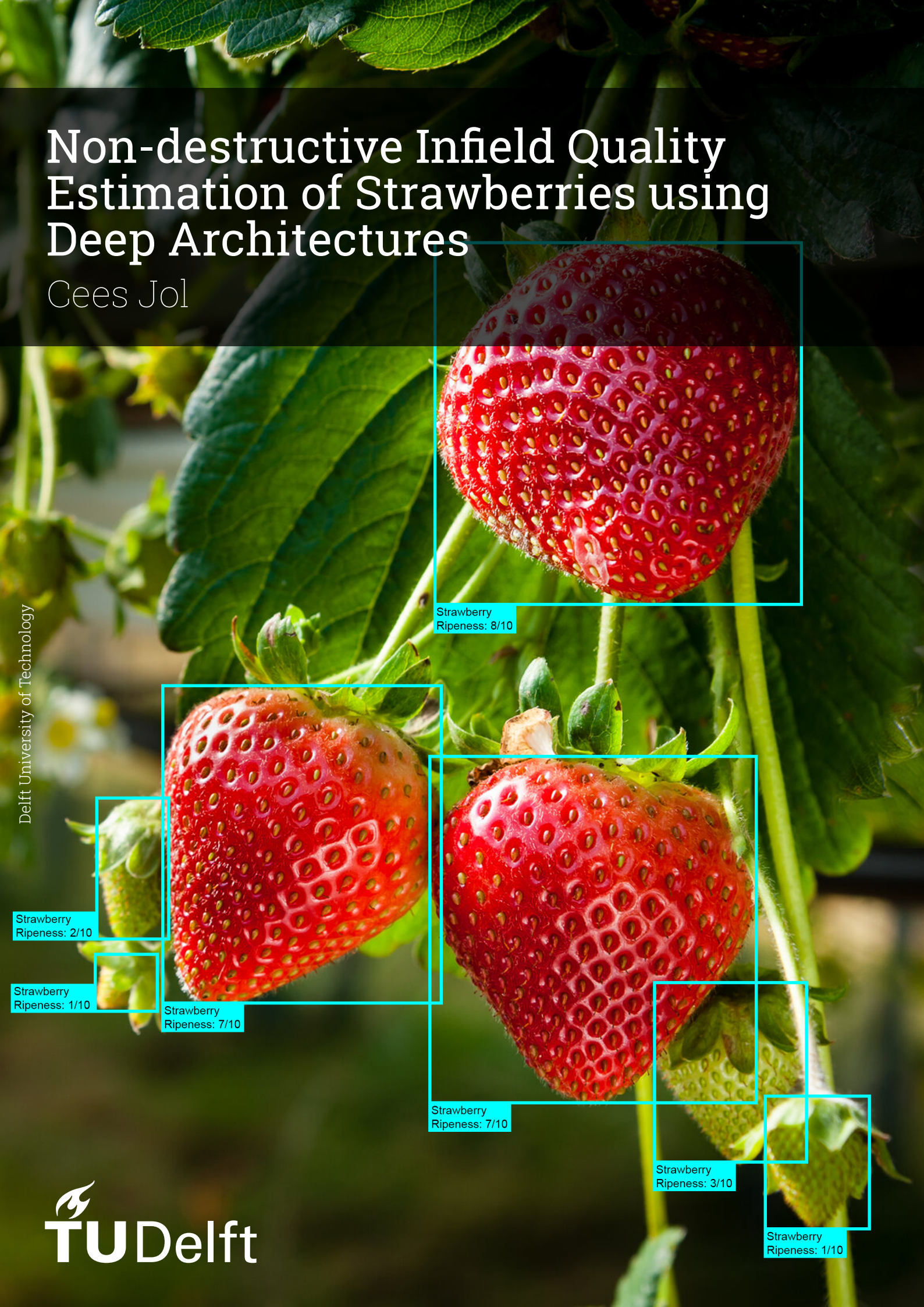
Non-destructive Infield Quality Estimation of Strawberries using Deep Architectures

Cees Jol



Delft University of Technology

TU Delft

Non-destructive Infield Quality Estimation of Strawberries using Deep Architectures

by

Cees Jol

submitted in partial fulfillment of the requirements for the degree of

Master of Science

at the Delft University of Technology.

Student number:	4653904
Thesis advisor:	Dr. Jan C. van Gemert
Daily co-supervisor:	Junhan Wen
Project Duration:	April, 2022 - November, 2022
Institution:	Delft University of Technology
Faculty:	Faculty of Electrical Engineering, Mathematics & Computer Science
Research group:	Pattern Recognition and Bioinformatics
Lab:	Computer Vision

Preface

This report describes the work I performed for my thesis, the final part of the Master of Science in Computer Science at the Delft University of Technology.

I would first like to thank my daily co-supervisor, Junhan Wen, who provided me with guidance throughout the project. I received countless new ideas and useful feedback during the meetings. These have helped to steer the project in the direction it did. Second, I would like to thank my thesis advisor, Jan van Gemert, for helping me with approaches towards problems and for providing ideas during the project. Finally, I would like to thank my friends and family for proposing new ideas and asking thorough questions.

Cees Jol
Delft, November 2022

Contents

1

Introduction

Strawberries are non climacteric fruit, which means that they do not continue to ripen after harvesting. Further, they have short shelf-life time and thus need to be harvested at the right time to reduce waste. Quality attributes such as sweetness can help to indicate optimal harvesting time. Classic methods to measure quality are labor-intensive and possibly destructive. Computer vision methods are a promising solution that are automated and non-destructive.

Quality attributes can be used to determine proper harvesting moments of strawberries. As a strawberry ripens, sweetness increases and firmness decreases. Computer vision methods are able to analyze strawberries without the need to manually inspect them. Quality attributes can be estimated based on observations about the strawberry, such as color and texture [32]. We use deep learning architectures to try to predict quality attributes.

Aside from quality, we aim to estimate strawberry size. The markets connected to our greenhouse separate strawberries by size, specifically, their width in millimeters. Strawberries placed in the wrong market could lead to waste. As we estimate size infield, it is not trivial to estimate the size of a strawberry: some strawberries are further away than others, which means we cannot map the number of pixels directly to the width of a strawberry. In this report, we outline a stereo vision method to estimate the size automatically.

Since pictures are taken infield, strawberries can be occluded, *e.g.*, covered by other fruits or leaves. To properly estimate the width of occluded strawberries, we use an image inpainting algorithm, which aims to recover the original shape.

The rest of the report is structured as follows. Chapter 2 is a scientific paper that describes the thesis. The chapters that follow provide a background on certain topics. First, chapter 3 gives a background on deep learning. Then, chapter 4 gives a background on strawberry quality evaluation. Finally, chapter 5 gives a background on stereo vision.

2

Scientific paper

Non-destructive Infield Quality Estimation of Strawberries using Deep Architectures

Cees Jol[1], Junhan Wen[2], Jan van Gemert[1]
[1]Computer Vision Lab, [2]Algorithmics Group
Delft University of Technology

Abstract

Strawberries have a short shelf-life time and thus need to be harvested at the right time to reduce waste. To this end, information about quality attributes is useful. Recently, many computer vision methods have been proposed. Most literature analyzes postharvest, which means that strawberries can only be analyzed after harvesting. As a result, these methods cannot be used to find a good timing to harvest. We analyze strawberries preharvest, so that we can analyze until we find a good timing to harvest. We show that predicting ripeness, sweetness, and firmness of strawberries is possible infield. Further, we analyze strawberry size to find a fitting market. Since we analyze size infield, we find two challenges: occlusions and lack of depth information. We perform inpainting to try to recover the original shape. Results are good on artificial occlusions, but varying on real occlusions as it is difficult to adapt to all kinds of occlusions. We use stereo vision and depth estimation to estimate size. Stereo vision improves size estimation slightly.

1. Introduction

Strawberries are non climacteric fruits, which means that they do not continue to ripen after harvesting. Further, they have a short shelf-life time. Underripe or overripe strawberries are not desirable and contribute to fruit waste. Thus, for optimal quality and maintenance of quality, they need to be harvested at the right time [20].

Several attributes, such as sweetness and firmness, can indicate strawberry quality. Classic methods to obtain quality attributes involve manual ratings from experts based on color and texture, as well as laboratory evaluations of chemical constituents [26]. These methods are possibly destructive, time consuming, and labor-intensive [7].

More recent methods analyze strawberries using computer vision. In most literature, strawberries are analyzed after harvesting [6, 9, 26]. This results in accurate and automated quality prediction. However, as quality is only predicted after harvesting, a good timing to harvest cannot be found: a strawberry that is harvested too early does not continue to ripen. To overcome this obstacle, we estimate quality attributes infield.

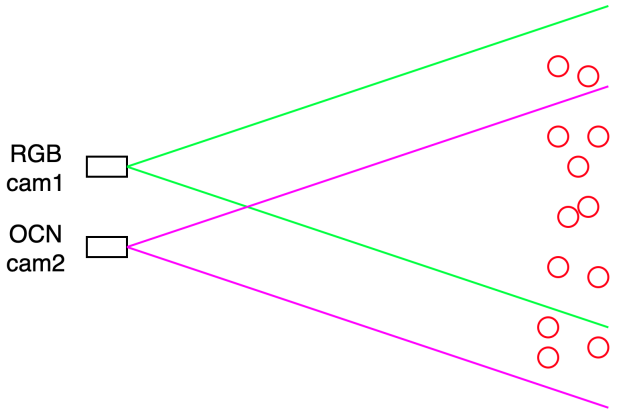


Figure 1. A top view illustration of our setup. Red circles denote strawberries. Two cameras point at an array of strawberry plants, allowing us to use stereo vision for depth estimation. The second camera uses Orange Cyan NIR (OCN) color bands instead of RGB, allowing us to observe more color bands.

Our setup is in a greenhouse in the Netherlands, which has plants that grow the Favori strawberry. Strawberries deemed ripe are manually harvested every few days.

Our setup, showed in Figure 1, contains two cameras that both take a picture every hour. We have three of such sets, which point at different strawberry plants. Two example pictures are in Figure 2. There is an RGB and a Near Infrared (NIR) camera that are parallel to each other. The second camera has three channels: Orange, Cyan, and NIR (OCN). Using more color bands than RGB might improve quality prediction.

The stereo setup allows us to estimate strawberry size, for which relevant measurements and camera specifications are known. We calculate size as strawberries are placed in markets based on their size; specifically, their width in mil-

limeters. Strawberries placed in the wrong markets could lead to waste.

We also have many sensors that measure environment factors, such as certain temperatures, radiation, and more. Such environment factors correlate with certain quality attributes [5,11,15,23]. We use them to try to improve quality prediction.

We have two types of data of the strawberries: preharvest labelling information and postharvest quality information. First, each strawberry is labelled by coordinates of the segment in the image and a *track_id*. This id tracks a strawberry across its lifetime, *i.e.*, the id is the same for the same strawberry across images.

Second, we have data of four quality attributes per strawberry. They are: sweetness (*Brix*), firmness (kg/mm^2), ripeness, and size. Brix is a a measure of the soluble solids content of a substance, primarily sugar. Sweetness and firmness are measured using instruments. Ripeness is an estimated parameter based on three experts that have rated pictures of the fruits on a scale from 1 to 10. Around 7 to 8 is optimally ripe; below that is underripe, and above that is overripe. Size is the width of the strawberry in millimeters, and is divided into three classes: *tiny* ($<$ 20mm), *small* (20mm - 25mm), and *coarse* ($>$ 25 mm).

We conclude the following based on our experiments. Infield prediction of measured attributes (Brix, firmness) is decently possible and of an annotated attribute (ripeness) is well possible. Environment data improves quality predictions. Strawberry inpainting can be used to recover the original shape with decent accuracy on artificial occlusions. Finally, using stereo vision slightly improves strawberry size estimation.

2. Related work

2.1. Visible spectrum

While customers judge fruit by their external qualities, such as shape and color, internal quality parameters influence customer appreciation [14]. Computer vision has been used extensively to determine quality parameters of fruits. An accuracy of 85.6% on three classes of ripeness was achieved on strawberries using RGB images [9]. RGB is highly correlated with *total soluble solids* (TSS) [1]. TSS is a measure of soluble solids, primarily sugar, within a substance. 79.2% accuracy was achieved on six classes of TSS using a Support Vector Machine (SVM) and an RGB camera [3]. CaffeNet reached 95% accuracy on classifying whether strawberries were mature [12].

Computer vision can get good accuracy on quality prediction because chemical changes inside strawberries are visible on a camera. Various chemical changes occur during fruit ripening. Chlorophyll is a green pigment frequently found in plants. The destruction of chlorophyll plays a role in the ripening process [4]. Anthocyanin is a pigment that appears red in strawberries and is synthesized during the ripening process [10]. Such chemical concentrations can be observed visually: anthocyanin and chlorophyll pigments have been shown to be visible in the regions around 535 nm and 680 nm, respectively [16]. These regions are visible to the eye.

In our setup, we use computer vision and deep architectures to predict quality parameters.

2.2. Hyperspectral imaging

Hyperspectral imaging is a technique that combines spatial and spectral information: each pixel contains an entire wavelength spectrum [25]. To determine ripeness, many works first select optimal wavelengths and then use classifiers to determine ripeness classes of strawberries [7,26] and other fruits such as persimmon [25]. For example, 98.6% accuracy on determining ripeness has been achieved [7]. Hyperspectral imaging has also been shown to be effective to predict measurable quality parameters, such as firmness [16], sweetness [2, 6, 19], and titratable acidity [2, 22].

Certain constituents in strawberries absorb infrared light and reflect frequencies that indicate quality attributes [7]. Our eyes only see a part of the electromagnetic spectrum. Hyperspectral imaging can observe ranges beyond what the eye can see, which could improve quality predictions as it shows more details of light absorbance.

A disadvantage of hyperspectral imaging is that devices to acquire such images are often expensive and complicated [27], making them unpractical for some farmers. For this reason, our setup does not include a spectrometer, so we cannot use hyperspectral imaging. However, we do have both an RGB and an OCN camera. Using a wider variety of wavelengths could improve quality prediction.

2.3. Environmental influence

Several environmental factors correlate significantly with strawberry quality, and can thus be used to improve quality predictions. Radiation, temperature, and relative humidity are all correlated with Brix values [5]. Optimal temperature and light increases Brix values [23].

Temperature of both greenhouse and soil influences growth [11]. Weather patterns, such as solar radiation and wind, can also influence growth [15].

Our setup has many sensors that measure temperature, radiation, and more, and we will use them to try to improve quality predictions.

2.4. Inpainting

Infield images of fruits can be occluded by obstacles such as other fruits and leaves. This makes it difficult to estimate their true size.

(a) Example of an RGB picture of the strawberries.

(b) Example of an OCN picture of the strawberries.

Figure 2. An example of an RGB and an OCN picture taken at the same location and time. Pictures are taken every hour. As the OCN camera is placed slightly to the right of the RGB camera, the strawberries appear shifted to the left. Using two cameras allows us to use stereo vision for strawberry size estimation and more color bands through the OCN camera.

A possible solution is image inpainting. The goal of inpainting is to complete missing parts of an image. It has been performed both for specific tasks, such as face completion [13], and to inpaint a wide variety of images [18].

For infield fruit occlusions, one approach has been to manually occlude a dataset of tomatoes, and train U-Net to recover the original shape [8]. The authors explain and solve a problem that is similar to ours, and thus we take a similar approach.

3. Method

3.1. Quality prediction

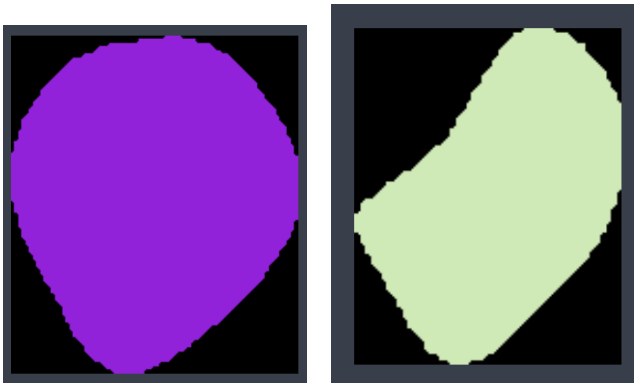
We run the strawberry pictures on two convolutional neural networks (CNNs): LeNet-5 and ResNet-18. In LeNet, we replace the Tanh activation layer with a ReLU activation layer and average pooling with max pooling. In ResNet, we add a second linear layer so that the network can learn from environment data.

We use Mean Squared Error (MSE) as loss function and Adam as optimizer with a learning rate of 3e-5. We train on 60% of data, validate performance on 20% of the data, and use the remaining 20% of our data for demonstrated results. We run on 100 epochs and use early stopping to select the model with best performance on the validation set. We use a batch size of 8 and a weight decay of 0.01. For each run, we randomly divide the data to the training, validation, and test set.

We have a dataset with ripeness values for 254 strawberries and quality attributes for 184 strawberries. However, some quality attribute entries are missing: 6 entries have no value for Brix and 36 entries have no value for firmness.

Figure 3. Three data augmentation techniques on a strawberry sample. Each technique is applied subsequently on the results of the previous one, leading to $2^3 = 8$ permutations. The original strawberry is at the top left. The bottom left strawberry is horizontally flipped. The second column displays random rotation. The third and fourth column displays random cropping. The augmenting process results in more data to train on.

Since we have a relatively low amount of images, we augment the data as follows. Each strawberry is flipped horizontally, rotated randomly uniform between $-30°$ and $30°$, and cropped by factor of 1.5. This turns every strawberry into eight strawberry samples: all of the three permutations of the augmentations. The augmentations aim to increase the size of our data set, without changing attributes that would alter the quality prediction, such as the color. The data augmentation is only performed on the training set. An example of an augmented strawberry sample is show in Figure 3.



(a) Strawberry segment with a circularity of 0.263: not occluded.

(b) Strawberry segment with a circularity of 0.245: occluded.

Figure 4. Two examples of strawberry segments: an occluded segment has a lower circularity, which indicates that the segment might be occluded. We aim to only inpaint occluded strawberries to avoid possible performance loss.

3.2. Inpainting

In the pictures taken at our greenhouse, some strawberries are occluded. This degrades size estimation performance, as the width in pixels might be less than it would be without occlusions. In this section, we outline our image inpainting algorithm that aims to recover the original strawberry shape.

3.2.1 Determining occlusions

We created a simple method to determine if a strawberry is occluded or not. We have two motivations for this. First, when gathering a data set to perform inpainting on, we aim to include only non-occluded images, so that the ground truth has no occlusions. Second, during size estimation, we aim to only inpaint occluded strawberries, as performing inpainting on a non-occluded strawberry could decrease the performance of size estimation.

We aim to calculate circularity of strawberries to estimate if they are occluded or not. Many formulas for circularity have been proposed [21]. An occluded strawberry has a similar circumference as a non-occluded strawberry, but a smaller surface. Therefore, we use a ratio between them to try to tell if the strawberry is occluded:

$$\text{circularity} = \frac{\sqrt{\text{surface}}}{\text{circumference}}. \quad (1)$$

Since the circumference grows linear, but the surface grows squared, we take the square root so that the ratio stays the same regardless of the size of the strawberry. Empirically, a circularity of 0.25 or less indicates an occluded strawberry. An example of detecting occlusions using circularity is illustrated in Figure 4.

Occasionally, an occluded strawberry might be just above this threshold. However, then the difference in width is probably small, which means that for our purpose of size prediction, this should not significantly degrade performance.

3.2.2 Inpainting algorithm

We perform inpainting on strawberry segments. We gathered 279 images of our data to train on. We use 25% of the images for the *occlusion set*, which are used to artificially occlude the input image. Of the remaining part of the images, 75% went to training set and the rest to the test set.

In a real occlusion, the predicted bounding box is often too small, *i.e.*, the part of the strawberry that is occluded, is not in the bounding box. Thus, we reserve some space for the inpainting algorithm to fill in values. The strawberries keep their aspect ratio and are scaled down to take up at most half the input image. They are placed in the center of the input image.

At every epoch, each strawberry is occluded by overlapping it with a randomly chosen transparent strawberry from the occlusion set. This strawberry is also placed in the center, but shifted in both the x and y direction. We aim to use varying levels of occlusion per training sample. First, with 50% probability, a shift value of either 32 or 64 pixels is chosen. To prevent the occluding strawberry from completely overlapping the visible strawberry, in either the x or y direction, the strawberry is moved the full shift value; in the other direction, it is moved randomly uniform between 0 and the shift value. After occluding, the strawberry is recentered, since on a truly occluded strawberry, the input would also be centered.

To model multiple occlusions, 50% of the time, we place two occlusions. We use U-Net for the image inpainting task. The inputs and outputs are both 256x256.

We use two configurations. The first configuration uses RGBA inputs, the second configuration uses binary inputs. Binary input images are obtained by only keeping the opacity channel, and using a threshold of 0.5 to determine the binary value. The motivation behind binary inputs is that it makes it easier for the network to focus on the shape, rather than on the colors.

The loss function is MSE. For RGBA, it covers all four channels, for the binary inputs, it covers only the binary channel. Additionally, for indication of performance, we calculate the Intersection over Union (IoU) and the difference in width between the network output and the ground truth.

3.3. Stereo matching

The same strawberry usually appears on both the RGB and the OCN camera. We need to match each strawberry

Figure 5. A method to try to calibrate the two cameras after pictures were taken. Through translation, scaling, and rotation, vertical disparity is eliminated and horizontal disparity is similar left and right. This is needed for good accuracy on two subsequent tasks: matching of strawberries of the two cameras and disparity calculation. We seem to be able to match the pictures well, based on some aspects in the image such as the small white bar in the middle.

segment that is on the RGB camera to the one that appears on the OCN camera so that we can estimate the depth of a strawberry.

In our setup, the cameras are not calibrated. This could lead to invalid matching of strawberries between the RGB and OCN cameras and inaccurate disparity estimations. Thus, we first calibrate the RGB and OCN pictures as best as we can. Then, we aim to find the best fitting match between the strawberry segments.

3.3.1 Camera calibration

To try to calibrate the RGB and OCN images, we aim to overlap them perfectly, which means two things. First, there should be no vertical disparity anywhere. Second, there should be similar horizontal disparity regardless of position of the image. To this end, for each camera set, the OCN picture was translated, rotated, and scaled, as demonstrated in Figure 5.

3.3.2 Matching algorithm

We need to match each strawberry from the RGB image to a strawberry in the OCN image, *i.e.*, for each RGB strawberry segment, find the segment on the OCN camera that displays the same strawberry. We propose a simple loss function that involves two components: the distance in location and the distance in size. The intuition is that for each RGB strawberry segment, the further away the OCN segment is and the more different it is in size, the less likely it is that they are matches.

We follow a method outlined in [24] to calculate the distance in size between the strawberries:

$$ds = \frac{|\text{surface}_{\text{RGB}} - \text{surface}_{\text{OCN}}|}{\text{surface}_{\text{RGB}} + \text{surface}_{\text{OCN}}}, \tag{2}$$

where *surface* is the number of pixels of a segment. The formula calculates the difference in size and is in range $[0, 1]$.

The formula we use for the distance in location is:

$$dl = \sqrt{dx^2 + dy^2 \cdot \alpha}, \tag{3}$$

where dx is the distance between the x-coordinates, after subtracting the expected disparity in pixels; dy is the distance between the y-coordinates; and α is a tunable parameter. $\alpha \geq 1$ increases the weight of vertical disparity in the loss function. While horizontal disparity depends on the distance of a strawberry to the camera, and is thus flexible, vertical disparity should be near zero. Thus, for matching, vertical disparity should have more impact on the loss.

We multiply the two equations to calculate a loss between each strawberry in the RGB image and each strawberry in the corresponding OCN image:

$$loss = dl \cdot (ds + \beta), \tag{4}$$

where β is a tunable parameter that stabilizes the loss value for low values of ds. As we wish to minimize distances in location and size, we choose the strawberry with the smallest loss as a match.

We use $\alpha = 10$ and $\beta = 0.01$, as empirically, we find that it results in good matches. The exact values of the parameters is not important.

3.4. Depth estimation

Strawberries are divided into classes by their width in millimeters (mm). We can estimate the width of a strawberry sample by calculating the depth and counting the width in pixels.

We follow a standard method to calculate depth by using stereo vision as outlined in [17, 28]. We calculate the depth in millimeters as follows:

$$D = \frac{bf}{d}, \tag{5}$$

where b is the baseline distance between the two cameras in millimeters, f is the focal length in pixels, and d is the disparity between the RGB and the OCN segments in pixels. The focal length is

$$f = \frac{w_{\text{img[px]}}}{2\tan(\frac{\theta}{2})}, \tag{6}$$

where for each camera, $w_{\text{img[px]}} = 4000$px is the width in pixels of the output and $\theta = 41^\circ$ is the angle.

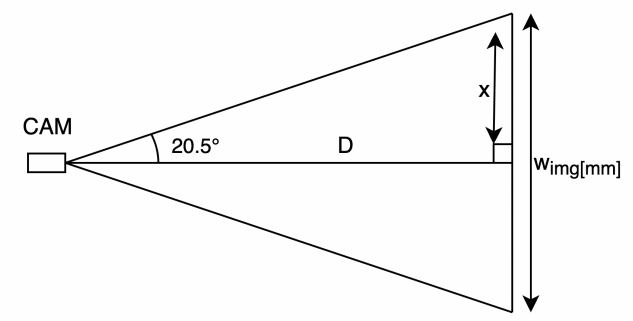


Figure 6. A triangulation method to find the width of the image in millimeters at a given depth, D. The camera is located at distance D from the strawberry, We use an angle of 20.5° as the camera FOV is 41°. Using triangulation, we find x; $2x$ is the width of the entire image. The width is later used to find the width of a strawberry in millimeters.

3.5. Size estimation

Once we have calculated the depth, we aim to calculate the width of a strawberry in millimeters. Here, we first calculate the width of the entire image in millimeters at the depth of the strawberry. Then we use the fraction of the width that the strawberry occupies in the image to find the width of the strawberry.

First, to calculate the width of the entire image, we use a simple triangulation method, as illustrated in Figure 6. Given that we know the angle of the camera, we can find the width of the image in millimeters at a given depth using

$$w_{\text{img[mm]}} = 2\tan(\frac{\theta}{2}) \cdot D, \tag{7}$$

where the width is multiplied by 2 as $w_{\text{img[mm]}} = 2x$.

Second, we multiply with the fraction of the width of the image that the strawberry occupies. This gives the width of the strawberry in millimeters:

$$w_{\text{str[mm]}} = \frac{w_{\text{str[px]}}}{w_{\text{img[px]}}} \cdot w_{\text{img[mm]}}, \tag{8}$$

where $w_{\text{str[px]}}$ is the width of a strawberry in pixels. The above formulas can be used to calculate the size of a strawberry in millimeters.

4. Experiments

4.1. Quality prediction

To find the best performance on quality prediction, we compare the following configuration options:

- input: RGB, OCN, or both;
- model: LeNet-5, ResNet-18.

Color mode	Model	MSE
RGB	LeNet	**1.39** $\pm$ 0.438
OCN	LeNet	1.69 $\pm$ 0.486
Dual	LeNet	1.51 $\pm$ 0.454
RGB	ResNet	1.81 $\pm$ 0.581
OCN	ResNet	2.06 $\pm$ 0.782
Dual	ResNet	2.02 $\pm$ 0.621

Table 1. Results of Brix estimation. MSE is the average MSE loss over five experiments, we also denote the standard deviation here. LeNet on RGB inputs has the best performance.

Color mode	Model	MSE $\cdot 10^{-2}$
RGB	LeNet	**3.60** $\pm$ 0.549
OCN	LeNet	5.77 $\pm$ 1.74
Dual	LeNet	3.76 $\pm$ 0.223
RGB	ResNet	4.27 $\pm$ 0.580
OCN	ResNet	5.78 $\pm$ 0.974
Dual	ResNet	4.84 $\pm$ 1.12

Table 2. Results of firmness estimation. Both models appear to have best performance on OCN. LeNet on RGB input has the best performance.

For ripeness, we only evaluate RGB input images, as the ripeness values were determined by experts on the RGB images.

The CNNs we use have both a convolutional part, of which the output is inserted into a multilayer perceptron (MLP). The environment data is added to the network by concatenating it with the output from the convolutional network. For each environment feature, we add both the average of the past four days before harvest and the average over the past four days one week before harvesting.

The OCN image is the strawberry that matches the RGB image. It is found by using our matching algorithm. To train the network on both the RGB and the OCN image, the convolutional part is duplicated: the first part is used on the RGB image and the second part on the OCN image. The outputs are concatenated and inserted into an MLP. It is possible that the matching strawberry is not visible on the OCN camera due to the disparity. In this case, for the dual network, we insert a black and transparent image instead, *i.e.*, $rgba(0, 0, 0, 0)$. For the OCN network, we skip the data point.

To properly compare configurations, for each configuration, we perform 5 runs with each a fixed and incrementing random seed that is consistent across different configurations. We report the mean and standard deviation for each configuration.

The best performance on estimation of all attributes is

Color mode	Model	MSE $\cdot 10^{-1}$
RGB	LeNet	**6.30** $\pm$ 0.739
RGB	ResNet	7.84 $\pm$ 2.04

Table 3. Results of ripeness estimation. The best performance is achieved on LeNet.

Attribute	Unweighted	Weighted	Loss increase
Brix	1.39 $\pm$ 0.438	2.02 $\pm$ 0.639	45.3%
Firmness	4.01 $\pm$ 0.940	3.60 $\pm$ 0.549	-10.3%
Ripeness	6.76 $\pm$ 1.06	6.30 $\pm$ 0.739	-6.6%

Table 4. Results of changing the loss function using LeNet. For comparison, all the loss values we display are MSE. Networks were trained on either WMSE or MSE. On LeNet, loss weighting increases the MSE loss for Brix, but decreases the MSE for firmness and ripeness. Thus, for these configurations, we trained on weighted loss in other tables in this section.

obtained using LeNet on RGB inputs, as shown in Table 1, Table 2, and Table 3.

4.1.1 Weighted loss function

The quality attributes are not uniformly distributed, as shown in Figure 7. Thus, certain values have few occurrences. We found that the network typically only predicts the common values. In a new data batch with many uncommon values, performance could be degraded.

As a solution, we aim to increase the weight of the loss on uncommon values. A weighted loss function multiplies the loss by how uncommon the value is. For example, on a binary dataset that has two instances of X and one instance of Y, the loss on the Y value will be weighted such that it counts twice as much. We refer to this loss weighting as Weighted MSE (WMSE). Further, we can add an exponent to our weight values. We try both exponents of 0.5 and 2, and refer to them as Square Root WMSE (SQWMSE) and Squared WMSE (SWMSE), respectively.

We find that WMSE and SQWMSE improves performance on LeNet on firmness and ripeness, but degrades performance on other configurations. WMSE performs better than SQWMSE. We show loss increase for each quality attribute in Table 4. The tables in this section all present MSE loss, but were trained using the loss that gave best performance.

4.1.2 No environment data

We add environment data as input to our networks to improve predictions. We aim to find the effect of adding environment data on each attribute. To this end, we run the models with best performance without environment data. We show our results in Table 5. Most notably, Brix prediction is strongly impacted by environment data.

Attribute	With env	Without env	Loss increase
Brix	1.39 $\pm$ 0.438	2.53 $\pm$ 0.714	82.1%
Firmness	3.60 $\pm$ 0.549	3.86 $\pm$ 0.463	9.48%
Ripeness	6.30 $\pm$ 0.739	6.65 $\pm$ 0.966	5.56%

Table 5. Results of quality attribute estimation of LeNet on RGB input images, either with or without environment data. Performance drop is most significant for Brix estimation. Performance on firmness and ripeness estimation is slightly decreased from omitting environment data.

Color mode	MSE $\cdot 10^{-3}$	IoU	Width diff
RGBA	**5.46**	0.871	2.00
Binary	7.21	**0.874**	**1.70**

Table 6. Results of inpainting on the test set. Width diff is the difference in width between the network output and the ground truth. The models have roughly equal IoU, but the binary model has a higher loss but a lower width difference. It is not immediately clear which configuration is best.

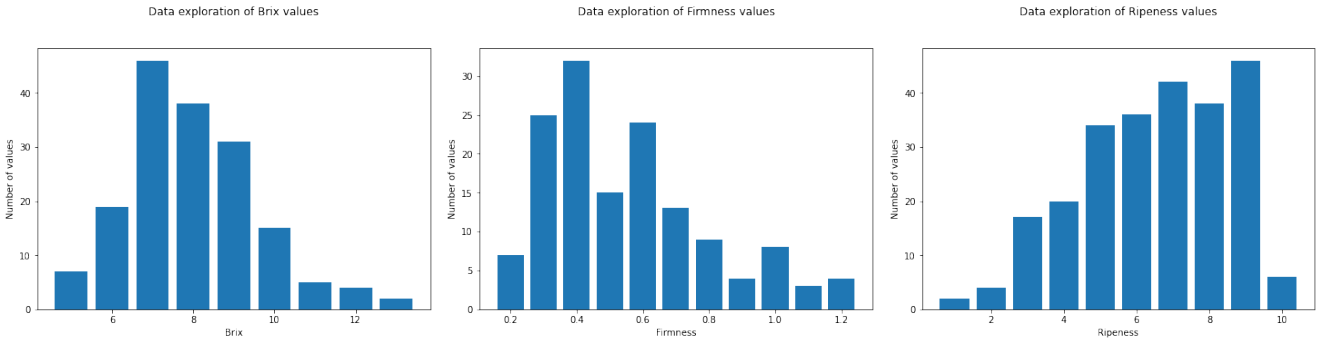
4.2. Inpainting

We test our inpainting model on two configurations: RGBA and binary input images. Results are in Table 6. The model with RGBA input images achieves a lower loss, while the model on binary input images achieves a higher IoU and a lower width difference.

Given artificial occlusions, the network performs well, see Figure 8. However, on real occlusions, *i.e.*, occlusions present in our data, performance varies. We show an example in Figure 9. It is difficult to occlude in a wide enough variety of ways such that the network learns a general way to inpaint.

4.3. Stereo matching

As there is no ground truth information for matching, we only evaluate the stereo matching algorithm manually. We show an example of a result of the matching algorithm in Figure 10. Most strawberries are matched correctly, with two major exceptions. First are a few tiny strawberries; the performance on those for this research is not relevant, as they are not near being harvested. Second, the large strawberries at the top are from the plant above the cameras: they are far closer, and thus have much larger disparities. Performance is not relevant here either, as these strawberries are not part of our measurements.



(a) Brix distribution. (b) Firmness distribution. (c) Ripeness distribution.

Figure 7. Distribution of quality attributes in our dataset. Each bar plots the number of strawberry samples for each possible value of the quality attribute. The distributions are not uniform, so the network could be biased towards the most common values, and thus never predict the lowest or highest values.

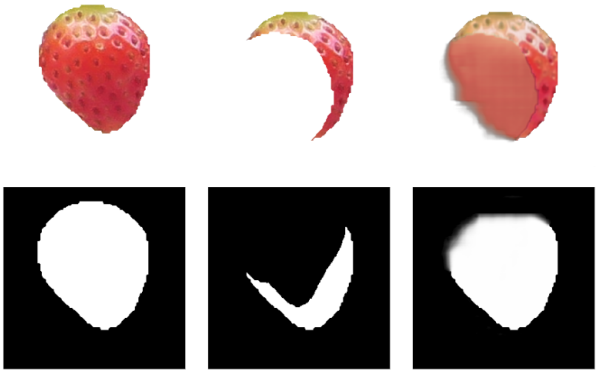


Figure 8. Example of results of the inpainting algorithm on a strawberry. The first row uses RGBA inputs, the second row uses binary inputs. The left column is the ground truth. The middle column is the network input. The right column is the predicted output. The performance on both configurations is good: the original shape is reasonably recovered, allowing us to estimate the strawberry width properly.

Figure 9. Examples of results of the inpainting algorithm on *real* occluded strawberries. As the occlusion is present in our data, we do not have the ground truth. The first row uses RGBA inputs, the second row uses binary inputs. The left column is the network input. The right column is the predicted output. There is no non-occluded ground truth of the strawberries, as the strawberries are occluded infield. The performance on the RGBA configuration is poor and on the binary configuration is mediocre.

4.4. Depth estimation

Using the disparity, we can calculate the depth. Figure 11 shows a visualization of depth.

It is difficult to assess the performance of the algorithm, as there is no ground truth information of depth. Further, it is difficult for the human eye to observe depth differences. For example, the visualization shows that some strawberries have similar depth left and right on the bottom of Figure 11, but is difficult to verify if this is accurate. In Figure 12 we show that locally, the algorithm seems to perform well.

4.5. Size estimation

Our size estimation results are in Table 7. The accuracy is the percentage of strawberries where the predicted size was equal to the true size class. It increases when using stereo vision size estimation, but decreases when using inpainting.

Method	Accuracy
Baseline	63.1%
Stereo	**64.5%**
Inpainting	62.4%
Stereo and inpainting	63.8%

Table 7. Effects of disparity calculation and inpainting on size estimation. Stereo vision improves results slightly, while inpainting degrades results.

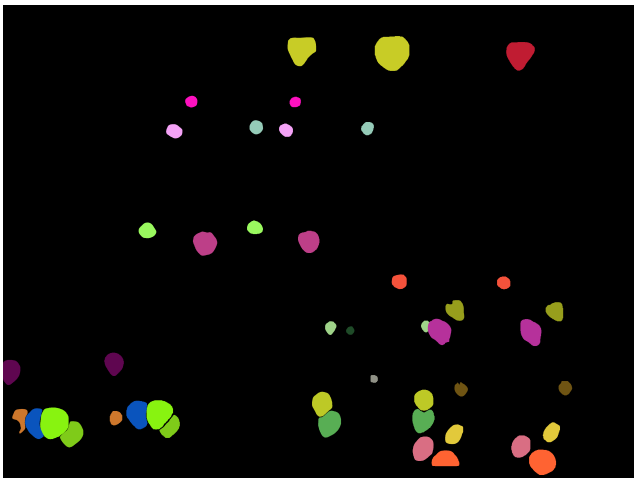


Figure 10. Example of matching of the images of Figure 2. Strawberries appear twice since both the strawberry segments from the RGB and the OCN camera are shown. Segments shown with the same color were matched by our algorithm. When manually evaluating, the matching performance is good on strawberries that are relevant for our size measurements. Proper matching allows us to calculate disparity, depth, and finally the size of a strawberry.

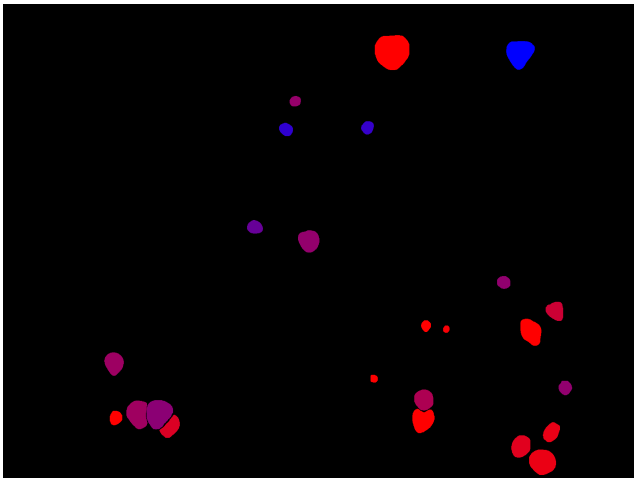


Figure 11. Example of depth estimation from Figure 10. The more blue a segment it is, the closer it is; the more red, the further away. Depth ranges roughly from 800mm to 1000mm. Depth information allows us to more accurately calculate the size of a strawberry.

5. Discussion

5.1. Quality prediction

The networks were trained on the images from the same setup as they were tested on. This means that the networks will likely perform worse when tested on strawberries from other setups. On a new setup, the network should likely be retrained on that setup to obtain good performance.

Our performance is worse than many related works. However, many related works use hyperspectral imaging. A disadvantage of hyperspectral imaging is that devices to acquire such images are often expensive and complicated [27], making them unpractical for some farmers. Depending on the budget of a farmer, our approach might be more suitable. Also, many related works do not analyze strawberry quality infield, meaning it is easier to control aspects such as lighting.

Compared to the range of values, the networks make a smaller error on ripeness than on Brix and firmness. This is probably because ripeness was manually evaluated based on pictures, so a neural network can simply learn how humans evaluate.

Figure 12. Close up of three strawberry segments on the bottom left of Figure 11. The depth estimation algorithm seems to put occluded segments further away, which indicates good performance locally.

5.2. Inpainting

Inpainting performance appears significantly worse on real occlusions than on artificial occlusions. It appears that the network does not learn a general representation of a strawberry, but rather, learns how to inpaint certain occlusions. Thus, the network does not adapt to all types of occlusions.

An argument could be made that metrics like MSE and IoU were not designed indicate shape similarity. While this is true, they do give an indication of reconstruction quality.

5.3. Depth estimation

During harvesting, occasionally someone bumped into the camera, after which the cameras were not calibrated. Further, we have no ground truth labels for depth. These factors have limited the effectiveness of the stereo depth estimation.

6. Conclusion

We analyze strawberry quality attributes and size infield. Infield predictions allow us to analyze strawberries before harvesting. We achieve reasonable accuracy on predictions on Brix, firmness, and ripeness. We also aim to calculate

size infield. We used inpainting to recover the shape of occluded strawberries. Inpainting did not improve size estimation. The inpainting performance is good on artificial occlusions, but varying on real occlusions, as it is hard to model all types of occlusions. Calculating depth through stereo vision improved size estimation, but only slightly, as our cameras were not calibrated.

References

[1] P.D. Abeytilakarathna, R.M. Fonseka, J.P. Eswara, and K.G.N.A.B. Wijethunga. Relationship between total solid content and red, green and blue colour intensity of strawberry (fragaria x ananassa duch.) fruits. *Journal of Agricultural Sciences*, 8, 07 2013. 2

[2] Maria Luisa Amodio, Francesco Ceglie, Muhammad Mudassir Arif Chaudhry, Francesca Piazzolla, and Giancarlo Colelli. Potential of nir spectroscopy for predicting internal quality and discriminating among strawberry fruits from different production systems. *Postharvest Biology and Technology*, 125:112–121, 2017. 2

[3] Jayanta Kumar Basak, Bolappa Gamage Kaushalya Madhavi, Bhola Paudel, Na Eun Kim, and Hyeon Tae Kim. Prediction of total soluble solids and ph of strawberry fruits using rgb, hsv and hsl colour spaces and machine learning models. *Foods*, 11(14), 2022. 2

[4] C J Brady. Fruit ripening. *Annual Review of Plant Physiology*, 38(1):155–178, 1987. 2

[5] Fatima I. Pereira da Silva, Sabine K. Schnabel, Bastiaan Brouwer, and Manon G. Mensink. Monitoring strawberry production to get grip on strawberry quality. *GreenCHAINge Fruit & Vegetables*, 2018. 2

[6] Gamal ElMasry, Ning Wang, Adel ElSayed, and Michael Ngadi. Hyperspectral imaging for nondestructive determination of some quality attributes for strawberry. *Journal of Food Engineering*, 81(1):98–107, 2007. 1, 2

[7] Zongmei Gao, Yuanyuan Shao, Guantao Xuan, Yongxian Wang, Yi Liu, and Xiang Han. Real-time hyperspectral imaging for the in-field estimation of strawberry ripeness with deep learning. *Artificial Intelligence in Agriculture*, 4:31–38, 2020. 1, 2

[8] Liang Gong, Wenjie Wang, Tao Wang, and Chengliang Liu. Robotic harvesting of the occluded fruits with a precise shape and position reconstruction approach. *Journal of Field Robotics*, 39, 10 2021. 3

[9] Indrabayu Indrabayu, Nurhikma Arifin, and Intan Sari Areni. Strawberry ripeness classification system based on skin tone color using multi-class support vector machine. In *2019 International Conference on Information and Communications Technology (ICOIACT)*, pages 191–195, 2019. 1, 2

[10] M.N. Islam, Mehnaz Mursalat, and Mohidus Samad Khan. A review on the legislative aspect of artificial fruit ripening. *Agric & Food Secur*, 5, 06 2016. 2

[11] Dong Sub Kim and Sung Kim. Prediction of strawberry growth and fruit yield based on environmental and growth data in a greenhouse for soil cultivation with applied autonomous facilities. *Wonye kwahak kisulchi: Korean journal of horticultural science and technology*, 38, 12 2020. 2

[12] Xin Li, Jun Yu Li, and Jing Tang. A deep learning method for recognizing elevated mature strawberries. *2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 1072–1077, 2018. 2

[13] Yijun Li, Sifei Liu, Jimei Yang, and Ming-Hsuan Yang. Generative face completion, 2017. 3

[14] Manuela Mancini, Luca Mazzoni, Francesco Gagliardi, Francesca Balducci, Daniele Duca, Giuseppe Toscano, Bruno Mezzetti, and Franco Capocasa. Application of the non-destructive nir technique for the evaluation of strawberry fruits quality parameters. *Foods*, 9(4), 2020. 2

[15] Mahesh Maskey, Tapan Pathak, and Surendra Dara. Weather based strawberry yield forecasts at field scale using statistical and machine learning models. *Atmosphere*, 10:378, 07 2019. 2

[16] Binu Melit Devassy and Sony George. Estimation of strawberry firmness using hyperspectral imaging: a comparison of regression models. *Journal of Spectral Imaging*, 10, 06 2021. 2

[17] Dhaval K. Patel, Pankaj A. Bachani, and Nirav R. Shah. Distance measurement system using binocular stereo vision approach. *International journal of engineering research and technology*, 2, 2013. 5

[18] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[19] Ana Santos, Sara Ricardo Rodrigues, Marta Laranjo, C. Melgão, and R. Velázquez. Non-destructive prediction of total soluble solids in strawberry using near infrared spectroscopy. *Journal of the Science of Food and Agriculture*, 102, 03 2022. 2

[20] K. Sturm, D. Koron, and F. Stampar. The composition of fruit of different strawberry varieties depending on maturity stage. *Food Chemistry*, 83(3):417–422, 2003. 1

[21] Sylwia Szerakowska, Maria Sulewska, Jerzy Trzciński, and Barbara Woronko. Comparison of methods determining particle sphericity. *Applied Mechanics and Materials*, 797:231–237, 11 2015. 4

[22] María-Teresa Sánchez, María José De la Haba, Miriam Benítez-López, Juan Fernández-Novales, Ana Garrido-Varo, and D.C. Perez-Marin. Non-destructive characterization and quality control of intact strawberries based on nir spectral data. *Journal of Food Engineering*, 110:102–108, 05 2012. 2

[23] Yilian Tang, Xun Ma, Ming Li, and Yunfeng Wang. The effect of temperature and light on strawberry production in a solar greenhouse. *Solar Energy*, 195:318–328, 2020. 2

[24] Kris van Melis. Measuring the size of strawberries using binocular photos. Dissertation for Bachelor of Computer Science and Engineering, 2022. 5

[25] Xuan Wei, Fei Liu, Zhengjun Qiu, Yongni Shao, and Yong He. Ripeness classification of astringent persimmon using hyperspectral imaging technique. *Food and Bioprocess Technology*, 7:1371–1380, 2013. 2

[26] Chu zhang, Chentong Guo, Fei Liu, Wenwen Kong, Yong He, and Binggan Lou. Hyperspectral imaging analysis for

ripeness evaluation of strawberry with support vector machine. *Journal of Food Engineering*, 179:11–18, 2016. 1, 2

[27] Jingang Zhang, Runmu Su, Qiang Fu, Wenqi Ren, Felix Heide, and Yunfeng Nie. A survey on computational spectral reconstruction methods from RGB to hyperspectral imaging. *Scientific Reports*, 12(1), jul 2022. 2, 9

[28] Manaf Zivingy. Object distance measurement by stereo vision. *International Journal of Science and Applied Information Technology (IJSAIT)*, 2:05–08, 01 2013. 5

3

Backgound on Deep Learning

3.1. Neural networks

3.1.1. Goal

The goal of a neural network is to approximate a function f^* using f [6]. A neural network has several parameters θ, which it tunes over time to approximate the original function. The performance is expressed as *loss*; the goal is to minimize this loss. Loss functions are often differentiable. The derivative can be used for minimizing the loss function because it indicates how to change the parameters to minimize the loss [6]. By moving in the negative sign of the derivative, we can reduce the loss. We call this *gradient descent*, where the gradient is a vector of partial derivatives. We visualize this in figure 3.1.

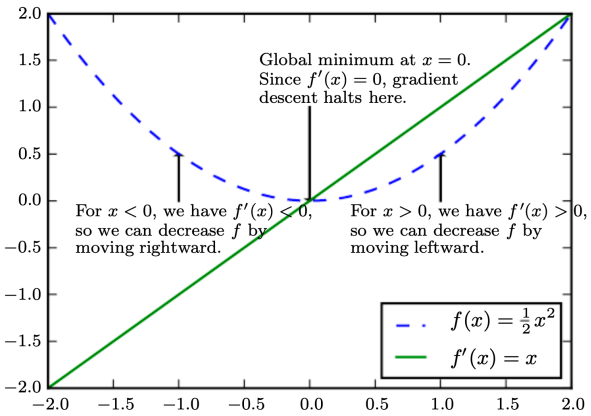


Figure 3.1: Simple example of gradient descent [6]. The blue line is the loss function and the green line is its gradient. To minimize the loss function, we can move in the negative direction of the gradient. Although in this 1-dimensional situation, the solution can be found trivially by solving for $f'(x) = 0$, this solution does not work in practice: most problems are multidimensional, and could have thousands of dimensions. Thus, we use gradient descent instead.

3.1.2. Single perceptron

A perceptron is a single-layer neural network, illustrated in figure 3.2. The network can receive multiple inputs.

Two simple functions to learn are *OR* and *Exclusive Or (XOR)*. Both functions take two binary inputs and return a binary output. While the OR function returns *true* if either inputs are true, XOR returns true if its inputs are not equal, see table 3.1. For the problems, we aim to separate the two binary classes, true and false. We denote these classes as 0 and 1. We use Mean Squared Error (MSE) as loss function, which is

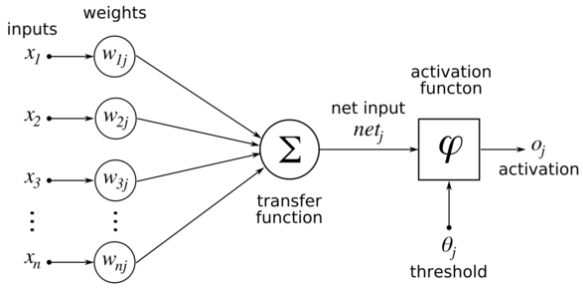


Figure 3.2: An illustration of a perceptron [23]. A perceptron receives inputs and multiplies them by its weights.

Data		
Input	OR	XOR
0 0	0	0
0 1	1	1
1 0	1	1
1 1	**1**	**0**

Table 3.1: Input and output values of the OR and XOR gates. We denote *true* as 1 and *false* as 0. The OR gate returns true if either input is true, while the XOR gate returns true if the inputs differ. The only different output between the two gates is when both inputs are true, illustrated in bold text.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f^*(x) - f(x;\theta))^2, \tag{3.1}$$

where θ indicates the set of parameters w and b, and $J(\theta)$ is the loss function over those parameters. We can use a linear equation for our model function f:

$$f(x;w,b) = x^\top w + b. \tag{3.2}$$

Our current network is capable of linearly separating classes. For the OR function, a linear function can separate the two classes, as shown in figure 3.3.

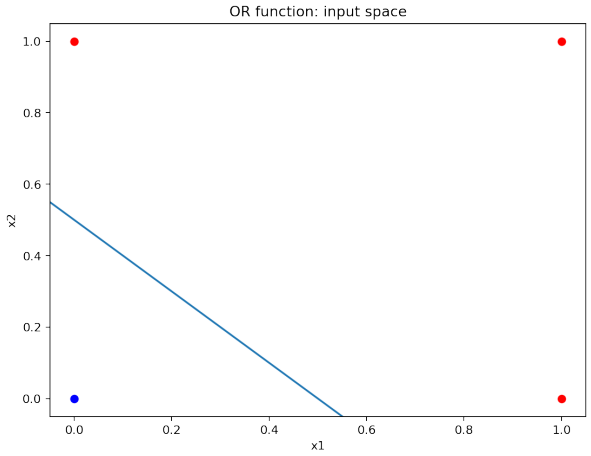


Figure 3.3: Visualization of the OR problem. The OR problem has two binary inputs and a binary output. The two axes indicate the value of the two inputs, and the color represents the output value. The red dots are true (1), the blue dot is false (0). A linear separation is possible and an example is shown by the blue line. This means that a linear classifier can learn to perfectly separate the classes.

However, no linear function can separate the two classes for the XOR function, as shown in figure 3.4. To achieve this, we need multiple layers and nonlinear activation functions.

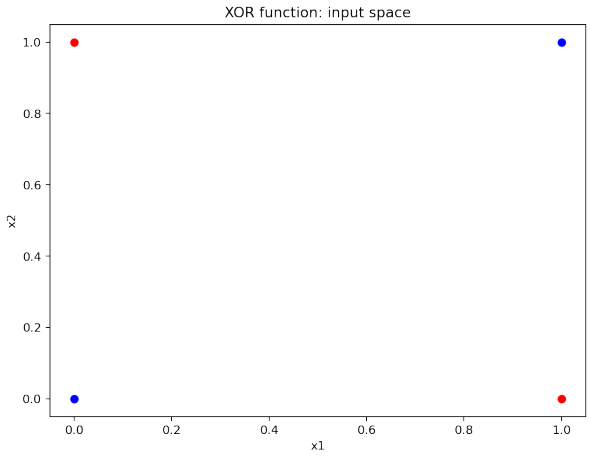


Figure 3.4: Visualization of the XOR problem. The XOR problem has two binary inputs and a binary output. A linear separation is *not* possible. This means that a linear classifier cannot learn to perfectly separate the classes.

3.1.3. Multi-layer perceptron

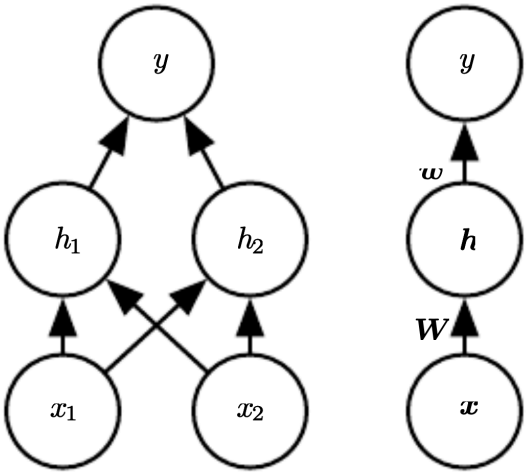


Figure 3.5: *(left)* An MLP uses multiple layers to learn nonlinear functions. This illustration shows one hidden layer h. An MLP can have any amount of hidden layers and neurons in each layer. Both hidden layers and the output layer have weights and biases. *(right)* A more abstract version of the network. In practice, we use vectors and matrices for efficient computation.

We can use a multi-layer perceptron (MLP) to learn nonlinear functions. We illustrate an example of an MLP in figure 3.5. Recall our equation was:

$$f(x; w, b) = x^\top w + b. \tag{3.3}$$

In our MLP, we use:

$$h = f^1(x; W_1, b_1) = W_1^\top x + b_1, \tag{3.4}$$

$$y = f^2(h; w_2, b_2) = w_2^\top h + b_2. \tag{3.5}$$

Here, W_1 is a matrix instead of a vector, as there are two hidden neurons, and thus four weight parameters. w_2 and b_2 have replaced the previous use of w and b. The full equation becomes

$$f(x; W_1, b_1, w_2, b_2) = f^2(f^1(x)). \tag{3.6}$$

Recall that our goal was to approximate a nonlinear function. It might appear that we have now achieved this. However, since all terms are linear, this equation is still linear. If we plug in Equation 3.4 into Equation 3.5, we get:

$$y = w_2^\top h + b_2 = w_2^\top (W_1^\top x + b_1) + b_2 = w_2 W_1 x + w_2 b_1 + b_2. \tag{3.7}$$

Using $a = w_2 W_1$ and $b = w_2 b_1 + b_2$, we can write this as

$$y = ax + b, \tag{3.8}$$

which is again linear. Adding more layers would not make the model nonlinear either. As our function is still linear, we still cannot separate the classes in the XOR problem. A usual solution is adding a nonlinear activation function.

3.1.4. Activation functions

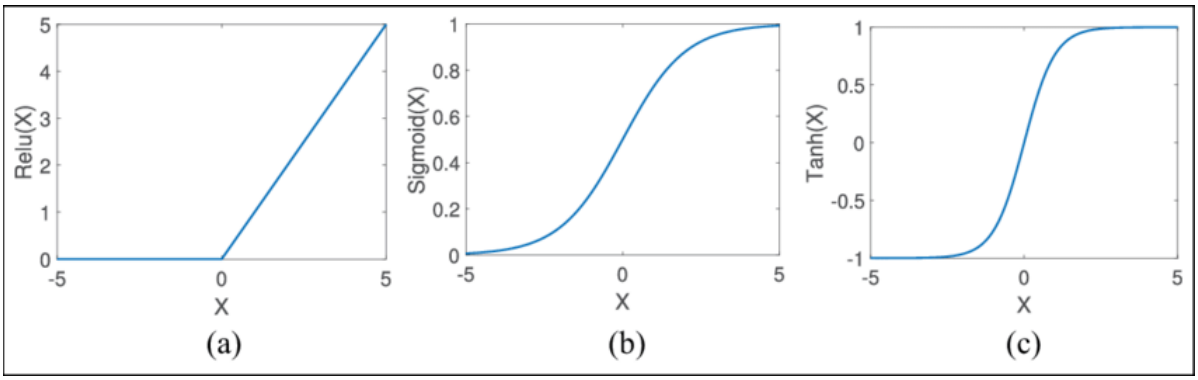


Figure 3.6: An illustration of three common activation functions [3]: (a) ReLU; (b) Sigmoid; and (c) Tanh.

A nonlinear activation function is used in neural networks to introduce nonlinearity. This allows us to approximate functions that are nonlinear, such as the XOR problem. Three commonly used activation functions are *rectified linear unit* (ReLU), *sigmoid*, and *tanh*. They are demonstrated in figure 3.6. We use ReLU in this example. It is defined as:

$$\sigma(x) = \max(0, x). \tag{3.9}$$

This turns our equation into:

$$f(x; W_1, b_1, w_2, b_2) = f^2(\sigma(f^1(x))). \tag{3.10}$$

After a few iterations, the first layer can now learn to create a hidden space where the two classes are linearly separable. In figure 3.7, we show a possible hidden space. The next layer can then learn two separate the two classes in this hidden space. The question remaining is now: how can the network learn such a hidden space? We explore this in the next section.

3.1.5. Back-propagation

So far we have explored network input. In problems like XOR, we pass two binary inputs to our network, and receive a single binary output. Producing our output y based on inputs x is called forward propagation.

The network needs to learn from the example inputs that we provide. To train the network, back-propagation is typically used. First, we calculate the loss of our network. Then, information of our loss function flows backward through the network, allowing us to calculate the gradient [6].

Following a notation style of Roger Grosse, we use bar notation to simplify notation of derivatives:

$$\bar{v} \triangleq \frac{\partial \mathcal{L}}{\partial v}, \tag{3.11}$$

where $\bar{v}$ is the partial derivative of the loss function $\mathcal{L}$ over a quantity v. Given our equations:

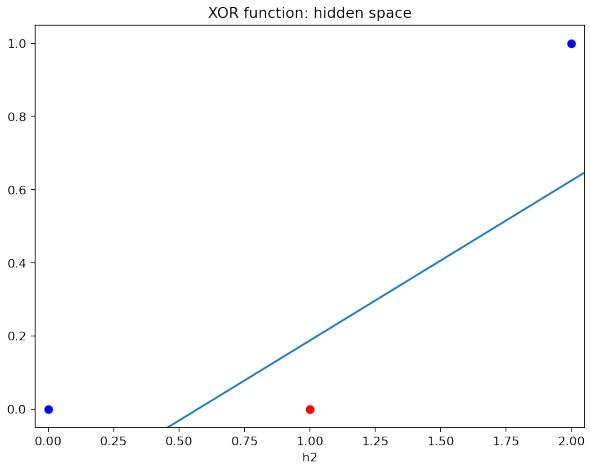


Figure 3.7: Visualization of a possible hidden space for the XOR problem. A linear separation in this learned space is now possible. This means that a linear classifier can learn to perfectly separate the classes, as demonstrated by the blue line.

$$z = wx + b, \quad y = \sigma(z), \quad \mathcal{L} = \frac{1}{2}(y - t)^2, \tag{3.12}$$

we can use the following equations for backpropagation:

$$\bar{y} = \frac{\partial \mathcal{L}}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(y - t)^2 = y - t, \tag{3.13}$$

$$\bar{z} = \bar{y}\frac{\partial y}{\partial z} = \bar{y}\sigma'(z), \tag{3.14}$$

$$\bar{w} = \bar{z}\frac{\partial z}{\partial w} = \bar{z}x, \tag{3.15}$$

$$\bar{b} = \bar{z}\frac{\partial z}{\partial b} = \bar{z}. \tag{3.16}$$

We measure how well we can approximate a function through loss. The goal is to minimize the loss. However, when moving a step through the gradient, the loss usually does not hit a minimum immediately. Thus, we propagate forward and backward many times. Each such iteration is called an *epoch*. At each epoch, we can update our parameters θ as follows:

$$\theta' = \theta - \epsilon \nabla_\theta J(\theta), \tag{3.17}$$

where ϵ is the *learning rate*, which determines how far we move into the negative direction of the gradient.

By updating our parameters using gradient descent, we can learn parameters that approximate functions. An MLP can approximate the mapping of any continuous function [9]. In the next section, we discuss several optimization techniques that help to approximate more difficult functions.

3.2. Optimization

3.2.1. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a stochastic extension of gradient descent. In large training sets, computing the loss function over all samples is often too computationally expensive. SGD works under the expectation that we can approximate the gradient of all samples using a minibatch of a few samples [6]. For instance, we might be able to approximate a dataset consisting of millions of samples with a few hundred samples. These samples come from a minibatch $\mathbb{B}$ with m' samples. We use the following equation for the gradient estimation:

$$\theta' = \theta - \epsilon \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x^i, y^i, \theta). \tag{3.18}$$

3.2.2. Learning rate

The learning rate indicates how fast we move into the negative direction of the gradient. In this section, we explore tuning this parameter. We create an example *loss space*, visualized in figure 3.8. Recall that our goal is to minimize this loss.

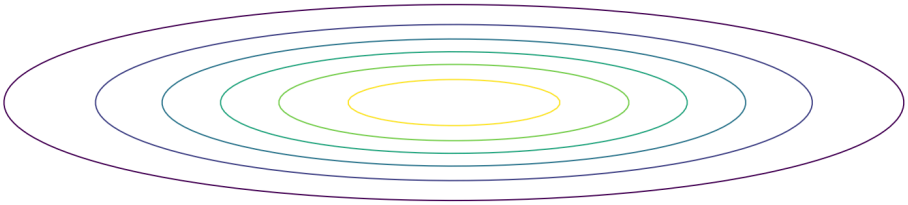


Figure 3.8: Visualization of a two-dimensional loss space. The center, yellow oval indicates the lowest loss: the loss increases as we move outwards. We aim to find the point of minimal loss. In this visualization, a human could easily just pick a point of minimal loss. However, in practice, we often have many more dimensions, and finding the minimum becomes difficult. Thus, we use gradient descent to minimize the loss.

With a learning rate that is too high, we might not *converge*. Convergence means that the network has found a minimum.

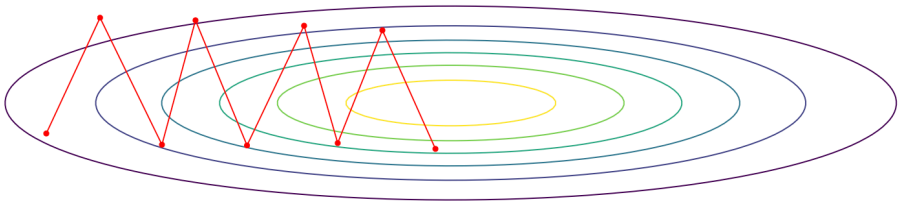


Figure 3.9: The red dots represent a loss value for a certain parameter configuration of the network. A line through the dots shows the progression of the loss attained. The dot on the far left is the first dot: we move inwards as we aim to minimize the loss, which is at the center of the figure. In this figure, due to a too high learning rate, we fail to converge.

With a learning rate that is too low, it might take long to converge.

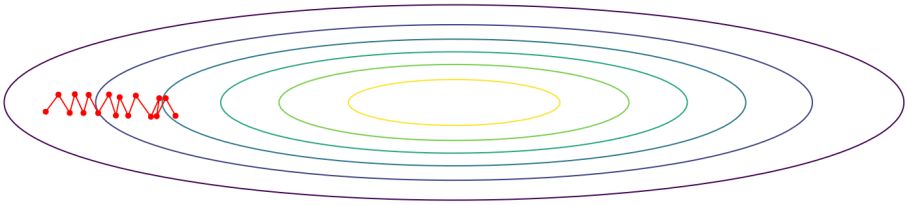


Figure 3.10: Due to a too low learning rate, it takes long to converge.

SGD calculates an expectation of the loss. Thus, the loss can be noisy: also towards the minimum.

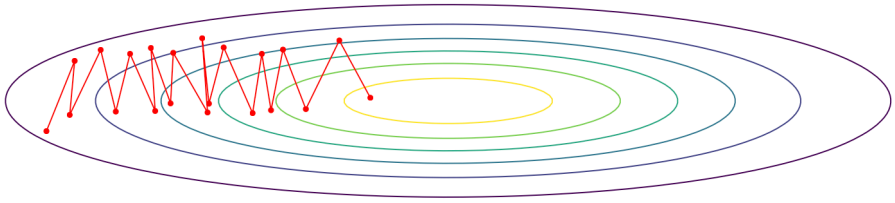


Figure 3.11: Since SGD is noisy, we might not converge.

We can *decay* the learning rate over time to find a good solution.

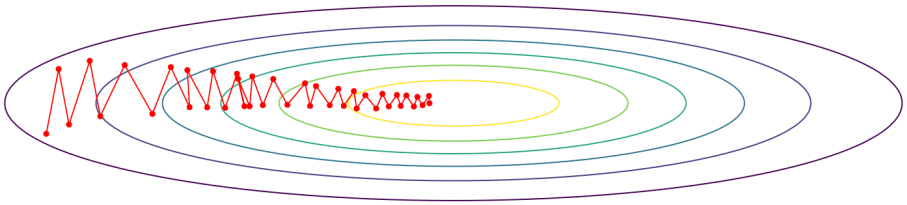


Figure 3.12: Using learning rate decay, we find the optimum after some epochs.

However, the convergence is quite slow. We can do better by having separate learning rates for each parameter. In the next three sections, we explore three gradient descent optimization techniques.

3.2.3. Momentum

Momentum aims to accelerate learning. The name momentum comes from physics, where a force moves a particle through a space [6]. We try to accelerate the learning in a direction where the gradient updates decrease the loss, and decelerate the learning in a direction where the gradient updates do not decrease the loss as much. We use figure 3.11 to give an intuition behind this algorithm. Note that in this figure, the gradient updates oscillate vertically, while we want to move to the right. We can keep a moving average of the past few gradients and use them to indicate how much we should accelerate or decelerate gradient descent:

- In the past few gradients in the vertical direction, the gradient is mostly oscillating, so the average will be close to zero.
- In the past few gradients in the horizontal direction, the gradient moves to the right, so the average will be large.

We update the parameters based on a moving average of the gradients, so that the algorithm takes a less oscillated path towards the minimum than SGD.

Momentum is implemented as follows. We use v as the *velocity*, initialized at zero. It is similar to the momentum of a particle. We decay v by $\alpha \in [0, 1)$ after each epoch. This α parameter acts similarly to friction in physics: it decays previous contributions exponentially. By doing this, we are calculating an exponentially weighted average of the gradient. This speeds up learning by increasing the step size for parameters with large gradients, and reduces oscillations by decreasing the step size for parameters with small gradients. Using g to indicate the gradient, we can formulate momentum as follows [6]:

$$v \leftarrow \alpha v - \epsilon g, \tag{3.19}$$

$$\theta \leftarrow \theta + v. \tag{3.20}$$

We ignore bias correction with this formulation as it is often not implemented in practice.

3.2.4. RMSProp

Similarly to momentum, Root Mean Squared Propagation (RMSProp) [11] aims to reduce noisiness of the gradient updates. This effect is illustrated in figure 3.11, where the gradient updates move much

vertically, which does not help to converge. Similarly to momentum, we aim to move faster in the horizontal direction, where the gradient steps are large, and move slower in the vertical direction, where the gradient steps are small.

At each epoch, RMSProp calculates an exponentially weighted average of the squares of the derivatives. When the derivative is sloped more steeply, such as in the vertical direction in figure 3.11, the weighted average of the squares of the derivative is larger. We can divide by this number to reduce oscillations. We keep track of a weighted moving average, r, which is initialized at zero. We use a decay rate $\rho \in [0, 1)$. The following equations allow us to implement RMSProp [6]:

$$r \leftarrow \rho r + (1 - \rho) g \odot g, \tag{3.21}$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\sqrt{r + \delta}} \odot g, \tag{3.22}$$

$$\theta \leftarrow \theta + \Delta\theta. \tag{3.23}$$

Here, δ is small number, such as 1e-6, which is added to prevent dividing by zero. We again ignore bias correction with this formulation as it is often not implemented in practice.

3.2.5. Adam

Adam [15], derived from *adaptive moment estimation*, essentially combines momentum and RMSProp. We use ρ_1 and ρ_2 as decay rates. t indicates the time step. We can use the following equations to implement Adam [6]:

To update the first and second moment estimates, we use:

$$s \leftarrow \rho_1 s + (1 - \rho_1) g, \tag{3.24}$$

$$r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g. \tag{3.25}$$

To correct for bias in the moments, we use:

$$\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}, \tag{3.26}$$

$$\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}. \tag{3.27}$$

Finally, to compute and apply the update, we use:

$$\Delta\theta \leftarrow -\epsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}, \tag{3.28}$$

$$\theta \leftarrow \theta + \Delta\theta. \tag{3.29}$$

3.2.6. Batch normalization

Normalizing input features can speed up learning. We refer to normalizing as centering and normalizing values to a zero mean and unit variance. Normalizing speeds up training because it equalizes impact of features on the loss function regardless of its possible values. For instance, if one feature ranges from $[0, 1]$, and another feature ranges from $[0, 1000]$, then the second feature will have a much larger impact on the loss.

Beyond normalizing input features, we could also normalize features in the hidden layers of the network. For this, batch normalization [12] is commonly used.

We can use the following equations to normalize our hidden layers:

$$\mu_i = \frac{1}{n} \sum_{j=1}^{n} z^j, \tag{3.30}$$

$$\sigma_i^2 = \frac{1}{n} \sum_{i=1}^{n} (z^j - \mu_i)^2, \tag{3.31}$$

$$z_i^{norm} = \frac{Z - \mu}{\sqrt{\delta + \sigma^2}}. \tag{3.32}$$

Hidden layers should not always have a zero mean and unit variance. Thus, we introduce two learnable parameters, γ and β.

$$z_i^{norm} = \gamma_i z_i^{norm} + \beta_i. \tag{3.33}$$

γ and β are updated the same way as the weights of the neural network.

3.3. Regularization

A model *generalizes* if it performs well on inputs it has not been trained on. To understand the relevance of this, consider a case where a network aims to classify dogs and wolves. As in the dataset, all pictures of wolves have snow in them, the network simply learns to classify dogs and wolfs based on snow, regardless of which animal is in it [24]. However, in unobserved inputs, an image with a wolf might not have any snow, likely leading to a bad prediction.

The most straightforward approach to verifying if our model generalizes, is to split our data into a *training* and a *test* set. The model trains only on the training set; it is evaluated on the test set.

A model can *underfit* when the model cannot get good performance on any set. A model can *overfit* by performing much better on the training set than on the test set, which indicates a lack of generalization. Regularization is a method which aims to reduce overfitting. In this section, we outline a few common approaches towards regularization.

3.3.1. Early stopping

When a model overfits, its error on the training set - the training error - tends to decrease over time, while the test error begins to increase [6]. This is illustrated in figure 3.13. Early stopping refers to creating a copy of the model every time the test error is at a new minimum. We then use this model for later inference rather than the most recent one. Depending on the implementation, the model training is halted when the test error starts to increase.

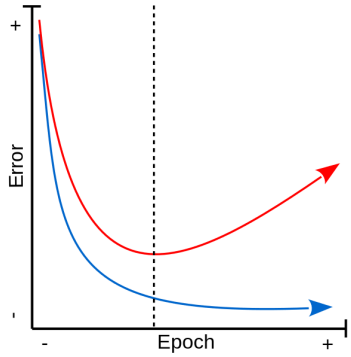


Figure 3.13: An illustration of model error on a model that overfits. The blue line indicates the train error. The red line indicates the test error, which starts to increase after a while. At the dotted line, the model starts to overfit. Using early stopping, we return the model with best performance on the test set.

3.3.2. Weight decay

Weight decay is a technique that aims to avoid weight parameters with high values. Such parameters can introduce high variance. As a possible solution, we can add the sum of squares of the weights to the loss. The idea is that smaller weights lead to less variance, which means a more generalized model. We can update our loss to the following to implement weight decay [6]:

$$J(\theta) = \text{MSE}_{\text{train}} + \lambda w^\top w, \tag{3.34}$$

where λ controls the amount of regularization.

3.3.3. Batch normalization

Batch normalization has a small regularization effect on the gradients [30]. The mean and variance statistics are calculated on the minibatch. The statistics on each minibatch differ slightly from each

other. This difference introduces random jitter perturbations to the partition boundaries, which has a regularizing effect [1].

3.3.4. Data augmentation

Deep learning algorithms typically regularize more with more data. However, acquiring data is usually expensive. Data augmentation is a common technique to expand the current dataset. It is particularly helpful on small datasets.

We use three types of augmentations:

- horizontal flipping;
- rotation: randomly uniform between $[-30°, 30°]$;
- cropping; zooming in by a factor of 1.5 and taking a random crop within this image.

We illustrate an example of an augmented strawberry sample in figure 3.14.

Figure 3.14: Three data augmentation techniques on a strawberry sample. Each technique is applied subsequently on the results of the previous one, leading to $2^3 = 8$ permutations. The original strawberry is at the top left. The bottom left strawberry is horizontally flipped. The second column displays random rotation. The third and fourth column displays random cropping. The augmenting process results in more data to train on.

Data augmentation techniques should be chosen based on the task. For example, for strawberry quality prediction, augmentations that alter colors might not be suitable, as quality parameters depend on color.

3.4. Convolutional neural networks

MLPs can be used for computer vision tasks, such as image classification. However, since MLPs are fully connected, they have a large amount of parameters. For example, an MLP with 1000 hidden units and a 256x256 RGB input image has 256x256x3x1000 = 197 million parameters in the first layer. It is not computationally feasible to use an MLP for large images.

Instead, we can use a convolutional neural network (CNN). A CNN is a neural network that uses convolutional operations and is often used for image analysis.

3.4.1. The convolution operation

We can use a *filter* to detect a certain pattern in an image. An example filter, shown in figure 3.15, acts as a vertical edge detector.

1	0	-1
1	0	-1
1	0	-1

Figure 3.15: An example of a 3x3 filter that detects vertical edges.

We can apply this filter on an input image, which is represented as a matrix. A *convolution* operation is an element-wise matrix multiplication. The first part of the multiplication is an input matrix, *i.e.*, the input image. The second part of the multiplication is a filter.

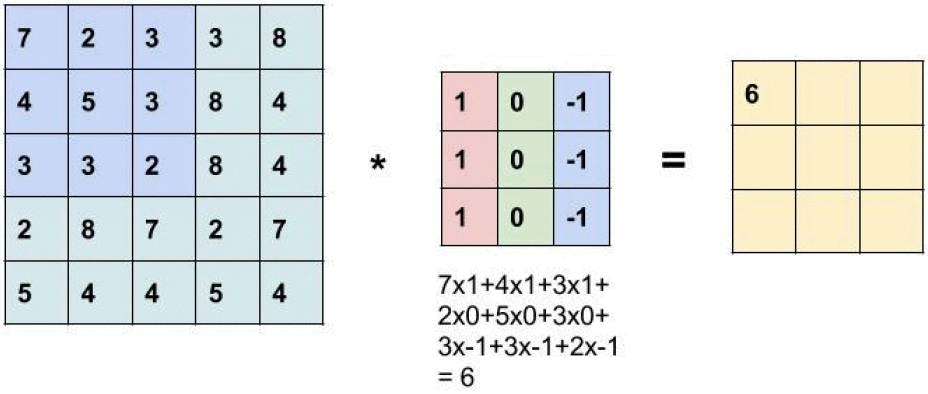


Figure 3.16: A single output of a convolution operation. The filter in the middle is multiplied element-wise with the top-left part of the image. The output goes to the top right of the output image.

We can apply the same multiplication on the entire input matrix.

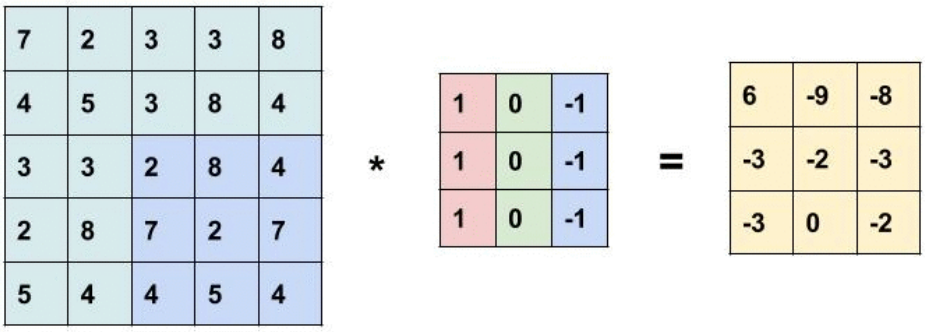


Figure 3.17: The full output of the convolution operation.

This example used a vertical edge detection filter. In practice, the values in the filter are the weights of the network. The neural network learns the weights over time.

Technically, the convolution operation first flips the filter both horizontally and vertically. The term *cross-correlation* defines an operation where this action is omitted. However, in practice, literature refers to this operation - without flipping - as convolution. The flipping is not implemented for simplicity.

3.4.2. Padding

As demonstrated in figure 3.17, the size of the output becomes smaller after the convolution operation. This has two disadvantages. First, the image shrinks over time: when using many layers, we end up

with an output image that is much smaller than the input image. Second, the information at the edges is used less than the information towards the center. For instance, with a 3x3 filter, a pixel at the corner is used once, while a pixel in the center is used 9 times.

A common solution to this is padding. If our input matrix is $n * n$ and our filter is $f * f$, then the output of convolution is $(n - f + 1) * (n - f + 1)$. So, a 5x5 input with a 3x3 filter gives a 3x3 output. When padding the image, we add a border to the image, *i.e.*, we add pixels on all sides of the image. Padding is typically applied with zeroes, called *zero-padding*. After padding the image with p pixels at each side of the image, the size becomes $(n + 2p - f + 1) * (n + 2p - f + 1)$. Using $p = 1$, we get a 5x5 output: the image size stays the same. The result is that pixel values at the edge are convolved over just as much as pixel values towards the center.

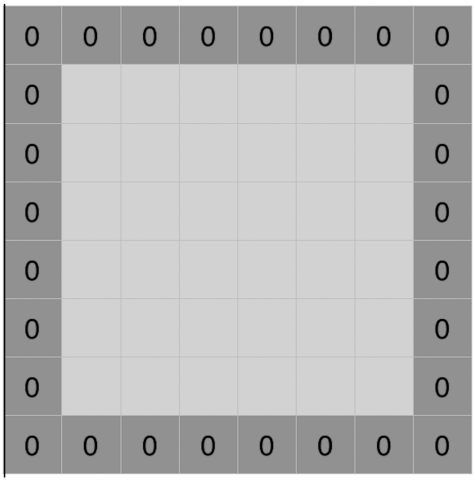


Figure 3.18: Padding is applied by appending pixels on the all sides of the image. This figure shows a border of 1 pixel.

Given that the output size is $(n + 2p - f + 1) * (n + 2p - f + 1)$, we can find that using $p = \frac{f-1}{2}$, the output size is equal to the input size.

There are three common types of convolution, each with their own amount of padding:

- valid convolution: here, no padding is applied, thus the output shrinks by $f - 1$ pixels;
- same convolution: by padding the input with $\frac{f-1}{2}$ pixels, our output size is equal to the input size;
- full convolution: by padding the input with $f - 1$ pixels, our output size increases by $f - 1$ pixels.

The choice of type of convolution depends on the goal: to either decrease, keep, or increase the image size.

3.4.3. Stride

We can add a level of stride to the convolution operation. The stride is an integer that indicates with how many steps the filter moves. The default stride is 1. When using a stride of 2, the filter moves 2 steps at a time. The result is a smaller output image. The dimensions when using a stride are:

$$\lfloor \frac{n + 2p - f}{s} + 1 \rfloor * \lfloor \frac{n + 2p - f}{s} + 1 \rfloor. \tag{3.35}$$

For example, in the output of figure 3.19, each dimension is now $\lfloor \frac{7+0-3}{2} + 1 \rfloor = 3$.

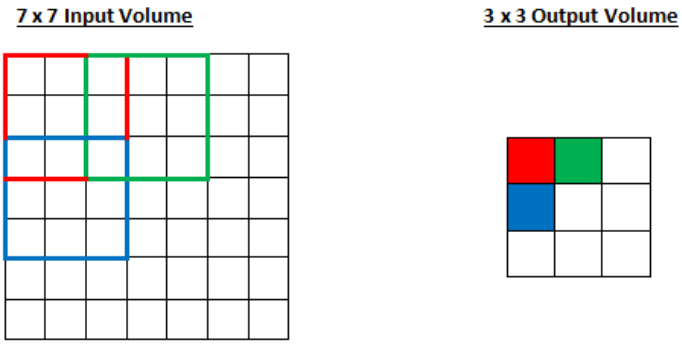


Figure 3.19: A 7x7 input is convolved over using a 3x3 filter and a stride of 2. The stride indicates the step size of the convolution operation. The output size is 3x3.

3.4.4. Pooling

Pooling is an operation that decreases the size of feature maps by summarizing the features. It increases translation invariance [6]. It is computed per feature channel. Two common types of pooling are *max pooling* and *average pooling*. Empirically, max pooling gives better results, and is thus used more commonly. Max pooling takes the maximum of each region, while average pooling takes the average of each region.

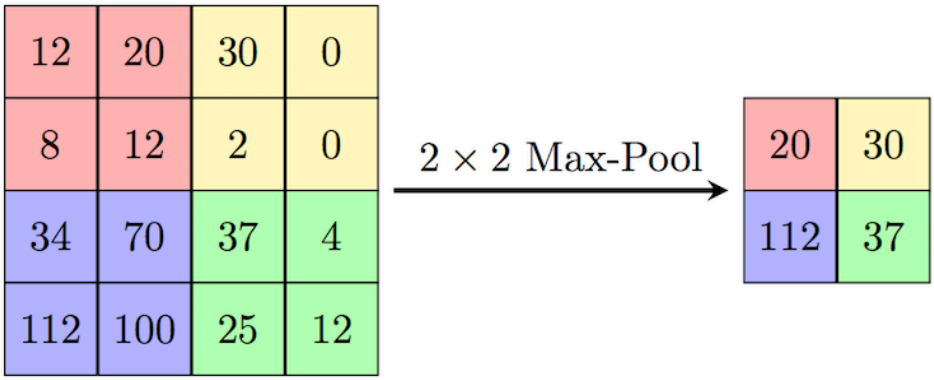


Figure 3.20: A max pooling operation. The maximum value of each region, denoted by a distinct color, is chosen for the output. A 2x2 pooling operation reduces the output size by a factor 4. Pooling operations have no learnable parameters.

3.4.5. Networks

A network typically uses many consecutive layers of convolution. Each layer applies a few steps, as visualized in figure 3.21:

- convolve the input image with filters;
- apply a non-linearity function, such as ReLU;
- apply a pooling operation.

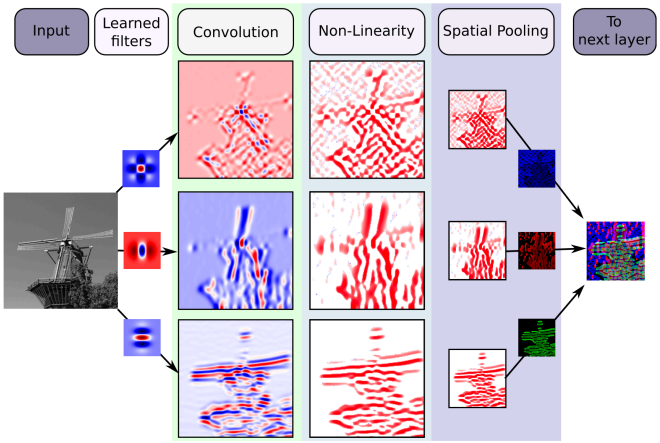


Figure 3.21: A visualization of a possible input image processed by a convolutional layer. We apply convolution, non-linearity, and spatial pooling, and the result moves to the next layer.

In a CNN, filters with different sizes allow the network to look at the image from a low-level and a high-level. Relatively larger filter sizes means that a filter operates on a larger part of the image. Rather than increasing the filter size, we decrease the image size. This has a similar effect, but is more efficient, as smaller images reduce computation time. As the image size shrinks throughout the layers, deeper layers detect more high level features, such as shapes; earlier layers detect more low level features, such as edges.

3.4.6. Transposed convolutions

After subsampling operations such as pooling, the image size shrinks. Transposed convolutions are convolution operations that allow us to increase the image size, rather than decrease it.

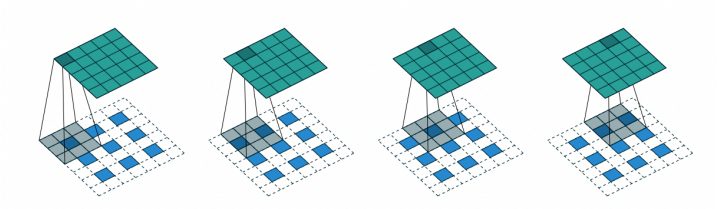


Figure 3.22: Transposed convolutions with a 5x5 output [4]. The green matrix is the output of the operation; the blue matrix is the input. The blue matrix is 3x3, but padded with a 1 pixel border and a stride of 2 pixels.

3.4.7. A few notable CNN architectures

LeNet LeNet [16] was one of the earlier convolutional neural networks. It consists of convolutional, pooling, and fully connected layers. LeNet used average pooling and a Tanh activation function.

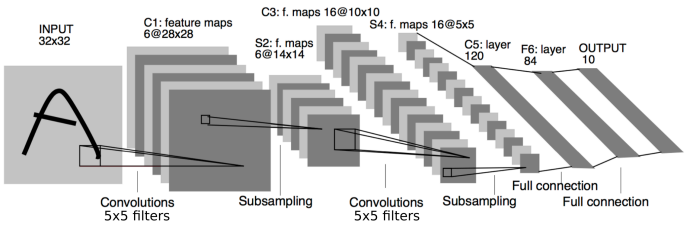


Figure 3.23: LeNet [16]. A 32x32 black and white input is passed through three convolutional layers, two pooling layers, and two fully connected layers.

ResNet Many visual tasks have benefited from deep models [10]. However, as depth increases, accuracy eventually decreases. One problem that causes this is *vanishing/exploding gradients*. This refers

to gradients that are repeatedly multiplied until they either hit zero or infinity, hurting the performance of the network. Indeed, certain CNNs have been shown to perform worse with an increasing amount of layers [10].

A residual neural network (ResNet) [10] uses *shortcut connections* that can skip layers. The shortcut connections allow input to go through less layers, which reduces the impact of vanishing/exploding gradients.

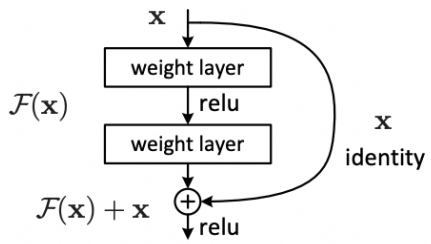


Figure 3.24: A residual learning building block [10] used in ResNet. Inputs can bypass layers by going through the shortcut connection. The shortcut connection is an identity function.

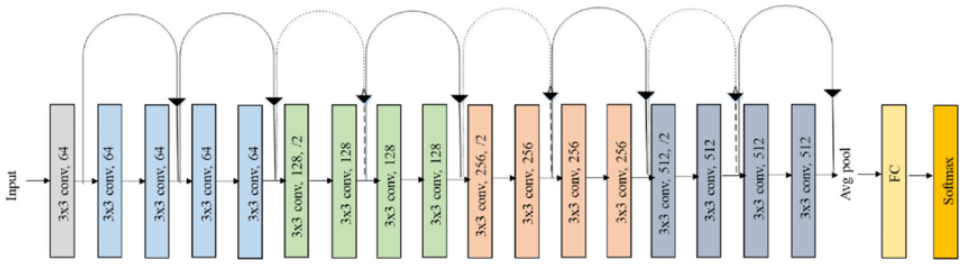


Figure 3.25: ResNet-18 is a variant of ResNet with 18 layers. The top arrows indicate shortcut connections. These shortcut connections reduce the vanishing/exploding gradient problem [10].

U-Net U-Net [25] is a CNN originally made for biomedical image processing. In such tasks, localization should often be outputted [25]. The architecture is shown in figure 3.26. The network has a contractive path on the left and an expansive path on the right. This creates a U-like shape, hence the name, U-Net. The contractive path consists of 3x3 convolutions, ReLU, and 2x2 max pooling with stride 2. The expansive path replaces the pooling operation with a 2x2 transposed convolution.

As the architecture in the original paper does not use padding, the output size is smaller than the input size. Later implementations[1] of the network added padding to make the output and input sizes equal, which makes the network more suitable for segmentation. We also use padding in our scientific paper so that the sizes stay the same.

[1] PyTorch implementation of U-Net on Github. `https://github.com/jvanvugt/pytorch-unet/blob/master/README.md`

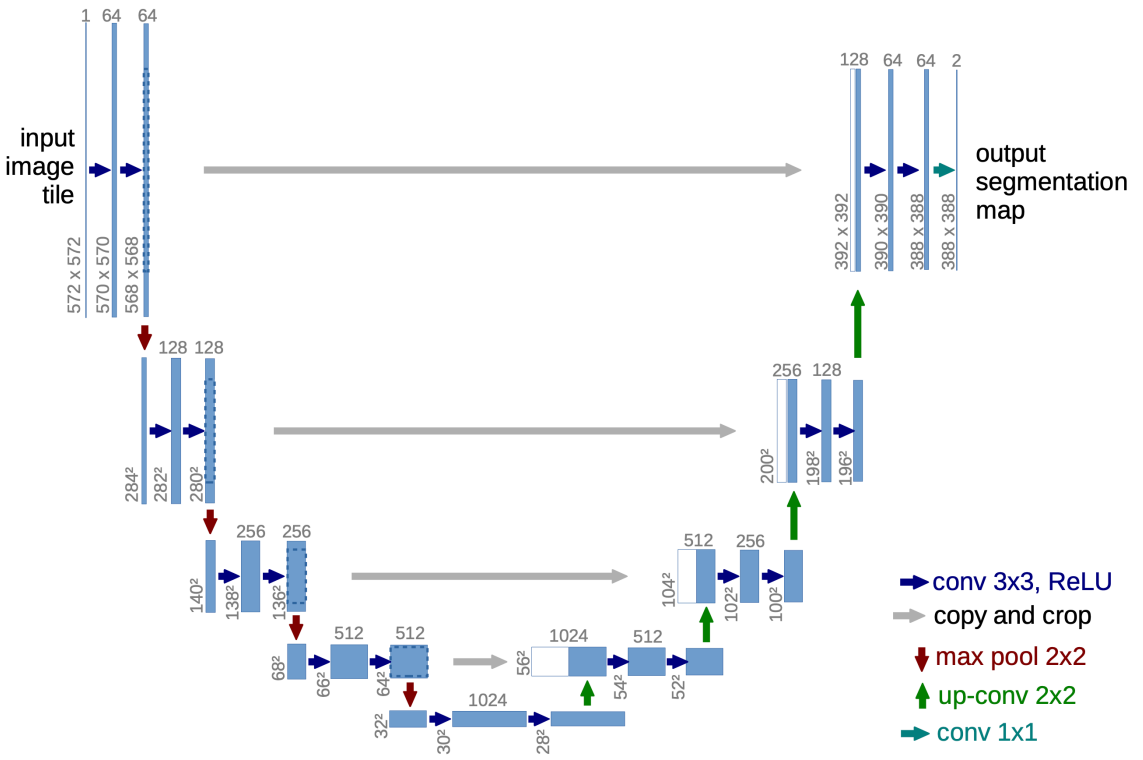


Figure 3.26: U-Net [25]. The numbers at the top indicate the number of channels and the number on the side indicates the image size. A blue box represents a feature map and a white box represents a copied feature map. The horizontal lines indicate concatenations for upsampling. The left part of the network reduces the image size and increases the number of channels; the right part of the network increases the image size and reduces the number of channels.

4

Strawberry quality

This chapter aims to provide an answer to the following question: *why are we able to determine internal quality attributes of strawberries from pictures and environment data?*

4.1. Ripening process and chemical constituents

One way that fruit ripening is observed is through color change. Fruit ripening usually involves changes in pigment concentrations. Mainly, chlorophyll is degraded and other pigments, such as anthocyanins, are biosynthesized [26]. Chemical concentrations can be observed visually: anthocyanin and chlorophyll pigments have been shown to be visible in the regions around 535 nm and 680 nm [18]. These regions are visible to the eye.

4.1.1. Photosynthesis and chlorophyll

Photosynthesis is a process where plants synthesize nutrients from carbon dioxide and water under influence of sunlight. This is an endothermic reaction, which means it absorbs heat. Thus, sunlight is necessary. The following equation can represent the reaction [8]:

$$CO_2 + H_2O \xrightarrow[\text{plants}]{\text{sunlight}} [CH_2O] + O_2. \tag{4.1}$$

Plants contain chloroplasts, which store the energy of sunlight [27]. The chloroplast contains the pigment chlorophyll, which is a light-absorbing pigment. It allows plants to absorb energy from light. Specifically, it absorbs energy from red and blue wavelengths and reflects green wavelengths. This makes fruit appear green.

Chlorophyll content is directly linked to sugar content [14]. During the ripening process of plants, chlorophyll is degraded [2]. Thus, a change in color due to a change in chlorophyll concentrations can be observed: the fruit becomes less green.

4.1.2. Anthocyanins and carotenoids

Anthocyanins are pigments that give a red or blue color [26]. They are common in strawberries [31]. It is synthesized during the ripening process [13]. They are not involved in photosynthesis, but serve other purposes, such as protection against UV light [14].

Unlike anthocyanins, carotenoids are essential in photosynthesis [7]. Strawberries have low carotenoid content, which decreases during ripening. Carotenoids expand the wavelength range that a plant is able to drive photosynthesis in the following way [7]: chlorophylls cannot absorb light well in the 450-550nm region, yet carotenoids absorb light strongly here. They transfer this energy to the chlorophylls. Further, like anthocyanins, carotenoids prevent photo-damage under conditions of excess light.

4.1.3. Firmness and shelf life

Fruit firmness decreases over time, contributing to its quality. It is largely due to cell wall degradation [26]. This loss of firmness results in a short shelf life [20]. Strawberries that lose too much firmness could lead to waste [19]. Pectin, a structural fiber, forms a major component of the cell walls, which

are extensively modified during ripening [19]. During fruit ripening, pectin is broken down by enzymes. Enzymes are proteins that catalyze chemical reactions. The breakdown of pectin makes the fruit softer as cells are separated from each other [22].

4.2. Environment data

Environmental data significantly impact quality attributes such as sweetness. In this section, we outline a few possible causes of these impacts of two relevant environment factors: temperature and carbon dioxide.

Temperature affects fruit photosynthesis [29]. The reactions of photosynthesis are controlled by enzymes [5]. Enzymes work most efficiently at around 10-20 degrees Celsius [17]. At lower or higher temperatures, enzymes work less efficiently, which decreases photosynthetic rate. This results in less glucose content in strawberries.

Elevated levels of carbon dioxide (CO_2) significantly increases sweetness in strawberry fruits [28]. Higher CO_2 results in higher *dry weight*. Dry weight is a measure of the weight of a fruit after drying. The effect could be due to higher photosynthetic rate.

5

Stereo vision

In our greenhouse, two cameras are placed in parallel, which take pictures of the strawberry plants. Given our stereo setup, we can calculate the *depth* of a strawberry. The depth is the length from the cameras to the strawberry, parallel to the cameras. Our markets divide strawberries based on their width in millimeters, but we are only able to measure the width in pixels. These cannot trivially be converted: a strawberry that takes up the same amount of pixels, but is further away, is actually larger. Although we know that the distance from the cameras to the strawberries is roughly 900mm, it varies by around 100mm. With stereo vision, we are able to calculate the depth; using the depth, we are able to calculate the size in millimeters.

This chapter is structured as follows. In section 5.1, we explain how we calculate the focal length of our cameras, which we need to calculate the depth. In section 5.2, we explain how we calculate depth.

5.1. Focal length calculation

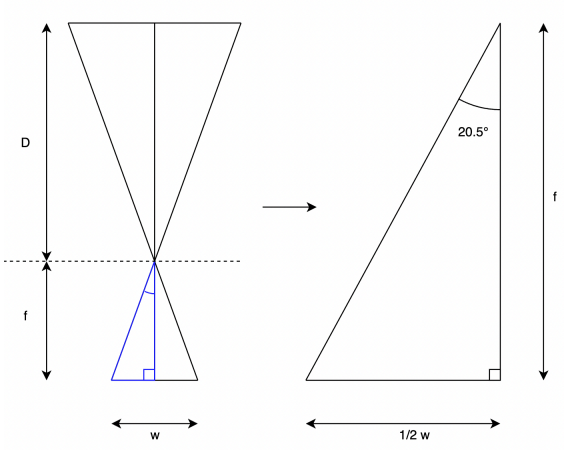


Figure 5.1: We can calculate the focal length using trigonometric ratios. In the left side of the image, the bottom line is the camera film; the top line is the where the strawberries are, roughly. The right side of the image is a zoomed-in version of the blue part of the left side of the image. We can find the length of the opposite (focal length, f) using the length of the adjacent (image width, $\frac{1}{2}w$) and the angle (camera angle, θ).

We can find the focal length of a camera using trigonometric ratios. We know the width of the pictures taken by the camera in pixels, namely, $w = 4000$px. We also know that the angle of the camera is $\theta = 41°$.

We can create a triangle with an angle of $\frac{41°}{2} = 20.5°$ by drawing a line from the center of the projection in the camera to the center of the reality. We know that opposite angles are equal, therefore, in the blue triangle, our angle is $\frac{41°}{2} = 20.5°$ as well. We illustrate this in figure 5.1.

In our triangle, $\frac{1}{2}w$ is the *adjacent* line and $\frac{\theta}{2} = 20.5°$ is our angle. We can use this to find the focal length, which is the *opposite* line, with the following equation:

$$\tan(\frac{\theta}{2}) = \frac{\frac{1}{2}w}{f}. \tag{5.1}$$

To get the focal length, we rewrite this to:

$$f = \frac{w}{2\tan(\frac{\theta}{2})}. \tag{5.2}$$

Using details of our cameras, we find:

$$f = \frac{4000\text{px}}{2\tan(\frac{41°}{2})} \approx 5349\text{px}. \tag{5.3}$$

The focal length allows us to calculate the depth.

5.2. Depth calculation

Similar triangles can be used to find an equation for depth [21]. We illustrate our setup in figure 5.2. A simplified version of this figure, which only has the relevant details for triangulation, is shown in figure 5.3.

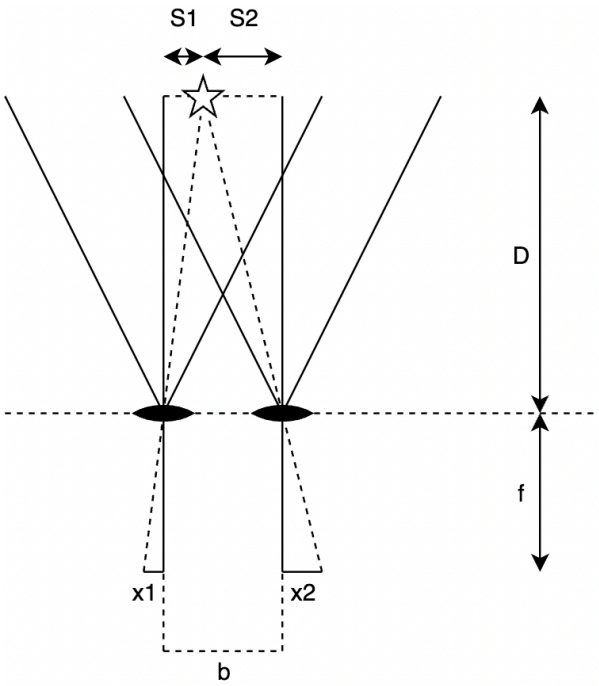


Figure 5.2: Our setup. We illustrate our two parallel cameras in black in the middle of image. They point at an example object, here denoted as a star.

x_1 and x_2 are the disparities in pixels, observed by the two cameras, and S_1 and S_2 are the disparities in millimeters. Using similar triangles, we find the following two equations:

$$\frac{S_1}{x_1} = \frac{D}{f} \quad \text{and} \quad \frac{S_2}{x_2} = \frac{D}{f}. \tag{5.4}$$

Rewriting leads to

$$S_1 = \frac{x_1 D}{f} \quad \text{and} \quad S_2 = \frac{x_2 D}{f}. \tag{5.5}$$

Observing figure 5.2, we see that the following equation holds:

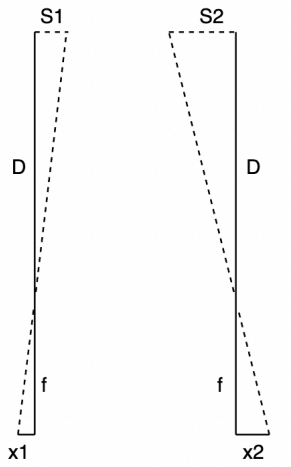


Figure 5.3: A simplified version of figure 5.2 that only shows relevant details for triangulation.

$$b = S_1 + S_2. \tag{5.6}$$

Inserting the values from Equation 5.5 gives:

$$b = \frac{x_1 D}{f} + \frac{x_2 D}{f} = \frac{(x_1 + x_2)D}{f}. \tag{5.7}$$

We can define the disparity d as the sum of the disparities in pixels x_1 and x_2:

$$d = x_1 + x_2. \tag{5.8}$$

Plugging this into the previous equation gives us:

$$b = \frac{dD}{f}. \tag{5.9}$$

Finally, rewriting to obtain the depth gives

$$D = \frac{bf}{d}. \tag{5.10}$$

The baseline between the cameras and the focal length are constant. This means that the depth is a function of the disparity: the depth is inversely proportional to the disparity.

Finally, using the same triangulation method as described in section 5.1, we calculate the size of strawberry in millimeters, given the depth and the size of a strawberry in pixels. This is described in detail in section 3.5 of the scientific paper.

References

[1] Randall Balestriero and Richard G. Baraniuk. *Batch Normalization Explained*. 2022. DOI: 10.48550/ARXIV.2209.14778. URL: https://arxiv.org/abs/2209.14778.

[2] C J Brady. “Fruit Ripening”. In: *Annual Review of Plant Physiology* 38.1 (1987), pp. 155–178. DOI: 10.1146/annurev.pp.38.060187.001103. eprint: https://doi.org/10.1146/annurev.pp.38.060187.001103. URL: https://doi.org/10.1146/annurev.pp.38.060187.001103.

[3] Jason Carson et al. “Artificial intelligence approaches to predict coronary stenosis severity using non-invasive fractional flow reserve”. In: *Proceedings of the Institution of Mechanical Engineers Part H Journal of Engineering in Medicine* 234 (Aug. 2020). DOI: 10.1177/0954411920946526.

[4] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. DOI: 10.48550/ARXIV.1603.07285. URL: https://arxiv.org/abs/1603.07285.

[5] *Enzymes*. URL: https://www.bbc.co.uk/bitesize/guides/z9pjrwx/revision/2 (visited on 09/30/2022).

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[7] Hashimoto H, Uragami C, and Cogdell RJ. “Carotenoids and Photosynthesis”. In: *Subcell Biochem* 79 (2016), pp. 111–39. DOI: 10.1007/978-3-319-39126-7_4.

[8] D. O. Hall and K. K. Rao. *Photosynthesis*. 6th ed. Cambridge University Press, 1999.

[9] Eric J. Hartman, James D. Keeler, and Jacek M. Kowalski. “Layered Neural Networks with Gaussian Hidden Units as Universal Approximations”. In: *Neural Computation* 2.2 (June 1990), pp. 210–215. ISSN: 0899-7667. DOI: 10.1162/neco.1990.2.2.210. eprint: https://direct.mit.edu/neco/article-pdf/2/2/210/811985/neco.1990.2.2.210.pdf. URL: https://doi.org/10.1162/neco.1990.2.2.210.

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[11] Geoff Hinton. *Coursera, video lectures*. 2012.

[12] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167.

[13] M.N. Islam, Mehnaz Mursalat, and Mohidus Samad Khan. “A review on the legislative aspect of artificial fruit ripening”. In: 5 (June 2016). DOI: 10.1186/s4006601600575.

[14] Leepica Kapoor et al. “Fruit ripening: dynamics and integrated analysis of carotenoids and anthocyanins”. In: *BMC Plant Biology* 22 (Jan. 2022). DOI: 10.1186/s12870-021-03411-w.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[16] Yann Lecun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[17] Samuel Markings. *The Effect of Temperature on the Rate of Photosynthesis*. 2018. URL: https://sciencing.com/effect-temperature-rate-photosynthesis-19595.html (visited on 09/30/2022).

[18] Binu Melit Devassy and Sony George. “Estimation of strawberry firmness using hyperspectral imaging: a comparison of regression models”. In: *Journal of Spectral Imaging* 10 (June 2021). DOI: 10.1255/jsi.2021.a3.

[19] Candelas Paniagua et al. “Fruit softening and pectin disassembly: An overview of nanostructural pectin modifications assessed by atomic force microscopy”. In: *Annals of botany* 114 (July 2014). DOI: 10.1093/aob/mcu149.

[20] Candelas Paniagua et al. “Structural changes in cell wall pectins during strawberry fruit development”. In: *Plant Physiology and Biochemistry* 118 (2017), pp. 55–63. ISSN: 0981-9428. DOI: https://doi.org/10.1016/j.plaphy.2017.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0981942817301791.

[21] Dhaval K. Patel, Pankaj A. Bachani, and Nirav R. Shah. “Distance Measurement System Using Binocular Stereo Vision Approach”. In: *International journal of engineering research and technology* 2 (2013).

[22] *Pectin*. 2022. URL: https://en.wikipedia.org/wiki/Pectin#cite_note-Grierson-7 (visited on 09/25/2022).

[23] *Perceptron*. 2022. URL: https://nl.wikipedia.org/wiki/Perceptron (visited on 10/07/2022).

[24] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *”Why Should I Trust You?”: Explaining the Predictions of Any Classifier*. 2016. DOI: 10.48550/ARXIV.1602.04938. URL: https://arxiv.org/abs/1602.04938.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

[26] Graham B. Seymour, Jane E. Taylor, and Gregory A. Tucker. *Biochemistry of Fruit Ripening*. 1st ed. Dordrecht: Springer Dordrecht, 1993.

[27] National Geographic Society. *Photosynthesis*. 2022. URL: https://education.nationalgeographic.org/resource/photosynthesis (visited on 09/25/2022).

[28] SY Wang and JA Bunce. “Elevated carbon dioxide affects fruit flavor in field-grown strawberries (Fragaria × ananassa Duch)”. In: *Journal of the Science of Food and Agriculture* 84.12 (2004), pp. 1464–1468. DOI: https://doi.org/10.1002/jsfa.1824. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jsfa.1824. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jsfa.1824.

[29] Wu X.Y. et al. “The difference in temperature between day and night affects the strawberry soluble sugar content by influencing the photosynthesis, respiration and sucrose phosphatase synthase.” In: *Hort. Sci. (Prague)* 48 (2021), pp. 174–182. DOI: 10.17221/169/2020-hortsci.

[30] Greg Yang et al. *A Mean Field Theory of Batch Normalization*. 2019. DOI: 10.48550/ARXIV.1902.08129. URL: https://arxiv.org/abs/1902.08129.

[31] Yosuke Yoshioka et al. “Use of image analysis to estimate anthocyanin and UV-excited fluorescent phenolic compound levels in strawberry fruit”. In: *Breeding science* 63 (June 2013), pp. 211–7. DOI: 10.1270/jsbbs.63.211.

[32] Chu zhang et al. “Hyperspectral imaging analysis for ripeness evaluation of strawberry with support vector machine”. In: *Journal of Food Engineering* 179 (2016), pp. 11–18. ISSN: 0260-8774. DOI: https://doi.org/10.1016/j.jfoodeng.2016.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0260877416300024.