DOI: https://doi.org/10.46354/i3m.2019.emss.019

OPTICAL QUALITY CONTROL USING DEEP LEARNING

Franz Wiesinger[a], Daniel Klepatsch[b], Michael Bogner[c]

[a],[b],[c] University of Applied Sciences Upper Austria – Department of Embedded Systems Engineering
Softwarepark 11, A-4232 Hagenberg, Austria

[a]franz.wiesinger@fh-hagenberg.at, [b]daniel.klepatsch@fh-hagenberg.at, [c]michael.bogner@fh-hagenberg.at

ABSTRACT

Optical quality control is still often performed by people and always carries the risk of human error. A modern approach in order to solve this issue is the usage of artificial intelligence to boost performance and reliability.
This paper focuses on implementing a prototype for optical quality control based on the YOLOv3 algorithm. This is a state-of-the-art object detection system that uses deep learning to detect different classes of objects within an image.
Instead of different kinds of objects, the classes in this prototype were different quality levels of a strawberry. The dataset for this task was gathered by taking photos and using images from the internet. The strawberries on these images were labeled and fed to the YOLOv3 algorithm for training.
Despite the poor detection rate, the results showed that it is generally possible to use such systems for detecting different quality levels of products.

Keywords: optical quality control, artificial intelligence, neural networks, deep learning, object detection

1. INTRODUCTION

Artificial intelligence (AI) has been around since the 1960s, but in the past, applications were limited by computing power, algorithms, data or the money invested in research. This has changed over the past years. By utilizing Graphics Processing Units (GPUs) and parallel computation, the available power made it possible to solve problems of higher complexity. Due to ongoing success, funding for AI rose which led to more research and constantly improving algorithms.
Today, there is a lot of data that humans are not able to process anymore. To make use of all the collected data, artificial intelligence and machine learning is used to analyze it, try to find patterns and make predictions, based on the findings. All these systems are highly specialized on one task and are far from being general artificial intelligence. But they can outperform humans in these tasks and therefore improve speed or accuracy of certain processes.
A subfield of AI is machine vision, which enables algorithms to see the world through images and other data. This opens a new field for a broad range of applications. For example autonomous driving, object detection, text analysis as well as generative algorithms. Optical quality control is a task that is performed by humans at the conveyor belt but can be improved by utilizing machine vision.
Implementing such a system is not a trivial task though. It takes a professional approach and a lot of training data to make reliable predictions. These automated systems are meant to reduce costs and improve quality assurance. Mistakes have negative effects on both of these intended advantages. On the one hand, the decision to eliminate a good product leads to higher production costs. On the other hand, shipping a damaged product can cause a bad reputation or even endanger customers in case of unsafe electronics or delivering food. Therefore, a well-conceived and correct modeling of AI systems is required in order to accomplish this task. Tests and simulations in industrial environments will show if the intended results can be achieved.
The objective of this paper was to use a state-of-the-art object detection AI algorithm to not only identify objects within an image, but also determine their optical quality. A custom dataset of images of strawberries with varying quality was used to train the prototype. These quality levels are represented by different classes.

2. ARTIFICIAL INTELLIGENCE BASICS

In order to understand how the algorithm used in this paper and AI algorithms in general work, the following chapters will give a brief overview over some basic building blocks for artificial intelligence.

2.1. Artificial Neuron

The most basic building block of artificial intelligence algorithms is the artificial neuron. It is based on the function of natural neurons in the human brain and is a mathematical approximation that can also be implemented on electronic systems.
A natural neuron consists of three main parts: dendrites, the cell body and the axon. Signals received through the dendrites are forwarded to the cell body, where their processing takes place. After that, they are passed on to the axon, which is the communication channel to other neurons. It can split up into many branches to communicate with more than one neuron. The point

where the axon is connected to the next neurons dendrite is called a synapse. It also processes the incoming signal before forwarding it to the dendrite.
Artificial neurons are based on this simplified function and represented as the following mathematical function:

$$y=f\left(\sum_i \left(x_i w_i\right)+b\right) \quad (1)$$

The incoming signal through one dendrite is called x_i. Its first processing step is modeled by multiplying a weight to the signal, which is called w_i. The weighted signal is then forwarded to the cell body where the final processing takes places. All weighted signals are summed up and a bias b is added. This is the argument for the activation function *f(x)* (Hijazi S. et al. 2015). The most common activation function is called the Leaky Rectified Linear Unit function (Leaky ReLU) and is depicted in Figure 1.

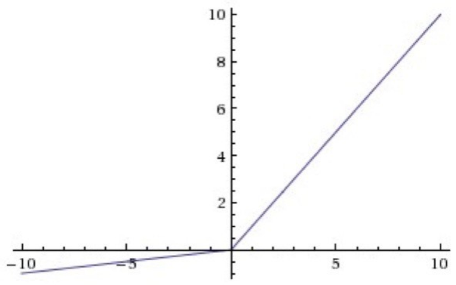

Figure 1: Leaky ReLU function.

The ability of an AI algorithm to learn things is given by adjusting the weights of the incoming signals for each neuron. Every training step calculates the error between the predicted result and the actual result, the ground truth, and the weights of each neuron is modified according to its influence on the error.

2.2. Neural Networks

While neurons in the human brain are connected to other neurons, artificial neural networks are structured in layers, which contain neurons that are not connected to other neurons in the same layer. Basically, there are three types of layers: an input layer, hidden layers and an output layer.
The input layer contains the information about the input data. For example, every neuron could contain the information of one pixel from an image. Usually there is at least one hidden layer, which is used to increase the capacity of the network and the ability to learn more complex functions. In a fully-connected layer, the output of every single neuron is connected to all neuron inputs of the next layer. The output layer determines the classification result (Hijazi S. et al. 2015).
A classification task, for example, would be if a neural network has to determine, if there is a dog or a cat in an image. The input layer would contain the pixels from the image, the hidden layers are required to learn certain characteristics of the images and the output layer would contain two neurons, one for a cat and one for a dog. The network then propagates the input data signals through the network and outputs confidence values to the output neurons. These values are interpreted as how confident the network is whether there is a cat or a dog in the image. This kind of network is depicted in Figure 2.

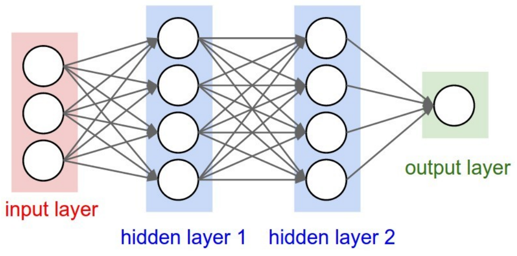

Figure 2: Fully-connected neural network.

2.2.1. Learning Procedure

A neural network is able to learn certain tasks by adjusting the weights of all neurons. In every learning step the output of the network and the ground truth are compared by using a loss function. This function can also take different aspects of the result into consideration and weigh their error value differently. That way, a network can be told what parts of the result are more important to optimize than others.
The target of the learning procedure is to minimize the result of the loss function. The most common way is to use the gradient descent algorithm, which is used to find a local minimum in a function by always going in the direction of the steepest negative gradient. To apply this method to a neural network an additional technique is required: backpropagation.
This method allows the training algorithm to calculate the impact of each neurons weights on the resulting loss by using partial derivation which leads to the gradient. These calculations are used to determine whether the weights are increased or decreased. A parameter called learning rate determines the size of the in- or decrease. If the learning rate is too small, the algorithm might take a very long time to get acceptable results. On the other hand, if it is too large, the optimal result might not be found due to too large changes in the weights (Gschwend D. 2016).

2.3. Deep Learning

Deep learning is a subfield of artificial intelligence and consists of networks with a more complex structure than simple neural networks. One difference is that common networks are rather shallow in terms of number of layers. They often have less than ten layers. In comparison, deep learning networks have up to 1000 layers which increases the capacity of the network as well as the complexity of the task the network can learn. Complex tasks like machine vision and natural language recognition require such deep networks.
The fundamental concept of deep learning algorithms is the idea that the input data have underlying representations, which is composed of various levels of

abstraction. By varying the number of layers, different levels of abstraction can be extracted from the training material which may lead to better accuracy of the network.

2.3.1. Convolutional Neural Network

One of the most important types of deep learning networks is the convolutional neural network (CNN). It is inspired by the way the visual cortex in the brain works and is therefore very much suited for performing image or video processing tasks.
Instead of fully-connected layers with single neurons, the CNN relies heavily on convolutional layers. Each convolutional layer consists of one or more filters, called kernels. A very common example is the Sobel operator which is used for horizontal or vertical edge detection in images. The kernel and the input image are convoluted which results in a two-dimensional feature map. Every kernel in a convolutional layer produces one feature map as output and has usually two or three dimensions, depending on the input data of the current layer. All outputs are stacked and result in a three-dimensional output which can be used by the next layer. Instead of changing the weights of each neuron, the values in the kernels are altered during learning. This results in a great reduction in memory usage.
Pooling layers are used to reduce the dimensions of input data by taking a two-by-two or three-by-three subsets and reduce it to one value by taking the maximum (max-pooling) or the average value (average-pooling).
Fully-connected layers are often used at the end of CNNs where the feature maps are used to perform a classification.
Dropout layers are similar to fully-connected layers and consist of neurons. The only difference is, that during each training step, a certain amount of neurons are deactivated. This helps to prevent the problem of overfitting, where the network memorized all the training data and their solution but is unable to perform well on unseen data (Gschwend D. 2016).

2.4. Types of Algorithms

When it comes to image processing with a neural network there are basically four different tasks that can be performed, which are depicted in Figure 3.

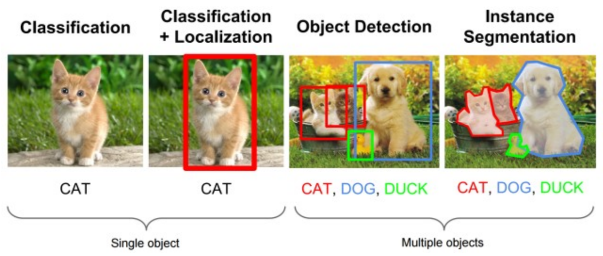

Figure 3: Different types of image processing algorithms.

The classification task is used on images which contain one object. The algorithm tells the user to which of the predefined classes the depicted object belongs to.
Localization extends the classification task by additionally drawing a frame around the object in the image.
Object detection is similar to localization but there is a variable number of objects in the image. The algorithm has to localize the objects and perform a classification on each one (Ng A. 2017).
Segmentation extends the object detection by not only drawing a frame around the object but draw its silhouette.
The prototype in this paper uses an algorithm for object detection since segmentation is significantly more complex but offers no major benefits.

2.5. Quality of Results

To determine the quality of the results, the total loss is insufficient due to problems like overfitting. The loss might be at a minimum but the algorithm does not perform well on unseen data, it just memorized the training data. Therefore, another measure of quality has to be established. This is the mean average precision (mAP). Before calculating the mAP, four result states have to be defined:

- True Positive (TP)
- False Positive (FP)
- True Negative (TN)
- False Negative (FN)

True positive means, that there was a detection and it was right. False positive means, that there was a detection, but it was wrong. True negative means that there was no detection and that this is right. False negative means that there was no detection, but there should have been. A detection is correct if the intersection over union (IoU) between the predicted object and the ground truth, depicted in Figure 4, exceeds a defined threshold.

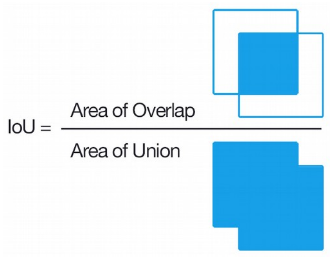

Figure 4: Intersection over union.

Now the precision and the recall of the algorithm can be calculated:

$$precision=\frac{TP}{TP+FP} \tag{2}$$

$$recall=\frac{TP}{TP+FN} \tag{3}$$

By varying the confidence threshold for the object classification, many precision and recall values can be calculated. These are then plotted in the Precision-Recall (PR) curve, where the recall is on the horizontal axis and the precision is on the vertical axis. The integral over that plot is somewhere between 0 and 1 and is called the average precision (AP). To get the mAP, several APs have to be calculated with different IoU thresholds.
The final mAP shows the actual quality of the results the algorithm produces (Henderson P., Ferrari V. 2016).

3. PROTOTYPE

The goal of this paper is to show the possibilities of using a CNN for image processing tasks like optical quality control. Instead of building a custom network, a well established algorithm called You Only Look Once (YOLO) was used. In the original paper it performed very well in comparison to other object detection algorithms of that time in speed as well as accuracy. All comparisons are based on the publicly available "*Common Objects in Context"* (COCO) and Pascal "*Visual Object Classes"* (VOC) image datasets (Redmon J. et al. 2016).
The working environment on the hardware side was as follows:

- Intel Core i5-8400 CPU
- 16 GB RAM
- Gigabyte Nvidia GTX 980 GPU

The software environment consisted of a Windows 10, 64-Bit operating system and the Darknet neural network framework, which was created by the inventor of the YOLO algorithm, Joseph Redmon. Since the framework is not compatible with Windows, an adjusted fork of it was used. This was created by the GitHub-User AlexeyAB. To accelerate the training phase, the Nvidia CUDA and cuDNN techniques were used. These allow the software to speed up all calculation tasks by utilizing the GPU. For generating the image labeling, a tool named Yolo_mark was used. This tool was also created by AlexeyAB.
The actual task, the prototype is supposed to perform, is the optical quality control of strawberries. The quality is categorized in six different classes:

- unripe
- partially ripe
- ripe
- lightly damaged
- heavily damaged
- rotten

If strawberries are totally white, they count as unripe. As soon as there is any red color on the fruit, it is counted as partially ripe. Fruits are ripe, if they are all red and no damage can be found. If there is any light damage or any dry spot, it counts as lightly damaged. Are the strawberries any more than lightly damaged, they count as heavily damaged. As soon as the fruit has any brown spot or signs of mold, it counts as rotten.

3.1. YOLO algorithm

The YOLO algorithm is faster than other systems because it does not use complex pipelines. The deformable parts model, for example, slides a window over the image and performs a classification of the current content of the window at each step. This has to be done in several sizes all over the image. YOLO avoids this by just looking at the image once via the CNN and predicting the objects in it. This also has the advantage, that the algorithm learns to include contextual information, like the background, into the decision making.
The concept of the whole algorithm is, that the input image is split into a S-by-S grid. Each cell is responsible for predicting B frames of objects, which center points are in that cell. In addition to that, each cell predicts C object classes. This means, that each cell outputs two frames and one set of class probabilities. So only one object class can exist in one cell. Before all that, a 24 convolutional layers deep CNN performs the feature extraction and uses several fully-connected layers at the end to map the features to the actual values that represent the object frames and classes. S, B and C can be chosen by the user. A method called non-max suppression is used in post processing to remove low-confidence or overlapping object frames.
The loss function that is going to be optimized during training is the commonly used least squares method but with a few tweaks. First, the error for predicting frames is weighted equally to the class probability. This was changed by giving the error for frame prediction a higher weight, so this becomes more important. Second, errors in predicting the frame size had the same impact on small and big frames. This was changed by predicting the square root of the frame, so the same offset has more impact on small frames than on larger ones (Redmon J. et al. 2016).

3.1.1. YOLOv2

YOLO received two incremental improvements where YOLOv2 was the first one. The accuracy was increased by 15%, the reduction to 19 convolutional layers and the removal of all fully-connected layers made it significantly faster and a combination of classification and detection training made it stronger.
A major change was the change from predicting the whole object frame to predicting offsets to reference frames, called anchor frames. These anchor frames are predefined by the user and their proportions should match the most frequent objects in the dataset. This helps the algorithm while training. This can be automated by a dimension clustering which uses a k-means algorithm to find the optimal proportions.
Additionally, the training of the network was performed with varying image scales. This is possible, since all fully-connected layers were removed. Small scales boost the speed of computation and high scales improve

the accuracy. The variation improves the overall quality, since the network learns to predict objects on several different scales (Redmon J. et al. 2017).

3.1.2. YOLOv3

The third iteration of the YOLO algorithm brings three major changes. Instead of 19 convolutional layers, YOLOv3 uses another base CNN with 53 convolutional layers and 23 residual blocks. In theory, the deeper a network gets, the harder it is to train the whole network. In the worst case, a layer learns its identity function and just forwards the input data instead of having a negative impact on the result. Practical attempts show though, that this negative impact does occur. To solve the problem, residual blocks were introduced which routes a bypass over some layers and sums up the last layers output and the input data.

$$y=F(x)+x \tag{4}$$

That way, the layers only have to learn to set their outputs to zero instead of the identity function.
The second change addresses the class confidence value. In YOLOv2 a softmax function was used to normalize the confidence for all classes for one frame and to pick the best result as detection. Now, a logistic function is used on each confidence value. This makes it possible to determine several classes per frame.
The third improvement is the frame prediction over different scales. While its predecessors directly predicted the object frames, this version predicts only three of the nine frames at the same point. This is now followed by a combination of convolutional layers, residual blocks and an upscaling block, which leads the higher resolution feature maps. After that, the next three frames are predicted and this whole part is repeated once more. This leads to an increase in accuracy, especially on small objects (Redmon J. et al. 2018).

3.2. Dataset generation

The most important part about training an AI is the dataset that is used. The best algorithm is only as good, as the dataset it gets to study.
For this prototype, the whole dataset was acquired in three steps. Custom photos were taken in the first two steps and images from the internet where used during the third step.

3.2.1. Custom photos

A pack of strawberries were picked up at the super market and left alone for four full days. After that time, the fruits had major flaws and also showed signs of mold. The test setup for taking the images consisted of a bright lamp, a white sheet of paper to get a uniform background and a small turntable to spin the fruits around and take images from different angles. 202 photos were taken with this indoor setup using a Samsung Galaxy A5 (2017) smartphone camera. An example image is depicted in Figure 5.

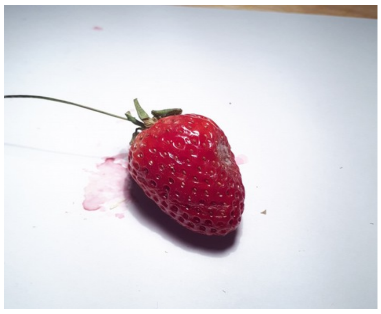
Figure 5: Photo from the first part of the dataset.

For the second step, the current strawberries were left alone for another three days and then a pack with fresh, partially unripe fruits was added. The setup was taken outdoor because using the sunlight led to better quality images. This time, 322 photos were taken by a Canon EOS 600D single-lens reflex camera. An example is depicted in Figure 6.

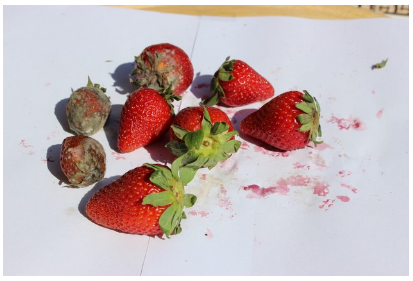
Figure 6: Photo from the second part of the dataset.

3.2.2. Additional image data

To increase the dataset, additional images from the Google Images were taken. A tool called Google Images Download from GitHub allows it, to automatically download a specific number of images based on a search phrase. For each of the following phrases, 250 images were downloaded:

- new strawberries
- young strawberries
- strawberry
- strawberries
- bad strawberries
- old strawberries
- rotten strawberries
- damaged strawberries

These 2000 images were then numbered and sorted out. Similar images, images with a too low resolution, bad quality images and all non-real pictures (like cartoon strawberries) were deleted. This led to a large dataset with images of strawberries that would be labeled and eventually fed into the algorithm for training.

3.2.3. Labeling

The labeling of the dataset was the most time consuming part in creating the prototype. Every image and every object had to be labeled by hand. This means that a box, which is associated to a pre-defined class,

has to be drawn over every object in an image. The algorithm then uses the labeling data (position, size and class of the box) as ground truth for training the neural network. It takes a training image, tries to predict its objects and classes and compares the result to the ground truth to improve the prediction.
As already mentioned, this task was accomplished by using the Yolo_mark tool. At first, the user defines the classes that the dataset contains. After that, the configuration and a directory containing the training images is passed to the tool. It then shows the user the first image in that folder and a slider to change the displayed image. With the number keys or a second slider, the user can choose a class for which he wants to draw a box on the image. By using the arrow keys, the next or previous image can be displayed.
This way, ever strawberry in the training images was labeled with the corresponding class and box. A screenshot of the tool is depicted in Figure 7.

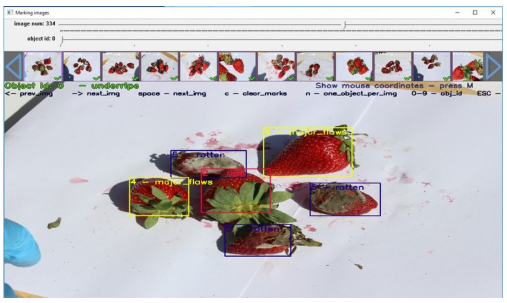

Figure 7: Screenshot of Yolo_mark.

3.2.4. Splitting the dataset

After the labeling was completed, the dataset had to be split into three parts: training data, validation data and testing data. The training data is used by the network to learn how the objects look like and what shapes their frames have. Validation data is used to test the networks performance on unseen data. Based on the validation results, the best model is picked. Test data is then used on this model to determine the actual accuracy. Validation and test data have to be totally independent from the training data.
To split the dataset, a Python script was created. The user passes the folder with all images and the percentages for each part. It then randomly creates the three parts where the images are split according to the percentages. Afterwards, it checks if each part has roughly the same percentage of each object class as the other parts. If these percentages do not match within a user-defined margin, it recreates the three parts with new random picks. This is repeated until the object percentages are within the margin or the user aborts the script. The best split up to that point is then used.

3.3. Training approach

For training the network, an iterative process is used. First, the network is trained by using built-in functions of the Darknet framework. Several parameters like the path to the folder with the training data and the number of batches are passed to the program.
It then executes the training until it completes or the user interrupts the training. Every 100 iterations, the current state of the network is saved so that progress is not lost in case of an error and that the most accurate network can be chosen afterwards.
After the training the results are analyzed. If they are of insufficient accuracy, alternations and optimizations have to be made on the setup. This includes common methods of optimization like increasing the dataset, introduce additional dropout layers or artificially increase the dataset by reusing old images and perform modifications on them.
After the modification of the setup, the training is restarted and the results are once more analyzed.

4. RESULTS

Due to timing limitations, only a total of ten training iterations were executed. The changes and results of every iteration are described in this chapter. Only iterations one and nine are accompanied by figures, because the results did not improve between these two. Therefore, the figures of iteration one show what the general results look like and the figure of iteration nine shows the first improvement in accuracy.
For the first training iteration, the dataset was split in 90% training data and 10% validation data and a standard YOLOv3 network with no modifications was used. The training batch size was 64, which means that the loss of 64 images is averaged and backpropagated. In the following notation a training iteration corresponds to the processing of one batch and a system iteration corresponds to one prototype training iteration. The training took place with pre-trained YOLOv3 weights, which should increase the accuracy of the network. Instead of using a steady learning rate, the default training algorithm also uses a burn-in learning rate, which is depicted in Figure 8.

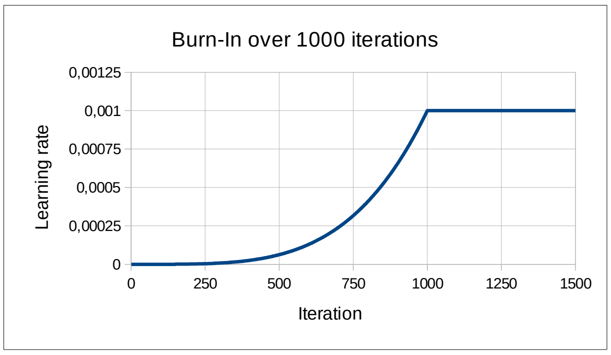

Figure 8: Burn-in learning rate of YOLOv3.

The total loss steadily decreased over the total of 30,000 training iterations, which is depicted in Figure 9 and Figure 10. At first, the loss showed a big variation since the network had no idea of what is depicted on the training images. After about 100 iterations a downward trend established whose slope steadily decreased afterwards. After 30,000 iterations, no significant decrease in loss could be determined and the training was considered to be finished.

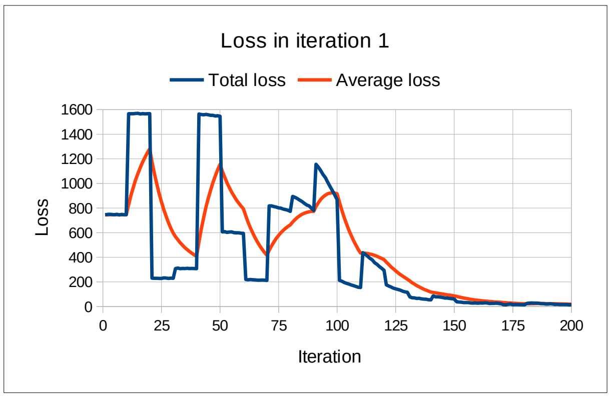

Figure 9: Total and average loss between 0 and 200 iterations.

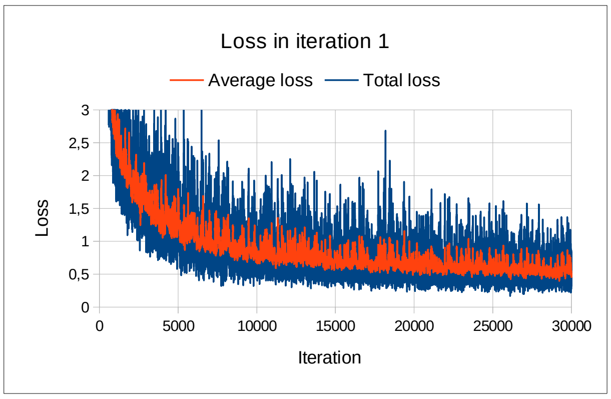

Figure 10: Total loss and average loss over 30,000 iterations.

During the training, the framework stores the weights, which represent the current state of the network every 100 training iterations. These states are then used to calculate the mAP for every 100th iteration. Afterwards, a curve can be drawn to compare the the accuracy of the model using training data and test data. The results of the first system iteration, depicted in Figure 11, show that the mAP curves differ a lot between the training data and the testing data. While the network memorized most of the training data with a maximum mAP of 90%, it had problems identifying objects on unseen images. The highest mAP was about 45%. This means that the network is heavily overfitting.

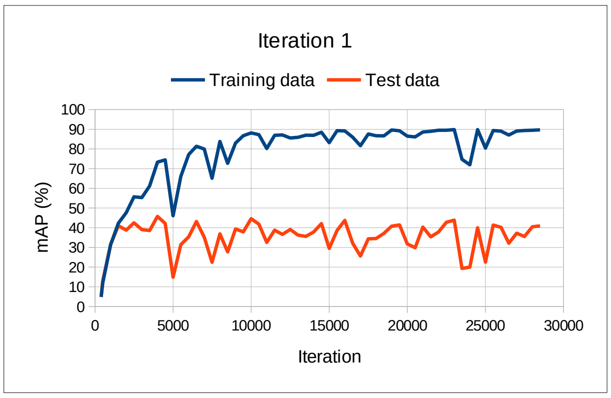

Figure 11: mAP comparison between training data and test data of the first system iteration.

To prevent this problem, an artificial increase of the dataset was performed for the second system iteration. Following techniques were used:

- Image scaling between 75% and 125% per axis
- Moving the image between -40% and +40% per axis
- Rotation between -8° and +8°
- Shearing with an angle between -8° and +8°
- Gaussian blur
- Contrast modification between -20% and +30%

For each original image in the dataset, 20 images were generated. Each image was created with a set of randomized parameters from the above listed. In addition to the artificial increase, the dataset was extended by 100 images from the internet which did not contain any strawberries.

Despite the modification, the mAP was roughly the same as in the first system iteration.

The third system iteration introduced the dimension clustering for the anchor frames, mentioned in the YOLOv2 algorithm. Additionally, less pre-trained layers were used to determine their effect. Only 61 instead of 74 were used.

After 10,000 training iterations, the results were worse than in the second system iteration. The mAP over the training data was only at 70% and the mAP over the validation data was around 40%.

For the fourth system iteration, 74 pre-trained layer weights were used again. Instead, the training on multiple input scales were disabled and the batch size was reduce from 64 to 32.

The results after 10,000 training iterations are the same as the ones from the third iteration.

The Darknet framework features an internal random data augmentation. Brightness and scaling randomization was increased for the fifth system iteration.

This resulted in even worse results, the validation mAP was again around 40% but the training mAP did not exceed 50%.

The sixth system iteration took a different approach. It took the weight from the fifth iteration at 8,000 training iterations. At that point, both the training and the validation mAP were around 40%. Then, the first 80 of the total 107 layers were frozen, so that the weights were not updated. The remaining 27 layers were trained like before. Again, the results did not improve, both mAPs stopped increasing at about 45%.

System iteration seven used a compact YOLOv3 model, the Tiny-YOLOv3, which consists of a total of 23 layers. The eighth system iteration halved the learning rate. Both iterations did not show any improvement.

For the ninth iteration, a modification to the dataset was used. The total of six different quality classes were reduced to just three. Unripe and partly ripe were merged as well as all three damaged classes.

For the first time, an increase in the mAP was achieved. While the training mAP rose up to 95% after 20,000 training iterations, the validation mAP reached up to 65% at one point, depicted in Figure 12. This was an increase of about 15% compared to all previous iterations. An example for what an output looks like is depicted in Figure 13.

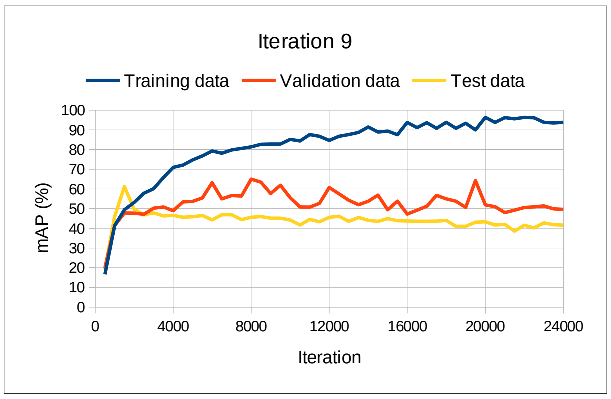

Figure 12: mAP comparison between training data, validation data and test data of the ninth system iteration.

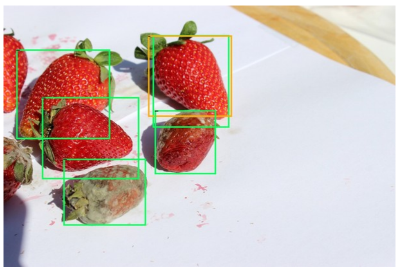

Figure 13: Example of a network prediction after the ninth system iteration. Green frames mark a ripe strawberries and orange frames mark lightly damaged strawberries.

With the additional data augmentation in the tenth system iteration, both mAPs dropped for 5% in comparison to the ninth iteration.
To summarize all results, Figure 14 depicts the comparison of the best validation mAPs of every system iteration. It clearly shows that common methods of optimization did not improve the accuracy but reduction of object classes did.

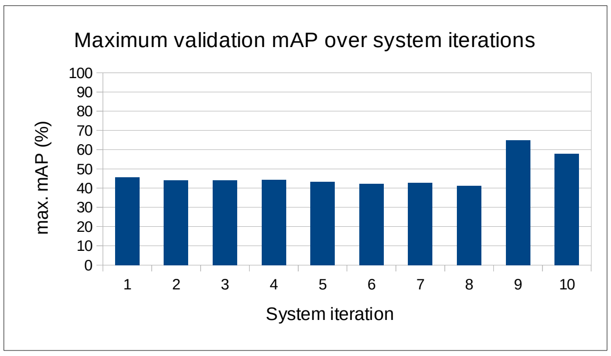

Figure 14: Comparison of the best validation mAPs of every system iteration.

5. CONCLUSION

Based on the ten system iterations performed with the prototype, several conclusions can be made.
Generally, there is no universal optimization technique that instantly boosts the accuracy of a network. It always depends on the task, that should be achieved and many other factors. Optimizing a neural network is done by fine-tuning certain parameter to achieve the best possible result under the given circumstances.
Since YOLOv3 is a reliable algorithm out of the box, there was no need for restructuring the whole network or adding different layers. The opposite can be said about the Darknet framework, which was created by the inventor of the YOLO algorithm. On the one hand, the framework is open source and accessible for everyone. On the other hand, it is lacking a documentation and source code commentary. This makes it difficult to use the framework. Especially if someone's intention would be to implement a custom network, since the functionality of each available layer has to be looked up in the source code.
The results in this paper show, that the dataset has the most impact and has to be treated with a lot of care. Most likely, inconsistent labeling of the dataset caused rather bad results in terms of accuracy. It was not clear enough defined, whether a strawberry is not, lightly or heavily damaged.

Finally, it can be said that artificial intelligence is generally able to distinct different object from each other. As seen in many other demonstrations, machine vision is becoming better and better at recognizing objects. It comes down to the modelling of the system, which enables a distinction of the same object in different quality levels. This mainly includes the well-conceived creation of the dataset used to train a network. Tests under industrial production conditions show the quality of the system and help the engineers to incrementally improve the system. Also, a reduction to two quality levels is often sufficient and can boost the performance. Companies already collected huge amounts of data about their products which they can use to implement automated quality control systems and profit from their advantages like reliability or cost effectiveness.

REFERENCES

Hijazi S. et al., 2015. Using convolutional neural networks for image recognition. Available from: https://ip.cadence.com/uploads/901/cnn_wp-pdf [accessed 8 April 2019]

Redmon J. et al., 2016. You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, 779-788. June 8-10, Boston, Massachusetts.

Redmon J. et al., 2017. YOLO9000: better, faster, stronger. Proceedings of the IEEE conference on computer vision and pattern recognition, 7263-7271. July 22-25, Honolulu, Hawaii.

Redmon J. et al., 2018. YOLOv3: An Incremental Improvement. Available from: https://arxiv.org/pdf/1804.02767.pdf [accessed 8 April 2019]

Gschwend D., 2016. ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network. Master Thesis. ETH Zürich.

Ng A., 2017. C4W3L01 Object Localization. Available from: https://www.youtube.com/watch?v=GSwYGkTfOKk [accessed 8 April 2019]

Henderson P., Ferrari V., 2016. End-to-end training of object class detectors for mean average precision. Proceedings of the Asian Conference on Computer Vision, 198-213. November 21-23, Taipei, Taiwan.