Master's thesis

Master's Programme in Computer Science

Camera-based food identification and weight estimation in a buffet-style restaurant

Teemu Sarapisto

October 24, 2022

Faculty of Science

University of Helsinki

Supervisor(s)

Assoc. Prof. Arto Klami, M.Sc Chang Rajani

Examiner(s)

Assoc. Prof. Arto Klami, Ph.D Pierre-Alexandre Murena

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Teemu Sarapisto

Työn nimi — Arbetets titel — Title

Camera-based food identification and weight estimation in a buffet-style restaurant

Ohjaajat — Handledare — Supervisors

Assoc. Prof. Arto Klami, M.Sc Chang Rajani

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	October 24, 2022	66 pages

Tiivistelmä — Referat — Abstract

In this thesis we investigate the feasibility of machine learning methods for estimating the type and the weight of individual food items from images taken of customers' plates at a buffet-style restaurant. The images were collected in collaboration with the University of Turku and Flavoria, a public lunch-line restaurant, where a camera was mounted above the cashier to automatically take a photo of the foods chosen by the customer when they went to pay. For each image, an existing system of scales at the restaurant provided the weights for each individual food item.

We describe suitable model architectures and training setups for the weight estimation and food identification tasks and explain the models' theoretical background. Furthermore we propose and compare two methods for utilizing a restaurant's daily menu information for improving model performance in both tasks. We show that the models perform well in comparison to baseline methods and reach accuracy on par with other similar work.

Additionally, as the images were captured automatically, in some of the images the food was occluded or blurry, or the image contained sensitive customer information. To address this we present computer vision techniques for preprocessing and filtering the images. We publish the dataset containing the preprocessed images along with the corresponding individual food weights for use in future research.

The main results of the project have been published as a peer-reviewed article in the International Conference in Pattern Recognition Systems 2022. The article received the best paper award of the conference.

ACM Computing Classification System (CCS)
Computing methodologies → Machine learning → Machine learning approaches → Neural networks

Avainsanat — Nyckelord — Keywords

food classification, food weight estimation, convolutional neural networks, computer vision

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Contents

1 Introduction

In buffet-style restaurants, having access to information about how much of each available food the customers take can be useful both for the restaurant and its customers. For example, restaurants could study if customers taking more of a certain food have higher customer satisfaction or throw more food away [20]. The customers, on the other hand, may be interested in keeping track of the nutritional details of their meals for trying to living healthier [41] or to lose weight [27]. In fast-food restaurants and others selling fixed-size or prepackaged food this information is relatively easy to keep track of, since nutritional details can be predetermined for each food. For self-service buffet-style restaurants it is much more difficult to keep track of the nutritional composition of the customers' meals, since the customers are free to compose their meal from a variety of available foods, and choose how much they take.

A system for estimating how much of each available food a customer takes is in use for example at the self-service lunch-line restaurant Flavoria located in Turku, where a weight measurement scale has been installed next to each type of food being served, and the customers measure the weight of each food type they take themselves right after adding it to their plate [20]. Based on the measured weights, the nutritional contents of the meals are provided to the customers via a mobile application. While in principle this system is sufficient for collecting information about the weights of each meal component and the nutritional contents of each meal, in practice this system has many shortcomings. For example, the scales can be expensive, may need frequent recalibration, and can be cumbersome to use as the whole plate needs to be re-weighted every time a new type of food is added. Furthermore the scales are error-prone: the customers may either accidentally lean on the scales or add other objects than food on them while measurement is in progress.

In this thesis we investigate the use of computer vision and machine learning methods for identifying and estimating the weight of each individual food item on a plate based on an image. We apply the methods on a new dataset collected in collaboration with Flavoria containing both images of the restaurant's customers' meals as well as the corresponding weights for each food item in the meal. The images were collected for this project by mounting a camera system above the cashier, while the automated weight measurement system was already in place and is described by Mäkinen et al. [20]. Furthermore we

publish the preprocessed dataset for everyone to use*.

Many recent studies in food computing are also based on machine learning as evident from their prominence in a recent survey [30]. A more closely related review of image-based food identification and weight estimation methods by Lo et al. [26] shows that these two tasks are particularly dominated by machine learning approaches. While many of the works report promising results on the food identification task, weight estimation is still considered a difficult problem [30] [26]. Furthermore, most methods presented in previous work on weight estimation are either hindered by requirements of fiducial markers such as rice [7] or checkerboards [13] to be visible in the images, or do not fit our problem setting as they are limited to work with fixed weight foods [31, 13]. The closest work to ours is the parallel work by Thames et al. [40]. However, they only predict weight for the whole meal instead of its individual components limiting their method's usefulness from the restaurant's perspective.

We begin by explaining the theoretical background of the computer vision and machine learning techniques applied in this thesis in Chapter 2. We cover how circles can be found from an image with rule-based computer vision techniques, and how machine learning and neural networks can be used in image-based regression and classification tasks. We also cover deep learning techniques such as batch normalization, transfer learning, and dataset augmentation, that are known to increase the performance of most neural network models.

In Chapter 3 we describe the specifics of our problem scenario. We present how the image and weight data was collected, and give key statistics of the dataset. We also describe automatic preprocessing techniques for both kinds of data.

Next, in Chapter 4 we describe our method for developing an efficient model for the two tasks. As a new contribution specific to the restaurant setting, we introduce two techniques for incorporating menu information. We present a series of experiments designed to provide empirical evidence that the choices made in model development were justified and improve the model in the target tasks. We also compare the results to relevant baselines in order to confirm that the machine learning models reach better performance than naive implementations.

Lastly we discuss the results of a final version of our model and compare our results to other work. We also further elaborate how the performance varies between different food types and how it relates to the number of training samples available for each food.

*Dataset available at https://zenodo.org/record/5850856

The main results of the study have already been published as a peer-reviewed article in the International Conference on Pattern Recognition Systems 2022 [35] where the article received the best paper award of the conference. This thesis extends the article in many aspects: it adds a more in-depth background section, introduces a new model combining classification and regression, and provides a more thorough experimental section with an ablation study and details such as convergence graphs.

The camera system used to capture the images was built by Mikko Toivonen, and set up at Flavoria, Turku by Ville-Veikko Saari and Lauri Mäkinen. Based on signals sent by the system at Flavoria, Ville-Veikko programmed the automatic capture of images, created JPEGs from the original raw images, selected camera settings such as the exposure time, and helped to manually remove images with sensitive data. Lauri Mäkinen helped in making sure the dataset was ready for release, and then packaged and published the dataset. While all the weight preprocessing detailed in this thesis was conceived and implemented by the author, the Flavoria system may do some preliminary preprocessing eg. to filter or discard completely insensible data. The weight collection system and the weights were provided entirely by Flavoria.

The author was responsible for all computer vision and machine learning related development presented in this thesis including coming up with, training, evaluating, and implementing the models with the PyTorch library [33]. Furthermore the author developed the computer vision based automatic image cropping and filtering tool based on the circle Hough transform function of the OpenCV library [2].

2 Background

The computer vision methods applied in this thesis can be divided into two categories. The first category are *rule-based* computer vision methods where all parameters are predetermined or selected manually, while the second category are adaptive or *machine learning* methods where the parameters are selected based on data.

In Section 2.1 we present how circles of specific size can be found from an image with a collection of rule-based computer vision techniques. This is applied in Section 3.2.2 for determining whether an image contains a plate, and if so where it is in the image.

The rest of the chapter focuses on adaptive methods. The state-of-the-art of adaptive methods for image data is largely dominated by neural network based solutions [22] and so they are the main focus of this thesis. Even though neural networks are currently a very hot topic by themselves, they are still a natural part of and share a lot with the broader category of data-based adaptive methods collectively referred to as machine learning [11]. Hence we cover the basics of machine learning in Section 2.2 as it works as a general introduction to the topic, while neural network specific details are covered in Section 2.3.

2.1 Locating circles in an image

In this section we describe how circles of specific size can be found from an image in a heavily constrained imaging setting with a collection of classical computer vision techniques. In Chapter 3.2.2 we show how finding circles can be applied for preprocessing food images by determining whether an image contains a plate of given size, and if so where the plate is in the image.

Many computer vision algorithms deal with the challenges posed by complex illumination and the inverse problems inherent in learning 3D information from 2D images [39]. However as further described in Chapter 3.1 , all the images in the dataset used in this thesis have been taken from a fixed distance where plates of identical size and shape lie on a flat surface (the cashier desk) almost perfectly perpendicular to the camera. This considerably simplifies the problem of finding the plates, since when viewed through the camera the plates are well approximated by circles of equivalent size, and so the solution only needs

to be robust to changes in lighting.

We begin with a description of the convolution operation that is the basis of many computer vision systems and show how it can be applied to edge detection [3]. Then we discuss how the circle specific version of the generalized Hough transform [6] can be used on the results of edge detection to find circles and thus food plates.

2.1.1 Convolution

In mathematics convolution refers to an operation where the integral of the product of two functions is taken. However computer vision is mostly concerned with discrete functions on discrete data such as image pixels, where the (2D) convolution is customarily used to refer to the operation

$$g(i,j) = \sum_{k,l} I(i+k, j+l)F(k,l)$$

where F is a two dimensional filter matrix (also called the kernel) and I is a two dimensional data matrix [39]. Intuitively, the operation can thought of as "sliding" the filter "window" over the data matrix, where on each step an element-wise product between overlapping parts of the two matrices is taken and the resulting values are summed.

Strictly speaking this is discrete cross-correlation, while convolution is otherwise the same operation, except that the kernel is reversed, but we adopt the same naming style as in other machine learning literature and refer to the operation as convolution [11]. Convolution has the practical effect of summing together the local neighborhoods of each pixel in a target image as weighted by the kernel. Convolution is easiest to understand through image examples, see the visualization in Figure 2.1.

The simplest useful example of 2D convolution is to place in each cell F_{ij} of the filter matrix the same value $\frac{1}{N}$ where N is the number of total cells in the matrix. Convolving an image with this matrix has the effect of doing a *Box blur*. If instead we fill the kernel with the point density values of a 2D gaussian, a *Gaussian blur* is achieved. Gaussian blur is a form of low-pass filtering the image, as high frequency details are removed, and is often used as a preprocessing step in classical computer vision tasks such as edge detection [39]. Convolution is also widely used in machine learning based computer vision [11]. Both these topics are covered in the following chapters.

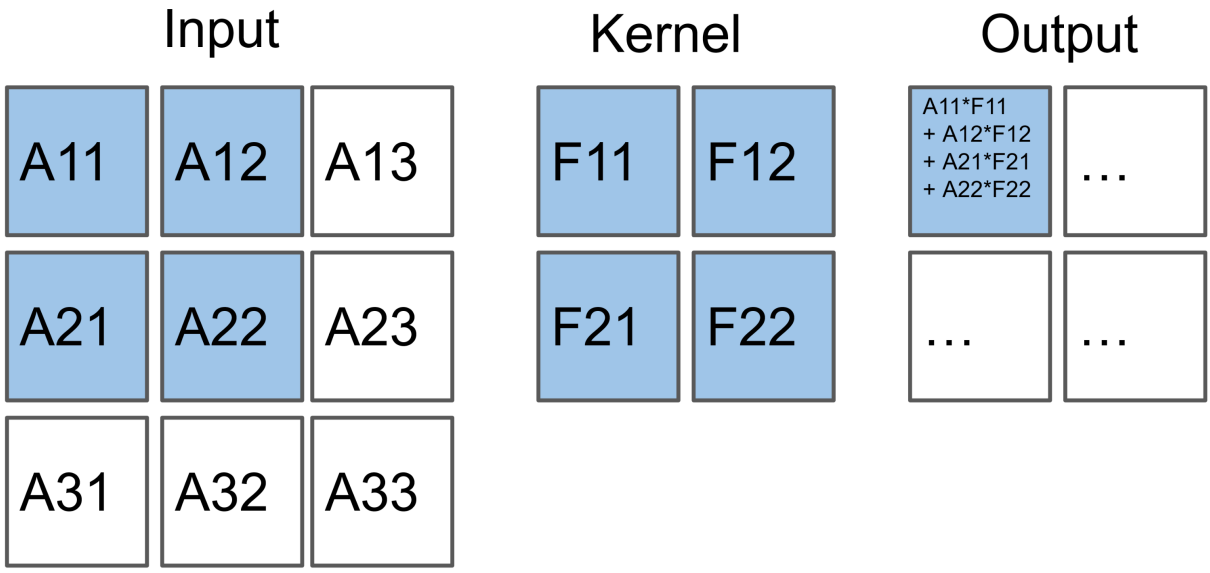

Figure 2.1: Illustration of 2D convolution of a 3×3 input with a 2×2 kernel. This is a *valid* convolution without padding, and so the output is 2×2.

2.1.2 Edge detectors

Edges can be defined as points where the intensity of the image changes rapidly, and so they can be localized based on the gradient of the image intensity [39]. Based on this kind of characterization John Canny provided a mathematical formalization for an edge detector that is optimal with respect to the three criteria of low error rate, good localization ability, and that a single edge should only produce one edge response, assuming the edges have been corrupted by some amount of Gaussian noise[3].

An efficient approximation of the optimal Canny edge detector can be implemented with a four step algorithm performed on black-and-white version of an image. As the first step the image is blurred with Gaussian blur to remove high frequency noise. The second step creates edge proposals based on the intensity gradient's direction and magnitude. The third step uses non-maximum suppression to remove multiple responses to the same edge that may occur due to thick lines, and finally as the fourth step hysteresis is applied on the found edges to discard weak edges while keeping strong edges connected where possible. We now go through the steps and reasoning behind them with more detail.

As images from any practical imaging sensor tend to have high frequency noise, most edge detection algorithms begin with low-pass filtering using the Gaussian filter [39]. The Gaussian filter is usually used for images due to it being the only circularly symmetric

filter that is also separable [39]. This means its response is the same in all directions, and that it is separable into one 1D convolution per dimension making it more efficient to compute. For example given a non-separable 2D kernel of size $K \times K$ each of the K^2 values of the kernel need to be multiplied with a target image pixel and summed in a normal 2D convolution, while in the separable case it suffices to do two 1D convolutions of kernel size K leading to only $2K$ kernel values being applied to each pixel of the image.

One way to find the gradients of image intensity is to convolve the image with kernels $P_x = [-1, 1]$ and $P_y = P_x^T$, or with the Sobel filter [37] where both essentially perform the differentiation in the style of finite differences. Differentiating once per both axis, x and y

$$\sqrt{(P_y * I)^2 + (P_x * I)^2},$$

suffices to find the slope and magnitude of the image [3].

However due to both differentiation and convolution being linear operations

$$\nabla(G * I) = \nabla(G) * I$$

where G represents the Gaussian kernel in either axis (as it can be done individually for both due to separability), and ∇ refers to taking the derivative for each dimension. This means we can calculate the gradient and perform Gaussian blur in a single step by convolving the image with a kernel corresponding to the first derivative of the gaussian. In addition to being efficient, this method is also reasonably accurate; Canny showed that that the first derivative of the Gaussian is a very good approximation of the optimal edge detection operator [3].

These steps often result in multiple responses to edges, and so *non-maximum suppression* is applied. For each pixel resulting from the algorithm so far, we compare the values of pixels in both positive and negative directions of the two axes, and their diagonals in both directions, and filter out pixels not having the maximum intensity in all the eight directions. This retains the strongest local response while removing weaker ones, thus creating lines that are one pixel wide.

Finally as the last step, *hysteresis* is applied as shown in Figure 2.2, where the intensities of pixels forming continuous edges are compared against two thresholds, `High` and `Low`, giving hysteresis its alternative name *double thresholding*. Here pixels under both thresholds are never kept, pixels above both thresholds are always kept, and pixels between the two thresholds are kept if they are part of an edge that at some point reaches the `High` threshold without being connected by pixels under the `Low` threshold.

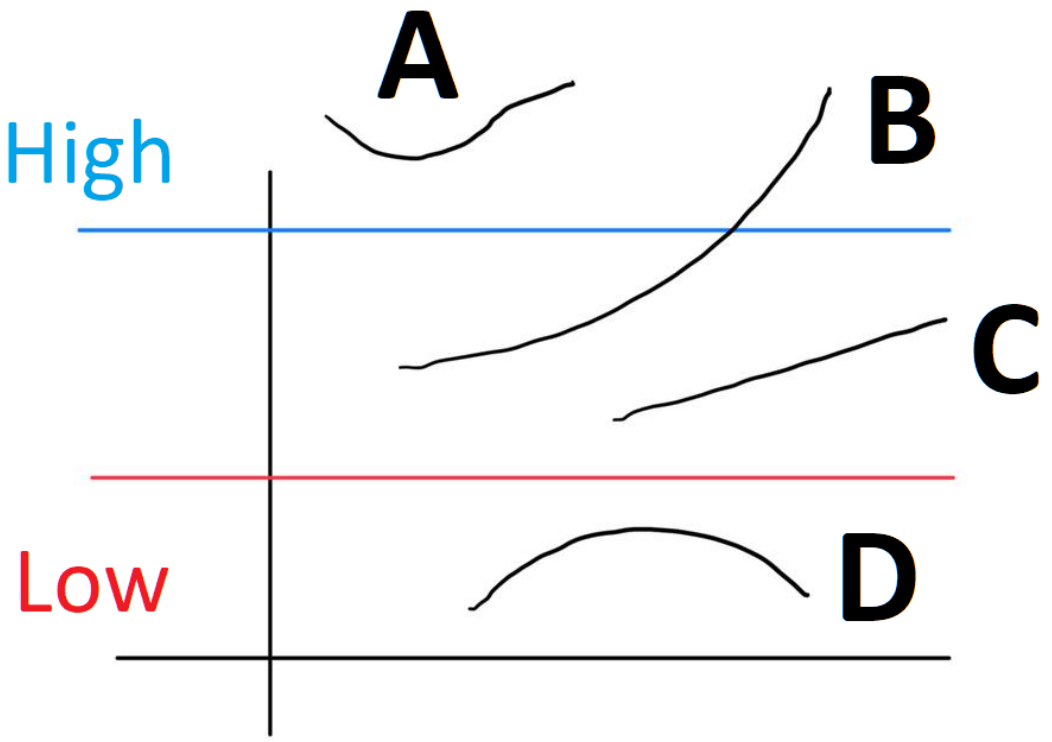

Figure 2.2: Hysteresis. All pixels of lines A and B are kept as they are connected to an edge reaching above the `High threshold`, while all pixels of C and D are discarded.

The so called Canny edge detector consisting of these four parts is very commonly used to this day as a part of further processing [39].

2.1.3 Hough transform

The transform was initially published by Hough for use in detecting straight lines from bubble chambers [16], with the improved generalized Hough transform by Duda and Hart [6] making the transform applicable for detecting arbitrary parameterizable curves. Since we are only interested in detecting circles, we focus on the version of the generalized Hough transform sometimes referred to as the *circle Hough transform* [6].

The algorithm is usually applied on a a binary edge image with a value of 1 indicating the pixel belongs to an edge, and being 0 otherwise. Here the edges are ideally one pixel wide. This edge image can be obtained for example with the Canny edge detector defined in the previous section.

Given the binary edge image $B \in \{0,1\}^{W \times H}$ we create an *accumulator* image $A \in \mathcal{N}^{W \times H}$

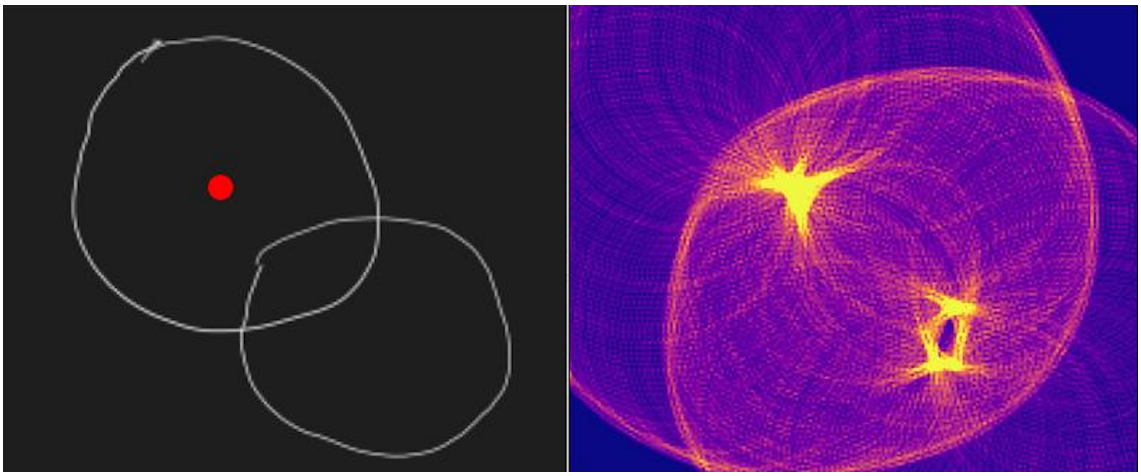

Figure 2.3: Left side is an image of two hand drawn circles of slightly different size, where the red dot is located at the position of the maximum value of the accumulator image on the right. Right side is the circle Hough transform accumulator image for a single radius r for the image on the left. There are two bright areas in the right image roughly corresponding to the centers of the two circles. As r is larger than the diameter of the circle on lower right, the accumulator forms a circle shaped bright area near the smaller circle's center but doesn't reach a single high value there thus leading to the accumulator maxima being on the more condensed top left side.

of the same shape where all values are initialized as zero. Intuitively for every positive pixel $b = 1$ at some coordinates c_x and c_y in the edge image B, all pixels $a \in A$ lying on the edge of a circle centered at (c_x, c_y) with a radius of r pixels are incremented by one.

We can compute the coordinates of each point on the circle from the circle equation as

$$x = c_x + r\cos\theta$$

$$y = c_y + r\sin\theta$$

by going through values of $\theta \in [0, 2\pi]$.

This results in an accumulator image as seen in Figure 2.3. The local maxima of the accumulator image can be used as the centroids of circles. The number of circles detected can be varied by changing the threshold of minimum accumulator value for being counted as a circle center.

2.2 Machine learning

Hand-crafting detectors even for exceptionally distinguishable foods is extremely laborious and error prone as the number of possible foods rises to more than a handful. This

motivates the use of adaptive methods that use data to automatically figure out what the important features are and how to extract them for solving a specific well-defined task.

Machine learning research is specifically concerned with finding efficient methods for solving problems by utilizing data without explicitly having to program the algorithm [1]. The word learning in machine learning highlights the fact that the parameters of machine learning models are *learnt* based on data, where the process of learning parameters is often called training. Specifically, the two types of problems we are trying to solve, recognizing the type of food and its weight, can be cast as machine learning problems of classification and regression [1].

The classification problem arises from the question "what foods are in this image?", as we want to predict which foods or *classes* out of a predefined and finite set are present in a particular input image. Furthermore, it is an instance of a task called *multi-label* classification, as there can be many different foods in the images in our dataset. This differs from multi-class classification where there can only be one class present at a time out of many possible classes.

To answer the other type of question "how much do the foods in this image weight?" we wish to find a function that maps inputs from the space of possible food images to the continuous space of real-valued and non-negative weights of food. This can be thought of as a type of regression analysis from the field of statistics, as we are modeling how a continuous dependent variable (the weights) is related to explanatory variables (patterns in the images) often also called *features*.

The selection of the objective function and the methods for improving the model's performance as measured by the objective function are highly dependent on the problem setting. As classification and regression are usually pursued in a specific scenario called supervised learning, the rest of the section is devoted to setting up the supervised learning framework and how models can be improved in it in a process called optimization.

We start by describing the basics of machine learning and optimization before moving on to the more specific methods in neural networks. We describe how neural networks are trained, and what fully connected and convolutional layers do and what are their differences. Finally we cover some techniques for improving the performance of deep networks trained on limited data.

2.2.1 Supervised learning

In this thesis we focus on a specific type of machine learning called supervised learning, where for all inputs x we know the expected outputs y, often called the ground truth or "labels". As an oversimplification the whole mission is then to find the function that maps the inputs to the outputs.

This is in contrast to unsupervised learning where there are no target labels, and instead the focus can for example be in exposing structures in the data or grouping them into clusters. Semi-supervised learning exists between the two opposites, where for some inputs there are labels but not for all of them. Both semi-supervised and unsupervised methods are not covered further in this thesis.

The difficulty in supervised learning arises from the fact that there are only a few examples of the true mapping between the inputs and the outputs in any practical training dataset compared to the space of all possible pairs of inputs and outputs. This means we only have a very limited (and always noisy) estimate of the true mapping to learn from making it paramount that the model learns to generalize to unseen inputs.

As we cannot measure the *generalization ability* with the same data that was already used for learning the model parameters, the dataset is usually split into multiple parts, including at least separate training and test sets. The training set is named so because creating a model based on data is often called training the model. The test set is never used to train the model and hence can be used to estimate the model's performance for unseen data.

Sufficiently large and complex models can learn the details of the training dataset in too fine detail ending up also modelling the noise that is inherent in any practical dataset. This usually leads to a problem called *overfitting*, where the model performs excellently for data from the training set, while failing miserably for the test set. This can be seen as the model having failed to generalize and learn the important patterns of the data.

To combat overfitting, regularization techniques are often utilized. These can include adding extra terms to the objective function that encourage specific types of solutions more resistant to overfitting, or in the case of using an iterative training process, simply stopping the training before the model starts to overfit. In our experiments overfitting is mostly addressed by two methods, early stopping (covered in Section 2.3.4) where the training is stopped before it starts to overfit, and by the use of convolutional networks that are inherently resistant to overfitting as covered in detail in Section 2.3.3.

2.2.2 Optimization

Optimization in mathematics is the pursuit of finding better parameters for a model or a function in terms of an objective function [4]. In the quintessential optimization scenario a function is either minimized or maximized so we cast the training of our machine learning models as a problem of minimizing a function. In the scenario where model parameters are chosen by minimizing a function, the objective function is generally called a loss function, and the process of minimizing is referred to as minimizing the loss.

We can represent both classification and regression models as a function $f(x|\theta)$ with parameters θ that maps the inputs x to predicted outputs $\hat{y}$. A prediction can thus be represented as an equation

$$\hat{y} = f(x|\theta).$$

Since we are in the supervised learning setting and have labels y, in the simplest case we can just use some distance measure between predictions and labels as the loss function to be minimized. An example of an often used loss function $\mathcal{L}$ is the squared difference of a predicted value and the label value

$$\mathcal{L}(x, y, \theta) = (y - f(x|\theta))^2 = (y - \hat{y})^2$$

where the optimization task is to find some parameters θ for which the prediction $\hat{y}$ is as close as possible to y. This corresponds to a minima of the loss function wrt. θ.

In addition to the direct prediction error, the loss sometimes has extra regularization terms that may help avoid overfitting. Two often used regularization terms are the sum of absolute values or the sum of squared values of all parameters. Regularizing with the sum of absolute values encourages sparsity in the parameters (many parameters tend to be zeros) while the sum of squared parameter values encourages parameter values with many small values instead of a few large ones [18].

For many complex machine learning models such as the ones we'll be covering in this thesis, the optimization problem

$$\arg\min_{\theta}(y - f(x|\theta))$$

does not have a closed-form solution for selecting the optimal values of θ. This means that finding the globally best parameters (global minimum) is difficult (or impossible) and

that optimization must be performed with numerical methods. By selecting the difference measure, regularization terms, and the model so that the loss function is differentiable and continuous (almost) everywhere with respect to the model parameters, we can use gradient based methods for optimization.

Gradient descent

Having a differentiable loss function wrt. model parameters makes it possible to compute the gradient of the parameters. Gradient describes the direction where the function increases the fastest wrt. its parameters, and as we want to decrease the loss by modifying the parameters, we move the parameter values in the direction of the negative gradient.

Starting with randomly initialized parameters θ_0, the new parameters θ_t at step t are iteratively computed based on the last step's parameters θ_{t-1} with

$$\theta_t = \theta_{t-1} - \eta \nabla_\theta \mathcal{L}$$

where η is the *learning rate* (LR), and ∇ is the gradient operator. Finding a proper learning rate is crucial, as it sets *how far* we move in the loss function space as the gradient or partial derivatives only contain information about the steepness and direction of the function at the particular point, and no information about distance or the local neighbourhood.

The learning rate (LR) is considered a *hyperparameter*, because it is not selected in the training process but is still a parameter. The LR is usually selected empirically based on observing the behavior of the loss during the first few iterations of the training process, where it is monitored that the loss has a clearly decreasing trend. Having a much too large learning rate will usually result in instability that often results in some values reaching values near infinity resulting in a crash due to lack of numerical precision. A slightly too large LR often leads to oscillatory behavior resulting in the iterative process never reaching a local optima or *converging*. Too small LR on the other hand takes more iterative steps to converge than needed and is thus undesirable.

Usually in the machine learning context we have many training samples so the loss is computed as a mean over the training samples

$$\frac{1}{N} \sum_{i}^{N} \mathcal{L}(x_i, \theta)$$

and the gradient is computed relative to the total loss.

However plain GD is relatively weak at finding good solutions as it tends to explore a relatively small area of the loss surface. GD with equivalent LR η, model parameters θ, and training set X is deterministic and will always converge to the same local optimum. Furthermore, computing a full GD step for a massive dataset can be extremely computationally expensive. To help the training process consider a wider group of possible solutions a more explorative version of GD called *stochastic gradient descent* (SGD) is often used.

In SGD the loss and the gradient are computed for one randomly chosen sample $x_i \in X$ at a time, helping the optimization process cover more area. One sample SGD is often too extreme in its volatility. As a useful middle ground that provides some stochasticity while still being relatively stable, the so called mini-batch SGD is commonly used. In mini-batch SGD the loss and gradient are averaged over a group or *batch* of samples at a time.

SGD is a relatively simple optimization method. Many proposed extensions to SGD exist, with one of the most popular being ADAM [19]. However SGD generalizes better in many scenarios than its extensions [43] and is thus still widely used.

In summary, when using gradient based optimization, the training regime of a machine learning model consists of over predicting some output based on an input, measuring the difference or error compared to the ground truth, computing the partial derivatives of all parameters wrt. the loss, and finally changing the parameters towards the direction where loss decreases. The (stochastic) gradient descent based methods offer no guarantees for convergence to a good solution, and can get stuck in poor local optima.

2.2.3 Task-specific loss functions

As mentioned in the previous section, a loss function is used to cast the training of a machine learning model into an optimization problem where the loss function is minimized. This implies that after successful training the model should have learnt to create predictions with low loss, further implying that the selection of a loss function that properly fits a given task is important as the loss function essentially dictates what the model learns to do during training.

For regression tasks characterizing the loss is relatively simple, as a low squared error or a low absolute error between the prediction and the ground truth value often suffices. For

a SGD mini-batch of N samples, we can then optimize for example for the mean absolute error (MAE)

$$\sum_{i}^{N} |\hat{y} - y|$$

or the mean squared error (MSE)

$$\sum_{i}^{N} (\hat{y} - y)^2$$

where both simply take the mean over the individual prediction errors.

For classification the problem is usually formulated so that the model output is considered to be an estimate of the probability that a given sample contains some class. As many statistical and machine learning models by themselves usually predict unnormalized values, the model outputs are often forced to lie in the interval $(0, 1)$ with a suitable function depending on the classification setting.

For single class classification where the number of classes $C = 2$, for example the standard logistic functions

$$f(x) = \frac{1}{1 + e^{-x}}$$

where $f(x) \in (0, 1)$, is a common choice.

Deriving a classification loss

Minimizing the cross-entropy between two distributions is equal to minimizing the difference of the two distributions [11]. We can use this definition to come up with a loss function for binary classification.

The definition of cross-entropy is

$$H(p, q) = -E_p \log(q)$$

where E_p is the expected value operator with respect to distribution p. Based on the definition of the expected value of a discrete random variable, we can represent cross-entropy as a sum over the i possible outcomes

$$H(p, q) = -E_p \log(q) = -\sum_{i} p_i \log q_i.$$

Since for binary classification there are two possible outcomes, if we treat the predictions and ground truths as distributions and replace the p_i in this equation with model predictions $y \in (0, 1)$ and q_is with ground truth labels $\hat{y} \in \{0, 1\}$ so that y is the predicted probability of the positive class and $1 - y$ is the predicted probability of the negative class proceeding similarly for the ground truth label, we get

$$H(p, q) = -\sum_i p_i \log q_i = -[y \log \hat{y}] - [(1 - y) \log(1 - \hat{y})])$$

where the both outcomes have been surrounded with brackets, which is also known as binary cross-entropy (BCE).

Furthermore for N samples, we can take the average of this over the samples to get a scalar-valued loss function

$$\mathcal{L} = \frac{1}{N} \sum_n^N H(p_n, q_n)$$

to be minimized.

Since in this thesis there can be many foods in each image, the problem is called multi-label classification. Here we can form the loss by taking the average of C binary classifications for each of the N images

$$\mathcal{L} = \frac{1}{NC} \sum_n^N \sum_c^C H(p_{nc}, q_{nc})$$

where the subscript refers to the cth class for the nth sample.

2.3 Neural networks and deep learning

Neural networks have a long history in machine learning. Research into them can be traced back at least to the 1943 paper by McCulloch and Pitts [28] where they presented the first type of artificial neuron called the perceptron that attempted to (loosely) model how a biological neuron works. On the other hand neural networks can be viewed as a natural part of centuries long research into universal function approximators [11, ch. 6.6].

Neural networks are constructed by composing mathematical transformations or functions grouped as layers [11]. In order to obtain outputs of the network, the layers are applied on the input data either sequentially, recursively, or in parallel depending on the model type, the model architecture, and whether it is being used in a distributed fashion. Research into networks with particularly many layers is often referred to as *deep learning* as the

number of layers can be thought to be the depth of the network and for more depth to enable learning higher order and possibly more abstract representations of the inputs [22].

From the perspective of this thesis, neural networks are an *end-to-end* method for learning both the feature extractors as well as the classifier/regressor based on those features in one unified iterative process. This unification helps the model learn the specific feature extractors needed for efficiently solving the task at hand.

Moving on to more practical terms, in this thesis we focus on *feedforward networks*. A feedforward network has a fixed number of layers applied in succession where each layer consists of a linear transformation (eg. matrix multiplication or convolution) that is usually followed by a non-linear element-wise function called an activation function.

To create a prediction from some input data with a feedforward network, the first layer of the network is applied on the input data itself, and each following layer consecutively receives as input the output of the previous layer where the output of the last layer is considered the output of the network. Here the layers between the input and output layers are referred to as *hidden layers*. Other network types than feedforward exist, such as the recurrent neural network (RNN) where some layers can be applied multiple times [11].

We start by describing a basic network based on the simplest layer type, the fully-connected (FC) layer, and use it to introduce network training essentials such as what are layer weights and how to initialize them, how to train the network efficiently, and what non-linear activation functions are commonly used in the layers. Finally we describe the weaknesses of FC layers, show how convolutional layers work, and how convolutional layers address many of the challenges FC layers have dealing with 2D image data.

2.3.1 Fully-connected layers

A fully-connected feedforward network (sometimes also called a multilayer perceptron) is the simplest type of network, and can conceptually be represented as a graph as in Figure 2.4. In practice the transformation represented by a single fully-connected layer is usually implemented as a 2D matrix multiplication and some non-linear element-wise function on the result

$$y = g(Wx)$$

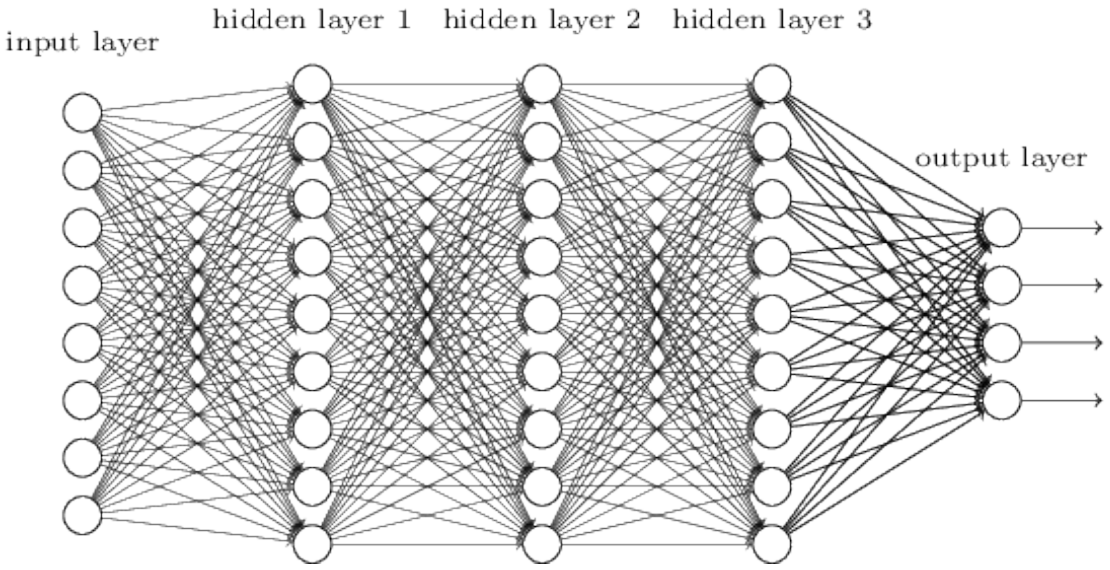

Figure 2.4: Conceptual visualization of a fully connected feedforward network. Input layer is just the input data, the layers between the input and output are sometimes called hidden layers, and the output of the network can be read from the output layer. The layers are fully-connected, as each unit is connected to all units of the previous layer. Image from [32]

where $W \in \mathcal{R}^{E \times D}$ is a learned weight matrix, $x \in \mathcal{R}^{D \times 1}$ is an input vector, $y \in \mathcal{R}^{E \times 1}$ is the output vector, and g is the element-wise function often called an activation function. As this transformation changes the dimension of the input vector from D to E, a single layer is able to transform its inputs into an output vector of arbitrary dimension.

A bias term $b \in \mathcal{R}^{E \times 1}$ is sometimes added to the result of the matrix multiplication before applying the activation function

$$y = g(Wx + b).$$

Adding the bias term adds similar expressive capability to the linear transformation as the intercept term does in linear regression, but otherwise does not affect the computations and we drop it from future equations for notational brevity.

By definition of matrix multiplication, each element $y_e \in y$ is a weighted sum of all elements of the input vector

$$y_e = g(\sum_d W_{ed} \cdot x_d)$$

where for each y_e there is a separate D dimensional weight vector as the row W_e of the weight matrix. Consequently a FC feedforward network can be considered to be a directed acyclic graph (DAG) where each FC layer is a group of E nodes, and each node has one

input weight W_{ed} for each node in the previous layer.

The purpose of the activation function is to bring the capability to do non-linear transformations to a network, as a series of matrix multiplications can only represent linear functions. Two commonly used activation function examples are the hyperbolic tangent [24]

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

where $\tanh(x) \in (-1, 1)$, and the nowadays common rectified linear unit (ReLU) [10]

$$\mathrm{ReLU}(x) = \max(0, x).$$

2.3.2 Training and backpropagation

In the context of a simple FC feedforward network, the weight matrices W and the bias vectors B in each layer are the network's trainable parameters. As the non-linear function composed of all the layers of a FC feedforward network can be extremely complex, a closed form solution for finding the optimal parameters that minimize the loss does not exist. For this reason training NNs is almost always done with iterative gradient-based methods such as SGD we described in Section 2.2.2. While SGD is relatively simple and there exist methods that try to improve upon it such as ADAM [19], in many scenarios SGD is still known to generalize better [43], and hence we only use SGD due to its simplicity.

We established the supervised learning framework for iterative learning in Section 2.2.1 and thus know how to compute a loss for a machine learning model. However computing the gradients for each parameter of a NN wrt. the loss efficiently is not trivial, especially since modern NNs can have billions of parameters.

Backpropagation

Backpropagation (BP) is an algorithm for finding the gradient of the loss function wrt. the feedforward network parameters efficiently with a dynamic programming style algorithm [24]. We work towards a description of the computational problem solved by BP by first explaining how the loss gradient is calculated in the mathematical sense.

As we have seen, feedforward networks with FC layers are nothing more than compositions of differentiable elementary functions. This makes it possible to calculate the loss gradient wrt. the network parameters by using the chain rule of calculus.

Let us consider a network with L layers, where l_i is the ith layer and i increases from input layer toward the output layer. Since the network is a DAG the parameters θ_i of a layer l_i only have an effect on the function composed by layers $l_{i+1}...l_L$ in addition to the loss function itself. Furthermore, in order to compute the partial derivatives of the loss function wrt. the parameters θ_i of a particular node in layer l_i it suffices to compute the partial derivative of the composite function formed by all paths in the graph starting from the node N leading to the loss function. Herein lies the computational problem, as even for a DAG there can be an exponential number of paths from *every* node to the loss function, and in a naive method the partial derivatives would be recomputed for every path and subpath.

The partial derivatives for the last layer l_N can be computed easily and computationally cheaply as it does not depend on anything else than the loss function. From there, after computing the partial derivatives for a layer l_i and storing them so that they can be reused in the computations of layers l_{i-1} ... l_0 it can be thought that we are *propagating* the derivatives *backward* in the network. This gives the BP algorithm its name.

Historically BP can be considered a special case of reverse-mode automatic differentiation published first by Seppo Linnainmaa in his Master's thesis from University of Helsinki in 1970 [25]. Automatic differentiation in turn is based on the chain-rule of differentiation, which is generally attributed to be invented by Leibniz already in 1676 [11]. Backpropagation was suggested for training neural networks for the first time in 1986 [34] while the first practical usage was in 1989 for the recognition of zip codes from hand written numbers [23].

Modern deep learning networks are based on reverse-mode automatic differentiation [33]. In practice the ready-made library implementations are always used.

Initialization

Initializing the parameter values or weights of the network nodes' inputs randomly while ensuring useful statistical properties is important for efficient learning, especially in deep networks [15]. For networks of any size, the most important thing is to *break symmetry*, as units with identical starting weights and inputs will stay identical throughout training.

One of the simplest methods used for initialization is to draw the initial weights from a Gaussian distribution with zero mean and unit standard deviation [21]. While this initialization method breaks symmetry and is sufficient in theory, initialization of deep

networks often requires extra attention to work in practice [9]. Unless the weights of networks are initialized carefully with scaling in mind, each layer may have a scaling effect at the beginning of training, where the layer output values have a larger magnitude than their inputs by some factor $\alpha \neq 1$. Assuming a succession of L identical layers is applied on the input, this would mean the layers have a scaling factor of α^L in total. For a large L This can lead to some layers receiving input values that have extremely large magnitude or are very close to 0. In practice, values with large magnitude slow down or freeze training by causing small or zero gradients due to saturating activation functions. On the other hand, due to lack of numerical precision, values too close to zero may be rounded to zero, which in turn can result for example in illegal values from division by zero corrupting the training process.

To explain the basic idea of how initial values for weights can be picked to ensure each FC layer has a scaling factor of $\alpha \approx 1$, we start with a simplified case with no activation function before describing the effect of activation functions. Since FC layers with D dimensional input have a separate set of D weights for each output, it is enough to discuss the solution from the perspective of having a single output variable, as exactly the same process applies for every output.

We can formulate a way to make the scaling factor $\alpha \approx 1$ in all layers by requiring that the random variable O_l, corresponding to some output variable of the lth layer, has unit variance

$$\mathrm{Var}(O_l) = 1.$$

We drop the l subscript from the outputs O for notational brevity. Now consider the inputs of a layer to be a collection of D random variables where I_d is the dth variable, each with a mean of 0 and a standard deviation of 1. We can make $\mathrm{Var}(I_d) \approx 1$ for the first layer by normalizing each dimension of the input data to have approximately zero mean and unit variance based on the empirical mean and standard deviation, where the empirical estimates are computed from the training dataset. Furthermore consider the weights to be a collection of D random variables where S_d is the dth such variable defined by a some distribution with mean and variance are now trying to find good values for.

Since without an activation function the output random variable is defined as

$$O_l = \sum_d I_d S_d,$$

its variance is defined by

$$\mathrm{Var}(O) = \mathrm{Var}(\sum_d I_d S_d).$$

Knowing that for the variance of a sum of independent variables (they do not have to be from the same distribution) it holds that $\mathrm{Var}(\sum_j X_j) = \sum_j \mathrm{Var}(X_j)$ (see eg. [4]), we can show

$$\mathrm{Var}(O) = \mathrm{Var}(\sum_d I_d S_d) = \sum_d \mathrm{Var}(I_d S_d). \tag{2.1}$$

Furthermore it holds for the variance of the product of independent variables [12, Eq. 2] that

$$\mathrm{Var}(X_1 X_2) = \mathrm{Var}(X_1)\mathrm{Var}(X_2) + E[X_1]^2\mathrm{Var}(X_2) + E[X_2]^2\mathrm{Var}(X_1)$$

where in the case where both variable's mean is 0 we can make it into the form

$$\mathrm{Var}(X_1 X_2) = \mathrm{Var}(X_1)\mathrm{Var}(X_2),$$

which we can apply to Equation 2.1, since $E[I_i] = 0$, and we can choose to set $E[S_d] = 0$, to obtain

$$\mathrm{Var}(O) = \sum_d \mathrm{Var}(I_d S_d) = \sum_d \mathrm{Var}(I_d)\mathrm{Var}(S_d).$$

Since the inputs are normalized so that $\mathrm{Var}(I_d) \approx 1$ we have

$$\mathrm{Var}(O) = \sum_d \mathrm{Var}(S_d).$$

where in order to fulfill our requirement

$$\mathrm{Var}(O) = \sum_d \mathrm{Var}(S_d) = 1$$

we can simply set

$$\mathrm{Var}(S_d) = \frac{1}{N}.$$

This setup where the weights are initialized by drawing them from a distribution with a zero mean and $\frac{1}{N}$ variance combined with a $\tanh(x)$ activation is recommended for example by LeCun et al. [24]. $\tanh(x)$ activation is very close to linear around zero and

$\tanh(0) = 0$, and so for inputs having zero mean and relatively small variance it has the desirable property of approximately retaining the mean and variance of its inputs. This also justifies using a distribution with zero mean for initializing the weights with a little more than just convenience in the derivation, as the desired properties are retained best exactly around zero.

For asymmetric activation functions such as ReLU we cannot use exactly the same derivation as above, since

$$E[I_d] \neq 0.$$

Still a very similar idea is applied for finding a variance that depends on the number of inputs in the derivation introduced by Kaiming et al. [15] that takes into account the characteristics of ReLU.

Finally, the specific type of distribution used does not seem to be important as long as variance is properly chosen. For example the PyTorch library [33] uses an uniform distribution by default while many articles mention the use of a normal distribution [24, 15].

2.3.3 Convolutional neural networks

Convolutional neural networks (CNNs) are feedforward networks with convolutional layers, where each convolutional layer contains multiple convolution kernels or *filters* [23]. As opposed to the use of convolution in rule-based systems as described in Chapter 2.1, in CNNs the kernel weights are learned based on data.

When applying convolutional layers to D dimensional data divided into C_{in} input channels, a separate D dimensional filter or kernel is usually learnt for every input channel. Furthermore if C_{out} output channels are desired, each output channel has their own set of C_{in} filters and the resulting $C_{out} \times C_{in}$ filter responses are summed element-wise over the C_{in} dimension to form each output channel response. The convolutional channels are visualized in Figure 2.5. This means there are $C_{out} \times C_{in}$ D-dimensional filters and every output channel is a connected to all input channels. A less used variation exists called depthwise-convolution where only some input channels are considered for each output channel, but we do not use it. Similarly as in classical computer vision, CNNs used for image data often use 2D convolution. The 3 color channels red, green, and blue (RGB) are often used as the initial channels.

The main argument for using convolutional as opposed to FC layers for image data is that

they considerably reduce the number of parameters required to cover a full image. For example if we consider a black and white one channel image of size 512×512 and want to output a 256 dimensional embedding, this requires a FC layer corresponding to a matrix of size $512^2 \times 256$ resulting in 67,108,864 learnable parameters for the single layer, while a convolutional layer with 1024 (a relatively large amount) individual 2D filters of size 3×3 would have $3 \times 3 \times 1024 = 9,216$ parameters.

CNNs can also be seen as exploiting the translational invariance of images. To give an example in the context of this thesis, a potato on a plate of food is still a potato irrespective of its location in the photo. The architecture of fully connected layers completely disregards this useful prior belief as each learned weight is only applied once and to a single location in the input during the forward pass for each sample. This corresponds to using a convolutional layer with the filter's size being the same as the input size and not using any padding. Another aspect of natural images that CNNs make use of is that objects often occupy only small local neighbourhoods of images rather than being spread out randomly over the image, which is exploited by the use of kernels.

Furthermore, as the filters are usually much smaller than the inputs, and every weight in every filter of a convolutional layer is applied to every or almost every input element (depending on the type of padding used for the convolution), this encourages the network to learn filters that are maximally applicable to every part of the image. Using small filters thus has two-fold benefits, they both lower the number of parameters needed to cover the whole input area, and since each weight is applied to many values on each forward pass they are also inherently more resistant to overfitting.

As a glimpse into what kind of feature detectors CNNs actually learn, Zeiler and Fergus [42] show the images that cause the strongest activations for a particular layer and in doing so demonstrate that the earlier layers of a CNN essentially learn edge and simple shape detectors while the latter layers focus on increasingly complex features and invariances within the input images [42]. For example the images causing the highest filter activations for the layer 2 in their network tend to be images with simple repeating patterns with similar rotation, while for layer 5 the images for one filter consist of dog faces and for another the images are different kinds of wheels in various orientations.

One common component in CNNs is a pooling layer [11], where for example only the maximum or average value of some feature map local neighbourhood is passed on effectively downsampling the feature maps. For example in a feature map with 2 spatial dimensions, pooling over a 2×2 square halves the size of both spatial dimensions which dramatically

increasing model training and inference speed while in some cases also boosting accuracy [21].

Backpropagation of convolutional layers is in a mathematical sense similar to that of FC layers. Since convolution is a linear operator [39] it can be represented as matrix multiplication even in the case of 2D convolution, provided that the input is flattened to 1D first, and thus it can in theory be backpropagated through like a FC layer. However in practice deep learning frameworks have ready made implementations that handle backpropagation through convolutional layers efficiently behind the scenes [33].

Historically CNNs have played an important part in neural network research. The first application of backpropgation to learning neural networks in a practical scenario was in using CNNs for handwritten zip-code recognition [23]. It could be argued that the most recent deep learning "craze" was also started when the ImageNet image classification competition was won for the first time with a neural network based model, where the network was also a CNN [21].

2.3.4 Training deep networks

Learning image content predictors along with the feature detectors end-to-end benefit from being trained on massive datasets. For example in the commonly used ImageNet dataset there are 1.2 million training images with around 1000 images for each of the 1000 classes [5]. Since as described in Chapter 3.1, in this thesis our dataset has around 1700 training images that includes around 130 different food types, it is beneficial to utilize all available techniques to get the most out of the limited data.

We explain five deep learning techniques to reach good model performance: batch normalization, skip connections (used for forming residual networks), transfer learning, dataset augmentation, and early stopping.

Transfer learning, augmentation, and early stopping

Transfer learning in general refers to transferring information between related domains [44]. From the perspective of neural networks, transfer learning is often to refer to the practice of *pretraining* a network first on a larger and more generic dataset than the target dataset. Networks pretrained on a large generic dataset before training on the target dataset often perform better than randomly initialized networks only trained on

the specific target dataset.

Augmentation for images makes use of the fact that most objects are the same no matter their orientation or location in an image. A common way to increase the amount of training data is to take different crops of images, rotate them slightly, or to change the saturation of the images slightly. Augmenting the dataset by rotating the images is particularly useful with convolutions as they are not rotationally invariant. While the new semi-synthetic training data acquired through augmentation often helps, completely new images are always better than artificial data gained through augmentations.

Early stopping refers to stopping the training of a neural network before the training loss has converged. An often useful technique is to have a separate validation set that is not used for training, and monitor the loss for the validation set. Often neural networks start to overfit before the training loss has converged, and at some point validation loss starts to increase. If intermediate model versions are saved during training, the model with the lowest validation loss can be picked, which is sometimes referred to as early stopping.

Batch normalization

While initialization methods can only provide computationally convenient layer-to-layer dynamics for the beginning of the training process, batch normalization (BN) attempts to maintain them also during training by normalizing the output statistics with batch-wise approximates for the mean and standard deviation [17]. More specifically, in BN the mean and standard deviation of network node outputs are reparameterized so that both output statistics are represented by a new learned parameter. This is achieved by first normalizing the layer output values to zero mean and unit standard deviation with batch-wise approximates, and then returning the statistical properties back with learned parameters that then represent the mean B and standard deviation σ. The output is then simply $\sigma f(x) + b$.

This removes the dependency of these statistics on the complex dynamics of the previous layers, and it instead only depends on the learned values helping learning. While this answers what BN does, it is still debated why exactly this modification is beneficial. The consensus is just that empirically it seems to speed up training.

Skip connections

The ResNet architecture by Kaiming et al. [14] is popular in food computing [30] and in many other application areas dealing with images. ResNets are mostly based on the basic building blocks of CNNs: convolutional layers, max-pooling layers, and batch normalization [14]. The addition that distinguishes ResNets from other networks is the grouping of layers into residual blocks.

The residual blocks consist of two convolutional layers both with a ReLU activation, but most importantly also feature a skip connection. The skip connection sums the output of the two layers with the input of the whole block and so make it easier for each block to represent the identity function, as they only need to pass on the inputs. Kaiming et al. empirically demonstrate this makes deep networks easier to optimize and argue that deeper rather than wider networks are often desirable in visual recognition tasks.

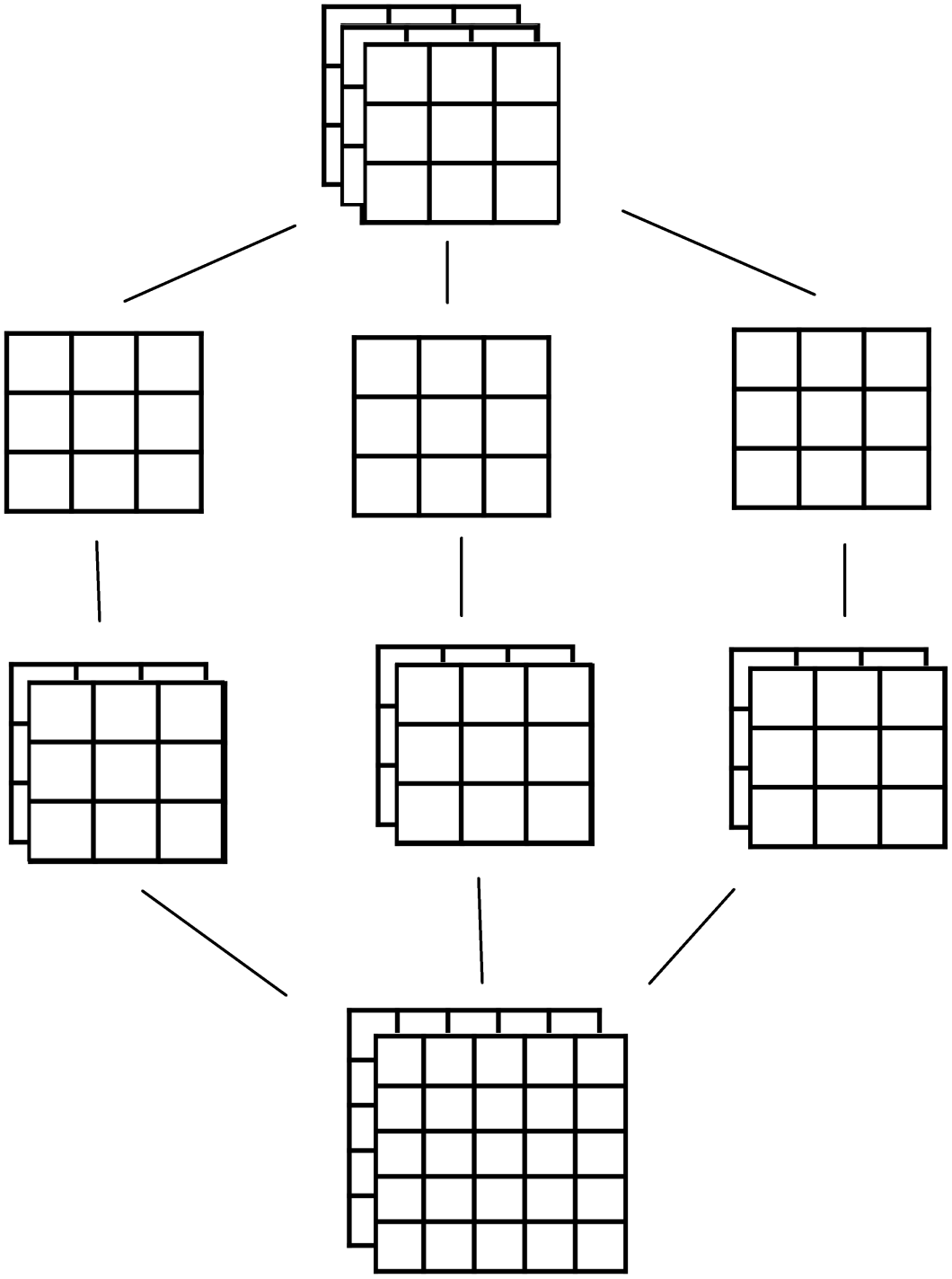

Figure 2.5: Illustration of feature maps and channels in a single 2D convolutional layer. Each of the 3 output channels (at the top) have 2 separate 3×3 kernels for each of the 2 separate 5×5 input channels (at the bottom). The output channels are formed by taking the element-wise sum of the channel's 2 feature responses created by convolving the input with the 2 input channel specific kernels.

3 Data collection and preprocessing

In this chapter we describe how the dataset of images and weights was collected and preprocessed. The weight preprocessing steps consist of a collection of heuristics and for images we show a method for finding and cropping plates.

We begin with a description of the Flavoria restaurant along with practical details of how a dataset of images and food weights was collected there. We characterize the dataset with some key statistics and show some example images. Next, we cover the most common problem scenarios that occurred during data collection and how they can be addressed with automatic preprocessing techniques.

3.1 Data collection

In this project a new dataset was collected in collaboration with Flavoria. Flavoria is a research restaurant in Turku that serves food in self-service buffet style lunch lines. The restaurant has an existing food weighing system where scales have been installed next to where each type of food is served [20]. One part of the lunch line with scales is depicted Figure 3.1.

The customers take the food and do the weight measurements themselves. For most side dishes such as rice the customers are free to take as much as they want, while for main dishes such as steaks, fish sticks, and meat balls there is a maximum number of items (unless an extra price is paid). The restaurant has registered customers to whom a tray is linked upon showing a QR code from a mobile phone app at the cashier when paying for the meal. One such tray is shown at the cashier in Figure 3.3. The customer can then view the nutritional information of their meal from the app based on the weight measurements. Not all users use the phone app and the trays' data that has not been linked to a user is referred to as unregistered data. While weight data is collected for both registered and unregistered clients, the unregistered users' data is of lower quality likely due to those users' lack of interest in doing the weight measurements properly as they are not interested in the results. Further details of the weight measurement system are described in Koivunen et al. [20].

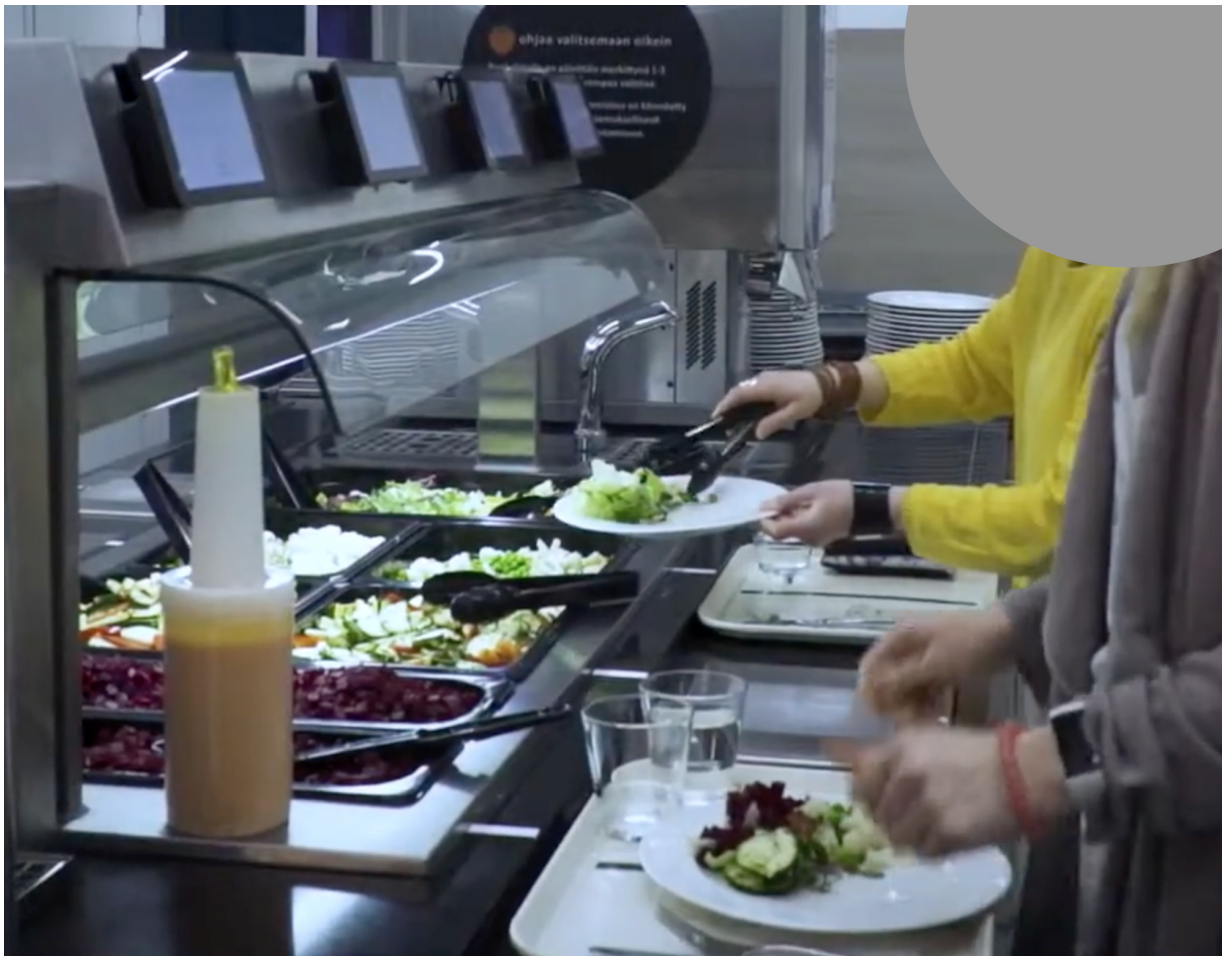

Figure 3.1: Part of the lunch-line at Flavoria with four scales. The scales are integrated into the desk the trays are on. The four displays at the top inform the customer when the weighting has finished for the corresponding scale. Picture from video, ©Flavoria [8]

In this project one RGB and one infrared (IR) camera was mounted above the cashier in order to associate an image of the full meal with the corresponding meal component weight data. The two camera setup is shown in Figure 3.2 while an example image taken with the RGB camera is shown in Figure 3.3. The camera was programmed to automatically take a photo of the tray both when the NFC tag of a tray was detected to have arrived at the cashier as well as when a registered user showed their personal QR code from their phone after payment. For this reason there are two separate images of some of the trays.

The weight and RGB image data was collected over a period of six weeks during lunch time on weekdays. The IR images were only collected for the latter three weeks. Approximately 4300 RGB images and 1800 infrared images were captured with around 12,000 weight measurements of the individual meal components.

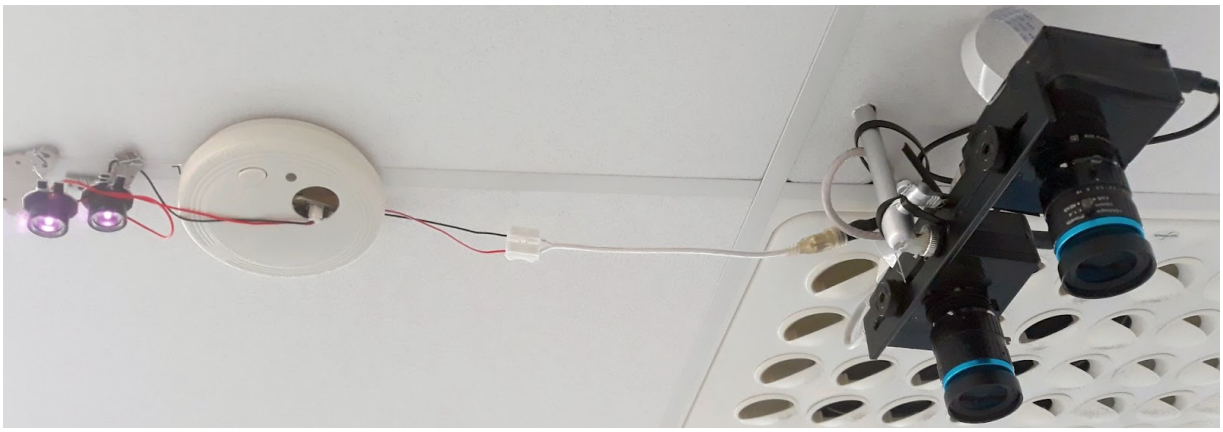

Figure 3.2: A dual camera setup was mounted above the cash register, one RGB and one IR camera

In preliminary experimentation it was that concluded that the IR images do not seem to contain useful extra information, possibly due to the IR camera only capturing a wavelength that does not contain information that could be useful for distinguishing between foods, and so the IR images were disregarded for further study in this project.

In one to two days of each week the customers were helped in using the system by Flavoria staff. This was done in order to improve data quality, since some customers had difficulties with using the scales.

3.2 Preprocessing

Here we describe the most common problems found in the initially collected data. Based on observations of how the customers use the lunch-line, we give likely explanations for what caused the errors, and describe the preprocessing steps taken to improve the quality of the final published dataset.

For images the most common errors in data collection were blurring due to fast movement during capture, and complete or partial occlusion of the food due to the customer's head being between the food and the camera. We address both problems in Section 3.2.2 by using the circle Hough transform discussed in Section 2.3 to determine if the image contains a plate, and by filtering out the images where no clear circle could be found.

For weight data, we plotted typical weight distributions of food items and manually browsed through some of the image and weight pairs to and found repeating patterns where the weight measurements have clearly failed. We design heuristics for filtering out or imputing measurements depending on what is reasonable for each type of failure.

Figure 3.3: Example image from the dataset before preprocessing

3.2.1 Weight preprocessing

Since the meal components are weighted manually by customers who may misunderstand the instructions or simply do not care for the weighting results, the weight dataset contains erroneous weights. Through preliminary manual inspection of the images and corresponding weight data, it became clear that the unregistered user's weight data is of such low quality that we chose to disregard it.

In preprocessing the registered user data the easiest error to detect is the case where the user fails to tare the the scales properly. At the beginning of each weight measurement the scale needs to be calibrated so that it knows starting total weight of the tray (including plates, food from earlier measurement points, and personal accessories etc.) in order to correctly measure the amount of food added at this station. However many careless users either do not wait for the initial measurement to complete or for example hold the plate in their hand during it thus accidentally measuring the plate and its existing foods into the

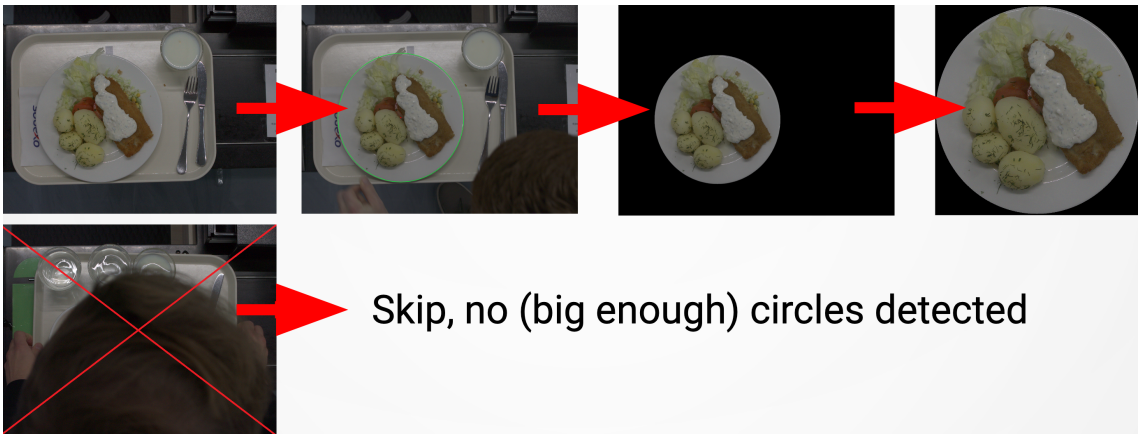

Figure 3.4: Preprocessing pipeline with Hough transform. Upper row shows a successful detection of a circle and the result image after cropping, while lower row shows an image that gets discarded due to missing the plate.

measurement. Another typical error scenario arises when a user places their mobile phone or other personal accessories onto the tray in between the initial and afterward weighting. All these cases result in erroneous weightings usually in the range of hundreds of grams. The average weight of food taken by customers varies drastically per food class, and some real weightings of some foods may range from under 100 grams to over 300 grams. This motivates the usage of a per-class cut-off for deciding which food weights are assumed to be failed.

In Figure 3.5 we plot a histogram of three foods' weight distributions and show with a horizontal dotted line the weight that is two standard deviations from the mean weight of that food class. From the figure we can see that almost all weights for these classes are centered around the same mean with a relatively small standard deviation, likely meaning other weightings are failed measurements. Since in these cases the food item was still taken even if the measurement was failed, and the food item will be visible in the image at the cashier, setting the weight of these foods to zero would confuse any models, and so the likely failed measurements' weights are set to the median weight of the food.

Additionally through manual inspection of the images it was identified that for a vast majority of measurements between 1 and 9 grams, no food at all of the given class was present. For this reason weights equal to or under 9 grams were set to zero grams.

As a final preprocessing step the methods described in Chapter 3.2.2 were used to make sure each image selected into the dataset actually contains a plate and to remove parts of the image outside the plate. The remaining images were manually inspected and images still containing sensitive data (eg. due to a credit card being directly above a plate of

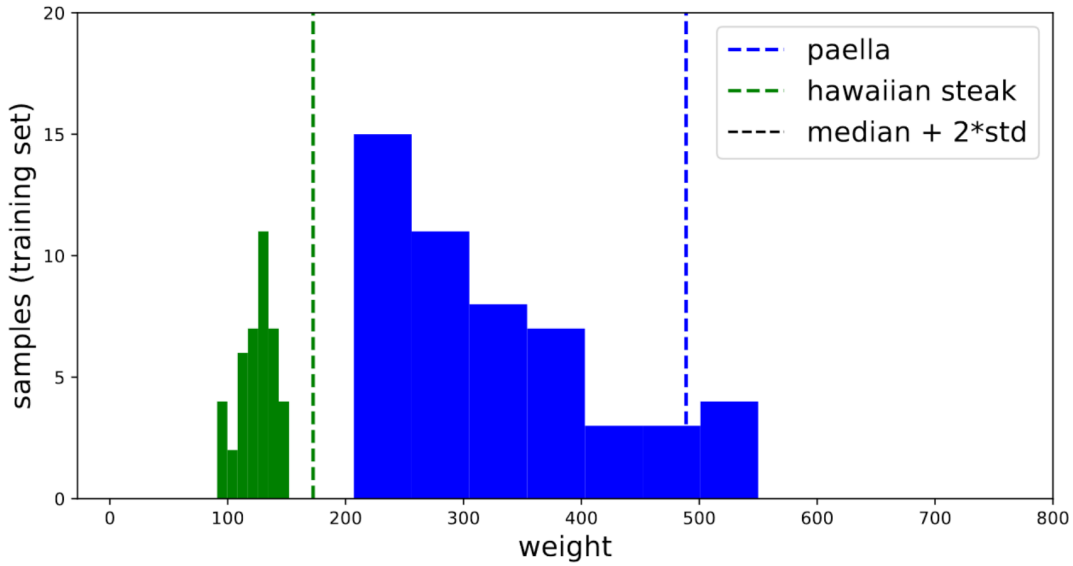

Figure 3.5: Weight histograms that shows the weight distributions for two types of food along with the (median + 2*std) cutoff points. Hawaiian steak is served as a single piece, while customers may take as much paella as they like. Weights above the cutoff are assumed to be erroneous and are set to median weight.

food) were discarded. The dataset was split into training and test sets with 1716 and 196 images respectively, while making sure images of the same tray always ended on the same side of the split. A validation set of 104 images was collected from the discarded unregistered user images and weight data by manually inspecting and choosing images with weightings that seemed reasonable.

The final preprocessed dataset contains around 2000 images and a total of 7,890 weight annotations. The dataset has been made publicly available at https://zenodo.org/record/5850856.

3.2.2 Image preprocessing

Firstly, we wanted to discard low quality images, for which there were two main reasons, either the tray was moving too fast while the image was being taken causing a unfocused image where the plate was too blurred, or a customer had leaned over their tray occluding the food partially or completely. Secondly, many of the tray images contained sensitive personal information such as credit cards where the name and number was clearly visible. In many jurisdictions storing this kind of information may not be allowed. Thirdly, removing parts of the image that do not contain food helps make sure the classification and

weight prediction models developed using the dataset in Chapter 4 are actually making the predictions based on the food in the image and not other irrelevant details visible in the images.

We implemented an automatic method for determining if an image contains a plate by using the Hough transform as described in 2.3 to decide if an image has a strong enough circle of a predefined size. We discard images without clear plates, and for images with plates we crop them so that everything outside the plates is removed as visualized in Figure 3.4.

Using circle Hough transform means the following parameters need to be selected manually:

- Upper and lower thresholds for hysteresis
- Circle radius
- Minimum accumulator value to count as a circle

We selected the parameters with trial and error by starting with an educated guess, then observed if there were too many, too few, too small, or too big circles detected, and adjusted the parameters accordingly. In the end we selected 50 and 25 as the upper and lower threshold values for hysteresis (out of a maximum value of 255 for 8-bit images), selected 220 pixels as the circle radius to be searched for from an image scaled to the size of 1024×768 (width times height, retaining the aspect ratio of 20:15). The minimum accumulator value to be counted as a circle center was set to 40.

After this process multiple circles were sometimes generated. We discard all circles whose centers are inside circles that have already been selected. This method was effective in finding only one circle from images containing a plate, and not finding circles where the image was blurry of the plate was blurry.

Finally, if a circle was found, we keep the image, and based on the coordinates of the circle center in the downscaled image, we calculate the location of the circle in the original 2000×1500 image, and crop that area. We discard the images where no circles were found.

4 Method and experimental validation

In this chapter we present the combination of new and pre-existing neural network based machine learning methods that were applied for the tasks of food identification and weight estimation in this project. There are currently few if any theoretical tools for predicting whether a large neural network or any components added to it will perform well in practice for some given dataset and related problem [11]. Since the purpose of this project is to investigate the feasibility of using image-based machine learning methods for two specific food computing tasks for use in the real world, the practical performance of the models is an important consideration. Building a well-performing neural network model is a cycle of repeatedly making educated guesses of what could work, and then performing experiments to gather empirical evidence of whether it has a positive effect on the model accuracy, and keeping the components that increase performance. This has two important consequences for this thesis, firstly, we build our method on top of a model architecture that has already been shown to work well in the image-based food computing setting, and secondly, we choose to immediately follow each major model component description, and the hypothesis of why it should work, with the relevant experiments that give empirical evidence on whether it actually also works in practice. We also discuss some of the intermediate results next to where the results were reported where it makes sense for justifying the next steps of the model development, while leaving the main discussion of the results in Chapter 5.

We begin by giving the general architecture of the model that our method is based on. This is followed by a description of how this model can be adapted for the two tasks of identification and weight estimation in our specific setting where there can be multiple foods in each image. Next, as it is critical that we are able to report the results in an informative way, we describe metrics that can be used for measuring performance in the two tasks. Having then described a basic model, we provide the first experiment, an *ablation study* where we investigate which parts of the base model are essential by measuring whether removing them has any effect on its performance.

Building on top of this base model, we suggest a method for possibly improving the model by providing the model with daily menu information in a suitable way, as this kind of information is realistically available in a restaurant setting. To our knowledge utilizing

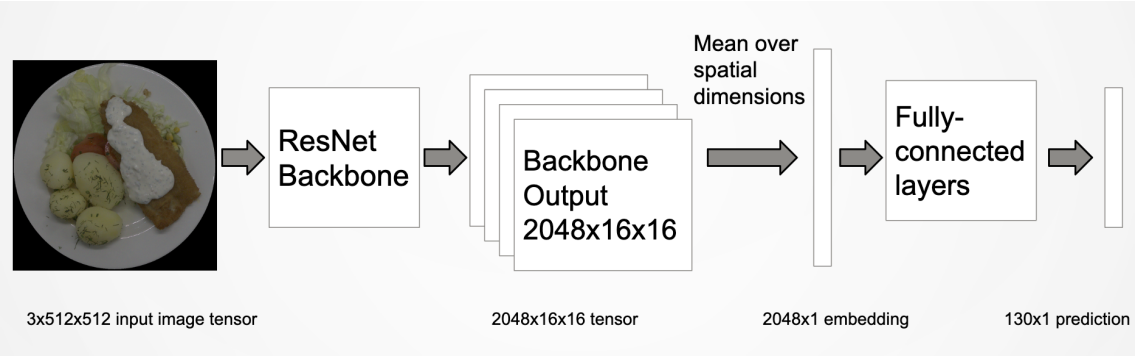

Figure 4.1: Model architecture and outline

the daily menu information with a neural network has not been attempted in any earlier work. Finally the effect of each part of the so called *menu module* is experimentally measured.

4.1 Model outline

The outline of our model pipeline from (preprocessed) input images to classification or regression predictions is visualized in Figure 4.1. As a brief summary, the classification model predicts a value for every food separately that we consider to be the model's probability estimate of whether this food exists in the input image, while the regression model directly predicts a weight for every food for the input image. We describe the task-specific parts, such as the loss functions, of the model in more detail in the subsections. We start with an outline of the model architecture that is shared by both tasks. It is important to note, that even though the architecture of the two models is in many parts identical for both tasks, the models are completely separate and have their own parameters.

As the backbone network in most of our experiments we use the 18, 50, or 101 layers deep versions of a residual network (ResNet) [14] pretrained on the ImageNet dataset [5]. The basic idea of ResNets is described in Chapter 2.3.4. We choose to use a ResNet because they are the backbone used in the vast majority of similar food computing and recognition papers published from 2017 onwards [30]. This provides a starting point that has already been proven to work reasonably well in the food computing setting, and makes it easier to compare the results of possible future research to ours as ResNet models pretrained on ImageNet are publicly available.

The ResNets we use output a tensor of size $2048 \times \frac{W}{32} \times \frac{H}{32}$ where W and H are the input

spatial width and height dimensions respectively. Since we exclusively use 512 images in our experiments the outputs will have 16 as both spatial dimensions. We take the mean over the spatial dimensions of the ResNet output in order to compress it to an embedding vector $E \in \mathcal{R}^{2048\times1}$ suitable for further processing in fully connected layers. Since a mean was taken over the spatial dimensions, the output has no dependency on the spatial size of the input making it possible to use the network for input of any size, though we don't make use of this capability.

It is common practice to follow the CNN backbone with 1-3 fully connected layers of similar dimension as there are channels in the output from the backbone [15, 36, 40], and so we add an extra 2048 dimensional FC layer before the final FC layer. The final C dimensional FC layer, where C is the total fixed number of different foods considered by the model, outputs the final output vector $O_B \in \mathcal{R}^{C\times1}$ of the base model, where the B subscript is used to refer to the base model output. In the ablation study we evaluate the usefulness of the extra 2048-dimensional FC layer against directly converting the 2048 dimensional embedding vector E to a C dimensional output with a single FC layer. How the output O_B is used to create the predictions for a task is covered in the task-specific subsections.

We train the networks using SGD (described in Chapter 2.2.2) and batch normalization (BN) and use early stopping (BN and early stopping described in Chapter 2.3.4) as a regularization method. We do not use any regularizing terms in our loss functions or the dropout method [38] since based on preliminary experimentation neither had any significant benefit. Goodfellow et al. [11] argue that BN may reduce the need for using dropout as BN already incorporates some randomness to the training process thus reducing generalization error which may explain why dropout is not effective in our case.

4.1.1 Weight estimation

The ground truth food weights (measured in grams) are obtained with the preprocessing steps detailed in 3.2.1. The ground truth weights are used to form a ground truth vector $y \in \mathcal{R}^{C\times1}$, where $y_c \in y$ is the ground truth weight for the cth food, and C is the total number of foods. There is one such vector y for each image I in the dataset.

During inference the base model output O_B is passed through a ReLU to remove nonsensical negative weight estimates. The result

$$\hat{y} = ReLU(O_B)$$

is then directly used as the weight prediction $\hat{y} \in \mathcal{R}^{C\times 1}$ of the regression network.

While the predictions of the regression network could be forced to be non-negative (for training and inference) for instance with $\hat{y} = \exp(O_B)$ or $\hat{y} = O_B^2$, when using these methods the model became too unstable to be trained properly. When not constraining the predictions to be non-negative, it is not obvious how negative predictions should be penalized in calculation of the loss, and for this reason we compare the effect of some stable options for the final activation function in the ablation study in Section 4.4.

The loss function used for training is the mean absolute error (MAE)

$$\mathcal{L} = \sum_{i}^{N} \sum_{k}^{C} |\hat{y}_{ik} - y_{ik}|$$

over the N samples of a mini-batch, where the ik subscripts refer to the fact that there is a y_i and $\hat{y}_i$ vector of size $\mathcal{R}^{C\times 1}$ for every sample in the batch, where C is the number of classes. We chose to use MAE rather than mean squared error (MSE), since as covered in Section 4.2 we use MAE also to report the results, and using a MAE loss tends to result in lower MAE than using a MSE loss. The fact that MAE is not differentiable at zero has no effect in practice.

It is useful to normalize the ground truth y to bring it closer to having a distribution with zero mean and unit variance, as due to the initialization methods described in Chapter 2.3.2, and due to BN being initialized with a zero mean, the outputs of the network will then be roughly in the correct ballpark right at the start of training, leading to faster convergence. Only having relatively small variation between all inputs and ground truths is also practically useful, since then if the training is stable for the first few iterations (one iteration corresponding to eg. processing the forward and backward pass for one randomly selected mini-batch), it is likely to be stable for the rest of the batches.

We normalize each ground truth food weight y_c by dividing it by the median weight of the corresponding food in the training set before using it to compute the loss during training. For example this means that predicting a vector $\{1\}^{C\times 1}$ would equal predicting median weight for every food. An alternative normalization would be to normalize with the maximum weight of all foods, or maximum for each food separately, but in preliminary experimentation we did not notice this has much of an effect.

The normalization scheme means the weight estimation network learns to predict weights in this space of normalized weights. If we need to express the predictions as grams, for example for reporting the prediction error more intuitively as grams, it can be computed

trivially by multiplying each predicted weight by the corresponding food-specific median weight.

4.1.2 Food identification

Since multiple different food items can be simultaneously present in each image, the problem is called multi-label classification. For the classification task the base model output O_B is passed through the standard logistic function h to create probability estimates

$$\hat{y} = h(O_B)$$

where $\hat{y} \in (0, 1)^C$. For inference, we can choose some threshold γ, and decide that all predictions $\hat{y}_c$ where $\hat{y}_c \geq \gamma$ correspond to the network predicting that an input image I contains the cth class.

Designing a classification loss when the target class distributions are imbalanced requires extra attention. Many publicly available datasets have been carefully selected to contain the same amount of samples of every class. This makes the classification training task easier since for this kind of dataset a model cannot reach a score better than $1/|C|$ where $|C|$ is the number of classes, merely by guessing and needs to learn to distinguish between the classes. However, If a class $c \in C$ is overly represented, eg. 80% of images contain class c, this may encourage SGD to learn to simply predict $\hat{y} = a$ for every input x as this can be a strong local optima difficult to break out from.

In the multilabel setting, the easiest way to counter this is to weigh the losses class-wise with the inverse frequency of the classes in the dataset. This means multiplying the loss corresponding to each class with $1/N_c$ where N_c is the number of images where the class is present in the training dataset. Effectively this increases the weight of the loss of rare classes, and decreases the weight of abundant classes. Research into statistically sound selections for the class weighting is still active [29], but we noticed using the inverse frequency weighting suffices for getting networks to start learning to predict a wide variety of classes with good accuracy even with a very imbalanced dataset.

The classification network is trained essentially as C separate binary classifiers within one network, with one binary classifier for each class. This is accomplished with a weighted binary cross-entropy (BCE) (BCE described in Chapter 2.2.3) loss computed separately for each class that is summed over the C classes

$$\mathcal{L} = -\sum_{c}^{C} \frac{1}{N_c} \text{BCE}(y, h(\hat{y}))$$

where N_c is the number of positives in the training dataset for the cth class, $\hat{y}_c$ is the cth element of the ground truth vector $\hat{y} \in \mathcal{R}^{C\times 1}$ containing 1 for classes present in the input image and 0 for others, and y_c is the cth element of the prediction vector $y \in \mathcal{R}^{C\times 1}$. As BCE deals with probabilities the logistic function h is used to transform the output y of the final layer of the network to the $(0, 1)$ range in order for it to represent the predicted probability of the input containing that class. Each class is predicted independently of others, and so the probabilities of classes $\sum_c^C y_c \neq 1$ does not need to equal 1. To produce a single scalar loss value for the total loss for a mini-batch, the mean over all the per-sample losses can be taken.

4.2 Metrics

In order to quantitatively measure the performance of the model and to be able to effectively communicate the results, informative metrics must be designed. Furthermore, the metrics should be task-specific. For example measuring mean absolute error (MAE) between the predicted weights and the ground truth weights is an intuitive metric for weight estimation, as it directly answers the question "by how many grams were the predicted weights wrong on average?", while the MAE for predicted probabilities against binary ground truth labels does not have any intuitive explanation.

More than one metric is usually needed, as different metrics highlight different aspects of the results. For instance, consider a naïve mean estimator that always predicts the mean weight of a food for every item, where the mean is computed over all the non-zero ground truth weights in the training dataset for that food. If for example an image contains four foods, and we calculate the average prediction error of a weight estimate against the ground truth weights only for these four items, this error is easily comparable to the mean estimator, and gives an idea of the accuracy of the estimator. Yet this measure does not tell anything about whether the weight estimator erroneously predicted high weight values for the foods that were not in the image, which in turn is captured by the average error over all items, while not being meaningful to compare to the performance of the mean weight estimator.

4.2.1 Weight estimation

We calculate MAE in three different categories to highlight different details of the results. We first show how the weight prediction error is computed in the three categories for a single sample with C food weights.

First, the total MAE over all possible food items is

$$P_{\texttt{All}} = \sum_{i}^{C} |\hat{y}_i - y_i|$$

where C is the number of food items, $\hat{y} \in \mathcal{R}^{C\times 1}$, $y \in \mathcal{R}^{C\times 1}$.

The second category is the MAE only over items in the daily menu. This is calculated with

$$P_{\texttt{Menu}} = \sum_{i}^{C} m_i |\hat{y}_i - y_i|$$

where $m \in \{0,1\}^C$ is a multi-hot menu vector that contains 1 for classes that exist in the day's menu, and 0 for others.

The third category is the MAE only over items where $y_i > 0$. This can be calculated with

$$P_{\texttt{True items}} = \sum_{i}^{C} g_i |\hat{y}_i - y_i|$$

where $g \in \{0,1\}^C$ is a multi-hot vector where

$$g_i = \begin{cases} 1, & \text{if } y_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Finally to compute this over a given *split* of the dataset, where usually we are interested in the test and validation split results, given the scalar error P_n corresponding to the MAE for the nth sample in one of the error categories, we get the total MAE P for a given category for the whole split with N simply by taking the mean

$$P = \frac{1}{N} \sum_{i}^{N} p_i$$

over all samples. The total split MAE is computed identically for all three categories by using the error p_i for that category. For the rest of the thesis we refer to these error categories calculated over all the samples of a split as `All`, `Menu`, and `True items` for notational convenience.

4.2.2 Food identification

For example in the dataset described in Chapter 3.1, the labels are relatively sparse, where only 4 out of 130 items are present in an image on average. This means that predicting that none of images contain any foods would have an average accuracy of around $126/130 = 97\%$. Clearly a better metric is needed than the proportion of correct classifications to the number of all predictions.

Precision is the proportion of predicted true positives (TP) to all predicted positives, or $\frac{TP}{TP+FP}$, where FP is false positives. Intuitively it measures how accurate the positive predictions were and takes into account the number of false positives. Since precision does not take into account false negatives, it alone is not a good metric. *Recall* on the other hand measures the fraction of how many of the real positives in the ground truth were found. This means recall can be used to detect a high rate of false negatives, while it doesn't take false positives into account. Together precision and recall form a good measure for classifier model accuracy since in order to reach a high precision and recall the model must make few false positive and false negative predictions.

Precision and recall can be condensed into a single number by taking their harmonic mean, also called the *F1-score*. The mean needs to be harmonic due to both precision and recall being fractions.

4.3 Experiment setup

We used the PyTorch library with Python to develop and run all experiments. We used the NVIDIA V100 GPU with 32GB of VRAM for training and inference in every experiment.

In all experiments all models are trained exclusively on the training set, while hyperparameters and early stopping is done based on the validation set results. The test set is only used for the final evaluation of the models in each experiment. The splits are described in Chapter 3.2.1.

We use stochastic gradient descent (SGD) with a learning rate (LR) of 0.01 for classification and 1.0 for weight regression. The LR was selected empirically by aiming for the highest possible value that still trains in a stable way. The value was searched for by starting from 0.00001 and increasing the value ten-fold until training destabilized and then returning to the last stable value. Destabilized training usually manifested as invalid values (NaNs)

within the first three epochs of training. The batch sizes used for SGD depend on the model size and experiment specific details.

The dataset images after cropping the plates were 951x951 pixels so we opted to choose the 512x512 as the image size for all experiments as it is the closest power of two size that does not require upscaling. Most pretrained networks that are publicly available have been pretrained with images with smaller spatial dimensions such as 228×228, but we noticed that the performance was slightly improved when using 512^2 images. The images were normalized so that they lie between $[0, 1]$ by dividing with the maximum value and by deducting the minimum value if it is greater than 0. Weight data was normalized so that the median weight of each class was 1 by dividing the weights by the class specific median weight based on the training data.

We tried various alternative optimization techniques such as learning rate decay, momentum with SGD, and manually switching to a smaller LR after the initial LR had converged wrt. the validation loss. None of these yielded noticeably different results. In our experiments ADAM [19], consistently fared worse than SGD with the validation set even after trying some alternative initial parameters. We did not use weight decay or other regularization terms in the loss as they did not have a discernible effect. We did not see any discernible effect in the generalization ability of the network if dropout regularization [38] was employed, possibly because batch normalization that we do use already incorporates randomness into the training process [11].

In most experiments we trained the networks for 8000 epochs as at this point validation loss had converged for all networks, if a different number of epochs was used it is reported separately in the experiment description.

The classification ground truths are formed from the ground truth vector y by setting all non-zero values to 1. For classification results, we count all predictions $\hat{y} \geq 0.5$ as a positive prediction for the cth class.

For each network the test error was measured for two versions of the model, the model from the last epoch and the model reaching the best validation error. Out of the two options for each model, the results of the version with a better test result in the `True items` category were listed for weight estimation, and the version with the highest F1-score for classification. This helps make the comparisons more fair by allowing all models enough time to converge, while still being reasonably fair for models that converge faster that may have already started to overfit.

4.4 Ablation study

We run experiments to see how 5 simple additional components to a backbone model affect learning. All models in this study use a ResNet18 [14] backbone pretrained on the ImageNet dataset [5] in a multiclass classification task, with the original model's FC layers removed. The `True items` MAE is reported for each model in Table 4.1 while Figure 4.2 shows validation errors during training in the three categories described in Section 4.2.1.

The results highlight that two techniques particularly boost performance. First is the dataset augmentation added in Model 2, where the training set images are rotated by 0-360 degrees at random and flipped with a 50% probability thus artificially increasing the amount of training data. The second effective, or almost required, method is to add an activation function at the very end of the network decreasing or removing penalization of negative weight predictions. This is especially apparent in Model 4 that is trained with an augmented dataset but doesn't have any final activation function and hence does not begin to learn properly.

Comparison between otherwise identical Model 2 and Model 5 shows that using a Leaky ReLU or LReLU

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise} \end{cases} \tag{4.2}$$

as the activation function noticeably boosts performance when compared to a ReLU. Furthermore even Model 1 without augmentations and Model 3 without an extra FC layer both having the LReLU activation reach better results than Model 5. This is a strong indicator that it is better to slightly penalize for negative predictions rather than not at all.

While the effect of the removal of the FC layer in Model 3 has a relatively small effect on the final test set results in Table 4.1, it seems to have a drastic effect on the speed of convergence. Only after 5000 epochs of nearly zero progress the Model suddenly starts improving. This indicates that it is beneficial to consider the E dimensional output vector of the backbone (after taking the mean over spatial dimensions) as an embedding vector of the image on which further processing can be done.

All models except Model 4 seem to exhibit some form of overfitting in the `All` category. However if we presume the existence of menu information, this category is not important, as all values outside menu can be zeroed out.

Model	LReLU	FCLayer	Augmented	ReLU	`True items`
1	✓	✓			50.31
2	✓	✓	✓		**29.62**
3	✓		✓		34.10 (32.25)
4		✓	✓		88.28
5		✓	✓	✓	62.81

Table 4.1: Ablation study results.

1. Base model 2. Dataset augmented with randomly rotated versions of images 3. Not using an extra FC layer (12Kth epoch result in parenthesis) 4. No activation function at the end all 5. Using ReLU as the final activation .

Model 4 was trained for an extra 4k epochs as it had not properly converged, see Figure 4.2.

4.5 Incorporating the menu information

Uniquely to the buffet-style restaurant setting the information which specific food items are on offer for the customers on each day is usually available. Furthermore in our dataset there are very few training samples (1700) compared to the number of distinct food classes (130), while only 10 of those foods are available each day on average. Since knowledge about the menu that is relevant to a given image is both realistically available and could greatly lower the difficulty of the task, these two offer strong motivation for investigation into how to supply menu information to the model.

A trivial method to utilize menu information is to just use a model that is not supplied any menu information and then to zero out any of its predictions outside the menu. This works as a convenient baseline in both classification and weight estimation. If an otherwise identical model that also utilizes menu information with a more sophisticated method beats this baseline, especially for foods within the menu, it indicates that the method helps in creating more accurate models.

Specifically we describe and try two alternative techniques for supplying a CNN model with menu information in this chapter. For both methods we encode the menu as a multi-hot vector $m \in \{0, 1\}^C$, where the values are zeroes for items not in the menu, and ones for values within the menu. As the CNNs we use primarily take 2D input and there is no convenient way to encode the menu information in 2D form, we must investigate other methods.

4.5.1 Menu module

One common way to incorporate one dimensional information of length A stored in vector form as $w \in \mathcal{R}^{A\times 1}$, that as such cannot be directly fed to 2D convolutional layers, is to concatenate the values to the embedding vector $e \in \mathcal{R}^{E\times 1}$ output by a backbone so that the concatenated embedding vector is then $e_{\text{concat}} \in \mathcal{R}^{(E+A)\times 1}$. For example Thames et al. provide a scalar estimate of food volume in addition to the input image to their network in this way [40]. We refer to using this method as `MenuConcat`, where the menu vector $m \in \mathcal{R}^{C\times 1}$ is concatenated to the embedding $e \in \mathcal{R}^{E\times 1}$ thus creating the concatenated embedding $e_{\text{concat}} \in \mathcal{R}^{(E+C)\times 1}$ which is passed on to the two final FC layers.

However a FC layer that takes this concatenated vector as input is only capable of multiplying each input value with a separate weight and then summing them together which makes it difficult to learn some operations, such as suppressing some of the inputs based on others. We hypothesize that it could be beneficial to make it possible for the network to utilize this kind of suppression in our scenario where the menu vector contains categorical yes-no type of information.

We propose the multiplication based `MenuProd` method to address this. While directly multiplying elements of the image embedding vector with elements of the menu vector would otherwise have the desired properties, the dimensions do not match between the output $O_B \in \mathcal{R}^{C\times 1}$ and E dimensional menu vectors. For this reason we define a learnable matrix $W \in \mathcal{R}^{E\times C}$ and first multiply the menu vector $m \in \mathcal{R}^{C\times 1}$ with it to obtain a weight vector

$$p = Wm$$

where $p \in \mathcal{R}^{E\times 1}$ that can then be multiplied with the embedding vector $e \in \mathcal{R}^{E\times 1}$ with the same dimensions element-wise to obtain the weighted embedding vector $o \in \mathcal{R}^{E\times 1}$. We also compare this to an alternative version we call `MenuSum` where the weight vector is summed to instead of multiplied with the embedding vector. In both `MenuSum` and `MenuProd` the menu weighting vector p can be thought of containing as a scalar weight for each embedding dimension that tells how much each feature channel should be weighted for this particular menu.

While here we focused on applying the method to 1D embedding vectors, this method could also be used to apply the menu information to data spread to D channels by learning a $D \times C$ matrix. In this case there would be one weight in the vector o for each channel D,

meaning all values within a channel would be weighted with the same value. Although it could also be useful in earlier layers of the network, we did not notice any improvement in using the method in any other than the final layers of the network. This is likely because the feature map dimensions begin to correspond to particular menu items only towards the end of the network, as the earlier layers of CNNs usually learn more general edge and shape detectors while only the latter layers specialize to more complex objects [42].

4.5.2 Experiments with the menu

The experiments in this section incorporate and build upon the components that improved weight estimation accuracy based on the ablation study. In this particular dataset each food weighting has a corresponding timestamp, and we emulate having the daily menu information based on what foods were weighted on each day according to the timestamp. As the images also have a timestamp we are able to produce a daily menu for each sample or image-weight pair.

We begin with an experiment comparing the performance between addition and multiplication of the menu weighting vector o with the image embedding vector e based on a small backbone network in both classification and regression settings. Then we perform experiments with two larger models to evaluate two different methods of incorporating menu information against not using any menu information, first on classification and then in the regression setting.

Multiplication vs addition

We evaluate `MenuSum` against `MenuProd` in both classification and weight estimation. The results for both models in both tasks are listed in Table 4.2. The validation set error progress during training is plotted only for the weight estimation task and can be seen in Figure 4.3. The differences are quite small, with `MenuProd` reaching a bit better results in regression, while for classification both methods' performance is practically identical.

When incorporating menu information, the features extracted by the backbone network into the embedding e and the menu weight vector p could be of different scales, and the network has to learn how to use this information properly. For `MenuProd` this is easier, as multiplication has an effect that is proportional to the magnitude of the the embedding vector right at the start, while `MenuSum` likely has no effect or too much effect at the beginning, which may explain why `MenuProd` converges slightly faster.

Table 4.2: Multiplication versus addition of menu weighting vector in classification and weight prediction. Separate models are trained for the different tasks. Macro F1 is an average taken over the per-class F1-scores, while Micro F1 is averaged over all predictions not taking classes into account

Model name	Macro F1	Micro F1	`All`	`Menu`	`True items`
ResNet18 + MenuProd	0.89	0.87	1.62	**15.71**	**25.10**
ResNet18 + MenuSum	0.89	0.87	**1.60**	16.68	27.02

Table 4.3: Classification results for the test set, obtained from [35]

Model name	Macro F1	Micro F1
ResNet50 + MenuProd	0.874	0.871
ResNet50 + MenuConcat	0.888	0.881
ResNet50	0.890	0.870
ResNet101 + MenuProd	**0.907**	0.881
ResNet101 + MenuConcat	0.902	**0.886**
ResNet101	0.897	0.879

Menu information in classification

We evaluate the effects of `MenuConcat` and `MenuProd` on classification against a model with no menu information and do comparisons with both 50 and 101 layered versions of ResNet for all three model types. The multi-label classification labels $y \in \mathcal{R}^{130}$ for each image are created by setting for each value $y_c \in y$ to $y_c = 1$ if there is more than 0 grams of that food in the image according to the weight data, and otherwise set $y_c = 0$.

We use two alternative ways to reporting F1-score, as neither is necessarily the correct one. Macro F1 is an average taken over the per-class F1-scores, while Micro F1 is averaged over all predictions not taking classes into account.

Figure 4.4 shows that while all models seem to converge quite fast, the two models without menu information ResNet50 and ResNet101 with green and brown colors respectively reach scores approximately 5 percentage points lower than the other models in all three metrics. However, in the test set evaluations seen in Table 4.3 the difference between models diminishes enough to be practically neglible. For every model increasing the backbone size slightly increases accuracy.

Curiously for all models the F1-score is up to 10 percentage points better in the test set evaluation than for the validation set. This is surprising, since as covered in Chapter

3.1 the validation set was prepared manually and should contain less noisy labels, yet the model performs better on the likely noisier test data. This may be due to the hand-picking of good quality image-weight pairs having a different distribution of noise than the actual data contained in both training and tests sets, that the model has overfit to.

4.5.3 Menu information in weight estimation

Similarly to the previous section we evaluate the effects of `MenuConcat` and `MenuProd` on weight estimation against models with no menu information. Here we also use two sizes of ResNets, the 50 and 101 layer versions.

In this experiment we introduce two baselines, one that predicts zero grams for all foods, and one that predicts the median weight for a class based on the training set food weights. We evaluate the results in four categories, out of which `With oracle` is a new one that refers to using a perfect accuracy oracle classifier in the `True items` category. The other metrics are described in Section 4.2.1. This helps highlight the fact the median predictor baseline results are not directly comparable with the `True items` category, as the baseline does not suffer from a

Additionally, for this experiment we introduce a new model utilizing menu information that we call `ComboModel` that joins together a classification and a regression model, and is trained jointly with a single loss function

$$\mathcal{L} = \frac{1}{C}\sum_{c}^{C} b_c(-\pi_c + |y_c - \hat{y_c}|) + (1 - b_c)\pi_c$$

where π_c is the prediction of the classifier branch for cth class, b_c is a binary indicator whether there is over 0 grams of weight for the cth class in the ground truth similarly to how the multi-label classification labels were created in the previous section.

Essentially this loss function is designed to only create non-zero loss and gradients for the classifier branch for classes not in the image, while producing a non-zero gradients for both branches for classes that are present in the image. In this way the weight estimation branch is not hindered by having to learn to do classification when the classifier branch handles that. During normal inference if $pi_c < 0$, the weight estimates are zeroed out, but for the `With oracle` category this can be skipped to acquire results that are comparable to the median baseline.

Since the weight estimation network has batch normalization layers with bias initialized to zero, it will predict zeros at the beginning. By shifting this output up by one we can

make the network predict 1 or the median weight for all classes right at the beginning of training due to having normalized the data so that the class median for each class equals 1. With this setup the model is already as good a weight estimator as the median baseline at the start of training in the `True items` category. Furthermore we initialize the classifier branch weights from the `ResNet101+MenuProd` model so it also performs well in the two other categories right from the start. We do not use the menu module with the weight estimation branch, but use it for the classifier. The ComboModel is trained in parallel on two GPUs, with the classifier branch running on one GPU and the weight estimator in the other.

As a good comparison model to the ComboModel we also list the results for a `Classifier-+MenuProd+median` model that predicts the class median weight for foods that were classified as present by a `ResNet101+MenuProd` classifier as trained in the previous section's menu classifier experiment.

Weight estimation results for all these models are listed in Table 4.4. Here the benefit of using menu information is more clear than in the classification experiment, while the bigger models also tend to be more accurate in this experiment. If we do not consider the combination models, `ResNet+MenuProd` model is best in all applicable categories. `MenuConcat` is able to beat the plain model in the `All` and `True items` categories but gets worse results in the `Menu` category than the plain model. It should be noted that the singular weight estimation models cannot be evaluated in the `With oracle` category since the weight estimation part cannot be separated from the classifier part. Out of all models the ComboModel is the best in all categories except `True items`, and as the `With oracle` category reveals this is partially due to the imperfect classifier zeroing out some of the predictions.

The differences between model results are not large, for example the improvement of the best result in the `True items` category from the worst result is approximately 9%, and the ComboModel is only 5% better at weight estimation than the median baseline if we disregard classification. In summary the proposed models are able to beat the baselines in all categories, bigger models are more generally accurate, and `MenuProd` reaches a better performance than the plain model in all categories while `MenuConcat` only outperforms the plain model in two out of three. A model combining classification and weight estimation in one model reaches the best results.

Table 4.4: Weight estimation results for the test set. The fields are blank where not applicable.

Proposed models

Model name	`All`	`Menu`	`True items`	`With oracle`
ResNet50 + MenuProd	1.52	15.58	24.04	N/A
ResNet50 + MenuConcat	1.90	16.48	25.07	N/A
ResNet50	2.02	15.21	26.58	N/A
ResNet101 + MenuProd	1.47	15.08	**23.74**	N/A
ResNet101 + MenuConcat	1.85	16.24	25.41	N/A
ResNet101	2.02	15.30	26.52	N/A
Classifier + MenuProd + median	1.29	14.75	25.78	23.33
ResNet101 ComboModel	**1.22**	**14.10**	24.64	**22.18**

Baselines

Zero prediction	3.164	36.45	93.07	93.07
Median prediction	7.263	83.67	N/A	23.33

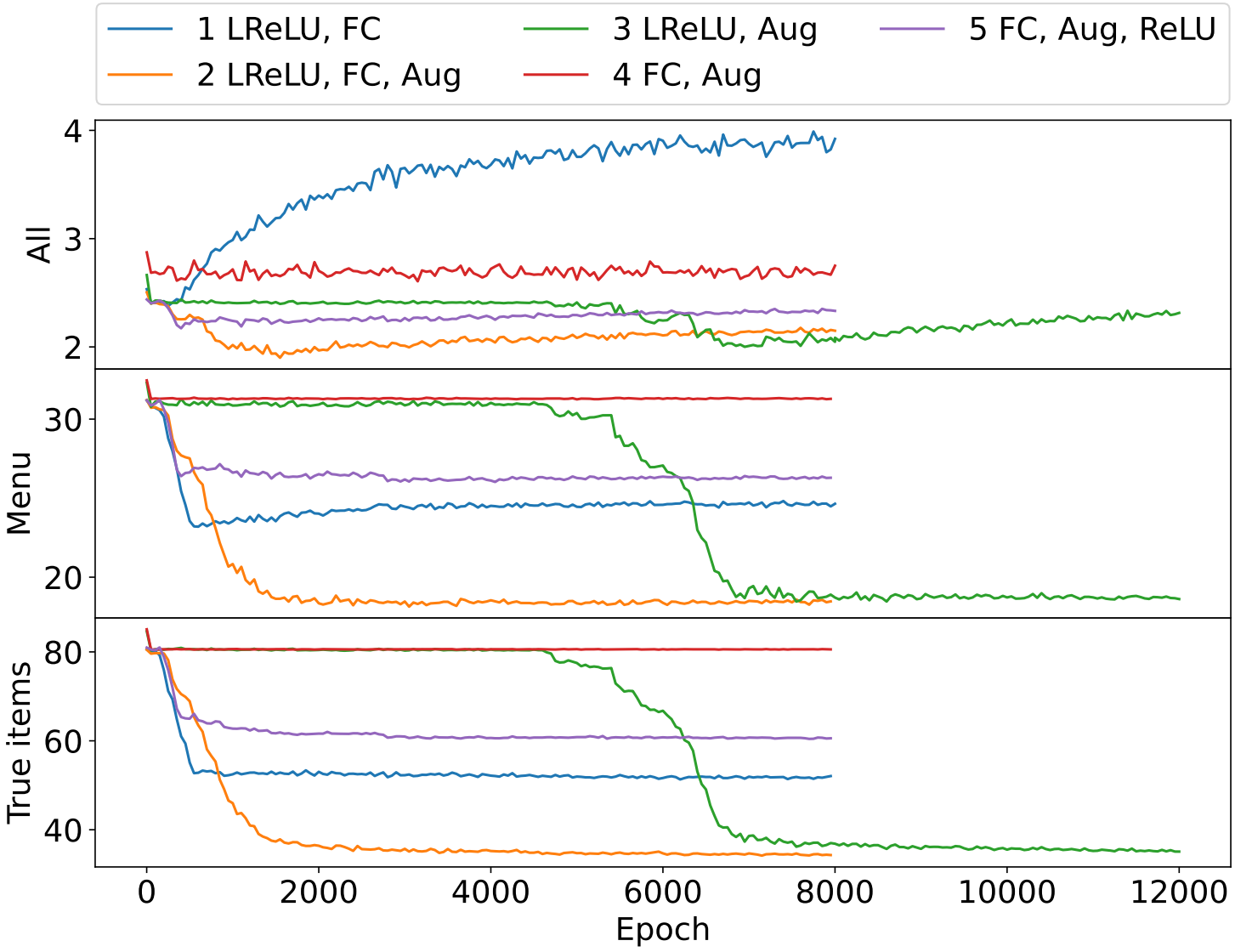

Figure 4.2: Evolution of validation losses during training for all 5 models involved in the ablation study. LReLU and ReLU mean there was a final activation function at the end of the network with a Leaky ReLU or ReLU respectively. FC means an extra FC layer was added at the end of the network, Aug means the dataset was augmented with random rotations and flipping. The three graphs measure mean absolute error (MAE) in three different categories for the validation set. Validation set is evaluated every 50th epoch. It was unclear whether the green model had converged after 8000 epochs so it was allowed to train for 4000 more epochs.

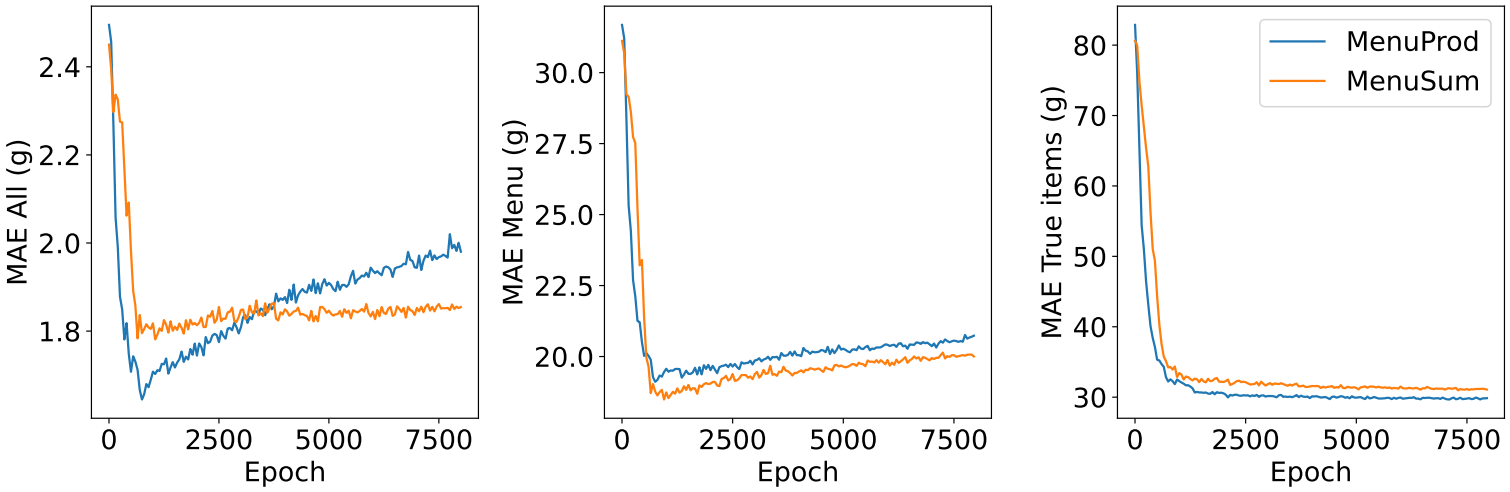

Figure 4.3: Validation error in three categories for `MenuSum` and `MenuProd` in the weight estimation task. `MenuProd` seems to learn a tiny bit faster as the blue line drops faster at the beginning of each graph

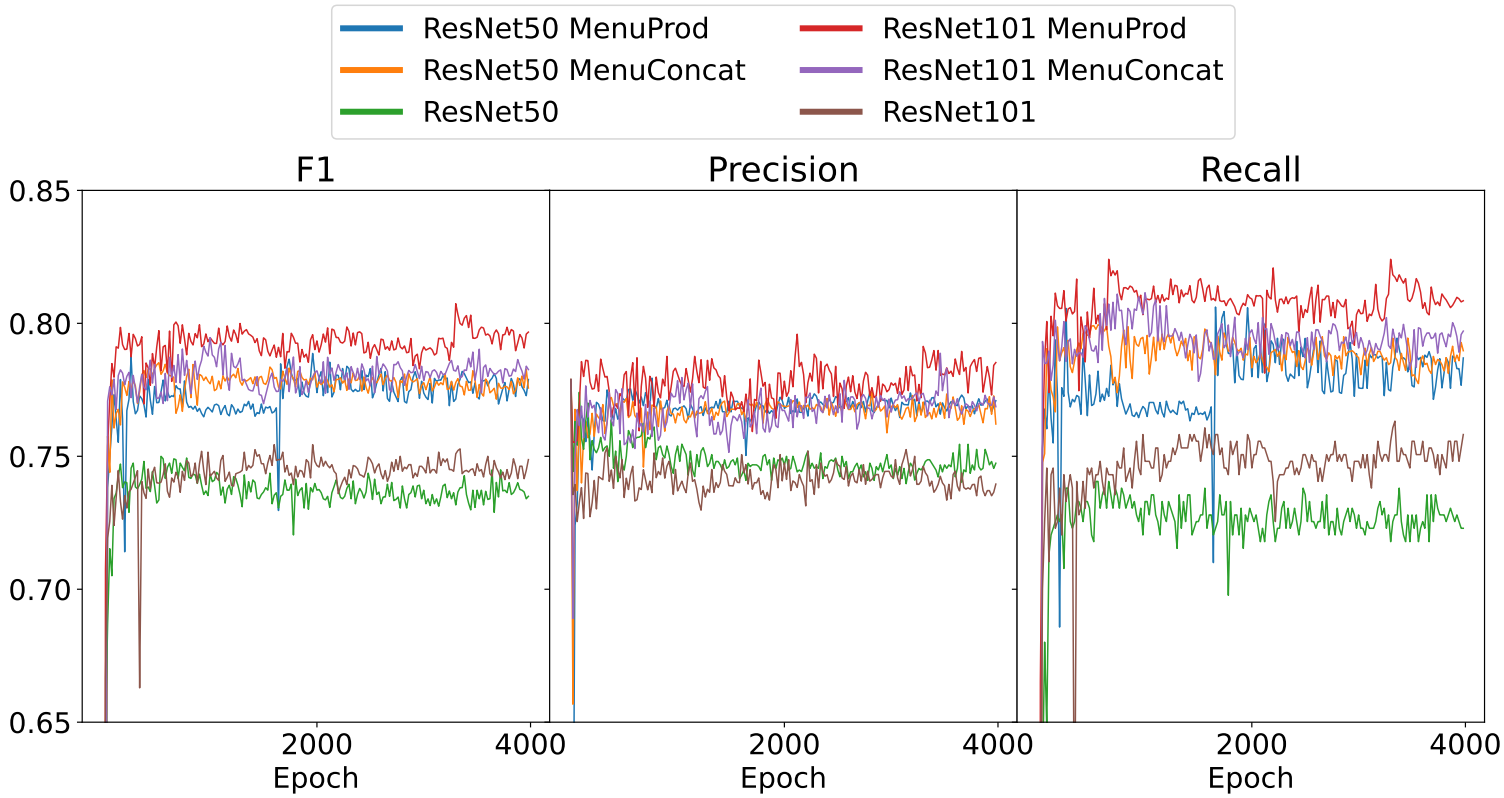

Figure 4.4: Evolution of F1-score, precision, and recall for the different classification models

5 Discussion and results

We were able to empirically demonstrate that incorporating menu information during training time improves results in all relevant error categories during inference for weight estimation, while for classification there is minimal benefit. The benefit is likely due to the large amount of classes (130) as compared to the small amount of training images (1700). Knowing the specific subset of foods that can be present in a given image input could either make it easier for the training procedure to find better local optima during training or for the model to be more confident during inference leading to an improved model.

Furthermore as the food names were provided directly by the restaurant, there were many overlapping classes such as the colored pairs in Figure 5.1 that are difficult or impossible to separate for computing the errors. Yet from the perspective of a customer of the restaurant, these types of errors do not matter as many of the overlapped but separate classes have similar or the same nutritional values.

We attained a good accuracy of over 0.9 F1-score in classification as averaged over classes, especially considering that there is some noise in the classes. The class-specific F1-scores are also quite good as can be seen from Figure 5.2. In the figure all classes with more than 200 images have an F1-score better than 0.8, for many classes the accuracy is perfect, and the low scoring classes have few samples. This indicates that with more training data the model would perform even better.

The rule-based computer vision system using Hough transform proved to be very good at detecting the plates. The system required relatively small time to set up, runs fast, and worked accurately with few false negatives and false positives based on manual inspection of the results.

Given the ill-posedness of predicting the weight of 3D objects from a 2D image, the fact that weight estimation results are not a major improvement (only around 5%) over the simple baseline of predicting the median weight is not surprising. Yet the baseline predictor could be relatively strong for many reasons, such as the fact that failed measurements were set exactly to median weight, and that for many foods such as the steak in Figure 3.5 the standard deviation between portions is very small making the median quite accurate. Additionally since measurements over 2 standard deviations from the mean weight were

Food item	True	Pred.
American salad	58	41
pickled cucumber slice	29	34
rice	125	136
chicken leg	227	232
grated carrot	77	0
carrot-cabbage	0	45
tomato-onion	32	0
tomato	0	58

Figure 5.1: Example image of a lunch plate imaged with the system, true weights (in grams) and weight predictions provided by the computer vision model `ResNet101+Menu`. While the model confuses some visually highly similar classes (indicated by the two colored pairs of lines), these mistakes would be largely insignificant for the customer because the nutritional content of the confused classes (e.g. 'grated carrot' and 'carrot-cabbage') is also similar. The image and table are taken from [20].

set to the median weight this will make the median estimator exactly correct for those items further boosting its accuracy. Still, we do manage to beat the weight estimation baseline. Similarly as in classification the classes with large error averages are also mostly classes with 100 samples or less as can be seen in Figure 5.3, meaning that the weight estimator likely would also improve with more data.

Disregarding the classification aspect, the lowest mean weight estimation error is (based on Table 4.4) $22/93 = 24\%$, where 93g is the mean error of the zero prediction baseline which means it is the the mean weight itself. We can estimate the error the model makes for a total plate by multiplying both numbers with the average number of foods (approximately 4) to reach the same error percentage of 24. This is of similar scale as the total mass error of 29.5% in similar parallel work by Thames et al. [40] further validating our work, although they train to predict the total mass of the whole dish instead of the weight of individual items. They further show that incorporating volume estimates based on depth information greatly improves their results to 13.7% MAE error which is to be expected considering the difficulty of only using 2D images for weight estimation.

Many ways to inch out small performance gains exist, but were not covered here. Instead of pretraining the backbone networks with ImageNet, pretraining on food specific datasets could improve performance. Ensembling the model usually increases performance by a

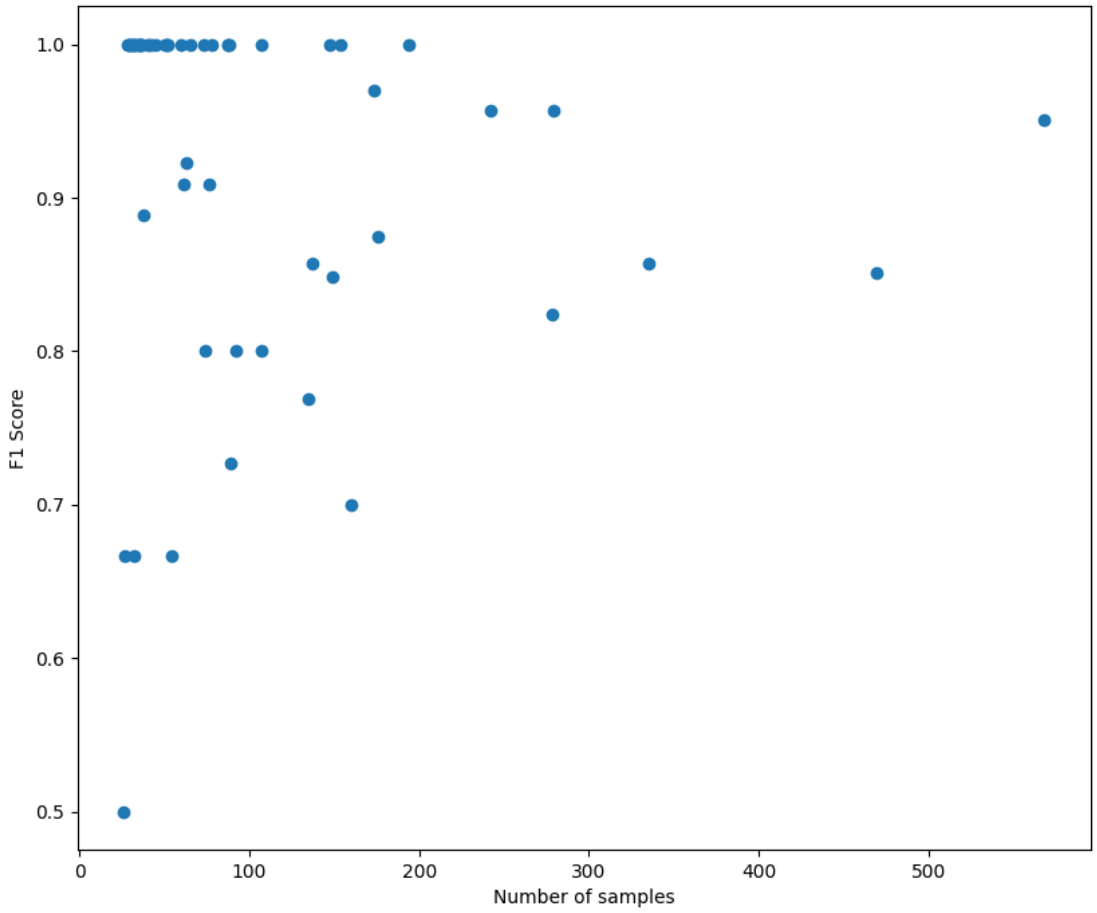

Figure 5.2: Generally more images means better f1 score. All classes with more than 200 images get a f1 score of at least 0.8

little. Taking random crops of the images and changing the saturation of the images randomly could be used to further augment the dataset and possibly increase accuracy.

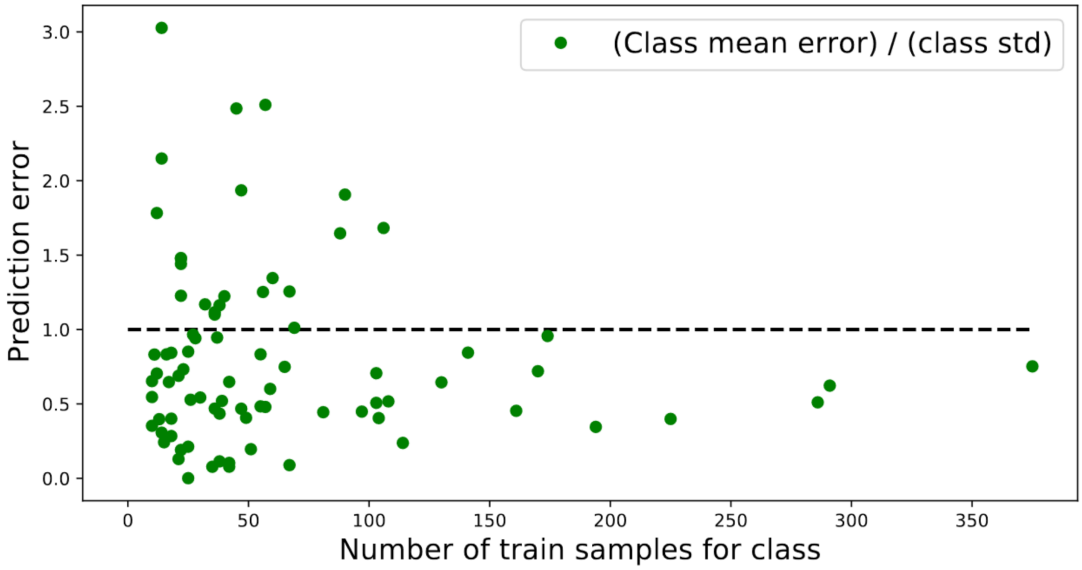

Figure 5.3: Classes with large average errors tend to have less than 100 samples. Here the dotted line represents the standard error, or in other words the average error that would be made by predicting the class mean weight for every item.

6 Conclusion

We described how modern machine learning and computer vision based techniques work and how they can be applied to food weight estimation and classification on a dataset of images and weight labels collected from a buffet-style restaurant. We also showed how failed weight measurements and images could be discarded in an automated fashion, and published a dataset for public use.

We further demonstrated and investigated methods for providing daily menu information to a model during training and showed they improve results. The developed models achieved good classification accuracy and promising weight estimation results especially considering the limited amount of training data.

Put together these parts create a strong starting point for a system for camera-based food identification and weight estimation for buffet-style restaurants. However, many problems remain to be solved. For example, the current model and image preprocessing setup does not take into account different camera and lighting scenarios, and would not generalize to other restaurants. Also weight estimation model is not significantly better than a classifier paired with a naïve mean estimator. Finally, the current system cannot filter out sensitive data placed directly on the plate. These shortcomings could at least partially be addressed by gathering more data, by applying domain adaptation techniques, and by incorporating sensors that can be used to estimate the volume of the food, for example with the help of an additional depth camera.

Acknowledgements

Special thanks to thesis advisors Arto and Chang for all the help and guidance throughout the thesis. Thank you everyone at Flavoria and the University of Turku for creating the weight data collection system and providing the data. Further thanks to all co-authors of the related publication.

The work would not have been possible without Mikko Toivonen creating the camera setup itself, and Ville-Veikko Saari, and Lauri Mäkinen physically installing it in Turku. Furthermore Ville-Veikko automated the camera to take the images, and manually went through all the images and removed those with sensitive data so that the dataset could be published. Ville-Veikko also created the JPG versions of the images from the original RAW images so that the dataset became much more manageable in size while retaining important information. Thank you Lauri Mäkinen for all the discussions, help with writing the paper, holding the conference presentation, and for handling the many practical things needed for getting the dataset published.

Thanks to the CNB study group for being there throughout my university studies and tolerating the monologues. Finally, thank you Hilla for all the emotional support during my work on this thesis and for listening through all the endless rants about it over the course of two years.

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[2] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[3] J. F. Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851. URL: https://doi.org/10.1109/TPAMI.1986.4767851.

[4] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. DOI: 10.1017/9781108679930.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[6] R. O. Duda and P. E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (1972), pp. 11–15. DOI: 10.1145/361237.361242. URL: https://doi.org/10.1145/361237.361242.

[7] T. Ege, W. Shimoda, and K. Yanai. "A New Large-scale Food Image Segmentation Dataset and Its Application to Food Calorie Estimation Based on Grains of Rice". In: *Proceedings of the 5th International Workshop on Multimedia Assisted Dietary Management, MADiMa @ ACM Multimedia 2019, Nice, France, October 21-25, 2019*. Ed. by S. G. Mougiakakou, G. M. Farinella, and K. Yanai. ACM, 2019, pp. 82–87. DOI: 10.1145/3347448.3357162. URL: https://doi.org/10.1145/3347448.3357162.

[8] Flavoria-tutkimusalusta. *Introducing Flavoria, the multidisciplinary research platform of the University of Turku*. Accessed: 2022-10-18. Flavoria. 2021. URL: https://www.youtube.com/watch?v=xzNHCwEZqhg.

[9] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Y. W. Teh and D. M. Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[10] X. Glorot, A. Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. Ed. by G. J. Gordon, D. B. Dunson, and M. Dudık. Vol. 15. JMLR Proceedings. JMLR.org, 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] L. A. Goodman. “On the Exact Variance of Products”. In: *Journal of the American Statistical Association* 55.292 (1960), pp. 708–713. DOI: 10.1080/01621459.1960.10483369. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1960.10483369. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1960.10483369.

[13] J. He, Z. Shao, J. Wright, D. A. Kerr, C. J. Boushey, and F. Zhu. “Multi-task Image-Based Dietary Assessment for Food Recognition and Portion Size Estimation”. In: *3rd IEEE Conference on Multimedia Information Processing and Retrieval, MIPR 2020, Shenzhen, China, August 6-8, 2020*. IEEE, 2020, pp. 49–54. DOI: 10.1109/MIPR49039.2020.00018. URL: https://doi.org/10.1109/MIPR49039.2020.00018.

[14] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://doi.org/10.1109/CVPR.2016.90.

[15] K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123. URL: https://doi.org/10.1109/ICCV.2015.123.

[16] P. V. Hough. “Machine analysis of bubble chamber pictures”. In: *Proc. of the International Conference on High Energy Accelerators and Instrumentation, Sept. 1959*. 1959, pp. 554–556.

[17] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by F. R. Bach and D. M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[18] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[19] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[20] L. Koivunen, S. Laato, S. Rauti, J. Naskali, P. Nissilä, P. Ojansivu, T. Mäkilä, and M. Norrdal. “Increasing Customer Awareness on Food Waste at University Cafeteria with a Sensor-Based Intelligent Self-Serve Lunch Line”. In: *2020 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2020, Cardiff, United Kingdom, June 15-17, 2020*. IEEE, 2020, pp. 1–9. DOI: 10.1109/ICE/ITMC49519.2020.9198571. URL: https://doi.org/10.1109/ICE/ITMC49519.2020.9198571.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. 2012, pp. 1106–1114. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[22] Y. LeCun, Y. Bengio, and G. E. Hinton. “Deep learning”. In: *Nat.* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[23] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Comput.* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541.

[24] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade - Second Edition*. Ed. by G. Montavon, G. B. Orr, and K.-R. Müller. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012, pp. 9–48. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8%5C_3.

[25] S. Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". MA thesis. Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[26] F. P. W. Lo, Y. Sun, J. Qiu, and B. Lo. "Image-Based Food Classification and Volume Estimation for Dietary Assessment: A Review". In: *IEEE J. Biomed. Health Informatics* 24.7 (2020), pp. 1926–1939. DOI: 10.1109/JBHI.2020.2987943. URL: https://doi.org/10.1109/JBHI.2020.2987943.

[27] C. B. Martin, K. A. Herrick, N. Sarafrazi, C. L. Ogden, et al. *Attempts to lose weight among adults in the United States, 2013-2016*. 2018. URL: https://www.cdc.gov/nchs/products/databriefs/db313.htm.

[28] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[29] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar. "Long-tail learning via logit adjustment". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=37nvvqkCo5.

[30] W. Min, S. Jiang, L. Liu, Y. Rui, and R. C. Jain. "A Survey on Food Computing". In: *ACM Comput. Surv.* 52.5 (2019), 92:1–92:36. DOI: 10.1145/3329168. URL: https://doi.org/10.1145/3329168.

[31] A. Myers, N. Johnston, V. Rathod, A. Korattikara, A. N. Gorban, N. Silberman, S. Guadarrama, G. Papandreou, J. Huang, and K. Murphy. "Im2Calories: Towards an Automated Mobile Vision Food Diary". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1233–1241. DOI: 10.1109/ICCV.2015.146. URL: https://doi.org/10.1109/ICCV.2015.146.

[32] M. A. Nielsen. "Neural Networks and Deep Learning". In: *Determination Press* (2015). URL: http://neuralnetworksanddeeplearning.com/chap6.html.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett. 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[35] T. Sarapisto, L. Koivunen, T. Mäkilä, A. Klami, and P. Ojansivu. "Camera-Based Meal Type and Weight Estimation in Self-Service Lunch Line Restaurants". In: *2022 12th International Conference on Pattern Recognition Systems (ICPRS)*. 2022, pp. 1–7. DOI: 10.1109/ICPRS54038.2022.9854056.

[36] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1409.1556.

[37] I. Sobel and G. Feldman. "A 3x3 isotropic gradient operator for image processing". In: *presented at the Stanford Artificial Intelligence Project (SAIL)* (1968).

[38] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313. URL: https://dl.acm.org/doi/10.5555/2627435.2670313.

[39] R. Szeliski. *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science. Springer, 2022. ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9. URL: https://doi.org/10.1007/978-3-030-34372-9.

[40] Q. Thames, A. Karpur, W. Norris, F. Xia, L. Panait, T. Weyand, and J. Sim. "Nutrition5k: Towards Automatic Nutritional Understanding of Generic Food". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 8903–8911.

DOI: 10.1109/CVPR46437.2021.00879. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Thames%5C_Nutrition5k%5C_Towards%5C_Automatic%5C_Nutritional%5C_Understanding%5C_of%5C_Generic%5C_Food%5C_CVPR%5C_2021%5C_paper.html.

[41] W. Willet and M. Stampfer. "Current evidence on healthy eating". In: *Annual review of public health* 34 (2013), pp. 77–95.

[42] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*. Ed. by D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars. Vol. 8689. Lecture Notes in Computer Science. Springer, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53. URL: https://doi.org/10.1007/978-3-319-10590-1%5C_53.

[43] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C.-H. Hoi, and W. E. "Towards Theoretically Understanding Why Sgd Generalizes Better Than Adam in Deep Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/f3f27a324736617f20abbf2ffd806f6d-Abstract.html.

[44] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. "A Comprehensive Survey on Transfer Learning". In: *Proc. IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555. URL: https://doi.org/10.1109/JPROC.2020.3004555.