the number of parameters affect both performance and inference
time of the resulting object detector.

2.3 Evaluation Metrics

The most frequently used metric to measure the performance of an object detection
model is mean Average Precision (mAP). Along with its variations, it sets the task of
object detection evaluation on a common ground, while being the official evaluation
metric for the best-known competitions.

Before presenting mAP, some basic concepts and how they adapt to object detec-
tion context are described first.

2.3.1 Intersection Over Union (IoU)

Intersection over Union (IoU) is a measure based on Jaccard Index that quantifies
how much two bounding boxes overlap. Given a predicted bounding box $BB_{pred}$
and a ground truth bounding box $BB_{gt}$, IoU is the ratio between the intersection and
the union of $BB_{pred}$ and $BB_{gt}$.

$$IoU = \frac{\text{area of overlap} (BB_{pred} \cap BB_{gt})}{\text{area of union} (BB_{pred} \cup BB_{gt})}$$

This measure helps evaluate whether $BB_{pred}$ is a successful detection or not,
based on a chosen threshold. This threshold is usually set to 50% but can also be
75% or 95%.

Figure 2.4 visualizes the IoU between a predicted bounding box and a ground
truth bounding box.

2.3.2 True Positives, False Positives, False Negatives and True Negatives

- **True Positives (TP):** Valid detections with IoU $\geq$ threshold.
- **False Positives (FP):** Invalid detections with IoU $<$ threshold.
2.3. Evaluation Metrics

- **False Negatives (FN):** Ground truths that are not detected.

- **True Negatives (TN):** These would refer to the correct misdetections in the image. Every part of the image where there is no predicted bounding box is considered a negative. In that way, there are too many such correctly not predicted bounding boxes and for this reason measuring that quantity makes no sense. Thus, true negatives are not used in the object detection concept.

2.3.3 Precision

Precision indicates the ability of the model to correctly detect bounding boxes. It quantifies the confidence that a predicted bounding box is indeed a valid detection. Precision is defined as the ratio of valid detections to the number of total detections.

\[ \text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{total detections}} \]

2.3.4 Recall

Recall measures the ability of the model to detect every single ground truth. A high recall score suggests that almost all ground truths will be successfully detected by the model. Recall is defined as the ratio of valid detections to the number of total ground truths.

\[ \text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{total ground truths}} \]

2.3.5 Precision - Recall Relationship

Precision and recall have an inverse relationship between them. They also depend on what threshold is used. More specifically, lowering threshold, will increase the recall (more ground truths will be detected) but also decrease the precision (there will be more invalid detections) and vice versa. Of course, these metrics depend also on the quality of each model under discussion.

2.3.6 Average Precision (AP)

Average Precision (AP) was introduced in the PASCAL VOC challenge [10] in order to unify the evaluation of both classification and detection tasks. Answering to the need for a single number metric, AP makes it easy, not only to evaluate the performance of a single object detection model, but also to compare different methods and approaches.

As described in the PASCAL VOC 2012 challenge [11], AP is computed as follows:

1. Compute a version of the measured precision/recall curve with precision monotonically decreasing, by setting the precision for recall \( r \) to the maximum precision obtained for any \( r' \geq r \).

2. Compute the AP as the area under the curve by numerical integration. No approximation is involved since the curve is piecewise constant.

As a note, prior to 2010 the AP was computed by sampling the monotonically decreasing curve at a fixed set of uniformly-spaced recall values \( 0, 0.1, 0.2, \ldots, 1 \) (method
known as 11-point interpolation). By contrast, VOC2010-2012 effectively samples the curve at all unique recall values. This have been done in order to improve precision and ability to measure differences between methods with low AP.

To make AP computation more clear, the following example is supposed:

- A dataset of images is given.
- There are 8 objects (of the same class) to be detected in the images.
- A model has made 15 detections.
- Every detection comes with a confidence score, showing how confident the model was that there is indeed an object in the predicted bounding box.

Using the IoU criterion described in Section 3.1.1 along with a threshold, each of the 15 detections is labelled as valid (true positive) or invalid (false positive). The detections are then sorted based on their confidence score and Table 2.2 is formed.

<table>
<thead>
<tr>
<th>Rank</th>
<th>TP/FP</th>
<th>total TP</th>
<th>total FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TP</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>FP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>FP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>FP</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>FP</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>TP</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>TP</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>FP</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>FP</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>TP</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>FP</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>TP</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>FP</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>FP</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>FP</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

where total TP and total FP are accumulative columns, counting the number of TP and FP accordingly.

Utilizing the formulas presented in Sections 2.3.3 and 2.3.4 along with the accumulative TP and FP columns, the columns Precision and Recall are computed as shown in Table 2.3
2.3. Evaluation Metrics

Plotting Precision against Recall is shown in Figure 2.5.

![Precision x Recall plot](image)

**Figure 2.5:** Precision x Recall plot.

It can be noticed that Recall increases, while Precision goes up and down, creating this zigzag pattern. Interpolating the precision, as described in the AP computation above, results in Figure 2.6. Interpolation is done here to reduce the impact of small variations in the ranking of detections.
AP is the area under the interpolated curve and can be computed as:

\[
AP = (0.125 - 0) \cdot 1 + (0.375 - 0.125) \cdot 0.428 + (0.625 - 0.375) \cdot 0.416
\]

\[
AP = 0.125 + 0.107 + 0.104
\]

\[
AP = 0.336
\]

So, the model evaluated in the example achieves \( AP = 33.6\% \).

Some further notes on AP:

- \( mAP \) is AP averaged over all object classes.
- In order to achieve a high AP score, a model should have precision at all recall levels. Thus, models which perform well only in a specific subset of a dataset are penalised.
- There are variations in the computation of \( mAP \). The most frequent among them involves the choice of IoU threshold. In PASCAL VOC challenge an IoU threshold value of 0.5 (\( mAP@.5 \)) is used, while in COCO competition [12], \( mAP \) is averaged over different IoU thresholds, from 0.5 to 0.95 with step 0.05 (\( mAP@[.5:.95] \)). Averaging over multiple IoU thresholds tends to reward precise localization.
- In a multiclass scenario, the observation of each class individual AP score is highly advised. The \( mAP \) score should be used as a single number metric, roughly describing performance of the model but monitoring each class performance is the way to address the model’s strengths and weaknesses.
Chapter 3

Related Work

3.1 Object Detection Model Map

Figure 3.1 shows how some state-of-art object detectors perform in terms of mAP and inference speed. Even though this figure illustrates only detectors based on Faster R-CNN, R-FCN and SSD meta architectures, it provides a glance at the whole picture of object detection models. As the plot suggests, variations on the model design like choices about the meta architecture, the feature extractor, the input image resolution, etc. lead to different results. Due to all these possible variations, it is not easy to present a strict comparison and ranking among all the object detectors proposed in the relevant bibliography.

3.2 Object Detection Framework

The main components involved in the object detection task are usually the following:

1. Feature Extraction: Utilizing an image classification convolutional neural network, input image is turned into a set of high level visual features.
2. **Regions of Interest:** The model has to decide the image areas in which it should search for objects. More specifically, it has to generate a set of candidate bounding boxes, that will be evaluated whether they contain valid detections or not.

3. **Localization and Classification:** For each of these candidate bounding boxes, the model has to determine whether there is an object there or not, as well as classify it in one of the classes it should recognize. Moreover, in this step, the spatial coordinates of each of these bounding boxes are refined in order to enclose the object as accurately as possible.

4. **Non Maximum Suppression:** At this point, there may be many valid bounding boxes. There may also be multiple detections of a single object. Non Maximum Suppression reduces the number of such multiple detections by replacing them with the one among them which has the greatest IoU with the specific ground truth.

### 3.3 Single Shot Multibox Detector (SSD)

Single Shot Multibox Detector (SSD) paper [13] was originally published in December 2015. While competing with state of art approaches in terms of performance and accuracy, SSD claims to be much faster, taking much less time during prediction phase.

#### 3.3.1 Key Concepts

SSD is a feed forward convolutional network model. It belongs to the Single Shot model family, indicating that both localization and classification tasks are performed in a single forward pass of the model’s network.

While not being the first approach towards real time detection, SSD, unlike previous efforts, manages to maintain high detection precision. Its core idea consists of eliminating bounding box proposals and the subsequent pixel or feature resampling stage while adding a series of improvements and tweaks in order to increase accuracy. More specifically, the following ideas are implemented:

- Object classes and bounding box location offsets are predicted using a small convolutional filter.
- A set of separate filters, varying in size, is used in order to detect different aspect ratio objects.
- These filters are applied to multiple feature maps from the later layers of the network to handle varying in scale objects.

#### 3.3.2 Model

Figure 3.2 illustrates the default SSD model architecture.

The very first layers of the network are based on a standard image classification architecture (here VGG-16, but it could be another as well), truncated before its last fully-connected layers.

Extra convolutional layers are added to the end of this part. These layers actually constitute the structure that performs the detections. These feature layers,
progressively decrease in size, allowing detection of objects among multiple scales. More precisely, the first layers of this structure (those corresponding to larger feature maps) are responsible for detecting smaller objects in the image, while larger objects will hopefully be detected by the subsequent ones.

Finally, a non maximum suppression step is applied to reduce the detections made until this point and combine them into the final detections.

### 3.3.3 Default Bounding Boxes Framework

SSD defines a set of (4 or 6 by default) manually and carefully specified, default bounding boxes of varying aspect ratio. These bounding boxes are evaluated, in a convolutional way, at each cell of several feature maps of different dimensions. For each such box, both location offsets \( \text{loc} \) and confidence of each object class \( \text{conf} \) are predicted.

Figure 3.3 illustrates the method described above, in an example using a set of 4 default bounding boxes in each feature map location.

That being said, for a \( m \cdot n \) feature map, evaluating \( b \) default bounding boxes at each feature map location results in \( m \cdot n \cdot b \) outputs. For each default bounding box both \( \text{loc} \) (involving 4 offsets) and \( \text{conf} \) (involving \( p \) confidence values in a \( p \)-class scenario) are computed, resulting in \( m \cdot n \cdot b \cdot (4 + p) \) final outputs to be computed.
3.3.4 Additional Notes

- A key difference between SSD and detectors using regional proposals, is that ground truths should be matched to a specific bounding boxes fixed set.

- By applying multiple bounding boxes of varying scale and aspect ratio, in multiple feature maps of varying scale, the space of possible ground truth bounding boxes shapes are emulated.

- Utilizing more default bounding boxes improves detection results in terms of precision, but adds up computations, making the model slower.

- While performing well on large objects, SSD may have trouble in detecting smaller objects, as they may not have any information at the top extra layers. This may be overcome by increasing the input image resolution.

3.4 Faster R-CNN

Faster R-CNN [14] was firstly introduced in June 2015 by Microsoft researchers. It was released as an improvement over its Fast R-CNN predecessor, making efforts to reduce the running time of the latter.

3.4.1 Key Concepts

Faster R-CNN is a Region-Based Convolutional Neural Network. Models following this approach were restricted due to their need of an external region proposal mechanism. These region proposal methods were a computational bottleneck for these models, being responsible for most of their running time.

The idea introduced in Faster R-CNN is to implement a region proposal mechanism that can share convolutional layers with the actual object detection network. On top of the convolutional features maps from the base image classification network, a few additional convolutional layers were added, referred to as Region Proposal Network (RPN), simultaneously regressing region bounds and objectness scores at each location on a regular grid, resulting in almost cost-free proposals. In this way, the system can be trained end-to-end, without the need to train multiple independent structures.

3.4.2 Model

Faster R-CNN model flow is illustrated in Figure 3.4.

Firstly, an input image is passed through a standard, image classification convolutional network, in order to generate its feature maps. RPN module, which is a deep fully convolutional network, takes as input these feature maps and proposes regions of interest, along with a objectness score for each of them. A RoI Pooling layer is applied on these proposals to convert them to a fixed size. Proposals are then passed to a fully connected layer with a softmax and a regression layer to be classified and to output the objects bounding boxes.
3.4. Faster R-CNN

3.4.3 Region Proposal Network (RPN)

The Region Proposal Network (RPN) is the component that generates regions of interest for the system. As stated above, the goal in Faster R-CNN is to share computation through a common set of convolutional layers between the RPN and the Fast R-CNN object detection network. RPN uses a sliding window method over the output feature maps of the last shared convolutional layer. At each window location, RPN generates at most \( k \) bounding boxes of different scale and aspect ratio, aka Anchors. Each of these windows is mapped to a lower dimension feature (256-d for the paper [14] setup) and then passed through a classification (\( \text{cls} \)) and a regression layer (\( \text{reg} \)). The \( \text{cls} \) layer computes the probability of an anchor being an object or background -a binary class scenario- resulting in a \( 2^k \) output. The \( \text{reg} \) layer outputs \( 4^k \) encoding the coordinates of the bounding boxes. These \( \text{reg} \) outputs are utilized to adjust and refine the corresponding default anchor so that it better fits the object.

The RPN workflow is shown in Figure 3.5.

3.4.4 Regions of Interest (RoI) Pooling

RPN module generates regions of interest that may differ in size. On the other hand, the fully connected layers for classification and localization, following after RPN, demand a specific input size. Regions of Interest (RoI) Pooling helps in this case. Given a feature map (generated by the base convolutional layers) and a matrix representing the regions of interest location (generated by the RPN), it splits each region proposed into a predefined number of roughly equal sized parts and then applies Max Pooling on each of these parts. In this way, a list of rectangles of different sizes is turned into a list of equal sized feature maps.
3.4.5 Additional Notes

- The whole Faster R-CNN system is a single unified network, where the RPN serves as the ‘attention’ of the Fast R-CNN module. In other words, it tells the latter where to focus.

- By replacing the Selective Search method used in Fast R-CNN with the RPN, Faster R-CNN reduced its running time from 2 seconds to 0.2 seconds per image.

- The number of regions proposals generated by the RPN affects the inference speed of the whole Faster R-CNN model.

- Training may be done either in an alternating way (firstly training RPN, then tuning Fast R-CNN and finally re-initializing RPN before starting over again), or approximately jointly (treating the system as merged). The joint version is an approximate one, but reduces training time by up to 25% compared to the alternating one.
Chapter 4

The Dataset

4.1 Data Source

The dataset used in this thesis is *The 2016 Sugar Beets Dataset Recorded at Campus Klein Altendorf in Bonn, Germany* [15]. An agricultural field robot was used to record this dataset by taking images of the field underneath itself as it was moving along the sugar beet farm. The collected data include a variety of information like RGB and NIR images, timestamps, 3D point-clouds, etc. In our case we make use only of the RGB images, as it is the most reasonable input type for the object detection task. As a note, it would be interesting to repeat the experiments described in Chapter 4 using the NIR images instead of the RGB ones, as this could result in useful remarks.

The full dataset consists of about 13 000 RGB images but a sample of 300 among them was actually used taking into account the available computer setup, as the involved models are computationally expensive. The specific sample was selected with respect to the record date, preferring the season during which both crop and weed are as visible as possible.

![Sample of images used.](image)

4.2 Bounding Box Annotation

In order to compose a useful dataset, these 300 images should first be annotated. That means that for each crop or weed appearing in these images, a bounding box should be drawn enclosing it as precisely as possible and labelling it with its class (sugar beet or weed).

This was achieved using *LabellImg* [16], a graphical image annotation tool designed to enable drawing and labelling object bounding boxes in images. *LabellImg* provides a graphical user interface (GUI) making the whole process very straightforward. In Figure 4.2 the *LabellImg* GUI is used to annotate an image from the dataset.

For each bounding box drawn there has to be a label given, referring to a corresponding object class. In this project, there are two classes, sugar beet and weed. The classes that should be detected by the model are described in a `label_map.pbtxt`...
file, where each of the classes is encoded as an id number starting from 1, as the zero id is reserved for the background (meaning no object) class.

Annotations made for each of the images, were saved as XML files (a single XML file per image, containing its bounding boxes) in PASCAL VOC format, the format used by ImageNet. In these XML files, each bounding box is described by its object label and its coordinates inside the image in xmin, ymin, xmax, ymax format.

At this point the dataset has the following structure:

1. A directory containing 300 RGB images encoded as jpg format.
2. A directory containing 300 annotations as XML files. Each of these file contains a list of the bounding boxes of the corresponding jpg image. Each bounding box should contain:
   (a) Its coordinates (with origin in top left corner) defined by 4 floating point numbers [ymin, xmin, ymax, xmax].
   (b) The class of the object in the bounding box.

### 4.3 Train and Test Split

The dataset was split to train and test subsets using a ratio of 80%-20% ratio. The first 240 images were used during the training phase of the models and the rest 60 images were utilized in evaluating them.

### 4.4 TFRecord conversion

The TensorFlow Object Detection API requires all the labeled data to be in TFRecord file format. The dataset described in this section is in PASCAL VOC format (annotations/labels are stored in individual XML files). For this format, the API provides the script `create_pascal_tf_record.py` which converts the dataset to TFRecord format. So,
4.5 Data Augmentation

It is true that most popular image datasets consist of images in the order of hundreds of thousands or even more. Object detection models are relatively complex neural networks in the sense that they have a lot of parameters to be tuned (in order of millions). To achieve good performance, the number of train examples shown to the model should be proportional to these parameters. So, these 300 images may be not enough in order to effectively train any object detection model.

This is a common problem when experimenting with deep learning applications. Fortunately, this can be tackled by augmenting the available dataset with synthetic data. Applying image operations like flipping, rotating, mirroring, etc. to the train images helps generate a number of synthetic images that are variations of the original train set. Then both the original and the synthetic images can be utilized during training phase, as the model will perceive all these images as distinct and unique examples and thus it will effectively learn from each of them.

Data augmentation can be achieved through a variety of approaches and software [17]. In our case there was no external augmentation done, as some augmenting operations are by default defined during the training phase of each model used in Chapter 4.
Chapter 5

Experiments

5.1 Setup

The following experiments were carried out on a machine belonging to the Information Systems and Databases Laboratory (ISLab) of Athens University of Economics and Business, running Ubuntu 16.04 LTS on a Intel Core i7-7700 CPU @ 3.60GHz x 8 and 16GB memory. In order to benefit from the installed Nvidia Quadro P400 graphics card, tensorflow-gpu 1.11.0, Cuda 9.0 and cuDNN 7.3.1 were installed.

5.2 Working with the API

The Tensorflow Object Detection API provides a large list of available models, ready to be customized and utilized for the needs of any object detection task. After choosing a model, the next step is to pick and experiment on its parameters, in order to fine tune it for the specific scenario.

The API uses protobuf files in order to configure a model. These configuration files define the pipeline of the workflow that will take place in order to train and then evaluate the model. Precisely, a config file consists of the following components:

- **model**: The meta-architecture (SSD, Faster R-CNN, etc.) and the feature extractor constituting the actual model are defined here. A number of parameters specifying clearly and definitely these structures are also set in this part. Moreover, the number of classes to be detected along with preprocess steps applied to the input images (e.g. resizing them to a specific size) are determined here.

- **train_config**: This part has information about the way the model will be trained. Batch size, optimizer, and training steps are defined here. The API provides training models either from scratch or using some pretrained weights. If the latter is the case, then the path to the weights file should be given in this part. Some data augmentation techniques, as proposed in the corresponding model’s paper, are also defined here.

- **train_input_reader**: Paths to the TFRecord formatted train set and the file defining the classes to be detected (label_map.pbtxt), as referred in Section 3.2.3), are set here.

- **eval_config**: The evaluation metrics set to be used is defined here (e.g. PASCAL VOC metrics, COCO metrics).

- **eval_input_reader**: Paths to the TFRecord formatted test set and the file defining the classes to be detected are set here.
After setting the above configuration, the API provides the `model_main.py` script, which conducts both training and evaluation of the described model.

Fine tuning a model to achieve maximum performance on a specific dataset is not trivial, as it is more of a trial and error approach. A number of different model setups and train configurations should be evaluated in order to find out the best solution.

### 5.3 Selected Models

In this thesis scenario, four models were selected to be utilized in order to face the specific dataset. These models were the following:

- SSD meta-architecture with Mobilenet_v2 feature extractor
- SSD meta-architecture with Inception_v2 feature extractor
- Faster R-CNN meta-architecture with Inception_v2 feature extractor
- Faster R-CNN meta-architecture with ResNet101 feature extractor

These models were selected from the Tensorflow object detection model zoo, where each available model is listed along with its recorded inference speed and performance. The SSD models were selected due to their good speed/performance tradeoff, while the Faster R-CNN models were selected due to their high performance. For each model, a number of different configurations were evaluated until the corresponding result was satisfying.

Both SSD models used 300x300 input images, while Faster R-CNN models used 300x400 input images. Attempting to use images of higher resolution as input to the Faster R-CNN with ResNet101 resulted in GPU running out of memory. A way that may help to avoid this situation is to reduce the batch size. On the other hand, a lower batch size may lead to training phase taking longer to complete.

The models were trained for 20000 steps and then were evaluated based on the PASCAL VOC Challenge 2010-2012 metrics set, recording the commonly used mAP at 0.5 IoU to compute the quality of the detections.

### 5.4 Experiments Results

#### 5.4.1 Training with pretrained weights

Models were trained utilizing pretrained weights. These pretrained weights are acquired by firstly training the corresponding models on some massive object detection datasets publicly available. Basic statistics for some of the most popular among them are shown in Table 5.1.

By firstly training on a generic dataset, like COCO, the models learn well to identify basic patterns and textures that tend also to appear in each dataset, resulting in good generalization. Thus, the idea in this approach is to build a robust model that will be able to adapt easily and quickly to any case-specific dataset.

Training on such a large dataset requires much computational time and power, but fortunately, these pretrained weights are already available and ready to be used.
5.4. Experiments Results

Table 5.1: Some well known object detection datasets. Records refer to the train sets of the datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Images</th>
<th>Bounding Boxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASCAL VOC 2012</td>
<td>20</td>
<td>5 717</td>
<td>13 609</td>
</tr>
<tr>
<td>COCO 2017</td>
<td>80</td>
<td>≈ 120 000</td>
<td>-</td>
</tr>
<tr>
<td>Open Images v4 [18]</td>
<td>600</td>
<td>1 743 042</td>
<td>14 610 229</td>
</tr>
<tr>
<td>ILSVRC 2017 [19]</td>
<td>200</td>
<td>456 567</td>
<td>478 807</td>
</tr>
<tr>
<td>iNat 2017 [20]</td>
<td>5 089</td>
<td>579 184</td>
<td>-</td>
</tr>
</tbody>
</table>

In this case, the models instead of starting the training phase from scratch, having initial random weights and trying to tune them directly on the specific dataset, they are initialized with these pretrained weights. Then the models are retrained on the specific dataset (here the crop and weed dataset) in order to be fine tuned and achieve maximum performance on it.

The mAP@.5, the train loss and the validation loss resulting from training the models utilizing the corresponding COCO pretrained weights are shown in Figure 5.1.

Figure 5.1 suggests the following:

- Faster R-CNN models perform undoubtedly better than SSDs, achieving 76.5% – 76.9% and 49.9% – 59.5% mAP@.5 scores respectively.
- mAP@.5 of Faster R-CNN models converges relatively fast, reaching its max score after about 600 training steps.
- mAP@.5 of SSD models converges with a slower rate, needing about double training steps compared to Faster R-CNNs.
- The best-performing model is the Faster R-CNN with ResNet101 feature extractor, resulting in 76.9% mAP@.5. The Faster R-CNN with Inception_v2 performs about equally well.
- Train loss of the models converges almost immediately, suggesting that due to the pretrained weights, it is easy for the models to adapt to the specific dataset.
- The SSD models end up with train and validation losses greater than that of the Faster R-CNN models. That indicates that the Faster R-CNN models not only learn better the specific dataset, but also that they generalize in a better way.
- Faster R-CNN models result in very low train and validation losses, close to zero.
- Validation loss of SSD models seems to be relatively high, possibly resulting in poor generalization. As a note, after a series of different train/test data splitting and training processes, the corresponding validation loss does not converge, suggesting that more train data are required in order to make this plot reliable for inference.
Chapter 5. Experiments

5.4.2 Training time

Figure 5.2 illustrates the time needed in order to train the selected models for 20,000 steps. As the Figure shows:

- Training requires a lot of time, as it is a computationally expensive procedure.
- Moreover, the Faster R-CNN with ResNet101 needs approximately 12 hours to be trained, making it the most time consuming among the selected models.
- It is interesting that the Faster R-CNN with Inception_v2 is trained in 4 only hours. Faster R-CNN models usually demand more training time than their SSD counterparts but here this is not the case.
5.4. Experiments Results

5.4.3 Detection of large, medium and small sized objects

A criterion to be considered when selecting the most suitable model for a specific dataset is its performance detecting objects of different sizes. Figure 5.3 illustrates how the models perform on detecting large, medium and small sized objects. As a note, the mAP used in this Figure, is not the PASCAL VOC version computed at 0.5 IoU. It is the one used in the COCO metrics set instead, averaging over all mAP scores from 0.5 to 0.95 IoU and with step 0.05.

From Figure 5.3 the following can be observed:

- All models detect larger objects more easily than smaller ones.
- Faster R-CNN with ResNet101 shows the best overall performance. It outperforms the rest of the models for each object size.

![Figure 5.2: Train loss against absolute training time (in hours) plot.](image)
Chapter 5. Experiments

5.4.4 Input size impact

This section explores the impact of the input image size on a model’s accuracy. Taking advantage of the Faster R-CNN with Inception_v2 relatively fast training, we retrain it by using the same train images but in a higher resolution, namely 480x640 (instead of 300x400). Figure 5.4 illustrates the result. It is clear that the high resolution images helps the model to achieve better accuracy. The mAP@.5 score increased in this way from 76.5% to 84.3%. In other words, by increasing the input by 60%, there was an 10.2% improvement in the accuracy of the model. On the other hand, this increase comes with extra computational weight (training time was increased from 4 to 6.5 hours -62.5% more time needed- for the same number of training steps).

5.4.5 Performance and Inference speed

Table 5.2 shows the achieved mAP@.5 and the inference time of the models after training for 20000 steps. In the above results, the models are mainly reviewed with respect to their accuracy but there is also value in researching the time needed for
5.4. Experiments Results

![Graph showing mAP@.5 for faster_rcnn_inception_v2 with respect to input size.](image)

Figure 5.4: mAP@.5 for faster_rcnn_inception_v2 with respect to input size.

each model to detect objects appearing in an image. As the Table 5.2 shows, the SSD models are faster than the Faster R-CNN models. The SSD with Mobilenet_v2 feature extractor has an inference speed of 0.624 seconds, being the faster, while the Faster R-CNN with ResNet101 needs over four times this time to perform detections (76.17% percentage decrease).

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP@.5</th>
<th>Inference time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssd_mobilenet_v2</td>
<td>0.595</td>
<td>0.624</td>
</tr>
<tr>
<td>ssd_inception_v2</td>
<td>0.499</td>
<td>0.975</td>
</tr>
<tr>
<td>faster_rcnn_inception_v2</td>
<td>0.765</td>
<td>1.229</td>
</tr>
<tr>
<td>faster_rcnn_resnet101</td>
<td>0.769</td>
<td>2.618</td>
</tr>
</tbody>
</table>

### 5.4.6 Detections

Figures 5.5 and 5.6 provide a visualization of the actual detections done by the models. Performance of the models seems to be clustered according to their meta-architecture, so detections by the most (Faster R-CNN with ResNet101) and the least (SSD with Inception_v2) accurate models are presented, while the rest are omitted.

These visualizations actually suggest the same with the mAP@.5 metric computed above, as the detections done by the Faster R-CNN are clearly more accurate than that done by the SSD model.
Figure 5.5: Predictions by faster_r-cnn_resnet101.

Figure 5.6: Predictions by ssd_inception_v2.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

We reviewed a set of state-of-art object detectors and utilize them in order to perform
detections on a specific real world dataset. From the experiments carried out and the
research done in the object detection field the following conclusions are drawn:

- Faster R-CNN models tend to achieve higher accuracy in terms of raw mAP,
  compared to SSD based models.

- If inference speed is the main concern, then the SSD models should be pre-
  ferred over the slower Faster R-CNN models.

- Thus, there is no one-size-fits-all model. The approach is more about selecting
  the most suitable model according to the specific characteristics and the needs
  of each case scenario. Choices about meta-architecture, feature extractor, input
  image size, software and hardware play a crucial role in the object detection
  results.

- It is highly recommended to use pretrained weights to initialize a model and
  then fine tune it by continuing training on the specific dataset. In this way, the
  convergence rate is much faster than that in the training from scratch approach.

- The SSD models, due to their architecture design, may perform poorly in de-
  tecting small sized objects. Small sized objects are mainly detected in the first
  convolutional layers and there may be not enough information of them in the
  last layers. The dataset used in the above experiments comes with a great
  number of small objects, as most of the weed instances are particularly small.

- This problem can be overcome by increasing the input image size. Of course,
  there is a tradeoff here, as high resolution input adds up computational weight.

- Apart from the small object issue, the SSD models may need more training
  data in order to improve and be more reliable.

- Even though the recorded inference time presented in Table 5.2 may be not
  capable of real time detections, it is highly expected that a better GPU would
  enable the same model configurations, especially the SSD ones, to detect ob-
  jects in real time. For benchmarking, the NVidia Titan X is commonly used.

- The choice of feature extractor is a significant one, as it affects both perfor-
  mance and inference time of the resulting model. There are feature extractors
  designed for either maximum performance (e.g. ResNet101), or high inference
  speed (e.g. Mobilenet_v2).
• Higher resolution image input can help improve accuracy of the models but it comes with extra computational weight.

6.2 Future Work

As future work, different configurations of the same -or other- models could be reviewed in order to further improve detections in the specific dataset.

Furthermore, a more comprehensive research could be done, experimenting with each model architecture and observing how each convolutional layer’s existence affects the performance and speed of the model.

Finally, repeating some of the above experiments, using the NIR images instead of the RGB ones could produce results worth reviewing.
Bibliography


