LANCASTER UNIVERSITY

A systematic exploration of food recognition using computer vision techniques

by

Laurynas Tumosa

A final year project report for
BSc Hons Computer Science (Study Abroad)

School of Computing and Communications

March 2017

Declaration of Authorship

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: _____

Signed: _____

Abstract

The dissertation focuses on computer vision techniques for classifying food from images. Different machine learning algorithms such as k-nearest neighbors classifier, support vector machines, linear classifier, multilayer neural network, convolutional neural network, and transfer learning were explored. Classifiers that use these algorithms were developed and their performance were evaluated and compared to find the method that works best.

Working Documents

The working documents of this project are availabe on https://github.com/laur1s/Dissertation and http://www.lancaster.ac.uk/ug/tumosa/.

Contents

List of Figures

List of Tables

Glossary

ReLU	rectified linear unit
K-NN algorithm	k-nearest neighbors algorithm
SVM	support vector machine
CNN	convolutional neural network

Chapter 1

Introduction

Food is a very important part of everyones life. One of the common sayings about food is "You are what you eat" meaning that eating healthy food like vegetables and fruits will make one healthier and eating fast food is bad for ones body.

Tracking what one eats is really helpful for maintaining a healthy diet or losing weight. Since these days we are living in such a fast paced world it is often hard to keep track of food that one eats during the day. Weight and calorie intake apps for smartphones are getting more popular since they allow users to capture their food intake on the go. However, approach that todays mobile applications use for entering food intake is obtrusive and not very user-friendly. Users are required to open their food diary applications and search for the eaten food using their smartphones' keyboards.

The goal of this project is to explore techniques for an automatic food classification form images. That could enable a user to capture a food he eats in an unobtrusive and user friendly way. A created machine learning model could use a picture of food taken by a smartphone's camera to classify it. In most of the smartphones camera can be accessed in a few clicks. People are used to taking pictures on their smartphones.

However, automatic food classification in not an easy task. There is no direct way to detect what kind of food product is in a picture. the digital image is just a matrix of numbers representing intensity of three different color components: red, green and blue (RGB) it is impossible to create an algorithm that could directly map image to the label of food item. Therefore, a machine learning is

needed to learn a model that can recognize specific type of food from an image and differentiate between various types of food. Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed (Samuel 1959).

Supervised learning is a machine learning task for mapping labeled examples as training data and makes label predictions for all unseen points (Mohri et al. 2012). This project explores various supervised learning algorithms that could be used for classifying the images as well as techniques for image processing for machine learning.

The aims of the project are to explore the machine learning algorithms that could be used for food image classification. Then, these algorithms are going to be systematically evaluated and their performance is going to be compared.

The key challenges, that are going to be faced are generation of food image dataset, preprocessing of images and creating and training of machine learning algorithms.

1.1 The structure of the report

In the next chapter, background theory and research work in this area are going to be introduced. Chapter 3 presents the chosen programming language and machine learning library, as well as evaluation method used to track the performance of the algorithms used. Chapter 4 displays why a dataset of food images were needed, what kind of requirements were set for it and how was it downloaded. In Chapter 5 implementation details of machine learning algorithms are described. Each algorithm is firstly shortly introduced, then implementation details are provided and finally it's classification results are shown. In Chapter 5 a specific type of machine learning- deep learning is explored. Chapter 6 compares the best results of the all classifiers that were built. Finally, in chapter 7 the conclusions are derived.

Chapter 2

Background

2.1 Introduction to the history of computer vision

Computer vision is a relatively old field in computer science. The subject itself has been around since the 1960s, but it is only recently that it has been possible to build useful computer systems ideas from computer vision Forsyth (2011). The birthday of Computer Vision is considered the summer of 1966. During that year a computer vision summer project was proposed in MIT AI lab (Papert 1966). The goal of the summer vision project was to use summer workers (students) effectively in the construction of a significant part of a visual system. It was thought that a vision was a relatively easy field of AI and that a landmark in the development of pattern recognition could be created during the summer. The task, however, was a lot harder than previously expected and no significant computer vision problems were solved during that project.

During development of a computer vision, it was noticed that is very hard to recognise an object by describing the whole image because of the pixel variation in images that are produced under different conditions. Important features have to be selected to make an image more resistible to various variations. Later these selected features are used for detecting a specific object. In 1999 a new method for image feature generation called the Scale Invariant Feature Transform (SIFT) was proposed by Lowe (1999). These features are invariant to image scaling, translation, and rotation, and partially invariant to illumination changes. Because

of that SIFT features performed much better than correlation-based template matching technique that was used before and could be seen as a major break point in computer vision.

Another method for detecting features in images was propoced by (Nakano et al. 2006). This paper states that since objects are composed of a combination of characteristic parts, a good object detector could be developed if local parts specialised for a detection target are derived automatically from training samples. To do this, Independent component analysis (ICA) was used which decomposes a signal into independent elementary signals. Then ICA vectors were applied to the candidate area and their outputs were used in classification. Using this approach face detector algorithm used in Fujifilm camera with real-time face detection was created. It is considered to be a first commercial application that could run computer vision algorithms in real time.

The important turning point in the development of computer vision field was a creation of ImageNet dataset and the ImageNet Large Scale Visual Recognition Challenge in 2010. The challenge is a competition where research teams submit image classification algorithms that classify and detect objects and scenes. The challenge is running annually since 2010 (Russakovsky et al. 2015). In 2010 the goal of the challenge was to estimate the content of photographs for the purpose of retrieval and automatic annotation using a subset of the large hand-labeled ImageNet dataset (10.000.000 labelled images depicting 10.000+ object categories) as the training set. An example images which represent a pizza class can be seen in Figure 2.2.

Because of a big amount of images and categories ImageNet was and still is a huge challenge for researchers who compete in the challenge. ImageNet encouraged improvement and innovation in computer vision techniques. In 2012 ImageNet competition was won by an approach that used convolutional neural networks-AlexNet (Krizhevsky et al. 2012). Since then deep learning became a dominant method for an image classification.

2.2 Why is computer vision a hard problem?

The true challenge to artificial intelligence proved to be solving the tasks that are easy for people to perform but hard for people to describe formally problems

FIGURE 2.1: Example images with a pizza class label from the ImageNet

that we solve intuitively, that feel automatic, like recognising objects in images (Goodfellow et al. 2017).

A human can recognise objects in images under all kinds of variations of illumination, viewpoint, scale, etc. In comparison, computer algorithms are very susceptible to all kinds of variations in images. Major variations in images were defined by Karpathy (2016) as:

1. Viewpoint variation. A single instance of an object can be oriented in many ways with respect to the camera.

2. Scale variation. Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).

3. Deformation. Many objects of interest are not rigid bodies and can be deformed in extreme ways.

4. Occlusion. The objects of interest can be occluded. Sometimes only a small portion of an object can be visible.

5. Illumination conditions. The effects of illumination are drastic on the pixel level.

6. Background clutter. The objects of interest may blend into their environment, making them hard to identify.

7. Intra-class variation. The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

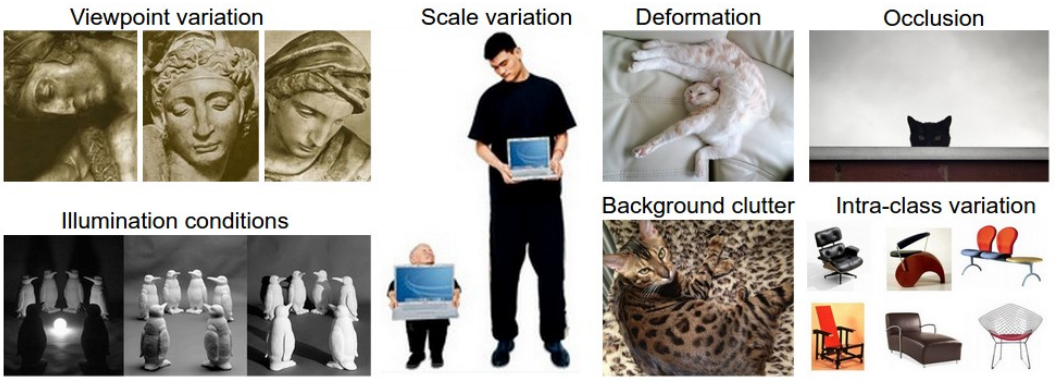

FIGURE 2.2: Example of various variations in images (Karpathy 2016)

The actual solution of the variation problem still has not been found. Currently, this problem is either ignored or some algorithms that are partly resistant to variations are used.

2.3 Saliency-aware food image segmentation for personal dietary assessment using a wearable computer

A computer system that can be used for an image-based dietary assessment was described by Chen et al. (2015). A wearable computer called eButton is used to capture the eating events. The eButton is a small, unobtrusive chest fob that can be pinned to clothing on the chest. Its camera takes pictures automatically at a rate of one picture per second during eating events. Then segmentation techniques are applied to segment a food from a container. Finally, person manually enters the food label to the application. Later food volume is estimated. There is no information about food volume measurement, nutrient database lookup and calculation of calories in the paper despite that these steps are shown in the eButton operational diagram. Figure 2.3

The main focus of the paper is food segmentation from its container. It states that major difficulties in automatic food segmentation are multiple food components

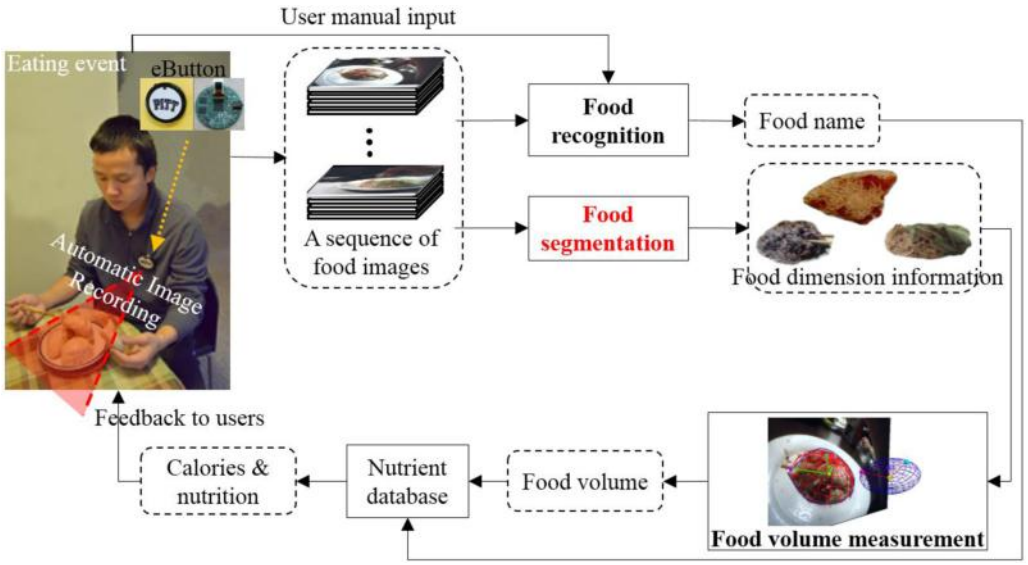

FIGURE 2.3: Personal dietary assessment using a wearable computer eButton.

in complex and varying configurations, coloured decorative patterns on plates and occlusion by non-food objects. The example images of these three difficulties are shown in Figure 2.3.

FIGURE 2.4: Major difficulties in automatic food segmentation

Researchers use two stages approach for food segmentation. Firstly, food container is detected in an image. The container is detected by shape convexity. Canny edge detector is used to obtain the edge information from a given image. Then, a number of square windows centred at edge pixels are randomly selected . A trained support vector machine with the histogram of a gradient as the classifier input is used to discard the non-container edge windows, increasing the detection efficiency.

For detecting the food inside a plate researchers used colour contrast to characterize the difference between the objects in the region against their surroundings, colour abundance to characterise a probability of a colour appearing in the local

window and spatial arrangement to recognise the areas with one highly dominant colour in the image. These 3 characteristics are then combined into one function.

The data set used to evaluate the segmentation approach in this research was combined from 30 eating events captured with eButton and 30 food images from the Jawbone database. Accuracy was measured by visually inspecting the quality of the segmentation and by comparing the image segmented by a computer to the images segmented by two research participants and measuring the difference.

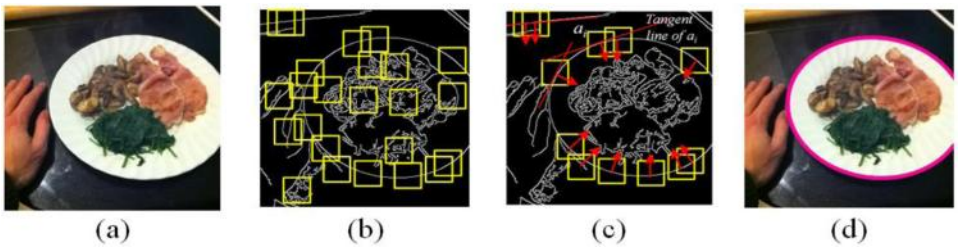

FIGURE 2.5: Container detection

The main limitation of this approach is that in some cases food is served not in a round shaped container e.g.: takeaway food box, lunch box which is usually rectangular. Some of the foods can also be eaten without using any container at all e.g: fruits, vegetables.

2.4 Survey of calorie counter applications

To get a better understanding what are the input methods offered by currently available calorie counter it was decided to try them. To download the apps keyword čalorie countingẅas entered into the search bar of Google Play Store. The top 3 apps in the search results were downloaded and tested. These apps were MyFitnessPal, Nutracheck and Lifesum.

MyfitnessPal is the most popular calories Counting app on the Google Play Store. The app has over 50 million downloads on Google Play Store Figure 2.6. When the app is launched for the first time the user is required to sign up. After entering his email and password the user has to complete a short survey about his goal, physical activity levels, gender, birthday, location, weight and height, desired weight and agree to the privacy policy and terms Figure 2.8. After that, the app calculates target daily calorie intake. For entering food user can scan the barcode of the food or use the text-based search. Food search and barcode scanning only work when

a phone is connected to the Internet. The database of food items is quite big and includes products from many different restaurants. However, adding a meal that was prepared by the user himself is complicated. Every ingredient of a dish must be searched and added separately. The portion size has to be entered using weight measurements e.g.: grams. It is often hard to estimate how much does a meal weight so it is a big disadvantage.

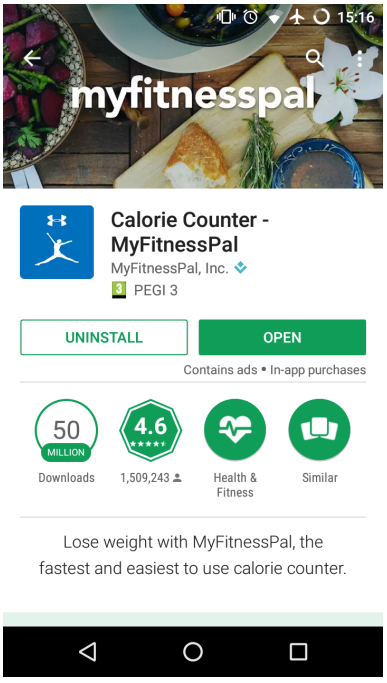

FIGURE 2.6: Google Play Store page of MyFitnessPal app

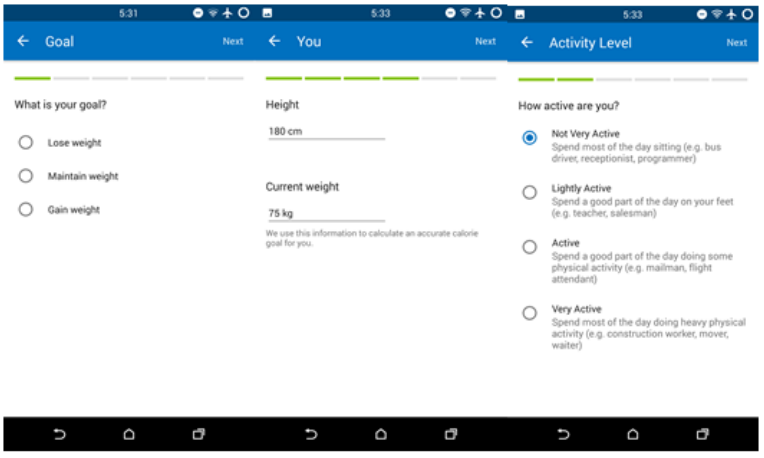

FIGURE 2.7: Procedure to create a MyFitnessPal account

The other app tested was Nutracheck. To start using Nutracheck a user is required to create an account. The registration procedure is very similar to MyFitnessPal's. For entering meals, the app also has a barcode scanner and search function. The

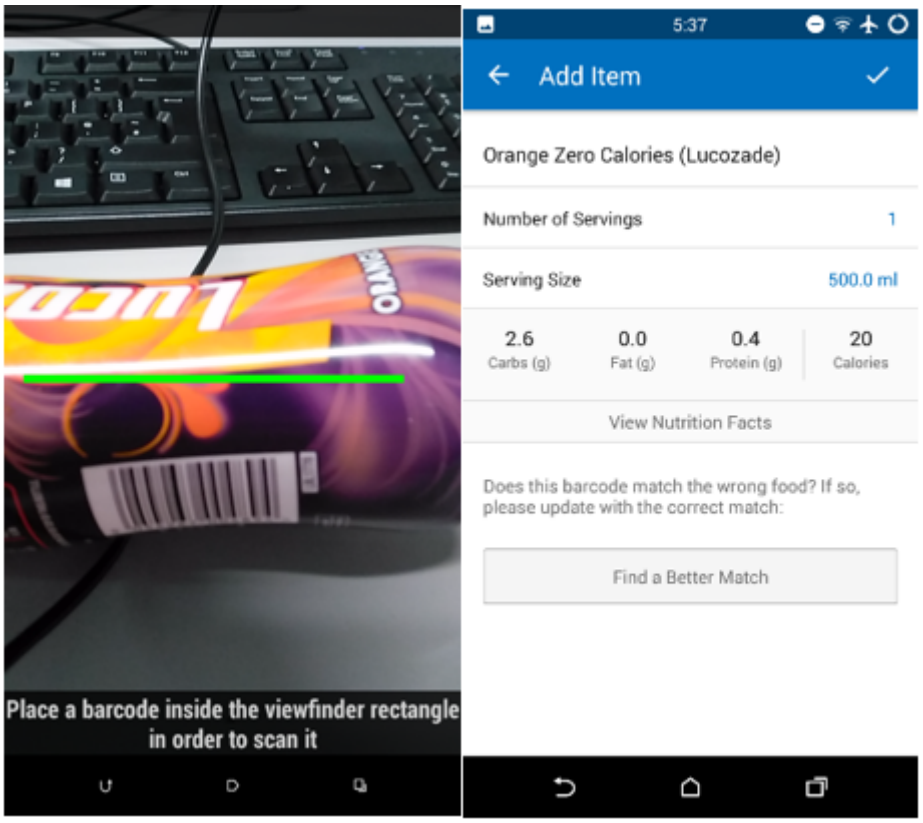

FIGURE 2.8: Barcode scanning with MyFitnessPal

advantage that Nutracheck has over MyFitnessPal is that it displays images of food in the search results Figure 2.9. However, a barcode scanner of this app recognised fewer barcodes that a barcode scanner in MyFitnessPal.

Final app that was tested was Lifesum. It also requires completing a survey in order to start using it and uses a barcode scanner and a search function for adding food. An interesting feature that this app has compared to other apps is food quality rating. It rates a quality of a food according to it nutrition facts and presents the user with a rank of food from A to F Figure 2.10.

From this short survey, it can be concluded that currently, available calorie counting apps lack innovation. They all use text-based search and barcode scanners as data entry points. If one wants to track his food, he must remember what did he eat and estimate how much did It weight. All tried apps require an Internet connection for entering food for tracking. It is clear, that food tracking applications could benefit from a model that can recognize and classify food from images.

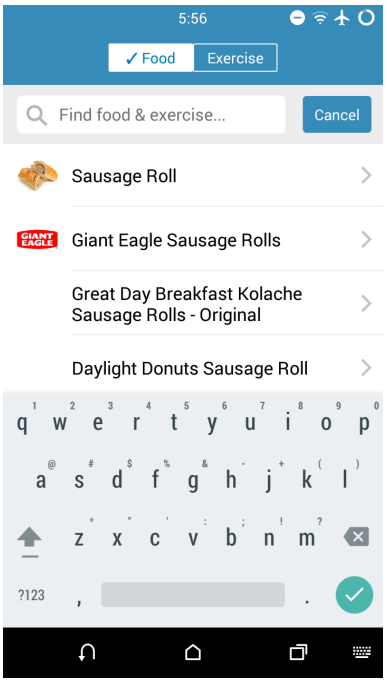

FIGURE 2.9: Food database search in Nutracheck

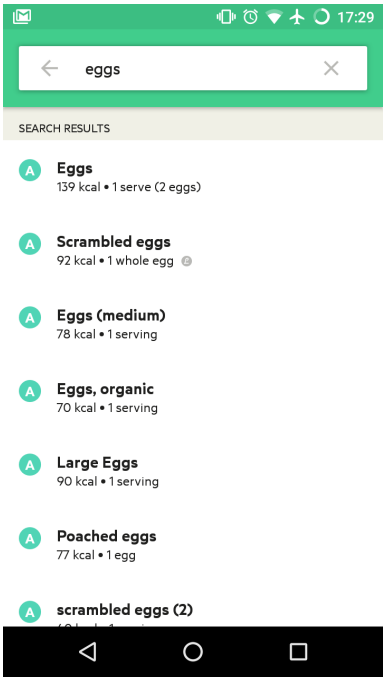

FIGURE 2.10: Food database search in LifeSum

2.5 Summary

In this chapter history of computer vision was reviewed. Then, it was explained why computer vision is such a hard problem. Finally, a paper about automatic food image segmentation was summarized and reviewed. This paper explored

dietary assessment only focusing on segmenting a food object from a background. An automatic food image classifier could extend the capabilities of personal dietary assessment with wearable devices. In the next chapter research methods of the project is going to be discussed.

Chapter 3

Research Methods

This chapter provides information about the research methods used in the project. Before conducting a series of experiments to explore and evaluate machine learning methods, it was needed to decide on how are these algorithms are going to be implemented and evaluated. The list of following things had to be decided first:

1. Choose a programming language.
2. Choose machine learning libraries.
3. Choose the evaluation method for evaluating the performance of machine learning algorithms

The choices made and the reasoning behind them are going to be explained next.

3.1 Choice of a programming language

It was chosen to use Python programming language for implementing this project. The language was chosen because it is one of the most popular languages used by machine learning researchers and developers. It is also free and open sourced. Since Python is an interpreted language, a code written in Python can be executed on multiple different platforms without having a need to change anything in the code. Thanks to Python's popularity, there are many publicly available and open sourced packages with efficient implementation of machine learning algorithms, that can be downloaded.

Availability of scientific computing tools for Python were also one of the main reasons why this language was chosen. Scientific python libraries used in this project were Matplotlib, NumPy, IPython and Jupyter Notebook (Jones et al. 2001–). Matplotlib provided ability to plot graphs, images and tables. NumPy was used for efficient arithmetical operations with matrices. IPython is a System for an interactive scientific computing (Pérez & Granger 2007). It allowed to work with code interactively by allowing to execute parts of code in blocks rather than running the full program. This function was very beneficial because it allowed to experiment with a code in blocks without having a need to re-execute the full code of a program. Some of the operations like loading the dataset, resizing the images in the dataset can take a long time to run, but in Ipython dataset is only needed to load once. Then, it is saved to the memory and can be reused in other blocks of code. Jupyter Notebook is an Ipython wrapper that launches a HTTP server. It allowed to execute Python code in a web browser and to save the results as HTML pages Figure 3.1. Combined together, these libraries allowed to interactively explore the data with ability to plot graphs, tables and to visualize the results.

The alternative language that was also considered was Matlab. Matlab is a domain specific programming language that was designed specifically for matrix programming. It also includes image processing and machine learning toolboxes. Because this language was designed for the specific purpose, machine learning and image processing functions are easier to use in Matlab than in Python. However, Matlab is an closed source language.It also only supports a number of platforms,where Python code can be runned on almost any computing environment. Because of these disadvatages it was decided to use Python.

3.2 Choice of Machine learning libraries

3.2.1 Why there was a need to use a machine learning library

It is possible to implement machine learning algorithms without using any additional libraries. However, there are many benefits provided by using a machine learning library. Algorithms provided in machine learning libraries can often by

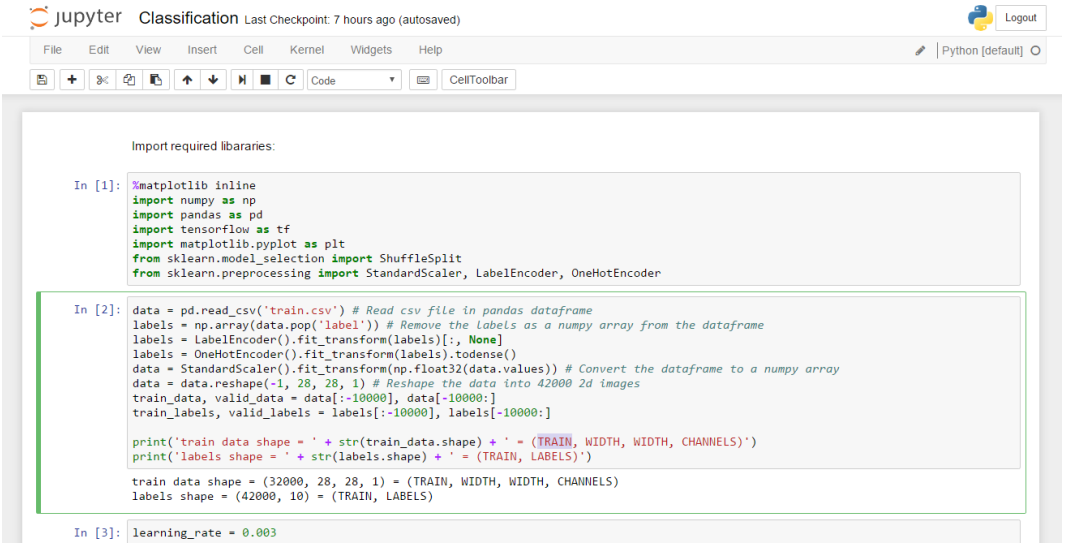

FIGURE 3.1: Example of a Jupyter Notebook

more efficient. It also takes a less time to adapt a provided algorithm for a task then to write a specific algorithm for it. Because of that, it was decided to research what machine learning libraries exists for Python.

3.2.2 Library used for machine learning

Sckit-learn python package was chosen as a library for working with machine learning models. This package provide state-of-the-art implementations of many well-known machine learning algorithms. It also has an easy-to-use interface tightly integrated with the Python language (Pedregosa et al. 2011). Large parts of this module are written in c or C++. Because of the fact that C and C++ are much faster than Python, it makes algorithms run faster compared to a standard Python implementation. Sckitlearn machine learning algorithms are high level API's. Programmer does not need about how these algorithms are implemneted. Nevertheless algorithms provided in this package are very flexible and can be configured to programmer's needs (Buitinck et al. 2013).

In the case of food image classification, the main advantages that Sckit-learn provided were the fact that it used same data input format for every machine learning algorithm and that parameters of algorithms could be easily configured to make classifiers work on image classifcaiton problem.

3.3 The evaluation method for the classification algorithms tried

The main indicator of performance in the classification task is a classification accuracy. Accuracy is calculated by dividing the number of correct predictions by the total number of predictions. By calculating and comparing accuracy scores for every machine learning algorithm tried, the best food image classification technique was found.

Other helpful performance indicators used to evaluate a classification performance when parameters of classification algorithms were tuned were confusion matrix, precision, recall and f measure. Confusion matrix shows how often and what classes are misclassified as other classes, precision it the total number of elements classified correctly divided by total number of elements that were classified as that class, recall is total number of elements classified correctly divided by total number of elements of that class in the database, F-measure is a measure combined from presiscion and recall accoring to a formula $F = 2\frac{Recall \times Precision}{Recall + Precision}$ (Ting 2011).

3.4 Summary

Chapter 4

Accumulation of a Food Image dataset

Before starting to experiment with machine learning algorithms for food classification a dataset of food images needed to be created.The dataset was needed because classification algorithms use dataset to learn a classification model from it.Because of that, a list of most important requirements for the image dataset were created:

1. The images of the dataset should be taken under a real life conditions.
2. There should be enough images in the dataset to learn the model for every machine learning algorithm that is going to be tried.
3. Variance of food images of the same class should be high.

4.1 The requirements for the food image dataset

Because of the intention to create a model that could classify pictures of food taken by people using cameras of mobile phones, it was very important that food images in the dataset could not be created in a lab conditions. People use different angles and distance for taking pictures. The light conditions in images often vary greatly too. It was very important to represent these conditions in the food image dataset.

The dataset also needed to have many images belonging to each class. Having a lot of training samples is the best way to make sure that the model learnt is generalised enough to perform well on a test dataset. Simple models trained on a lot of data perform better than more elaborate models based on less data (Halevy et al. 2009).Because of that it was decided that around 1,000 training images are needed for each class of food in the dataset.

Finally, intraclass class variance of images in the training dataset should also be high. The problem of datasets with low variance can be illustrated with a dataset of tree images containing only images of threes with green leaves.The model learnt by a classifier from this dataset will always classify a tree with orange leaves or a tree without leaves as not tree. Because of that a food dataset should also include different kind of images for a same class e.g. pizza with cheese, pepperoni pizza, slice of pizza, etc.

Accumulation of the dataset that satisfies these requirements was a challenging task. It would take a really long time to take enough photos of meals for creating a big enough dataset. Also, dataset containing images taken by only one or a few persons would not have enough variance. Because of that, possibilities to download food images from the Internet had to be explored.

It was observed that there are a lot of food images hosted on a photo sharing service flickr.com. After reading that Flickr has a Python API that could be used for searching for images according their tags or titles and then downloading these images resized to the specified resolution, it was decided to write a Python script using this API. After downloading the images with a tag "Fish and Chips" it was realized that there were a lot of images that contained this tag but were not actual food images e.g. Figure 4.1. Because of the time it takes to review every picture downloaded and delete images that does not represent the class it was decided to try to find a public food image dataset with images that were already classified.

FIGURE 4.1: Example of an image with a fish and chips tag, downloaded using the Flickr API

It was found that ImageNet classification challenge that was breefly described in section 2.1 includes many categories of food pictures. The only problem was

that the number of images in different classes varies greatly. Some classes like pizza have around 1300 pictures while other classes like salmon loaf have only 360 pictures. This problem was only discovered later when it was observed that when training K-NN classifier chicken with rice class, which had around 2 times less training examples than other classes was selected notably less as a target class of testing images. Because of that 4 image categories which all have around 1300 of images was chosen from the image Net data set. These 4 classes are fish and chips, pizza, salad and vanilla ice-cream.

Four classes of images were picked because the data set required to be small enough to be trained on a laptop computer. Picking four classes of food images allows machine learning algorithms to be trained faster, meaning that various configurations of algorithms can be explored without having to wait very long to see the final result. Also, it was considered that after exploring machine learning algorithms on four class classification problem and seeing the results about which algorithm performs the best it would be a relatively easy task to retrain the algorithm using different data set containing either more classes of food images or different set of classes that can be used for specific application.

4.2 Downloading the data set

Since ImageNet publicly only provides links to the original hosts of image files, Python Script was written to read links from text files, download the images from these links and save them to disk. The first problem that was faced after downloading the images was that many pictures were either corrupted or were template images provided by hosting sites that said that the photo was no longer available. The script was modified to solve this problem by aborting the download and moving on to a next image if a server returned a HTTP redirect code. This code modification solved the problem of blank images appearing in the downloaded data set. However, because of the fact that many images were no longer available to download from the hosting sites, the data set become much smaller than originally planned.

To overcome this problem a permission to access the original ImageNet download files was requested. After agreeing to use the data set only for non-commercial research and educational purposes, the permission to download the database was

granted. It allowed to access the original data set with all images included. The original archives were downloaded using a python method described in a section bellow.

4.3 Pre-processing of the images

Python class named Data was create to handle a creation of the image data set. The class is very flexible and allows images to be loaded using various ways. Constructor of the class takes one argument: the relative path of the folder, where the data is located. Then, images can be loaded from a pickle file located in /pickle folder or can be read directly from a /images folder. It is recommended to save the files to a pickle file after reading the files from a /images folder for the first time. A pickle is a binary file representing a python object. Reading the data set from a pickle takes much less time than reading images one by one because Python only needs to read one file compared to thousands of files while loading images. To be loaded correctly the pickle should be structured as a list of labels of every image, followed by a list of images. Method try_download_images is provided in this class to download the data set from ImageNet. It takes two arguments: user name and ImageNet access key, downloads the archive of images for each image class. Method named extract can be used to extract downloaded archived .tar files into images folder. Before extracting each archive it checks if files already exists and if that is the case extraction is aborted. Method resize_image can be used for resizing images in the dataset. It takes a list with height and width of requested image size and resizes the images. Finally train_test_split method can be used to shuffle the data and split it into training and testing data sets. It takes an optional argument- a fraction that shows percentage of data to be used for testing the classifier.

Chapter 5

Machine Learning Algorithms for food image classification

This chapter describes actions taken to implement food image classifiers. Each algorithm is firstly briefly introduced, then the details about how it was implemented are provided and finally the results of image classification are shown.

5.1 K-nearest neighbours classifier

5.1.1 Introduction to K-nearest neighbours algorithm

K-nearest neighbours is one of the simplest machine learning algorithms that can be used for a classification task. To train the classifier, all that needs to be done is saving the training data samples to a computer memory. When deciding a class for a data sample with an unknown class, the classifier calculates the distance between the data sample and every other sample in the data set. Then, the classifier picks the class of the sample according to class labels of k other samples that are located closest to it. Illustration of k-nearest neighbours algorithm performing a classification on an data entity with an unknown class is shown in Figure 5.1. The rationale behind such a method is based on the assumption that the features that are used to describe the domain points are relevant to their labels in a way that makes close-by points likely to have the same label (Shalev-Shwartz & Ben-David, 2014).

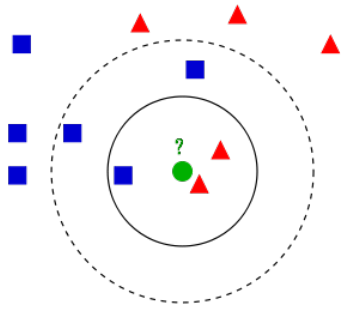

FIGURE 5.1: Example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle) *k-nearest neighbors algorithm* (2017).

5.1.2 Implementation of a K-nearest neighbours classifier

To start experimenting with K-nearest neighbours classifier, its module was imported from the Sckit-learn package. The KNN classifier provided in this package provides many different parameters to choose from. It was decided to try to run classification with images resized to 50x50 pixels. The images also needed to be flattened. Flattening is a process of converting a multidimensional matrix into a vector. To flatten all images in the dataset, the height, weight and colour channels of images, were merged into one channel.

In order to later test the classification performance, the dataset needed to be split into the separate training and testing datasets. The training dataset, as the name suggests, is used for algorithm to learn the model and the test dataset is used only for evaluating the performance of the model. At first it was decided, to use 70 percent of the randomly shuffled dataset as the training data and 30 percent as the test dataset (3168 training samples and 1359 test samples). That means that there were around 790 pictures for each of the four classes. After training a classifier it was noticed that it performed rather poorly. Therefore it was decided to increase the number of images in the training dataset to 90 percent of the dataset. In this case, there were 4698 examples of training images and 522 examples of testing images.

To improve the classification accuracy it was tried to examine with what K number of neighbours the classifier worked best. After running the algorithm multiple times with different values of K it was observed that the highest classification

score was achieved with a K value = 18. However, the best accuracy score which was 36.28 percent was not a very good result and it was necessary to improve the classifier.

To test if having larger image sizes would help the K-Nearest Neighbours classifier to perform better, images in the dataset were resized to 100x100 pixels resolution. When tested on k= 18 neighbours, the classifier showed the improved accuracy to 45.59 percent. To get a better understanding about how did the classifier performed a confusion matrix was plotted. From the confusion matrix Figure 5.2, it can be seen that Vanilla ice cream was classified correctly most often from the 4 classes. The accuracy of predicting it was 78.4 percent. The worst classified class was a salad which achieved only 0.02 percent accuracy.

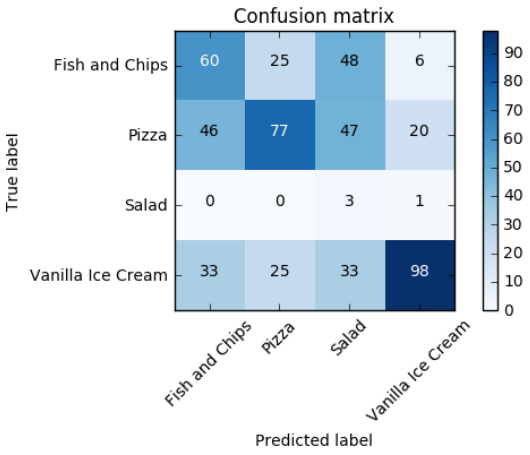

FIGURE 5.2: Confusion Matrix of KNN classifier with k= 18

5.1.3 Tuning of Hyperparameters

Hyperparameters of a machine learning algorithm is a set of parameters that algorithm uses and that have to be specified while training the algorithm e.g.: K-amount of neighbours considered in the KNN classifier. Traditionally, hyperparameters were specified by humans, but the development of computing technology currently allows the use of various search techniques for better parameter optimization.

There are two methods available for finding the best hyperparameters in the Sckit-learn package. These are an exhaustive grid search - GridSearchCV and a randomized grid search - RandomizedSearchCV. The Exhaustive grid search tries to

run an algorithm with all combinations of hyperparameters provided. The advantage of this approach is that the algorithm will always find the best combination of hyperparameters. However, exhaustive search algorithm has try every single combination of hyperparameters before choosing the best set. It can be very computationally expensive and take a long time. Randomized search is usually able to find the set of parameters that can perform roughly equivalent than a grid search. The fact that that randomized optimization could be a better optimization technique was proven by Bergstra and Bengio who empirically and theoretically proved that randomly chosen trials could be more efficient for hyper-parameter optimization than trials on a grid Bergstra & Bengio (2012).

For the K-nearest neighbours classifier, it was decided to optimize the following hyperparameters: a number of neighbours, a weight function that is used, and a distance function used. The number of neighbours considered were from k=1 to k=30, weight function was either uniform meaning that during the classification the influence of all k neighbours is the same when determining the class of the target image, or distance based, meaning that closer neighbours are more influential when the class of the target image is picked. For the distance function Manhattan and Euclidean distance were considered.

It was decided to try running the exhaustive grid search first to get the set of hyperparameters that works best for the classifier.Because hyperparameters sets are independent from each other, it was possible to run hyperparameter optimization task in a parallel,using multiple cores of a CPU. Still, it took 30 minutes to run the exhaustive search on selected hyperparameters. It was found that the algorithm was most accurate with k = 8 neighbours, the Euclidean distance function, and uniform weights. Then, the algorithm was tested on the test dataset and classified it with 56.32 percent accuracy. More than 10 percent increase in accuracy was a considerable improvement from the previously achieved score.

From a plotted confusion matrix (Figure 5.3) it can be seen that the optimized KNN classifier was able to predict both pizza and ice cream classes very accurately. This fact is also confirmed in a precision, recall and f score table Table 5.1. Average precision score of these two classes was around 0.8, however the recall was lower - 0.53. The recall was low because the classifier missclassified a lot of images from different classes as pizza and vanilla ice cream.

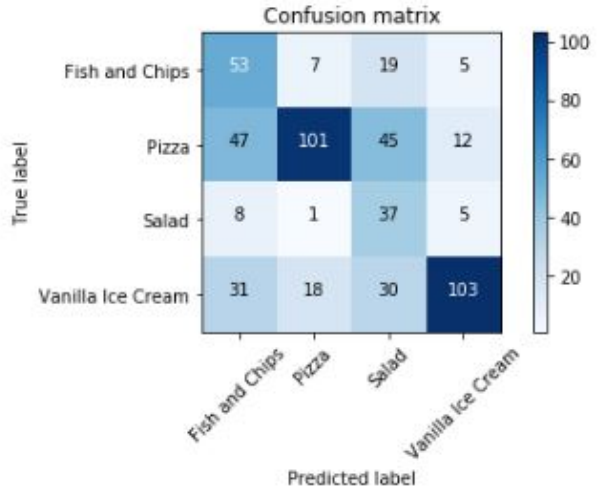

FIGURE 5.3: Confusion Matrix of KNN classifier with k= 8, distance = Euclidean, weights= uniform

Class	precision	recall	f1-score
Fish and Chips	0.38	0.63	0.48
Pizza	0.80	0.49	0.61
Salad	0.28	0.73	0.41
Vanilla Ice Cream	0.82	0.57	0.67
Average	0.69	0.56	0.59

TABLE 5.1: the precision, recall and f score of the best K-NN classifier

5.1.4 Results

To conclude, with right optimization of hyperparameters K- nearest neighbours classifier can classify images quite accurately. Images of a pizza and a vanilla ice cream class were classified much better compared to the images of the other two classes. The most misclassified class was a salad class. The reason for that was the fact that there were images of various kinds of salad in the dataset. Because of that variance, salad class images were very scattered in the prediction space, and the classifier was unable to find the right neighbours.

The advantage of the classifier it that it can be trained in a very short time. However, it is was very slow when predicting the class of an unseen image. Because of that, different classification algorithms needed to be tried.

5.2 Support Vector Machine

5.2.1 Introduction to Support Vector Machine

Support vector machine is the one of the most influential machine learning algorithms Boser et al. (1992). The algorithm searches for separators between classes that maximizes the distance between them. Advantages of this classifier is that it can work very well on high dimensional data. Support vector machine tries to find a separating hyperplane for each class such that distances between separators and each class are as large as possible. A figure showing how a support vector machine chooses a hyperplane for separating two classes can be seen in Figure 5.4. Because of the fact that this algorithm works well for a high dimensional data, it was decided to implement a classifier using SVM algorithm.

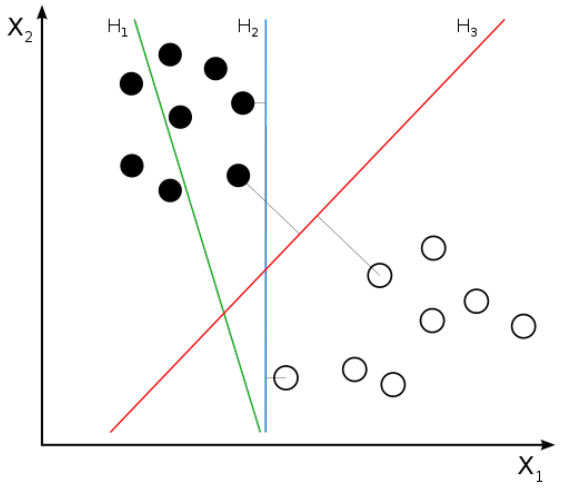

FIGURE 5.4: Three hyperplanes are shown that could be used to split the data into two classes. H1 does not separate the classes. H2 does, but only with a small margin. H3 separates them with the maximum margin so it is selected as the class separator *SVM* (2017).

5.2.2 Implementation of a SVM classifier

To start experimenting with support vector machine, its module was imported from the Sckit-learn package. To understand how the classifier works it was attempted to run the algorithm on our food data set resized to 100 x 100 pixels. At first it was decided to try to run it with a default set of hyperparameters.

The training of the SVM classifier took around 60 minutes. When it was tested on test image data set, the accuracy achieved was only 25 percent. To get a better understanding why accuracy score was so low, a confusion matrix was plotted Figure 5.7. The confusion matrix showed that for some reason the classifier classified all the test dataset as salad. The hypothesis why the support vector machine classified all the data into the one class was that a decision surface that it learnt was to simple.

The complexity of decision surface of SVM classifier is controlled by a hyperparameter C and gamma. C parameter describes by how much the classifier should avoid missclassifying the training data. A low C makes the decision surface smooth but missclassifies more training data, while a high C aims at classifying all training examples correctly. (*RBF SVM parameters* 2017). The Gamma parameter defines the influence that a single example has on the decision boundary. It was decided, to test what set of C and gamma could help to improve the classification accuracy.

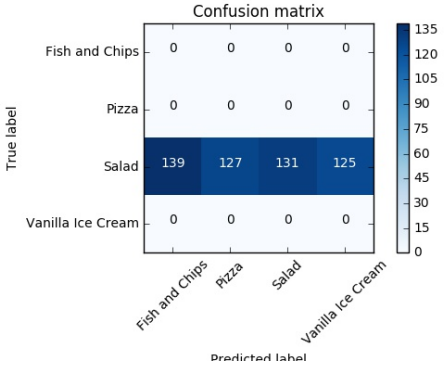

FIGURE 5.5: Confusion Matrix of Support Vector Machine Classifier trained with default parameters

5.2.3 Tuning of SVM hyperparameters

Because previous SVM model took a really long time to train, it was decided to use only 10 percent of the training dataset to find best hyperparameters for this algorithm.

Firstly, it was decided to increase the value of C. It was increased to the value of 10. With C=10 the algorithm trained on 10 percent of the dataset classified 26 percent of the test data correctly. After plotting the confusion matrix it was observed that now the classifier predicted all classes Figure 5.6 but it was very

inaccurate in predicting.The C was increased to 100 but that did not improve the classifier.

Finally, best set of C and gamma hyperparameters was found with randomized grid search. C values of $10, 100, 1000, 5000, 10000$ and gamma values of $0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1$ were tried. Grid search found that C=10 and gamma = 0.0001 were the best set of parameters. It was reasonable that classifier worked best with the lowest value of gamma because images of food vary a lot and because of that a single image should not have too much influence on the separation lines between classes. It was tried to train the algorithm using a full train data set.

SVM was trained on the whole training dataset with C = 10 and gamma = 0.0001. Despite the fact that the hyperparameters were optimal, accuracy of the classifier on the training set was only 25 percent. Plotted confusion matrix **??** shown that the model still classified all test dataset as salad class.

It was deducted that SVM algorithm was unable to classify the dataset correctly, because number of features considered when dividing the hyperplanes was too high. It was decided to use principal component analysis to reduce the number of features, and test if that solves the problem.

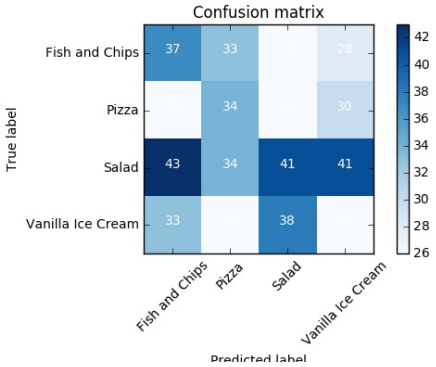

FIGURE 5.6: Confusion Matrix of SVM trained with c = 10
Numbers in squares are: [37 33 26 28], [26 34 26 30], [43 34 41 41], [33 26 38 26]

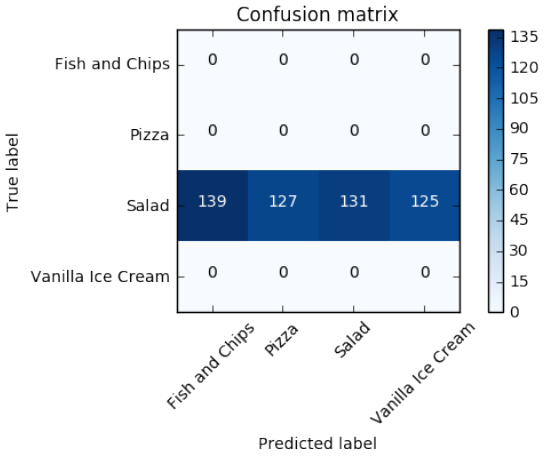

FIGURE 5.7: Confusion Matrix of SVM with C = 10 and gamma = 0.0001

5.2.4 Reducing dimentionality with principal component analysis

Principal component analysis is a simple, non-parametric method for extracting relevant information from confusing datasets (Shlens 2014). It automatically detects strong patterns in the data and ignores the noise. Because food images contain many patterns and are usually noisy, PCA should be a good solution.

The number of features were reduced from 30000 to 150 using this technique. Then, randomised grid search was performed to find the best C and gamma values for this support vector machine. It was found that algorithm classified the training data best with C = 1000 and gamma = 0.0001. After training, it was tried to classify the test dataset. The classifier achieved 60.73 percent accuracy which was the highest score compared to previously tried algorithms. Then a confusion matrix and a classification report were plotted.From the confusion matrix displayed in Figure 5.8, it can be seen that all classes except the salad class were correctly predicted around 83 times.The most common missclassification was a fish and chips class misclassified as a pizza.From the classification report displayed in Table 5.2 it can be seen that a vanilla ice cream class had the highest precision and recall. The classifier predicted most of the ice cream pictures correctly. Because of that it can be concluded that there was a strong separator between vanilla ice cream class and other classes.

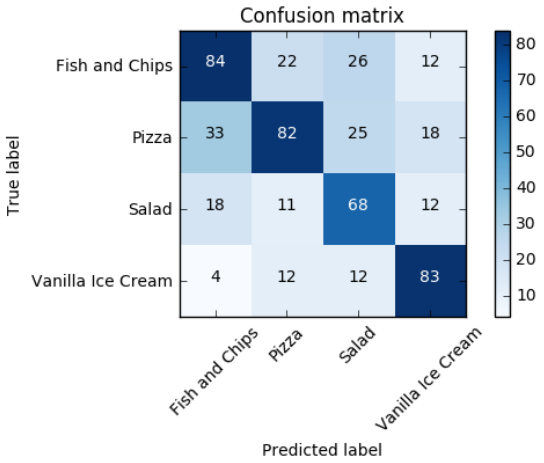

FIGURE 5.8: Confusion Matrix of SVM classifier on the dataset reduced with PCA

Class	precision	recall	f1-score
Fish and Chips	0.60	0.58	0.59
Pizza	0.65	0.52	0.58
Salad	0.65	0.52	0.58
Vanilla Ice Cream	0.66	0.75	0.70
Average	0.61	0.61	0.61

TABLE 5.2: A prescion, recall and f scores of the SVM classifier on the PCA reduced dataset

5.3 Conclusions

To conclude, test accuracy of averagely 60 percent achieved by both k-nearest neighbors and support vector machine with PCA models proved that machine learning models could work on classifying the food images. However, 60 percent accuracy is not good enough for using this model to classify the food in real life. Because of that other classification methods are going to be explored.

Chapter 6

Deep Learning approach for the food image classification

6.1 Short introduction to deep learning

Deep learning is often regarded as an exciting and new technology. In reality, the field dates back to 1940s. The earliest predecessors of modern deep learning were simple linear models motivated from a neuroscientific perspective. These models were designed to take a set of n input values $x_1, ..., x_n$ and associate them with an output y. These models would learn a set of weights $w_1, ..., w_n$ and compute their output $f(x, w) = x_1 w_1 + ... + x_n w_n$ Goodfellow et al. (2017). The same function is used in todays deep neural networks. A deep neural network contains a set of these functions connected in a way where the output of the previous function is the input to the next function. The result of $f(x, w)$ is then passed through the non-linear activation function to make the result non-linear. The most commonly used activation function today is a Rectified linear unit(ReLU). The function can be mathematically described as $f(x) = max(0, x)$ which means that it always outputs 0 for values less than 0 and x for values greater than 0. Illustration of this is shown in Figure 6.1. Neural networks using this function usually converge faster than networks using a different activation function (Nair & Hinton 2010).

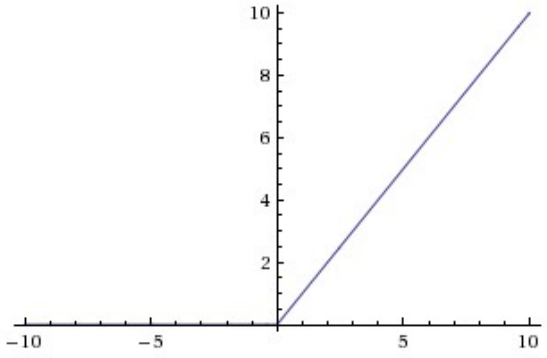

FIGURE 6.1: The ReLU activation function

6.2 Introduction to TensorFlow

TensorFlow is an interface for expressing machine learning algorithms and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms for deep neural network models. It has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields Abadi et al. (2016). Computations in TensorFlow are described in a directed graph. First, all operations are listed and added to the graph, then a TensorFlow session is called and executes the operations in a graph. An example of a TensorFlow program that adds two scalars and computes the result can be seen in Figure 6.2. Executing tf.add(5, 5) creates a node in the computational graph with 5 and 5 as the input values. The result of the add operation is only computed when a TensorFlow session is created.

It was decided to use TensorFlow to construct deep neural network models for food image classification because this library is fast and efficient. It is used by many deep learning researchers. However, because TensorFlow was created with primary deep learning researchers in mind, algorithms provided in this library are low level, and require additional implementations to make them work on a specific dataset.

Because of the complexity and a low-level implementation, it was a very challenging task to learn and understand how to use TensorFlow correctly. To get familiar

with TensorFlow, it was firstly tried to implement deep learning models for the default datasets that were provided in the TensorFlow library. After spending a significant amount of time studying deep learning methods and writing implementations of deep learning algorithms, it was eventually learnt what were the steps needed to train a neural network model with TensorFlow. However, before training the models the dataset, needed to be transformed into a format supported by TensorFlow.

```
import tensorflow as tf
a = tf.add(5, 5)
print(a)
```

```
Tensor("Add_3:0", shape=(), dtype=int32)
```

```
with tf.Session() as sess:
  res = sess.run(a)
  print (res)
```

```
10
```

FIGURE 6.2: Example of a TensorFlow program that computes the result of adding two scalars

6.3 Transformation of the dataset for using it with TensorFlow

Before using the food image dataset, additional preprocessing techniques needed to be applied. The class labels of this dataset were represented by strings e.g.: "Pizza". Non-numerical class labels cannot be used when computing a distance function or other functions that are used in machine learning. Therefore, class labels had to be transformed using one-hot encoding. One-hot encoding transforms every class label into a vector which has the length equal to the number of classes. For every class, one element in a vector is set to one while other elements are set to zeros. The previously used Sckit-learn library encoded the labels automatically, but in TensorFlow this task has to be done manually.

To one-hot encode, the labels of the dataset were first transformed into numeric values from 1 to 4. After that a vector equal to the number of classes was created and numeric values were transformed to one hot encoded values Table 6.1. After the transformation of the dataset was done, it was possible to use it for training deep learning networks with TensorFlow.

Label	Numeric label	One-hot Encoded label
Fish and Chips	1	[0, 0, 0,1]
Pizza	2	[0, 0, 1,0]
Salad	3	[0, 1, 0,0]
Vanilla Ice Cream	4	[1, 0, 0,0]

TABLE 6.1: Applying labels encoding to the dataset

6.4 Linear classifier in TensorFlow

It was decided to construct linear classifier as a first TensorFlow classification model. This classifier was chosen because it is very simple compared to other neural networks. Furthermore, a linear classifier is a building block of every deep learning model, so it was important to understand it well before building more complex classification models

6.4.1 Introduction to a linear classifier

Linear classifier can be described as a function $y = w \times X$. In a case of image classification, y is a class of an image, X is a vector of pixels of an image and w is a vector of weights that are associated with each pixel. The fact that each pixel has a separate weight associated with it allows this classifier to learn the important areas of pictures in each class automatically. The classifier can calculate which parts of image have a large influence when classifying an image and make the weights of these pixels large. The weight of pixels that have no or very little influence to the class of an image can be set to values close to zero.

6.4.2 Training of the linear classifier

To begin training the model,the weights were initialized with random values from a generated normal distribution with 0.01 standard deviation. After the weights were initialized, the classifier tried to classify the data with the initialized weights. Then, the error function was calculated and the weights were updated to the new values that decreased the error function. The update rate value was controlled by a learning rate. The learning rate that was used to train this model was 0.005.

To perform the weight optimization a gradient descent and Adam optimizer were tried.

Gradient descent is an algorithm that tries to minimize the training lost and the Adam optimizer is a gradient descent optimization algorithm Kingma & Ba (2014). Adam works well in practice and compares favorably to other optimization methods.Advantage of Adam optimizer is that it can start optimising weights with a high learning rate and then automatically decreases it as the algorithm continues to train. Consequently, Adam optimizer can find the optimal weights faster than a gradient descent.

It is possible to train an optimization algorithm on full dataset or use batch optimization. For using batch optimization the training dataset has to be split into batches of equal size. Then, optimization algorithm runs on the one batch of the dataset, updates the weights of the full dataset, and uses a different batch for optimizing the weights again. The advantage of using a full dataset for weights optimization is that more data usually leads to a better weight optimization. However, running this algorithm can take a very long time. Usually, batch optimization is faster and can achieve similar results.

To do a batch optimization the food image dataset was split into 54 batches containing 87 food images each. From all trained linear classification models the classifier using the Adam Optimizer with 0.05 learning rate and the batch optimization performed the best. It achieved 51.53 percent accuracy. A table of classifiers tested and their performance can be seen in Table 6.2. To be able to use these models without having to retrain them, they were saved to a disk.

6.4.3 Results

From the results table, it can be observed that optimization on the full dataset performed very poorly compared to the batch optimization method. This happened because the weights on batch optimization were updated 56 times more often. It can also be observed that Adam Optimizer was a better optimization algorithm than a gradient descent. It performed 7.09 percent better.

To conclude, despite the fact that a linear classifier is such a simple algorithm, it was possible to optimize it to classify the images with a reasonable accuracy.

Learning Rate	Optimization Function	Batch Size	Classification Accuracy
0.005	Adam Optimizer	85	51.53%
0.005	Adam Optimizer	full dataset	26.00%
0.005	Gradient Descent Optimizer	85	44.44%

TABLE 6.2: The influence of optimization function and its hyperparameters to the classification accuracy

6.5 Multi-layer neural network in TensorFlow

After building a linear classifier it was decided to expand it to build a deep neural network, that could learn a better representation of the data. The structure of the network was designed in a following way: 30.000 neurons in the input layer, two hidden layers containing 256 neurons each and a final layer containing 4 neurons that map to the 4 image classes.A ReLu activation function was used in hidden layers 1 and 2 to introduce a non-linearity to the the network.

After starting to train the network, it was observed that after couple of training epochs the training accuracy stopped improving and stayed the same during all the duration of training. When tested on the test dataset, the classifier classified the food images with only 24 percent accuracy, which was even worse than a random guessing. It was though, that the problem was with the learning rate being to high, but decreasing the learning rate did not solve the problem. Then, it was tried to change the structure of the network by changing the number of layers in the network and number of neurons in each layer, but modifying the structure did not solve the problem either. Various sets of different hyper parameters were tried to fix the network but nothing seemed to work. It was finally noticed that the problem was caused by a wrong activation function used in a final layer. TensorFlow library did not provide any warning about the error making it hard to detect the problem.

After fixing the error, the classifier was trained using the Adam optimizer with a learning rate of 0.0001. When tested this classifier achieved 56.32 percent accuracy. It was noticed that during the training of this classifier, the cost function decreased really slowly. Because of that, it was decided to increase the learning rate to 0.001. However, this learning rate seemed to be too large and the classifier accuracy decreased to 47.50 percent. Then 0.0005 learning rate was used. The network trained with this learning rate classified the test data with 54.02 percent accuracy.

After testing the different learning rates, it was concluded that 0.0001 was the optimal learning rate for this neural network.

Finally, it was decided to try to increase the number of layers in this neural network. It was expected that a deeper network would be able to learn a better representation of the dataset. Five layer neural network model with four hidden layers that have 256, 256,256 and 128 neurons classified the test dataset with 56.32 accuracy. This accuracy was the same as achieved with a three layer neural network. It was tried to improve accuracy by changing the number of neurons in different layers. Neural network that had less neurons in the final hidden layer performed a bit better but no other significant improvements for classifying the test dataset were noticed. A full table of multilayer classifiers that were trained and their performance can be seen in Table 6.3.

Learning Rate	Number of layers	Neurons in each layer	Accuracy
0.0001	3	256, 256, 4	56.32%
0.0005	3	256, 256, 4	54.02%
0.001	3	256, 256, 4	47.50%
0.0001	5	256, 256, 256, 128, 4	56.32%
0.0001	5	256, 256, 256, 64, 4	58.81%

TABLE 6.3: The influence of optimization function and its hyperparameters to the classification accuracy

6.5.1 Results

To conclude, a multi-layer neural network performed a bit better than a linear classifier. The best neural network design was a 5 layer neural network with 30,000 input neurons, 256 neurons in the each hiden layer and 4 neurons in the output layer. When trained with a 0.0001 learning rate it achieved 58.81 percent accuracy on the test dataset. The achieved accuracy was 7.28 percent better then the accuracy of the best linear model.

However, the design process of multi-layer neural networks was much more complicated. No formal definitions that state how many layers a network should have or how many neurons should be in these layers exist. Therefore, a lot of experimentation is needed to find an optimal network design. Training of multi-layer neural networks is also more computationally expensive and takes a longer time.

6.6 Convolutional neural networks in TensorFlow

6.6.1 Introduction to convolutional neural networks

A convolutional neural network (CNN) was introduced in 1989 to solve a problem of classifying handwritten digits LeCun et al. (1989). A typical layer of a convolutional neural network consists of three stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run trough a nonlinear activation function, such as rectified linear activation function. In the third stage, a pooling function is used to replace the output of a certain location in the network with a summary statistic of the nearby outputs Goodfellow et al. (2017). Most commonly used pooling operation in convolutional neural networks is max pooling Zhou et al. (1988). It converts the rectangular area of the output into one value representing the maximum value in that output (Figure 6.3). For training a convolutional neural network only preprocessing that has to be done is converting each image to the same size.

Convolutional neural networks work by extracting the features from images automatically. This approach tends to work better than a manual feature engineering.

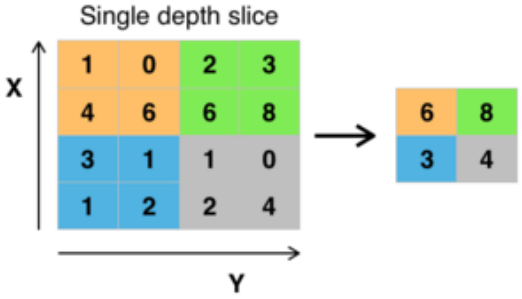

FIGURE 6.3: Illustration of the max pooling operation, the function outputs a maximum output in every 2 by 2 square (Wikipedia 2017)

6.6.2 Implementation of convolutional neural networks

Training a convolutional neural network requires a lot of parallel computing power. Therefore, GPU cards are used for training convolutional neural networks. Since, a computing device with separate GPU unit was not available, it was decided to focus on smaller CNN architectures that could be trained on a CPU.

It was chosen to design a convolutional neural network with three convolutional layers and one fully connected layer. First convolutional layer applies 1 by 1 convolution to the input image and creates four convolution images. These convolution images are then passed through an activation function at outputs of this function are passed into a second convolutional layer. Besides convolution a max pooling operation is also applied in the second and third layers of the network. The final convolutional layer outputs twelve 7 by 7 convolutional images. These images are passed into the fully connected layer and finally the class of an image is calculated in the final layer Figure 6.4

```
# The model
Y1 = tf.nn.relu(tf.nn.conv2d(X, w1, strides=[1, 1, 1, 1], padding='SAME') + b1)

Y2 = tf.nn.conv2d(Y1, w2, strides=[1, 1, 1, 1], padding='SAME')
Y2 = tf.nn.max_pool(Y2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],padding='SAME')  # reduce to 14x14
Y2 = tf.nn.relu( Y2+ b2)

Y3 = tf.nn.conv2d(Y2, w3, strides=[1, 1, 1, 1], padding='SAME')
Y3 = tf.nn.max_pool(Y3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],padding='SAME')  # reduce to 7x7
Y3 = tf.nn.relu(Y3+ b3)

# fully connected layer
fc = tf.reshape(Y3, shape=[-1, 7 * 7 * l3])

Y4 = tf.nn.relu(tf.matmul(fc, w4) + b4)
Ylogits = tf.matmul(Y4, w) + b
logits = tf.nn.softmax(Ylogits)
```

FIGURE 6.4: The convolutional network model in written TensorFlow

Firstly, the images of the dataset were resized to 26 by 26 pixels to reduce the dimensionality of the data. It was unknown how long it would take to train the network so it was decided to reduce the dimensionality even further by converting images to greyscale.

Then, a convolutional neural network was trained for 100 epochs. The best learning rate that was used in the deep learning network (0.0001) was too low for the convolutional network. The learning rate was gradually increased and it was observed that learning rate 0.003 suited the network best. The network with this learning rate achieved 100 percent accuracy on the training dataset in 68 epochs. Despite the fact, that the result looked very impressive, it was known that convolutional neural networks tend to overfit the data. That means that they learn to classify the training dataset very accurately but do not classify the test dataset well. Therefore, classifier was tested on the test dataset. It achieved 38.12 percent accuracy, which was not that good.

To test, if colour channels have an influence on the classification accuracy, the algorithm was modified to accept RGB images. After the network was trained it was observed that it achieved 47.51 percent accuracy on the test dataset. More than 9 percent increase in accuracy showed that colour is an important component for a food image classification and should be kept if possible. The architecture of both convolutional neural networks trained is shown in Table 6.4.

Input image	Learning Rate	Number of layers	Architecture	Accuracy
28x28x1	0.003	5	3 x conv, fc, o	38.12%
28x28x3	0.003	5	3 x conv, fc, o	47.50%

TABLE 6.4: Details about the first two convolutional models tried
conv - convolutional layer, fc - fully connecteed layer, o - output layer

6.6.3 Improving the Convolutional Neural Network

Since, both classifiers achieved a hundred percent accuracy on the training dataset and performed poorly on the test dataset, it was clear that the classifiers were overfitting the data. To fix this problem drop out layers were introduced to the networks. Drop out is a simple way to prevent neural networks from overfitting Srivastava et al. (2014). The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. The drop rate chosen was 50 percent. The Introduced drop layer helped to prevent the network from overfitting. However, because of the large drop out ratio the network was unable to optimize the weights and it classified worse than previously trained models. It was tried to increase the learning rate. This solution did not work because larger step size caused the optimization algorithm to keep overshooting the minimum. The other tried solution was increasing the probability to keep neuron connections to 80 percent. It improved the classification accuracy for both grayscale and RGB image classification but improvement was only marginal (49.04 percent accuracy for RGB test data and 40 percent accuracy for the grayscale image classifier). Lastly, it was tried to keep 90 percent of the neuron connections in the network but it decreased the accuracy on the test dataset. It was concluded that learning rate 0.003 and 20 percent dropout probability were the best set of hyperparameters for this network. Because there was no more hyperparameters that could be further optimized it was tried to increase the size of input images.

6.6.4 CNN on larger images

It was expected that larger image size will lead to a better accuracy score when performing image classification, because larger image sizes allow more information to be preserved in images. The network was redesigned to take 56x56x3 pictures as an input. One more convolutional and a max-pooling layer were also introduced

to the network. It was tried to find the optimal learning rate. The network was trained with 0.003 learning rate, and a 20 percent drop probability for 30 epochs. It was noticed that learning rate, which worked very well for the previous neural network, was too high for this network. Because of that, the gradient optimization algorithm kept overshooting the minimum an training accuracy and loss stayed the same during the training of the model. To find the optimal learning rate it was gradually lowered until it reached 0.0015. With this learning rate the algorithm was able to learn the optimal weights. When tested it achieved 60.72 test accuracy. It was tried to increase the number of epochs to 100, because more epochs meant that the weight could be optimized more correctly. After testing the model on the test dataset it was observed that the accuracy increased to 62.26 percent.

Finally, it was tested weather modifying the architecture of dropout layers could improve the performance of the classifier. In previous model, dropout was used in every layer except the fully connected one. The dropout was added to this layer too. However, after adding the dropout, the learning rate value of 0.0015 was too large. After decreasing the value of the learning rate to 0.001 it became possible to train the network. A trained network with 20 percent drop probability reached 69.34 percent accuracy and a trained network with 30 percent drop probability reached 63.98. Therefore, 20 percent was a better drop rate in both approaches. It was also tried to modify the network to use dropout only in a fully connected layer. Because only one dropout layer was used the dropout probability needed to be increased to 50 percent. The network was trained with a 0.001 learning rate. However, training accuracy of the model improved very slowly. Therefore, learning rate was increased and the optimal its optimal value of 0.002 was found. The trained model was tested on the test data set. The accuracy on the test was 51.54 percent. A table of CNN networks that were constructed for 56x56x3 images can be seen in Table 6.5.

Learning Rate	Drop probability	Network layers with drop	Training Epochs	Accuracy
0.001	20%	Every except fully connected	30	57.08%
0.0015	20%	Every except fully connected	30	60.72%
0.0015	20%	Every except fully connected	100	62.26%
0.001	30%	Every	100	63.98%
0.001	20%	Every	100	69.34%
0.002	50%	Fully connected	100	51.54%

TABLE 6.5: Different configurations for convolutional neural networks tried with 56x56x3 pictures

6.7 Transfer learning techniques for food Classification

6.7.1 What is transfer learning

Currently, all state of the art image classifiers are large convolutional neural networks. For training these networks large GPU clusters were used and it usually took multiple weeks of computing to train these models. For example, the Google's Inception v3 network was trained on a cluster of 50 computers with NVidia Kepler GPUs Szegedy et al. (2015). It is impossible to train this network using only one computer because the network has millions of parameters that require a massive amount of memory and computation power. The only way how this network could be retrained for a different dataset on a regular computer is by using a transfer learning approach.

Transfer learning is a technique that shortcuts a lot of this work by taking a fully-trained model for a set of categories like ImageNet and retrains it from the existing weights for new classes *How to Retrain Inception's Final Layer for New Categories — TensorFlow* (n.d.). Usually, only the two final layers of a neural network are retrained to save time. The reason why retraining only the final two layers is effective it that convolutional layers generally tend to activate to patterns of different shapes, colours and edges. Such patterns appear not to be specific to a particular dataset or task, but can be applicable to many datasets and tasks Yosinski et al. (2014).

It was chosen to retrain the Google's Inception v3 model. A python script for loading and retraining this model is publicly available in the Tensorflow's Github repository *Github Tensorflow retrain.py* (n.d.). The script provides the ability to tweak hyperparameters such as a learning rate, number of training epochs and a batch size.

```
python /tensorflow/tensorflow/examples/image_retraining/retrain.py \
--bottleneck_dir=/tf_files/bottlenecks \
--how_many_training_steps 500 \
--model_dir=/tf_files/inception \
--output_graph=/tf_files/retrained_graph.pb \
--output_labels=/tf_files/retrained_labels.txt \
--image_dir /tf_files/images
```

FIGURE 6.5: Terminal command used to launch python script that retrains the Inception v3 Model

6.7.2 Retraining of the Inception v3 model

The classifier was retrained with a learning rate = 0.01, a number of epochs = 500 , and a batch size = 100. The script split the dataset to use 80 percent of it as a training set, 10 percent of it as a testing set used to test the classifier after each epoch and the last 10 percent as a validation dataset used to test the trained model. The terminal command that was used to run the retrain.py can be seen in Figure 6.5.

When retrain.py was runt it firstly calculated bottlenecks for all the images in the dataset. A bottleneck is a term used for naming the layer just before the final output layer that actually does the classification. Calculation of bottlenecks took a significant amount of time. The script saved the calculated bottlenecks to the disk, so this task was only needed to run once.

After the calculation of the bottlenecks. The weights of the penultimate and final layers were set and optimisation of weights in these layers began. The training progress of the classifier was shown in the terminal (Figure 6.5). It was observed that just after 10 epochs the classifier reached 96 percent accuracy. After running the weight optimisation algorithm for 500 iterations, the final trained model achieved 98.2 percent accuracy when tested on 507 validation samples. That meant, that the classifier was able to classify 98 from a 100 pictures of unseen food correctly. This accuracy is much higher than accuracies achieved by every other classifier.

To get a better understanding how can this classifier perform on the real-world data 5 example images for each class were downloaded.These images were a had picked examples of each class. The python script that loads the learnt model from disk, and feeds classifier with images from the given folder was written. After running the classification on 20 accumulated images, one classification error was detected. The classifier classified a picture of deep pan pizza Figure 6.7 as fish and chips. It was expected that this example might get misclassified, because there were no deep pan pizzas in a training dataset. The smooth structure of a crust of the pizza can also resemble a battered fish.

```
2017-03-06 22:13:46.987121: Step 0: Train accuracy = 45.0%
2017-03-06 22:13:46.994374: Step 0: Cross entropy = 1.251378
2017-03-06 22:13:47.344756: Step 0: Validation accuracy = 28.0% (N=100)
2017-03-06 22:13:51.416093: Step 10: Train accuracy = 94.0%
2017-03-06 22:13:51.416289: Step 10: Cross entropy = 0.746439
2017-03-06 22:13:51.727425: Step 10: Validation accuracy = 96.0% (N=100)
```

FIGURE 6.6: Retraining process of the classifier, 96 percent accuracy was reached in just 10 epochs

FIGURE 6.7: A picture of a deep pan pizza that was missclassified as a fish

6.7.3 Conclusions

To conclude, transfer learning works very well for fine-tuning a previously trained neural network for a new dataset.It is the only solution for training a complex deep learning classifier without having huge resources of computing power. It performed the best of all classification methods that were tried and achieved 98.2 percent accuracy.

A disadvantage of this approach is that the originally trained model is needed in order to fine tune it. It can be hard to find a publicly available network model. In addition, the data that is used to retrain the model should not be very different from the data used to train the original model, otherwise it will not be able to optimize the weights.

6.8 Conclusions of Deep Learning Methods

To conclude, it can be clearly seen that deep learning models on average performed much better than a standard machine learning models. The accuracy achieved by a transfer learning model was very high. The model could used in a practice, because the error rate is very low.

However, finding the optimal parameters for the deep neural network models are much harder task. Creating architectures of neural networks is a hard task. It can not be know beforehand what structure of the network will work on specific task. Therefore, many architectures have to be tried to find a suitable architecture. Even then it is unknown if the chosen architecture is really the best, because the configuration space is very large.

Chapter 7

Results

Results of the best classification model of each method tried are shown in Table 7.1. From this table it can be seen that transfer learning is the best classification method with 98.20 percent accuracy. Convlolutional neural network is at the second place with 69.34 percent accuracy. It is important to state that it was actually possible to improve the convolutional neural network further by adding more layers and modifying the structure. However, it was not accomplished because it was unkown what structure was actually needed.

That is the main limitation of neural networks that was observed in this project. A person building deep or convolutional neural network has to make a lot of decisions that directly impact the performance of the network. Choosing the number of layers and the number of neurons in each layer is a very challenging task. Furthermore, activation function and weight optimization function needed to be selected too.

Method	Classification Accuracy
KNN	56.32%
SVM with PCA	60.73%
Linear	51.53%
3 Layer	54.02%
5 Layer	58.81%
CNN	69.34%
Transfer	98.20%

Table 7.1: Best classifiers of each method

Training a multi-layer neural network is a lot more challenging task than first thought. This task can be very challenging. h

Chapter 8

Conclusion

Objectives of the project were to practically understanding of machine learning algorithms that could be use for classifying images,find a suitable food image dataset, preprocess the images, build image classifiers using various machine learning algorithms, train these classifiers with the accumulated dataset and evaluate the performance.

Appendix A

A Computer System used for the project

Performance of Machine Learning algorithms can be directly mapped to a computing power of a computer system used. With more computing power it is possible to train algorithms using more training data or with using a bigger set of features. Because of that it is important to show the parameters and limitations of the computer system used.

CPU: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz

Maximum speed: 2.40 GHz Cores: 2 Logical processors: 4 L1 cache: 128 KB L2 cache: 512 KB L3 cache: 3.0 MB

Memory: 8.0 GB DDR3, frequency: 1600 MHz

Disk: Samsung SSD 850 EVO 500GB

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M. et al. (2016), 'Tensorflow: Large-scale machine learning on heterogeneous distributed systems', *arXiv preprint arXiv:1603.04467* .

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of the fifth annual workshop on Computational learning theory', ACM, pp. 144–152.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J. et al. (2013), 'Api design for machine learning software: experiences from the scikit-learn project', *arXiv preprint arXiv:1309.0238* .

Chen, H.-C., Jia, W., Sun, X., Li, Z., Li, Y., Fernstrom, J. D., Burke, L. E., Baranowski, T. & Sun, M. (2015), 'Saliency-aware food image segmentation for personal dietary assessment using a wearable computer', *Measurement Science and Technology* **26**(2), 025702.

Forsyth, J. P. D. A. (2011), *Computer Vision: A Modern Approach, 2nd Edition*, Prentice Hall.

Github Tensorflow retrain.py (n.d.). [Online; accessed 2017-03-03].
URL: *https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py*

Goodfellow, I., Bengio, Y. & Courville, A. (2017), *Deep Learning*, MIT Press.

Halevy, A., Norvig, P. & Pereira, F. (2009), 'The unreasonable effectiveness of data', *IEEE Intelligent Systems* **24**(2), 8–12.

How to Retrain Inception's Final Layer for New Categories — TensorFlow (n.d.). [Online; accessed 2016-11-29].
URL: *https://www.tensorflow.org/versions/master/how_tos/image_retraining/*

Jones, E., Oliphant, T., Peterson, P. et al. (2001–), 'SciPy: Open source scientific tools for Python'. [Online; accessed 2017-02-28].
URL: *http://www.scipy.org/*

Karpathy, A. (2016), 'Cs231n convolutional neural networks for visual recognition'. [Online; accessed 2016-12-28].
URL: *http://cs231n.github.io/classification/*

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *CoRR* **abs/1412.6980**.
URL: *http://arxiv.org/abs/1412.6980*

k-nearest neighbors algorithm (2017). [Online; accessed 2017-02-28].
URL: *https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg*

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* F. Pereira, C. J. C. Burges, L. Bottou & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 25', Curran Associates, Inc., pp. 1097–1105.
URL: *http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf*

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation applied to handwritten zip code recognition', *Neural computation* **1**(4), 541–551.

Lowe, D. G. (1999), 'Object recognition from local scale-invariant features', **2**, 1150–1157.

Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2012), *Foundations of Machine Learning*, The MIT Press.

Nair, V. & Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Nakano, R., Hotta, K. & Takahashi, H. (2006), 'An object detection method based on independent local features', *JOURNAL OF ROBOTICS AND MECHATRONICS* **18**(6), 744.

Papert, S. A. (1966), 'The summer vision project'.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research* **12**(Oct), 2825–2830.

Pérez, F. & Granger, B. E. (2007), 'IPython: a system for interactive scientific computing', *Computing in Science and Engineering* **9**(3), 21–29.
URL: *http://ipython.org*

RBF SVM parameters (2017). [Online; accessed 2017-02-28].
URL: *http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html*

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015), 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)* **115**(3), 211–252.

Samuel, A. L. (1959), 'Some studies in machine learning using the game of checkers', *IBM J. Res. Dev.* **3**(3), 210–229.
URL: *http://dx.doi.org/10.1147/rd.33.0210*

Shlens, J. (2014), 'A tutorial on principal component analysis', *arXiv preprint arXiv:1404.1100* .

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting.', *Journal of Machine Learning Research* **15**(1), 1929–1958.

SVM (2017). [Online; accessed 2017-02-28].
URL: *https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_(SVG).svg*

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015), 'Rethinking the inception architecture for computer vision', *CoRR* **abs/1512.00567**.
URL: *http://arxiv.org/abs/1512.00567*

Ting, K. M. (2011), *Encyclopedia of machine learning*, Springer.

Wikipedia (2017), 'Max Pooling — Wikipedia, the free encyclopedia', https://en.wikipedia.org/w/index.php?title=Convolutional%20neural%20network&oldid=768867528#/media/File:Max_pooling.png. [Online; accessed 16-March-2017].

Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. (2014), 'How transferable are features in deep neural networks?', *CoRR* **abs/1411.1792**.
URL: *http://arxiv.org/abs/1411.1792*

Zhou, Y.-T., Chellappa, R., Vaid, A. & Jenkins, B. K. (1988), 'Image restoration using a neural network', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **36**(7), 1141–1151.