

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:

Working Documents

The working documents of this project are available at <http://www.lancaster.ac.uk/ug/tumosa/> and <https://lancaster.app.box.com/v/FoodData>.

The Git repository of the project can be found at <https://github.com/laur1s/FoodClassifier>. The L^AT_EX source code of this report is available at <https://github.com/laur1s/Dissertation>.

Abstract

The dissertation focuses on computer vision techniques for classifying food objects from images. Food classification is an important task, that could be use for automatic dietary assessment. To accomplish this task, the dataset of food images was firstly obtained. Then, different machine learning algorithms such as the k-nearest neighbors classifier, support vector machine, linear classifier, multilayer neural network, convolutional neural network, and transfer learning were explored. The classifiers that use these algorithms were developed. To evaluate the classification performance, accuracy score of each classifier was calculated. The method that worked best, transfer learning, achieved 98.20 percent accuracy on the test dataset. It was concluded that food classification from images with reasonable accuracy was possible and deep learning models work much better for this task, compared to the other machine learning techniques.

Acknowledgements

I would like to thank to my project supervisor, Adrian Friday for guidance and valuable feedback provided for implementing the project and writing the report.

I would like to thank Christian Bizer who taught a data mining course at the University of Mannheim when I studied there. This course introduced me to data mining and machine learning and aroused my interest in this field of computer science.

I would like to thank Andrej Karpathy who was a lecturer at Stanford's CS231n: Convolutional Neural Networks for Visual Recognition course. He made all lecture recordings and the course material publicly available. These lectures were a great learning material and contributed greatly to the success of the project.

I would like to thank my family for believing that I could study computer science at university and for supporting and encouraging me.

Finally, I would like to dedicate this dissertation to my sister, Aiste.

Contents

Declaration of Authorship	i
Working Documents	i
Abstract	ii
Acknowledgements	iii
List of Figures	vii
List of Tables	ix
Glossary	x
1 Introduction	1
1.1 The Structure of the Report	2
2 Background	3
2.1 Introduction to the History of Computer Vision	3
2.2 Why is Computer Vision a Hard Problem?	4
2.3 Review of the Study of an Automatic Food Image Segmentation	6
2.4 Survey of Applications for Counting Calories	8
2.5 Summary	11
3 Research Methods	12
3.1 Choice of a Programming Language	12
3.2 Choice of Machine learning libraries	14
3.2.1 Why was a Machine Learning Library Needed?	14
3.2.2 Library Used for Machine Learning	14
3.3 The Evaluation Method for the Classification Algorithms	14
3.4 Summary	15
4 Accumulation of a Food Image Dataset	16
4.1 The Requirements for the Food Image Dataset	16

4.2	Search for the Food Image Dataset	17
4.3	Downloading the Dataset	18
4.4	Creating the Dataset	19
4.5	Summary	19
5	Machine Learning Algorithms for Food Image Classification	21
5.1	K-Nearest Neighbours Classifier	21
5.1.1	Introduction to K-Nearest Neighbours Algorithm	21
5.1.2	Implementation of a K-Nearest Neighbours Classifier	22
5.1.3	Tuning of Hyperparameters	23
5.1.4	Results	25
5.2	Support Vector Machine	26
5.2.1	Introduction to Support Vector Machine	26
5.2.2	Implementation of a SVM Classifier	26
5.2.3	Tuning of SVM Hyperparameters	27
5.2.4	Reduction of Dimentionality with Principal Component Analysis	28
5.3	Summary	30
6	Deep Learning approach for the Food Image Classification	31
6.1	Short Introduction to Deep Learning	31
6.2	Introduction to TensorFlow	32
6.3	Transformation of the Dataset For Using It with TensorFlow	33
6.4	Linear Classifier in TensorFlow	34
6.4.1	Introduction to a Linear Classifier	34
6.4.2	Training of the Linear Classifier	34
6.4.3	Results	35
6.5	Multi-layer Neural Network in TensorFlow	36
6.5.1	Results	37
6.6	Convolutional Neural Networks in TensorFlow	38
6.6.1	Introduction to Convolutional Neural Networks	38
6.6.2	Implementation of a Convolutional Neural Network	39
6.6.3	Improving the Convolutional Neural Network	40
6.6.4	Designing Convolutional Neural Network for Larger Images	41
6.7	Transfer Learning Technique	42
6.7.1	What is Transfer Learning	42
6.7.2	Retraining the Inception v3 Model	44
6.8	Summary	45
7	Results	46
7.1	Comparison of Accuracy Scores	46
7.2	Comparison of Time Complexity of the Classifiers	46
7.3	Possible Improvements and Limitations of Classification Models	48
7.4	Summary	49

8	Conclusions	50
8.1	Lessons Learnt	50
8.2	Limitations	51
8.3	Future Work	51
A	A Computer System used for the project	52
	References	53

List of Figures

2.1	Example ImageNet Images of a Pizza Class	5
2.2	Example of Various Variations in Images (Karpathy 2016a)	6
2.3	Personal Dietary Assessment with eButton	7
2.4	Major Difficulties in Automatic Food Segmentation	7
2.5	Container Detection; from (Chen et al. 2015)	8
2.6	Google Play Store page of MyFitnessPal App	9
2.7	Procedure to Create a MyFitnessPal Account	9
2.8	Barcode Scanning with MyFitnessPal	10
2.9	Food Search in Nutracheck	11
2.10	Food Search in LifeSum	11
3.1	Example of a Jupyter Notebook	13
3.2	Precision and Recall (Wikipedia 2017c)	15
4.1	Example of an Flickr Image with a Fish and Chips Tag	18
5.1	Example of k-NN classification. (Wikipedia 2017a)	22
5.2	Confusion Matrix of KNN Classifier with $K = 18$	23
5.3	Confusion Matrix of KNN classifier with $K= 8$, Euclidean Distance and Uniform Weights	25
5.4	Three hyperplanes that could be used to split the data	26
5.5	Confusion Matrix of Support Vector Machine Classifier Trained with Default Parameters	27
5.6	Confusion Matrix of SVM Trained with $C= 10$	28
5.7	Confusion Matrix of SVM with $C = 10$ and $\text{Gamma} = 0.0001$	29
5.8	Confusion Matrix of SVM Classifier on the Dataset Reduced with PCA	30
6.1	The ReLU Activation Function (Karpathy 2016b)	32
6.2	TensorFlow Program that Adds two Scalars	33
6.3	Snippet of the 3 Layer Network Model	36
6.4	Illustration of the Max Pooling	39
6.5	The Convolutional Neural Network Model in TensorFlow	39
6.6	Layer Diagram of the Convolutional Neural Network	40
6.7	Command Used to Retrain the Inception v3 Model	43
6.8	Retraining of the Classifier, 96 Percent Accuracy was Reached in 10 Epochs	45

6.9 Deep Pan Pizza that was Missclassified as a Fish	45
--	----

List of Tables

5.1	Precision, Recall and F Scores of the best K-NN Classifier	25
5.2	Precision, Recall and F Scores of SVM Model on the Dataset Reduced with PCA	29
6.1	One-Hot Encoding for the Dataset	34
6.2	The Influence of the Hyperparameters to the Accuracy of the Linear Classifier	36
6.3	The Influence of the Hyperparameters to the Accuracy of the Multi-layer Neural Network	37
6.4	The First two Convolutional Models Tried	40
6.5	Configurations for Convolutional Neural Networks tried with 56x56x3 Pictures	43
7.1	Best Classifiers of Each Method	47

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 not good for one’s 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, an approach that today’s 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 the keyboards of their smartphones.

The goal of this project is to explore techniques for an automatic food classification from images. This could enable users to capture food that they eat in an unobtrusive and user-friendly way. A created machine learning model could use a picture of food taken by a smartphone’s camera as an input method and classify it. In most of the smartphones, a camera can be accessed in a few clicks. People are used to taking pictures on their smartphones. Therefore, an automatic food classification using a phone camera would be a perfect solution to the food tracking problem.

However, automatic food classification is not an easy task. There is no direct way to detect what kind of food product is in a picture. A digital image is just a matrix of numbers representing an intensity of three different colour components:

red, green and blue (RGB). It is impossible to create an algorithm that could directly map an image to the label of a food item. Therefore, a machine learning is needed to learn a model that can recognise a particular 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 labelled examples as training data and making predictions of labels for all unseen points ([Mohri et al. 2012](#)). This project explores various supervised learning algorithms that could be used for classifying the images.

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 the generation of a food image dataset, preprocessing of images, implementing machine learning algorithms, and training classification models.

1.1 The Structure of the Report

In the next chapter, a background theory and research in the area of computer vision are going to be introduced. Chapter 3 presents a programming language and a machine learning library chosen for this project and explains the reasoning behind these choices. The evaluation method used to observe the performance of machine learning algorithms is also presented in this chapter. Chapter 4 discusses why a dataset of food images was needed, what kind of requirements were set for it and how was the dataset accumulated. 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 6 a particular type of machine learning- deep learning is explored. Chapter 7 compares the best results of all classifiers that were built. Lastly, in Chapter 8 the conclusions of the project 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 the MIT AI lab. 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 ([Papert 1966](#)). It was thought that vision was a relatively easy field of AI and that a landmark in the development of pattern recognition could be achieved 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. Pixel variation in images that are produced under different conditions is high. Therefore, important features have to be selected to make a picture more resistible to various variations. Later these selected features are used for detecting a particular 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 proposed by [Nakano et al. \(2006\)](#). The authors of this paper state 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 were derived automatically from the training samples. To do this, researchers used independent component analysis (ICA) which decomposed a signal into separate 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 the ImageNet dataset and the ImageNet Large Scale Visual Recognition Challenge in 2010. The challenge is a competition where research teams submit image classification models that classify and detect objects. The challenge is running annually since 2010. 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 ([Russakovsky et al. 2015](#)). Example images from the ImageNet which represent a pizza class can be seen in [Figure 2.2](#).

Because of many images and categories, ImageNet is still a huge challenge for researchers who try to build classification models for the challenge. ImageNet encourages 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 ImageNet Images of a Pizza Class

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

A human can identify objects in images under all kinds of changes in illumination, viewpoint, scale, etc. In comparison, computer algorithms are very susceptible to all sorts of variations. Major variations in pictures were defined by [Karpathy \(2016a\)](#) 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 a chair. There are many different types of these objects, each with their own appearance.

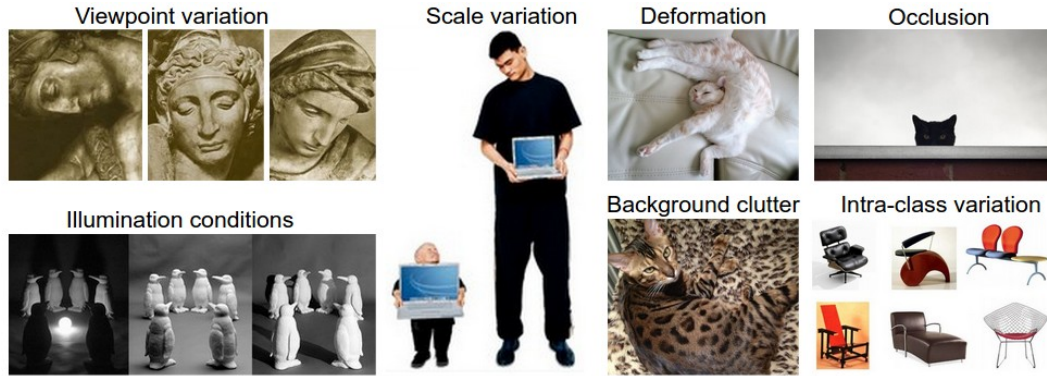


FIGURE 2.2: Example of Various Variations in Images ([Karpathy 2016a](#))

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

2.3 Review of the Study of an Automatic Food Image Segmentation

A computer system that can be utilized for an image-based dietary assessment was described by [Chen et al. \(2015\)](#). A wearable computer called eButton was used to capture the eating events. The eButton is a small chest camera that can be pinned on clothes. The camera can take pictures automatically at the rate of one image per second. Then, segmentation techniques are applied to segment food from a container. Finally, a person manually enters the food label to the application. The operational diagram of the eButton ([Figure 2.3](#)) states that food volume measurement, nutrient database lookup, and calculation of calories are also performed by the device. However, these operations were not described in the paper.

The main focus of the paper was a food segmentation from its container. Authors stated that major difficulties in automatic food segmentation are multiple food components in a container, coloured decorative patterns on plates and occlusion

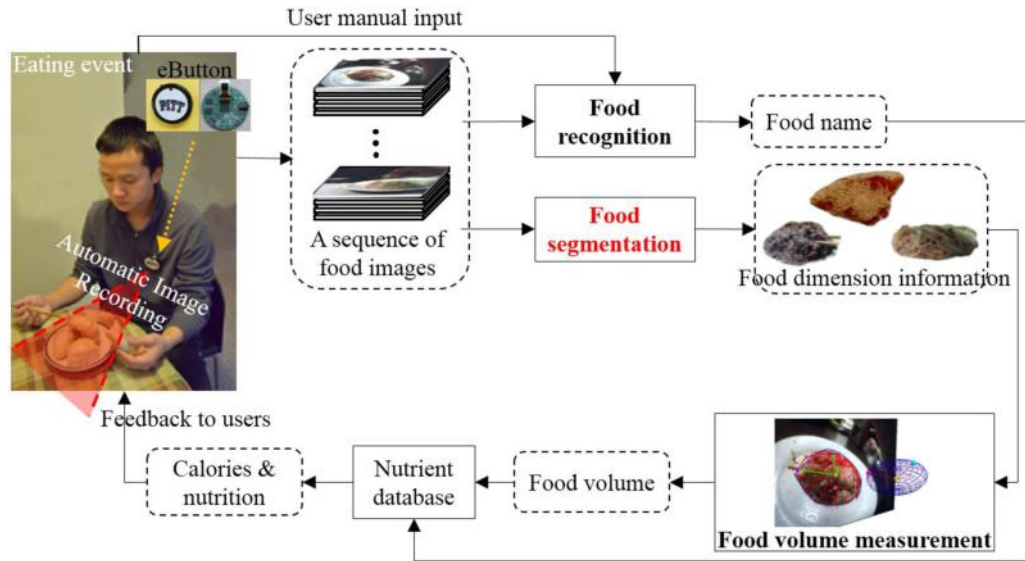


FIGURE 2.3: Personal Dietary Assessment with eButton from (Chen et al. 2015)

by other objects. The example images of these three difficulties are shown in Figure 2.4.

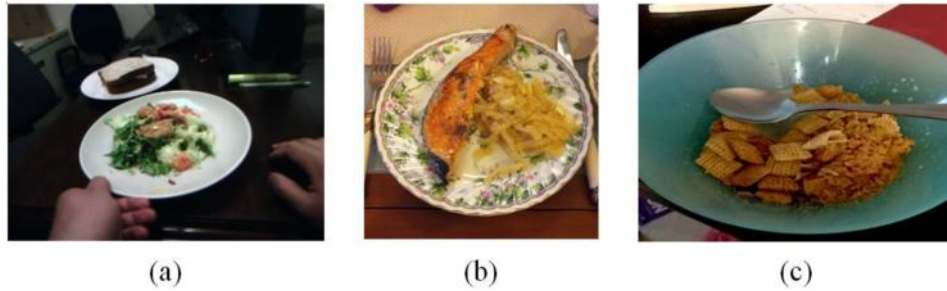


FIGURE 2.4: Major Difficulties in Automatic Food Segmentation from (Chen et al. 2015)

Researchers approached an image segmentation in the two stages. Firstly, a food container was detected in an image by its shape convexity. Canny edge detector was used to obtain the edge information from a given image. Then, some squares at edge pixels were randomly selected Figure 2.5. Then, squares not belonging to the container edge were discarded, and the area of the container was detected.

For segmenting food inside a plate, researchers used the color contrast between objects and surroundings, colour abundance and spatial arrangement. That allowed to recognise the areas in the image with one highly dominant colour. These three characteristics were then combined to segment food from a container.

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 pictures segmented by two research participants. The average error rate of the machine was $8.443 \pm 5.203mm$.

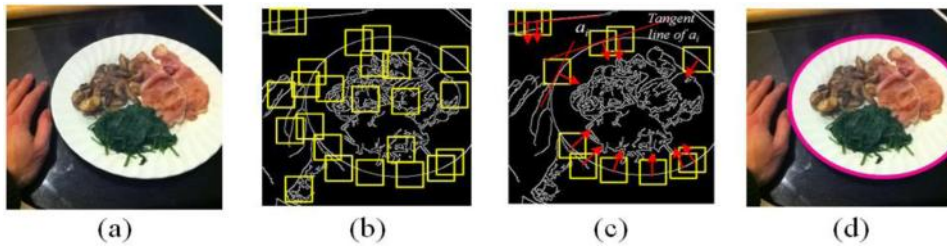


FIGURE 2.5: Container Detection; from (Chen et al. 2015)

To conclude, this study proved that food segmentation from images was possible. However, the dataset used to evaluate the segmentation approach was too small to make a general conclusion whether this approach is suitable for various kinds of food images. The main limitation of the image segmentation by shape convexity is that in some cases food is served not in a round shaped container, for example take away food box or a lunch box. Some of the foods can also be eaten without using any container at all, for instance fruits and vegetables.

2.4 Survey of Applications for Counting Calories

It was decided to try currently available applications for calorie counting, to get a better understanding what are the input methods offered by them. To get applications keyword “calorie counting” was entered into the search bar of the 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 calorie-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 broad and includes products from many different restaurants. However, adding a meal that was prepared by a user himself is complicated. Every ingredient of the dish must be searched and added separately. The portion size has to be entered using weight measurements, for example, grams. It is a significant disadvantage because it is often hard to estimate the weight of the meal.

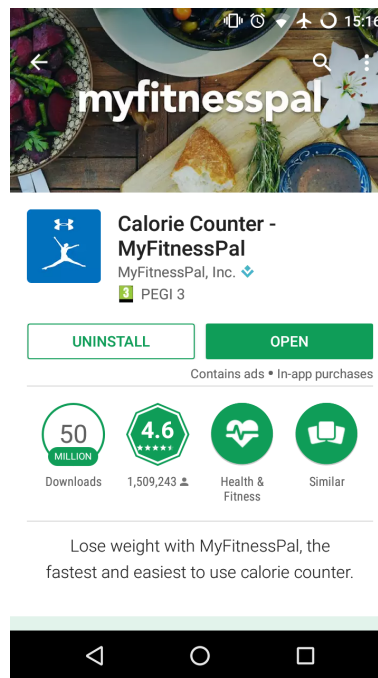


FIGURE 2.6: Google Play Store page of MyFitnessPal App

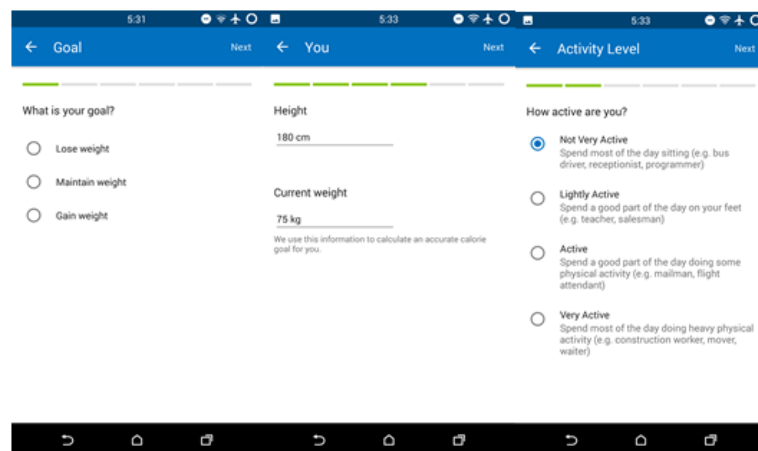


FIGURE 2.7: Procedure to Create a MyFitnessPal Account

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.

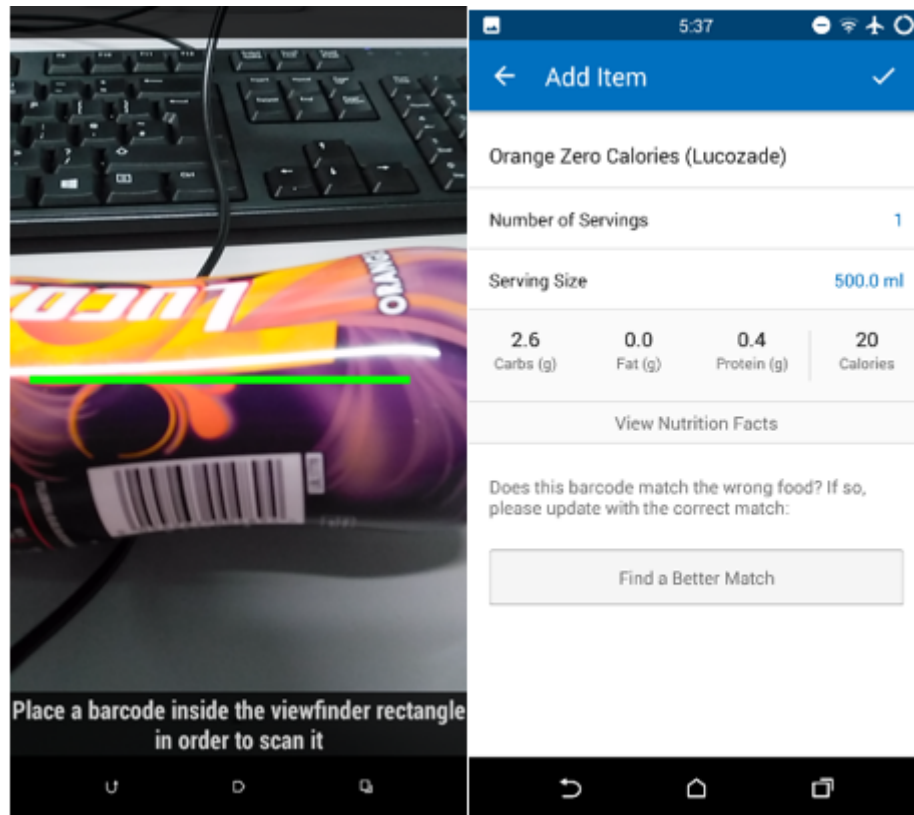


FIGURE 2.8: Barcode Scanning with MyFitnessPal

For entering meals, the app also has a barcode scanner and a search function. One advantage of Nutracheck 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 than a barcode scanner in MyFitnessPal when it was tried.

Final app that was tested was Lifesum. It also requires completing a survey, to start using it, and uses a barcode scanner and a search function for adding food. An interesting feature that this app has is a food quality rating. It rates a quality of food according to its 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. A text-based search and barcode scanners are used as data entry points in all applications. For tracking calorie intake, a user has to enter what he ate manually. The Internet connection is also required to use these applications. Ability to use phone's camera as an input method would significantly improve the usability of the food tracking applications.

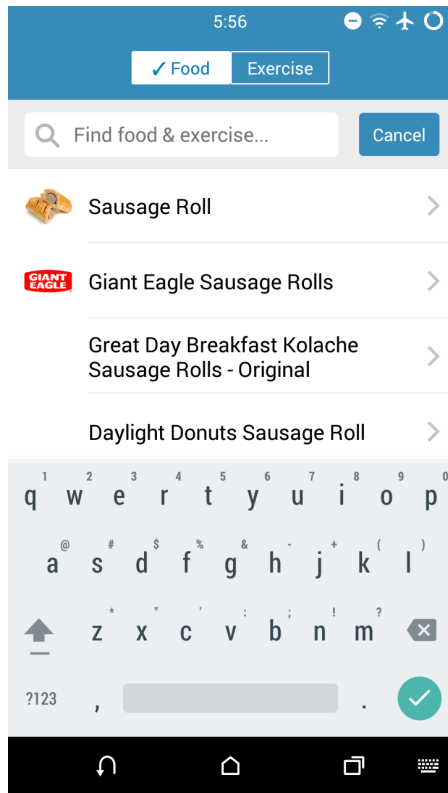


FIGURE 2.9: Food Search in Nutracek

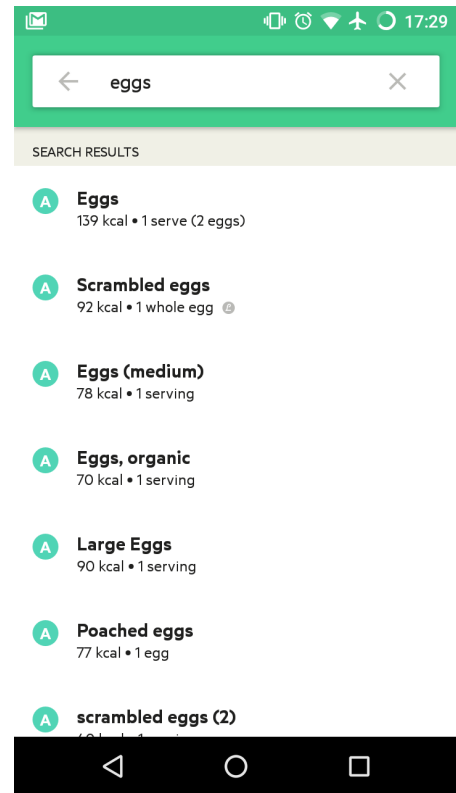


FIGURE 2.10: Food Search in LifeSum

2.5 Summary

In this chapter history of computer vision was reviewed. Then, it was explained why computer vision is such a hard problem. A paper that focused on automatic food image segmentation was summarised and reviewed. Finally, currently available applications for dietary assessment were reviewed. It was concluded, that an automatic food image classifier could extend the capabilities of personal dietary assessment. In the next chapter research methods of the project are discussed.

Chapter 3

Research Methods

This chapter provides information about the research methods used in this project. Before conducting a series of experiments, to explore and evaluate machine learning methods, it was needed to decide on how are these algorithms going to be implemented and evaluated. Choices of a programming language, a machine learning library, and an evaluation method are explained in this chapter.

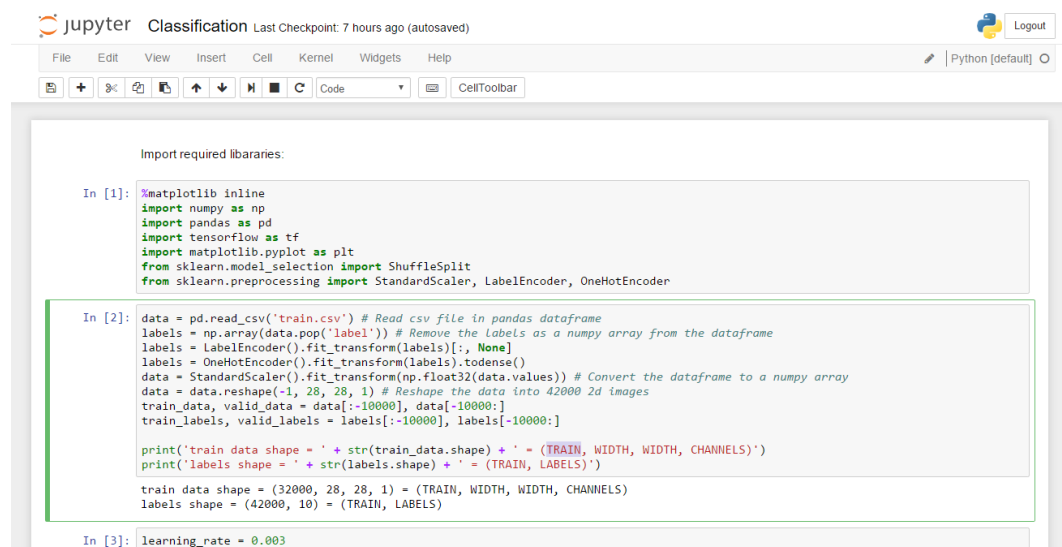
3.1 Choice of a Programming Language

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 source. Since Python is an interpreted language, code written in Python can be executed on multiple different platforms without having a need to modify the code. Because Python language has a large community of developers, there are many publicly available open source packages with efficient implementation of machine learning algorithms for this language.

Availability of scientific computing tools for Python was also one of the main reasons why this language was chosen. Scientific Python libraries that were used in this project were Matplotlib, NumPy, IPython and Jupyter Notebook ([Jones et al. 2001](#)). Matplotlib provided an ability to plot graphs, images, and tables. NumPy was used for efficient arithmetical operations with matrices. IPython allowed executing only a part of a program without a need to run a full code. It

is a system for an interactive scientific computing (Pérez & Granger 2007). Some of the operations, for example, loading the dataset or resizing the images can take a long time to complete. In IPython these operations are only required to be completed once. Then, results of these operations are saved in memory where they remain and can be reused. Jupyter Notebook allowed to execute Python code in a web browser and save the results as HTML pages (Figure 3.1). Jupyter Notebook is an IPython wrapper that launches an HTTP server. Scientific Python libraries allowed to interactively explore the data with an ability to plot graphs, tables and visualise 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 an image processing and machine learning toolboxes. Machine learning and image processing functions are easier to use in Matlab than in Python because this language was developed for a particular purpose. However, Matlab is a closed source language. It also only supports some platforms, where Python code can be executed in almost any computing environment. Therefore, Python programming language was a better option.



The screenshot shows a Jupyter Notebook titled "Classification" with a "Last Checkpoint: 7 hours ago (autosaved)" status. The interface includes a top bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help" menus. Below the menus is a toolbar with icons for saving, undo, redo, and other actions. The main area contains three code cells. The first cell imports required libraries: matplotlib, numpy, pandas, tensorflow, sklearn, and sklearn.preprocessing. The second cell reads a CSV file, removes the 'label' column, and reshapes the data into a 42000x28x28x1 array. The third cell sets the learning rate to 0.003.

```

Jupyter Classification Last Checkpoint: 7 hours ago (autosaved)
Python [default]

File Edit View Insert Cell Kernel Widgets Help
Python [default]

In [1]: Import required libraries:
import matplotlib inline
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import ShuffleSplit
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

In [2]: data = pd.read_csv('train.csv') # Read csv file in pandas dataframe
labels = np.array(data.pop('label')) # Remove the Labels as a numpy array from the dataframe
labels = LabelEncoder().fit_transform(labels).tolist()
labels = OneHotEncoder().fit_transform(labels).todense()
data = StandardScaler().fit_transform(np.float32(data.values)) # Convert the dataframe to a numpy array
data = data.reshape(-1, 28, 28, 1) # Reshape the data into 42000 2d images
train_data, valid_data = data[:-10000], data[-10000:]
train_labels, valid_labels = labels[:-10000], labels[-10000:]

print('train data shape = ' + str(train_data.shape) + ' = (TRAIN, WIDTH, WIDTH, CHANNELS)')
print('labels shape = ' + str(labels.shape) + ' = (TRAIN, LABELS)')

train_data shape = (32000, 28, 28, 1) = (TRAIN, WIDTH, WIDTH, CHANNELS)
labels shape = (42000, 10) = (TRAIN, LABELS)

In [3]: learning_rate = 0.003

```

FIGURE 3.1: Example of a Jupyter Notebook

3.2 Choice of Machine learning libraries

3.2.1 Why was a Machine Learning Library Needed?

It is possible to implement machine learning algorithms without using any additional library. However, there are many benefits provided by machine learning libraries. Algorithms available in machine learning libraries are often implemented more efficiently. Adapting a provided algorithm for a specific task also takes a less time than writing a particular algorithm for it. Therefore, it was decided to analyse what machine learning libraries are available for Python.

3.2.2 Library Used for Machine Learning

Scikit-learn python package was chosen as a library for working with machine learning models. This package provides state-of-the-art implementations of many well-known machine learning algorithms and 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 C and C++ are much faster than Python, it makes algorithms run faster compared to a standard Python implementation. Scikit-learn machine learning algorithms are high-level API's ([Buitinck et al. 2013](#)). Algorithms provided in this package do not require a knowledge of their implementation details. Nevertheless, Scikit-learn interface allows parameters of these algorithms to be configured.

In the case of food image classification, the main advantages that Scikit-learn provided were the fact that it uses the same data input format for every machine learning algorithm and ability to easily configure parameters, to make classifiers work on image classification.

3.3 The Evaluation Method for the Classification Algorithms

The key 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 is the total number of elements classified correctly divided by a total number of items that were classified as that class. A recall is the total number of objects classified correctly divided by a total number of items of that type in the database. A graphical illustration of precision and recall measures is shown in Figure 3.2. F-measure is a measure combined from precision and recall according to a formula $F = 2 \frac{Recall \times Precision}{Recall + Precision}$ (Ting 2011).

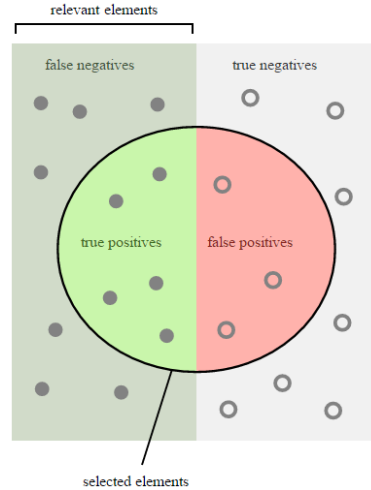


FIGURE 3.2: Precision and Recall (Wikipedia 2017c)

$$P = \frac{TP}{TP+FP}, R = \frac{TP}{TP+FN}$$

3.4 Summary

In this chapter reasoning behind a chosen programming language and machine learning library used was provided. Methods for evaluating the performance of classification algorithms were discussed. Accuracy score was picked as the primary method for performance evaluation. In the following chapter, accumulation of a food image dataset is discussed.

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. In this chapter, reasons, why the dataset is required, are explained, and a process of accumulating the dataset is described.

4.1 The Requirements for the Food Image Dataset

The dataset was needed for machine learning algorithms to learn a classification model from it. The most important requirements to the dataset were:

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

Food images taken in the lab conditions could not be used for training classification models. The intention of the project was to create the models that could classify pictures of food taken by various users. People often use different angles

and distance while taking pictures. The light conditions, too, can often vary considerably. It was not possible to represent these variations in a lab environment. Therefore, it was crucial to use a food image dataset with a high variance.

The dataset also needed to have many images that belong to each class. Having a lot of training samples is the best way to make sure that the trained model is generalised and, thus, is able to function 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 to have around 1,000 training images for each class in the dataset.

Finally, an intra-class variance of pictures in the dataset should also be high. The problem of datasets with little variation can be illustrated with a dataset of tree images containing only pictures of trees with green leaves. The model learned by a classifier from this dataset will always classify a tree with orange leaves or a tree without any leaves as a not tree. Therefore, a food dataset should also include a different kind of images for the same class, for example pizza with cheese, pepperoni pizza, and a slice of pizza.

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

4.2 Search for the Food Image Dataset

It was observed that there are a lot of food images hosted on a photo sharing service flickr.com. Flickr has a Python API, that could be used to search for images according to their tags or titles, and downloading them. A Python script using this API was written. However, after downloading some images with a tag “Fish and Chips”, it was realised that a lot of downloaded images contained this tag but were not actual food pictures e.g. [Figure 4.1](#). Because of the time it would take to review every picture, it was decided to try to find a publicly available food image dataset.



FIGURE 4.1: Example of an Flickr Image with a Fish and Chips Tag

It was found that ImageNet Classification challenge, that was briefly described in [section 2.1](#) includes many categories of food pictures. The only problem was that the number of images in different classes varies widely. Some classes, for example, pizza has around 1300 pictures while other classes like salmon loaf only has 360 pictures. This problem was only discovered later when it was tried to train a classifier on this class. The classifier never picked a salmon loaf class and was still able to achieve a high accuracy score. Therefore, it was chosen to use four categories of food which all have around 1300 pictures. These four classes were fish and chips, pizza, salad and vanilla ice-cream.

Only four classes of images were selected because the dataset was required to be small enough to be trained on a laptop computer. Shorter algorithm training times are beneficial for exploring various configurations of algorithms. It was considered that after investigating machine learning algorithms on four-class classification problem, it would be a relatively easy task to retrain the most accurate model using a dataset containing either more classes of food images.

4.3 Downloading the Dataset

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

To overcome this problem, permission to access the original ImageNet download files was requested. The permission was granted after agreeing to use the dataset only for non-commercial research and educational purposes. It allowed accessing the original data set with all images included. The original archives were downloaded using a python script and then extracted.

4.4 Creating the Dataset

Python class named Data was created to handle a creation of the image dataset. The class was designed to allow images to be loaded using various methods. The constructor of the class takes one argument: the relative path to the folder, where the data is located. Images can be loaded from a pickle file located in /pickle folder, extracted from .tar archives, or can be read directly from a /images folder. Loading the files from a pickle is the fastest method because Python only needs to read one file compared to thousands of files while loading separate images. A pickle is a binary file representing a Python object. To allow saving dataset to a pickle file, a Python method was created. The pickle file was structured as a list of classes of every picture, followed by a list of actual images. Method `try_download_images` was used to download the datasets from the ImageNet. This method takes two arguments - a username and the ImageNet access key. The `extract` method was used to extract downloaded .tar archive files into the pictures folder. Method `resize_image` was written to make resizing images in the dataset easier. It takes a list with height and width of requested image size and resizes the images. Finally, `train_test_split` method was written. It can be used to shuffle the data and split it into training and testing datasets. It takes an optional argument- a fraction that shows a percentage of data to be used for testing the classifier. These methods for manipulating datasets will be later used to preprocess the data for a classification task.

4.5 Summary

In this chapter, the requirements for the dataset were defined. The most important rule for the image dataset was to have many pictures of each class taken under real life conditions. To fulfil this requirement, it was decided to download food

images from the ImageNet. The dataset was formed from a four classes of food. Finally, a class for preprocessing and manipulating the dataset was created. In the following chapter, machine learning algorithms are introduced and classifiers that use these algorithms are built.

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 of how 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 classifier is one of the simplest machine learning algorithms. 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 a 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 a 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](#)).

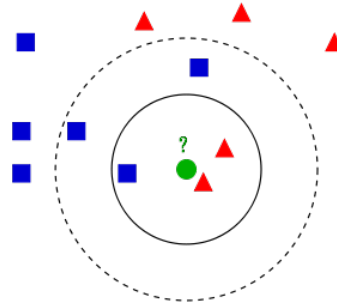


FIGURE 5.1: Example of k-NN classification. (Wikipedia 2017a)

The green circle could be classified either class depending on what value of K -neighbors is selected. If k value of 3 is chosen, then a green circle is assigned to the triangle class and if $k = 5$ circle is assigned to the square class.

5.1.2 Implementation of a K-Nearest Neighbours Classifier

The K-nearest neighbours classifier module was imported from the Scikit-learn package. The KNN classifier provided in this package provides many different parameters for modifying the algorithm. 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 pictures in the dataset, the height, weight and colour channels of images, were merged into one channel.

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 an 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). After training the 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 pictures. It was observed, that with this ratio the trained classifier achieved a higher accuracy.

In order to improve the classification accuracy, it was tried to find the optimal value of the number of neighbours. 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 of 18. However, the best accuracy score which was 36.28 percent was not an acceptable result, and it was necessary to improve the classifier.

In order to test if larger image sizes would help the KNN classifier to perform better, images in the dataset were resized to the increased 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 perform, 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 four classes. The accuracy of predicting it was 78.4 percent. The worst classified class was a salad which achieved only 0.02 percent accuracy.

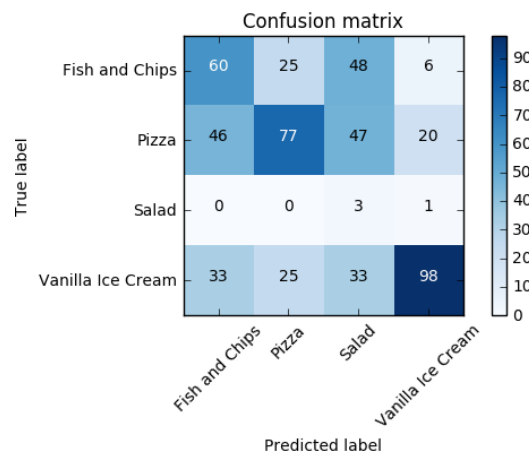


FIGURE 5.2: Confusion Matrix of KNN Classifier with $K = 18$

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 determined by humans, but the development of computing technology currently allows the use of various search techniques for better parameter optimisation.

There are two methods available for finding the best hyperparameters in the Scikit-learn package. These are an exhaustive grid search - GridSearchCV and a randomised 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 to try every single combination of hyperparameters before choosing the best set. It can be very computationally expensive and take a long time. Randomised search is usually able to find the set of parameters that can perform roughly equivalent to a grid search. The fact that that randomised optimisation could be a better optimisation technique was proven by Bergstra and Bengio who empirically and theoretically demonstrated that randomly chosen trials could be more efficient for hyper-parameter optimisation than trials on a grid ([Bergstra & Bengio 2012](#)).

It was decided to optimise the following hyperparameters in the K-nearest neighbours classifier: an amount of neighbours, a weight function that is used, and a distance function used. The number of neighbours considered was 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, 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 distances 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 of each other, it was possible to run hyperparameter optimisation task in a parallel, using multiple cores of a CPU. It took 30 minutes to run the exhaustive search on selected hyperparameters on a CPU with four cores. It was found that the algorithm was most accurate with $k = 8$ neighbours, the Euclidean distance function, and the uniform weight function. 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 optimised KNN classifier was able to predict both pizza and ice cream classes very accurately. This fact is also confirmed by a precision, recall and f score ([Table 5.1](#)). An average precision score of these two categories was around 0.8. However, the recall was lower - 0.53. The recall value was small because the classifier misclassified a lot of images from different classes as pizza and vanilla ice cream.

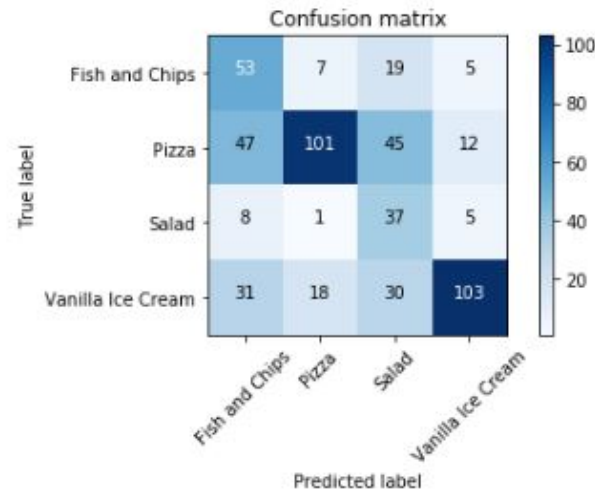


FIGURE 5.3: Confusion Matrix of KNN classifier with $K=8$, Euclidean Distance and Uniform Weights

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: Precision, Recall and F Scores of the best K-NN Classifier

5.1.4 Results

To conclude, with a right optimisation of hyperparameters K- nearest neighbours classifier can classify images quite accurately. Pictures 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. 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 is that it can be trained in a very short time. However, it is 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 according to [Boser et al. \(1992\)](#) is one of the most influential machine learning algorithms. The algorithm searches for separators between classes that maximises the distance between them. Advantages of this classifier are 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 this algorithm works well for a high dimensional data, it was decided to implement a classifier that uses the SVM algorithm.

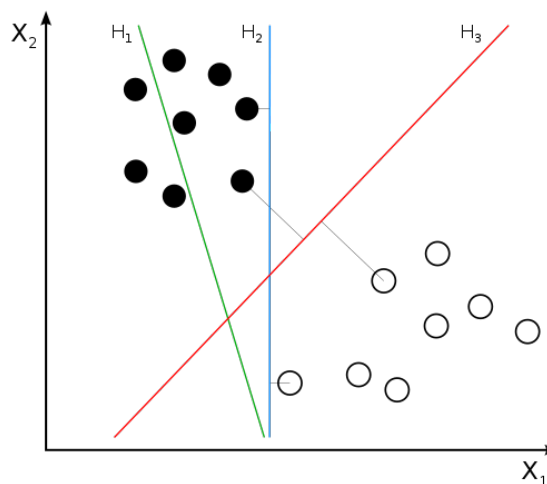


FIGURE 5.4: Three hyperplanes that could be used to split the data

H_3 is selected as a separator because it separates the classes with a largest distance between classes. ([Wikipedia 2017d](#))

5.2.2 Implementation of a SVM Classifier

Support vector machine module was imported from the Scikit-learn package to start experimenting with it. 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. The confusion

matrix was plotted (Figure 5.7), to get a better understanding why the accuracy score was so low. The confusion matrix showed that for some reason the classifier classified all the test dataset as a 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 too smooth.

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

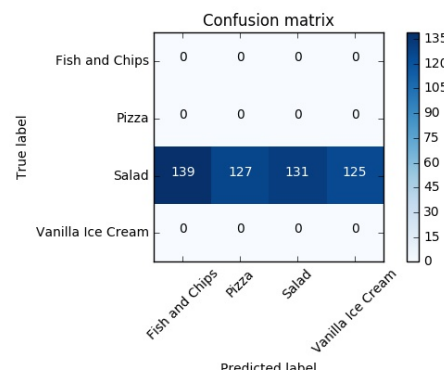


FIGURE 5.5: Confusion Matrix of Support Vector Machine Classifier Trained with Default Parameters

5.2.3 Tuning of SVM Hyperparameters

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

Firstly, it was chosen to increase the value of C. It was incremented 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 the predictions were very inaccurate. The C was increased to 100, but that did not improve the classifier.

Finally, the randomised grid search was used to find the best set of C and gamma hyperparameters was. 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 $\text{gamma} = 0.0001$ were the best set of parameters. It was reasonable that classifier performed better 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.

Then the support vector machine was trained on the whole training dataset with $C = 10$ and $\text{gamma} = 0.0001$. Despite the fact that the hyperparameters were optimal, the accuracy of the classifier on the training set was only 25 percent. Plotted confusion matrix (Figure 5.6) showed that the model still classified all test dataset as a salad class.

It was deducted that SVM algorithm was unable to classify the dataset correctly because a 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.

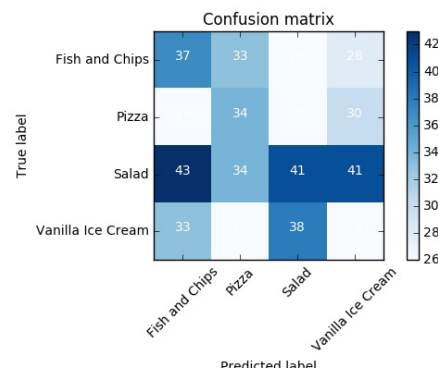
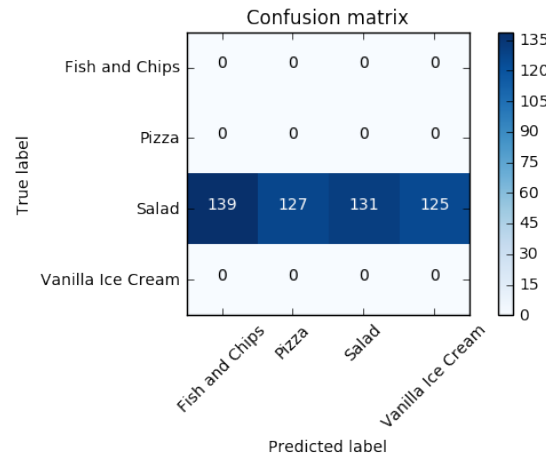


FIGURE 5.6: Confusion Matrix of SVM Trained with $C=10$

Numbers in squares are: $[37 \ 33 \ 26 \ 28]$, $[26 \ 34 \ 26 \ 30]$, $[43 \ 34 \ 41 \ 41]$, $[33 \ 26 \ 38 \ 26]$

5.2.4 Reduction of Dimentionalty 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

FIGURE 5.7: Confusion Matrix of SVM with $C = 10$ and $\text{Gamma} = 0.0001$

contain many patterns and are usually noisy, PCA should be a good solution.

The number of features was 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 $\text{gamma} = 0.0001$. 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 frequent misclassification 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 distinct separator between vanilla ice cream class and other classes.

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: Precision, Recall and F Scores of SVM Model on the Dataset Reduced with PCA

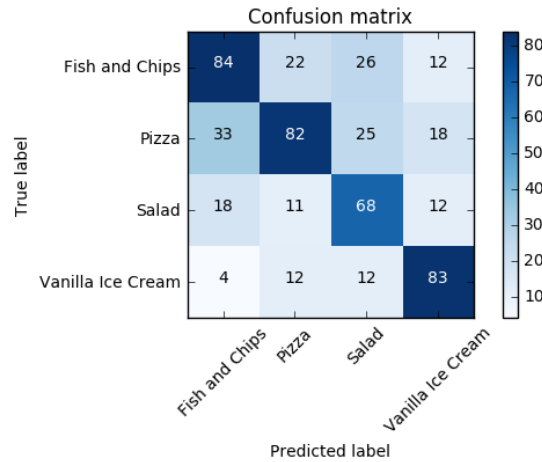


FIGURE 5.8: Confusion Matrix of SVM Classifier on the Dataset Reduced with PCA

5.3 Summary

To conclude, the average test accuracy of 60 percent was achieved by both k-nearest neighbours and support vector machine with PCA models. This result proves that machine learning models can be used for classifying the food images. However, 60 percent accuracy is not good enough for using these models for real world applications such as dietary assessment. Therefore, other classification methods need to be explored. In [section 2.1](#) it was discussed that since 2012 deep learning has become a dominant method for image classification. Deep learning methods are presented in the following chapter.

Chapter 6

Deep Learning approach for the Food Image Classification

In this chapter short introduction to deep learning is provided. Then the library used for building deep learning classifiers is described. Finally, deep learning models are constructed, and their performance is evaluated.

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 predecessor of deep learning was a linear classifier function. Linear models were designed to take a set of n input values x_1, \dots, x_n and associate them with an output y . These models would learn a set of weights w_1, \dots, w_n and compute their output $f(x, w) = x_1w_1 + \dots + x_nw_n$ (Goodfellow et al. 2017). This function was motivated by our understanding of how neurons work in living organisms.

The same function is used in today's deep neural networks. A deep neural network contains a set of linear 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). ReLU can be mathematically described as $f(x) = \max(0, x)$. 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](#)).

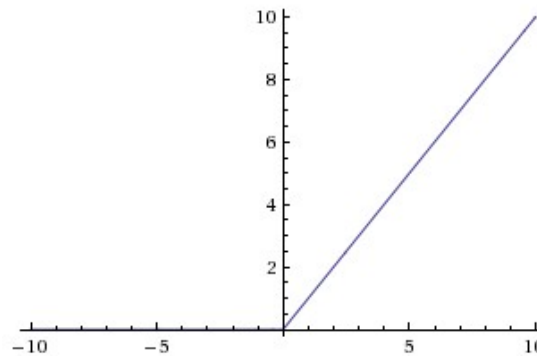


FIGURE 6.1: The ReLU Activation Function ([Karpathy 2016b](#))

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 [Abadi et al. \(2016\)](#). The system is flexible and can be used to express a wide variety of algorithms for deep neural network models. Computations in TensorFlow are described in a directed graph. First, all listed operations are added to the graph. Then a TensorFlow session is called which executes the operations in the 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 primary for conducting deep learning research, algorithms provided in this library are low level and require additional implementations to make them work on a particular 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 provided in the TensorFlow library. After spending a significant amount of time studying deep learning methods and writing implementations of deep learning algorithms, steps needed to train a neural network model with TensorFlow were eventually discovered. 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: TensorFlow Program that Adds 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 names cannot be used when computing a distance function and other functions that are used in machine learning. Therefore, class labels had to be transformed using the one-hot encoding. One-hot encoding converts every class name into a vector which has the length equal to the number of categories. 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 converted to one hot encoded values ([Table 6.1](#)). After

the transforming the labels of the dataset, it was possible to use it for training deep neural 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: One-Hot Encoding for 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 network models. 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 the 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 tiny impact to the class of a picture can be set to values that are 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 had been 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 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 optimisation a gradient descent and Adam optimizer was tried.

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

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

To do a batch optimisation, the food image dataset was split into 54 batches containing 87 food images each. From all trained linear classification models the classifier that used the Adam Optimizer with 0.05 learning rate and the batch optimisation 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 optimisation on the full dataset performed very poorly compared to the batch optimisation method. This happened because the weights on batch optimisation were updated 56 times more often. It can also be observed that Adam Optimizer was a better optimisation 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 optimise it to classify the images with a reasonable accuracy.

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

TABLE 6.2: The Influence of the Hyperparameters to the Accuracy of the Linear Classifier

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, which could learn a better representation of the data. The structure of the model was designed in the following way: 30,000 neurons in the input layer, two hidden layers containing 256 neurons each and a final layer containing four neurons that map to the four image classes. A ReLu activation function was used in hidden layers to introduce a non-linearity to the network (Figure 6.3).

```
#The network model
y1 = tf.nn.relu(tf.matmul(X, w1) + b1)
#y1 = tf.nn.dropout(y1, dropout)
y2 = tf.nn.relu(tf.matmul(y1, w2) + b2)
#y2 = tf.nn.dropout(y2, dropout)
logits = tf.matmul(y2, w3) + b3
```

FIGURE 6.3: Snippet of the 3 Layer Network Model

After starting to train the model, it was observed that after a 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 too high. However, 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 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 experimenting 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. The neural network that had fewer 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 the Hyperparameters to the Accuracy of the Multi-layer Neural Network

6.5.1 Results

To conclude, a multi-layer neural network performed a little better than a linear classifier. The best neural network design was a five layer neural network with 30,000 input neurons, 256 neurons in the each hidden layer and four 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 than the accuracy of the best linear model.

However, the design process of multi-layer neural networks was much more complicated. There are no formal definitions that state how many layers a network should have or how many neurons should be in these layers. Therefore, a collection of experiments 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 Tensor-Flow

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 by [LeCun et al. \(1989\)](#). A typical layer of a convolutional neural network consists of three stages. In the first step, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is passed through 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 particular location in the network with a summary statistic of the nearby outputs ([Goodfellow et al. 2017](#)). The 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 6.4](#)). 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.

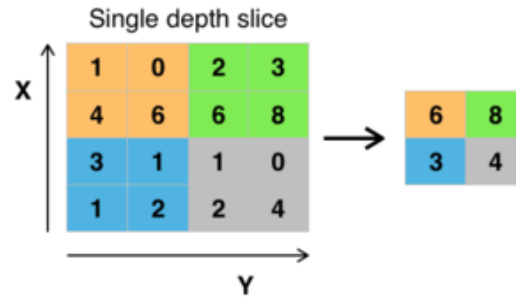


FIGURE 6.4: Illustration of the Max Pooling

The function outputs a maximum output in every 2 by 2 square ([Wikipedia 2017b](#))

6.6.2 Implementation of a Convolutional Neural Network

Training a convolutional neural network requires sufficient amount of 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 one by one convolution to the input picture 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 seven by seven convolutional images. These images are passed into the fully connected layer, and finally, the class of a picture is calculated in the final layer [Figure 6.5](#)

```
# 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 * 13])

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

FIGURE 6.5: The Convolutional Neural Network Model in 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 small 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, the classifier was tested on the test dataset. It achieved 38.12 percent accuracy, which was not a good score.

The algorithm was modified to accept RGB images to test whether colour channels have an influence on the classification accuracy. After the network had been 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 a major 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. The picture illustrating different layers in this network is shown in Figure 6.6.

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: The First two Convolutional Models Tried

conv - convolutional layer, *fc* - fully connected layer, *o* - output layer

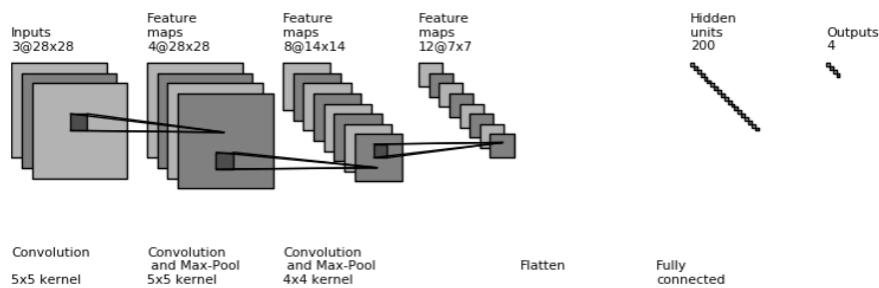


FIGURE 6.6: Layer Diagram of the Convolutional Neural Network

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. It was decided to add dropout layers to the network, to fix this problem. Dropout is a method used to prevent neural networks from overfitting. The key idea is to randomly drop units (along with their connections) from the neural network during training (Srivastava et al. 2014). Dropout function prevents neurons from co-adapting too much.

A 50 percent dropout rate was added to the every layer in the network. The Introduced drop layer helped to prevent the network from overfitting. However, because of the high dropout ratio, the network was unable to optimise the weights, and it classified worse than previously trained models. The learning rate of the network was increased, to solve this problem. However, this solution did not work. Larger step size caused the optimisation 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 the improvement was only marginal (49.04 percent accuracy for RGB test data and 40 percent accuracy for the grayscale image classifier). Lastly, a network that kept 90 percent of the neuron connections was built, but it decreased the accuracy of the model achieved 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 were no more hyperparameters that could be further optimised it was decided to build a neural network that could handle larger size images.

6.6.4 Designing Convolutional Neural Network for Larger Images

It was expected that larger image size would lead to a better accuracy score when performing image classification. Larger image sizes allow more information to be preserved in pictures. The network was redesigned to take 56x56x3 pictures as an input. One more convolutional and a max-pooling layer were also introduced to the architecture of the network. Then, it was tried to find the optimal learning rate. The network was trained with 0.003 learning rate and a 20 percent dropout 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 optimisation algorithm kept overshooting the minimum. Therefore, the training accuracy and loss stayed the same during the training of the model.

The learning rate was gradually lowered, to find the optimal learning rate. It was found that the largest learning rate that could be used was 0.0015. With this learning rate, the algorithm was able to learn the optimal weights. When tested, it achieved 60.72 percent test accuracy. It was observed that after 30 epochs the optimisation algorithm was still improving. Therefore, the number of epochs were increased to 100. That allow the weights to be optimised more precisely. After testing the model on the test dataset, it was observed that the accuracy increased to 62.26 percent.

Finally, the architecture of dropout layers was modified, to test whether it could improve the performance of the classifier. In the previous model, dropout was used in every layer except the fully connected one. The dropout was added to the fully connected layer too. However, after adding the dropout, the learning rate value of 0.0015 became too large. After decreasing the value of the learning rate to 0.001, it became possible to train the network. The 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 model to use dropout only in a fully connected layer. Because only one dropout layer was used, the dropout probability was increased to 50 percent. The network was trained with a 0.001 learning rate. However, training accuracy of the model improved very slowly. Therefore, the learning rate was increased, until its optimal value of 0.002 was found. The trained model was tested on the test data set. The accuracy of the model on the test dataset was 51.54 percent. The table of CNN networks that were constructed for 56x56x3 images can be seen in [Table 6.5](#).

6.7 Transfer Learning Technique

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. It usually takes multiple weeks to train these models. For example, the Google 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

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: Configurations for Convolutional Neural Networks tried with 56x56x3 Pictures

it 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 ([Tensorflow 2017a](#)). Usually, to save time, only the two final layers of a neural network are retrained. The reason why retraining only the last two layers is sufficient is that convolutional layers tend to activate to patterns of different shapes, colours and edges. Such patterns appear not to be distinct 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 Inception v3 model. A python script for loading and retraining this model is publicly available in the Tensorflow's Github repository ([Tensorflow 2017b](#)). The script provides the ability to tweak hyperparameters such as the 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 6.7: Command Used to Retrain the Inception v3 Model

6.7.2 Retraining 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 6.7](#).

When `retrain.py` was executed, it 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 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 6.7](#)). It was observed that just after ten 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 100 pictures of unseen food correctly. This accuracy is much higher than accuracies achieved by every other classifier.

To obtain a better understanding how can this classifier perform on the real-world data 5 example pictures 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 6.9](#)) 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 = 94.8%  
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 6.8: Retraining of the Classifier, 96 Percent Accuracy was Reached in 10 Epochs



FIGURE 6.9: Deep Pan Pizza that was Missclassified as a Fish

6.8 Summary

To conclude, it can be clearly seen that deep learning networks 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 be used in practice because the error rate is very low.

However, finding the optimal parameters for the deep neural network models is a much harder task. It requires an excessive effort to design an architecture of a neural network. What structure of the network will work on a particular task, cannot be known beforehand. Therefore, many architectures have to be tried to find a suitable architecture. Even then it is unknown if the chosen architecture is actually the best because the configuration space is gigantic.

In the next chapter, the results of all classification methods are compared.

Chapter 7

Results

In this chapter, the results of each classification method are compared. Then, advantages and disadvantage of these methods are discussed.

7.1 Comparison of Accuracy Scores

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. The convolutional neural network is in the second place with 69.34 percent accuracy. It has to be considered that with transfer learning a classification model was not built from the ground up. The difference in the accuracy score between other classification methods is not that marginal. The lowest accuracy - 51.53 percent was achieved by the Linear classifier while the support vector machine model achieved 60.73 percent accuracy. From these results, it can be concluded that the convolutional neural network achieved a much higher accuracy score than other classifiers that were built from the ground up.

7.2 Comparison of Time Complexity of the Classifiers

There are three kinds of time complexities that can be considered for evaluating machine learning classifiers. The first is a build-time complexity - how long does

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

it take to create a classification model. The second is a training complexity - the amount that it takes to train a classification model. The final is a runtime complexity - how long does the model take to classify an item.

The K-nearest neighbours classifier was simple to build. It only required a hyperparameter optimisation which did not take long. The training of the algorithm was also fast. In order to train this algorithm, the dataset was simply saved to the memory. However, the runtime performance of this classifier was not that great. During the runtime, this algorithm has to calculate distances between an unseen data sample and every other element in the dataset. Later, these distances are compared to find the K closest distances. Because these are computationally expensive tasks, it takes a long time for K-NN classifier to make a prediction.

It was a much more difficult task to get Support Vector Machine Algorithm to work. Despite the fact that the algorithm is designed to work on a high dimensional data, the food image dataset was too complex for it. The only way to use this algorithm was by firstly applying the PCA to the dataset. Therefore, building this model required a significant amount of experimentation and took a much longer time, before actual results were seen. It took a very long time to train the support vector machine. Prediction time was faster than the prediction time of the K-NN classifier but was not instantaneous.

The linear classifier is a very simple classification model. Therefore, it did not take much time to create this model. Training of this algorithm was fast because the only things calculated were a matrix product of training images and weights, and updated weights, computed by an optimisation algorithm. Prediction time was almost immediate.

The Deep neural network required much more experimentation, to make it work. Training of the algorithm took a long time, but the prediction was almost as fast as the linear classification model.

The convolutional neural network required even more engineering work. Some CNN models architectures that were tried had a very low classification accuracy. Various models had to be tested, in order to finally find the architecture that was working. Training of the algorithm took the longest amount of time from all of the algorithms implemented. However, the classifier was able to predict the classes of new pictures very quickly.

The transfer learning technique was simple because the model with a python script was provided by Google. In a case where one has to write a script that retrains the final layer, it can be a very hard task. The calculation of the bottlenecks took a long time. Therefore, the training complexity of this algorithm was high. However, for classifying the images, this algorithm took the same time as other neural networks.

To conclude, the algorithm that took the least time to built was KNN. Deep neural network models took the longest amount of time to build because it was a very hard task to find an appropriate network architecture. The classification algorithm that took the least time to train was KNN. For fastest prediction times, the linear classifier was the best option.

7.3 Possible Improvements and Limitations of Classification Models

The KNN classifier, the SVM, and the linear classifier model cannot be considerably improved. The Hyperparameters of these algorithms were optimised by using a grid or random search optimisation methods. Therefore, the only possibility for improving these models is training on a more powerful machine and using larger sizes of images.

In contrast, the accuracy of the deep neural network and the convolutional neural network could still be considerably improved. It can be done by adding more layers or modifying the structure of the networks. However, it cannot be mathematically calculate which structures of neural networks work the best. The number of ways

that these networks can be configured is infinite. Therefore, finding optimal structures of neural networks is possible but hard a task.

The difficulty of finding optimal structures of neural networks was the main limitation observed in this project. Choice of architecture of a network directly impacted the performance of the model. The number of layers and neurons in each layer also was important for the performance of the model. Moreover, the correct activation function and weight optimisation function had to be selected in order for the classifier to work properly. Finding all these parameters was a very hard task. It was unknown how to approach this problem. Some parameters were correlated (e.g. learning rate should be increased if a dropout is added), other parameters were connected but it was hard to observe patterns between their values, and some other parameters seemed to be independent from others.

7.4 Summary

To conclude, transfer learning achieved the highest classification score. However, this model was not built from the ground up. The best model that was built was the convolutional neural network which achieved 69.34 % accuracy. It was also stated that deep neural network models are very hard to develop and take much longer time to train when compared to other classification methods. That is a trade-off between using a deep learning model and other machine learning classifier. In the next chapter, the conclusions of the work are deducted.

Chapter 8

Conclusions

The objectives of the project were to practically understand machine learning algorithms that can be used for classifying images. To accomplish this task, a suitable food image dataset was found and food pictures were preprocessed. Then, image classifiers using various machine learning algorithms were built. Finally, these classifiers were trained on the accumulated dataset and their performances were evaluated.

8.1 Lessons Learnt

During the progress of this project, it was gradually discovered why image classification is such a hard problem. The problem that algorithms in image recognition task face is that images are very dynamic and are susceptible to different kind of variations. It was practically observed that machine learning algorithms that are successfully used in practice for tasks such as spam detection, credit card fraud detection and other tasks were unable to classify images with a sufficient accuracy. It was also observed that machine learning algorithms take a very long time to train. This is the main barrier for using machine learning techniques for vision. Overall, the project was successful. A framework for converting pictures to the dataset of images was built. Then, machine learning classifiers that used different classification algorithms were developed and their performance was systematically evaluated.

8.2 Limitations

The main limitation of the project was that machine learning algorithms take a very long time to train. Images are very high dimensional data, and an enormous amount of the computing power is required to train a good machine learning model for image classification. Therefore, this project was limited to using only four image classes. These four classes are not able to represent all types of food that is eaten by people.

The second limitation that was faced during this project was that it was not possible to explore larger deep learning networks on the hardware setup used. Therefore, a maximum accuracy score of the deep neural network and the convolutional neural network could not be reached. With deeper architectures of neural networks, it would be plausible to achieve better accuracy scores.

8.3 Future Work

To extend this project, one classification method should be explored more thoroughly and a classifier that distinguishes between a greater amount of food classes should be built using this model. A convolutional neural network would be the most promising model to explore further because it achieved a high accuracy score. Also, researchers have proven that a convolutional neural network is the best method that can be used for image classification. Furthermore, there have been few studies about a food classification with convolutional neural networks. A good number of classes for a new model would be around 100 types of most popular dishes. Because of the increased complexity, a GPU cluster will be needed to train this model. If this new model achieved the high accuracy rate, it would be possible to use this model in dietary assessment applications.

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.
- 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.

- 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. (2016a), ‘Cs231n convolutional neural networks for visual recognition’. [Online; accessed 2016-12-28].
URL: <http://cs231n.github.io/classification/>
- Karpathy, A. (2016b), ‘Cs231n convolutional neural networks for visual recognition’. [Online; accessed 2017-01-28].
URL: <http://cs231n.github.io/neural-networks-1/>
- Kingma, D. P. & Ba, J. (2014), ‘Adam: A method for stochastic optimization’, *CoRR* **abs/1412.6980**.
URL: <http://arxiv.org/abs/1412.6980>
- 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>
- Shalev-Shwartz, S. & Ben-David, S. (2014), *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, New York, NY, USA.
- 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.
- 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>
- Tensorflow (2017a), ‘How to retrain inception’s final layer for new categories — tensorflow’, https://www.tensorflow.org/versions/master/how_tos/image_retraining/. [Online; accessed 2016-11-29].

- Tensorflow (2017b), ‘tensorflow/retrain.py at master’, https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py. [Online; accessed 2017-03-03].
- Ting, K. M. (2011), *Encyclopedia of machine learning*, Springer.
- Wikipedia (2017a), ‘K-nearest neighbors algorithm — Wikipedia, the free encyclopedia’, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg. [Online; accessed 2017-02-28].
- Wikipedia (2017b), ‘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].
- Wikipedia (2017c), ‘Precision and recall — Wikipedia, the free encyclopedia’, https://en.wikipedia.org/wiki/Precision_and_recall. [Online; accessed 20-March-2017].
- Wikipedia (2017d), ‘SVM — Wikipedia, the free encyclopedia’, [https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_\(SVG\).svg](https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_(SVG).svg). [Online; accessed 2017-02-28].
- 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.