Assessment of Machine Learning Methods for the Classification and Weight Estimation of Meals

John-Henry Markbo
tfy12jm1@student.lth.se

August 31, 2019

Master's thesis work carried out at
the Department of Mathematics, Lund University.

Supervisors:

Mikael Nilsson, mikael.nilsson@maths.lth.se
Gabrielle Flood, gabrielle.flood@maths.lth.se
Claudia Lazarte, claudia.lazarte@food.lth.se

Examiner: Karl Åström, karl.astrom@math.lth.se

Abstract

In regions where illiteracy is widespread, it can be a challenge to keep protocol over the nutritional intake of subjects during long term dietary studies. People who take part in the studies can often not make records of their meals in a consistent manner, which can complicate data collection. One method used, which mitigates this problem, have the subjects of the studies take photographs of the meals. The images are then compared to already existing images of food, a photo atlas, where the weight is known, to yield a weight estimation.

In this thesis, the possibility to use such a photo atlas for machine learning algorithms to classify and segment the meal components was examined. To estimate weight of food from an image, a method was assessed where the amount of pixels of the segmented meal component was compared to a marker of known area. Mathematical transformations were used to correct for projective distortions affecting the marker area. To reduce color variations which can occur due to variations of lighting, methods of color correction were also examined as an aid in the classification.

The area of the ROC-curves (AROC) produced for each method were used to assess the classification methods. An example was a Random Forest algorithm used to classify pixels individually. To improve performance, the set of pixel features used to train the algorithm were chosen from pixels found to be especially hard to classify. In the thesis, the use of a Bag-of-Visual-Words approach produced the best results, where the found values for each meal component were above 0.90 in AROC. Using methods for color correction used yielded better results when implemented on images used in a convolutional neural network. An increase in AROC from 0.84, using original images, to 0.90, when using color corrected images was yielded.

Keywords: Random Forest, Bag-of-Visual-Words, U-net, X-rite, Color Correction, Perspective Correction

Acknowledgements

The author would especially like to thank Mikael Nilsson and Gabrielle Flood for their availability and support during the work of this thesis. Many thanks also to Yvonne Granfeldt, Claudia Lazarte and Karl Åström for the chance to have this project as my master thesis.

The author would also like to thank Elin Olofsson and Erik Norlander. I will miss our discussions in room MH:439, ranging from talking about the latest episode in *Game of Thrones* to discussing exciting new variants of neural networks.

Contents

Chapter 1
Introduction

In regions where illiteracy is widespread, it can be a challenge to keep protocol over the nutritional intake of subjects during long term dietary studies. People who take part in the studies can often not make records of their meals in a consistent manner, which can complicate data collection. This problem can today be partly overcome using photo atlases, containing photographs of meal components of different weights. The subjects of the studies take photographs of their meals and later have the size compared to corresponding meals in a photo atlas, in order to make an assessment of the nutritional intake over a period of time. This is referred to by Lazarte et al. as the **food photography 24-h recall method** [1].

Instead of manually making estimates based on images, the meal components can be classified automatically, and the size of each component can be used for estimation of weight, by extracting the necessary information from the images. From this information the nutritional value can be taken out using existing tables conveying nutritional data. This could have an advantage over manual assessment in both speed and accuracy, and for the alleviation of workload of the personnel conducting the studies and of the subjects of the dietary evaluation. A method for automatic assessment of food should be implemented in a way that minimises responsibility of the subjects taking part in the study, i.e. such a method should be developed as to not burden the subjects more than in the existing 24-h recall method. The reason for this is mostly in order to motivate participation in a study and increase the compliance with data collection.

The problem of classifying food from images using machine learning is inherently a difficult one to solve. Machine learning algorithms (discussed in section 3.3) have, over the years, found to be helpful in recognising various objects from images. Though some objects, or images thereof, are easier to classify than others. Take for instance airplanes. They may differ in size and appearance, but will have basically the same set of features, i.e. wings, oblong shape etc. The outcome can be quite different concerning images of food. There may be such a variance within the same class of a meal. This variance is hard for algorithms to capture; even with the same ingredients, an image of a salad may differ much in appearance from another. Still, much research is being made based on the recognition of food and the quantity present in an image.

1.1 Earlier work

The thesis will be based on the ideas presented by Lazarte et al. [1], where participants in dietary studies estimate the amount of meals by manually assessing photographs taken of them. A method that mimics the human perception of food quantities is therefore to be taken forward. The mobile app 'Foodvisor' has gained popularity over Europe the last year as a food classifier based on convolutional neural networks [2]. These networks usually require vast amounts of data to work adequately, and for that end uses the images taken by its users to improve the network. The automatic assessment of weight does also seem to be prone to error, and requires correction and input by the user to work as intended [2].

Bossard et al. [3] concluded results of classification of food images, creating the food image set "Food-101", an image set of 101 food categories, with $101,000$ images. They used a method of partitioning the images in that image set by using the SLIC algorithm, further explained in section 3.1.4, to label the images of food as a whole, without segmentation of the meals. Similarly, a deep learning approach, see section 3.3.6, was used by Mezgec Seljak [4], with more than an 80% accuracy for all methods tried, but without semantic segmentation. With the rise of deep learning algorithms in the beginning of the decade, the accuracy of classification of images have skyrocketed. In recent years, segmentation algorithms have emerged with great accuracy, where fully connected convolutional networks, which are explained further in section 3.3.6, are considered state-of-the-art methods for that end. Aguilar et al. proposed using a fully connected neural network for semantic segmentation, finding bounding boxes of the meal components [5], and with keen results. The application of using bounding boxes is not desirable in the scope of this thesis, as the boundaries of meal components are needed for quantity assessment.

Assessment of weights of food objects from images have been approached in different ways. One approach have been using stereo images with a reference object present in the images, such as Dehais et al. [6], with an average error of less than 10%. Zhu et al. were also able to achieve an error of less than 6% by fitting segmented food to geometrical models, cylindrical or prism-like, using single view images of food objects [7]. Liang et li [8] used a yuan coin, whose diameter is known, as a reference for size in every image, and having the food objects assigned one of three standard shapes: columnar, ellipsoid or irregular. This was in order to map the information from the 2D images to 3D information.

1.2 Outline and Delimitation of the Thesis

1.2.1 Image Acquisition

To lessen the workload of image acquisition on the authors part, only five different food classes were used. A first assessment of segmentation methods was made on meal components that were separated from each other. Had the results been more successful, and time had permitted, the methods would have been continued on images where meal components would have been adjacent to each other.

1.2.2 Segmentation

The first and most crucial part of the thesis project is the semantic segmentation, finding methods that not only classifies the meal components present in an image, but also provides a spatial labelling of them.

Classification of images has in recent years seen great results when done by convolutional neural networks (CNN), the theory of which is presented in section 3.3.6. This method requires large amount of images to amount to adequate results. In the scope of this thesis it does not seem reasonable to achieve such a large set of images. Instead, methods which do not require quite as much information will be used to perform classification, one of which is a CNN of a novel architecture, see section 3.3.6. The author expects that the standardised background used when photographing food should mitigate errors in classification, that are otherwise prone to occur due to variations in the training data images. The composition of the images is explained in section 2.

1.2.3 Weight Estimation

Quantity estimation of the food will be made using different approaches. Firstly, simply mimicking the approach of human visual assessment, comparing sizes in images to the size of meal components of a known weight. By doing this, the size reference used must be corrected for perspective errors, the theory of which is explained in section 3.2.2, as a marker in an image will have distorted shape depending on the position of the camera.

1.3 Aim of the Thesis

The aim of this thesis was to perform classification of meal components present on a plate and also provide an estimate for weight, given a pair of images of the same meal components at different viewpoints. For this end, a suitable machine learning algorithm that performs semantic segmentation was assessed. The segmented regions were then be used to aid in the final weight assessment. The highest goal set was to find an assessment method that can yield a nutritional estimate that does not differ more than 5% from the actual nutritional value.

Chapter 2
Photo Atlas

A photo atlas can be seen simply as a catalogue of images of different meal components, of varying weights. The photo atlas is used to assess the weight of meal components in other photographs by comparison. Five meal components were photographed and added to the photo atlas used in this thesis: rice, potatoes, sausage, broccoli and haricots verts.

All images of meal components were taken in pairs on the same white plate, placed on a tartan patterned table cloth. For the images used in this thesis, a marker of known size was added, a variant of a checkerboard.

Figure 2.1: Examples of all the different meal components of different weights in the atlas, with a black and white marker. All images were photographed in top view.

Each of the five meal components in the atlas has been photographed for equal range in weight, 50g-250g in steps of 25g, taken at two different angles, top view and (approx.) 60 degrees from the plate. The photo atlas consists of 90 images in total. The images were taken so that the table cloth covers the image space. The camera used was an iPhone 5s camera, taking images at 2448×3264 pixels. The focal length of the camera was kept the same between images from different views of the same object. Examples of the photographs that the photo atlas in this thesis contains, taken from top view, can be seen in figure 2.1. Each image contains

a marker of known size, either of the appearance as shown in figure 2.2a or as in figure 2.2b. Some extra images of hot dogs on a plate were also made, but were only used when assessing color correction methods, as discussed in section 3.1.2.

2.1 The Markers

The markers used in the thesis are variants of checkerboards, where the tiles are of known size. Checkerboard markers are commonly used in images as a mean of camera calibration, which is explained in section 3.2.1, as well as a size reference. Two markers were used: one in black and white, and one where a color chart has been integrated with the tiles. A common feature between the markers that were used is that the utmost dark tiles have rounded edges (against white background) such that the corners would not blend in with the background and cause difficulties when assessing the position of the corners. The colored marker adds reference points for known instances of colors, which were to be used for color correction, which is discussed in section 3.1.2.

2.1.1 Black & White Marker

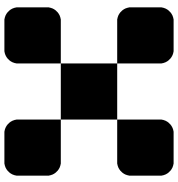
One of the markers is a 3×3 checker board, of which 5 tiles are dark and 4 are white. Each tile is a square with a side of 16 mm. This marker can be seen in figure 2.2a.

2.1.2 X-rite Colorchecker Marker

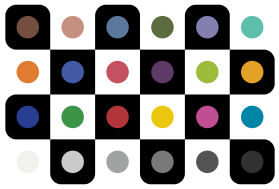
A marker of 24 square tiles of sidelength 12 mm, arranged in 4×6 grid, equally divided into dark and white tiles. Each tile contains a colored disk, all in all 19 unique colors, as one row contain disks of differing grayscale intensities. Each separate tile of the marker corresponded to colors as detailed in [9].

Having the color values in each pixel compared to the pixel values in the color marker, calibration of color can be made in the photograph. The color variation between images of the same object should then be reduced, which in turn should simplify classification of the images.

This marker can be seen in figure 2.2b.



(a) The black and white marker used for size reference in images.



(b) The checkerboard marker, where each tile contains a color comparable to a tile in a X-rite color checker. Used for size reference, as well as color correction in images.

Figure 2.2: Variants of markers used in the thesis.

Chapter 3

Theory

3.1 Image Analysis

For the automatic classification of images, one needs to first extract information from them in a consistent manner by using image analysis.

In this section, feature representation of colors and the correction of such are discussed, as well as a common feature representation of texture, using Speeded-up Robust Features. A method for partitioning an image in subsections of similar, neighboring pixels is also discussed at the end of this section.

3.1.1 Color Representation

To humans, colors are a visually striking feature of food and, as such, there is reason to believe that the information held within color features can be of use in machine learning algorithms. Color is not an inherent feature of objects though, but is all due to the perception of the beholder. A human perceives color in, the visible range of wavelength of light, 380 – 700 nm. The perception of color in the human eye is determined by three colorsensitive light receptors which each can perceive a certain range of wavelength of red, green and blue colors. The combination of the stimuli which light creates then yields the perception of color.

The most common representation of colors in modern devices are the various RGB color spaces [10]. The CIE LAB color space is a representation of color which more closely corresponds to how the human eye perceives color, as well as having a device independent representation in contrast to RGB, which is device dependent, and therefore the coordinates in the colorspace may shift between devices [10].

CIE RGB Color Space

Most modern devices are using pixels consisting of three different diodes, each emitting light at narrow band of wavelength corresponding to how humans perceive red, green and blue light. For the CIE RGB color space, taken forward by the **C**ommission **I**nternationale de l'**E**clairage in 1931, the LEDs emit light at mean wavelengths of about 700 nm, 566.1 nm and 435.8 nm, respectively [10].

Each of the three diodes that make up a pixel can be assigned a value between $0 - 255$. 0 being the diode turned off and 255 corresponding to the highest light intensity it can emit. It was shown in the nineteenth century that a combination of light from red, green and blue lamps can be become indistinguishable from another colored lamp [11]. In the same sense, this is carried out today on modern displays. Combining each diode with their own individual brightness value allows the perception of essentially any color.

CIE LAB Color Space

The LAB representation is a three dimensional color space which more closely relates to how humans perceive color, in the way that equal changes in the quantitative vectors values result in an equal change in perception of the color [12]. It is also device independent, and will therefore carry the same vector representation for any color between devices, i.e. monitors or the like.

The color space is represented by a color plane, comprised of two (a, b) vectors, each of the two values ranging between $(-127, 128)$. The L-coordinate shows the brightness of the color in a scale $0 - 100$, where 100 is the brightest [12].

3.1.2 Color Correction

The images used in the segmentation and classification of food objects are prone to be affected by different kinds of lighting, and hence distortion in color is bound to take place. To keep the color features invariant of light exposure or shading, the colors in the image may be mapped to a groundtruth colorchecker, the X-rite marker described in section 3.1.2. Finlayson et al. [13] describes a few methods for color correction in images using an X-rite color checker, some of which are described below.

Linear Mapping

Assuming a RGB-color space, a linear mapping is found through the equation system;

$$\mathbf{M}\rho = \mathbf{x}, \tag{3.1}$$

where $\mathbf{M}$ are the weights which maps the color vectors ρ to the ground truth colors $\mathbf{x}$, and ρ and $\mathbf{x}$ are vectors of size $3 \times N$, N being the number of corresponding color vectors between the marker colors, which means that $N = 24$ when using the X-rite colorchecker. The weights are represented by the 3×3 matrix $\mathbf{M}$ which minimises the least square error:

$$\min_{\mathbf{M}} ||\mathbf{M}\rho - \mathbf{x}||. \tag{3.2}$$

The least square solution is found using the Moore-Penrose pseudoinverse for solving the equation system:

$$\mathbf{M} = \mathbf{x}\rho^T(\rho\rho^T)^{-1}. \tag{3.3}$$

The found mapping can then be applied to all color vectors in the image to perform the linear color correction.

Polynomial Mapping

Below are vectors whose elements are polynomials whose factors are coordinates in the color space, up to the fourth degree. It can be shown that any polynomial with factors made up from each vector channel are linearly independent from any other such polynomial [13]. Adding elements of polynomials of higher degrees to the vector will therefore add further information and improve the color correction by capturing nonlinear behavior found in the image.

$$\mathrm{P}^{(1)} = (R, G, B)$$

$$\mathrm{P}^{(2)} = (R, G, B, RG, RB, BG, R^2, G^2, B^2)$$

$$\begin{aligned}\mathrm{P}^{(3)} = (&R, G, B, RG, RB, BG, RGB, R^2G, R^2B,\\ &G^2R, G^2B, B^2G, B^2R, R^3, G^3, B^3)\end{aligned}$$

$$\begin{aligned}\mathrm{P}^{(4)} = (&R, G, B, RG, RB, BG, RGB, R^2G, R^2B, G^2R, G^2B,\\ &B^2G, B^2R, R^3, G^3, B^3, R^2G^2, R^2B^2, G^2B^2, R^3G, R^3B,\\ &G^3R, G^3B, B^3G, R^2GB, B^2RG, G^2RB, R^4, G^4, B^4)\end{aligned}$$

here the polynomial vectors can be seen up to the fourth degree, where R, G and B are the intensity values corresponding to each device pixel LED, as described in section 3.1.1. The polynomial mapping then can be found by starting from the expression

$$\mathbf{M}\mathbf{P}^{(k)} = \mathbf{X}, \tag{3.4}$$

where $\mathbf{P}^{(k)}$ is a polynomial vector of the k:th degree. $\mathbf{M}$ is a matrix containing the weights assigned each of the polynomial vector elements, which maps $\mathbf{P}^{(k)}$ to $\mathbf{X}$, the new color space. Expression (3.4) can therefore be written as:

$$\begin{bmatrix}\mathbf{w}_R^T\\ \mathbf{w}_G^T\\ \mathbf{w}_B^T\end{bmatrix}\mathbf{P}^{(k)} = \begin{bmatrix}R'\\ G'\\ B'\end{bmatrix}. \tag{3.5}$$

Here $\mathbf{w}_R, \mathbf{w}_G, \mathbf{w}_B$ are vectors mapping a polynomial vector to an element of a coordinate in the ground truth color space. The prime signs on the coordinate on the RHS denotes the new

color space to which the image is mapped. The below expression can be used to calculate the weights.

$$\begin{bmatrix} (\mathbf{P}_1^{(k)})^T & 0 & 0 \\ 0 & (\mathbf{P}_1^{(k)})^T & 0 \\ 0 & 0 & (\mathbf{P}_1^{(k)})^T \\ \vdots & \vdots & \vdots \\ (\mathbf{P}_n^{(k)})^T & 0 & 0 \\ 0 & (\mathbf{P}_n^{(k)})^T & 0 \\ 0 & 0 & (\mathbf{P}_n^{(k)})^T \end{bmatrix} \begin{bmatrix} \mathbf{w}_R \\ \mathbf{w}_G \\ \mathbf{w}_B \end{bmatrix} = \begin{bmatrix} R'_1 \\ G'_1 \\ B'_1 \\ \vdots \\ R'_n \\ G'_n \\ B'_n \end{bmatrix}.$$

The subscripts denotes the n corresponding coordinates in the different color spaces that are to be mapped to each other. The weights can, similarly to the linear mapping in the above section 3.1.2, be found using the pseudoinverse to solve the equation system. While polynomial color correction can be used to find better results than a linear mapping, the effects of noise are increased. Also, Finlayson et al. have stated that polynomial mapping using an X-rite colorchecker is not enough to perform sufficient color correction [13].

Root Polynomial Mapping

The mapping between color spaces using root polynomials is essentially the same as in section 3.1.2, but using vectors where the k:th root are taken on the monomial elements of the k:th degree, seen below.

The root polynomial vectors can be seen to hold fewer elements than the polynomial vectors of the same order, as seen in section 3.1.2. This gives an an advantage over polynomial mappings, in the sense that a mapping found using root polynomial vectors still can capture nonlinear color distribution, but requires less information to do so. [13].

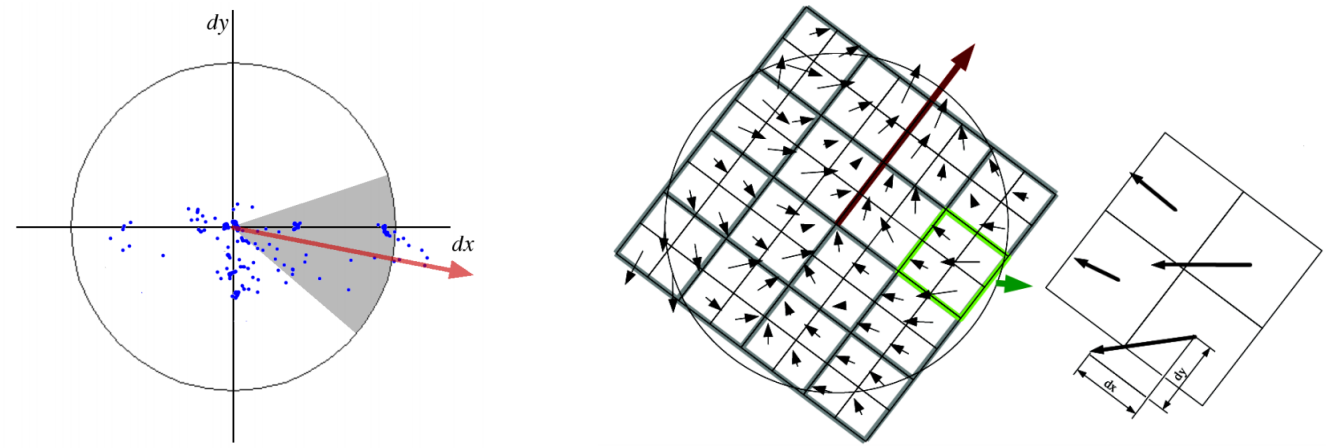
$$\begin{aligned}
\rho^{(2)} &= (R, G, B, \sqrt{RG}, \sqrt{RB}, \sqrt{BG}) \\
\rho^{(3)} &= (R, G, B, \sqrt{RG}, \sqrt{RB}, \sqrt{BG}, \sqrt[3]{RGB}, \sqrt[3]{R^2G}, \sqrt[3]{R^2B}, \sqrt[3]{G^2R}, \sqrt[3]{G^2B}, \sqrt[3]{B^2G}, \sqrt[3]{B^2R}) \\
\rho^{(4)} &= (R, G, B, \sqrt{RG}, \sqrt{RB}, \sqrt{BG}, \sqrt[3]{RGB}, \sqrt[3]{R^2G}, \sqrt[3]{R^2B}, \sqrt[3]{G^2R}, \sqrt[3]{G^2B}, \sqrt[3]{B^2G}, \sqrt[3]{B^2R}, \sqrt[4]{R^2G^2}, \\
&\quad \sqrt[4]{R^2B^2}, \sqrt[4]{G^2B^2}, \sqrt[4]{R^3G}, \sqrt[4]{R^3B}, \sqrt[4]{G^3R}, \sqrt[4]{G^3B}, \sqrt[4]{B^3G}, \sqrt[4]{R^2GB}, \sqrt[4]{B^2RG}, \sqrt[4]{G^2RB}).
\end{aligned}$$

Using the above vectors in expression (3.4), yields the weights for the root polynomial mapping.

3.1.3 Speeded-up Robust Features

Finding distinguishable features in an image is of the essence when using machine learning systems. The significance of color as a feature has been mentioned. Speeded up robust-features (SURF) are designed to be robust descriptors of image texture, using grayscale images as input. More in-depth details about SURF can be found in the original paper by Bay et al. [14]. The features extracted from an image point, a pixel, are based on the intensity found in the neighborhood found around that specific image point. Around the point of interest, a descriptor

point, a circular area is taken out. By sliding a circular segment of angle $\frac{\pi}{3}$, the orientation of the descriptor point is found by finding which placement of the circle segment that yields the highest value of Haar wavelet response. This is visualised in figure 3.1a, [14]. This yields the dominant direction, denoted by a red arrow in figure 3.1a and as a brown arrow in figure 3.1b. Around the interest point, a square area is placed and positioned in the direction of the found dominant orientation, as seen in figure figure 3.1b. The area is divided into a 4×4 grid, which in turn is also divided into a 4×4 grid. For each of these smaller gridded areas, the Haar wavelet response is made at 25 equally spaced sample points. The wavelet response in the vertical and horizontal direction, compared to the found point orientation, is called d_x and d_y, respectively. For each of the 16 subregions, the sum of each wavelet response is found resulting in a 4-dimensional feature vector for each subregion, $\mathbf{v} = (\sum d_x, \sum |d_x|, \sum d_y, \sum |d_y|)$, [14]. The full descriptor feature vector is therefore 64-dimensional for each chosen point of interest in an image. These features are invariant of scale and rotation of the image. Normalising the feature vectors further make them invariant to contrast as well.



(a) Visualisation of the orientation found for each point of interest. The blue points correspond to Haar wavelet responses, and the gray area the circle segment where the wavelet responses are the most prevalent.

(b) Visualisation of SURF features extracted. The vectors d_x and d_y are the horizontal and vertical components, respectively, of the Haar wavelet response for points found in the subregions of the grid.

Figure 3.1: Figures visualising the extraction of a SURF vector from an interest point. Using the dominant orientation found in figure 3.1a, denoted by a red arrow, a (SURF) vector representation of the descriptor point can be found, as seen in figure 3.1b. [14]

3.1.4 Simple Linear Iterative Clustering

The simple linear iterative clustering (SLIC) algorithm, as described Achanta et al. [15], partitions an image into a specified amount of regions in which the pixels are similar to each other. These regions will be referred to as 'superpixels'. For specific objects in an image, superpixels

usually conforms to the boundaries of discernible regions, making it a helpful tool in segmentation algorithms. The procedure is reminiscent of k-means clustering, but works considerably faster as the algorithm only ever acts on a small area of the total data. A point grid is initialised by setting the variable k, which is to be the number of desired superpixels, which are roughly the same in area. The distance between the points are then set to be $S = \sqrt{\frac{N}{K}}$, where N is the number of pixels in the image. At each point in the grid, a cluster center, C_k, is initialised, around which a superpixel is meant to be placed. It is assumed that a pixel belonging to a certain cluster center is within an area of size $2S \times 2S$ around the center. To find the pixels that will adhere to region boundaries present in an image, a balance should be found between color values of the pixels and their distance from the cluster center. The 3×3 pixel neighborhood around the cluster center is searched to find the point of lowest gradient values of the image, and the center of the clusters are moved there from the original grid points, to move them away from image edges and away from pixels whose data have been corrupted. The gradient calculated uses intensity and color plane of the CIELAB representation when calculating the gradient $\mathbf{G}(x, y)$ of an image point, in the below manner [15]:

$$\mathbf{G}(x, y) = \| \mathbf{I}(x + 1, y) - \mathbf{I}(x - 1, y) \|^2 + \| \mathbf{I}(x, y + 1) - \mathbf{I}(x, y - 1) \|^2, \tag{3.6}$$

where $\mathbf{I}(x, y)$ is the LAB vector values $[l, a, b]$, as explained in section 3.1.1, for a pixel at position (x, y) in an image.

A pixel is labeled to belong to whatever cluster center is closest to it and $\| \cdot \|$ corresponds to the L_2-norm. The process is iterated and cluster centers are assigned new positions based on the neighbouring gradient vectors until convergence is found. To fix for any segments which are not connected to their specified cluster, such segments are made to belong to the largest cluster of pixels. An overview of the SLIC algorithm is described as algorithm 1.

Algorithm 1: Simple Linear Iterative Clustering, as described by Achanta et al. [15]

Initialize K cluster centers $C_j = [l_j \; a_j \; b_j \; x_j \; y_j]^T$, $1 \leq j \leq K$
by sampling pixels at regular grid steps $S = \sqrt{\frac{N}{K}}$,
where N is the number of pixels in the image.
Move cluster centers to the lowest gradient position in a 3×3 neighborhood.
Set label $l(i) = -1$ for each pixel i.
Set distance $d(i) = \infty$ for each pixel i.
repeat
  for *Each cluster center* C_k **do**
    for *Each pixel* i *in a* $2S \times 2S$ *region around* C_k **do**
      Compute the distance D between i and C_k.
    end
    if $D < d(i)$ **then**
      set $d(i) = D$
      set $l(i) = k$
    end
    Compute new cluster centers
    Compute residual error E
  end
until $E \leq$ *threshold*;

3.2 Computer Vision

Assessing information from images in a way that mimics human understanding of visual impressions is a core concept in the field of computer vision. In this section, a common mathematical model of a camera, the pinhole camera model, is explained, as well as how this model can be used to form a two-dimensional projection from three dimensional space. Using the same camera, in the mathematical sense, means that homographies between images of the same object can be found. These homographies map the projections to each other.

3.2.1 Pinhole Camera Model

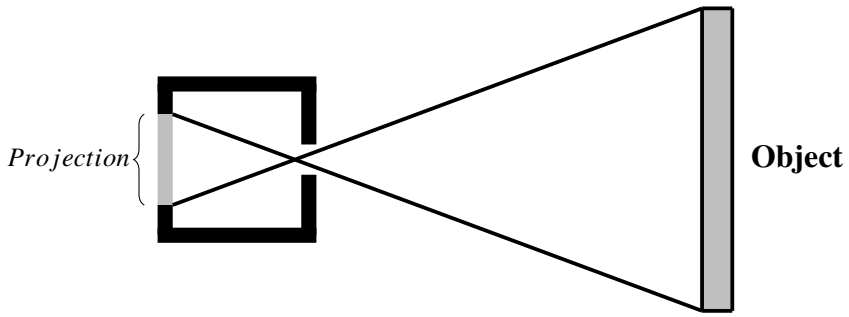


Figure 3.2: Shematic image of pinhole camera.

The pinhole camera is a rudimentary camera. It has no lens and gets its name from the small opening in which light enters. In computer vision, modern cameras are often modelled as an ideal pinhole camera [16]. The pinhole camera can simply be seen as a box, and when light from an object enters the pinhole entrance, an upside down projection is formed at the back wall of the box, see figure 3.2.

Assume a coordinate system, $\mathbf{x}_c = [x_c, y_c, z_c]$, in which the camera shutter, the pinhole, is defined at the point of origin, denoted by F_c in figure 3.3. Further assume that the plane in which projections are formed, normal to the viewing direction (denoted by z_c) of the camera, in a point $z_c = z'$. This is the image plane, spanned by the image coordinate vectors u and v, depicted in grey in figure 3.3, whose middle point is called the principal point. Here is also an example of a 3D-point, in the world coordinate system $\mathbf{X} = [X, Y, Z]$, to be projected on the image plane, $\mathbf{P} = [X_p, Y_p, Z_p]$, depicted.

For the same camera but moved another point in space, a new camera coordinate system, $\mathbf{x}' = [x', y', z']$ is formed by a transformation by rotation and translation. A translation vector and rotation matrix will map the different coordinate systems, $\mathbf{x}_c$ and $\mathbf{x}'$, to each other in such a way that

$$\mathbf{x}' = \mathbf{R}\mathbf{x}_c + \mathbf{t}. \tag{3.7}$$

A translation vector $\mathbf{t}$, of size 3 (given the 3-dimensional space), is a vector whose elements are added to a coordinate, effectively shifting the position of an object at a specific coordinate, i.e. performs a transformation $R^3 \rightarrow R^3$.

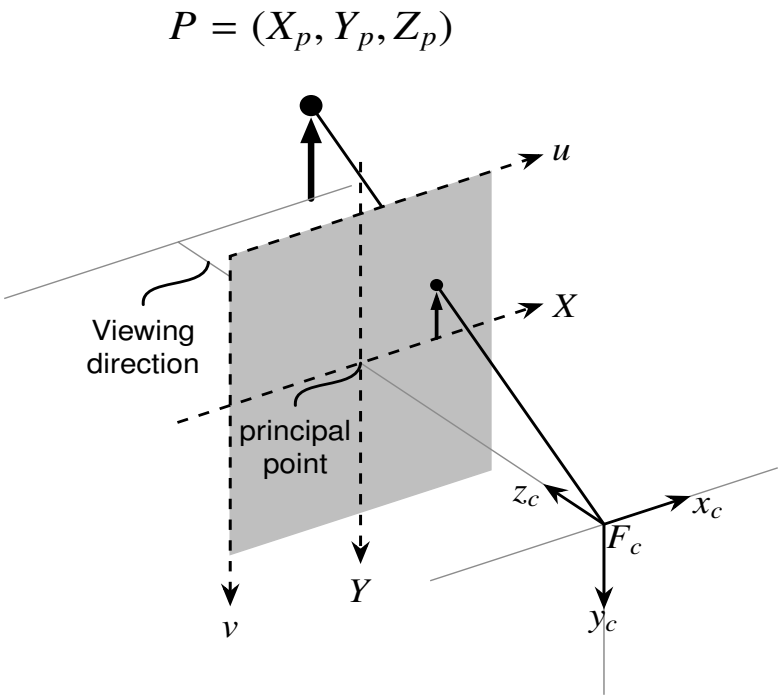


Figure 3.3: Example of coordinate system of a camera. The image plane is seen as the gray plane spanned by u and v, whose middle point is called the principal point.[17]

A rotation matrix also transforms a point to the same space. In three dimensional space a rotation matrix have the form of $\mathbf{R} = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ cos(\theta_x) & 0 & -sin(\theta_x) \\ sin(\theta_x) & 0 & cos(\theta_x) \end{bmatrix}, \mathbf{R}_y = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix}, \mathbf{R}_z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Here, θ_x, θ_y and θ_z are the rotation angles around the $x-, y-$ and $z-$axis, respectively. The elements of a translation vector, $\mathbf{t} = [x_t\ y_t\ z_t]^T$, are the values by which each coordinate vector value is translated in space.

Expression (3.7) can be further compressed by adding an element to the coordinate vector such that $\mathbf{x} = [x\ y\ z\ 1]^T$, for which the projection can be written as:

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}. \tag{3.8}$$

This will yield the mapping between the different coordinate systems belonging to each camera, and in effect, a mapping between the projections. The mapping from real world to

image matrices is found by using the inner parameters of the camera: skew, aspect ratio and focal length, as well as the principal point. The **K**-matrix, as seen below, maps, or corrects, the image plane to fit a photography:

$$\mathbf{K} = \begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

The skew factor s corrects the the skewness found in the projection, making distorted pixels rectangular and placed accordingly in an image grid. The aspect ratio, γ is used to rescale the axis of rectangular pixels so that the sides of the pixels become of square shape. The focal length is used to fit the projected points to pixels in an image, and the principal point is the coordinate of the image where the center point of the projection plane is mapped to, see figure 3.3.

Knowing the rotation and translation mapping the coordinate system of a camera to a reference system, as well as the inner parameters of the cameras used, each camera can be assigned its own specific camera matrix **A**:

$$\mathbf{A} = \mathbf{K}\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}. \tag{3.10}$$

If the inner parameters are unknown, they can be estimated using a photographed object of known size. Finding the inner parameters is known as camera calibration [18].

The camera matrix **A** is then a mapping between a point in three dimensional space **X** and its corresponding point in the image plane, **x**, up to a certain scale, denoted by the constant λ.

$$\lambda \mathbf{x} = \mathbf{A}\mathbf{X}. \tag{3.11}$$

Considering the projection of 3D-points all belonging to the same plane, a coordinate system can be chosen such that $z = 0$ for all points in space that are to be projected onto the image plane. This simplifies the calculations from expression (3.8) by replacing the rotation matrix **R** with $\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix}$, where each element is a column of the rotation matrix **R**, as described by Zhang [19].

$$\mathbf{K}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix}\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}. \tag{3.12}$$

Here, the matrix $\mathbf{H} = \mathbf{K}\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}$ is the mapping which relates a planar object to its projection. Matrices which map different projections to each other instead, and not the depicted 3D object. These mappings then performs a projective transformation of one image to the other.

3.2.2 Projective Transformation

Since a mapping between an object and its projection can be established, a mapping can also be calculated between different projections by having two (or more) images of the same object, made from the same camera. Having the same inner parameters, as discussed above, a mapping may be found which relates those images, or more specifically, relates the points $\mathbf{x}_k = [x_k, y_k, 1]$

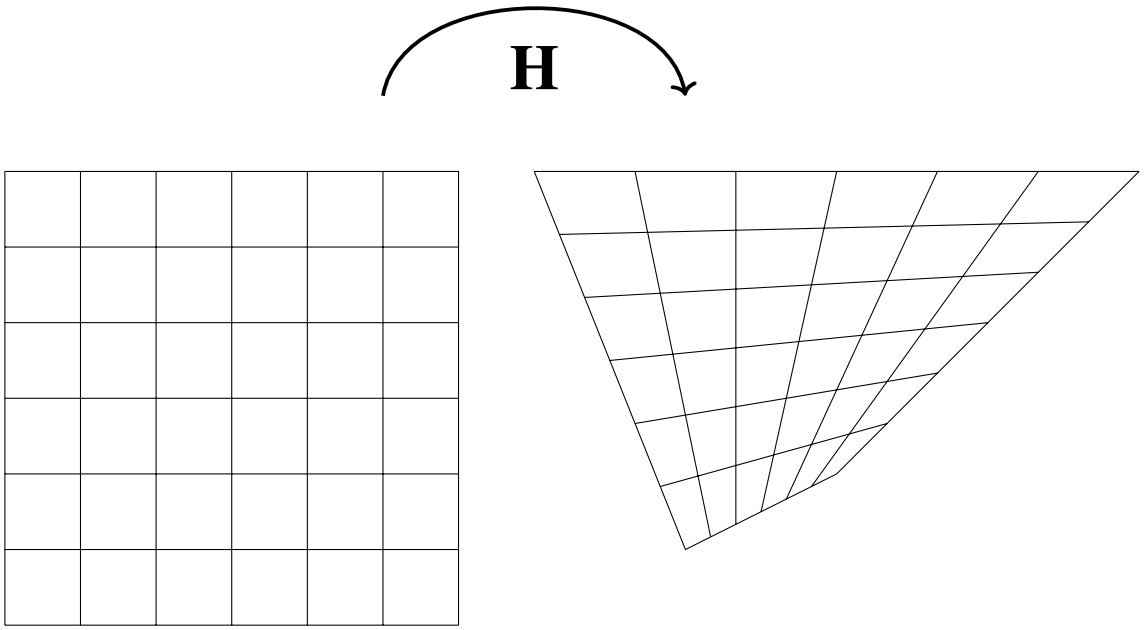


Figure 3.4: A planar object (left) subjected to a projective transformation, performed by the matrix **H** (right). [20]

in one image plane to the $\mathbf{y}_k = [x'_k, y'_k, 1]$ in another image plane. This is known as a homography mapping, a projective transformation of an image. The homography mapping is here assumed to be of a planar object, such as the object being mapped in figure 3.4.

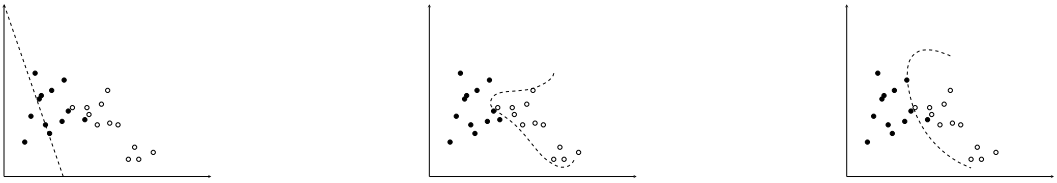
$$c_k \mathbf{x}_k = \mathbf{H}\mathbf{y}_k, \quad k = 1 \dots n, \tag{3.13}$$

where $\mathbf{H}$ is the 3×3 homography matrix, n is the number of corresponding points. The homography matrix is normalised in such a way that the $\mathbf{H}_{33} = 1$. As such, the homography matrix contains 8 unknown values to be determined, and additionally one unknown value for each point correspondence, c_k. For each k there are three equations, and therefore the least number of points correspondances should be $n = 4$, as $3n \geq 8 + n$. From these four (or more) corresponding points, a mapping $\mathbf{H}$ can be found, [21].

3.3 Machine Learning

Machine learning is a collective term for computational methods that can be used to perform tasks which traditionally have required human intelligence. All algorithms have in common that they recognise patterns in a given sequence of data. Common usages of machine learning methods are regression analysis, to predict an outcome given precedent events, as well as classification of data, which is the end for which machine learning is used in this thesis.

Commonly in machine learning, the algorithm is handed an amount of data which it tries to find a suitable pattern for. This is called training data, and the features of the training data should closely resemble the test data, for which the algorithm is meant to act on. The performance of a method is assessed by the correspondence in prediction of data and the label



(a) Example of an underfitting classifier.

(b) Example of an overfitting classifier.

(c) Example of an classifier of with reasonable accuracy.

Figure 3.5: Examples of results using classification algorithms which either underfit (**a**) or overfit (**b**) or finds a balance in the bias-variance trade off (**c**).

which denotes the ground truth correspondence of that data. Commonly used algorithms are the **support vector machine algorithm**, see section 3.3.2, as well as **decision tree algorithm**, discussed in section 3.3.3. Most notable has been the advent of **neural networks** this past decades, see section 3.3.6, and in the form of convolutional neural networks for machine learning applied on images, see section 3.3.6. These networks usually require large image sets to be used as training, the stage where the algorithms seeds out the patterns in the data. For some methods, the discriminant features need not be denoted by labels, but instead the algorithm itself finds patterns to distinguish the data into categories. In many applications though, the data used in the pattern recognition algorithms must first be chosen (wisely). This is the difference between supervised and unsupervised learning. Many algorithms face the problem of overfitting to training data, performing so well on the training data that the learned pattern performs poorly on any other data, and underfitting, where the algorithm finds no pattern at all. Overfitting is associated with showing large variance in the classification, performing too well on training data, and poorly on test data. Biased fitting performs poorly on both training and test data, as it finds no pattern in the data set. An example of this is shown in figure 3.5, where a machine learning method is to form a decision boundary in a two-dimensional feature space.

3.3.1 Accuracy Measurements

Confusion Matrix

A common way of estimating the accuracy of a classification algorithm is to use a confusion matrix, which gives information not only on accurately classified data, but also provides information on the ratio of data which is misclassified as belonging to another group. When binary classification is used, the labels of the two classes are often termed as positive or negative, for example: '*Food is present in this image*' would be a positive outcome for a certain image, and the opposite, '*Food is not present in this image*', would be deemed a negative outcome. In table 3.1 below, the accuracy of classification of class 1 and class 2 can be visualised. The number TP and TN both represent the amount of data that has been classified correctly, and stand for **T**rue **P**ositive values and **T**rue **N**egative values, while FP and FN represent the amount of data that are misclassifed as belonging to the other class, standing for **F**alse **P**ositive values and **F**alse **N**egative values. $N = TP + TN + FP + FN$ is the total number of data points classified.

		Actual class		
		Class 1	Class 2	Total
Predicted class	Class 1	TP	FP	$TP + FP$
	Class 2	FN	TN	$FP + TN$
	Total	$TP + FN$	$FP + TN$	N

Table 3.1: Example of confusion matrix concerning two classes. TP (true positive) represents the amount of positive incidents which were classified as such. In the same manner TN (true negatives) represents the quantity of correctly classified negative incidents. FP and FN corresponds to the quantity of negative and positive outcomes, respectively, falsely classified to belong to the other class. [22]

Classifying data into n classes yields a confusion matrix $\mathbf{C}$ of size $n \times n$. Each entry in the diagonal elements of the matrix then corresponds to the amount of data which were correctly classified to the individual class. An off diagonal entry in element $\mathbf{C}_{ij}$, $i, j \leq n$, $i \neq j$, then correspond to the amount of data which were classified to belong to class i, but actually belong to class j, [23].

ROC-Curves

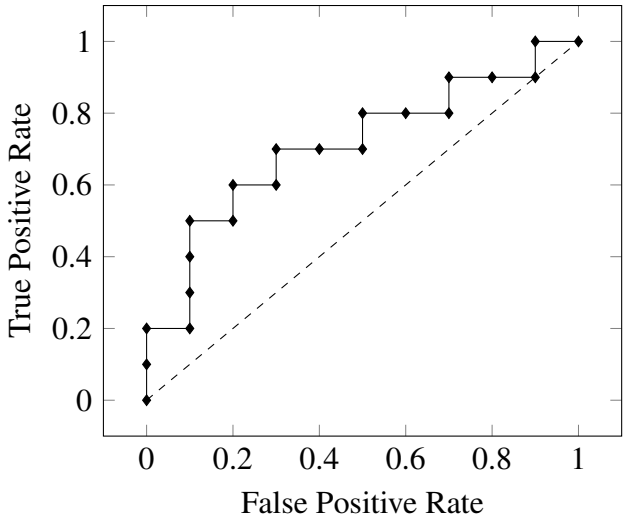


Figure 3.6: Example of ROC-curve. Points on the ROC-curve which coincide with the dashed linear curve mean that that particular classification have an equal chance of belonging to each of the two classes, and is therefore undesirable. [24]

Receiver operating characteristics (ROC) are a way to visualise the outcome of a classification algorithm. Using the concepts explained above for the binary classification matrix, the true

positive rate (TPR), $TPR = \frac{TP}{TP+FN}$ are plotted against the false positive rate (FPR), $FPR = \frac{FP}{FP+TN}$. In that sense, a single (binary) confusion matrix can be said represent a point on the ROC-curve, corresponding to a certain parameters used in classification.

For instance, one may move the threshold in probability for which a classification algorithm should exceed, to classify data as a positive outcome. A low threshold meaning that data is classified to belong to a certain class even though the calculated probability of it doing so is very low. The ROC-curve can then be used for assessment of the classification algorithm, finding the threshold which yields a high TPR while also keeping the FPR low. This is usually found by maximising the area under the ROC-curve, although sometimes it is not applicable, for instance when a high FPR is acceptable, as long as a high TPR is also obtained.

The area under the ROC-curve (AROC) yields a quantitative measurement of the performance of the algorithm used. The worst value of AROC conceivable is 0.5, corresponding to a curve close to the dashed line seen in figure 3.6, meaning that algorithm always perform guesswork, while a value of 1.0 means it performs classification with complete accuracy.

3.3.2 Support Vector Machine

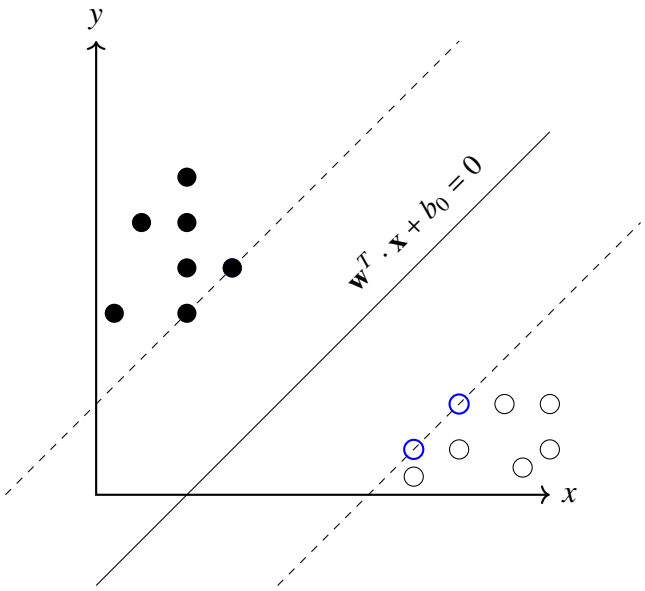


Figure 3.7: Example of a support vector machine in a two-dimensional feature space. The support vectors, annotated with blue color, are found at the edge of the decision margin, annotated by dashed lines. [25]

When differentiating between data in an n-dimensional space, one would like to find a function $f : R^n \rightarrow [-1, 1]$, a positive or negative outcome so to speak, as described by [26]. Support vector machine (SVM) is an algorithm used to find a decision boundary, a hyperplane separating data points into either of two classes in an n-dimensional space [27]. The support vectors are the data points that lie closest to the decision boundary, depicted in blue in figure 3.7 and as such are the hardest to classify. The goal of training a support vector machine is not only to find a hyperplane that separates the classes, but finding the optimal decision boundary which

yields the greatest margin between the support vectors, and in effect, the classes as a whole. That margin is the distance between the support vectors, which are used in the training to find the hyperplane defined below:

$$\mathbf{w}^T\mathbf{x} + \mathbf{b}_0 = 0, \tag{3.14}$$

where $\mathbf{w}$ is a weight vector for the input vector $\mathbf{x}$ and $\mathbf{b}_0$ is the bias. For a binary classification problem, the data subsets said to belong to either of the classes, $\mathbf{c}_1$ and $\mathbf{c}_2$, should be separated in a way that expression (3.14) becomes positive for $\mathbf{x} \in \mathbf{c}_1$, and negative for $\mathbf{x} \in \mathbf{c}_2$.

The SVM described above can only distinguish between linearly separable classes. If a data space is found not to be linearly separable, one can perform a fitting transformation of the data space in order to achieve a new linearly separable data set, or find the hyperplane that minimise the amount of data that is misclassified. More in-depth information about SVMs can be found in literature such as Friedmans et al. work on the subject [28].

3.3.3 Decision Trees

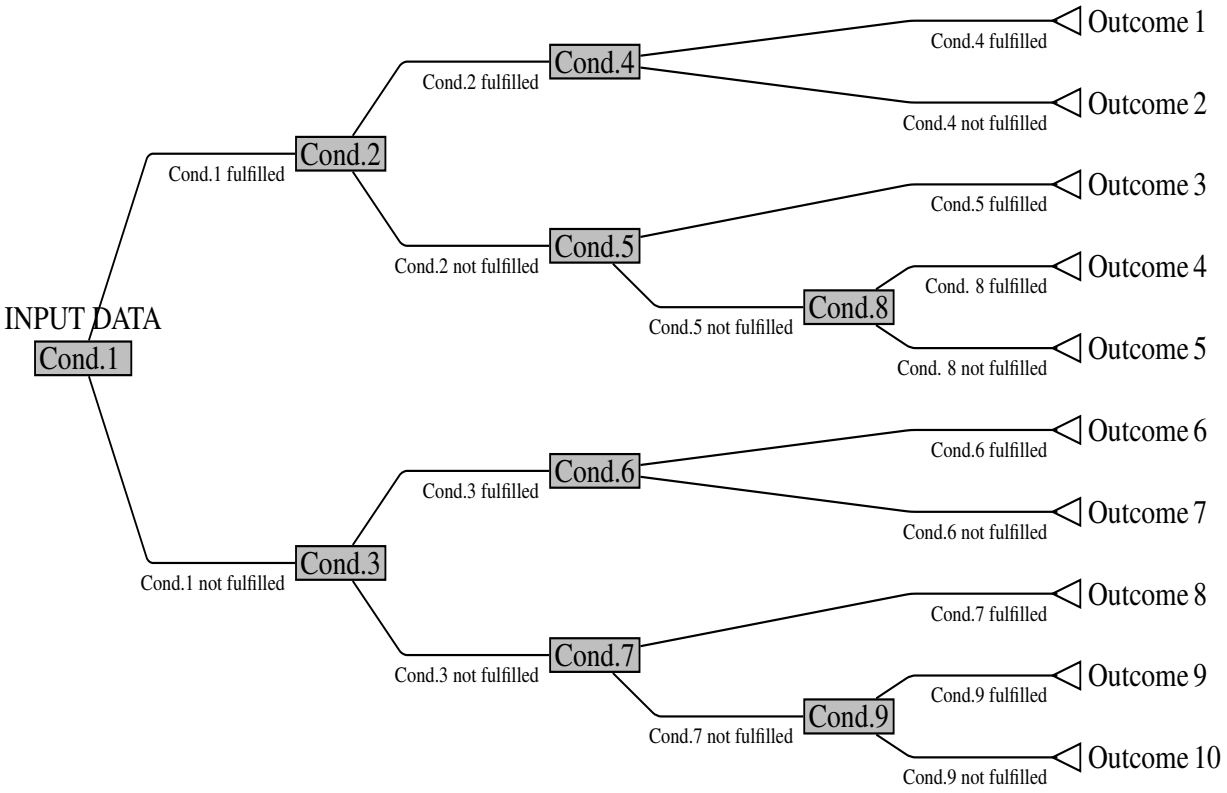


Figure 3.8: Schematic image of an example of a decision tree. Decision nodes are denoted by gray, and contain a condition which decides the next condition by which the data is subjected to. Given a certain input data, the information from it traverses through the branches until a final leaf is found, making the prediction of the outcome.[29]

The decision tree is a series of branching decision nodes, for which an example is schematically depicted in figure 3.8. Starting from the 'root' node of the tree, a decision is made based on the information provided from the feature set of given data. At each node, the features are evaluated and if a criteria is met, a certain branch from the node is chosen. This branch leads to another decision node, where another branch is chosen depending on that nodes criteria and so on. Each node divides the data by a hyperplane in the data space, as explained by Rokach et al. [30], which can result in a more complex decision boundary than can be achieved by an SVM. This continues until the last node, a 'leaf', is reached and the data is classified according to what class that particular leaf is associated with. Decision trees are inherently prone to overfitting, and there are several ways to mitigate this problem. One method is the so called 'pruning' method, removing decision nodes to some extent. A large tree, i.e. a tree containing many different decision nodes, is usually overfitted. Pruning reduces the number of nodes and decision branches, and the nodes removed are chosen in a way so that the ability to capture the general behavior of the data is not lost, as suggested by Breiman et al., cited by Rokach et al. [30].

Another method to avoid overfitting in a decision tree is to make a conclusion based on the results of several trees, fittingly called a forest. Each decision tree in the forest is 'grown' using smaller samples that are taken out of the data. The smaller samples taken out to grow a tree are not removed from the original data set, so that every tree has the same data set to randomly extract samples from. The ensemble of decision trees then finally votes on the outcome, as Breiman puts it [31], which mitigates the overfitting nature of a single decision tree in that the patterns found by the most amount of trees is probably the best assessment for a classification algorithm.

3.3.4 K-means Clustering

K-means clustering is an unsupervised machine learning algorithm meant to partition data in an n-dimensional space into a chosen number of groups of similar data points. If the total data set is $\mathbf{x} = (x_1, x_2, ..., x_n)$, we want to divide these data points into k subsets, $k \leq n$, $\mathbf{S} = (\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_k)$. The sought after set of clusters $\mathbf{S}$ are found by minimising the sum of the Euclidean squared distances between data points and the centroid of each points assigned cluster, as described by Overgaard [32]:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathbf{S}_i} \|\mathbf{x} - \mu_i\|^2. \tag{3.15}$$

Here, μ_i is the centroid of $\mathbf{S}_i$. Centroids are initialised by extracting k samples from the existing data set, $\mathbf{x}$, randomly. A new centroid is assigned by computing the middle point of the found clusters. This procedure is iterated until the assigned centroids no longer move considerably. These centroids are then the set μ, whose corresponding cluster are found in $\mathbf{S}$, to which the data $\mathbf{x}$ is to be assigned to according to expression (3.15). An example of k-means clustering for $k = 2$ can be seen in figure 3.9, and a common algorithm for the k-means method, Lloyds algorithm, is presented as Algorithm 2.

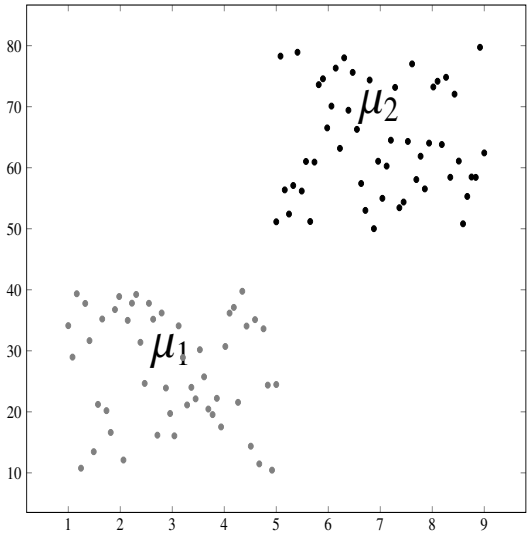


Figure 3.9: Example of k-means clustering between two classes. μ_1 and μ_2 denote the cluster centroids. [33]

Algorithm 2: Lloyds algorithm, as presented by N.C. Overgaard [32]

Initialize cluster centers $\mu_1, \mu_2, ..., \mu_k$, from random points from the data set.
repeat
  Assign class labels to data points $S_i = \{ \mathbf{x} = \mathbf{x}_i \,|\, \mu_i, \text{the closest centroid to } \mathbf{x}_i \}$
  Update centroids $\mu_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in \mathbf{S}_i} \mathbf{x}$
until *Convergence is found*;

3.3.5 Bag-of-Visual-Words Model

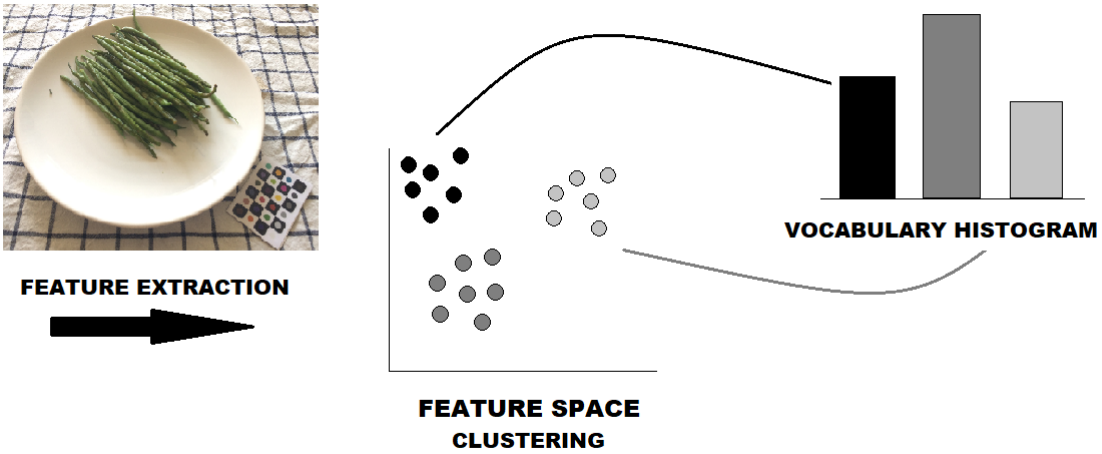


Figure 3.10: A simple schematic image of the procedure of the bag-of-visual-words method.

The bag-of-word model encodes the quantitative features of the data before classification. It gets its name from its use in language processing, where encoded feature vectors are extracted by counting the occurrence of words in a text, disregarding other information provided, such as grammar, placement of a specific word in relation to other words, etc. The analogy of a bag is then clear as we 'fill' it with the encoded features, the count of the words, found in an arbitrary text. That 'filled bag', a histogram, in vector representation can then be used as a feature vector for that particular text for use in machine learning algorithms.

The analogy can be transferred to images, making it a bag-of-visual-words. Using extracted feature vectors of length n from the training images, K-means clustering can be used in the n-dimensional feature space. The centroids found for each cluster will then be used for the encoding of the features. For each feature, the closest cluster centroid, in Euclidian distance, is found. A feature vector therefore adds a count to a stack in a histogram, where each stack corresponds to a certain centroid. A simple visualisation of the procedure can be seen in figure 3.10.

Using the above procedure, subsets of an image can be encoded into vectors which effectively describes the frequency of certain types of features inside that subset. Using the encoded vectors, the whole image can then be classified using common machine learning algorithms, such as SVM or random forest, as discussed by Coates et al. [34].

Spatial Pyramid Model

In order to capture spatial properties of an image, a so called spatial pyramid can be used. For a (rectangular) image of interest, we form a pyramid of p levels. Each level is the image, (with the original image being 'the ground floor', so to speak) divided into a grid of 2^p equally large regions. Each region is now treated as its own image, to be classified. The features found in the subregions are weighted according to which level in the pyramid it stems from. Features found from smaller regions are valued higher than features extracted from a coarser grid. [35]

3.3.6 Artificial Neural Networks

Artificial neural networks (ANN) are machine learning algorithms, of varying schematic appearance, which are modelled to resemble animal brain cells in how they interact in order to learn and store knowledge. In essence, a neural network is a nonlinear function which maps an n-dimensional data vector to a corresponding output, i.e. a class. Nodes in the network, denoted by circles in figures 3.11, 3.12 or 3.14., are structured in layers, and each node correspond to a value. The first layer, the input layer, is where the raw data that is to be mapped to the output nodes is entered. If there are layers in between the input and output nodes, these are called hidden layers. The connection between nodes between neighboring layers correspond to a weight, denoted in the figures in this section as ω, a value which the node is multiplied with before reaching the connecting node.

The nodes of the hidden layers receive this weighted data and use it as the argument to its specific activation function, $\phi_n^{(k)}$ (for the n:th node in the k:th layer), which is explained later in this section. The output of that function, $h_n^{(k)}$, is in turn weighted and used as input to the connected nodes in the next layers. This process is iterated until the data reaches the output layer. The weighted outputs from the last hidden layer becomes the argument in the output activation function, ϕ_o, which performs the final classification.

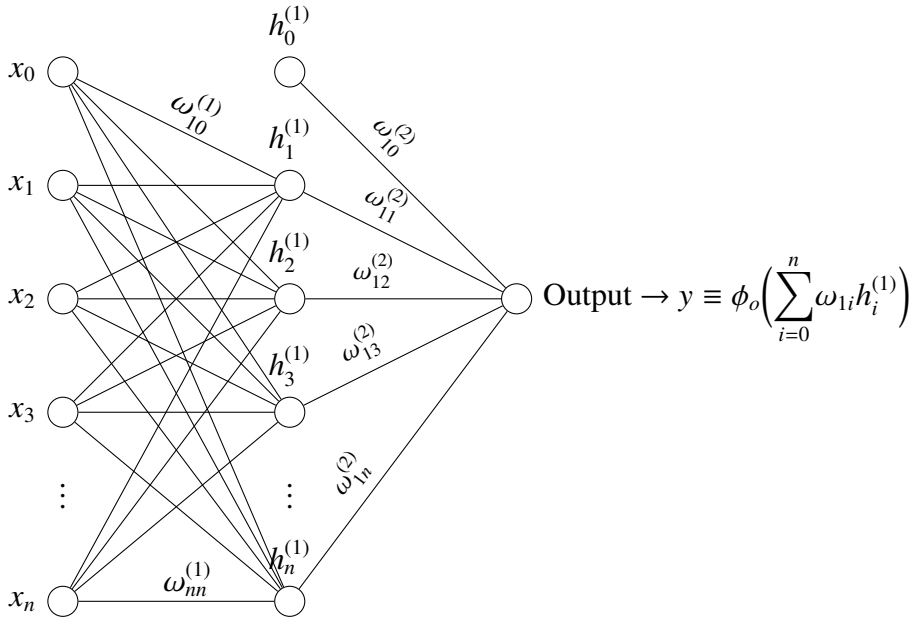


Figure 3.11: Example of an ANN, a multilayer perceptron (MLP). The columns of nodes from left to right are, in order, called the input layer, hidden layer and lastly the output layer. This network contains one hidden layer, but an ANN can contain several or none such layers. An ANN will always have at least one input node and one output node. Further details concerning the MLP and notations in the figure are explained in section 3.3.6. [36]

The input layer and the hidden layers all have a bias node, non-dependent on the input data used, which also have weights ascribed to them. The bias nodes are usually set to be unitary, and the contribution from the bias nodes are thus completely dependent on their assigned weights. When training an ANN, the weights are constantly updated until the network capture the sought after patterns in the input data set. This is done by comparing the results found in the output layer with ground truth data, and determining an error, or loss, using the difference of these using an algorithm called backpropagation, which will be explained later in this section. Much of what will be discussed in this section, and section 3.3.6, are based on notes from lectures given by Mattias Ohlsson [37].

The most simple form of an ANN is the one which only has one output node and no hidden layers, namely the perceptron.

Perceptron

The perceptron, see figure 3.12, maps the input data directly by the summation of all the weighted input and a bias term, $x_0 = 1$ in figure 3.12. This activation is used as input for the activation function $\phi_o(x)$. When classifying input data, the output y should correspond to the class label, which is binary in the case of the perceptron. A simple example of an activation

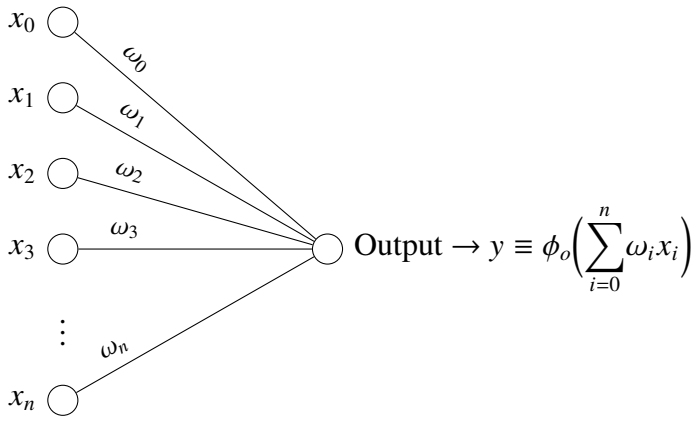


Figure 3.12: Example of an ANN, the simple perceptron. Bias node is set to $x_0 = 1$. The notations of a weight ω_n, mean that they weight input from the n:th input node to the output node. ϕ_o is the activation function of the output node. [36]

function would be the McCulloch-Pitts function:

$$\phi_o(a) = \begin{cases} -1, & a \leq 0 \\ 1, & a > 0 \end{cases}. \tag{3.16}$$

If the input, the activation, is positive, then the output of the activation function is positive, 1, denoting correspondence to one of the binary classes. Otherwise the result is -1, denoting correspondence to the other class. Using expression (3.16), the weights are used to form a decision plane. Using the perceptron in figure 3.12 with $n = 2$, the activation, a, becomes $a = \omega_0 + \omega_1 x_1 + \omega_2 x_2$. The decision hyperplane takes the form of a function $\omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$ in the (x_1, x_2)-plane as a line, and the labels of the data at either side of the line is assigned by the activation function according to expression (3.16).

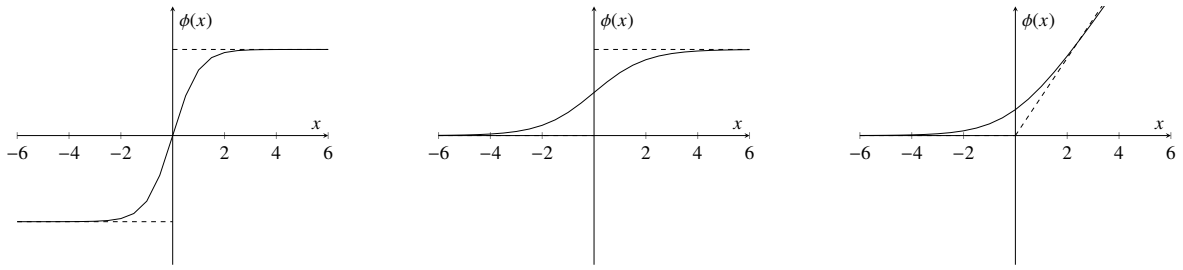
Activation Function

The McCulloch-Pitts function captures the idea of activation functions in general, to make a decision if data belongs to a certain output label. As will be discussed in section 3.3.6, a differentiable function should be used, and usually of sigmoid appearance, in order to mimic the behavior of the McCulloch-Pitts function. Such a function can be seen plotted in figure 3.13a, namely the hyperbolic function, $\phi(x) = tanh(x)$.

Similar to (3.16), the step function can also be used to find a decision boundary. A differential function which mimics this is the logistic function $\phi(a) = \frac{1}{1+e^{-a}}$, as seen in figure 3.13b. When using classification algorithms which differentiate between more than two classes, a generalisation of the logistic function can be used, namely the softmax function, which calculates a probability distribution over all the possible outcomes in the classification:

$$\phi(a) = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}. \tag{3.17}$$

Here, a_i is the activation in the i:th node of a certain layer of n nodes. The rectifier linear function (ReLU) is also a popular choice for neural networks, and is simply $\phi(a) = max(0, a)$.



(a) The mcCulloch-Pitts function (dashed) and the hyperbolic function (continous).

(b) The step function (dashed) and the logistic function (continous).

(c) The rectifier linear function (dashed) and the softplus function (continous).

Figure 3.13: Examples of commonly used activation functions (continuous) and the non-differential functions (dashed) related to them.

A differentiable function which corresponds to the ReLU function is the so called softplus function, $\phi(a) = log(1 + e^a)$, and is depicted in figure 3.13c. The derivative of the softplus function is the logistic function, as seen in figure 3.13b.

All nodes in an ANN, except for the input layer, holds an activation function. To achieve non-linear mapping of a data, non-linear activation functions are needed. Using linear functions can only ever yield linear transformations of the data to the output, for which a multilayered network would not be needed in the first place, only a perceptron. For multilayered networks, the logistic function and the hyperbolic tangent function are often used, but the linear rectifier function is the most used for convolutional ANNs, which are described in section 3.3.6.

As previously stated, the activation function yields an output based on the weighted input from the nodes of previous layers. The performance of an ANN is therefore largely dependent on the weights assigned to each node connection. The output of the activation function is used to achieve better performance by updating the weights, by using backpropagation.

Backpropagation

Backpropagation is the algorithm from which a neural network gains knowledge. The weights of the nodes in a neural network are usually initialised randomly within a certain range of given values. Backpropagation updates the weights, which is usually referred to as 'training' of the network. The idea is to find an estimate of how much the output classification of known data, as performed by the network, corresponds to the actual class the data belongs to. This estimate is called the loss function. From Bayes rule, the probability that vector $\mathbf{x}$, weighted by a set of weights $\mathbf{w}$ is classified to belong to one of two classes K_i, $i = \{1, 2\}$, is calculated by the below expression:

$$P(K_i|\mathbf{x}, \mathbf{w}) = \frac{P(K_i)P(\mathbf{x}, \mathbf{w}|K_i)}{P(\mathbf{x}, \mathbf{w})}, \quad \begin{cases} \sum_i P(K_i|\mathbf{x},\mathbf{w}) = 1 \\ \sum_i P(K_i) = 1. \end{cases} \tag{3.18}$$

Given binary classification, one may describe the two probability function $P(K_1|\mathbf{x})$ and

$P(K_2|\mathbf{x})$ by using a function $y(\mathbf{x})$:

$$\begin{cases} P(K_1|\mathbf{x}) = y(\mathbf{x},\mathbf{w}) \\ P(K_2|\mathbf{x}) = 1 - y(\mathbf{x},\mathbf{w}). \end{cases} \tag{3.19}$$

Using labels d_n, $n = \{0, 1\}$ to differentiate between the classes, the probabilty that $\mathbf{x}$ belong to either of the two classes can be expressed as below:

$$P(d_n|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^{d_n}(1 - y(\mathbf{x}, \mathbf{w}))^{(1-d_n)}. \tag{3.20}$$

The probability $P(d_n|\mathbf{x}_n, \mathbf{w})$, where $\mathbf{x}_n$ is known to belong to class n, should be maximised by changing the variable weight set $\mathbf{w}$ accordingly. Instead of maximising the above expression, the negative logarithm can be minimised, meaning that the optimal set of weights, $\mathbf{w}_{opt}$, is found in the following way:

$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \Big(- \sum_{n=1}^{N} (d_n \log(\, y(\mathbf{x}_n, \mathbf{w})) + (1 - d_n) \log(1 - y(\mathbf{x}_n, \mathbf{w}))) \Big), \tag{3.21}$$

This is known as the cross entropy error for binary classes. When there are more than two classes to choose between, another error function may be found, using the same reasoning as for finding expression (3.21):

$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \Big(- \sum_{n=1}^{N} \sum_{k=1}^{C} (d_{nk} log\, y_k(\mathbf{x}_n, \mathbf{w}) \Big), \quad d_{nk} = \begin{cases} 1, & \mathbf{x}_n \in K_k \\ 0, & \mathbf{x}_n \notin K_k \end{cases} \tag{3.22}$$

where the expression within the larger brackets is called the loss function (or error function). It is a measure of difference in prediction of the network and ground truth value. To find the optimal set, gradient descent learning is used, which seeks to find a minimum in the loss function.

Algorithm 3: Stochastic gradient descent with momentum, as described by Olsson [37]

Initialize weights $\mathbf{w}_{ij}^{(0)}$
set iteration index $k = 1$
set learning rate η appropriately
set momentum term α appropriately
repeat
  for *Each data sample* $\boldsymbol{x}_k$ **do**
    $w_{ij}^{(k)} = w_{ij}^{(k-1)} - \eta \frac{\delta E(\boldsymbol{w})}{w_{ij}^{(k-1)}} + \alpha \Delta w_{ij}^{(k-1)}$
    $k = k + 1$
    $D = |w_{ij}^{(k)} - w_{ij}^{(k-1)}|$
  end
until *D does not change considerably*;

The loss function is the essential tool in supervised learning as it yields a quantitative measure of error dependent on the weights in the network. From the activation in the n:th node of

the output layer and the activation function σ, the cross-entropy error function is then calculated using the ground truth labels d_n:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}(d_n log(\mathbf{x}_n) + (1 - d_n)log(1 - y(\mathbf{x}_n))), \tag{3.23}$$

For the network to improve, the error function should be assessed in order to update the assigned weights for the network to perform better. The weights should be chosen in a way that puts the error closer to a minimum, which can be achieved by updating the weights in the negative gradient direction such that $w_k = w_{k-1} - \eta\frac{\delta E(\mathbf{w})}{w_k}$, where η is the learning rate, i.e. the step size. The learning rate is a parameter to be set to individaul situations. Choosing η to large might result in that the minimum of the function is stepped over, while choosing η to be small means that it might never reach the minimum of (3.23) in a reasonable amount of time.

In order to decrease the computations needed, **stochastic gradient descent learning** is often used, where the the weight updates are based on a subset of input data, chosen at random instead of performing calculations using all data at hand, which is less computationally expensive.

To further speed up the search for minima in (3.23), a momentum term can be added when updating the weights, $w_{ij}^{(k)} = w_{ij}^{(k-1)} - \eta\frac{\delta E(\mathbf{w})}{w_{ij}^{(k-1)}} + \alpha\Delta w_{ij}^{(k-1)}$, where the superscript denotes the iteration order of the updates. This means that the learning rate increases with α in the same direction as the previous step took, and will therefore 'gain momentum' as it continues to step in the same direction, and 'decelerate' if the step direction is changed. The process of calculating the stochastic gradient descent with momentum is presented in algorithm 3.

Deep learning

While the perceptron can capture linearly separable data, it can never fit a decision plane satisfyingly to other kinds of distributed data. Deep learning takes the perceptron further by adding one, or several, hidden layers between the input layer and output layer. This mapping of data between layers become increasingly complex as data moves through the network. As the method is computationally expensive, it is only in the last decade there have been outstanding results from neural networks for classification and prediction. While the multiple layered perceptron is an example of deep learning, convolutional neural networks takes this further by using images directly as input data and finding patterns for those.

Multilayer Perceptron

The figure 3.14 shows an example of a multilayer perceptron (MLP), where a hidden layer has been added in the architecture of the ordinary perceptron. Each input vector is subject to a non linear activation function for each layer it moves through. Adding layers will result in the network being able to capture increasingly complicated patterns in the data, at computational expense. While a perceptron can only form one hyperplane as a decision boundary, each node in an added hidden layer makes it possible to add several hyperplanes, which may classify data that is not linearly separable. Three nodes in a hidden layer, for example, makes it possible to enclose data points in a two-dimensional feature space, as each node correspond to a hyperplane. Convolutional neural networks also contain layers between input and output layers, but

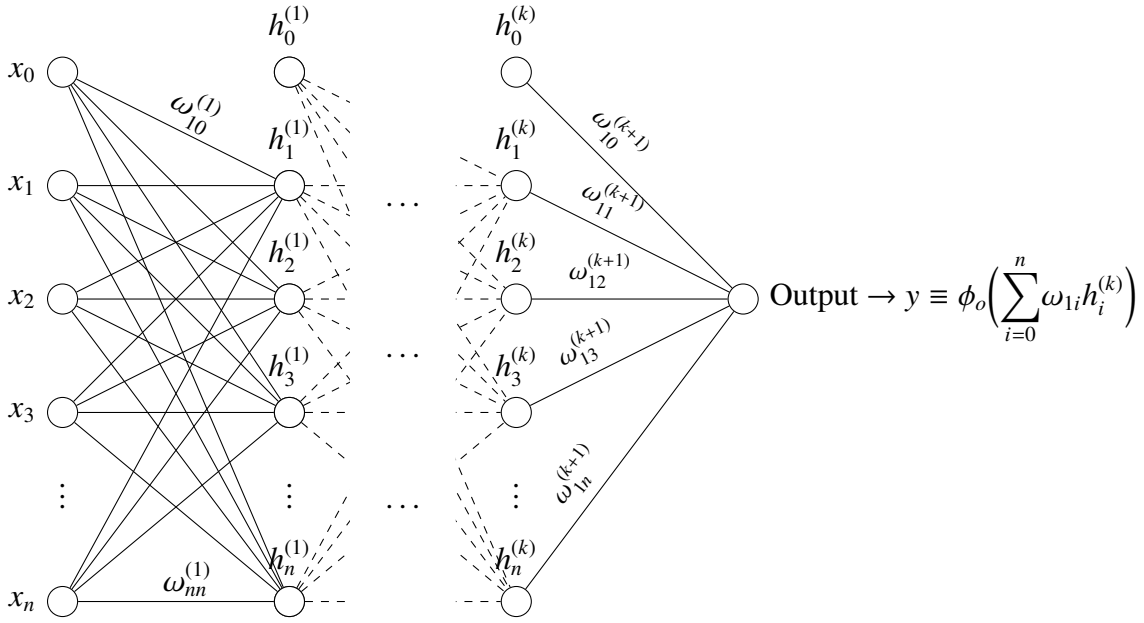


Figure 3.14: A schematic image of a general, fully connected, multilayer perceptron with one output node. Nodes whose subscript are denoted by zero are bias nodes, and are set to $x_0 = h_0^{(m)} = 1$. The notations of a weight $\omega_{ij}^{(k)}$, mean that they weight input from the j:th node of the k:th layer, to the i:th node of the $k+1$:th layer. In a general network, $h_n^{(m)}$ denotes the output of the activation function in the n:th node of the m:th hidden layer. $m = 1$ for the above network. The activation function of the output node is denoted by ϕ_o. [36]

these are all two-dimensional.

Convolutional Neural Networks

Convolutional neural networks (CNN) are a variant of ANNs whose architecture allows it to perform mapping based on an input of images. In the earlier presented neural networks, the input was presented as a one-dimensional vector $\mathbf{x} = (x_1, x_2, ..., x_n)$. For a convolutional network, the input of images is treated as a two-dimensional input vector, a matrix, $\mathbf{X} = (X_{11}, ..., X_{nn})$. As the name suggests, the mathematical operation convolution is central to the learning process of a CNN. Convolution 'filters', so to speak, an input matrix, $\mathbf{I}$, with a kernel $\mathbf{K}$. The discrete case of filtering is considered when working with CNNs and the operation is written out below:

$$S_{ij} = (I_{ij} * K_{ij}) = \sum_m \sum_n I_{i-m,i-n} K_{m,n}. \tag{3.24}$$

As can be seen in figure 3.15, the resulting image S is of smaller size than I. Filtering an image often results in the downsampling of it, mapping the information from one matrix to a

smaller one. Usually, a parameter called 'stride' is defined and describes how a kernel shifts when filtering a matrix, as it transverses from the left to right, and top to bottom of the input matrix. Each tile in such a matrix can be said to correspond to an image pixel. In figure 3.15 the stride s is set as $s = 1$. In figure 3.16 and the max pooling kernel has a stride $s = 2$. Usually, if a chosen stride results in the kernel stepping 'out of bounds', i.e. goes outside the limits of the matrix **I**, that matrix is enlarged by adding a rim of zeros around **I**, referred to as 'zero-padding'.

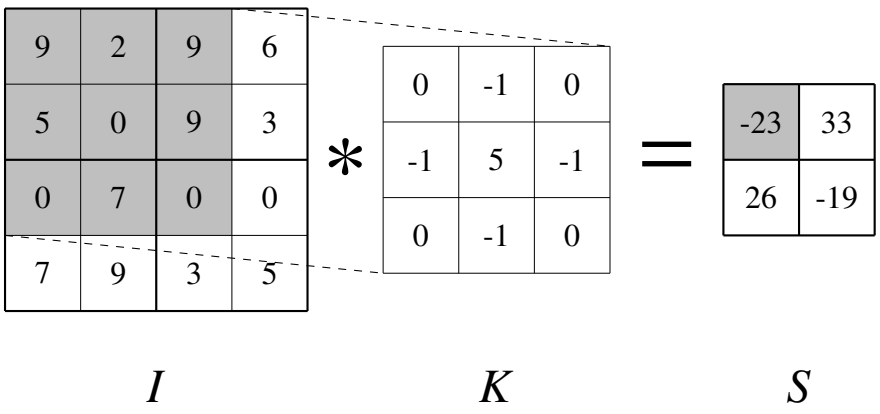


Figure 3.15: A visualised example of expression ((3.24)), with convolution on image **I** using a 3×3 kernel **K**. Here the stride is set to be $s = 1$. [38]

Max Pooling

Max pooling is a commonly used type of filtering in which the largest element in a specified part of a matrix of size is taken out, as is visualised in figure 3.16, where a max pooling kernel of size 2×2 is used on a 4×4 image matrix, using a stride of $s = 2$.

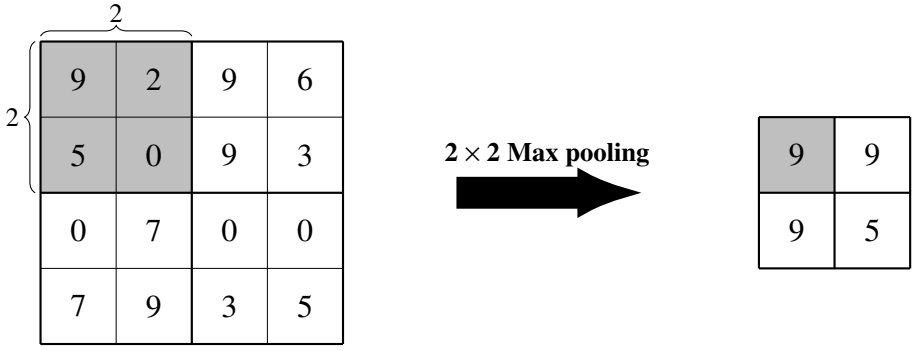


Figure 3.16: Example of max pooling kernel, with stride $s = 2$. [38]

Drop Out Method

The drop out method is used to avoid overfitting, which is essential when the training data set is small. It has shown great regularization properties when implemented in ANNs. In an MLP, the idea is that nodes, with an probability p, do not yield an activation in the network, and can therefore be seen as if 'dropped out' from the network. In CNNs, a dropout layer works similarly, in that the activation from a preceeding layer is set to zero with a certain probability p. The weights which are still tied to the nodes that are not dropped out during backpropagation are then updated by averaging over the found weights over several training sessions where drop out has been used.

Data Augmentation

The performance of any machine learning method is dependent on the amount of data that is used in training, especially in algorithms concerning convolotional neural networks. This kind of training data is not always available and therefore data augmentation is used to add more information to the methods using the original data without causing over-fitting. The idea is to change the original data images in a way that is perceived by the network as new data. A simple example is to rotate or mirror the images, and also using specific parts of an image or simply translating the image in different directions can substantially increase performance. Another way to increase the data set is to perform an elastic deformation to the images, as Ronneberger et al. did in their original implementation of the network [39]. This method of creating 'new' information from the existing training data also result in a more robust classifier, as the algorithm learns to be invariant of the orientations in the images, and therefore being robust to rotations, translation and deformations that might occur in unknown data that is to be classified.

U-Net

U-net is a fully connected CNN used for semantic segmentation of images. It have shown considerably good results in the field of segmentation using small training sets [39]. Ronneberger et al. reports satisfying results on segmentation of cells using image sets of only 30 pictures. This miniscule amount of training data makes the system heavily reliable on the use of extensive data augmentation (the process of increasing the amount of data in a set by modifying the data in the available image set, explained later in this section). The name, U-net, stems from the schematic image of the network, see figure 3.17, which resembles an **U**, which corresponds to the network firstly downsampling the image in several steps and later upsample it. Each step involves a downsampling from half of the former size, by using a series of convolutional filter followed by a ReLU filter (where each input value is used as an argument in the rectifier linear function, $\phi(a) = max(0, a)$), and downsampled by a 2×2 max pooling filter. Before downsampling by max pooling, the convolved image in each step is cropped and used in conjunction with a later upsampled image, in order to effectively use the information gained at each downsampling step. Drop out filters are also used, where data from a previous layer is set to zero at random before being used as input in the following layer. The last layer which performs the classification is a softmax layer, see expression (3.17).

The 'encoder depth' D is a parameter which decides the factor that the image is downsampled to, i.e. the number of steps before the upsampling begins, where the image has been

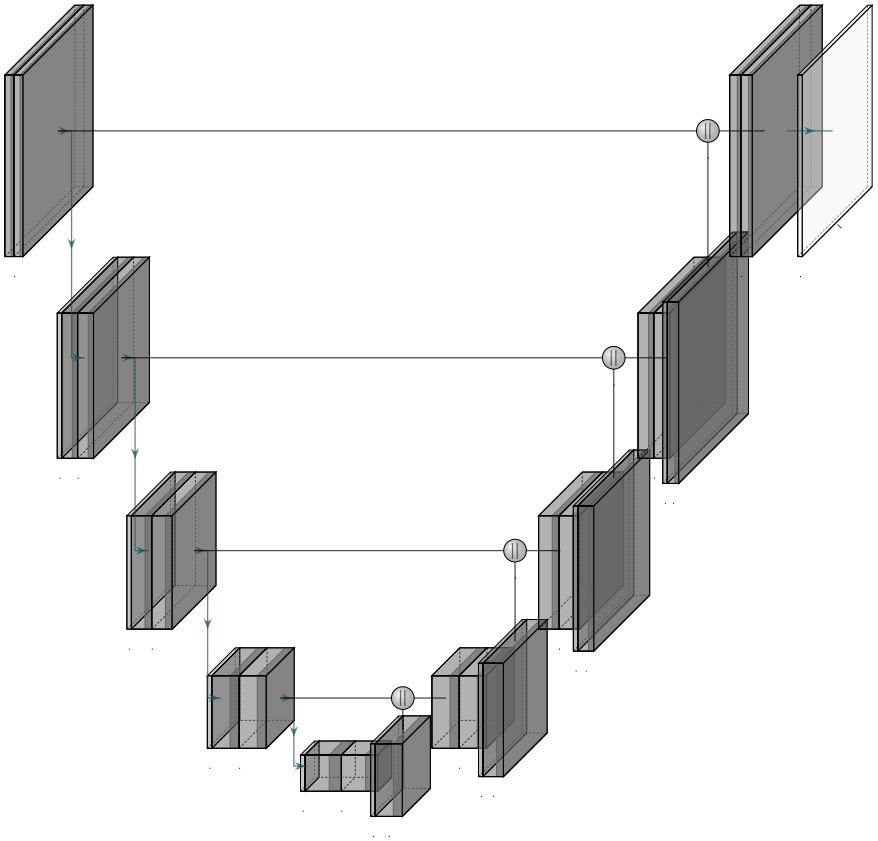


Figure 3.17: A schematic image of the encoder-decoder CNN that is U-net. Each vertical arrow represent a downsampling using convolutional layers, and each horizontal line corresponds to a concatenation, denoted by the circles, of an image to an upsampled image. The final layer in the top right corner is a layer which has the softmax function as activation function. [40]

downsized by a factor of 2^{-D}. In figure 3.17, a schematic image of a U-net with encoder depth of $D = 4$ is depicted.

Chapter 4
Methodology

4.1 Color Correction

To perform the color correction, the colors in the X-rite marker in an image must be mapped to a ground truth color value, as described in section 3.1.2. Using the checkerboard cornerpoints in the marker, whose coordinates were provided by supervisor Mikael Nilsson, a projective mapping could be found to the cornerpoints in the groundtruth marker. This was done with MATLABs function 'estimateGeometricTransform' [41], which finds a mapping between images taken with the same camera, as discussed in section 3.2.2. Using the found mapping, the coordinates of the colored disks in the ground truth were then mapped to corresponding color markers, so that the corresponding color vectors are found between the same marker points in the images. From this mapping, a new image set of color corrected images could be found.

An alternative image set was also made by first performing a linear mapping, as described in section 3.1.2, between the marker in the image and the groundtruth marker. This linear mapping was made for each of the three RGB-channels and an average of the mapping for each channel was used on the color vector in each of the colors in the colorchecker. This mapping was performed in an attempt to even out the distribution of light on the marker, before performing the root polynomial mapping.

4.2 Weight Estimation

Weight estimation was to be performed by comparison of the known weights of each training image. This approach was meant to mimic the method used in [1], where visual assessment of food classes was used to estimate weights of food. The correspondence between the amount of food pixels to the reference object, of known size, was assessed.

4.2.1 Perspective Correction

Assuming a successful segmentation of food components, the ratio between the pixel size of food components and marker tile were then compared to the areas of foods found in the photo atlas. The factor which then came into play here was that the images were not taken at the exact same height or angle, which can cause distortions in the results. This because the photographed marker would not have a consistent area throughout an image set. To mitigate this, a square area, spanned by four checkerboard cornerpoints on the marker (of equal length between neighboring points), were considered. From the top view, the sidelengths of the square were taken out, from which a mean length of the chosen square area was calculated. Starting from the top left corner of the square, a new square was placed perpendicular to the edges of the image, with the above mentioned mean sidelength. To this newly formed square, a projective mapping, as described in section 3.2.2, was then calculated from each of the two images, taken at top view and side view. This was meant to result in a consistency between images, as the marker area would remain the same, and could therefore be compared to the segmented area of the meal component for an estimation of the area. A visualised example of the method described above can be seen in figure 4.1.

4.3 Segmentation

4.3.1 Support Vector Machine

A support vector machine approach was first used in order to separate the foreground from the background. The SVM was trained on the training images using the RGB-values in each pixel as features.

Sliding Window Approach

Instead of training on single pixel one at a time, a sliding window approach was also used. Here the classification was based on the features of patches in 9×9 pixels. The training images where cropped so that the image rows and columns became divisible by the window size in order to utilise this technique.

4.3.2 Decision Tree - Hard Mining Approach

Hard mining is a method to increase performance in machine learning. One would want to avoid the learning algorithm to excessively train on easily distinguishable training data, as the method is not expected to improve from training on data whose patterns it has already learned. Hard mining mitigates this problem by drawing training data, iteratively, from already classified data, but only choosing data that the algorithm was particularly indecisive with.

Using a set of 17 atlas images for each of the five food classes, the hard mining approach was initialised by taking out a random sample of 100 pixels of foreground, and 100 pixels of background. This sample was used to train a random forest algorithm, using the three LAB-values in combination with SURF-vectors from each pixel as feature vectors of length 67. The acquired data was in turn used to classify the next image in the image set. From this newly

classified image, the probabilities of each pixel belonging to a certain class were taken out, so that only pixels classified below a certain probability were sampled from. This probability threshold was chosen to be 0.7. This means that if a pixel was classified to belong to a certain class with a probability of 0.8, it would not be part of the set used for future training. From these samples, which proved harder than others to classify, training data of foreground and background, 100 pixels respectively, were extracted. Taking training data from an earlier classified image to classify the next image was performed in a loop of seventeen images for each of the five food classes, and the final trained decision tree was used to classify an image which was not used in training procedure. This was done by plotting a ROC-curve, where each unique classification probability was used as a threshold for classification, in order to assess the confidence value which yields the optimal AROC during classification.

4.3.3 Bag-of-Visual-Words Method

In the acquisition of features, 10 training images (rescaled to $\frac{1}{25}$ of the original size) were used. For each pixel, a SURF feature vector (64 values for each pixel) was taken out as well as a LAB color vector (3 values for each pixel). Using Matlabs implementation of SLIC [42], the training images could be partitioned into a chosen number of superpixels. Using the feature vectors extracted in this manner, MATLABs implementation of the k-means method, the function 'knnsearch' [43], was used to find cluster centroids of the features in the 67-dimensional feature space.

A vocabulary was then made using these centroids. Any features taken out in the testing phase were compared to this vocabulary, and each instance of a superpixel was said to belong to the same class as the cluster centroid it was closest to (in Euclidean distance). This adds to a stack in a histogram, in which each stack corresponds to a centroid. This histogram is then normed and used as encoded feature vectors for training in a random forest algorithm.

Each training image was divided into a certain number of superpixels, each compared to the groundtruth mask. If 70% or more area of a superpixel contained foreground pixels, then the area of the superpixel was labeled to contain food, otherwise background. For this label, the features to be encoded were taken from the bounding box containing the current superpixel.

A spatial pyramid of three layers was also applied on each superpixel bounding box, as a means for acquiring spatial information from the surroundings of each superpixel. The bounding box was divided into 2×2 tiles at one level of the pyramid, and 4×4 at the top of the pyramid. Encoded feature histograms were found for each tile at each level and the average was taken out for the whole level, as well as weighted average, depending on how many tiles were used at that particular level. This simply means that features found at levels with a finer grid were weighted to be more important. The average top level histogram was weighted by a factor of 4, the middle layer average histogram is weighted by a factor of 2, and the histogram of the original bounding box was weighted by a factor of 1. Each of these histograms were added and normed to form a feature vector for the superpixels.

An ablation test was performed, where three different parameters were changed between sessions: The number of feature clusters, the number of superpixels (which the images was to be partioned into), as well as the probability threshold. This threshold decided what probability, for a superpixel to belong to a certain class, must be exceeded to be classified as such. The number of clusters for each class were made to range between $8 - 24$, in steps of 8. The number of superpixels ranged between $150 - 550$, in steps of 100. The confidence threshold,

for which the probability of a superpixel belonging to each of the different classes, ranged between $0.4 - 0.9$, in steps of 0.1. As such, the process was iterated 75 times. For each of these, a ROC-curve and AROC value for each of the five food classes used could be taken forward for assessment. The set of parameters which yielded the five AROC values which together yielded the highest L_2-norm value, was deemed the optimal set of parameters, of those used in the ablation test.

4.3.4 U-net

The practicalities concerning training the CNN used are presented below. The computational ability of the computer used gives an indication on the speed of training a network. The dataset used is important for the performance, and can be enhanced using data augmentation.

Software and Hardware

A computer with a single CPU, Intel Core i5-4278U, was used for training the network. MATLAB Rb2019 implementation of U-net architecture was used [44], based on the work by Ronnerberger et al. [39].

Data Augmentation

Each image was rotated 90°, 180° and 270°, and, in addition to the set, were mirrored horizontally. This increased the data set fourfold. These augmentations were also applied to images where the meal components had been zoomed in, in such a way that the foreground take up roughly 75% of the area of the image. This resulted in a sixteenfold increase of the training set. This increase was further doubled, in accordance to [39], by performing a random elastic deformation of each image. These above mentioned examples are visualised in figures 4.2 and 4.3. The elastic deformation was performed by applying a vector field on the image to cause apparent displacement using the MATLAB implementation of which was taken forward by Franco [45].

Training

Due to the limitations in memory and performance in the software used, the image sets were resized to 256×256 pixels RGB-images. The minibatch size was set to 16, and the learning rate to $\eta = 10^{-4}$. The number of epochs were set to 20, but the training could be cancelled before that if the accuracy on a validation data set, images which were not part of the training, stayed roughly the same during the last occurring epochs. For the results shown in section 5.2.4, the network was trained for 20 epochs, 3440 iterations for the network using images corrected for color, and 3346 iterations for the original images. Only binary classification was assessed, using '*Food*' and '*Background*' as labels.

Figure 4.1: An example of how projective correction can be performed. The two columns are photos of the same plate with a meal component taken at different angles, the bottom row showing the original images. The perspective correction, as described in section 4.2.1, of the images and their corresponding ground truth mask, are shown in the middle row and top row images, respectively.

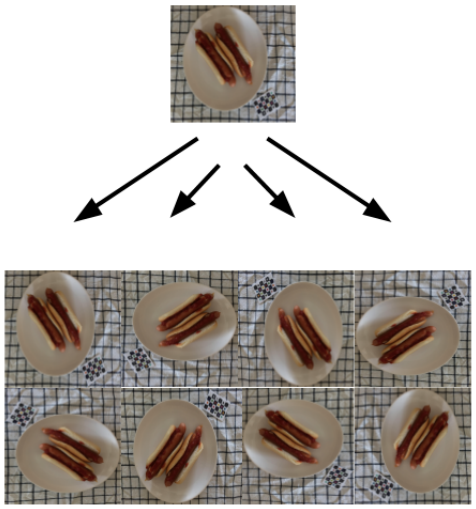


Figure 4.2: Example of data augmentation, where an image has been rotated as well as mirrored, increasing the data set eightfold.

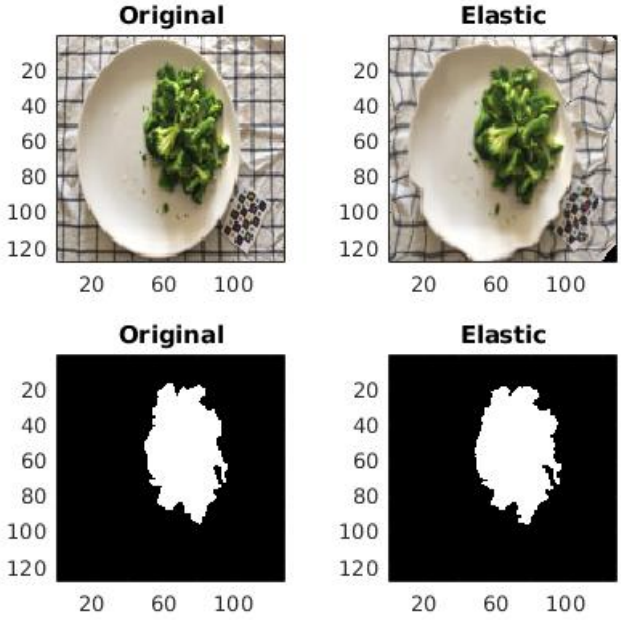


Figure 4.3: Example of data augmentation, where an image has been subject to an elastic deformation (top row), as well as its corresponding ground truth mask (bottom row). It can be seen that the images differs slightly from each other in shape.

Chapter 5

Results

5.1 Color Correction

In this section, the results concerning the work of color correction in this thesis are presented, according to the theory presented in section 3.1.2, will be presented. Visual assessment was made to evaluate the performance of the methods used. Figure 5.1 shows the color correction using linear mapping, while figure 5.2 shows an example of color correction by using root polynomial mapping. Both can be seen to brighten the image in a way that is expected, but it can be seen in figure 5.1 that the white cloth is given a blue shade, while figure 5.2 seems more true to the original white color. Another example of using the root polynomial mapping on a rice dish can be seen in figure 5.3, where the correction has resulted in brightening the image too much, possibly because of the shading on the X-rite marker. Since the correction brightens the shaded marker up to a ground truth value, everything outside will appear much brighter than desired. The same rice dish can be seen subjected to a color correction method meant to mitigate this problem, described in section 4.1, in figure 5.4. In this figure, the color correction seems brighter than desired at certain spots in the image, but overall seems to show positive results. The rice dish appears as white instead of slightly yellow, and the cloth is also closer to the original white color when corrected, than the shade which can be seen on the cloth before color correction.

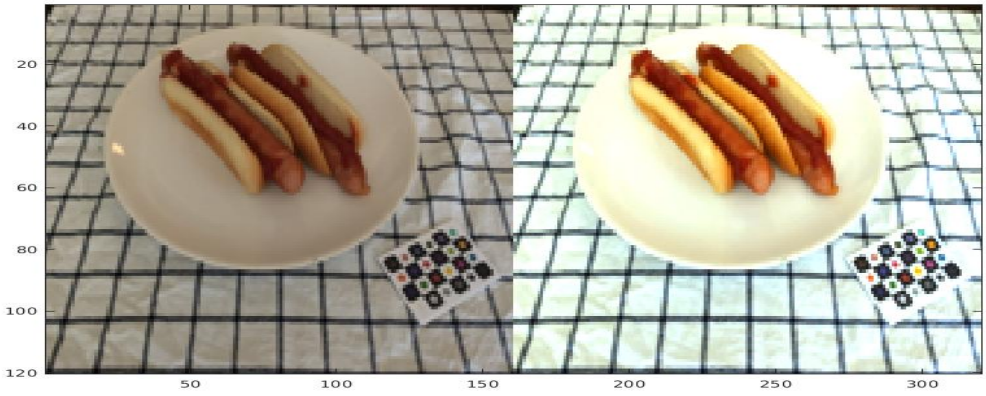


Figure 5.1: Example of color correction as made by linear mapping. The image to the left is the original captured image, and the right is corrected using the colorchecker as color reference.

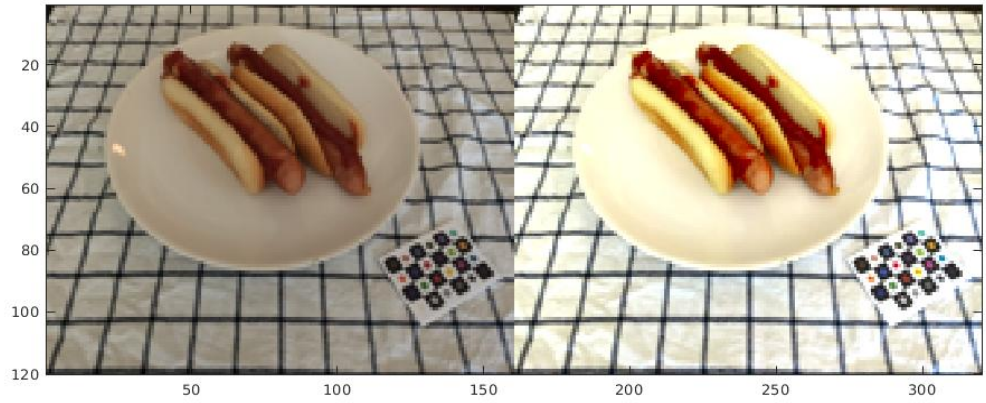


Figure 5.2: Example of color correction as made by root polynomial mapping, of the fourth order. The image to the left is the original captured image, and the right is corrected using the colorchecker as color reference.

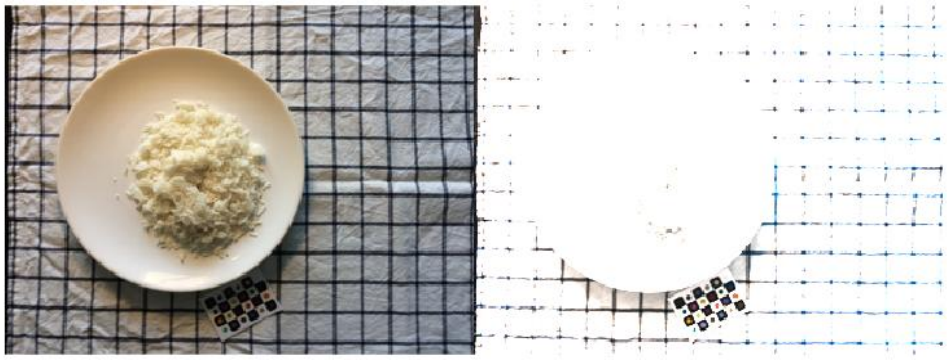


Figure 5.3: Example of an attempted color correction as made by root polynomial mapping, of the fourth order. The image to the left is the original captured image, and the right is corrected using the colorchecker as color reference. The colorchecker can be seen as partially shaded by the edge of the plate.

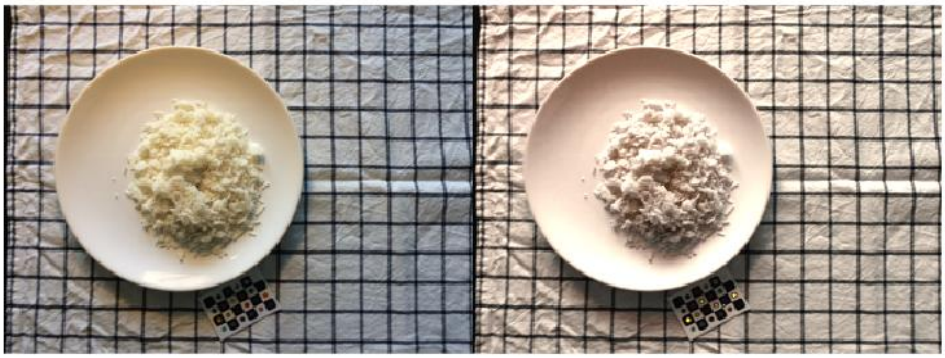


Figure 5.4: Example of color correction method. The image to the left is the original captured image, and the right is corrected using the colorchecker as color reference. The correction was made by first finding a linear mapping between colorchecker and ground truth values. These found values were then subjected to a root polynomial mapping, of the fourth order.

5.2 Segmentation

In this section, the results of the methods for semantic segmentation explained in section 4.3 are presented. Example images used throughout the chapter are of a rice dish and a plate of haricots verts, both meal components of weight 250 g. Using only two dishes was partly not to fill this section with images when the ROC-curves presented showcases the quantitative results for test images of the dishes used, i.e. that were not part of the training of the machine learning algorithm.

5.2.1 Support Vector Machine

No quantitative results were extracted when using the support vector machines as a classifier, the images which were produced showed large portions of background being classified as food, and as such the results were visually assessed to be too poor to continue working with SVMs.

5.2.2 Random Forest: Hard mining

Figure 5.5 displays the ROC-curves found when using the hard mining approach. Here the same classifier was used for each test image, which deemed a pixel to belong to a certain class if the probability of it doing so was above 50%, otherwise the pixel was classified as background. Disregarding the results achieved when classifying sausage (which is most likely due to some error in the implementation), it can be seen that rice is the hardest of the meal components to classify. As it is a white dish against a white plate, it is expected to also be harder to distinguish the food from the background. Examples of classification can be seen by the hard mining approach in figures 5.6 and 5.7, where only the pixels that were classified as haricots verts and rice, respectively, are seen on the right as a yellow mask, exemplifying the performance of the classifiers. In the figures 5.8 and 5.9, all pixels are denoted for what class they were found by the trained classifier to belong to. It can be seen in figure 5.8 that many pixel of the haricots verts dish was mistakingly classified as broccoli. In figure 5.9, the low performance of rice, as seen in figure 5.5, is further visualised as a large portion of the dish can be seen wrongly classified as background.

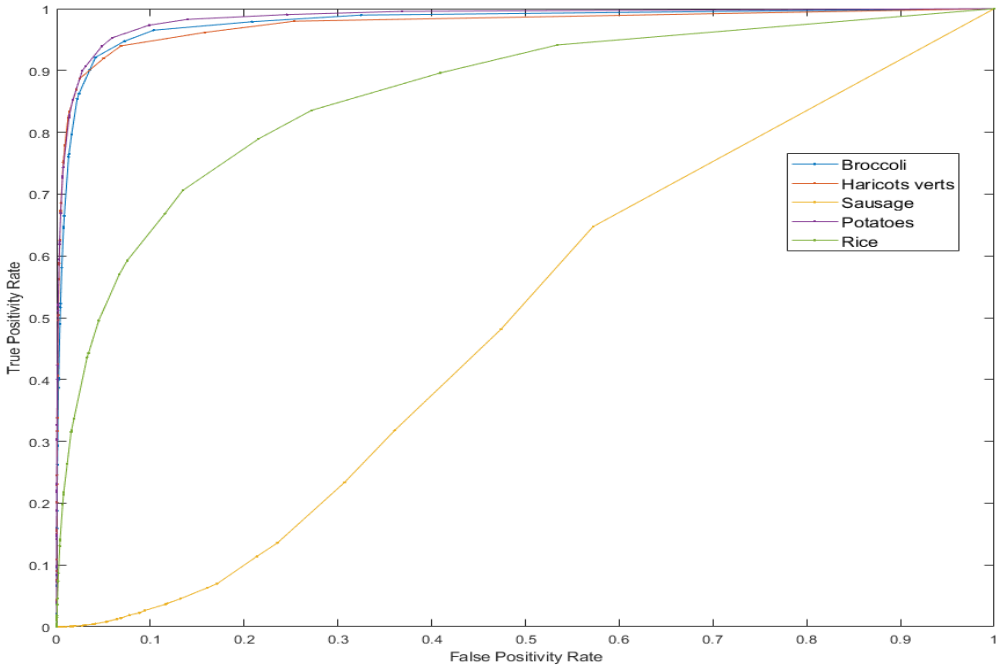


Figure 5.5: ROC-curves achieved using hard mining approach, binary classification between food and background. The threshold for sampling of pixles were at pixels with a probability of 0.7 or less to be classified correctly. Area under ROC-curve is 0.9760 for broccoli, 0.9699 for haricots verts, 0.5496 for sausage, 0.9767 for potatoes and 0.8387 for rice.

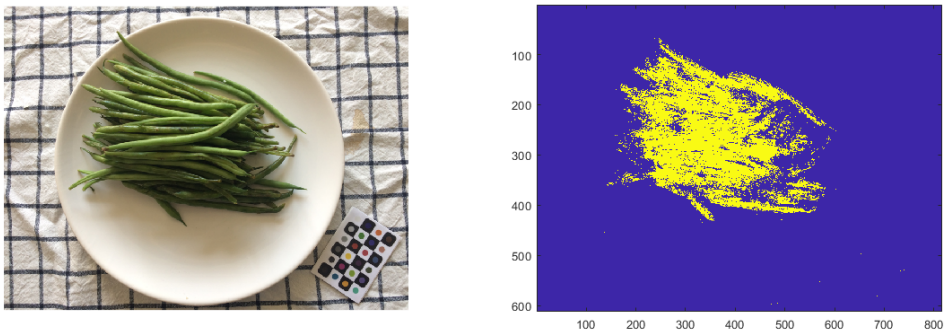


Figure 5.6: Example of dish being classified using hard mining approach. Only pixels that were classified to be haricots verts are seen (in yellow) in the right image. The pixels that were not classified as haricots verts, as well as pixels that were classified as background, can be seen in dark blue.

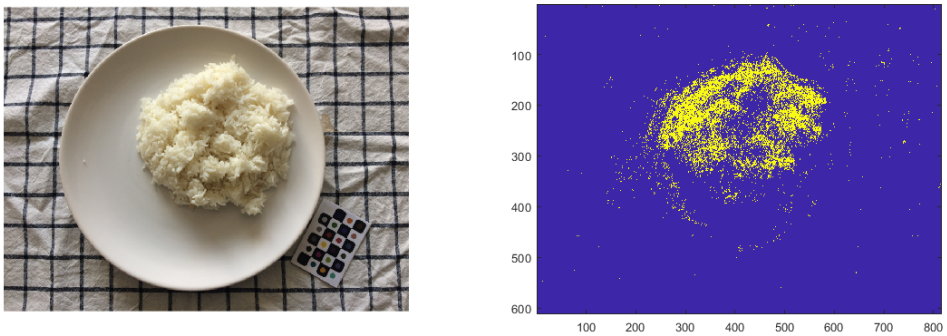


Figure 5.7: Example of dish being classified using hard mining approach. Only pixels that were classified to be rice are seen (in yellow) in the right image. The pixels that were not classified as rice, as well as pixels that were classified as background, can be seen in dark blue.

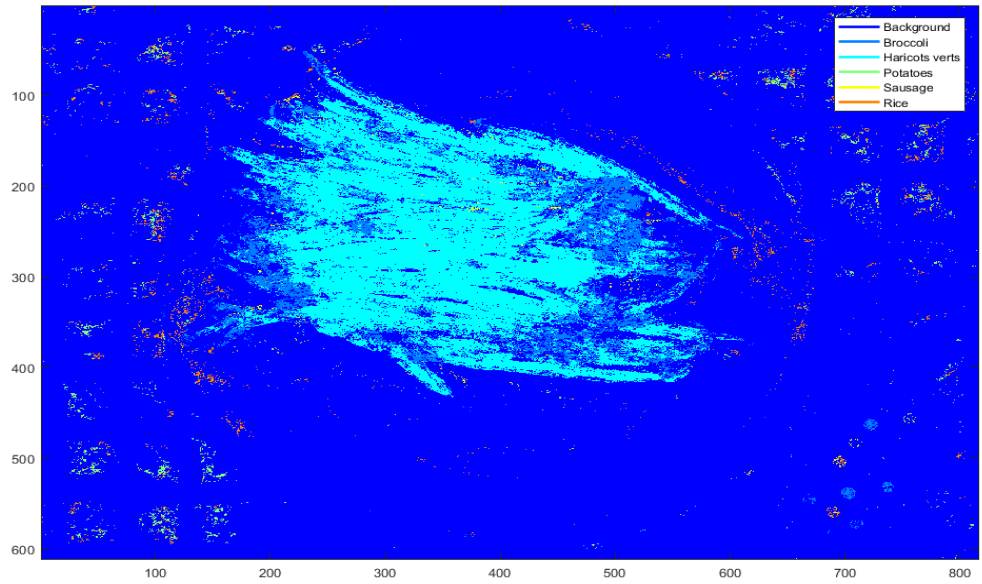


Figure 5.8: Example of dish, haricots verts being classified using hard mining approach. The different colors correspond to a certain food class that the pixels were classified to belong to.

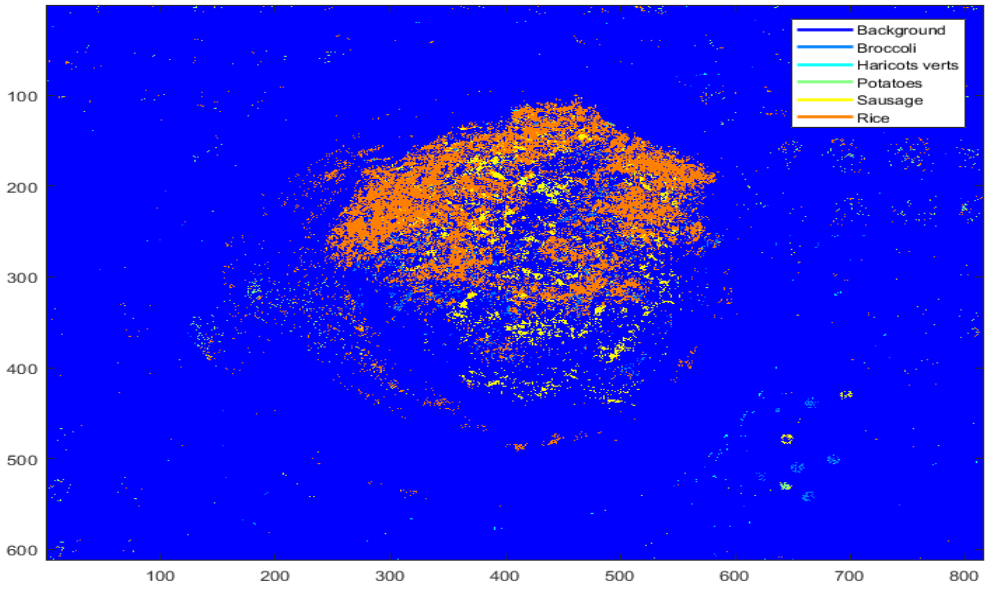


Figure 5.9: An image of classified pixels for a rice dish using hard mining approach. The classified pixels are denoted by colors which are explained by the legend in the figure.

5.2.3 Bag of Visual Words

For the bag-of-visual-words methods, the ROC-curves can be found in figure 5.10, where the AROC can be seen to be higher than that achieved using hard mining (as seen in figure 5.5). Figures 5.11 and 5.12 display examples of the classification of meals using the bag-of-visual-words approach. As when using hard mining, the haricots verts can be seen being misclassified as broccoli in figure 5.11, and also background. Also similarily, in figure 5.12, the erroneous classification can be seen to be between rice and background.

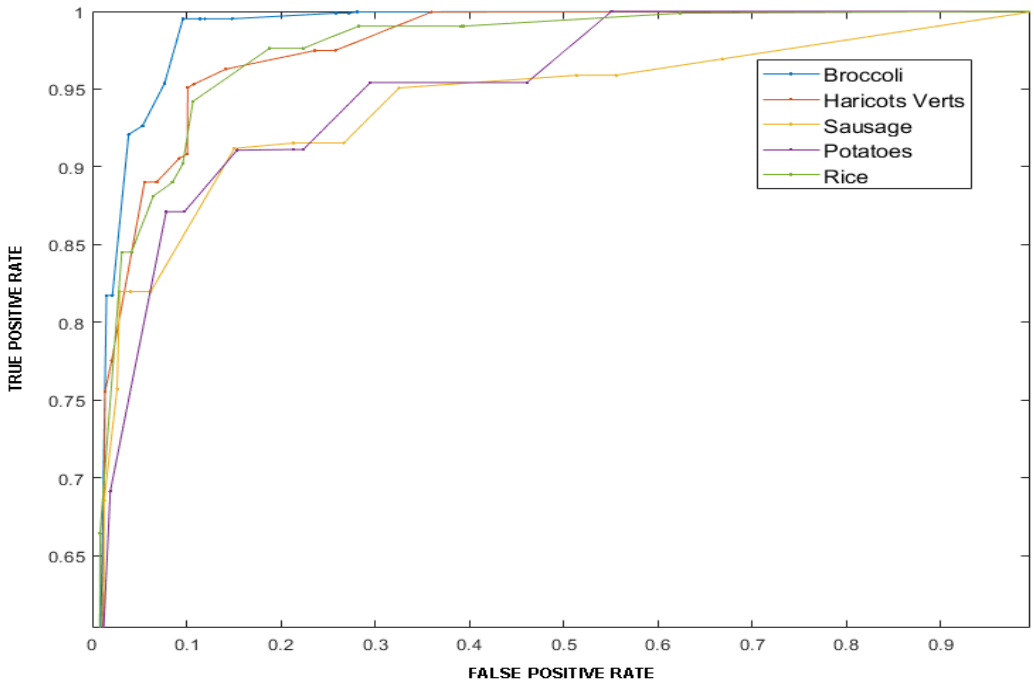


Figure 5.10: ROC-curves achieved using the bag-of-visual-words approach. Area under ROC-curve is 0.9859 for broccoli, 0.9730 for haricots verts,0.9402 for sausage, 0.9147 for potatoes and 0.9635 for rice. These values were overall the best achieved in the ablation test, and correspond to using 16 clusters for each class in the feature space, 150 superpixels, and a 0.7 confidence threshold.

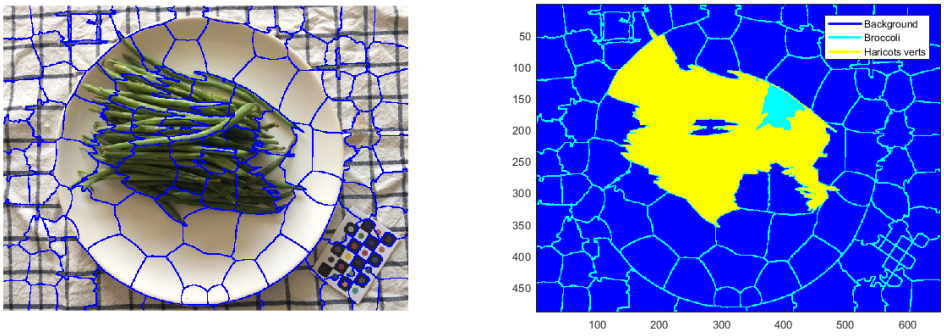


Figure 5.11: Visualisation of the bag of visual words method for classification, performed on haricots verts. Here, 8 clusters were used in the feature space, with about 100 super pixels, and a confidence threshold of 0.5.

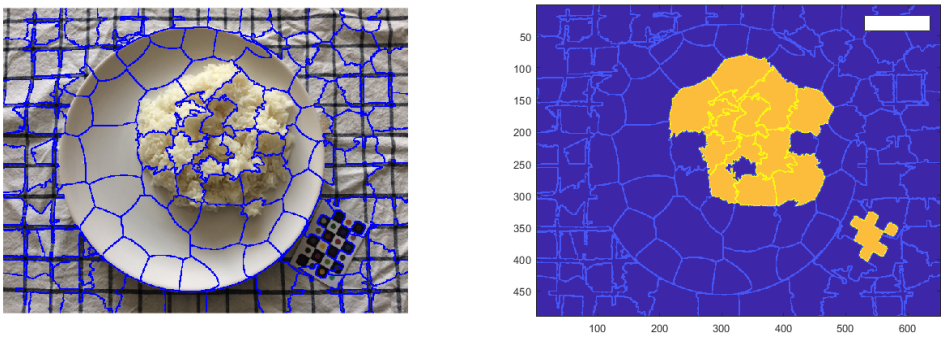


Figure 5.12: Visualisation of the bag of visual words method for classification, performed on a rice dish. Here, 8 clusters were used in the feature space, with about 100 super pixels, and a confidence threshold of 0.5.

5.2.4 U-net

Concerning the results of U-net, two ROC-curves were yielded, where the classifier only differentiated between food and background. Figure 5.13 shows the ROC-curve for a network that has been trained and tested on images which have not been color corrected, while figure 5.14 show the resulting ROC-curve when having used color correction on the photo atlas. Even when only differentiating between two classes, the AROC values achieved are worse than expected, and certainly worse than any of the AROC values when using the bag-of-visual-words model for distinguishing between six classes, as seen in figure 5.10. Examples of the binary classification using U-net can be seen in figures 5.15 and 5.16, when training the U-net on images without using color corrected images, and figures 5.17 and 5.18, when using images which had be color corrected. As can be recognized from the other methods used, the figures

5.15 and 5.17 containg haricots verts are easier to segment, albeit poorly, than images of rice, seen in figures 5.16 and 5.18.

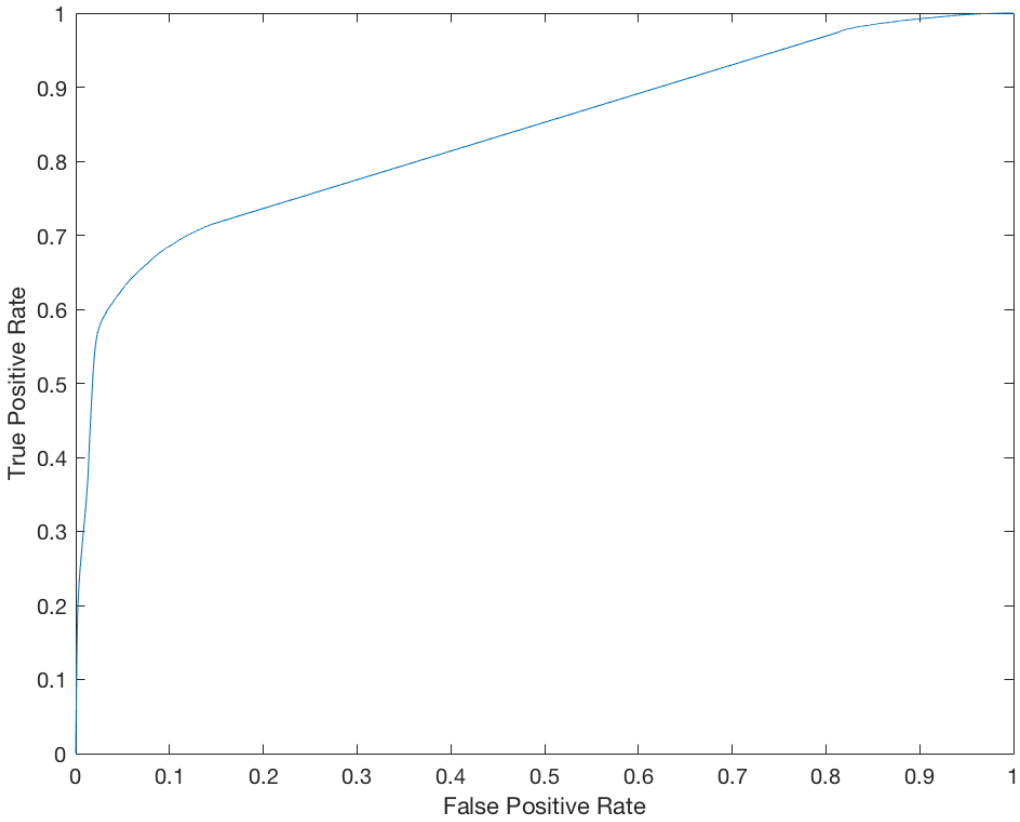


Figure 5.13: ROC-curve achieved classifying images with a U-net, using binary labels ('food', 'background') on images which were not color corrected. Area under ROC-curve is 0.8386.

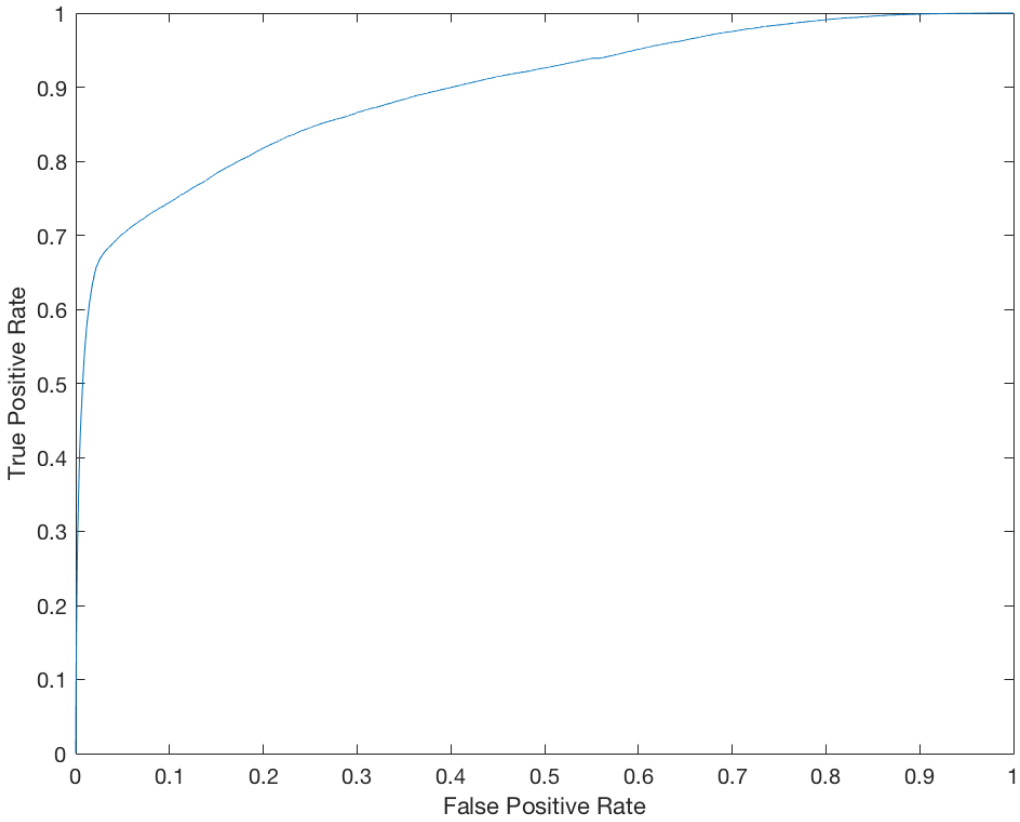


Figure 5.14: ROC-curve achieved classifying images with a U-net, using binary labels ('food', 'background') on images which were color corrected. Area under ROC-curve is 0.8971.

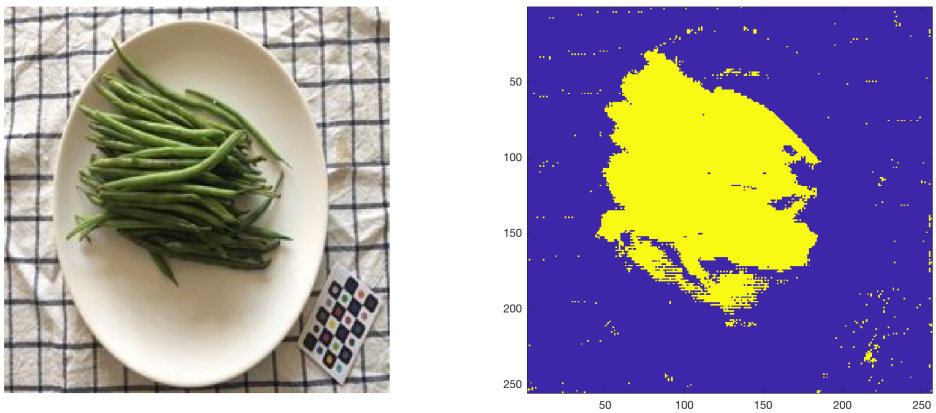


Figure 5.15: Example of food segmentation using U-net on haricots verts. For the training and validation of the network, images that were not color corrected were used.

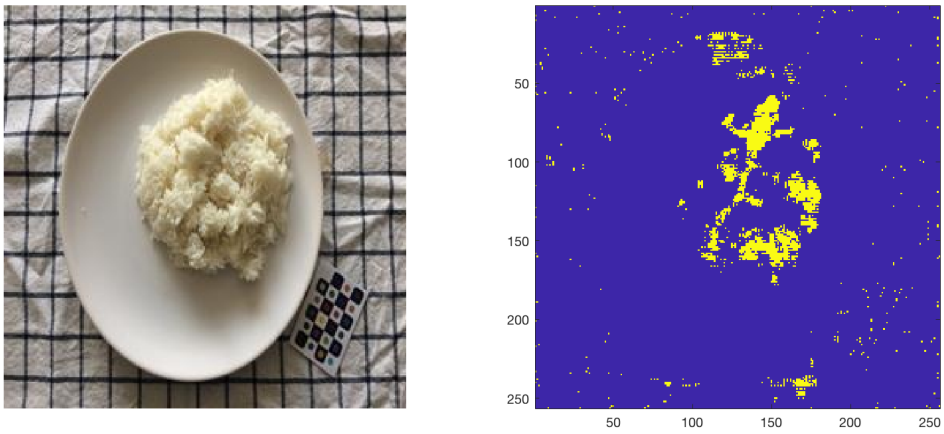


Figure 5.16: Example of food segmentation using U-net on rice. For the training and validation of the network, images that were not color corrected were used.

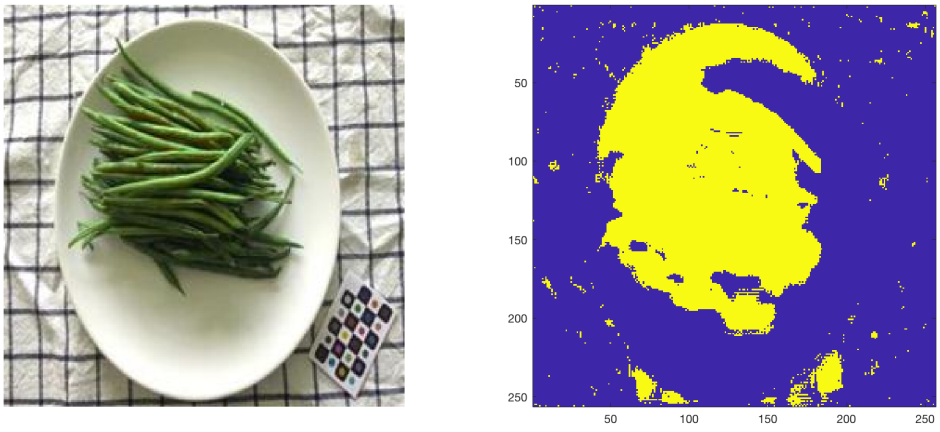


Figure 5.17: Example of food segmentation using U-net on haricots verts. For the training and validation of the network, images that had been color corrected were used.

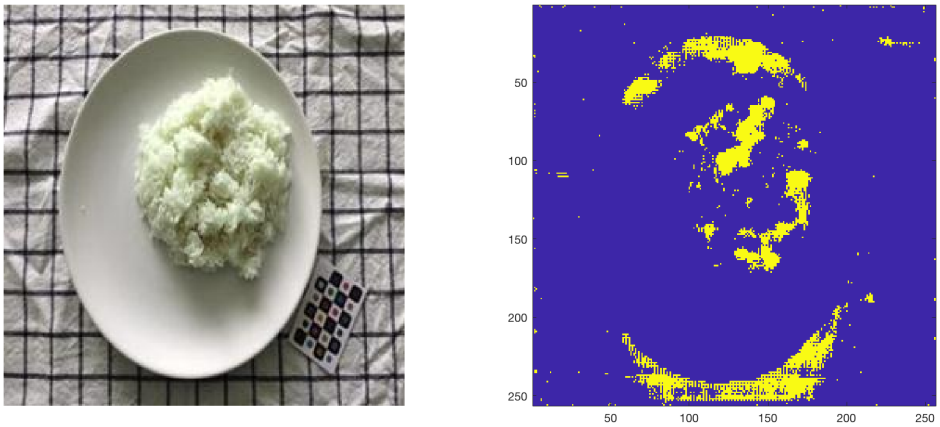


Figure 5.18: Example of food segmentation using U-net on rice. For the training and validation of the network, images that had been color corrected were used.

5.3 Weight Estimation

Using the images in the photo atlas, corrected for perspective as discussed in section 4.2, a correspondence between weight and area could be found, as presented in figures 5.19 and 5.20 for top view images and side view images, respectively. A linear regression curve was chosen as visual assessment of the data which indicates a linear relation between area and weight. Not surprisingly, it can be seen that meal components with a firm structure, such as potatoes or sausage, more closely follow the linear regression line than meal components which are more prone to deformations, such as rice.

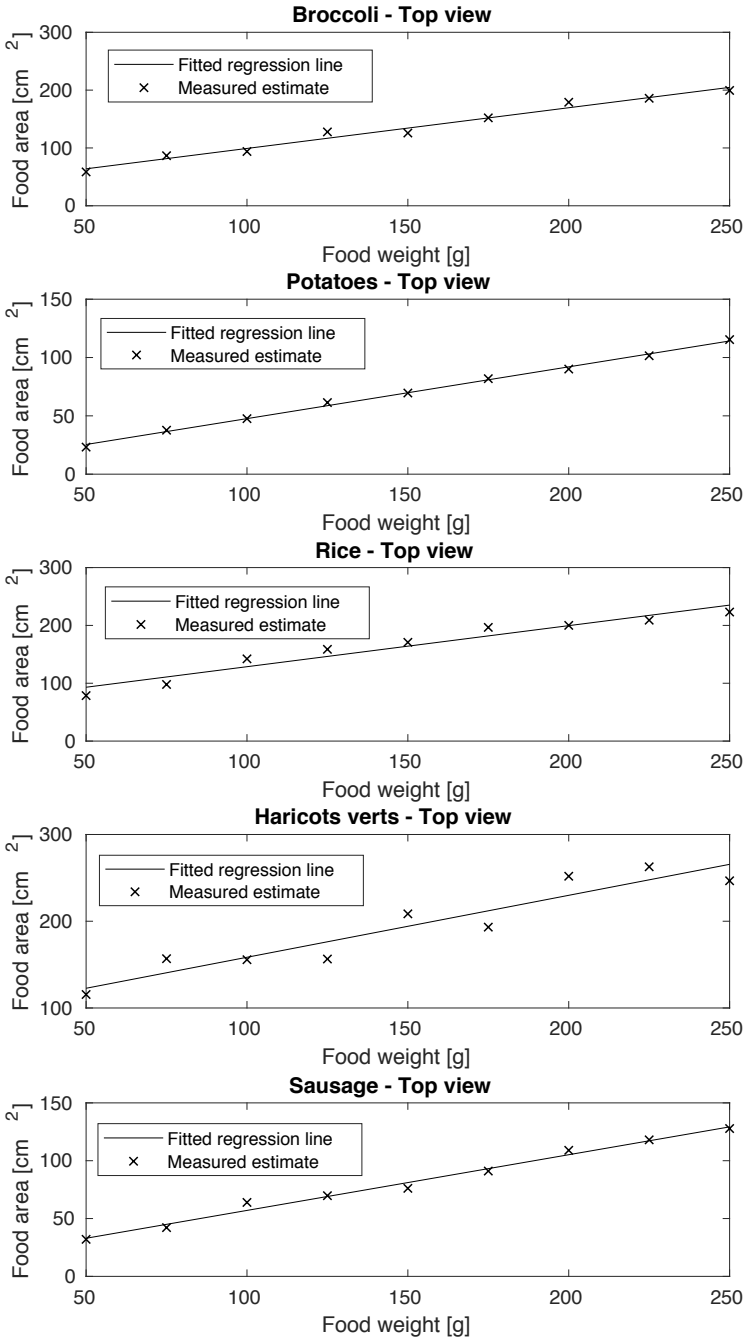


Figure 5.19: The area of the food in the photo atlas, as a function of measured weight. The area is calculated by comparison of a marker of known size after perspective correction. The regression function is found by using the MATLAB function 'polyfit' [46].

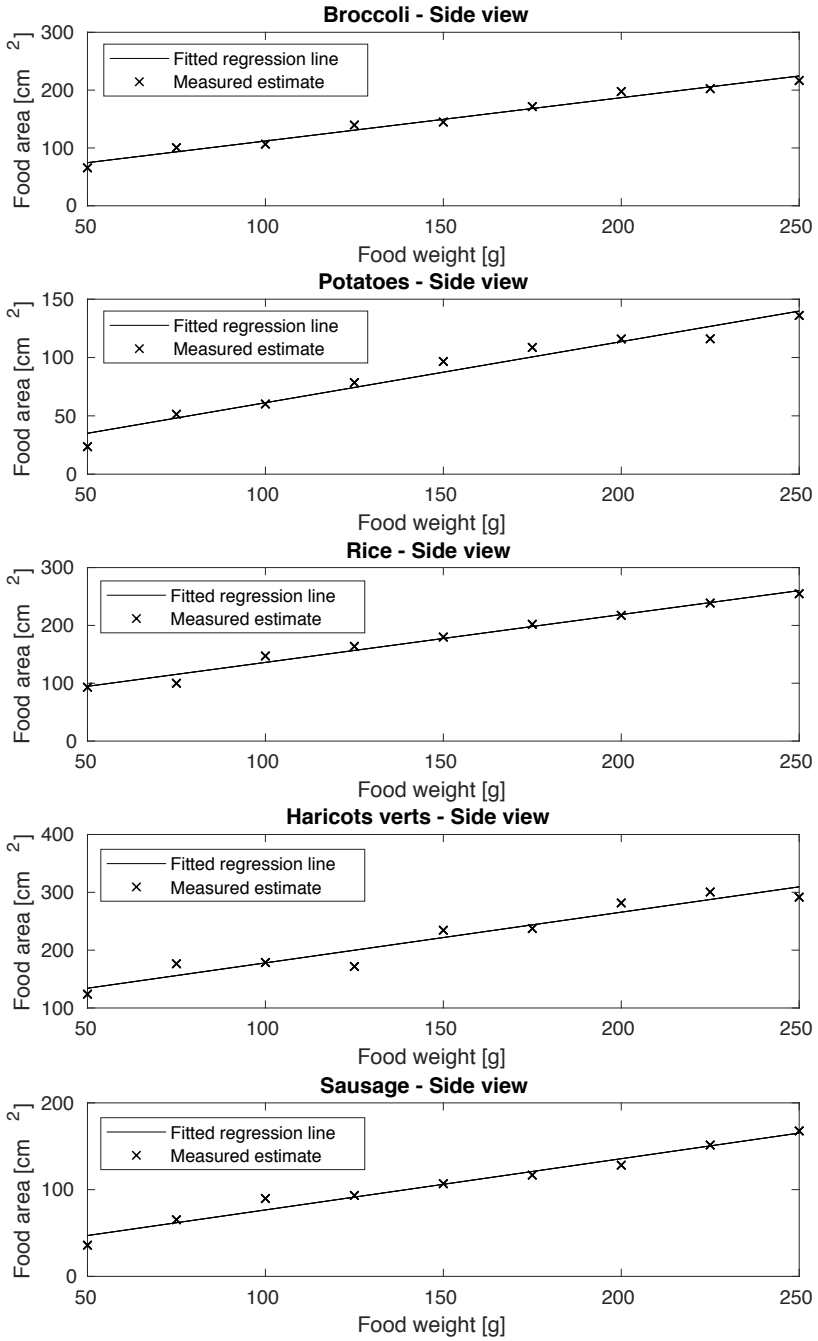


Figure 5.20: The area of the food in the photo atlas, as a function of measured weight. The area is calculated by comparison of a marker of known size after perspective correction. The regression function is found by using the MATLAB function 'polyfit' [46].

Chapter 6
Discussion

6.1 Choice of Food Classes

Sausage, potatoes and broccoli were chosen because they differ in color and texture from each other, and should make a sharp region boundary between themself and the white plate used. This would, presumably, simplify classification of these components. Rice would, with the same reasoning, be difficult to find a boundary for, and would therefore be used as an example of a hard classification problem. In the same manner, haricots verts resembles broccoli in color, but maybe not as much in texture. As such it would also serve as an indication as for how well a classification algorithm fared when using food classes of similar appearance.

6.2 Color Correction

Only visual assessment was used when evaluating the performance of the color correction algorithm. One way to make the assessment more quantitative could have been placing another X-rite colorchecker on a plate, where food was usually placed, and perform a color correction on that. In such a way, the error between the ground truth value and the found values on the other X-rite chart could be used to assess the method more thoroughly.

As it stands, the linear method of color correction, exemplified in figure 5.1, is, by visual assessment, less correct than the root polynomial color correction method, as seen in figure 5.2. The linear method seems to portray the white cloth in a shade of blue, and altogether give a more 'cold' appearance to the image.

A factor of error found, when using the X-rite color marker, was unsatisfactory illumination of the marker, i.e. failing in keeping the amount of light exposure on the marker the same as on the object that was to be color corrected. Many images were taken with the marker partially shaded, resulting in skewed results. Objects in the image that are not shaded are subjected to a transform which exaggerates the color mapping, resulting in images that seem overexposed in

light, as figure 5.3 is an example of. This was not an uncommon result when performing color correction throughout the image sets. Indeed, the succesful result of root polynomial color correction, seen in figure 5.2, seems to be an outlier when directly performing the correction. Figures 5.2 and 5.3 show the general behavior found in the image sets after root polynomial color correction, with or without averaging the lighting over the marker.

The method which was used to mitigate the effects of shading on the marker, linear averaging of the shading, is hard to motivate. It removes the initial point of having a robust set of reference colors present in the image. The colors acting as references changed depending on the distribution of shade which was partially cast on it. Averaging the distribution of that shade is essentially the same as using a different reference marker for each image. However, for the purpose of which the method is taken forward, parameters such as lighting and shading will always be subclinical, as to not put too much workload on the participants in a dietary study. As can be seen in figure 5.4, the result of the proposed method is, at least by visual assessment, on par with the results in figure 5.2, the successful root polynomial mapping. The rice in the image is corrected from having a yellow sheen (possibly from lighting of a nearby lamp) to become clearly white, as one expects rice to be. Despite the apparent faults, the author would like to argue that all images were subjected to the same transform, and in that sense are only dependent on lighting as an outside parameter, which is no worse than the images were without an attempted color correction. Even so, the somewhat better results seen in figure 5.14 in comparison to those in figure 5.13 are an indication that the proposed color correction does improve classification. The implementation of color correction was attempted at a late stage in the work of the thesis, and as such, it was only used when assessing the change in performance of U-net when using the different image sets.

6.3 Weight estimation

The perspective correction used in the scope of this thesis did not account for the actual three-dimensional structure of the meal components. Instead it is assumed the objects in the images are planar, which is not the case for any of the food objects, but as can be seen from the figures 5.19 and 5.20, the areas found after perspective correction, as visualised in figure 4.1, the same meal components taken at different angles closely correspond to each other. This indicates a robustness of the method, at least concerning the five different classes of food used in the scope of this thesis. Because of the assumption of the meals being in the plane, the error is expected to be very much dependent on the distribution of a meal component, and as such, the 'flatter' the meals placed on the plate, the more accurate should the size estimation be. Conversely, the more meal components do not adhere to the same plane as the marker in the image, the results are expected to be more faulty. Since no segmentation method with satisfactory results were found, no conclusive tests were performed to evaluate the performance of the weight estimation.

The fitted polynomials fitted to the measured area of the meal components in figures 5.19 and 5.20 are of the first order. One might consider polynomials of higher orders, but having a linear regression was decided to be the most natural choice, given the apparent linear behavior of the data points extracted, overall.

6.4 Segmentation

6.4.1 Support Vectors Machines

It was clear that in its implementation during this thesis, the support vector machine was not suitable for the features chosen, RGB-values of pixels. The data presented in the images was probably not linearly separable, and as such the SVM was deemed to be unfavorable for the task at hand. Methods to linearise the features of the image for better classification were considered, but not implemented in favor of using more suitable machine learning algorithms for classification of multiple classes, such as decision trees.

6.4.2 Hard Mining - Random Forest

The author is stumped at the results of classification of sausage in figure 5.5, as the ROC-curve corresponding to it characterises complete guesswork. This erroneous result is possibly due to a fault in the implementation that the author is yet to make sense of. One would expect sausage to at least show better results than rice in classification, which it does not. The results for rice were in comparison to other classes (not taking the results for sausage into account) unsatisfying, as it does only yield its highest TPR while attaining a relatively high FPR as well, as is visualised in figure 5.7. These results were expected however, as the color, and perhaps the texture also to some extent, of rice is similar to other pixels in the image, such as the white plate or the white patches on the table cloth used. In the same manner, figure 5.6 shows the visual classification of haricots verts, which seems to hold true mostly, in accordance to the ROC-curve corresponding to haricots verts in figure 5.5.

In figures 5.8 and 5.9, all pixels are shown in a color corresponding to the class they were classified to be. In figure 5.8 it can be seen that many of the haricots verts pixels are wrongly classified as broccoli. This was not unexpected as the appearance, especially the color, is similar. In figure 5.9 it can be seen that many pixels belonging to rice are misclassified as background (plate, cloth, etc.), and vice versa. This is also not unexpected, because of the similarities in color.

6.4.3 Bag of visual words

The bag-of-visual-words approach have the advantage of classifying images in chunks of pixels, and not each pixel individually. This advantage is due to the SLIC algorithm which performs part of the segmentation, as discussed in section 3.1.4, as the partitioned superpixels follow the boundaries of the meal components fairly well. This can be seen in figures 5.11 and 5.12.

In figure 5.11, the same misclassification that occurred when using hard mining exist here also, that haricots verts is classified as broccoli (and also background). In the same way, figure 5.12 shows that much more rice is correctly classified in comparison than when using the hard mining method (figure 5.7). At the same time there can still be seen in figure 5.12 that misclassification between rice and background still occur.

The ROC-curves seen in figure 5.10, and in effect the corresponding AROC values, show an overall improvement to the pixelwise classification used in hard mining, whose ROC-curves

can be seen in figure 5.5.

The ablation test discussed in 4.3.3 resulted, over two tests made, that using 100 superpixels and a confidence threshold of 0.5 yielded the best results over both tests. The number of clusters in the feature space did not seem to matter as much for the results, but were among the lower values of the three alternatives. This number varied from 8 to 16 between the tests, and indicates maybe that the data in the feature space overlaps in such a way that using many centroids distorts the information yielded.

The author was surprised by the low classification rate of sausage, which have, seemingly, much more striking features than, for instance, rice. This might be mitigated by choosing a higher number of superpixel which the image is to be partitioned into, although the results in the ablation test, as discussed in section 4.3.3, speak against a higher number of superpixels. The choice of using a spatial pyramid approach for extraction of features might not haven been the best. The features used in the classification are dependent on the shape of the superpixel, whose corresponding bounding box might overlap to much with the background. This probably leads the algorithm to classify a superpixel as background as a whole, which might especially affect food of oblong and thin shape.

The method of using Bag-of-visual-words is rudimentary, but the standardised background of the images seem to improve the accuracy of the method. A shifting background might yield very different feature values for the same label, 'background' and cause inconclusive results.

As can be seen from figure 5.11, the faulty classification that occurs was usually not between meal components, except in the case between haricot verts and broccoli, but often things were mistakenly classified as background, or background was wrongly classified as meals. One explanation for this is that there was such a large amount of data related to background, that it might often overlap with other data points in the feature space, making it hard to find separate centroids which obscures the encoding of features. This could probably have been mitigated by reducing the number of background data when performing the method, so that the features corresponding to background are no more prevalent than that of the food classes.

6.4.4 U-net

It is worth noting that the number of epochs used in the training of the neural network, from which the results in section 5.2.4 stems, were considerably low, only 20 epochs. Accuracy might have increased with longer training sessions, although the assessment of the author was that the results were not expected to improve, as validation accuracy on the networks had been essentially the same over the last five to ten epochs.

It was attempted to achieve a segmentation which could differentiate between food classes, instead of just being able to locate pixels belonging to food. Since there were no conclusive results, those were omitted in this report.

Although one interesting results was that the area under the ROC-curves corresponding to a binary U-net classification was increased somewhat when training and testing on images which had been corrected by color, as is seen in figures 5.13 and 5.14. This strengthens the assumption that color correction, even when using suboptimal lighting conditions, can be used to achieve a better accuracy. Even with the slight increase in AROC value, the resulting images, altogether yielded by U-net, seem much poorer in classification performance than other methods assessed in this thesis, as seen in figures 5.15, 5.16, 5.17, 5.18.

For the results presented in section 5.2.4, no pretrained networks were used. The network used in the original paper [39] were not implemented. Even if that network had been used on medical images, it was still trained for semantic segmentation, and as such could have improved results. Using a pretrained network would have been particularly useful given the limited amount of training data present in the photo atlas. As such, the method was perhaps not given all the prerequisites to reach its full potential, as it has achieved before [39].

Chapter 7
Conclusion

One important objective of the thesis was to find suitable machine learning algorithms that can perform semantic segmentation of images based on small amounts of training data, for images of meals on a standardised background. From the acquired segmentation a quantity estimation could be yielded by comparison by a marker of known size, present in all images, and corrected for projective distortions.

A common factor for all the machine learning algorithms were chosen for their apparent characteristic of being able to achieve adequate results without excessive amounts of training data.

The ROC-curves and AROC values corresponding to the Bag-of-Visual-Words method show surprisingly decent results, as seen in figure 5.10, with high AROC-values for all food classes, even for rice, which was expected to be especially hard to classify. The author sees these results as an indication that superpixels can be used as an aid in algorithms for food classification. As it stands, the method is still prone to error, but probably performs well because of the standardised background in the images. It seems that the initial partitioning of an image into superpixels is helpful in the classification, as a mean of preprocessing, helping the semantic segmentation adhere to visible boundaries of meal components. Using a more sophisticated encoding method of the features from a superpixel, such as a neural network, might yield better results in future work.

Using neural networks to find a classification network took an substantially longer time to train than the other two methods evaluated. The results were with apparent flaws when performing classification between the multiple classes. The results achieved when training it for binary classification were still below what was achieved by other assessed methods. This was probably due to the implementation and further research on the subject might yield better results.

The images which had been color corrected, in accordance to section 3.1.2, were also used when training a U-net to see if a difference in classification accuracy could be found. As it stands, a slight increase in accuracy occurred when using images which had been color corrected, as seen in figures 5.13 and 5.14, where the area under the ROC-curves went from 0.8386

to 0.8971.

The author believes that even though the weight estimation treats all meal objects as two-dimensional, a quantitative measure is still found, and can supplement or confirm the results yielded in the 24h recall method, had a robust method for semantic segmentation been found in the thesis.

The results in this thesis show an indication that color corrected images, as presented in the report, can be used to improve results in classification, and that the use of superpixels can be used to simplify classification using machine learning algorithms on food, especially in the case of meal components adjacent to eachother. This could be a future project were some of the conclusions presented in the thesis could be of use.

Bibliography

[1] C. Alegre Y. Granfeldt C. Lazarte, M. Encinas. Validation of digital photographs, as a tool in 24-h recall, for the improvement of dietary assessment among rural populations in developing countries. 2012.

[2] . Photograph your meal with the foodvisor app for caloric, nutritional estimates. https://www.digitaltrends.com/mobile/foodvisor-calorie-counting-app/.

[3] L. Van Gool L. Bossard, M. Guillaumin. Food-101 – mining discriminative components with random forests. 2014.

[4] B. Koroušić Seljak S. Mezgec. Nutrinet: A deep learning food and drink image recognition system for dietary assessment. 2017.

[5] M. Bolanos P.Radeva E. Aguilar, B. Remeseiro. Grab, pay and eat: Semantic food detection for smart restaurants. 2017.

[6] S. Shevchik J. Dehais, M. Anthimopoulos. Two-view 3d reconstruction for food volume estimation. 2017.

[7] C. Liu E. J. Delp C. J. Boushey S. Fang, F. Zhu. Single-view food portion estimation based on geometric models. 2015.

[8] J. Li Y. Lianga. Deep learning-based food calorie estimation method in dietary assessment. 2018.

[9] Colorchecker: Colorimetric data. http://xritephoto.com/documents/literature/en/ColorData-1p_EN.pdf.

[10] D. Pascale. A review in color spaces ...from xyy to r'g'b', 2002. http://www.babelcolor.com/index_htm_files/A%20review%20of%20RGB%20color%20spaces.pdf.

[11] G. Hoffman. Cie color space. http://docs-hoffmann.de/ciexyz29082000.pdf.

[12] G. Hoffman. http://docs-hoffmann.de/ciexyz29082000.pdf.

[13] A. Hurlbert G.D. Finlayson, M. Mackiewicz. Color correction using root-polynomial regression. 2015.

[14] T. Tuytelaars L. Van Gool H. Bay, A. Ess. Speeded-up robust features (surf). https://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf.

[15] K. Smith A. Lucchi P. Fua S. Susstrunk R. Achanta, A. Shaji. Slic superpixels compared to state-of-the-art superpixel methods. 2012.

[16] M. Oskarsson. Lecture notes: computer vision (fman85, lth), January 2018. http://www.maths.lth.se/media11/FMAN85/2018/forelas1_4Xs3EDp.pdf.

[17] tex.stackexchange.com. https://tex.stackexchange.com/questions/96074/more-elegant-way-to-achieve-this-same-camera-perspective-projec

[18] M. Oskarsson. Lecture notes: computer vision (fman85, lth), January 2018. http://www.maths.lth.se/media11/FMAN85/2018/forelas3.pdf.

[19] Z. Zhang. A flexible new technique for camera calibration. 2000.

[20] tex.stackexchange.com. https://tex.stackexchange.com/questions/319172/gimp-like-perspective-transform-in-tikz.

[21] M. Oskarsson. Lecture notes: computer vision (fman85, lth), January 2018. http://www.maths.lth.se/media11/FMAN85/2018/forelas2.pdf.

[22] tex.stackexchange.com. https://tex.stackexchange.com/questions/20267/how-to-construct-a-confusion-matrix-in-latex.

[23] H. Cherifi V. Labatut. Accuracy measures for the comparison of classifiers. 2011.

[24] tex.stackexchange.com. https://tex.stackexchange.com/questions/464887/drawing-roc-curve-analysis.

[25] www.tuicool.com. https://www.tuicool.com/articles/6Rn2uq,.

[26] M. Pontil T. Evgeniou. Workshop on support vector machines: Theory and applications.

[27] R. Berwick. An idiot's guide to support vector machines (svms). http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.

[28] J. Friedman T. Hastie, R. Tibshirani. The elements of statistical learning: Data mining, inference, prediction. 2005.

[29] tex.stackexchange.com. https://tex.stackexchange.com/questions/251183/adding-triangles-to-latex-decision-trees.

[30] O. Maimon L. Rokach. Data mining and knowledge discovery handbook. http://www.ise.bgu.ac.il/faculty/liorr/hbchap9.pdf.

[31] L.Breiman. https://www.stat.berkeley.edu/ breiman/randomforest2001.pdf.

[32] N.-C. Overgaard. Lecture notes: Medical image analysis (fman85, lth), December 2015. http://www.maths.lth.se/media11/FMAN30/2015/Lecture12-Clustering-and-segmentation.pdf.

[33] tex.stackexchange.com. https://tex.stackexchange.com/questions/243494/i-am-trying-to-draw-a-clustering-euclidean-diagram.

[34] A. Y. Ng D. Coates. Learning feature representations with k-means. 2012.

[35] J. Ponce S. Lazebnik, C. Schmid. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. 2006.

[36] tex.stackexchange.com. https://tex.stackexchange.com/questions/140782/drawing-a-neural-network-architecture.

[37] M. Ohlsson. Lecture notes: Fytn14 theoretical physics: Introduction to artificial neural networks and deep learning, November 2018. https://liveatlund.lu.se/departments/theoreticalPhysics/FYTN14/FYTN14_2018HT__-99____/CourseDocuments/Chapt_FeedForward.pdf.

[38] github.com. https://github.com/MartinThoma/LaTeX-examples/tree/master/tikz/max-pooling.

[39] T. Brox O. Ronneberger, P. Fischer. U-net: Convolutional networks for biomedical image segmentation. 2015.

[40] H. Iqbal, December 2018. https://github.com/HarisIqbal88/PlotNeuralNet/blob/master/examples/Unet_Ushape/Unet_ushape.tex.

[41] mathworks.com. https://se.mathworks.com/help/vision/ref/estimategeometrictransform.html.

[42] mathworks.com. https://se.mathworks.com/help/images/ref/superpixels.html,.

[43] mathworks.com. https://se.mathworks.com/help/stats/knnsearch.html.

[44] mathworks.com. https://se.mathworks.com/help/vision/ref/unetlayers.html.

[45] D. Franco. https://se.mathworks.com/matlabcentral/fileexchange/66663-elastic-distortion-transformation-on-an-image.

[46] mathworks.com. https://se.mathworks.com/help/matlab/ref/polyfit.html.