ADDITIONAL INTERNSHIP APPLIED MATHEMATICS

Certificate Detection and Recognition

Name	Britt van Veggel
Student No.	5173590
Supervisors	Willem van Engen (Questionmark) Chiel Fernhout (Vantage AI) Lieke Kools (Vantage AI)
Start date	August 3, 2020
Email	b.w.j.r.vanveggel@student.tudelft.nl
Date	February 6, 2021

Contents

Introduction

The field of machine learning is growing everyday, not only in academia, but also within companies. As a lot of mathematicians end up working within the field of data science, I was curious to see what kind of projects data scientists works on. While the data scientist's toolbox contains many sophisticated techniques, I preferred to have a look within the corner of deep learning, because this field is very trendy and has cool applications.

I wanted to combine this eagerness to learn about deep learning with my idealistic nature. So I set out to find an NGO that could very much use some extra hands on this matter. I heard about an NGO called Questionmark while watching one of my favorite shows, *De Keuringsdienst van Waarde*. This NGO strives to collect any available information on the food products in the product range of the big, Dutch supermarkets and analyses these products to draw conclusions on the state of the Dutch food industry. The focus is mostly laid on sustainability, health and human rights. This seemed like an inspiring place to contribute some data science to.

I sent them an email and quickly received a reply that I was more than welcome to drop by in the near future. They had multiple projects at hand, that all seemed very interesting. But the project that appealed to me most was about detecting and recognising certificates on images of food packaging. Both as it could be used to improve the data quality of the Questionmark database. On a more personal level, the theory behind what I would be learning is widely used and achievable within the set time-frame of two months given I knew nothing about neural networks at the time.

As Questionmark did not have the capacity to supervise me to the extend that I would need to complete this project, I asked the data science consultancy Vantage AI if they would like to accompany me in this project. I coincidentally knew they were doing multiple projects on image recognition, so they would be the ideal mentors. Vantage AI was very excited to help me out with this.

The purpose of this essay is to explain the theory I studied for this project and procedure of my model to someone wanting to either continue with this project or start a comparable project up themselves. Therefore, it also contains a very detailed documentation and recommendations on further improvement. Besides from that, this report is used to justify how I've spend my time while at Questionmark, and how I reflect on this project with regards to my future carreer.

In chapter 1, Theory, I will explain what a Convolutional Neural Network is, how it works, basic algorithmic implementations of CNNs and any additional theory that is needed in order to work with CNNs. In chapter 2, approach, I will explain how I approached learning about this theory, course of the development process. Then, chapter 3 contains a broad analysis of the results. It also includes some recomendations for further development. Furthermore, chapter 4, Documentation, contains the entire documentation of all the scripts created for this project. Lastly, chapter 5 contains my personal experience and reflection on this project.

1. Theory

In this chapter, the theory behind the project will be explained. This is necessary because when another student will follow up on this project, there will be one document in which to find the underlying theory. Besides from that, studying and understanding all this theory was part of the project, thus should at least slightly be covered in the final report.

The theory discussed will be on Convolutional Neural Networks, also called ConvNets or CNNs. A CNN is a form of neural network, in which convolutions are used. What these convolutions are, will be explained in the first section (1.1), later on Neural Network will discussed (1.2). Next, we will speak about an implementation of a CNN, named the YOLO Algorithm (1.3). And lastly, techniques to optimize and evaluate the training process will be discussed. This will be a data split method (1.4), Data Augmentation (1.5) and Confusion Matrices (1.6).

1.1 Convolutions

Within a computer an image is represented as a matrix. A simple example of this can be seen in Figure 1.1. In this picture we see a smiley-face with light eyes. This is converted to a matrix, where we can see that the white cells have a value of 0 in the matrix on the corresponding cell, while the black cells have the value of 1 and the grey cells have a value of 0.5.

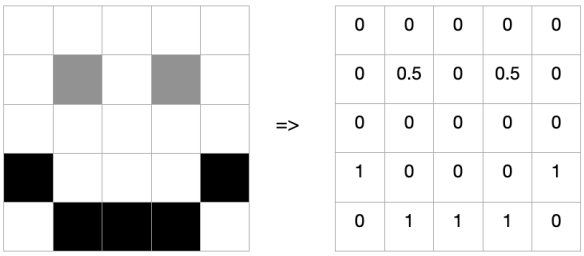


Figure 1.1: Picture of a smiley-face converted to a matrix

But, how does this work with colored images? A colored image can be split into a red part, a blue part and a green part, as is simply shown in Figure 1.2. Here, it is clear that as each picture represents the intensity of that color, and thus can be seen as a grayscale picture, as it just on a light and dark spectrum.

These three grayscale pictures can be translated to matrices, and combined. This set of multiple matrices is called a *tensor*. So for a picture of n by m pixels, we can convert it to an $n \times m \times 3$ tensor.

Now that we have translated a picture to a tensor, we can apply a convolution to it. But, what is a convolution? A convolution is an operation in which a matrix is applied to another (generally

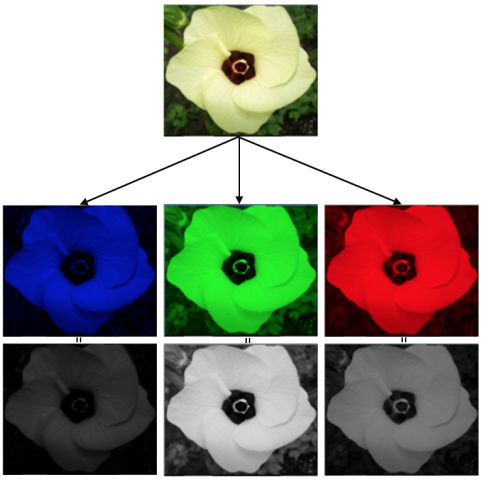


Figure 1.2: Splitting a colored picture of a flower into 3 RBG pictures

larger) matrix by component-wise multiplying all the values of the matrix to the submatrices of the (generally larger) matrix, adding those values together and assigning it to the corresponding cell in the resulting matrix. It can best be described visually. So, we will look at an example of a convolution that detects a change in a picture from light to dark to the 'picture' seen in Figure 1.3, and will be refered to as matrix 1.3.

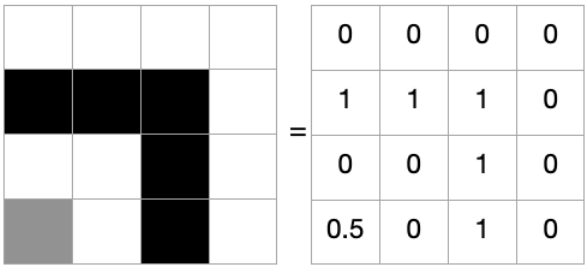


Figure 1.3: Matrix 1.3: The picture to which we will apply a convolution detecting a change from ligth to dark

The 'smaller' matrix we will use for our convolution will be the one seen in Figure 1.4 and will be refered to as matrix 1.4.

When we apply matrix 1.4 to matrix 1.3, we component-wise multiply it to all the 2×2 submatrices of matrix 1.3. This will look as in Figure 1.5. In this picture, we can see how the red 2×2-matrix is componentwise multiplied to matrix 1.4 and these values added up together, to become the value of 1,1 coordinate of the resulting (red) matrix. The (yellow) 1,2 component of the resulting matrix is formed by the yellow 2×2-matrix at [1:2, 2:3] which is componentwise multiplied with matrix 1.4. But, what does this matrix tell us? It tells us that higher values in the resulting matrix indicate that there is a stronger transition from low values to high values near. We call these values *activations*, so we'd say, the first component has a high activation. In the case of red and yellow, we see a clear transition from low values to high, and this translates to an high activation of value 2 on the corresponding matrix entries in the resulting matrix. For the blue matrix, we see that the transition is to a low activation, and only on one component, resulting is a relatively low value.

0	0
1	1

Figure 1.4: Matrix 1.4: The matrix we will apply to the matrix seen in Figure 1.3 by means of the convolutional operation.

Figure 1.5: Applying matrix 1.4 as a convolution to matrix 1.3.

Now, this is still very abstract, but in Figure 1.6 we can see more clearly how different convolutions activate when applied to a picture of a cat. We can see in Figure 1.6A, we see that horizontal lines are activated, whereas on Figure 1.6D, vertical lines are activated. There are many more convolutions applied, and the activation in the resulting matrix is varies a lot.

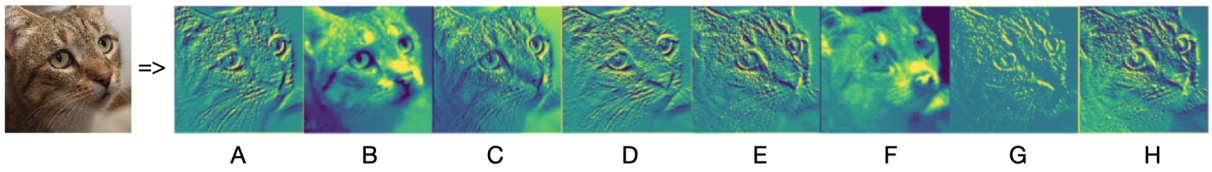


Figure 1.6: Activation of different convolutions applied to a picture of a cat

Now, the cat pictures (or corresponding matrices) will still not make it clear to a computer what is a cat or what is not. Therefore, if we continue to apply differently activating convolutions to a matrix, we can find different patterns. For example, if we apply a convolution that detects (or activates) vertical lines to a matrix, and then one that detects (or activates) horizontal lines, we can combine that information to detect (or activate) a corner. And if we've found four properly connected corners, we've found a square.

By applying multiple convolutions to a picture subsequentially, all information of a photo will eventually be extracted. This information can be used to conclude what objects are present in a picture. When detecting objects in pictures many different approaches can be used. The sequence and architecture in which convolutions are applied to a picture can be called a model. We will go deep into models in section 1.2 about neural networks.

There are many 'nuances' that can be used when applying convolutions, without getting too technical, we will explain the most prominent ones in the next few subsections.

1.1.1 Strides

As we have seen in Figure 1.5, the matrix 1.4 is applied to *every* submatrix of matrix 1.3. But, we can also skip a few submatrices in order to decrease computational time. The amount of submatrices skipped is called the *number of strides*. In the example shown in Figure 1.5, the number of stides is 1×1, because the next submatrix used is only 1 row apart from the first row of the former submatrix and the same goes for columns.

In generally if we speak of a stride of $n \times m$, the submatrices are n rows and m columns apart from eachother. In Figure 1.7 we will exemplify strides of different sizes for matrix 1.3. Here, the colored squares incidate submatrices that are used for this particular amount of strides, and the dotted squares are matrices that would be used if the matrix had been of a larger size. We can see again that in the 1×1 case, all submatices are used, while in the 1×2 strides case, the submatrix on every row is included but skippes every other column.

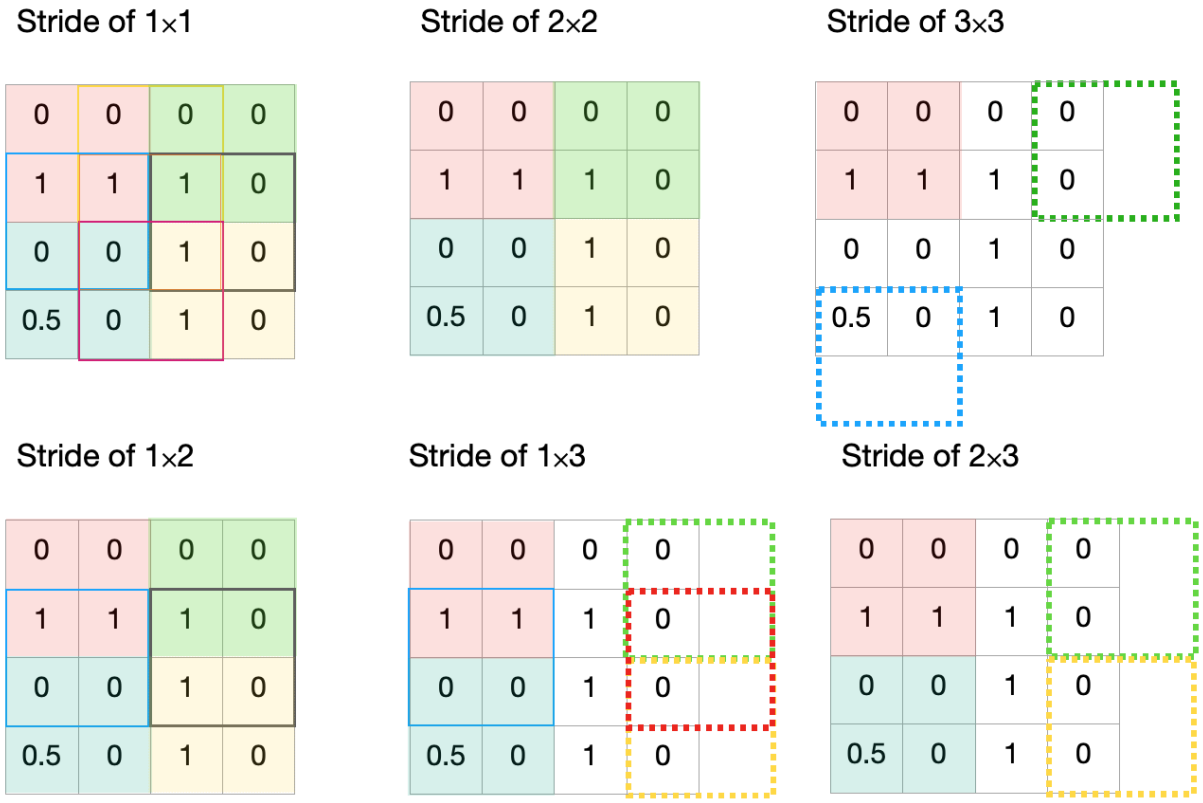


Figure 1.7: Three types of strides applied to the 4×4-matrix 1.3. The colored squares and full-lined squares are submatrices that are used for this particular amount of strides, and the dotted squares are matrices that would be used if the matrix had been of a larger size.

1.1.2 Pooling

Another way to decrease computational time, is to decrease the size of the matrix we are applying a convolution to. Doing so would be called *pooling* or adding a *pooling layer*. The idea of a pooling layer is to represent submatrices by one value in a smaller matrix. This value is attained by applying some kind of function to the matrix, most commonly the maximum value of the values in the submatrix or the average of the values. In Figure 1.8 we see an example of applying max- and average pooling of a stride 2×2 applied to matrix 1.3.

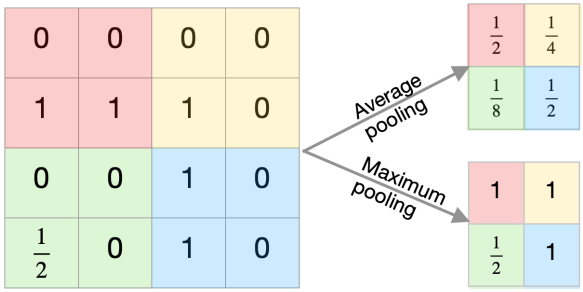


Figure 1.8: Results of applying average and maximum pooling with stride 2×2 to matrix 1.3.

1.1.3 Padding

There are two classical drawbacks to applying convolutions. The first is that the resulting matrix is a different size from the original matrix to which the convolution was being applied. The other one is that information in the border of the matrix is picked up in less convolutions than information in the middle. The latter might need some explanation. In Figure 1.5, we can see that the 0-value in the top left cell is only used in the calculation of the red cell in the resulting matrix, while the cell in the second row and second column is used for the calculation of the red, yellow and green cell. Therefore, it weighs much heavier in the resulting matrix that the top left 0-value. The prominent solution introduced to tackle this problem is *padding* (or a *padding layer*). Padding is when a border of size p (p new top and p new bottom rows and p columns on each side) is added around a matrix. Most commonly, the values in these borders are 0, but in some applications the values can be equal to the value of the cell most adjacent cell. An example of adding $p = 0$, $p = 1$ and $p = 2$ to matrix 1.3 is shown in Figure 1.9.

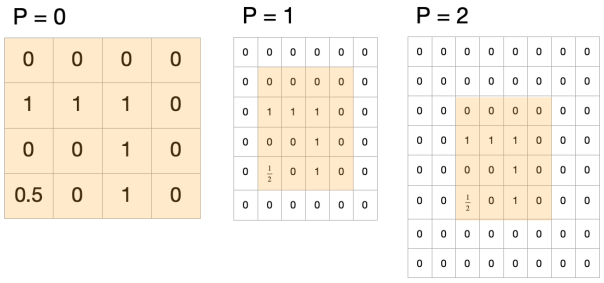


Figure 1.9: Padding added to matrix 1.3 of sizes $p = 0$, $p = 1$ and $p = 2$.

1.1.4 Sequential Model

As loosely stated before, different convolutions can follow one and other up in order to break down all the information in an image. The order and manner in which different types of convolutions, pooling layers, paddings and strides are applied are called the sequential model or the architecture (there are more types of layers, but there are too many to state. The ones listed above are the main ones). By applying these layers in a certain order to a photo, we can break down the components of a picture. The first few layers indicate small components of the input picture, such as color transitions, horizontal lines, vertical lines, etc. Going deeper into the model, we combine this information to find things such as corners, circles and other simple shapes. Eventually, going much deeper into the model all this information is combined to find a pattern indicating, for example, a certain logo, a dog or a bicycle. How this indication is made exactly is done by (artificial) neural networks. We will get into this in section 1.2.

But first, in order to build some intuition, a visualisation of such a sequential model is given in Figure 1.10. Here we see how an input picture of (for example) 360×220 pixels is shaped down to an 100 dimensional vector (or to put it accurately: a $1 \times 1 \times 100$ tensor) using convolutional layers, pooling layers, padding layers and other types of more specific layers that we will not really get in to.

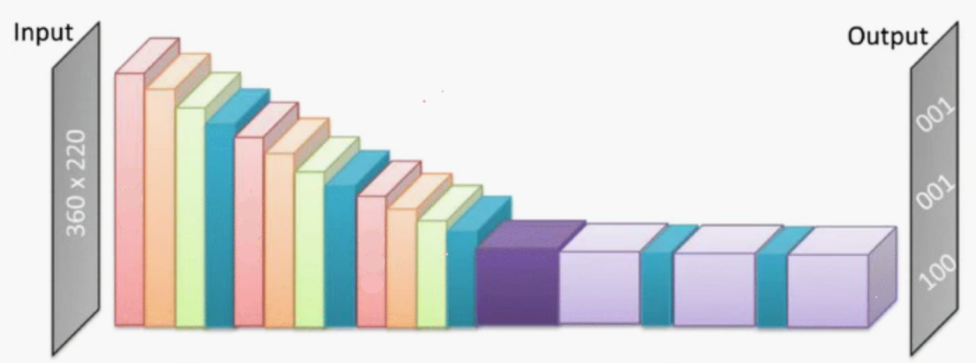


Figure 1.10: An example of a sequential convolutional model

Generally, when working with CNNs, the architecture is not created from scratch. There are famous architectures that are commonly used and slightly altered if necessary. Some of the most prominent ones are AlexNet, VGGNet, GoogLeNet and ResNet. Now, how these architectures train on the proveded dataset is done using an artificial neural network, which will be discussed in the next section.

1.2 Artificial Neural Network

An artificial neural network is a form of machine learning that falls within the deep learning spectrum. Machine learning is a field in which a software tries to improve its own performance, and deep learning is a type of machine learning which is more structured and hierarchical, where multiple layers are applied in order to improve the understanding of the algorithm. An artificial neural network is loosely inspired off of a biological neural network in the brain, where different sections of the brain send information to each other in order to process information.

A neural network is easiest understood when looking at Figure 1.11. As can be seen there, a neural networks exists of layers (the yellow dots form one layer, the blue dots form another layer, and so on) and connections between these layers. The idea of a neural network is that, when given a certain input, it applies a few calculations (convolutions or such) to it and classifies the input in order to see what calculations to apply to it in the next layer. All this is done in order to eventually be able to assign it to a category.

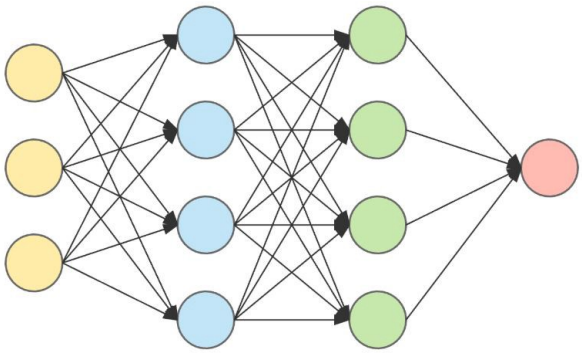


Figure 1.11: Simple visualisation of a neural network

These nodes assign certain weights to certain inputs. This is done to make sure certain information influences the outcome heavily, while other information does not have a heavy impact (for example, when wanting to classify a bike, one does not need to take the color of the frame too much into consideration). When the network is provided data with proper labels, the neural network can adjust these weights when comparing this data in order to make proper predictions for future data. It knows how to adjust these weights by looking at the loss function. This will be more widely explained in subsection 1.2.1. But the network adjusts the weights of the nodes in such a matter that this loss function in minimized (most commonly using the gradient descent algorithm). This process of processing data and adjusting the neural network in order to make proper predictions to future input data is called *training*.

1.2.1 Loss function

The loss function is a function that measures the different errors. These different errors can be things like wrong predicted class, false positives or negatives, misplaced bounding box, low probability for a correct prediction, etc. These different errors are measured using the MSE and then added together with different weights for different types of errors. These weights (mostly reffered to as λ's) can be altered, and thus the overall loss can be adjusted for the specific use the neural network is build for.

The loss function, as stated before, decides how the neural network trains itself. It aims to minimize this loss function. The loss function (and what part corresponds to what error) used in our model is portrayed in Figure 1.12.

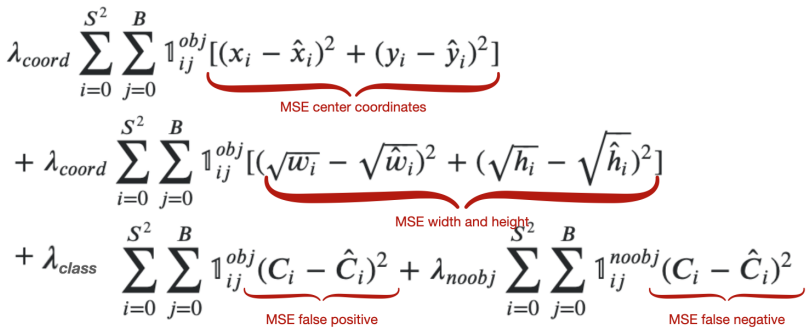


Figure 1.12: The loss function with indication of which error is measured

1.2.2 Transfer Learning

Training a neural network needs a lot of computational power. As this takes time and needs very strong GPUs. Therefore, one uses a neural network created and trained for/on another large dataset, and applies that infrastructure to a smaller different dataset and adjusts it slightly. So for example, a certain CNN can be trained for detecting vehicles. In order to do this is needs to be able to recognise squares, circles, etc. But this is also needed for recognising certificates. So we keep this part of the training, but alter the last few layers of the neural network in order to recognise certificates rather than vehicles. Therefore, one needs less training but does have some basics of image recognition and detection.

1.3 YOLO

The YOLO algorithm is one of the fastest image detection and recognition algorithms around. It is quite difficult to understand, so the explanation will be kept simple and intuitive.

YOLO splits a convolutionally reformed image into a $n \times n$ grid. Then, each gridcell predicts K bounding boxes. Each bounding box has coordinates (in the shape of center coordinates, and height and width), the predicted class and a probability of accuracy.

An object is considered to lie in a specific cell if the center coordinates of the anchor box lie in that cell. Because of this, the center coordinates are always calculated relative to the cell, yet the height and width are calculated relative to the whole image size.

An example can be seen in Figure 1.13. It is important to note that in this figure it looks like the CNN makes the predictions on the gridcell in the image itself, but actually each gridcell contains a lot of information from the neural network, and thus contains more information than, for example, only a car light.

Because a lot of predictions are being made for every gridcell around the same object, some slightly different bounding boxes may do a prediction on the same object in the image. A visual example of this can be seen in Figure 1.14. The solution for this lies in applying non-maximal suppression to those detection. NMS uses intersection over union. In the next subsections, these concepts will be explained.

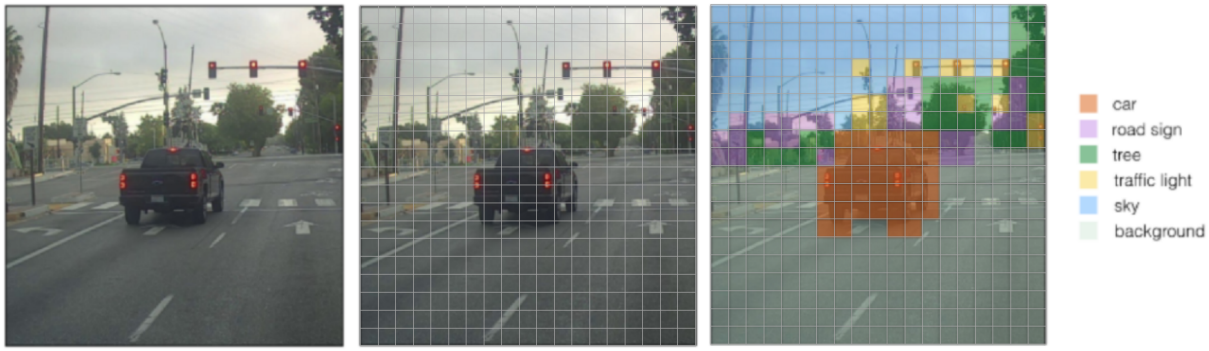


Figure 1.13: An example of a YOLO algorithm applied to a picture. The first picture is the original, the second shows the 19×19 grid and the third picture shows the prediction made for each gridcell.

Figure 1.14: Predicted bounding boxes before applying non-maximum suppression

1.3.1 Non-Maximum Suppression (NMS) and Intersection over Union (IoU)

Because multiple bounding boxes are being predicted that could refer to the same object in the image, we need to suppress the unimportant bounding boxes. In Figure 1.14, we see an example of multiple bounding boxes reffering to the same thing. For example, the yellow bounding boxes correspond to the traffic lights. There are many, but there is only one that perfectly gridles around the traffic light. In order to select this perfect bounding box, we would need to suppress the overlapping, less accurate boxes.

The way to establish if two bounding boxes refer to the same object in the image, can be done using *intersection over union (IoU)*. This can be calculated using the ratio between the overlap between to bounding boxes, and the union of area they cover together. This is quickly understood intuitively when looking at Figure 1.15. Once the IoU between two boxes surpasses a certain threshold, it can be established that they refer to the same object in the image.

Once we have established that two bounding boxes refer to the same object (by have an IoU larger than a certain threshold), we need to find out which prediction is better. This can be easily done using the probability indication that comes along with the bounding box. So the prediction with the highest probability is kept, while the bounding box with the lower probability is dropped. This concept is called *non-maximum suppression*, as the non-maximum probability bounding box is suppressed. A visualization of this can be seen in Figure 1.16.

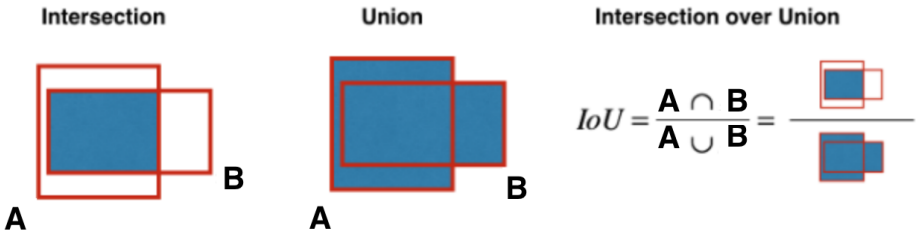


Figure 1.15: How to calculate the IoU of two bounding boxes

Figure 1.16: An example of non-maximum supression.

1.4 Train - Test - Validation Split

When a neural network trains, it is shaped to predict optimally on the provided dataset. But, in order to avoid overfitting on the provided data, the neural network also checks if the model fits on a separate dataset, that is not used to train on. Therefore, one splits the data into a train set and a validation set. The validation set is of course much smaller than the train dataset. A third set, named the test set, is also introduced. This set is used to evaluate the model on while training.

There are many ways to split this set, but the division is generally train 64% (= 80% of 80%), test 20% and validation 16% (= 20% of 80%) of the overall data.

1.5 Data Augmentation

Data augmentation is a technique used to expand the dataset by adding slightly altered copies of datapoints to the dataset. This can be done in the dataset, but also *on the fly*. Data augmentation on the fly means that, while training, the model loads a picture, makes a copy of it and augments that copy, trains on the augmented copy and then deletes the copy, leaving the original untouched.

There are many ways to augment a picture. Most augmentation is done by changing the coloring settings of a picture, adding slight noise to the picture and by adjusting the shape slightly. Some examples are shown in Figure 1.17.

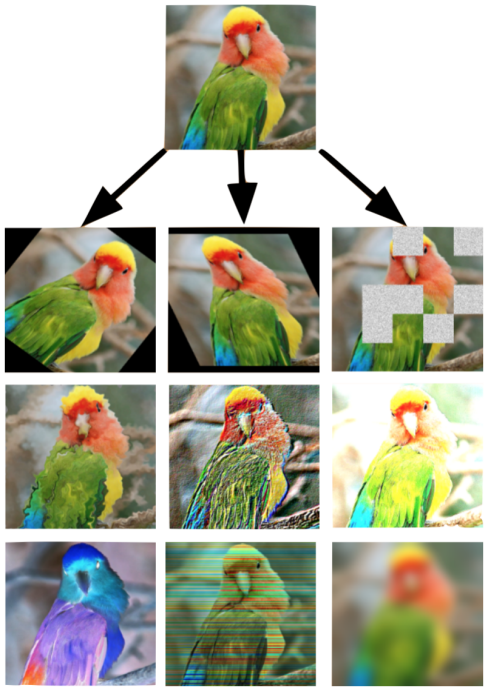


Figure 1.17: Some examples of possible data augmentation steps applied to a parrot.

1.6 Confusion Matrices

In order to evaluate en visualize the performance of a neural network, a confusion matrix is used. A confusion matrix is a squared table that displays how often certain predictions were made, and what the actual label of this object in the image was. This way, one can see how different labels are being mixed up. Each row and each column correspond to a certain label that can be detected. But, the rows display the actual annotated label, and the columns display to the label predicted by the neural network.

The aim is to predict the right label. Therefore, it is most desirable to have high scores on the diagonal, and the lowest possible score on the off diagonal. An example can be found in Figure 1.18. In this confusion matrix, one can see that out of the 58 pictures of cats, 57 were predicted as cat and 1 picture was predicted to be a cow. Out of the 69 pictures of cows, 66 were predicted properly, one was predicted o be a dog and 2 were not recognised to be animals at all.

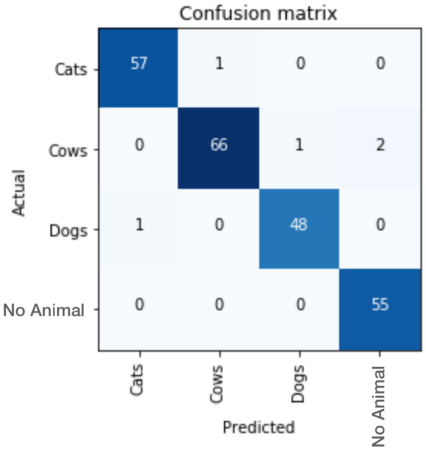


Figure 1.18: An example of a model trying to detect cats, dogs and cows or the absence there of in picture.

2. Approach

In this section I will explain how I proceeded to complete this project. First, the approach to learn the basics of the theory will be explained. After that I will chronologically explain the process.

2.1 Sources Used

As the field of neural networks was entirely new for me, I needed to start with reading up on this field. I started off with the Coursera course 'Convolutional Neural Networks' by Andrew NG. This course included multiple video's about subtopics, accompanied with a test and a programming exercises. After this I continued watching multiple lectures of the Stanford YouTube lectures of the course Convolutional Neural Networks for Visual Recognition. Furthermore, I watched many youtube videos on the YOLO alrogithm, because I found the way it worked very vague and difficult. Sadly, I cannot recall all videos, but the youtube channels I mostly used were: carykh, 3Blue1Brown, DeepLizard and Siraj Raval.

Besides from that, I did multiple basic tutorials. The main tutorials I have completed are:

- Andrew NG's Coursera Course *Convolutional Neural Networks*;
- MachineLearningMastery.com's *Mini Deep Learning for Computer Vision Crash Course*;
- A three day CoLab tutorial from Vantage AI (about multiple classes of Flowers, Cats vs. Dogs and Transfer Learning);

2.2 Development Process

In this section I will talk elaborately about the development process of the CNN in chronological order. The procces started off by getting the Xomnia implementation running, and from here it was clear the dataset needed to be expanded. I will dig into my search for the right way to train the network. Next, I will spent some time on some evaluation tools I've build and on the parameters I have optimized.

2.2.1 Getting the Xomnia implementation running and evaluating the first results

After I'd spent a while studying the above mentioned theory and getting used to Questionmarks dataset, I was ready to get started on the neural network. A while before I started at Questionmark, a few trainee's from Xomnia had attempted to make a simple image recognition algorithm

using ImageAI within a day. This script was no longer than 6 lines of code. I was going to expand this simple code to a working code for which the performance can be easily quantified.

The first step was finding out how to get the trainer running with the data that had been provided. It took quite a few days to get this running as I needed to get accustom to the API of ImageAI and understand what structure and set up were needed for the mounted directory. Also, I was provided two (annotated) datasets. One was a mix of webshop-esque pictures and photos taken in real life, the other set was a set of only photos taken in real life. Because we wanted the dataset to reflect the picture we would use (pictures from webshops), we knew we would not use the IRL dataset. But because we could, we would train once with and once without that dataset.

Once the model was be able to train, it was important to understand how to analyze the performance of the trained model. My supervisor Chiel advised me to search for an evaluation function in the ImageAI documentation and implement it, and so I did. When running this, it turns out that the model wasn't as well trained as I thought. The results shown in Listing 2.1 and Listing 2.2. Most important thing to analyze here is the mAP. This is the average precision looks at how many true positives have occurred for each label relative to the overall amount of positives (true and false). Therefore, it measures how often the model recognises a certificate properly. But, it does not say anything about the false negatives.

```
Using IoU : 0.5
Using Object Threshold: 0.3
Using Non-Max Suppres.: 0.5
asc: 0.0000
beterleven: 0.0000
bio: 0.0667
ebio: 0.0000
eko: 0.2857
fairtrade: 0.1956
msc: 0.0000
organic: 0.2885
planetproof: 0.6985
utz: 0.1756
vegan: 0.0000
weidemelk: 0.0000
mAP: 0.1425
```

Listing 2.1: Results of first run on 11 September without DIY data. The values after the labels are the average precision of that label.

```
Using IoU : 0.5
Using Object Threshold: 0.3
Using Non-Max Suppres.: 0.5
asc: 0.0033
beterleven: 0.0000
bio: 0.8000
ebio: 0.0000
eko: 0.1837
fairtrade: 0.5032
msc: 0.0000
organic: 0.2956
planetproof: 0.5851
utz: 0.2292
vegan: 0.2857
weidemelk: 0.0000
mAP: 0.2405
```

Listing 2.2: Results of first run on 11 September with DIY data. The values after the labels are the average precision of that label.

2.2.2 Analyzing and improving the dataset

It was clear that the first results were not even nearly useful. It is not due to the algorithm, as this is a proper implementation, therefore it was clear it would be cause by our data. The data needed to look like pictures in a webshop, some examples of which are displayed in Figure 2.1. The most commonly shared features are

- monotonous background color (most commonly white);
- no angle in the packaging;
- colorful packaging;
- packaging in center of the picture.

Of course, there are exceptions, and these needed to be slightly included as well, but these are the most commonly occurring features.

(a) Pack of butter with Beterleven - 3 sterren certificate and European Biologic certificate

(b) Milk carton with PlanetProof certificate and Weidemelk certificate

(c) Canned fish with MSC certificate

Figure 2.1: Examples of what pictures from webshops look like.

From a quick glance, it is clear that this data is no where near where it needs to be: a big portion of the data does not represent the type of pictures we want the algorithm to be able to predict on in the future. In Figure 2.2 some examples can be seen alongside an explanation of why these pictures are not useful.

So, it was very clear a more fitting dataset needed to be acquired and annotated. The data of Questionmark is stored in Metabase. I wrote a query with Willem that would create a CSV file of all the picture URLs for a certain certificate. Then I wrote a scraper for these images, `Scraping.py` (section 4.1).

LabelImg was used to annotate the scraped data, this is a program that allow the user create an annotated xml file by drawing and labeling bounding boxes on an image.

In a meeting with Chiel and Lieke from Vantage AI, we discussed what a useful amount of data would be. Because labelling data can take a long time and there is more to gain (and learn) in tweaking parameters rather than adding more data, it would be smartest to start off with somewhere between 60 and 100 images per label (so about 1200 images in total). If the results were disappointing, more could be added. Because some of the data would be faulty, we could not assume that if we scraped 100 images, we would attain 100 images with said label. Thus we had a margin. I spent a few days labelling data but I acquired a beautiful dataset of about 1346 items.

2.2.3 Distribution of the data

Next, it was important to randomly split the data into the 3 needed directories: Train, Test and Validation. Therefore I made a script that randomly splits the data in subsets containing 64%,

(a) Pictures taken of multiple packages in real life

(b) Pictures of just certificates (also ones we do not want to detect) on an arbitrary background

(c) Way too many unrealistic certificates at unrealistic places on a drawn package.

Figure 2.2: Examples of faulty data

20% and 16% of all images respectively and make an overview of the exact distribution per label. This script would also save the distribution such that a specific distribution can easily be assigned again. The biggest challenge I learned to deal with was working with ElementTree to analyze my annotated xml files.

We also implemented it in such a way that the entire dataset would copy itself and then be assigned to different directories. This was done so that we could alter this data for the training, without entirely changing the data if the results were not optimal.

Besides from that, it is very important to verify that nothing had gone wrong while splitting and assigning the datasets. Therefore I also included a few verification scripts:

- `check_missing_files` : to see if there are any files in either images or annotations that are not in the other directory;
- a path listing within `write_split_to_dir` to see if the clearance of the images and annotations file has succeeded;
- `certificates_in_dir` : counts per subdirectory how many images per certificate are present, to see if the certificates are equally distributed;
- `visualise_data_for_label` : to see how the data looks and how the bounding boxes are interpreted.

The first split was distributed as seen in Table 2.1.

2.2.4 Training on the new data

The model was now ready to train again. As Google CoLab offers a free NVIDIA-GPU with limited usage, it was a logical place to start. The batch size could not be larger than 4, as a larger batch size would overuse the RAM memory. After a bit of research, it was also clear that the

certificate	Alles	train	test	validation
asc	68	41	16	11
beterleven1	90	55	22	13
beterleven2	66	42	17	7
beterleven3	72	52	13	7
bio	52	35	11	6
demeter	97	69	17	11
eko	89	65	9	15
fairtrade	164	110	29	25
msc	89	59	19	11
organic	242	163	40	39
planetproof	65	39	16	10
rfalliance	38	28	5	5
utz	78	51	16	11
vegan	43	27	9	7
weidemelk	93	60	22	11
total	1346	896	261	187

Table 2.1: Split of the dataset at 5okt.

`batch_size`	4
`num_experiments`	200

Table 2.2: First few settings

GPU could be used for about 12 hours straight. Therefore, this was a nice place to start. The first few settings were as seen in Table 2.2.

At first there were problems to get the model running. This was apparently due to two factors that were discovered after quite a bit of pottering:

- deleting the cache files is essential in order to get ImageAI to train
- all images should be `.jpg`, and the dataset contained `.jpeg` files. To solve this, I build `jpeg_to_jpg_xml` to change the images to `.jpg` and also incorporate this in the annotated xml files.

The model was now ready to train. The run was not entirely completed, as it stopped after 94 epochs rather than the primarly set 200 epochs. This was due to the GPU limitations of CoLab. But, because it only saved the models that had a loss improvement relative to the last saved model, the last model saved was at 71 epochs.

Because we did not need a lot of accuracy on where the certificates were located, just that they were spotted somewhere, the IoU was set to 0.2. The model evaluation resulted in the results seen in Listing 2.3, which were clearly a great improvement compared to the results of Listing 2.1. In hindsight, it turned out that the predictions were set to be made on the validation set (on which it kind of trained trained) rather than the test set, which it had never seen. Therefore, they are very high and should be lower. It was changed quickly after, so it does not affect the scores too much, but this can especially explain why the Beter Leven certificates scored so high and never reached this level ever again later on.

```
Using IoU :  0.2
Using Object Threshold :  0.3
Using NMS :  0.5
asc:            0.7256
beterleven1:    0.8749
beterleven2:    0.8571
beterleven3:    1.0000
bio:            0.9769
demeter:        1.0000
eko:            0.7023
fairtrade:      0.5855
msc:            0.8781
organic:        0.9481
planetproof:    0.9455
rfalliance:     0.8333
utz:            0.8894
mAP:            0.8284
```

Listing 2.3: Results of first model run with complete dataset (02-10-2020).

2.2.5 Finding the right computer to train on

These results were very pleasing, but it was not entirely clear if these were most optimal, provided that the trainer crashed. After a bit of research it was clear that CoLab had some limitations to their free GPU usage for our project. There were a lot of options to explore. They are listed in the order they were attempted with pro's and con's in Table 2.3.

In the upcoming few weeks, I spent my time trying to work out what the most optimal way was to properly train the network. Because this included a lot of waiting while the network was training, I spend my time building some evaluation tools (which we will get into in the next section), processing data for Questionmark, getting in to more theory and working some of it out for this report.

In hindsight, this plan should have been worked out earlier, because it costed so much time to figure this out without knowing what the normal approach was. It also took a lot of time getting used to AWS, even though I still would not be able to fully operate this system myself.

2.2.6 Building additional evaluation tools

While trail-and-erroring my way through possible training options, I worked on methods to evaluate the different models, which could of course also be used to asses what computer is best to train on. It would be nice to see exactly how each label would perform, both in true positives, false positives and false negatives. This information can be visualised in a confusion matrix (as explained in section 1.6). In order to measure this, I implemented something that would log all needed data for a confusion matrix within the predicting script that would keep count of what type of positive or negative the prediction made is. This was very interesting to do, as I would need to work out things like

Approach	Pro's	Con's
Repeatedly starting a new training on last model from previous training in CoLab	- Free - No need to alter script	- Would need to wait for GPU usage to restore, which was very unpredictable. - As the last successful model was saved, it was never clear at what epoch the model was training. - CoLab is quite an unstable platform as it can change the environment without notice.
Buying extra GPU time from Google CoLab	- No issues with limited GPU usage - No need to alter script	- Not available outside US and Canada, so not an option - CoLab is quite an unstable platform as it can change the environment without notice.
Training on CPU of local computer	- Free - Always accessible	- Way too slow to be actually useful - Crashes regularly - All files would need to be rewritten to work on a local computer
Training on an AWS computer	Would surely work	- Expensive - Would take a lot of time to learn how to work with AWS computer / IP connection in terminal - Need to alter all scripts to work on AWS computer

Table 2.3: Options to train the model with larger batch size and more epochs, and order in which they were approached / explored.

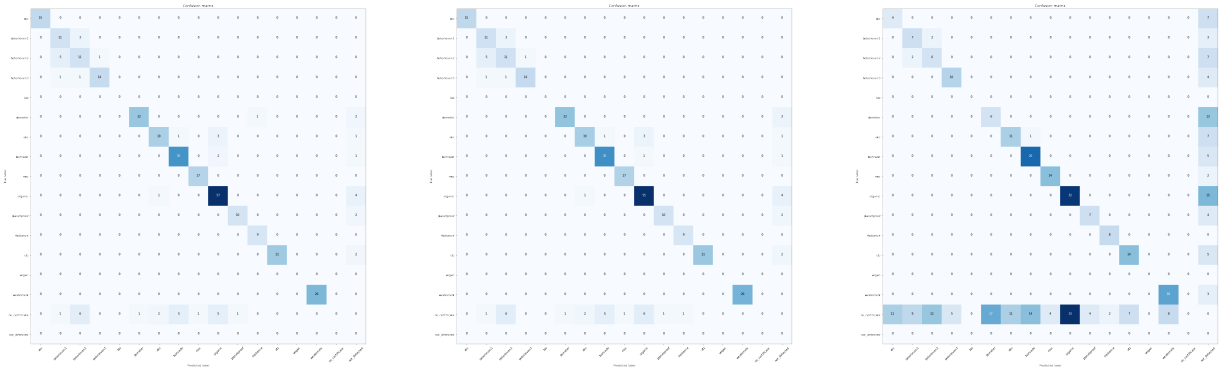
- How do I ensure that bounding boxes of predictions and actual labels overlap enough in order to make sure that the prediction is actually meant for the right location?
- How do I measure a false negative if the algorithm (initially) iterates over the separate predictions?
- How do I visualise this information?
- How do I visualise false negatives and false positives in a confusion matrix?

So, I wanted to build something that would look at the overlap between bounding box predictions in order to check if the predictions correspond to about the same object in the image. So, I implemented an algorithm that returns a true positive if the IoU between the predicted bounding box and annotated bounding box is larger than `iou_tres`. Also, the ImageAI predictor did not suppress all predictions as I needed it to, and because I did not know how to alter this within the ImageAI implementation, I added this myself. This means that the predictor of ImageAI and the evaluation of ImageAI are less accurate than the predictor and evaluation I have build.

Now that I knew the number of false and true positives and negatives, I could quantify the recall, precision and accuracy, and I could use these results to build confusion matrices. So I build a confusion matrix visualisation tool and began evaluating.

2.2.7 Optimizing evaluation parameters

The first confusion matrices looked very pleasing. They can be found in Figure 2.3. These confusion matrices look very good, but it's very important to note again that I predicted this on the wrong dataset. So, the results don't give an honest representation of the actual performance. But the confusion matrices work and can be visualized.



(a) IoU 0.1 (b) IoU 0.2 (c) IoU 0.5

Figure 2.3: Confusion matrices for different IoU's for the training of October 5th. Horizontal axis is the predicted label, vertical axis is the true label. The labels are: asc, beterleven1, beterleven2, beterleven3, bio, demeter, eko, fairtrade, msc, organic, planetproof, rfalliance, utz, vegan, weidemelk, no_certificate and not_detected. (Light to dark corresponds with low to high values)

It wasn't very clear what the best choice was for the IoU, except that 0.5 was way too high. Therefore, I wrote a small script that would iterate over different IoUs within the domain [0.01, 0.26] and visualize the performance. This was around the time I started using the right dataset, therefore, these results are valid. The plot of these IoU results can be seen in Figure 2.4. It is

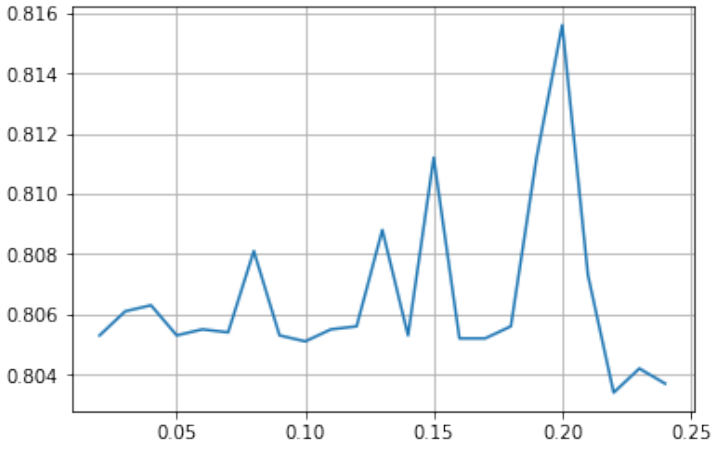


Figure 2.4: mAP values for different IoU values.

obvious the optimal value for the IoU in this case is 0.2. But, as the plot heavily fluctuates, this can differ for different models. To make the comparison between different models computation-

ally feasible, I decided to just use 0.2 as standard IoU. This would provide a more structured comparison between models.

2.2.8 Developing tools to optimize training aspects and setting up experiments

Now that the model worked and different evaluation methods were available, it was important to optimize aspects of the training that could affect the performance. This could be parameters, but also distribution of the dataset, choices in the augmentation, etc.

It was clear that the predictions for Bio and Vegan were off. After some consultation with Questionmark and Vantage AI, I decided to exclude these labels from the dataset because of two reasons. Firstly, the dataset contained a relatively small portion of these certificates and also, these labels are not important for the research Questionmark wants to do. Therefore, I build a function that would remove all Bio or Vegan labels from the annotation if present. This function is called `filter_labels_in_subset` and `filter_labels_everywhere`. In this way, the labels are still included in the Alles directory, but won't be used in the training.

After altering and running this, it appeared this gave a slight improvement. I decided to just build scripts that could tweak different aspects of the training process, and test in the end how these different tweaks would affect the performance. So, from then on I looked into the building scripts that would affect the following aspects:

- Decreasing the amount of Fairtrade and EU Organic labels in order to have a balanced dataset and thus create less false positives for these labels. This could not be done by deleting the datapoint from the dataset, because a lot of datapoints contained multiple certificates (especially together with EU Organic). Therefore, it was most convenient to select a subset of these datapoints, and draw a square over the certificates. An example of this can be found in Figure 2.5;

- Preprocessing the data in order to make it more universal. This can increase performance as there are less 'distractions' on each image. An example of this can be found in Figure 2.5;

- Easily alter weights of the loss function. The weights that can be altered affect the weighting of the false positives, mix-up between classes and the false negatives. For this it was important to understand how the open source code of ImageAI worked and how to alter it in my script;

- Easily alter aspects of the on the fly data augmentation. Two factors that are mainly taken into account is the zooming in-and-out of the picture, the ratio distortions of the picture, and the coloring in the picture. An example of altering all these aspects can be seen in Figure 2.7. This was again an example of figuring out how the open source code worked and implement the alteration of it;

I ran a training and evaluation every time I build some of these aspects. I will not speak about the results because these trainings were more in order to see if the script would run, rather than evaluate the usefulness of these factors. Also, I made the mistake for a while of prediction on the validation set, which it in some sense trains on. This inflated the results and it is therefore

(a) Original (b) Certificate hidden

Figure 2.5: An example of a picture before and after hiding the certificate.

(a) Original (b) Preprocessed

Figure 2.6: An example of a picture before and after preprocessing.

(a) Original (b) Augmented

Figure 2.7: An example of a picture before and after augmentation.

more useful to evaluate on the final trainings which were set up in a more experimental sense. These results will be elaborated on widely in chapter 3.

In order to properly compare the results more easily, a quantitative evaluation method on top of the confusion matrices would be helpful. I could use the results from the prediction on the test set in order to calculate the precision and recall per certificate, an average of this and an overall accuracy. This would be more insightful than the ImageAI mAP because these calculations were not very clear, and seemed different to the results from the confusion matrices.

Once all of this was build, I was ready to test all these different aspects in an experimental manner and compare the results in order to find the most optimal settings. The results of which can be read in the next section.

3. Results

In this chapter, the results of the final training sessions will be set out. The analysis of the results will focus on the following factors:

- Different quantitative results like precision, accuracy and recall (this will more thourougly be explained in section 3.3);
- Number of epochs needed;
- Distribution of the datasets;
- Confusion matrices;

The different factors and parameters that have been experimented with are listed below. How these factors and parameters are set exactly, will be explained in the settings table of each model.

1. Exclude AH Bio and Vegan label from dataset by deleting components from annotations (not needed for Questionmark, relatively small quantity so might not train well);
2. Balance dataset with by crossing out Fairtrade and Organic label (relatively huge presence in dataset, has shown to give many false positives);
3. Preprocessing with gaussian noise (uniformity of the dataset might cause it to train faster and slight noise can enhance the performance);
4. Altering the weights for the loss function (for example, the `no-object scale` (penalizing false positives harsher) and `xyhw scale` (penalizing displacement of the anchor boxes less);
5. Altering data augmentation settings (such as intensifying hue, saturation, exposure or altering the ratio distortion and boundaries on the zooming).
6. Adding more datapoints / pictures without certificates (in the form of deleting the Rainforest Alliance certificate)

3.1 Settings

In this section the different model parameters and settings will be set out precisely. The settings and the date of running (in order to distinguish between them) are listed in Table 3.1.

Alteration Date	Default 3 dec	1 5 dec	2 6 dec	3 7 dec	4 8 dec	4 9 dec	5 10 dec	5 11 dec	6 13 dec
Settings dataset									
Excluding Bio & Vegan?	No	Yes	No	No	No	No	No	No	No
Balanced dataset?	No	No	Yes	No	No	No	No	No	No
Preprocessed?	No	No	No	Yes	No	No	No	No	No
Added unlabeled data?	No	No	No	No	No	No	No	No	Yes
Settings augmentation									
`hue`	18	18	18	18	18	18	18	50	18
`saturation`	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3	1.5
`exposure`	1.5	1.5	1.5	1.5	1.5	1.5	1.5	3	1.5
`zoom_min`	0.5	0.5	0.5	0.5	0.5	0.5	0.75	0.5	0.5
`zoom_max`	1.5	1.5	1.5	1.5	1.5	1.5	1.25	1.5	1.5
`ratio_distortion`	0.7	0.7	0.7	0.7	0.7	0.7	0.4	0.7	0.7
Settings loss weights									
`obj_scale`	5	5	5	5	5	5	5	5	5
`noobj_scale`	1	1	1	1	2	3	1	1	1
`xywh_scale`	1	1	1	1	1	1	1	1	1
`class_scale`	1	1	1	1	2	1	1	1	1
Training settings									
`batch_size`	8	8	8	8	8	8	8	8	8
`num_experiments`	150	150	150	150	150	150	150	150	150

Table 3.1: Tested model settings for the final models. The number for the alteration corresponds to the number of the alteration enumerated in chapter 3

3.2 Distribution dataset

In this section the dataset distribution will be listed for every different training. In Table 3.2 and Table 3.3 the distribution of the datapoints into the train, test and validation subdirectory are shown in quanity and in percentage of the set respectively. In these tables three different distribution scenarios are set out. These scenarios are the default distribution, the distribution of altrication 1 (excluding AH Bio and Vegan certificate), altrication 2 (balancing out the Fairtrade and EU Organic certricates) and altrication 6 (adding more nonlabeled data and removing the rfaliance certificates). Because of the page restrictions, the certificates cannot be named in the tables. Therefore, the labels are alphabetical letters that correspond to the certificates in the following manner:

- **A** ASC
- **B** Beter leven 1 ster
- **C** Beter leven 2 sterren
- **D** Beter leven 3 sterren
- **E** AH Bio
- **F** Demeter
- **G** EKO
- **H** Fairtrade
- **I** MSC
- **J** EU Organic
- **K** Planetproof
- **L** Rain forrest alliance
- **M** UTZ
- **N** Vegan
- **O** Weidemelk

Certificate		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Alles		68	90	66	72	52	97	89	164	89	242	65	38	78	39	93
Default	*Train*	48	50	45	44	32	63	63	41	45	73	37	29	45	25	56
	Test	13	21	9	20	14	20	21	10	30	19	13	3	19	12	20
	Validation	7	19	12	8	6	14	5	11	14	17	14	6	14	2	16
1	*Train*	48	50	45	44	0	63	63	111	45	163	37	29	45	0	56
	Test	13	21	9	20	0	20	21	27	30	41	13	3	19	0	20
	Validation	7	19	12	8	0	14	5	26	14	38	14	6	14	0	16
2	*Train*	48	50	45	44	32	63	63	41	45	73	37	29	45	25	56
	Test	13	21	9	20	14	20	21	10	30	19	13	3	19	12	20
	Validation	7	19	12	8	6	14	5	11	14	17	14	6	14	2	16
6	*Train*	47	58	47	55	32	65	50	101	60	148	44	0	49	21	60
	Test	12	15	13	9	8	17	20	31	17	40	14	0	17	10	20
	Validation	10	17	6	10	12	15	20	33	12	56	6	0	12	12	12

Table 3.2: Distribution of the different datasets in *number* of datapoints.

Certificate		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Alles		5,1	6,7	4,9	5,4	3,9	7,2	6,6	12,2	6,6	18,0	4,8	2,8	5,8	2,9	6,9
Default	*Train*	6,9	7,2	6,5	6,3	4,6	9,1	9,1	5,9	6,5	10,5	5,3	4,2	6,5	3,6	8,0
	Test	5,3	8,6	3,7	8,2	5,7	8,2	8,6	4,1	12,3	7,8	5,3	1,2	7,8	4,9	8,2
	Validation	4,2	11,5	7,3	4,8	3,6	8,5	3,0	6,7	8,5	10,3	8,5	3,6	8,5	1,2	9,7
1	*Train*	6,0	6,3	5,6	5,5	0	7,9	7,9	13,9	5,6	20,4	4,6	3,6	5,6	0	7,0
	Test	5,1	8,2	3,5	7,8	0	7,8	8,2	10,5	11,7	16,0	5,1	1,2	7,4	0	7,8
	Validation	3,6	9,8	6,2	4,1	0	7,3	2,6	13,5	7,3	19,7	7,3	3,1	7,3	0	8,3
2	*Train*	6,9	7,2	6,5	6,3	4,6	9,1	9,1	5,9	6,5	10,5	5,3	4,2	6,5	3,6	8,0
	Test	5,3	8,6	3,7	8,2	5,7	8,2	8,6	4,1	12,3	7,8	5,3	1,2	7,8	4,9	8,2
	Validation	4,2	11,5	7,3	4,8	3,6	8,5	3,0	6,7	8,5	10,3	8,5	3,6	8,5	1,2	9,7
6	*Train*	5,6	6,9	5,6	6,6	3,8	7,8	6,0	12,1	7,2	17,7	5,3	0 XA	5,9	2,5	7,2
	Test	4,9	6,2	5,3	3,7	3,3	7,0	8,2	12,8	7,0	16,5	5,8	0	7,0	4,1	8,2
	Validation	4,3	7,3	2,6	4,3	5,2	6,4	8,6	14,2	5,2	24,0	2,6	0	5,2	5,2	5,2

Table 3.3: Distribution of the different datasets in *percentage* of entire directory.

3.3 Quantitative results

In this section we will lay out how every different model preformed. These results include the following measurements:

- Mean Average Precision (mAP) according to the build in ImageAI evaluation (it is not entirely clear when a prediction is seen as true or false (how this is done with the IoU, and how Non-Max Supression works into this, therefore it is not entirely clear what this takes into account);
- Mean Average Precision (mAP) based on the prediction and its (self build) evaluation.

This is calculated as follows:

$$\sum_{l \in L} \frac{TP(l)}{TP(l)+FP(l)}. \tag{3.1}$$

Here L is the set of labels. A true positive holds when the IoU between the predicted bounding box and the annotated bounding box is larger than 0.1 and when the NMS is set to be 0.8);

- Mean Average Recall (mAR) based on the prediction and its (self build) evaluation. This is calculated as follows:

$$\sum_{l \in L} \frac{TP(l)}{TP(l)+FN(l)}; \tag{3.2}$$

- Accuracy based on the prediction and its (self build) evaluation. This is calculated as follows:

$$\frac{\sum_{l \in L} TP(l)+TN}{\sum_{l \in L} TP(l)+TN \sum_{l \in L} FP(l)+\sum_{l \in L} FN(l)}; \tag{3.3}$$

- The Average Precision on the images without a certificate. This is calculated as

$$\frac{TN}{TN+\sum_{l \in L} FN(l)}. \tag{3.4}$$

The calculation was chosen to be done in this manner in order to seem as if the no certificate is also a label. (Therefore, a false negative can be viewed as a false positive for no certificate). Because most mistakes were made for

- The Average Recall on the images without certificate. This is calculated as

$$\frac{TN}{TN+\sum_{l \in L} FP(l)}. \tag{3.5}$$

In Table 3.4 all these results are listed. The results for the mAP build into ImageAI have been included, but do not weight as heavily as the other measurements. The most important measurement to judge the models upon is the accuracy. The significant improvements relative to the default model are in bold.

The choice of the amount of epochs can be considered weird. But, this is due to the fact that ImageAI only saves the models for which the loss function value increases relative to the last best scoring model. Most models had run for about 150 number of experiments / epochs, but only saved the best scoring models which were at about 50 epochs.

Taking the accuracy into account most, we see that alteration 1 (excluding AH Bio and Vegan), alteration 2 (balancing the dataset for Fairtrade and EU Organic) and alteration 6 (adding more datapoints without certificates) improve the accuracy most. We can see that this accuracy improvement is mostly affected by the increased scores of the negative AR and AP's, meaning there is a increase in true negatives in relation to false negatives and false positives. For alteration 2 and 6, we see that they also improve the mAR, which also indicates more true positives in relation to the false negatives.

For the final model, we decided to include alteration 1, 2 and 6. In the results for the final model, we see a heavy improvement in all categories, except the mAP. But the map is still incredibly high.

What we can also see that as the number of epochs increases, the negatives AP (and thus the false negatives relative to the true negatives) slightly decreases. While, on the other hand, as the number of epochs increase, the mAP and mAR slightly increase.

We can conclude that the final model with 57 epochs has scored the best.

Alteration	**Date**	**Epochs**	**mAP ImageAI**	**mAP**	**mAR**	**Accuracy**	**Negatives AP**	**Negatives AR**
Default	3 dec.	65	0,761	0,931	0,832	0,770	0,244	0,392
		53	0,759	0,873	0,837	0,716	0,284	0,**442**
1	5 dec.	51	0,**794**	0,922	0,822	0,**793**	0,**345**	0,**527**
2	6dec.	93	0,740	0,880	0,**855**	0,**780**	0,**390**	0,**533**
		65	0,743	0,903	0,**851**	0,**785**	0,**400**	0,**542**
		46	0,733	0,905	0,***858***	0,**803**	0,**390**	0,**542**
3	7dec.	48	0,690	0,922	0,769	0,717	0,208	0,301
		46	0,685	0,924	0,783	0,736	0,216	0,293
4	8dec.	51	0,770	0,923	0,786	0,739	0,**344**	0,**452**
		48	0,768	0,922	0,781	0,734	0,**340**	0,**434**
4	9dec.	67	0,606	0,938	0,748	0,730	0,212	0,310
		50	0,603	0,**943**	0,749	0,743	0,214	0,319
5	10 dec.	45	0,591	0,463	0,687	0,398	0,157	0,227
		33	0,739	0,937	0,752	0,730	0,191	0,300
5	11 dec.	55	0,759	0,930	0,817	0,775	0,289	0,**440**
		41	0,601	0,562	0,748	0,477	0,204	0,278
6	13 dec.	41	0,765	0,***947***	0,**831**	0,**819**	0,**563**	0,**684**
		34	0,755	0,945	0,**831**	0,**822**	0,**540**	0,**684**
Final	14 dec.	57	0,**803**	0,902	0,**846**	0,***852***	0,**713**	0,***885***
		47	0,**807**	0,897	0,**845**	0,**842**	0,**703**	0,**857**
		41	0,***812***	0,891	0,**842**	0,**843**	0,***722***	0,**876**

Table 3.4: Quantitative results of the final models. The bold values are values that improved significantly with respect to the default, and the bold italic values, are the most optimal improvements.

3.4 Confusion matrices

In this section, the confusion matrices of the final model will be shown and the difference relative to the highest scoring default model will be discussed. The confusion matrices for all alterations can be found in the appendix.

The y-axis contains the actual, annotated label and the x-axis contains the predicted label. Besides from the labels, this confusion matrix contains three other ticks: no_certificate, not_detected and true_negatives. No_certificate indicated a false positive, not_detected indicates a false negative and true_negatives, as one might expect, indicates the true negatives.

The comparison between the default model and the final model can be found in Figure 3.1. In the default model, the red circled parts are some of the parts that I aimed to improve. In the final model, I highlighted green some of which that clearly improved and circled red some of the parts that had not improved. This will be elaborated on more thoroughly next.

There are many observations to be made from this model.

- There are much less off-diagonal elements in the final model relative to the default model. This means that there are less mix-ups of different lables (or false positives). Most mix-

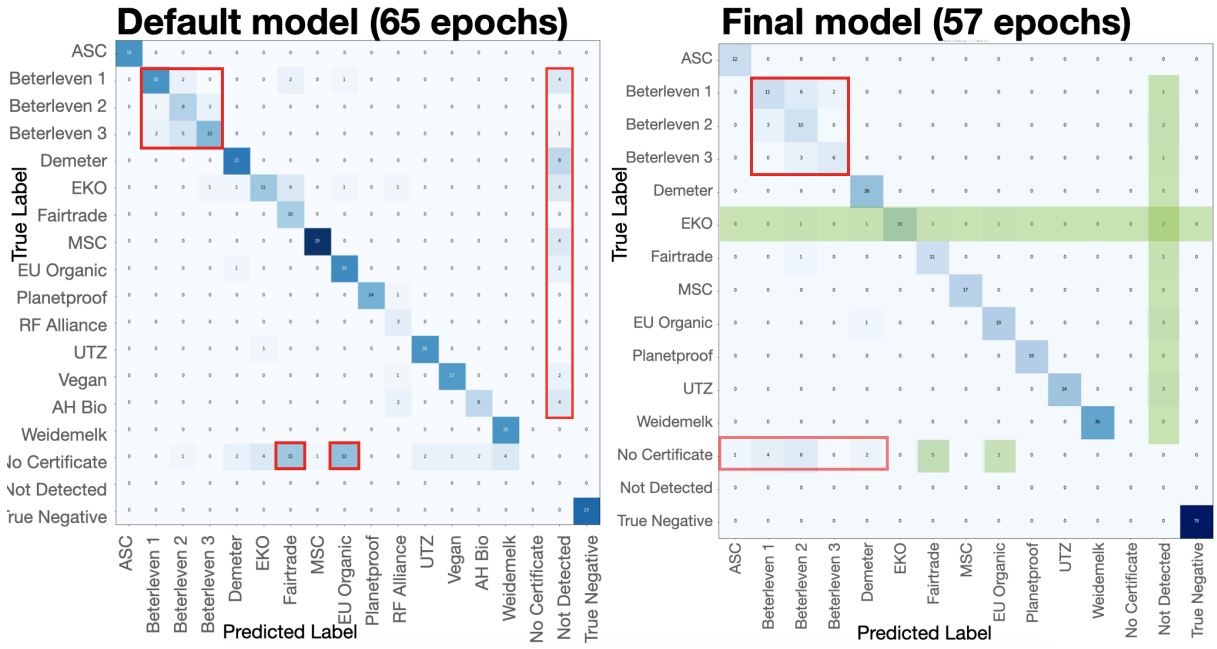


Figure 3.1: Confusion matrices of the best scoring default model (65 epochs) and the best scoring final model (57 epochs). The red squares on the left indicate aspects up for improvement, but on the right they show aspects that did not improve. The green highlighted parts on the right indicate improved aspects.

ups are only incidental (not more than once, there is no clear pattern). The only structural mix-ups that occur are for the BeterLeven certificates, but we will get to this later on;

- The false negatives have heavily decreased in the final model relative to the default model;
- The total amount of false positives has also decreased. Where the EKO, UTZ, Weidemelk, Vegan and AH Bio certificates were more often false recognised in the default model, this misrecognition has shifted to the BeterLeven certificate and ASC certificate in the final model (but relatively less often). This total amount of false positives for EU Organic certificate has decreased, but the Fairtrade certificate have not;
- There are more true negatives. This is mostly due to the fact that there are more labels that do not contain a certificate, but also because there are less false positives;
- The mix-up between the BeterLeven 1, 2 and 3 Sterren certificates has not improved a lot. This will probably not be solved by experimenting with hyper perameters (this can be seen when analyzing the confusion matrices for all different alterations). One way to try to solve this is by including much more datapoints. If this then does not significantly improve, training the model to just recognise a BeterLeven certificate can be tried, followed up by training another ConvNet to recognise the difference between 1, 2 and 3 stars;

3.5 Recommendation

In this section, a few options will be noted in order to improve this model if necessary.

- Train with more datapoints. Especially pictures that contain no certificate, as a slight increase in such datapoints immensely decreased the false positives;
- Also count the datapoints without certificates in *Split_en_overzicht.py*;
- Train this model on recognising only a Beterleven certificates, and build a second CNN to recognise 1, 2 and 3 stars separately;

4. Documentation

This section give an elaborate explanation of how the different scripts work. Each section constains an explanation of the purpose of the script, the input needed, what it outputs, what components it contains and what packages need to be imported in order for it to work.

First off, the scraper will be explained. Next, the script that splits the data will be explained. After that trainer will be explained and lastly the evaluation script.

4.1 `Scraping.py`

4.1.1 Purpose

This script is made in order to scrape the pictures of a provided csv-files of picture-URLs to be saved and written into a directory.

4.1.2 Input

In a directory make a sub-directory (preferably named *csvs*). In here, place csv files containing only the picture-URLs. This directory structure should look as the example shown in Listing 4.1.

```
images
|- csvs
   |-- asc.csv
   |-- demeter.csv
       :
|- asc0.jpg
|- asc1.jpg
|- asc2.jpg
|- demeter0.jpg
|- demeter1.jpg
|- demeter2.jpg
   :
```

Listing 4.1: Example of the directory structure expected for `Scraping.py`. In the *plaatjes* directory, the scrapted images will be put with a unique number as suffix. The URLs from which these images are scraped are listed in csv files that are put into the subdirectory *csvs*.

Each line of the output csv file contains one URL of a picture. When invoking the command, one needs to declare the `nr_pic_per_cert` as the amount of pictures that need to be scraped per csv-file and the path to the directory to which to write the pictures and in which the csv subdirectory can be found (in our example the *plaatjes* directory).

4.1.3 Output

This script writes the scrapted images as `.jpg` files to the directory as seen in Listing 4.1. The files will have a naming of the form `name_cvsfile###.jpg` where `name_cvsfile` is the name of the cvs-file the picture-URLs are in (without `.cvs`) and `###` is a number indicating the how-many'th picture-URL in the cvs file it corresponds to.

4.1.4 Components

This script only contains one defined function, namely `scrape(path , nr_pic_per_cert = 100)`.

4.1.4.1 `scrape(path , nr_pic_per_cert = 100)`

The `scrape`-function's purpose is to download the images corresponding to the URLs in the provided csvs. It works as follows:

- Makes two lists of strings of all csv files in the subdirectory csvs (one with *.csv* and one without).
- Then, per certificate `cert_name`, it scrapes `nr_pic_per_cert` pictures from the URLs in the `cert_name.csv` file using request. And saves these pictures as `.jpg` files using Image and ByteIO in the *pictures* directory with a distinctive numbering.

This function has two arguments: `path` and `nr_pic_per_cert`. The default setting for `nr_pic_per_cert` is 100.

4.1.5 Packages

- `Image` from `PIL`
- `requests`
- `BytesIO` from `io`
- `pandas`
- `os`

4.2 `Split_en_overzicht.py`

4.2.1 Purpose

The main goal of this script is to prepare the dataset for training. There are a few steps to this and a few options to be tweaked. The script includes the following steps (the items bulleted with an o are optional and can be turned off with a boolean that will be named in subsection 4.2.2):

- Convert all `.jpeg` files to `.jpg` and adjust the annotations accordingly;
- o Randomly split the names of the dataset Alles into three groups according to a certain distribution and write this to a cvs file;
- o Write the files in the (formerly saved) cvs file to the directories;

- Filter out any label that is not listed in `labels` (so delete all labels in annotations containing non-listed labels);
  - o Balance the Fairtrade and EU Organic labels by crossing out the certificate in the picture and removing the label from the annotation;
  - o Preprocess the data with Gaussian blur to be more uniform;
  - o Visualize the bounding boxes for a specific label within the train set;
- Make an overview of how the certificates are distributed over these three sets (both in numbers and percentage of the dataset);

4.2.2 Input

This script needs two types of input: the declaration of a few global variables and a mounted drive directory structure. The global variables that need to be declared are the following:

- `DATA_DIR`: the path to the overall data directory;
- `labels`: The labels that need to be within the dataset (these are generally `asc, beterleven1, beterleven2, beterleven3, demeter, eko, fairtrade, msc, organic, planetproof, rfalliance, utz, weidemelk, bio, vegan`);
- `split_and_csv_bool`: Indicating whether the datapoints should be divided into three parts (by a certian distribution) and written to the cvs files `test_anno.csv`, `train_anno.csv`, `validation_anno.csv`, `test_img.csv`, `train_img.csv` and `validation_img.csv`;
- `reassign_csv_bool`: Indicating if the filenames listed in the `..._img.csv` and `..._anno.csv` files should be copied to the proper directory;
- `preprocess_bool`: Indicating if the data written into the train, test and validation directories should be preprocessed;
- `balance_certificates_bool`: Indicating whether a certain amount of Fairtrade and EU Organic labels should be removed (crossed out and removed from annotation file) in order to have a more balanced dataset;
- `visulaisation_bool`: Indicating if some exemplary data within the train set should be visualised (costs a lot of running time).

Even if the booleans are all set to false, the declarations of the functions are ran, because some functions are used in different sections.

The mounted directory should be ordered as shown in Listing 4.2. Here, the directory *Alles* > *images* contains all the images in .jpg format, and *Alles* > *annotations* contains all the .xml files that indicate where the bounding boxes are and to which class they belong. These annotations are in *PascalVOC* format. The sub-directories *train, validation* and *test* may also already be present and filled as seen in Listing 4.3 [1].

[1] The algorithm deletes these directories and replaces them with equally named empty ones.

```
data
|- Alles
   |-- images
   |-- annotations
```

Listing 4.2: The set-up of the data directory in order to properly split the data. Here, the directory *Alles* > *images* contains all the images in .jpg format, and *Alles* > *annotations* contains all the accompanying .xml files that indicate where the bounding boxes are and to which class they belong.

```
data
|- Alles
   |-- images
   |-- annotations
|- train
   |-- images
   |-- annotations
|- validation
   |-- images
   |-- annotations
|- test
   |-- images
   |-- annotations
```

Listing 4.3: The set-up of the data directory in order to properly split the data. Here, the sub-directory *images* contains all the images in .jpg format, and *annotations* contains all the accompanying .xml files that indicate where the bounding boxes are and to which class they belong.

4.2.3 Output

The output of this script covers three things:

- The directories train, test and validation contain two sub-directories: *images* and *annotations*. The *images* sub-directory will contain all labeled (possibly altered and preprocessed) pictures in a `.jpg` format. The *annotations* sub-directory will contain all the corresponding (possibly adjusted) `.xml` files. The directory data will eventually look as in Listing 4.3. This split is divided into resp. 64%, 16% and 20% of the *Alles* data.
- It outputs a visualisation of a certain label in the train set with the accompanying bounding boxes;
- It prints for all datasets Alles, train, test and validation the number of datapoints for each certificate and what percentage of that directory they make up.

4.2.4 Components

This script contains many sections with sub-functions. Rather than speaking about all the functions in this script, the different sections will be explained.

4.2.4.1 Convert .jpeg to .jpg

The first section *Convert .jpeg to .jpg* converts image files in *Alles* > *images* from `.jpeg` files to `.jpg` files, and reassures this is also denoted in the corresponding xml annotation. To reassure this, the function `jpeg_to_jpg_xml(path)` was build. This function has as attribute only the path.

`jpeg_to_jpg_xml(path)`
—∘ `path`: A string of the path that leads to the *Alles* directory.

This function first changes all the filenames of images containing *.jpeg* to *.jpg* files using `os.rename`. Then, it iterates over all the xml annotations and changes the roots for filename and path from *.jpeg* to *.jpg* if needed using `glob` and `ElementTree`. It does not check that the jpeg files and xml files that are altered correspond to each other, because this would unnecessarily complicate the implementation. This might cause some issues, but if there are non-correspondent files, this will cause problems later on in the script anyways.

4.2.4.2 Split dataset

In this section, 6 cvs files are created that contain the names of the files that should be written to one of the sub-directories as seen in Listing 4.3 excluding Alles.

For each dataset (train, test and validation) two textfiles named `*_img.txt` and `*_anno.txt` are created (with `*` being either `train, test` or `validation`), indicating which .jpg file with corresponding .xml file should be put into either train, test or validation. The split is made in such a way that the trian set contains 64% of the files, validation 16% and test 20%.

This section will run if the boolean *split_and_csv_bool* is set to true.

It contains the following functions:

`split_filenames(DATA_DIR, fold)`
—∘ DATA_DIR: A string of the path to the data directory. In this directory should be the directory in which the sub-directory *Alles* can be found (so, not the path to the *Alles* directory it self).
—∘ fold: A float of the factor to which the split will be made. Must be within domain $[0,1]$. The data in the directory *Alles* will be split into the following percentages as follows:

$$\begin{cases} \text{train:} & (1-fold)^2 \\ \text{validation:} & (1-fold)fold \\ \text{test:} & fold \end{cases} \tag{4.1}$$

This function first changes all the filenames of images containing *.jpeg* to *.jpg* files using `os.rename`. Then, it iterates over all the annotation files using `glob` and changes the roots for filename and path from *.jpeg* to *.jpg* if needed using `ElementTree`. It does not check that the jpeg files and xml files that are altered correspond to each other, because this would unnecessarily complicate the implementation. This might cause some issues, but if there are non-correspondent files, this will cause problems later on in the script anyways.

`write_df_to_csv(dir, df)`

—∘ DATA_DIR: A string of the path to the entire data directory (a folder generally named data).

—∘ dir: A string of the directory to which the files must be written. Must be train, test or validation (it raises an error if not).

—∘ df: The dataframe containing the names of the files that should be written to the dir directory.

This function creates two .txt-files named dir*_img.txt* and dir*_anno.txt* within the dir directory containing the paths of all the files that should be written to the dir directory.

split_and_write(DATA_DIR, fold)

—∘ DATA_DIR: A string of path to the entire data directory (a folder generally named data).

—∘ fold: A float of the fold that should be called in the function split_filenames. Automatically set to 0.2.

This function first runs split_filenames and then runs write_df_to_csv for the train, test and validation directory.

4.2.4.3 Write files to directories

In the section *Write files to directories*, the files listed in the text files *_img.txt and *_anno.txt are copied to the file they are destined for. This section runs if the boolean reassign_csv_bool is set to True.

clear_img_anno(dir)

—∘ dir: a string with the directory to which this function must be applied. Should be train, test or validation.

This function creates empty sub-directories images and annotations in the dir directory.

check_missing_files(dir)

—∘ dir: a string with the directory to which this function must be applied. Should be train, test or validation.

This function checks if the split and assigning of files to the dir directory has gone correctly. It prints out what files are in dir if there exist any, and what file appears in either images or annotations and not in the other, if there are any.

4.2.4.4 Filter data

This section aims to remove all annotation elements that contain a label that is not listed in the global variable *labels*. It applies this to the train, test and validation sub-directories, as to not change the Alles data.

filter_labels_in_subset(path, labels)

—∘ path: A string of the path that leads to an annotations directory.

—∘ labels: A list of strings of all the labels (as formulated in the annotations) that need to be included.

This function iterates over all .xml-files in `path` and seeks if they contain a label that is not in `labels` and then removes that label (or *object*) from the the tree in the xml file (leaving all the other labels/objects in the xml file).

`filter_labels_everywhere(path, labels)`

- `path`: A string of the path that leads to the data directory (globally declared as *DATA_DIR*).
- `labels`: A list of strings of all the labels (as formulated in the annotations) that need to be included.

  This function applies `filter_labels_in_subset` to the train, test and validation sub-directories.

4.2.4.5 Balance dataset by crossing out labels

This section aims to cross out and dis-annotate 55% of the EU Organic and 61% Fairtrade certificates in order to decrease the amount of datapoints for those labels. The goal is to prevent false positives for these two certificates. The labels are crossed out by a purple box the size (and location) of the bounding box as stated in the annotation. This section is executed if *balance_certificates_bool* is set to True.

`draw_over_label_and_alter_xml(path, filename, label)`

- `path`: A string of the path that leads to one of the three main sub-directories (train, test or validation).
- `filename`: A string of the .xml-file to which to apply this function.
- `label`: A string of the certificate that needs to be crossed out and dis-annotated.

  This function draws a purple box over the certificate `label` in `filename` the accompanying .jpg-file and removes the object from the .xml-file.

`list_only_label(path, label)`

- `path`: A string of the path that leads to an annotations directory.
- `label`: A string of the certificate that needs to be crossed out and dis-annotated.

  This function returns a list of .xml-file names in `path` that contain an `label` certificate in the annotation.

`remove_perc_labels_in_dir(path, label, fraction_to_remove)`

- `path`: A string of the path that leads to one of the three main sub-directories (train, test or validation).
- `label`: A string of the certificate that needs to be crossed out and dis-annotated.
- `fraction_to_remove`: A float indicating what fraction of the files with a `label` certificate should be altered.

  This function applies `draw_over_label_and_alter_xml` to `fraction_to_remove` of the list created by `list_only_label(path, label)`.

`remove_perc_everywhere(path, label, fraction_to_remove)`

- `path`: A string of the path that leads to the data directory (globally declared as *DATA_DIR*).
- `label`: A string of the certificate that needs to be crossed out and dis-annotated.
- `fraction_to_remove`: A float indicating what fraction of the files with a `label` certificate should be altered.

  This function applies `remove_perc_labels_in_dir` to the all the trhee sub-directories.

4.2.4.6 Preprocessing

This section preprocesses the entire (split) dataset. This is done using OpenCV. An example of such preprocessed data can be seen in Figure 2.6. This is done in order to universalize the dataset aiming to decrease trainingtime and increase performance. An example of such a preprocessed image This part is executed when `preprocess_bool` is set to True.

```
prepr_dir(path, ks = 0)
```

- `path`: A string of the path that leads to an images directory in a train, test or validation sub-directory.
- `ks`: Float indication to what ks to apply in OpenCV's `GaussianBlur`

  This function sets out to replace all images in `path` with one that has subtracted a OpenCV's `GaussianBlur` version of itself.

```
preprocess(DATA_DIR )
```

- `DATA_DIR`: A string of path to the entire data directory (a folder generally named data).

  This function applies `prepr_dir` to the three train, test and validation subsets.

4.2.4.7 Visualisation

This section sets out to visualize 25 datapoints in the final train dataset with the annotated bounding boxes. It is standardized to display images with a 'beterleven3' certificate. It is executed when the boolean *visualisation_bool* is set to True.

```
xml_to_csv(path)
```

- `path`: A string of the path that leads to an annotations directory.

  This function creates a dataframe with columns filename, certifiate, xmin, ymin, xmax, ymax where every row of the dataframe corresponds to a file and an annotated certificate.

```
visualise_data_for_label(DATA_DIR, label)
```

- `DATA_DIR`: A string of the path that leads to an annotations directory.
- `label`: A string of the label for which you display datapoints that contain that label.

  This function visualizes 25 images that contain a `label` certificate in the annotated .xml-file.

4.2.5 Check splits

This section lists the amount of certificates and percentage relative to all datapoints that are in a directory for the Alles, train, test and validation sub-directories.

`certificates_in_dir(dir)`

—∘ `dir`: A string of the directory in `DATA_DIR` that leads to an annotations directory.

This function counts how many datapoints of each certificate are in the *dir* directory and what percentage this amount is relative to all datapoints in that directory.

4.2.6 Packages

- `pandas`
- `os`
- `numpy`
- `datasets, linear_model` from `sklearn`
- `train_test_split` from `sklearn.model_selection`
- `shutil`
- `datetime`
- `glob`
- `xml.etree.ElementTree`
- `lxml.etree`
- `pyplot, text` from `matplotlib`
- `Rectange` from `matplotlib.patches`
- `math`
- `cv2`
- `random`

4.2.7 Remarks

- In the section *write files to directories*, the files listed in the created text files are *copied* to the accompanying directories. This process can be accelerated by using a link (`ln`) function. But, it does not work in CoLab.

4.3 `Trainer.py`

4.3.1 Purpose

The main goal of this script is to train the model on the provided dataset.

4.3.2 Input

This script needs two types of input: the declared variables and a proper structure of a dataset.

The variables that need to be declared are the following:

- `DATA_DIR` : the path to the overall data directory; MODEL_PATH: the path to the `.h5` file from which will be transfer learned (so far, generally has been `pretrained-yolov3.h5`);
- `labels`: a list of all labels included in the annotations;
- `hue_distortion`: augmentation parameter of the ImageAI `BatchGenerator` class. Affects how broad the domain is in which the hue may randomly be augmented;
- `sat_distortion`: augmentation parameter of the ImageAI `BatchGenerator` class. Affects the intensity of the saturation;
- `exp_distortion`: augmentation parameter of the ImageAI `BatchGenerator` class. Affects the intensity of the exposure;
- `zoom_min`: augmentation parameter of the ImageAI `BatchGenerator` class. Sets a lower bound up to which a picture can be zoomed in.
- `zoom_max`: augmentation parameter of the ImageAI `BatchGenerator` class. Sets an upper bound up to which a picture can be zoomed out.
- `ratio_distortion`: augmentation parameter of the ImageAI `BatchGenerator` class. Sets an upper bound up to which the height and width ration can be distorted.
- `train_obj_scale`: loss function factor of the ImageAI `BatchGenerator` class deciding the intensity on which a false negative will be penalized;
- `train_noobj_scale`: loss function factor of the ImageAI `BatchGenerator` class deciding the intensity on which a false positive will be penalized;
- `train_class_scale`: loss function factor of the ImageAI `BatchGenerator` class deciding the intensity on which a class mixup will be penalized;
- `nr_epochs`: number of epochs or experiments that the trainingprocces will do. In every epoch all pictures are analyzed. The number of epochs affects the accuracy, at first it will increase, but with too many, the model can overfit. Therefore, it is important to analyze the loss value in time;
- `batch_size`: the number of batches the (training) dataset is split into. After training on a batch, the training weights are updated to be optimal. Therefore, a higher batch size, slightly increases the accuracy as well.

The general settings for the training parameters and the factors used within the `BatchGenerator` class can be found in Table 3.1 under default.

The structure of the dataset must be as seen in Listing 4.4.

```
data
|- Alles
   |-- images
   |-- annotations
|- train
   |-- images
```

```
    |-- annotations
|- validation
    |-- images
    |-- annotations
|- test
    |-- images
    |-- annotations
|- logs
|- json
|- models
    |- pretrained-yolov3.h5
|- cache
```

Listing 4.4: The set-up needed for training.

4.3.3 Output

There are two types of output this script gives, and these can be split into two groups:

- files that are output
- training information

Let's start with the files that are output:

- `detection_model-ex-###-loss-....h5`: the model of training experiment ### with loss value ... in the model subdirectory;
- `detection_test_data.plk`: a cache file for the training in the cache subdirectory. Important to be deleted before a new training is started;
- `detection_train_data.plk`: a cache file for the training in the cache subdirectory. Important to be deleted before a new training is started;
- `detection_config.json`: a JSON file in the json subdirectory that stores all important factors about the model. Important to be saved properly after training in order to be used in the evaluation and prediction;
- `events.out.tfevents.###`: a tensor flow file that stores the course of the loss function value. Can be found in the logs subdirectory;

The printed output looks as seen in Figure 4.1.

4.3.4 Components

This script contains many sections with sub-functions. Rather than speaking about all the functions in this script, the different sections will be explained.

4.3.4.1 Edit ImageAI augmentation implementation

In this script the class *BatchGenerator* from ImageAI is edited. All the factors that have been changed, can declared in the input variables.

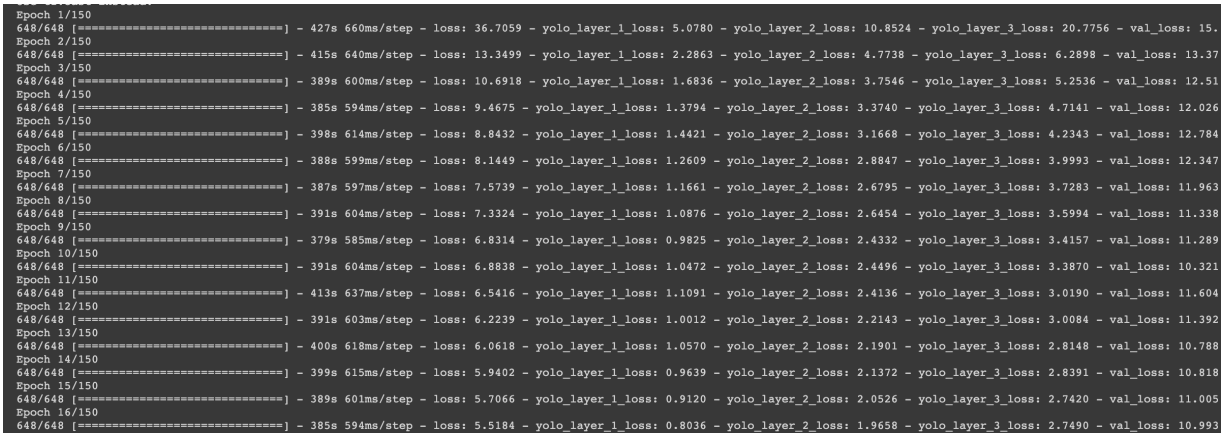


Figure 4.1: An example of the way the output looks for the trainer

4.3.4.2 Load and prepare model

This section imports the trainer, sets it to be a YOLOv3 algorithm and declares what directory should be linked.

4.3.4.3 Remove cache files

This section removes the existing cache files if they have not been deleted yet. This is done because otherwise, the trainer will not run properly.

4.3.4.4 Train

This section sets specific parameters for the model and then trains on the data. This is also the section where the information on how the training is going can be found.

4.3.5 Packages

- `tensorflow`
- `matplotlib.pyplot`
- `matplotlib.image`
- `Image` from `IPython.display`
- `DetectionModelTrainer` from `imageai.Detection.Custom`
- `os`
- `cv2`
- `numpy`
- `copy`
- `BoundBox, bbox_iou` from `imageai.Detection.Custom.utils.bbox`
- `apply`$_{r}$$andom_{s}cale_{a}nd_{c}rop, random_distort_image, random_flip, correct_bounding_boxes from$`imagea`
- `DetectionModelTrainer` from `imageai.Detection.Custom`
- `normalize, evaluate, makedirs` from `imageai.Detection.Custom.utils.utils`

4.4 Evaluate.py

4.4.1 Purpose

The main goal of this script is to train the model on the provided dataset.

4.4.2 Input

This script needs two types of input:

- A specific declaration of certain parameters
- A dataset structured in a certain way

The variables that need to be declared are the following:

- `DATA_DIR` : the path to the overall data directory;
- `MODEL_PATH`: the path to the `.h5` file that corresponds to the model you want to evaluate or predict upon;
- `JSON_PATH`: the path to the json file that corresponds to the model of `MODEL_PATH`;
- `TEST_DIR` : the path to the directory containing an images and annotations sub-directory that should be evaluated;
- `labels`: a list of all labels included in the annotations;

The dataset structure should be as seen in Listing 4.5. It is very important that the json file properly corresponds to the model used. Otherwise, this script will not work.

```
data
|- Alles
   |-- images
   |-- annotations
|- train
   |-- images
   |-- annotations
|- validation
   |-- images
   |-- annotations
|- test
   |-- images
   |-- annotations
|- logs
|- json
   |- detection_config.json
|- models
   |- detection_model-ex-047--loss-0003.283.h5
|- cache
```

Listing 4.5: The set-up needed for training.

4.4.3 Output

This script outputs the following three aspects:

- An evaluation from the ImageAI code;
- An evaluation of the predictions done on the test set. Focusses on precision and recall;
- A confusion matrix based on the prediction done on the test set.

4.4.4 Components

This script contains multiple sections and sub-functions. Rather than speaking about all the functions in this script, the different sections will be explained.

4.4.4.1 Declare functions

In this section a few functions are declared that are used throughout the rest of the .py file.

`iou(box1,box2)`

- `box1`: a list with the coordinates of box1 listed as [x1,y1,x2,y2]
- `box2`: a list with the coordinates of box2 listed as [x1,y1,x2,y2]

  This function calculates the iou for box1 and box2.

`yolo_non_max_suppression(scores, boxes, classes, iou_threshold = 0.5)`

- `scores`: The probability scores for a certain detection as detected by `detectObjectsFromImage`
- `boxes`: The bounding boxes for a certain detection as detected by `detectObjectsFromImage`
- `classes`: The detected classes scores for a certain detection as detected by `detectObjectsFromImage`
- `iou_threshold`: the threshold on the IoU for which two bounding boxes are considered to overlap (and thus predict on the same item)

  This function tries to find the most promising detection from all detections within a certain IoU on the basis of highest probability.

`what_labels_in_xml(image_path)`

- `image_path`: A string of the path that leads an image. This image must be contained in an images subdirectory in a directory that must also contain an annotations subdirectory with corresponding xml-annotation files.

  This function returns all annotated labels present in the image `image_path`.

4.4.4.2 Load, prepare and evaluate model

This section loads the model, links it to the proper directory and model, etcetera, in order to use the build in ImageAI evaluation that will be printed out. This script also contains a commented script, that iterates over different ious, to see which one provides an optimal mAP. It is commented out because it takes a very long time to run and has already shown that 0.2 generally is the optimal iou.

4.4.4.3 Predict results on testset

This section iterates over all items in the test set and per picture predicts what labels are present. Then it saves and prints these predictions. It also keeps a tracker for the later on created confusion matrices. This uses the following subfunctions:

`xml_to_csv(path)`

- `path`: A string of the path that leads to an annotations directory.

  This function creates a dataframe with columns filename, certifiate, xmin, ymin, xmax, ymax where every row of the dataframe corresponds to a file and an annotated certificate.

`plot_yolo(image_path, f, probs, boksen, klassen)`

- `image_path`: A string of the path that leads to a directory that contains the jpg-file `f`
- `f`: a jpg file that must be printed
- `probs`: a list of probability scores for different detections that is ordered corresponding to boxen and klassen
- `boxen`: a list of bounding box coordinates of the detections that is ordered corresponding to probs and klassen
- `klassen`: a list of class names of the detections that is ordered corresponding to boxen and probs

  This functions prints the image `f` in the `image_path` file and print it with provided classes and bounding boxes.

`keurmerk_predict(print_results=False,` $\texttt{min}_p rob = 30, iou_t res = 0.2$`)print_results:` $abooleanindicatingwhethertheresultsshouldbeprinted$

- `min_prob`: threshold below which predictions are dropped
- `iou_tres`: iou treshold for which two detections are considered to correspond to the same thing in the picture

  This function iterates over all the pictures in the test directory, makes predictions on them, prints these predictions alongside the picture (if `print_results = True`) and also step by step tallies the true positives, true negatives, false positives and false negatives.

4.4.4.4 Confusion Matrix

This section creates a confusion matrix and prints out the average precision and average recall for each label.

`plot_confusion_matrix(cm, classes,normalize=False, title='Confusion matrix', cmap=plt.cm.Blues)`

- `cm`: an numpy matrix of a confusion matrix
- `classes`: a list of strings containing the names of the different classes within the confusion matrix
- `normalize`: boolean indicating if the confusion matrix should be normalized

—∘ `title`: a string containing the name of the confusion matrix

—∘ `cmap`: the pyplot color map that should be used

This function prints and plots the confusion matrix. Normalization can be applied by setting ‘normalize=True‘.

`create_confusion_matrix(df_confusion, cm_plot_labels, true_negs)`

—∘ `df_confusion`: the dataframe created in `keurmerk_predict`

—∘ `cm_plot_labels`: a list of strings corresponding to the labels that are in the confusion matrix

—∘ `true_negs`: integer containing the number of true negatives

This function translates the `df_confusion` dataframe to a proper confusion matrix (in the shape of a dataframe) that contains the `cm_plot_labels` created in `create_cm_plot_labels` and the true negatives.

`calc_precision_recall(cm, labels)`

—∘ `cm`: a numpy matrix of the confusion matrix

—∘ `labels`: the labels of the certificates included in the dataset

This function calculates precision and recall per label and the mean of these. In this function something is a false positive if anything other than the accurate certificate was detected and no certificate scores not included in mean.

`create_cm_plot_labels(labels)`

—∘ `labels`: the labels of the certificates included in the dataset

This function adds *no_certificate*, *not_detected* and *true_negatives* to the list of labels, which is the entire list of labels in the confusion matrix.

`df_confusion_to_numpy(df_cm)`

—∘ `df_cm`: the dataframe of the confusion matrix created in `create_confusion_matrix`

This function translate a dataframe confusion matrix to a numpy matrix confusion matrix.

4.4.4.5 Tensorboard

This sections builds a tensorboard out of the log files.

4.4.5 Packages

- `os`
- `re`
- `cv2`
- `numpy`
- `json`
- `argparse`
- `scipy.io`
- `scipy.misc`

- glob
- numpy
- pandas
- PIL
- math
- tensorflow
- itertools
- copy
- xml.etree.ElementTree
- lxml.etree
- matplotlib and within this package Rectangle, imshow and text
- Image from IPython.display
- Path from pathlib
- confusion_matrix from sklearn.metrics
- keras and many enlisted subfunctions found in the Download Packages section
- imageai.Detection and many enlisted subfunctions found in the Download Packages section

5. Personal Experience

This experience has been very useful when it comes to defining what I would want to do as a future career. Before I started this project, I was quite convinced I wanted to go into artificial intelligence because of its cool applications and presumable close connection to mathematics. Also, a lot of friends of mine were going into deep learning and seemed to like it. For that reason, I chose to do this internship.

I expected to constantly have to come up with algorithms and techniques to implement certain theory. While watching the Coursera course, I was already wondering how I could implement convolutions and a non-max suppression.

As soon as I started building the model, it was clear that I would mainly be spending my time trying to get the ImageAI implementation running and that I would mostly be jumping from error to error. I did not really like this, as it did not ask a lot of my creativity, but more of me getting into documentation. I am not too good at google-ing these things, and I do not really enjoy it. But to be fair, the documentation of ImageAI is very bad and it might be easier with different open source code.

Besides from that, working on an AWS computer via a terminal is really difficult. I understand that it is just something you have to learn and then it goes easily, but the path to understanding it is just not one I enjoy amounting myself to.

Lastly, I think writing documentation is really boring. As has been stated before, I really like to come up with new ideas and ways to solve problems, and writing documentation is just reviewing that idea after you're certain it's good.

On a more positive note, I did really enjoy coming up with different ways to quantify the performance and experiment with different optimization techniques for the neural network. I do think I would enjoy working with programming in the future, just not with open sourced code too much.

Besides from that, I find it really fulfilling to work on more idealist projects in a surrounding that is very research based. Some of the meetings we had were so focused on aspects of the food industry that rapidly needed to become more sustainable. I found that really inspiring.

All in all, I am very happy that I did this project. Besides from teaching me a lot about neural networks, programming, open source code and terminals, I have learned that artificial intelligence is just not something I would want a future career in, even though I would not mind working with it once for a specific project in the future. I have had a great time working for the Questionmark Foundation and collaborating with Vantage AI.

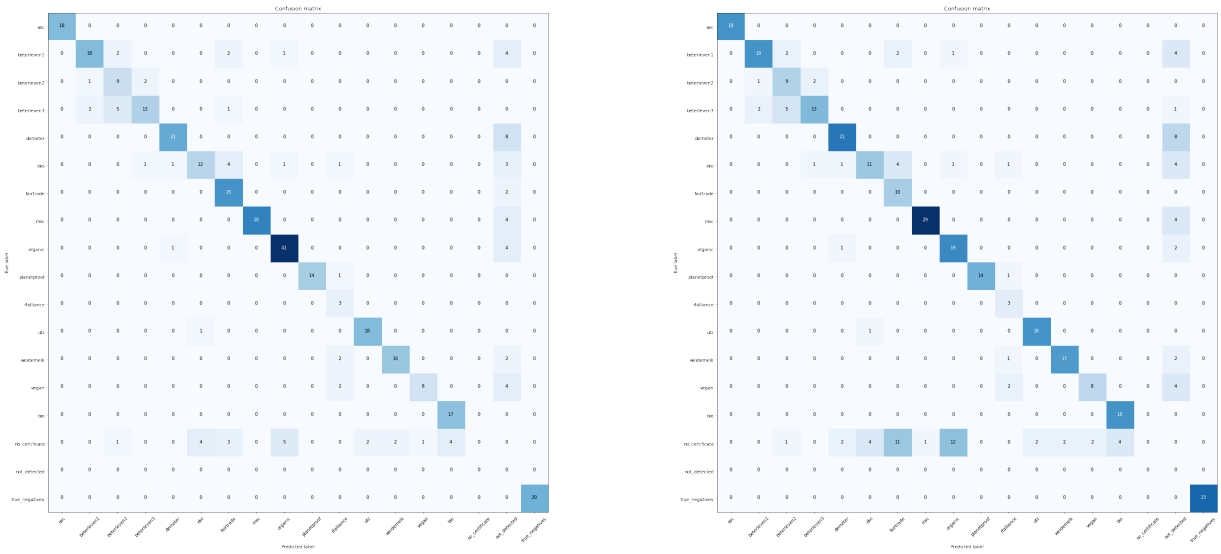
Appendix

Confusion matrices

This section contains all confusion matrices of all the different projects.

It might not be entirely clear what the labels are, as they were printed quite small. To clear this up, these are the labels in order (from top to bottom / left to right):

1. asc
2. beterleven1
3. beterleven2
4. beterleven3
5. demeter
6. eko
7. fairtrade
8. msc
9. organic
10. planetproof
11. rfalliance
12. utz
13. weidemelk
14. vegan
15. bio
16. no_certificate
17. not_detected
18. true_negatives



(a) 65 epochs

(b) 53 epochs

Figure 5.1: Confusion matrices for the Default model (3 dec).

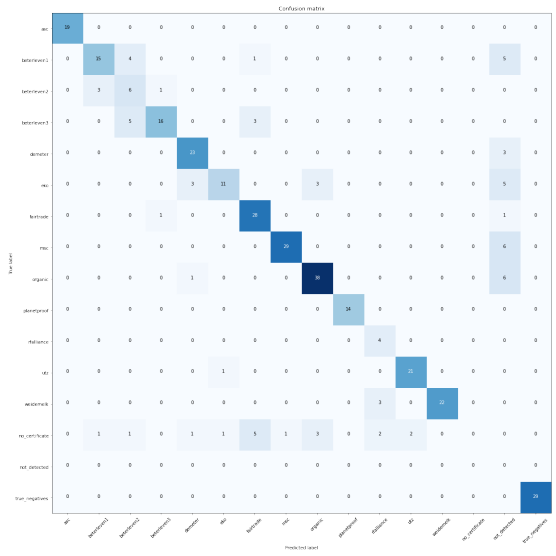
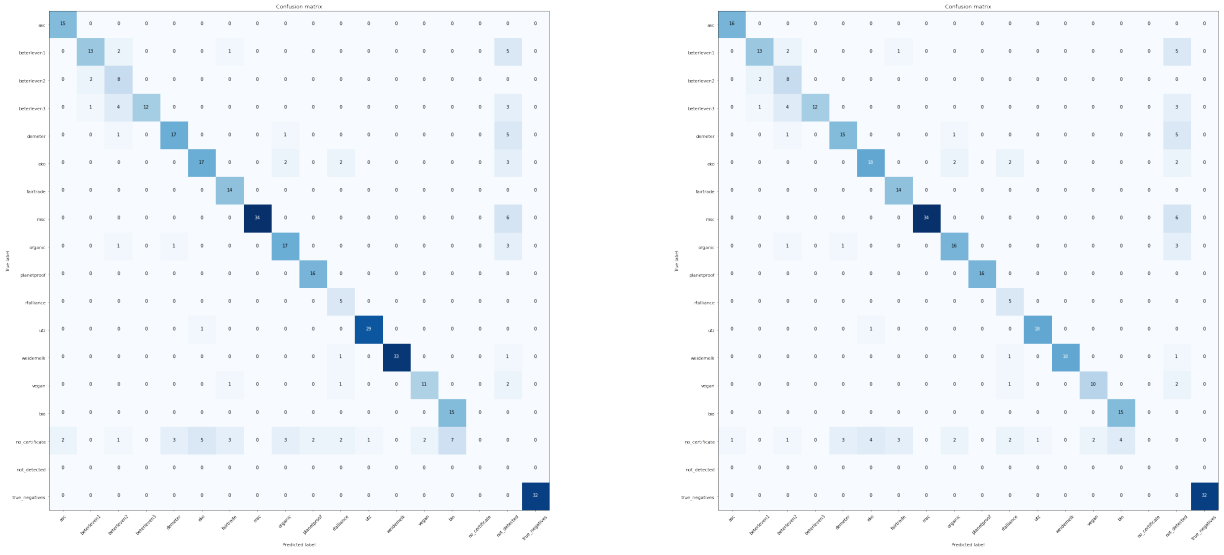


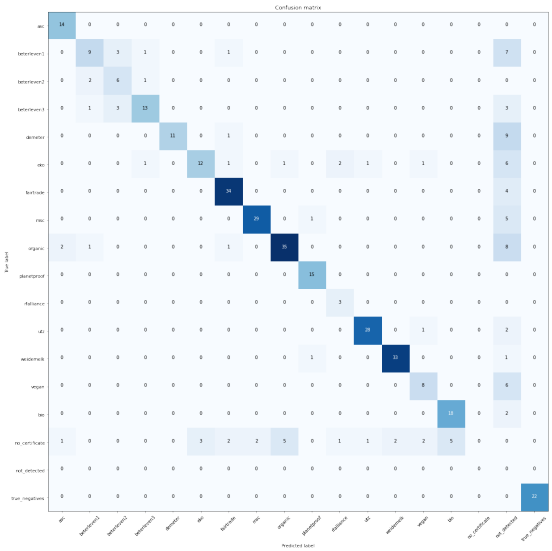
Figure 5.2: Confusion matrix for the model without AH Bio and Vegan certificates (5 dec).



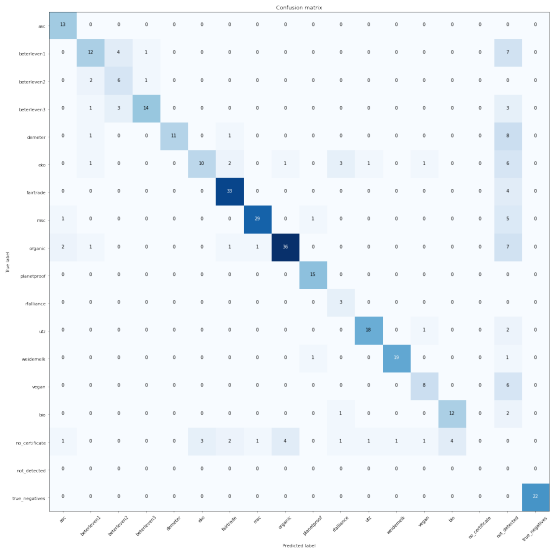
(a) 93 epochs

(b) 65 epochs

Figure 5.3: Confusion matrices for the model with Fairtrade and EU Organic certificate balanced out (6 dec)

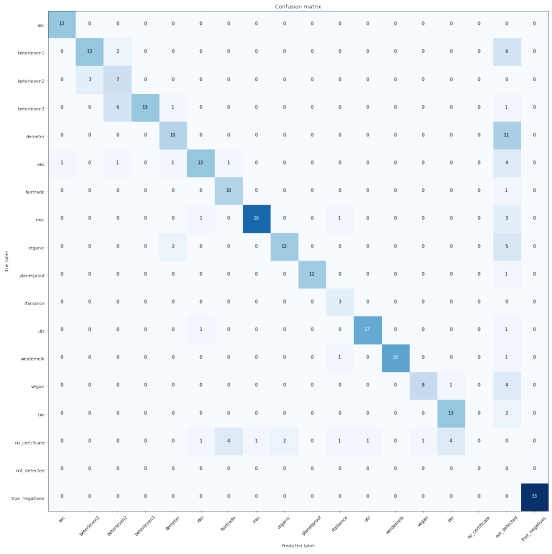


(a) 46 epochs

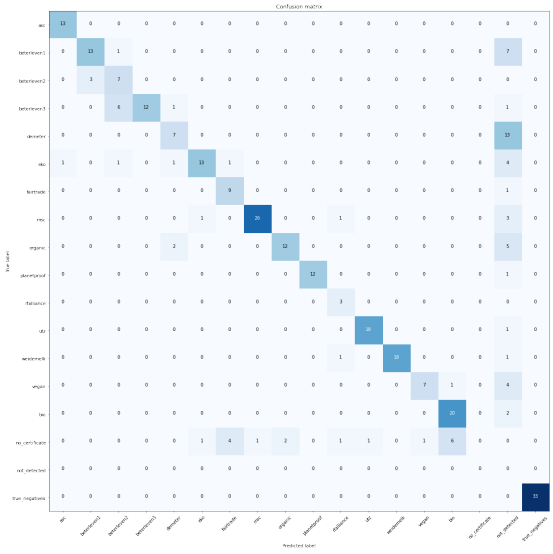


(b) 48 epochs

Figure 5.4: Confusion matrices for the model with the preprocessed data (7 dec)

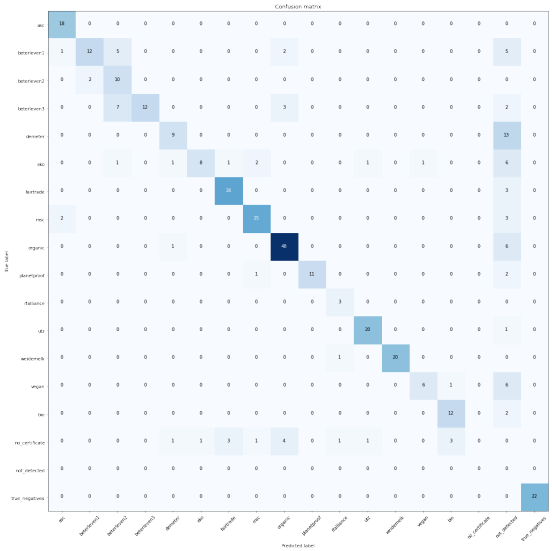


(a) 51 epochs

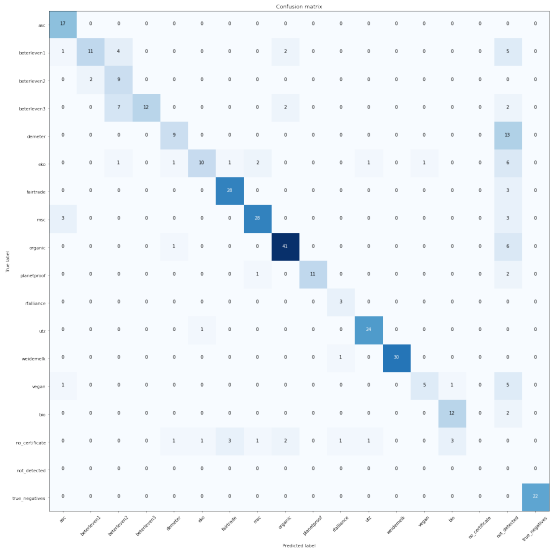


(b) 48 epochs

Figure 5.5: Confusion matrices for the model with the loss weights as (5,2,1,2) (8 dec)

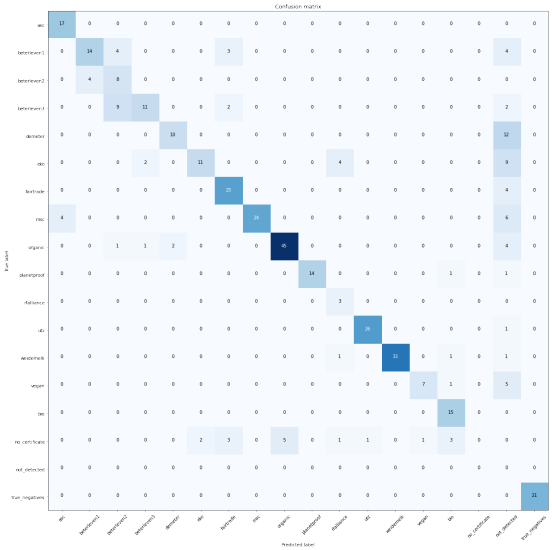


(a) 67

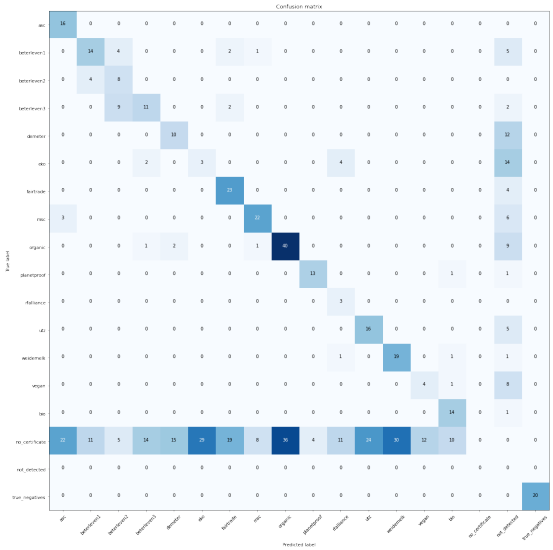


(b) 50 epochs

Figure 5.6: Confusion matrices for the model with the loss weights as (5,3,1,1) (9 dec)

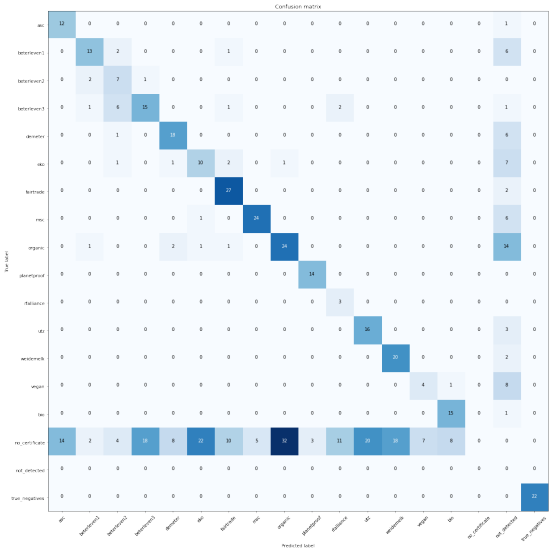


(a) 33 epochs

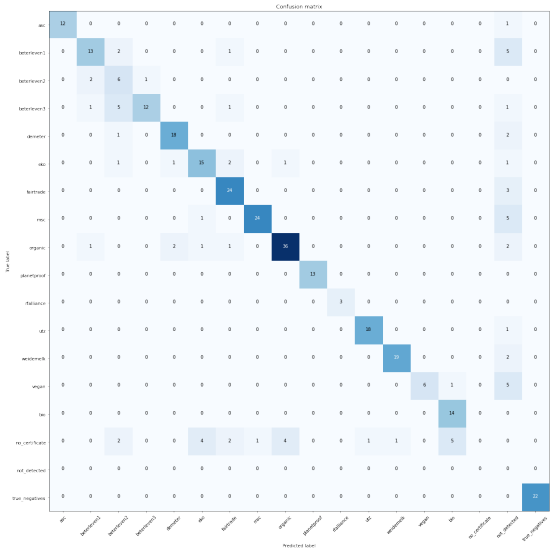


(b) 45 epochs

Figure 5.7: Confusion matrices for the model with zoom settings and ratio distortion as (0.75,1.25,0.4) (10 dec)

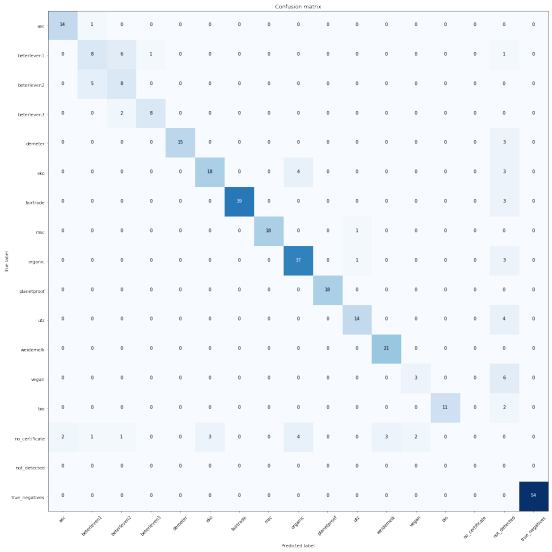


(a) 41 epochs

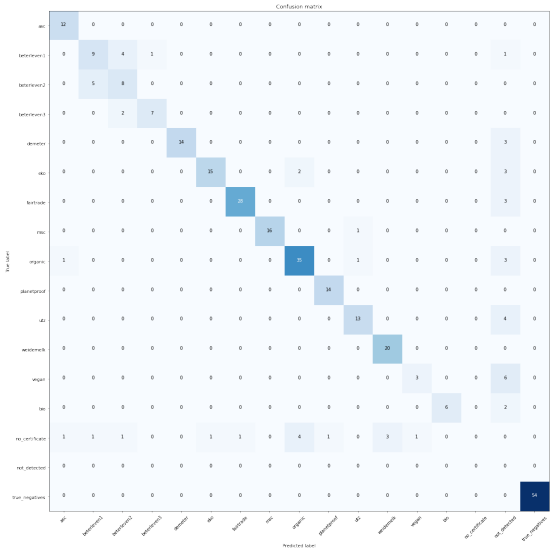


(b) 55 epochs

Figure 5.8: Confusion matrices for the model with hue, saturation and exposure as (50,3,3) (11 dec)

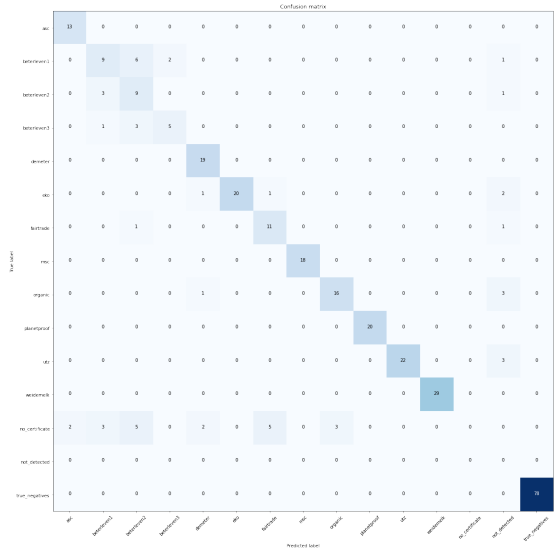


(a) 34 epochs

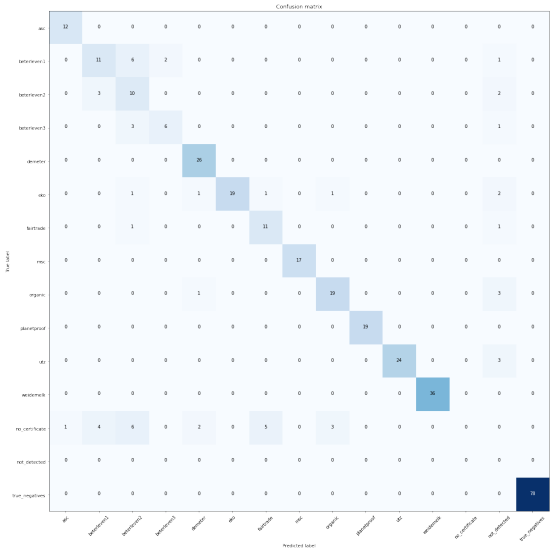


(b) 41 epochs

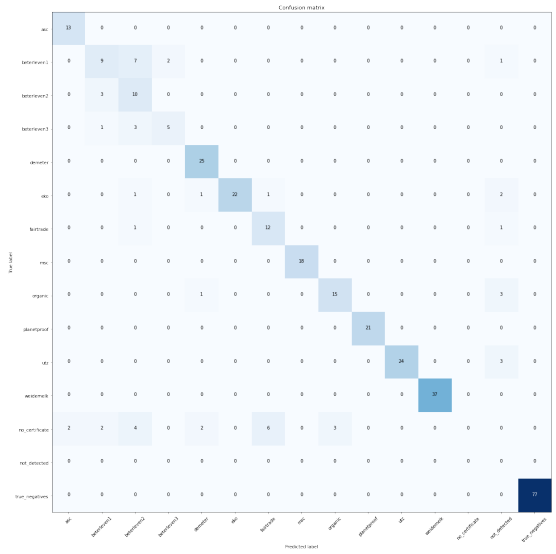
Figure 5.9: Confusion matrices for the model with more unlabeled data and without `rfalliance` (13 dec)



(a) 41 epochs



(b) 47 epochs



(c) 57 epochs

Figure 5.10: Confusion matrices for the final model