UNIVERSITY OF HERTFORDSHIRE

School of Computer Science

Modular BSc Honours in Computer Science

6COM1053 – Artificial Intelligence Project

Final Report

June 2020

TITLE OF PROJECT

Deep learning for instance segmentation of cucumbers in greenhouses

A P Hernández

16084125

Supervised By: Farshid Amirabdollahian

Contents

Abbreviations 5

1. Introduction 6

Project approach 6

2. Investigation 7

Earlier works 7

Why Deep leaning 7

Deep learning revolution for computer vision 7

Object detection using CNN 9

Mask R-CNN 9

Backbone 9

Feature Pyramid Network 10

ROI Classifier & Bounding Box Regressor 11

Segmentation masks 11

Non-max Suppression 11

Transfer Learning 12

Optimization Function 13

Metrics 14

3. Design 15

Dataset acquisition 15

Dataset preparation 15

Enviroment 16

Testing 17

4. Implementation 17

Experiment 1: Learning Rate Comparation 19

Experiment 2: Data augmentation 21

Experiment 3: Fine tune with training schedule 22

5. Evaluation 22

Project evaluation 25

Future work 26

Personal Development 27

Conclusion 28

References 29

Table of figures 31

Appendix 32

Data augmentation 32

Code 34
Model preparation 34
Dataset class and preparation 36
Data augmentation 39
Training 40
Evaluation mAP 40

Abstract

This Project presents a precise and reliable detection system for cucumbers. The aim is to build a detection system using deep convolutional neural networks. Such a system is key to build an autonomous agricultural robot to use for automated harvesting, yield estimation and plant disease prevention. Mask-RCNN (regional convolutional neural network) is a recently proposed state-of-the-art algorithm for object localization, object instance segmentation and object detector. We adapt this model, through transfer learning using the backbone network Resnet101 and combining with the Feature Pyramid Network (FPN).

Abbreviations

Regional convolutional neural network (R-CNN)

Deep neural networks (DNN)

Deep convolutional neural networks(CNN)

Region Proposal Network (RPN)

Feature Pyramid Network (FPN)

Regions of interest (ROI)

Non-maximum Suppression(NMS)

Regional CNNs (R-CNN)

Region of Interest Pooling (RoIPool)

Stochastic Gradient Descent (SGD)

Intersection over Union (IoU)

1. Introduction

According to the world Resources Institute by 2050 there will be 10 billion people on this planet projecting a challenge forward to feed an ever-growing world population. This situation combined with the problem of climate change are driving up the prices of agricultural supplies such as agrochemicals and creates water scarcity. It is driving small and medium farm enterprises to deal with the pressure of small profit margins while competing with the prices of big farm enterprises.

As stated in the Journal of Political Economy (Floyd, 1965) one of the most cost-demanding factors in agriculture is the farm labour. Therefore, robot harvesting (high repeatability and more working hours) is a great solution to this problem by reducing the costs of manual labour in the fields. Traditional methods cannot meet any longer the ever-increasing demands of the food industry. This has led to the development of numerous functional harvesting robots.

Despite the large number of harvesting robots available, the detection system is still lacking behind other tasks such as picking and manipulating the vegetables. This is why a fast and precise object detection is key to develop a robot harvesting system. Previous work on this area has been done combining traditional machine learning techniques including k-means and SVM with computer vision in order to identify the key objects in images. This approach has had limited success as it is able to correctly identify the object but the accuracy decreases when changes in illumination, partially covered vegetables and shadowing of the leaves to the vegetables.

In contrast with the traditional machine learning algorithms, deep neural networks have better accuracy in changing environments due to the autonomous ability to learn and the strong feature extraction of their algorithms. This is crucial when dealing with different types of lighting and uncertainties in the harvesting fields. The initial consideration for this project was using Faster R-CNN following the steps of Inkyu et al. (2016) that he successfully applied to the detection of sweet pepper. An investigation in depth about the available solutions using the family of algorithms R-CNN showed that actual applications were mainly to detect easy picking fruits such as apples, pears and peppers as the way of harvesting them is pulling. Cucumbers are more complicated as they require to cut the stem that connects them to the plant, meaning a more precise contour and object shape is required in order to select the orientation and cutting points to harvest. The new framework, Mask R-CNN from the same authors of Faster R-CNN, extended the functionality allowing the detection of both bounding boxes and pixel level annotations.

Project approach

The idea behind the project is to use convolutional neural network to detect cucumbers in different lightning and growing conditions. For this given task we will make use of an already existing network trained for general detection of common objects. The advantages of using an already trained network are the computational power savings, as to train from scratch could take weeks. In this way the network may not recognise cucumber at first but it already knows a lot of features of natural images and we use that for the baseline of the new network that we will fine tune to add a new category, cucumbers. For the task of predicting a mask inside the bounding box we will explore the different approaches in the R-CNN family of algorithms to see which one best suit our needs.

2. Investigation

Earlier works

Previous works employed in fruit detection used the feature engineering method that takes human based descriptions on colour, shape and texture. Those descriptions are then feed into machine learning techniques such as support vector machines, clustering or Bayesian classifiers. In 2004 a researcher (M. DUNN, 2004) presented the first attempts to use image processing for fruit detection on grapevines. The early approach did not yield the results expected as accuracy decreased quickly with changes in the growing environment occur.

Recent approaches involved using convolutional neural networks for object detection. An example of this is the work done by (Sa, 2016), where he used deep convolutional neural networks for fruit detection. The system combined RGB and Near-infrared images to build the detection system using Faster R-CNN.

Why Deep leaning

In a traditional classification task there are only two possible solutions, either a yes or no along with the probability score, making the objective to draw with precision the decision boundary that separates between categories. Using logistic regression works when the data is linearly separable but it does not perform well extracting the non-linear relationship. In SVM a trick can be used to make it work on non-linear separable data by applying transformations mapping the original input space into a higher dimensional feature space.

Using the kernel trick in SVM can produce good results but they are not as good as the results using CNN to classify non-linear separable data. This is important as the project is a classification task in image recognition. CNN performs particularly good going through huge amounts of image data and finding the non-linear relationships of the categories.

This project needs object detection and contour drawing of the detected categories. This is a bit different from the regular classification algorithm as in object detection not only classifies but locates the objects inside the image drawing a bounding box around. Nowadays deep learning is the best technology to use for computer vision.

Deep learning revolution for computer vision

In 2012 a new area of machine learning was established called deep learning. This area has produced breakthrough advances in machine learning especially in computer vision. It was possible with the introduction of deep convolutional neural networks (CNNs) (Krizhevsky, 2012).

The initial architecture of the network proposed by Krizhevsky had 8 layers: five convolutional layers and three fully-connected layers. The breakthrough introduced by Krizhevsky introduced some new approaches to this network that make it a breakthrough in computer vision. The highlights of the approaches described in the paper:

- ReLU Nonlinearity: The use of ReLU activation function made the training 6x times faster without compromising the accuracy
- Dropout: Each neuron has a probability of turning off. This technique allows to every iteration to use a new set of model parameters, forcing the network to learn stronger features and makes it more robust against overfitting. The downside is that makes the training slower, doubles the time with a probability of 50% dropout.
- Overlap pooling to reduce network size(?)

After the introduction of Alexnet in 2012 several other architectures were published: GoogLeNet (Szegedy et al., 2014), VGGnet (Simonyan and Zisserman, 2014) and ResNet (He, Zhang, Ren and Sun, 2015).

that we have chosen for the backbone architecture of the project

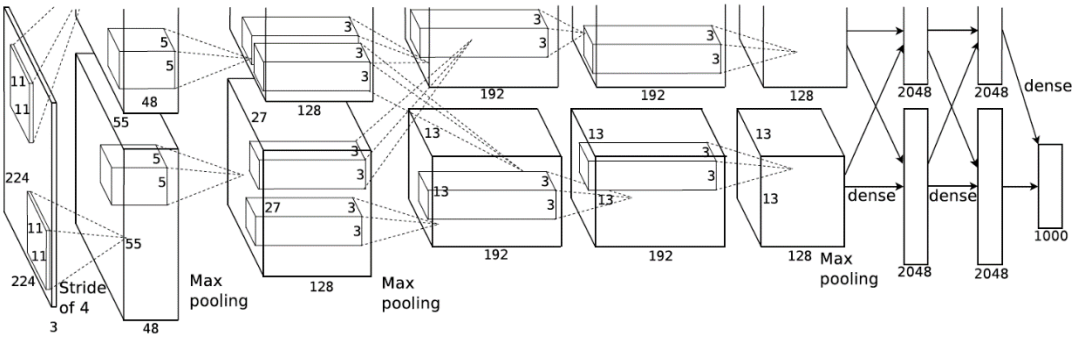

Figure 1, AlexNet architecture

CNNs were inspired by observing the biological processes involved in the connectivity between neurons in the visual cortex of animals. The theory is that neurons at different levels in the hierarchy of the visual cortex process different features of the image. Low level neurons scans for low-level image features such as the edges. These low-level neurons feed the output to higher level neurons that compute more complex features (e.g. small object shapes). Last layer of neurons compute all the lower-level features into high level features (e.g. larger object shapes such as a person or a building).Using these principles CNNs are constructed making them a powerful tool for image classification that could take on more challenging computer vision techniques.

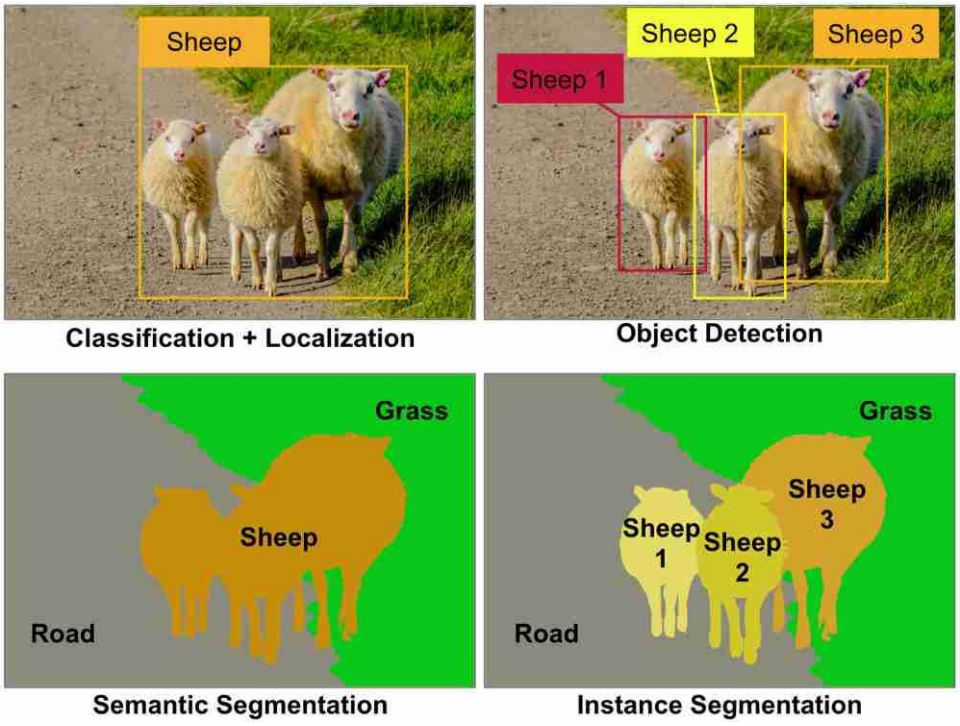

Figure 2, Computer vision tasks

The selected computer vision task for this project was instance segmentation to allow precise pixel-level detections of the shapes of objects detected. This is particularly important in the detection of cucumbers as they require precise location to locate the stem and collect them efficiently.

Object detection using CNN

Object detection bounding box around the different objects in the image. It finds the objects by creating regions of interest in the image, classifying and refining the proposals generated. Traditional approaches to object detection were computing intensive because of the problem generating huge numbers of regions. In 2014 regional CNNs (R-CNN) were introduced (Girshick, 2014) to address the problem of selecting big numbers of regions. The method consisted in selecting a fixed number of regions, just 2000, what he called "region proposals". The region proposals are generated by an algorithm called selective search (Uijlings, 2020). These region proposals are fed into a CNN that produces an output of a 4096-dimensional feature vector. The CNN extracts the features of the image and pass them to a SVM to classify the presence of the object inside the proposal.

Although R-CNN was a breakthrough there was still a huge amount of resources needed to train the network classifying the 2000 region proposals of every image in the dataset. This computing cost can be seen reflected in the 47 seconds that takes for a single image to detect the objects within. This was improved in the next iteration of the algorithm by the same author, Fast R-CNN (Girshick, 2015). This new version introduced the Region of Interest Pooling (RoIPool) that removed the need to feed all the regions proposals to the CNN, instead we pass the image to the CNN to generate a convolutional feature map. The reason this new method is faster is because the convolution operation is done once per image in place of the 2000 region proposals. The training times improved almost by a factor of 10 and inference times by a factor of 20.

The following year the same author introduced Faster R-CNN (Ren, He, Girshick and Sun, 2016) that improved in the selection of region proposals. Faster R-CNN eliminates the selective search algorithm and uses a separate network to learn the proposals, called Region Proposal Network (RPN). We will talk about RPN in the next section. Faster R-CNN was at the time of being released the model with the best performance for object detection outperforming the rest of existing solutions.

Mask R-CNN

Mask R-CNN is a deep neural network designed to solve instance segmentation problem. It was created by the same authors (He, Gkioxari, Dollár and Girshick, 2017) as the rest of the R-CNN family and extends the Faster R-CNN network by adding an extra feature to do instance segmentation. At a high level the network can be explained using a two-stage framework where the first part generates the regions of interest and the second one classifies the categories and generates bounding boxes and their respective masks.

Backbone

The backbone of the convolutional neural network is the ResNet101 that we used as a feature extractor. In 2015 the ResNet architecture was introduced (He, Zhang, Ren and Sun, 2015) to propose a solution to train deeper neural networks. It features a new novel architecture with "skip connections between layers", called residual blocks, and heavy batch normalization.

The residual block architecture is now an essential component of modern neural networks allowing to effectively train model of hundreds of layers.

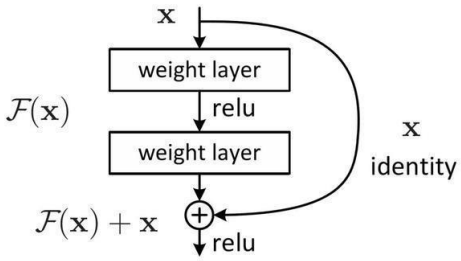

Figure 3, Residual Block

The problem with deeper networks is the vanishing gradient that makes the loss function goes to zero after several iterations. This results in weights not updating the values comes with the consequence of no new learning occurring. In ResNet the gradients flow directly though the connections between deep layers and shallow solving the problem.

Feature Pyramid Network

The FPN improves the process of feature extraction in the backbone by adding a second pyramid network to take the high-level features from the first pyramid and passes them down to lower layers to allow features at every layer to have access to both high and lower features.

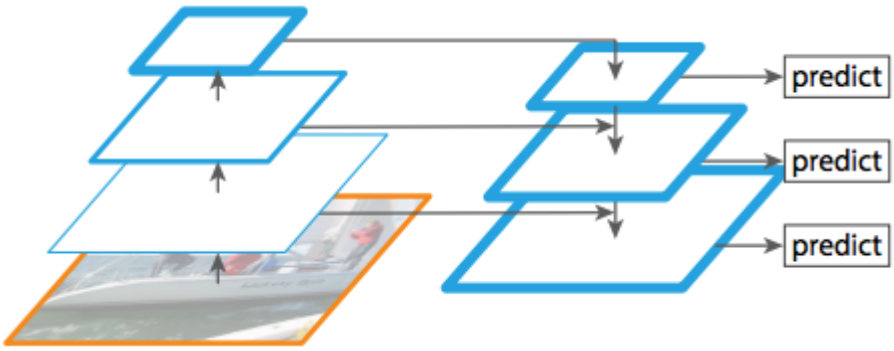

Figure 4, Pyramid Network

The Region Proposal Network (RPN) is a lightweight neural network that allows the scan of an image to propose regions of interest (ROI). The ROIs distributed across the image are called anchors. The RPN generates two outputs for each anchor: class and bounding box.

- **Class:** Indicates whether the anchor is likely to contain the background or the foreground, in this case the cucumber.
- **Bounding box refinement**: The positive anchors (foreground anchors) are not always centred to the object detected. The coordinates to refine an anchor to fit the object are given by the RPN calculating delta(% in coordinates, width and height) to refine the positive anchor to fit the object.

There are thousands of anchors per image and many of them overlap the same objects from the same class. To avoid duplicates and select the best bounding box for the object a technique called Non-max Suppression was selected.

ROI Classifier & Bounding Box Regressor

The regions of interest (ROI) obtained by the RPN are taken as inputs to produce a more precise classification, called SoftMax. The RPN had only two classes foreground and background, this network can classify multiple specific classes. In this stage similarly to what the RPN does to refine the bounding box a bounding box regressor is used to further refine the bounding box to adjust the location of the object

Segmentation masks

The last part of the Mask R-CNN framework is the output of segmentation masks. This task is done by the mask branch that is a convolutional network that takes the positive ROIs generated by the ROI classifier as inputs and outputs a low resolution minimask (28x28). Loading the model in inference mode scales them up to the size of the ROI and colours the contour of the object to the pixel level.

Non-max Suppression

Is an algorithm to compute the overlapping between different regions of interest. It is present in almost all modern machine learning pipelines. The pipeline of Mask R-CNN generates hundreds of proposals for the same object. The filter works by selecting regions of interest above a certain threshold of confidence, in this case is set at 0.7 to filter some low probability anchors. The image below shows an example.

Figure 5, Anchors before NMS where hundreds of category independent region proposals are generated

Figure 6, Anchors and detections after NMS with labels

Non-max suppression makes sure that only the object is identified once. It computes the optimal bounding box among all generated. The algorithm works like this:

Input: A list of Proposal boxes B, corresponding confidence scores S and overlap threshold N.

Output: A list of filtered proposals D.

Algorithm:

1. Select the proposal with highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2. Now compare this proposal with all the proposals — calculate the IOU (Intersection over Union) of this proposal with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B.
3. Again, take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D.
4. Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have high IOU than threshold.
5. This process is repeated until there are no more proposals left in B.

IOU calculation is employed to measure the overlap between two proposals. NMS filters the proposals to just one per object.

Transfer Learning

Is a machine learning technique consisting on storing knowledge about how to solve one problem and applying that to a different but related problem. Andrew Ng stated that “transfer learning will be the next driver of machine learning success”. This will open the door for new applications of machine learning with limited success or previously untapped.

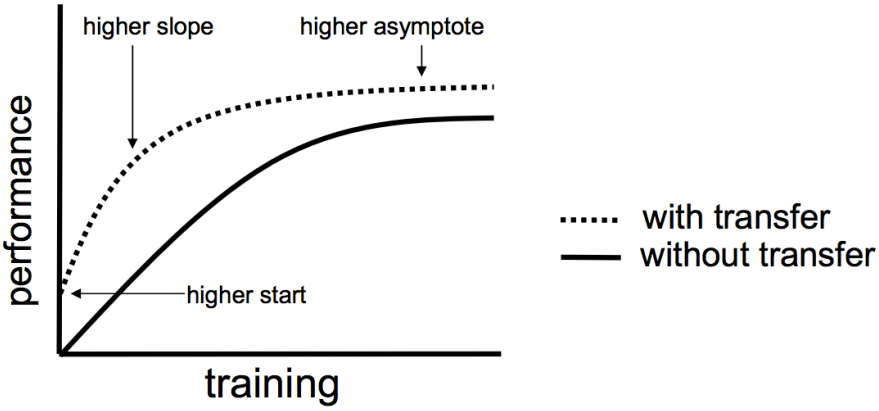

Figure 7, Transfer learning

Using transfer learning allows much lower training times with high accuracy and limited data. As modern convolutional neural networks have more layers the number of parameters increase exponentially, sometimes in the range of millions, making training the whole network unpractical for some applications.

We will be using transfer learning pretrained in the COCO dataset as it has already knowledge of features in natural images. This will allow for fast iterations of the experiments as we do not need to train each time from scratch and allows to select which parts of the network to train on.

For better results using transfer learning a technique called Fine tuning is used. It can be used when the new dataset is not radically different from the one that has trained the network originally. The COCO dataset that we will be using has learned universal features from natural images like the curves and edges of objects. This features are stored in the early layers of the model.

To fine tune the network we need to remove the last classification layer and freeze the layers of the base model. We then feed our dataset to the classification layer to train on the new categories of objects in our problem. We can improve further the tuning of the network by reducing the learning rate and allowing deeper layers to train.

Optimization Function

In deep learning an optimization function is used to reduce the cost function. In deep learning the cost function is defined as:

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(y'^{i}, y^{i}\right)$$

Figure 8, Optimization Function

Where the cost function J is the mean of the loss computed by the predicted value Y' and the ground truth Y.

The optimisation function for this project was decided between two algorithms, Adam (Kingma and Ba, 2014) and SGD (Stochastic Gradient Descent). From the first experiments comparing two popular optimisation functions showed that Adam optimization algorithm perform better lowering the training loss but SGD obtained the lowest validation loss. This is

because adaptative methods such as Adam perform well on early stages of training but but are outperformed by SGD at later stages of learning (Shirish Keskar, 2017). For this reason, we decided to use SGD to train a model for better generalisation purposes.

“A gradient measure how much the output of a function changes if you change the inputs a little bit.” — Lex Fridman (MIT)

Gradient Descent is and iterative algorithm that starts from a random point of a function and finds the lowest point of the function. It is a popular optimization technique present in almost all deep learning models. A gradient is the slope of a function that measures the degree of change of a variable in response to the changes of another variable. Stochastic means randomness, in Stochastic Gradient Descent (SGD) a few samples are randomly selected from the whole dataset in each iteration. The optimisation function uses a batch (Number of samples) to calculate the gradient in each step.

The steps of the algorithm are

1. Find the slope of the objective function with respect to each parameter/feature. In other words, compute the gradient of the function.
2. Pick a random initial value for the parameters. (To clarify, in the parabola example, differentiate “y” with respect to “x”. If we had more features like x1, x2 etc., we take the partial derivative of “y” with respect to each of the features.)
3. Update the gradient function by plugging in the parameter values.
4. Calculate the step sizes for each feature as : step size = gradient * learning rate.
5. Calculate the new parameters as : new params = old params -step size
6. Repeat steps 3 to 5 until gradient is almost 0.

Metrics

The cucumbers instances will be evaluated by comparing the predicted bounding boxes and segmentations to the ground truth in the validation dataset. The metric will be the average precision at IoU (Intersection over Union) at value 0.5. This evaluation metric refers to the overlapping of the detection bounding box or segmentation mask with the annotation of the image. IoU is calculated by dividing the area of overlapping between two objects by their area of union. For an IoU of 1 the regions need to match exactly whereas an IoU of 0 means they do not overlap at all. The instance segmentation method produces a number of predicted instances and their confidence, the detection score.

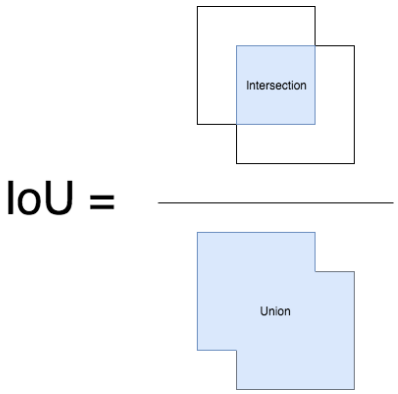

Figure 9, Intersection over Union

3. Design

Dataset acquisition

The images used to make the dataset for this project were given by the supervisor from an existing project recorded from an undisclosed location. The hardware used to record the images was the webcam Logitech C920 with a resolution of 768x640 and a 10 frame/second rate.

Dataset preparation

The images were all resize to 1024x800 to feed the CNN to reduce computation needs as well as training and testing times. A total of 306 RGB images were used to create the dataset with 80%(250)used for training and the 20%(56) left for validation of Mask R-CNN. This subset of the whole dataset was selected because of the time involved making the pixel annotations from scratch for this project rather we will make up for it using data augmentation.

The tool used to create the masks was VIA (VGG Image Annotator) because of the simplicity and ease to use. It is a self-contained html document that can be opened in a browser and work with no installation or setup.

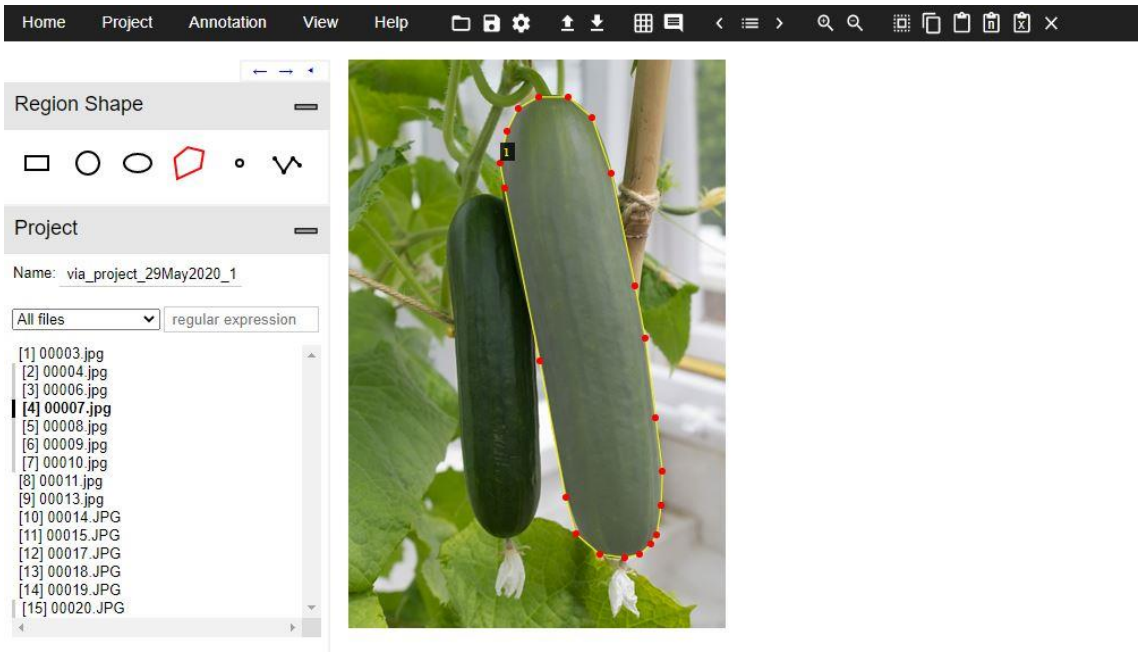

Figure 10, Annotation in VGG

The task of creating pixel level masks is time consuming, taking a few minutes per image, but once adjusted to keyboard shortcuts it gets easier reducing the time involved per image. The problem presented in this project is binary, meaning the annotations only have two classes: cucumber and background.

The images were in JPG file format with the annotation files generated by VIA in JSON format. JSON format uses a set of polygon points that define the mask annotation that can be used to train the network using the x y coordinates of the annotation.

Enviroment

The initial consideration for the training of the CNN was to use a desktop computer with an NVIDIA GTX 960 4GB DDR5 and 24GB RAM memory. The installation of the different components needed for the project was successful. Several problems became present in the training part, due to low GPU memory. We tried switching the batch size to 1 to lower the gpu memory needed during training without any result. Due to the limitations on the hardware the reasonable choice to train the network was using cloud computing.

A handful of cloud computing services were considered but finally Google Colab was chosen because of the low setup needed and availability of GPUs for free use. Colab is a Google research project aimed to help the machine learning education and research allowing anyone to use it with a Google account and share the code examples to anyone through a link. It runs on Jupyter notebooks and lets you use Google Drive as storage for the experiments.

Although Colab is an already configured environment there are installations needed of some software libraries to make the Mask R-CNN work. Software libraries:

Python: One of the most popular programming languages specially used in machine learning for the flexible and numerous tools available to do deep research. It is a high-level, interpreted, general-purpose programming (Python.org, 2020).

Jupyter Notebooks: Open source web application that allows to create and share documents of code and visualizations.

TensorFlow: End-to-end open source platform for machine learning. It is a flexible and complete ecosystem featuring an enormous community with tons of resources (TensorFlow, 2020).

Keras: Open source library written in python to develop neural networks applications. It is user-friendly through a simple interface to make the task of debugging easier. It offers a high-level API to minimize the user actions required for most cases (keras, 2020).

Imgaug: Library to make data augmentation on data to use in machine learning experiments. It features numerous types of transformations and allows to build simple yet powerful image augmentation routines.

Matplotlib: Software library to create visualizations in Python.

Numpy: Scientific computing library written in Python for management and computations of arrays and use of mathematical functions.

VGG Image Annotator: Is a simple manual annotation software for image that is lightweight and can be run offline.

Software versions of the libraries employed in this project:

Table 1, Different software versions used for the project

Library	Software Version
Python	3.6
TensorFlow	1.15
Keras	2.25
Imgaug	0.4
Matplotlib	3.2.1
Numpy	1.18
VGG Image Annotator	1.0.5

Testing

Every piece of code needs to be tested to make sure it is working as intended. To test the neural network we keep a close look to the training losses values both in training and validation datasets to verify everything is running as intended. Once the network is already trained we verify making detections on the validation dataset and looking at the precision-recall curves and mAP. We can inspect the activation on different layers to see why the network is making the decisions

4. Implementation

For the experiments we used the implementation of the Mask R-CNN framework developed by Matterport Inc, publicly available in GitHub (matterport , 2020). It uses TensorFlow and keras making it customizable for the numerous experiments. The initial weights are based on the COCO Dataset(Lin et al., 2014). Based on the implementation we forked and modify the files for the project.

To configure the training pipeline for the experiments, there are four elements to prepare:

- Dataset
- Model Preparation
- Configuration of hyperparameters
- Train Command

Dataset

The dataset is loaded and prepare to pass it to the train command via the class cucumberDataset. This class has functions that loads the images and annotations with the masks and prepares them for training or validation purposes. An example of code to prepare both validation and training datasets is shown below.

```
# Training dataset.
dataset_train = cucumberDataset()
dataset_train.load_cucumber(dataset_dir, "train")
dataset_train.prepare()

# Validation dataset
dataset_val = cucumberDataset()
dataset_val.load_cucumber(dataset_dir, "val")
dataset_val.prepare()
```

Model Preparation

In order to train the model we first need to initialize and select model mode(inference or training), load the weights and compile to pass the learning rate and optimize function.

To initialize the model for training we select training mode and pass the configuration file and model directory to save the checkpoints during training.

```
model = modellib.MaskRCNN(mode="training", config=config,
model_dir='/content/drive/My Drive')
```

We need to load the pretrained weights onto the model. We are using COCO pretrained weights as they have already knowledge of features of common objects in hundreds of categories. We will remove some parts of the pretrained weights in order to initialize them for our project.

```
model.load_weights('/content/drive/MyDrive/TFG/Via/mask_rcnn_coco.h5',
by_name=True, exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
"mrcnn_bbox", "mrcnn_mask"])
```

One last step is needed to prepare the model for training. We compile the model to pass the optimizer function and set the losses. First the initialization of the optimizer. Here we are loading the SGD optimizer.

```
optimizer = keras.optimizers.SGD(
            lr=learning_rate, momentum=momentum,
            clipnorm=self.config.GRADIENT_CLIP_NORM)
```

We then compile the model using:

```
model.compile(optimizer=optimizer,
            loss=[None] * len(self.keras_model.outputs))
```

Configuration

The baseline configuration of the hyperparameters used in this project. Some hyperparameters were changed from the ones used in the implementation.

Number of classes	2
Images per GPU	2
Steps per epoch	470
Batch size	2
Backbone network	ResNet 101
Mask shape	28x28
RPN Anchor scales	32, 64, 128, 256, 512
Train ROIs per image	200
Learning Momentum	0.9

In order to pass a certain configuration to the model first we need to edit the hyperparameter we want to change in the configuration file. For the configuration of this model we changed the following parameters:

- Learning_Rate: We will test different learning rates to find the optimal one.
- Steps_per_epoch: We have set this value to 470. The number of images in the training dataset multiply by the batch size
- Validation_steps: This value was increased to 100 to increase validation accuracy at the cost of slowing down a bit the training phase.
- Num_classes: There are only 2 classes, cucumbers and the rest is background.
- Detection_max_instances: Lower the number of detections per image to 25 as it is rare for images to have double digit detections.

Train

Finally, we pass the model the training function. With this command you can choose the length of the experiment defining the number of epochs, select the number of layers to train the network on as well as pass the augmentation routine and callbacks. We have included a timer to monitor the time that takes to execute every experiment.

```
start = time.time()
model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=20,
                layers='heads',
                augmentation=augmentation,
                custom_callbacks=callbacks)
print('Elapsed time', round((time.time() - start)/60, 1), 'minutes')
```

Experiment 1: Learning Rate Comparation

Selecting the right learning rate is crucial. Too high will and too low will. For this reason, we tested a few different learning rates including the one from the implementation which is set to 0.001.

We let the model run for 20 epochs only as we are testing the right learning rate. We are not applying any kind of augmentation to the dataset because we need to test under same conditions. Here is the comparation in training and validation loss.

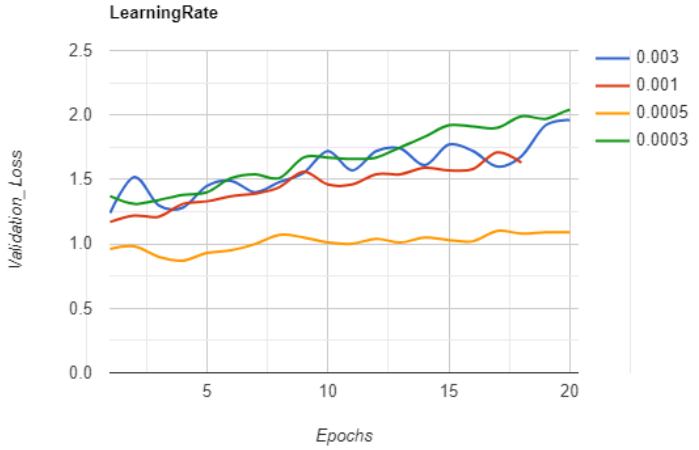

Figure 11, Validation loss graph

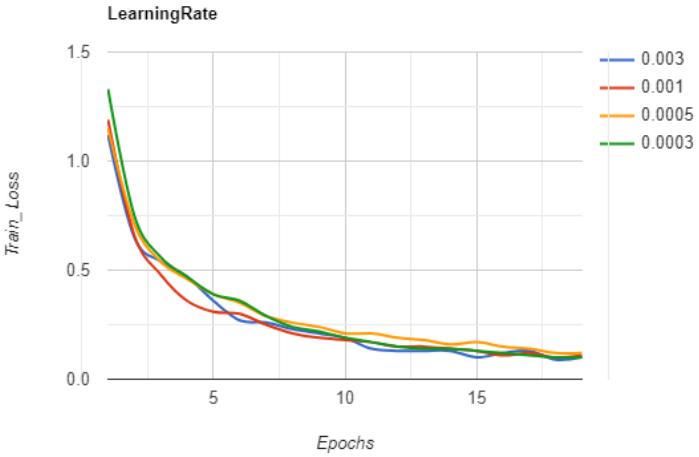

Figure 12, Training loss graph

From the results shown above we can appreciate using the SGD optimizer makes the training loss of all different learning rates similar but in the validation loss the 0.0005 performed the best. Plotting both losses on the same graph shows how they diverge. We stopped the training in epoch 20 as it was already clear that in the first 5 epochs there was a singular aspect, the validation loss was increasing. This is a common behaviour indicative of overfitting and we will try to address this issue in the following experiments.

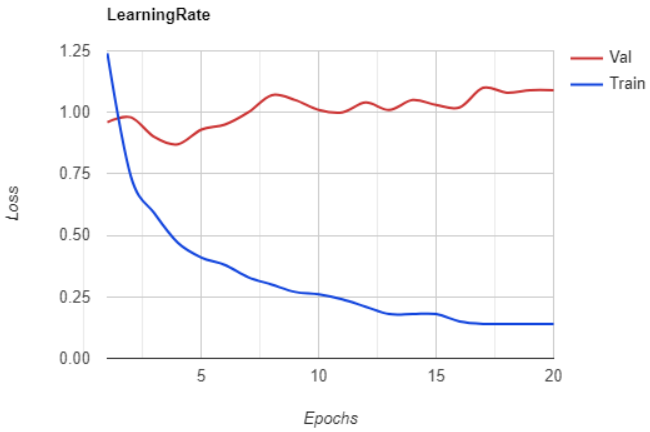

Figure 13, Loss values during training

Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the features and noise in the training data to the extent that it negatively impacts the performance of the model on new data (Gupta, 2020). There are different ways of addressing overfitting in deep learning, we have decided to use data augmentation and evaluate how well the model performs.

Experiment 2: Data augmentation

Recent advances in convolutional neural networks have been thanks to the publicly available datasets to test the experiments on. Data augmentation is a technique to increase the variety of data used to train models. This strategy does not require collecting new samples instead it uses different techniques to modify the samples such as flipping, blurring and rotating.

Data augmentation is often presented as a type of regularization. According to (Goodfellow, 2020) “Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but no its training error”.

The Mask R-CNN framework is prone to overfit due to the complexity and the backbone used, ResNet 101. Due to the possibility of overfitting the network using a relatively small dataset, we have tested several kinds of augmentation to create a more diverse dataset and measure the changes in the network.

We choose to use Imgaug to make the transformations in the images.“Imgaug is a library for image augmentation in machine learning experiments. It supports a wide range of augmentation techniques, allows to easily combine these and to execute them in random order or on multiple CPU cores, has a simple yet powerful stochastic interface and can not only augment images, but also keypoints/landmarks, bounding boxes, heatmaps and segmentation maps” (imgaug, 2020). We constructed four different augmentation routines to test different types of transformations and see how that affected the accuracy. For visualization purposes we will show an example of each transformation from the original image in the appendix.

1) Geometric transformations: Flipping vertically and horizontally.

2) Arithmetic transformations. Modifications of the pixels values in images. Here we use either one of these two functions: Add (random values between -40 and 40 added to the image) and Multiply (Multiplies the image by a random number between 0.5 and 1.5). These transformations alter the contrast in the images to see whether that affects detection in different lightning conditions.

3) Arithmetic transformation. Modifications from the previous routine plus Gaussian blur and Linearcontrast. This modifications blur the image and change the contrast to see if that improves detection in different lightning conditions.

4) Combinations of geometric (horizontal and vertical flips) and arithmetic transformations(Add and multiply).

5) Segmentation transformation: Partially modify images to their superpixel representation. The image gets segmented in 64 superpixels and inside those the pixels are given a probability of 50% to get converted into the average pixel color

Experiment 3: Fine tune with training schedule

In order to further tune the network we will repeat the test previously done with a training schedule to train the whole backbone of the network. We will try the experiments with the following training schedule:

1. Train the top layers of the network for 15 epochs with a learning rate of 0.001
2. Resume the training from the fourth layer and up of the backbone and train with a learning rate of 0.0005 for another 15 epochs
3. Finally we let the whole network to be trained with a really low learning rate of 0.0001 for an additional 15 epochs.

We change the learning rate as deeper layers are training to avoid changing radically the features that has already learned as we are only slightly tuning the network for our problem.

Issues

During implementation we faced some issues:

- Drawing the precision-recall plots over a batch of images throws an error we could not fix so we can only plot the precision-recall for each detection.
- Usage limits from the free version of Google Colab, forcing us to upgrade the paid version to allow training without limits.
- Error not allowing the training to start. This was solved downgrading the keras library version from 2.3.0 to 2.2.5.
- Error in the annotation loading function that made us change the format of the annotations from COCO like dataset to JSON format from the VGG annotator.

5. Evaluation

The validation dataset contains 53 images randomly selected from the original dataset. This images were not shown to the network during training. The metrics are explained in more

detail in section 2.9, for the evaluation we will be comparing the mean average precision results from heads training only, the top layers of the network, and the network with all the layers fine tuned for our problem.

First the results from the experiment 1 are shown in the section because the rest of the experiments are based on that learning rate of 0.0005. We only have modified the learning rate fine tuning the network making it smaller as more layers were involved during training to preserve the knowledge learned previously.

Table 1: Mean average precision with intersection over union set at 0.5 for the single-class experiments. Metrics are calculated running predictions on all images of validation dataset and averaging the results.

Experiment	Heads training	Fine Tuning
Base Configuration	0.7104	0.8420
Augmentation 1	0.7604	0.8644
Augmentation 2	0.7650	0.8777
Augmentation 3	0.7736	0.9102
Augmentation 4	0.7491	0.8329
Augmentation 5	0.6479	0.7042

In the baseline configuration, without any modifications to the data and using a learning rate of 0.0005 the results are 71% precision for heads training and 84% precision for the fine tune network. This shows the big jump in precision when allowing not only the top layers to train, but the whole backbone to tune it to the new problem.

Applying augmentations using only the heads part of the network already shows the precision improvement in the range of 3-6%. We found in the augmentation 5 using the segmentation transformation has a detrimental effect on precision. The best augmentation were those that did not alter the geometry of the image but rather the arithmetic values allowing the network to pick up a broader set of lightning and contrast conditions.

The precision results from the fine tune network using augmentation are superior to the base configuration except for two cases, the segmentation augmentation and the combination of geometry and arithmetic . The segmentation transformation even with fine tuning did not reach the base configuration precision at 71%.

We can appreciate that fine tuning has a bigger impact on precision than applying data augmentation alone and we get even better precision using both techniques reaching average precisions as high as 91%.

We will take a look at the best performing model, the one using arithmetic transformations add, multiply, linear contrast and gaussian blur. We can see the detections and masks created compared to the original image.

Figure 14, Original image

Figure 15, Predicted cucumbers and masks

Figure 16, Original image

Figure 17, Predicted cucumbers and masks

We can see that is pretty good detecting cucumbers in close range, those are the only ones we care about as they are in the range of the robotic arm to pick them up. The system is able to detect cucumbers that are partially covered with branches and leaves. The precision-recall curves from the previous two images shows the model maintains a high precision at high recall values:

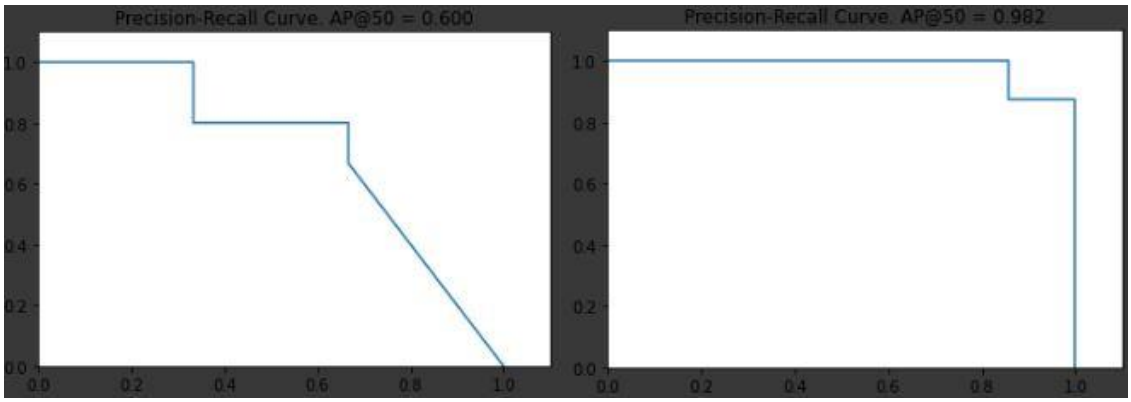

Figure 18, Precision-recall graphs from previous two detections in order, left for figure 15 and right one for figure 17

Project evaluation

The final artifact keeps the original aims for the project intact and can perform instance segmentation with high precision. It can be considered that the project has been successful.

The original timeline predicted for the project has changed to adapt the new circumstances encountered such as the disruption due to the Covid-19 and the new deadline for the project. Overall considering the circumstances the management of the project has been good and a high standard of work has been maintained.

My initial goal was to create a system that could detect cucumbers using bounding boxes only. This was later modified after the investigation from the problems that harvesting robots face

when deployed on the fields. I believe this work can be used in the agriculture sector with slight modifications such as making the dataset bigger so overfitting happens to a less degree and precision is improved.

The tools I used has been proved to be the right ones. The programming language python that I have learned in the course and used in personal projects has kept the complexity level accessible and the enormous community around has helped in the development of the system. I wanted to run everything in my desktop but that approach proved to be the wrong one due to hardware limitations and I was forced to use the online cloud computing service from google, Colab. Using such high-end graphics card the training times were low and could feed multiples images to the network during training without running out of memory.

The dataset preparation turned out to be more time consuming than I have previously considered. This led to a setback of the original plan to allow more time to be invested creating the pixel-level annotations.

Future work

Some ideas that are beyond the current scope of the project:

- More training data: The process of manually annotating pixel-level annotations is time consuming and further work on this area would be beneficial to improve precision and help prevent overfitting.
- Resnet 50: Implementation of a shallower network would have and compare the results between the different backbones. Using ResNet 50 would in theory prevent to a certain grade overfitting.
- Algorithm to draw picking points: The task of picking up a cucumber with a robotic arm requires to draw points for the robot to understand the shape of the cucumber and where the stem is in order to proceed with the harvesting.
- Cucumber counting: Implementation of a counting system using the network proposed in the project to allow the harvest estimation.
- Pest prevention: The network allows for multiclass training and could be trained to detect the most common pests found in greenhouses. This new feature could act as a prevention measure to avoid losing yield due to pests.

Personal Development

This project has been my first solo development of a machine learning application from beginning to end. I had little knowledge when I first got started but through perseverance and the thoughtful guidance of my supervisor, Farshid Amirabdollahian, I was able to learn more than I could have envisioned when I first got started.

It has not been a bed of roses the making of the project with setbacks, failures and overall challenging tasks. This has pushed me to learn more and structure my investigation in a useful manner. I really have enjoyed the making of the project dedicating more hours than the instructed because of my interest in the topic. I now get involved in discussions with the community commenting on issues in github or machine learning subreddits that have been beneficial to me throughout the making of the project and I wanna give that piece back to the community. This project has reiterated the importance of open source code and the images, annotations and code for this project is publicly available in github.

Conclusion

The revolution sparked in 2012 with the introduction of deep learning has led to a level of maturity in computer vison that yields impressive results with robust representations of the categories for applications previously untouched.

The results obtained in this project with the application of deep learning for instance segmentation shows a strong potential for agricultural applications of machine learning. The initial aim for the project has been proved to be successful. Considering the small dataset created for the project the results look promising.

The report presents a novel approach for cucumber detection system employing an off-the-shelf HD webcam. We have reached scores superior to 0.90 mAP (IoU=50). Cucumbers are a challenging crop to detect due to the color , variable shapes, size and lightning conditions. Adaptations of the methodology can be used for agriculture in monitoring, harvesting or crop yield estimation applications.

References

1. Gupta, T., 2020. *Deep Learning: Overfitting*. [online] Medium. Available at: <https://towardsdatascience.com/deep-learning-overfitting-846bf5b35e24#:~:text=Overfitting%20refers%20to%20a%20model,the%20model%20on%20new%20data%20.>.
2. Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2016. *Understanding Deep Learning Requires Rethinking Generalization*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1611.03530>.
3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2014. *Going Deeper With Convolutions*. [online] Cv-foundation.org. Available at: <https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html.
4. Simonyan, K. and Zisserman, A., 2020. *Very Deep Convolutional Networks For Large-Scale Image Recognition*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1409.1556 >.
5. He, K., Zhang, X., Ren, S. and Sun, J., 2015. *Deep Residual Learning For Image Recognition*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1512.03385 >.
6. Girshick, R., 2020. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1311.2524.pdf >.
7. Uijlings, J., 2020. *Selective Search For Object Recognition | International Journal Of Computer Vision*. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1007/s11263-013-0620-5>.
8. Girshick, R., 2015. *Fast R-CNN*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1504.08083>.
9. Ren, S., He, K., Girshick, R. and Sun, J., 2016. *Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1506.01497 >.
10. He, K., Zhang, X., Ren, S. and Sun, J., 2015. *Deep Residual Learning For Image Recognition*. [online] arXiv.org. Available at: < https://arxiv.org/abs/1512.03385 >.
11. He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. *Mask R-CNN*. [online] arXiv.org. Available at: < https://arxiv.org/abs/1703.06870 >.
12. Floyd, J. (1965). The Effects of Farm Price Supports on the Returns to Land and Labor in Agriculture. *Journal of Political Economy, 73*(2), 148-158. Available at: <www.jstor.org/stable/1829530>.
13. Ruder, S., 2017. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1609.04747.pdf>.
14. Kingma, D. and Ba, J., 2014. *Adam: A Method For Stochastic Optimization*. [online] arXiv.org. Available at: < https://arxiv.org/abs/1412.6980 >.
15. Imgaug.readthedocs.io. 2020. *Imgaug — Imgaug 0.4.0 Documentation*. [online] Available at: < https://imgaug.readthedocs.io/en/latest/>.
16. Shirish Keskar, N., 2017. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1712.07628.pdf>.
17. GitHub. 2020. *Matterport/Mask_RCNN*. [online] Available at: <https://github.com/matterport/Mask_RCNN>.
18. Python.org. 2020. *Welcome To Python.Org*. [online] Available at: <https://www.python.org/>.
19. Lin, T., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. and Dollár, P., 2014. *Microsoft COCO: Common Objects In Context*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1405.0312>.

20. Team, K., 2020. *Keras: The Python Deep Learning API*. [online] Keras.io. Available at: < https://keras.io/>.
21. M. DUNN, G., 2004. *Yield Prediction From Digital Image Analysis: A Technique With Potential For Vineyard Assessments Prior To Harvest*. [online] Research Gate. Available at: <https://www.researchgate.net/publication/230419629_Yield_Prediction_from_Digital_Image_Analysis_A_Technique_with_Potential_for_Vineyard_Assessments_Prior_to_Harvest> .
22. TensorFlow. 2020. *Tensorflow*. [online] Available at: < https://www.tensorflow.org/>.
23. Khandelwal, R., 2020. *L1 L2 Regularization*. [online] Medium. Available at: < https://medium.com/datadriveninvestor/l1-l2-regularization-7f1b4fe948f2>.
24. Sa, I., 2016. *Deepfruits: A Fruit Detection System Using Deep Neural Networks*. [online] Available at: < https://www.researchgate.net/publication/305824563_DeepFruits_A_Fruit_Detection_System_Using_Deep_Neural_Networks> .

Table of figures

Figure 1, AlexNet architecture

Figure 2, Computer vision tasks

Figure 3, Residual Block

Figure 4, Pyramid Network

Figure 5, Anchors before NMS where hundreds of category independent region proposals are generated

Figure 6, Anchors and detections after NMS with labels

Figure 7, Transfer learning

Figure 8, Optimization Function

Figure 9, Intersection over Union

Figure 10, Annotation in VGG

Figure 11, Validation loss graph

Figure 12, Training loss graph

Figure 13, Loss values during training

Figure 14, Original image

Figure 15, Predicted cucumbers and masks

Figure 16, Original image

Figure 17, Predicted cucumbers and masks

Figure 18, Precision-recall graphs from previous two detections in order, left for figure 15 and right one for figure 17

Appendix

Data augmentation

These images are just examples of the transformations. During training the modifications are selected randomly and with different degrees of transformations to make the dataset more diverse.

Base Image:

Vertical Flip:

Horizontal Flip:

Gaussian Blur:

Linear Contrast:

Add:

Multiply:

Code

Model preparation

```
model = modellib.MaskRCNN(mode="training", config=config,
model_dir='/content/drive/My Drive/TFG/Submision')
model.load_weights('/content/drive/My Drive/TFG/Via/mask_rcnn_coco.h5',
by_name=True, exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",  "mrcnn_bbox",
"mrcnn_mask"])
def compile(self, learning_rate, momentum):
        """Gets the model ready for training. Adds losses, regularization, and
        metrics. Then calls the Keras compile() function.
        """
```

```
        # Optimizer object
        optimizer = keras.optimizers.SGD(
            lr=learning_rate, momentum=momentum,
            clipnorm=self.config.GRADIENT_CLIP_NORM)
        #optimizer = keras.optimizers.Adam(lr=learning_rate, amsgrad=True,
clipnorm=5.0)


        # Add Losses
        # First, clear previously set losses to avoid duplication
        self.keras_model._losses = []
        self.keras_model._per_input_losses = {}
        loss_names = [
            "rpn_class_loss",  "rpn_bbox_loss",
            "mrcnn_class_loss", "mrcnn_bbox_loss", "mrcnn_mask_loss"]
        for name in loss_names:
            layer = self.keras_model.get_layer(name)
            if layer.output in self.keras_model.losses:
                continue
            loss = (
                tf.reduce_mean(layer.output, keepdims=True)
                * self.config.LOSS_WEIGHTS.get(name, 1.))
            self.keras_model.add_loss(loss)

        # Add L2 Regularization
        # Skip gamma and beta weights of batch normalization layers.
        reg_losses = [
            keras.regularizers.l2(self.config.WEIGHT_DECAY)(w) /
tf.cast(tf.size(w), tf.float32)
            for w in self.keras_model.trainable_weights
            if 'gamma' not in w.name and 'beta' not in w.name]
        self.keras_model.add_loss(tf.add_n(reg_losses))

        # Compile
        self.keras_model.compile(
            optimizer=optimizer,
            loss=[None] * len(self.keras_model.outputs))
```

```
        # Add metrics for losses
        for name in loss_names:
            if name in self.keras_model.metrics_names:
                continue
            layer = self.keras_model.get_layer(name)
            self.keras_model.metrics_names.append(name)
            loss = (
                tf.reduce_mean(layer.output, keepdims=True)
                * self.config.LOSS_WEIGHTS.get(name, 1.))
            self.keras_model.metrics_tensors.append(loss)
```

Dataset class and preparation

```
from mrcnn import utils
class cucumberDataset(utils.Dataset):

    def load_cucumber(self, dataset_dir, subset):
        """Load a subset of the Balloon dataset.
        dataset_dir: Root directory of the dataset.
        subset: Subset to load: train or val
        """
        # Add classes. We have only one class to add.
        self.add_class("cucumber", 1, "cucumber")

        # Train or validation dataset?
        assert subset in ["train", "val"]
        dataset_dir = os.path.join(dataset_dir, subset)

        # Load annotations
        # VGG Image Annotator (up to version 1.6) saves each image in
the form:
        # { 'filename': '28503151_5b5b7ec140_b.jpg',
        #   'regions': {
        #       '0': {
        #           'region_attributes': {},
        #           'shape_attributes': {
        #               'all_points_x': [...],
        #               'all_points_y': [...],
        #               'name': 'polygon'}},
```

```
        #       ... more regions ...
        #   },
        #   'size': 100202
        # }
        # We mostly care about the x and y coordinates of each region
        # Note: In VIA 2.0, regions was changed from a dict to a list.
        annotations = json.load(open(os.path.join(dataset_dir,
"via_region_data.json")))
        annotations = list(annotations.values())  # don't need the
dict keys

        # The VIA tool saves images in the JSON even if they don't
have any
        # annotations. Skip unannotated images.
        annotations = [a for a in annotations if a['regions']]

        # Add images
        for a in annotations:
            # Get the x, y coordinaets of points of the polygons that
make up
            # the outline of each object instance. These are stores in
the
            # shape_attributes (see json format above)
            # The if condition is needed to support VIA versions 1.x
and 2.x.
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in
a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in
a['regions']]

            # load_mask() needs the image size to convert polygons to
masks.
            # Unfortunately, VIA doesn't include it in JSON, so we
must read
            # the image. This is only managable since the dataset is
tiny.
```

```
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "cucumber",
                image_id=a['filename'],  # use file name as a unique
image id
                path=image_path,
                width=width, height=height,
                polygons=polygons)

    def load_mask(self, image_id):
        """Generate instance masks for an image.
       Returns:
        masks: A bool array of shape [height, width, instance count]
with
            one mask per instance.
        class_ids: a 1D array of class IDs of the instance masks.
        """
        # If not a balloon dataset image, delegate to parent class.
        image_info = self.image_info[image_id]
        if image_info["source"] != "cucumber":
            return super(self.__class__, self).load_mask(image_id)

        # Convert polygons to a bitmap mask of shape
        # [height, width, instance_count]
        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"],
len(info["polygons"])],
                        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
            # Get indexes of pixels inside the polygon and set them to
1
            rr, cc = skimage.draw.polygon(p['all_points_y'],
p['all_points_x'])
            mask[rr, cc, i] = 1
```

```
        # Return mask, and array of class IDs of each instance. Since
we have
        # one class ID only, we return an array of 1s
        return mask.astype(np.bool), np.ones([mask.shape[-1]],
dtype=np.int32)

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "cucumber":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)
import os
import sys
import json
import datetime
import numpy as np
import skimage.draw
# Training dataset.
dataset_train = cucumberDataset()
dataset_train.load_cucumber(dataset_dir, "train")
dataset_train.prepare()

# Validation dataset
dataset_val = cucumberDataset()
dataset_val.load_cucumber(dataset_dir, "val")
dataset_val.prepare()
```

Data augmentation

```
from imgaug import augmenters as iaa
aug = iaa.OneOf([
    iaa.Add((-40, 40)),
```

```
    iaa.Multiply((0.8, 1.5)),
    iaa.Fliplr(0.5),
    iaa.Flipud(0.5)
])
```

Training

```
import time
start = time.time()
model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE*2,
                epochs=30,
                layers='heads',
                augmentation = aug1)
model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=60,
                layers='4+',
                augmentation = aug1)
model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE/5,
                epochs=90,
                layers='all',
                augmentation = aug1)
print('Elapsed time', round((time.time() - start)/60, 1), 'minutes')
```

Evaluation mAP

```
# Compute VOC-style Average Precision
def compute_batch_ap(image_ids):
    APs = []
    for image_id in image_ids:
        # Load image
        image, image_meta, gt_class_id, gt_bbox, gt_mask =\
            modellib.load_image_gt(dataset, config,
                                   image_id, use_mini_mask=False)
        # Run object detection
        results = model.detect([image], verbose=0)
        # Compute AP
```

```
        r = results[0]
        AP, precisions, recalls, overlaps =\
            utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
                             r['rois'], r['class_ids'], r['scores'], r['masks'])
        APs.append(AP)
    return APs

# Pick a set of random images
image_ids = np.random.choice(dataset.image_ids, 53)
APs = compute_batch_ap(image_ids)
print("mAP @ IoU=50: ", np.mean(APs))
```