Division of Computing Science and Mathematics
Faculty of Natural Sciences
University of Stirling

Object Detection on Large-Scale Egocentric Video Dataset

Alexandra Maha Abbas

Dissertation submitted in partial fulfilment for the degree of Master of Science in Big Data

September 2018

Abstract

Egocentric Vision is an emerging field within Computer Vision that focuses on images and video recorded by head-mounted cameras. More and more egocentric datasets, including the EPIC-KITCHENS dataset, appear which give access to a unique viewpoint of people's interactions, attention and intention.

Problems in Egocentric Vision, like activity recognition and video summarization, all involve some form of object detection. Therefore, advances in egocentric object detection can help many areas in the field. This dissertation project tackled an object detection challenge on a large-scale egocentric video dataset, EPIC-KITCHENS.

The goal was to train a state-of-the-art object detection model that is capable of detecting all 352 object classes in the dataset. Further goal was to overperform the baseline results produced by Damen *et al*.

Following a thorough literature review in object detection and Egocentric Vision, a Convolutional Neural Network, namely RetinaNet was chosen to tackle the object detection problem on EPIC-KITCHENS. RetinaNet was trained using an open-source Keras implementation and Google Cloud Engine.

Unfortunately, due to mainly computing power constraints the dataset size and the number of object classes were reduced. As a result, RetinaNet inference model was able to detect a single object class. However, results produced by RetinaNet were still comparable with the baseline results in a single object class.

RetinaNet overperformed the baseline results and showed good generalisation ability to unseen environments.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following:

- Data collection and data labelling discussed in Section 2.4 was carried out by Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price and Michael Wray as described in [1].
- The technology review in Section 2.1 was largely taken from Andrew Ng [2].
- The code used for training and evaluating the object detection model was created by Fizyr B.V. *(www.fizyr.com)* and was used in accordance with the licence supplied.
- The code used for data sampling and data preprocessing discussed in Section 4.2.2 was written by me.

Signature **Date**

Acknowledgements

First, I would like to thank my thesis supervisor, Dr Kevin Swingler of the University of Stirling, for his support, guidance and useful comments throughout this dissertation project. In addition, I would like to extend my thanks to Dr Simon Jones for his role as a second marker.

Furthermore, I would like to express my gratitude towards Andrew Y. Ng for introducing me to Computer Vision via his exceptional online course on Coursera.

Thank you to The Data Lab for providing the funding of my placement on this course and the opportunity to attend several valuable events throughout the year.

Finally, I would like to thank my partner, Tamas Soos for his support and advice.

Table of Contents

List of Figures

List of Tables

1 Introduction

The amount of visual data exploded in the world in the last couple of years. According to Cisco, by 2021 video will make up 82% of consumer internet traffic [3]. Snapchat users shared 3,5 billion snaps each year in 2017 [4] and more than 400 hours of video were uploaded to YouTube every minute in 2015 [5].

This explosion of visual data is largely due to the growing number of sensors in the world. Most people carry around smartphones supplied with one, two or even three cameras. More and more people use wearable devices equipped with cameras. Cars use cameras to monitor roads and assist drivers in parking. Medical professionals use visual sensors to detect tumours [6], monitor the blood flow [7] and assist surgeries [8].

In order to utilize these vast amounts of data, it is critical to develop algorithms that can understand visual data, since it would be impossible to manually catalogue and annotate all images and video produced. Computer Vision is the study of visual data, it is an interdisciplinary field that touches on many areas of science, engineering and technology.

1.1 Computer Vision

Computer Vision is concerned to give the ability of understanding images and video to computers.

Images on computers are stored as grids of pixels. In case of multichannel colour images, each pixel is defined by a colour and stored as a combination of three additive primary colours: red, green and blue. By combining different intensity of these colours, which are called RGB values, we can represent any colour.

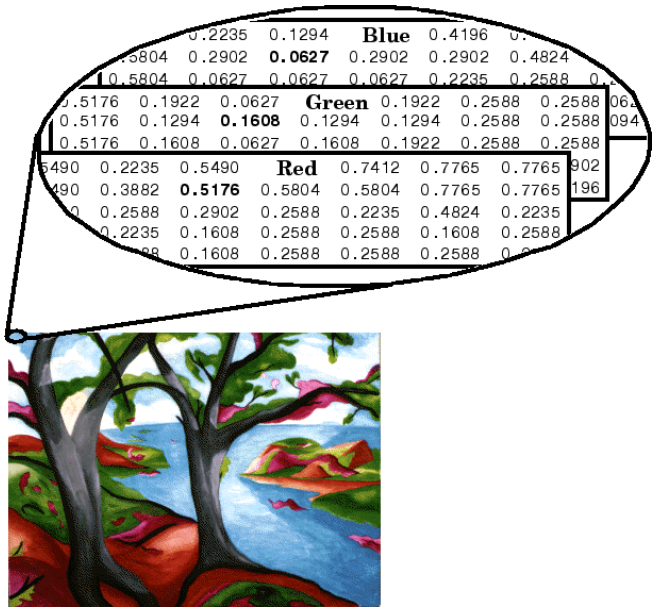

Figure 1. Image represented by RGB values from MATLAB Documentation [9]

Pixel values stored in the RGB matrix stack can be manipulated and analysed to process images, detect lower as well as higher level features like edges, shapes, simple and complex objects.

Remark Grayscale images are represented by a single grid of pixels where each pixel value is in the range 0 (black) to 1 (white). However, note that there are several different representations of grayscale images.

1.1.1 Problems in Computer Vision

Problems in Computer Vision can be categorised into several main, often overlapping tasks defined as follows.

Image classification - Given an image, a classification algorithm outputs a category label from a fixed set of categories that describes the content of the image. It is assumed beforehand that the image contains a single object of interest. Image classification is the most basic Computer Vision task that provides the foundations for more complex tasks like object detection and segmentation.

Semantic segmentation - Semantic segmentation assigns each pixel of an input image to a category label from a fixed set of categories.

Classification with localization - Given an input image, it outputs a single category label as well as a bounding box that localizes the object of interest in the image. Similarly, to image classification, we assume that the input image contains a single object of interest.

Object detection - Object detection is the most important and a core problem in Computer Vision. Given an input image that may contain several objects of interest of different category labels, an object detector outputs bounding boxes for each object along with category labels that describe the content of each bounding box. Unlike classification with localization, object detectors may have varying number of outputs.

Instance segmentation - Similarly to object detection, the goal is to assign a category label and a location to each object of interest in the image. However, rather than predicting bounding boxes, it predicts segmentation masks that determines which pixel of the input image corresponds to each object of interest. Instance segmentation is a combination of semantic segmentation and object detection.

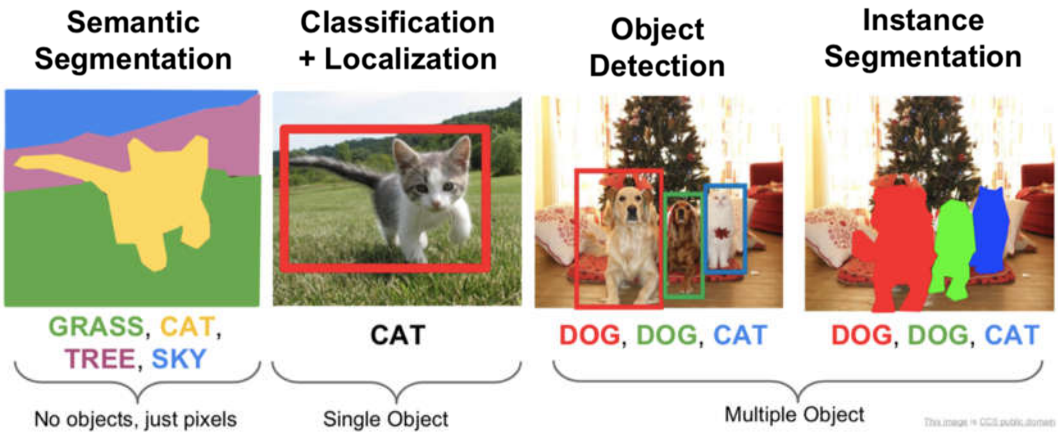

Figure 2. Figure showing the main tasks in Computer Vision from Stanford CS231n [10]

This dissertation will tackle an object detection challenge detailed in Section 1.3.

1.2 Egocentric Vision

Egocentric Vision is a subfield of Computer Vision that focuses on images and video recorded by wearable, generally head-mounted, cameras that give access to a unique aspect of people's interaction with objects and other people, their attention and intention.

Egocentric Vision enjoys rising popularity due to recently introduced wearable devices like Google Glass [11], Microsoft HoloLens [11] and Magic Leap [12], and related technologies like Augmented Reality (AR) [13] and Virtual Reality (VR) [14]. Wearable cameras try to promote the habit of "life logging" which refers to the activity when one wears an "always on" camera on one's head or neck that records his or her daily experiences. Particular cameras, like the head-mounted GoPro, are already popular in specific sports, for instance skiing, surfing or biking.

Consequently, the Computer Vision research community started to explore the exciting field of egocentric vision as a result of the appearance of large egocentric video datasets [15][16][17]. According to S. Bambach [18] research in the area can be categorised into three subdomains: (1) object recognition/detection; (2) activity recognition/detection; and (3) video summarization. Egocentric videos propose new challenges to the research community such as (1) constantly moving cameras; (2) not linear or unpredictable movement of cameras; (3) disappearing and reappearing objects; furthermore (4) objects often blur. Further characteristics of egocentric video include "central nature", i.e., objects of interest usually take central position in the field of view.

Egocentric Vision offers various new challenges, applications and research opportunities besides third-person images and video. According to [18], beyond the research tackling specific object recognition/detection challenges, almost all research focused on egocentric action detection or summarization of "life logging" videos use some form of object detection. This suggests that the importance of object detection in Egocentric Vision is crucial for developing more advanced methods.

1.2.1 Early Research

Early research in the field was focused on studying one's activity based on the objects that are in interaction with one's hands. S. Bambach [18] introduced the following three works in egocentric object recognition/detection that served as a basis for future research in the field.

One of the first attempts to perform object detection on egocentric video was done by Ren and Philipose [19] in 2009. They collected a small video dataset of everyday objects and used hand-crafted features along with classic Machine Learning techniques, namely SIFT (Scale-invariant Feature Transform) algorithm [20] and multi-class SVM, to perform object detection. Unfortunately, the model performed quite poorly due to various egocentric challenges like (1) limited visibility of texture of objects; (2) background clutter; and (3) hand partly or entirely covering object of interest. Consequently, they suggested predicting motion and location of objects prior to object detection as well as hand detection as future research directions.

Ren and Gu [21] built on the recommendations of [19] and used motion detection to find objects of interest in the field of view. In addition, they built on the observation that objects of interest are in contact with the hands and usually take central position in the view, furthermore motions performed by the hands are generally small and horizontal. They tested on the dataset collected by Ren and Philipose [19] and achieved 20% and 46% accuracy with SIFT [20] and HoG (Histogram of Oriented Gradients) [22] based systems consequently.

Later A. Fathi et al. [23] similarly used the finding that objects of interest tend to appear in the centre of the video. They decrypted objects from activities using the assumption that objects that commonly co-appeared in the view are part of the same activity and tend to

appear together. As part of the solution, they created background models for short parts of the video and treated each object that are not in interaction with the hands as part of the background. They tested the model on the dataset created by Ren and Philipose [19] and outperformed [21] by 19% in segmentation error rate.

The lack of large-scale datasets caused the early Computer Vision to rely heavily on hand-crafted features along with feature descriptor algorithms like SIFT [20] and HoG [22] since carefully hand-engineered features could compensate in some degree the small size of available datasets. Producing datasets suitable for object detection were even more difficult compared to object recognition as a consequence of the high cost of producing bounding box labels.

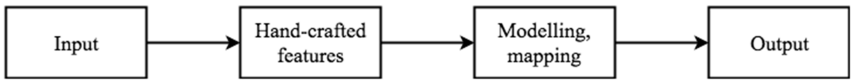

Figure 3. Workflow of classic Machine Learning

Given the above, it can be concluded that (1) the use of hand-engineered features and feature detector algorithms; (2) the lack of large-scale datasets; (3) the use of classic Machine Learning algorithms e.g. Support Vector Machines; (4) increasingly complex architectures; and (5) a 2-stage-long workflow consisting of feature extraction and mapping; characterized the early era of Computer Vision.

1.2.2 Recent Work

Better understanding of Neural Networks, appearance of GPUs and Big Data, led to the rise of Deep Learning in Computer Vision. Even though Convolutional Neural Networks (CNNs) were known to researchers much earlier [24], it was the paper by A. Krizhevsky, I. Sutskever, and H. Geoffrey E. [25] that convinced the community about the power of Deep Learning in Computer Vision. AlexNet, named after A. Krizhevsky, won the largest image recognition competition (ILSVRC) in 2012 and outperformed all the methods that were based on hand-engineered features. Since then many different variations of CNNs have been built and used for third-person and first-person image classification, object detection and segmentation tasks. In Section 2.3 a selection of state-of-the-art CNNs will be introduced and compared.

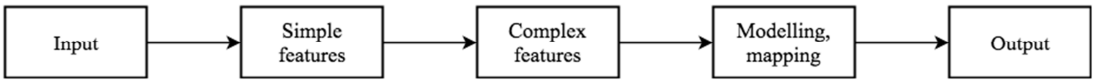

Figure 4. Workflow of Deep Learning

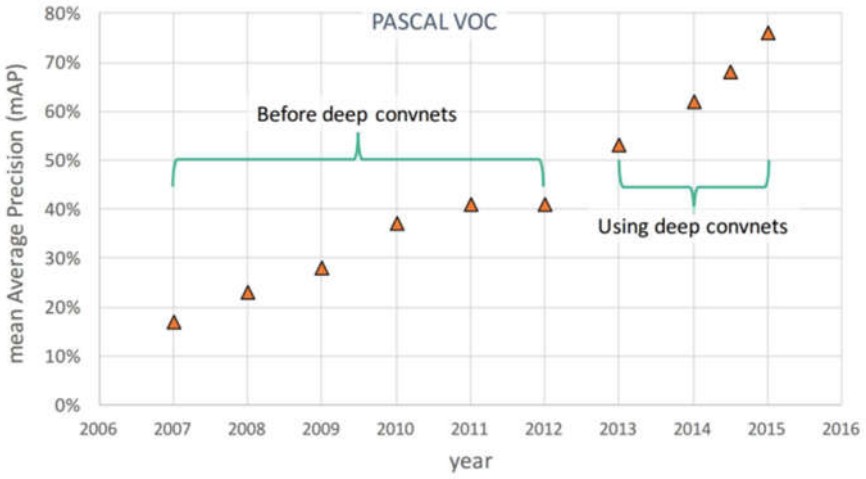

Figure 5. Impact of Deep Learning on the performance of object detection [10]

1.2.3 Applications

Egocentric Vision gives access to a new perspective that is linked to the camera wearer's ongoing experience, intention and attention. Consequently, this new perspective and the rising popularity of wearable devices offer new ways of applying Computer Vision systems.

Egocentric Vision, beyond traditional tasks such as activity recognition and video summarization, is able to produce such unconventional systems that learns where to pay attention to recognise activity [26], predicts functionality for regions in environment [27], predicts future walking trajectory [28], reveals physical interaction with surroundings [29], quantifies moments of eye contact [30], detects what a group looks at [31], or even predicts the wearer's identity [32] based on first-person video footage.

These systems could be applied in robotic vision, law enforcement, health monitoring, caregiving, Human-computer Interaction (HCI) and Augmented Reality (AR).

1.2.4 Egocentric Datasets

Following the appearance of large-scale third-person benchmark datasets like ImageNet [33] in 2009, PASCAL VOC [34] in 2010 and MS COCO [35] in 2014, egocentric datasets started to appear as well. Commonly used egocentric datasets are compared in Table 1. Note, that egocentric datasets are generally recorded in form of video, hence the number of frames and sequences are shown.

	Non-scripted?	Native env?	Year	Frames	Sequences	Object BBs	Object classes	Participants	No. env.s
EPIC-KITCHENS [1]	✓	✓	2018	11.5M	432	454.158	352	32	31
EGTEA Gaze+ [15]	x	x	2018	2.4M	86	0	0	32	1
BEOID [17]	x	x	2014	0.1M	58	0	0	5	1
ADL [36]	x	✓	2012	1.0M	20	137.780	42	20	20
CMU [37]	x	x	2009	0.2M	16	0	0	16	1

Table 1. Comparative overview of commonly used egocentric datasets taken from [1]

EPIC-KITCHENS [1] is a large-scale video dataset recorded in native kitchen environments with participants belonging to 10 nationalities. Participants performed non-scripted activities in their own kitchens that were recorded for three consecutive days. EGTEA Gaze+ [15] includes scripted preparation of 7 different meals by 5 participants in a single kitchen environment. BEOID [17] or Bristol Egocentric Object Interactions Dataset was recorded in 6 locations with 5 participants, it includes object interactions like coffee preparation. ADL [36] or Activities of Daily Living captures 18 daily indoor activities involving 42 different objects and 20 participants. It is annotated with bounding boxes in all frames. CMU [37] captures the preparation of 5 different meals by 18 subjects in a single kitchen environment where each frame is labelled for an action.

The largest egocentric dataset to date is by far the EPIC-KITCHENS dataset with 11.5M images. Besides EPIC-KITCHENS, all datasets use scripted activities where the participants were told what actions to perform. Most datasets were recorded in a single non-native environment. Only EPIC-KITCHENS and ADL are annotated for object bounding boxes where EPIC-KITCHENS has 352 object classes while ADL has 42.

Ultimately, from all egocentric datasets EPIC-KITCHENS and ADL are suitable for egocentric object detection, however EPIC-KITCHENS is substantially larger and annotated for four times more bounding boxes. This justifies the choice of using EPIC-KITCHENS for the task of egocentric object detection.

1.3 Scope and Objectives

Egocentric object detection is an important and emerging field in Computer Vision which can be applied in many practical areas. This dissertation project aims to tackle an object detection challenge on the large-scale egocentric video dataset, EPIC-KITCHENS [1]. EPIC-KITCHENS is a freshly published dataset with only a few publicly available results. At the time of the dissertation writing, there is no publicly available results for the dataset besides the benchmark results presented in [1].

This dissertation was an exploratory research into Deep Learning and object detection. The scope of the project can be defined as follows:

- review current literature in egocentric vision and object detection
- research and identify state-of-the-art object detectors
- investigate and assess available implementations of proposed object detector
- preprocess EPIC-KITCHENS dataset such that it is suitable for proposed object detector
- set up environment for training and evaluating proposed object detector
- obtain optimal configuration of implemented model through systematic hyperparameter search
- compare results gained to baseline results presented in Damen *et al.* [1]

The ambitious intention of this work is to build a model that is capable of detecting 352 object classes on the EPIC-KITCHENS dataset, furthermore to outperform the baseline results by Damen *et al.* [1].

1.4 Achievements

Great amount of literature had to be reviewed in relatively short time frame. The literature review covered the functionality and building blocks of Convolutional Neural Networks (CNNs), the foundations of convolutional object detection, and the architecture as well as performance of state-of-the-art object detectors.

RetinaNet [38] was proposed to tackle the object detection challenge on the EPIC-KITCHENS dataset. Following the assessment of several implementations of RetinaNet [39][40][41], the Keras RetinaNet framework [41] was chosen to train and evaluate the model on EPIC-KITCHENS.

One of the great challenges was to comply with the computing power and time constraints. A single GPU was available for this project, in contrast Damen *et al.* [1] used 8 GPUs to train a model over 6.5 days. This required the use of smaller dataset and reduced number of object classes. As a result, the model was trained to detect a single object class on sampled images from EPIC-KITCHENS.

Since Damen *et al.* [1] recorded the results for each object class, results produced by the specialised RetinaNet model and the baseline results were still comparable. Within the reduced scope, the built RetinaNet model overperformed the baseline results in a single class.

RetinaNet implementation presented in this dissertation can be extended by additional object classes, larger dataset, additional GPU power, different transfer learning protocol and different training style in the future.

1.5 Overview of Dissertation

This dissertation is organised into several chapters starting with the literature review and finishing with the description of the practical implementation and results of the proposed object detector.

Section 2 - Background details the literature research carried out in the beginning of the project. It covers the theoretical foundations of CNNs and convolutional object detection. Furthermore, it presents a selection of state-of-the-art object detectors including the detector used for producing the baseline results in [1]. Lastly, the EPIC-KITCHENS dataset is reviewed in detail including the data collection, annotation and baseline results.

Section 3 - Methodology describes best practices and guiding rules used for implementing the proposed object detector. It details the usage of data splits, cross validation and methods for hyperparameter optimization.

Section 4 - Implementation presents the practical decisions and steps taken while training, validating and testing RetinaNet on EPIC-KITCHENS. It also describes the data preparation steps including the method used for sampling images from the dataset.

Section 5 - Results and Analysis presents the experimental setups detailing the different hyperparameter settings used and the results obtained. In addition, it includes comparison of results to the baseline results.

Section 6 - Discussion is a short chapter focused on comparing the implementation and results of RetinaNet with the baseline. It discusses issues that help to assess the obtained results in context.

Section 7 - Conclusion gives an overview of the finished work including a critical reflection and recommendations for future work.

2 Background

This section presents the literature review corresponding to this dissertation project. It covers in detail the building blocks of Convolutional Neural Networks and the foundations of convolutional object detection. Furthermore, it introduces and compares four state-of-the-art object detection networks. It also reviews the baseline implementation and results presented in Damen *et al.* [1].

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are powerful state-of-the-art image processing methods widely used in Computer Vision for image recognition, object detection and segmentation. As mentioned earlier, it was AlexNet [25] by A. Krizhevsky, I. Sutskever, and H. Geoffrey E. that popularized CNNs in Computer Vision research.

Besides 2D images, CNNs can be successfully applied to 1D e.g. electrocardiogram as well as 3D data e.g. computed tomography (CT) scan, however this section reviews CNNs from the aspect of 2D image processing.

2.1.1 Convolution Operation

Large-scale image processing requires computationally expensive operations. In order to process images more effectively, Neural Networks use convolutions.

Convolution is one of the fundamental building blocks of CNNs. Namely, it is a matrix operation that is able to detect lower, e.g. horizontal edges, as well as higher level features, e.g. eyes, of the input image. It uses a filter, also called kernel, which is a matrix smaller in size than the input image matrix, usually 3x3. Values in the filter are called parameters. Convolution is calculated by sliding the filter over the entire image. Values in the image matrix, also called image tensor, are separately multiplied by the overlapping values in the filter then summed up into a single value.

As shown on the Figure 6 below, the filter is placed on the image and after an elementwise multiplication the values are summed up and form a single value in the output matrix of the convolution. Then, the filter is moved by any number of pixels and the operation is repeated. Stride is the number of pixels the filter is moved by. Every position produces a single value that is stored in the output matrix. The operation has three hyperparameters, filter size (denoted as f), stride (denoted as s) and padding (denoted as p). Filter size refers to the size of the filter. Padding refers to the number of rows and columns filled with zeros added to each side of the original image tensor. Stride and padding will be separately introduced in detail in later sections.

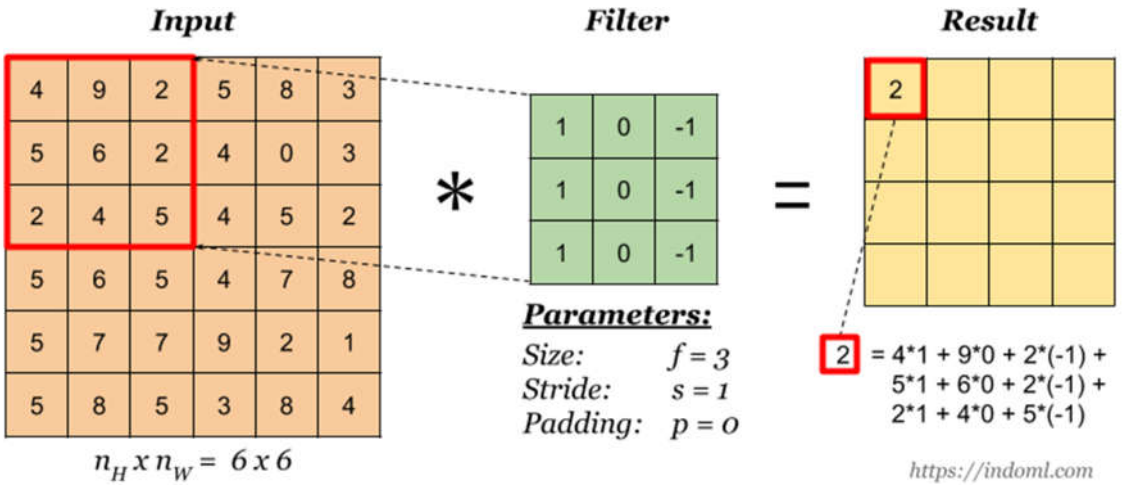

Figure 6. Convolving grayscale image with vertical edge detector [42]

There are different filters with handpicked values for detecting different features on images. For instance, vertical and horizontal edge detector filters are commonly used. Moreover, particular edge detector filters are able to differentiate between light to dark and dark to light edges.

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

Vertical edge detector

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

Horizontal edge detector

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

Figure 7. Commonly used filters with handpicked values

Earlier it was common to use filters with handpicked values. Researchers later discovered that instead of creating filters with handpicked values, it is more efficient to make the neural network learn the numbers in the filters using backpropagation. Consequently, values in filters are treated as parameters that are adjusted by the learning process. By using learning to find useful filters, low level kernels will be able to detect not only vertical and horizontal edges but more complex edges as well. As a result, Neural Networks are capable of learning both low and higher-level features, thus no need to handpick values in filters. Letting the network learn the parameters in filters gives great power to CNNs.

2.1.2 Padding

As seen in the previous section, applying convolution will shrink the size of a tensor. In addition, pixel values in the edges of the image are used much less in the operation than central pixels resulting in throwing away a lot of information. Therefore, before performing the convolution padding is commonly used. Padding means that you add one or more rows and columns to each side of the image usually filled with zeros, called *zero-padding*.

By applying zero-padding the size of the original image can be preserved and pixel values in the edges get more attention than otherwise. As mentioned earlier, padding is a hyperpa-

rameter of the convolution operation which determines the amount of zero-padding you add to each side of the image matrix.

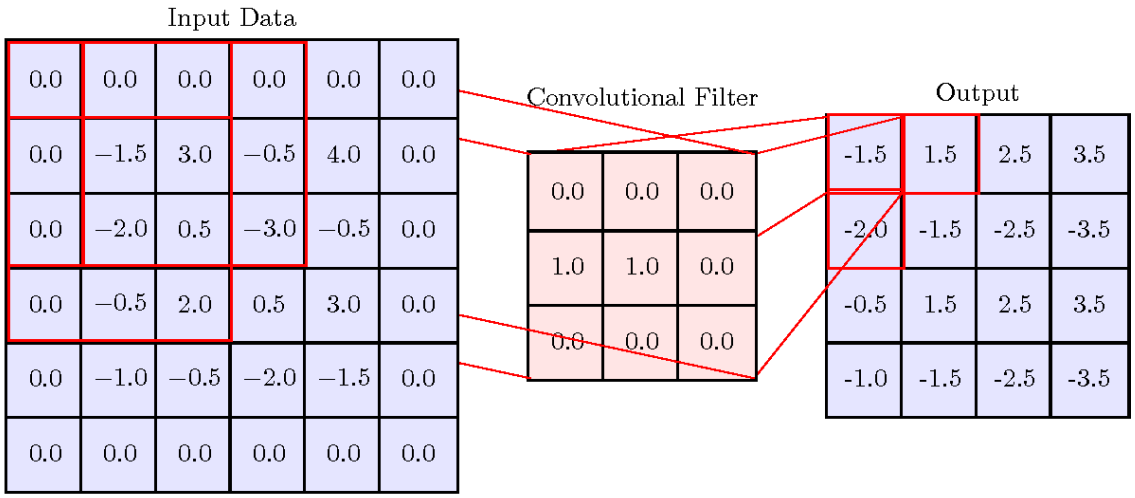

Figure 8. Applying $p = 1$ zero-padding on grayscale image matrix [43]

Example As shown on Figure 8 above, by adding $p = 1$ padding to each side of a 4x4x1 image matrix preserved its size after the convolution operation.

By convention there are two common ways to add padding to images, called *"valid" convolution* and *"same" convolution*. In case of "valid" convolution no padding is added to the image. "Same" convolution adds the necessary amount of padding to preserve the size of the original tensor.

2.1.3 Strided Convolutions

Stride is a hyperparameter that determines the number of pixels the filter is moved by on the image matrix. So far, I have shown examples of convolutions where the amount of stride was equal to 1. However, there are cases where the filter is moved by more than one pixel on an image. Strided convolutions refer to those operations where the amount of stride is more than one. Clearly, the larger the amount of stride the smaller the size of the output matrix. Therefore, strided convolutions are usually used when the size of the input matrix is too large and needs to be reduced.

Example AlexNet [25] by A. Krizhevsky, I. Sutskever, and H. Geoffrey E. used stride equals to 4 on the first convolutional layer along with filter size 11 and no zero-padding.

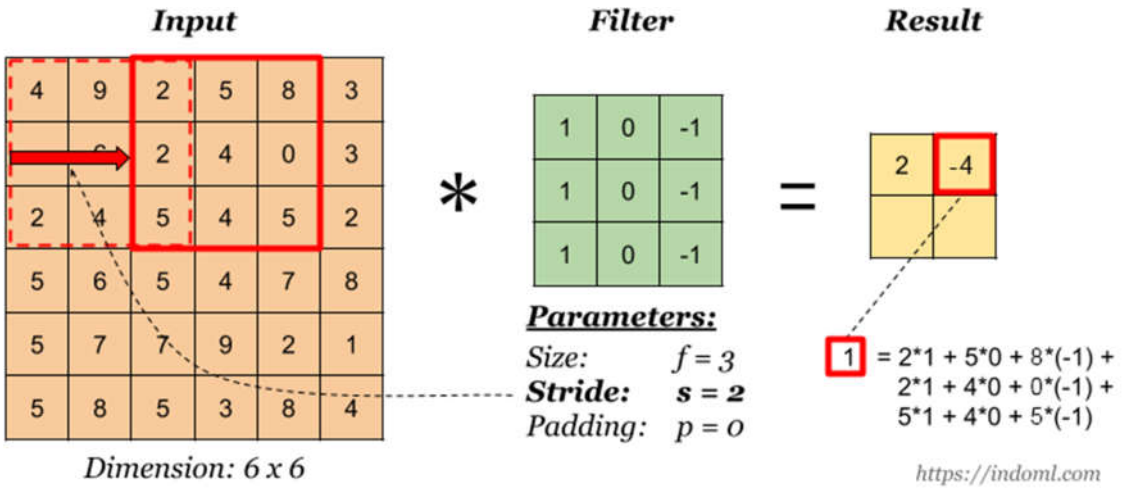

Figure 9. Applying $s = 2$ strided convolution on image matrix [42]

Remark Using the formula below where input size is denoted by n, stride is denoted by s, padding is denoted by p and filter size is denoted by f the size of the output matrix can be easily calculated.

$$\frac{n + 2p - f}{s} \; x \; \frac{n + 2p - f}{s} \tag{1}$$

2.1.4 Convolutions over Volume

While grayscale images can be represented as single 2D matrices, RGB images have more than one channels (red, green and blue) therefore they can be represented as three 2D matrices stacked on each other. It has been shown in Section 2.1.1 how to convolve a grayscale image i.e. a single 2D matrix with a single 2D filter. As expected, an image with a depth of 3 can be convolved with a filter with a depth of 3. In fact, the number of channels in the filter must be equal to the number of channels in the image. Each matrix in the filter stack is individually applied on a matrix in the image stack. After an elementwise multiplication the values are summed up into a single value that forms a single number in the output matrix. As a result, the output of the convolution operation is a single 2D matrix. Therefore, the depth of the output matrix is always equal to the number of filters applied in a convolution.

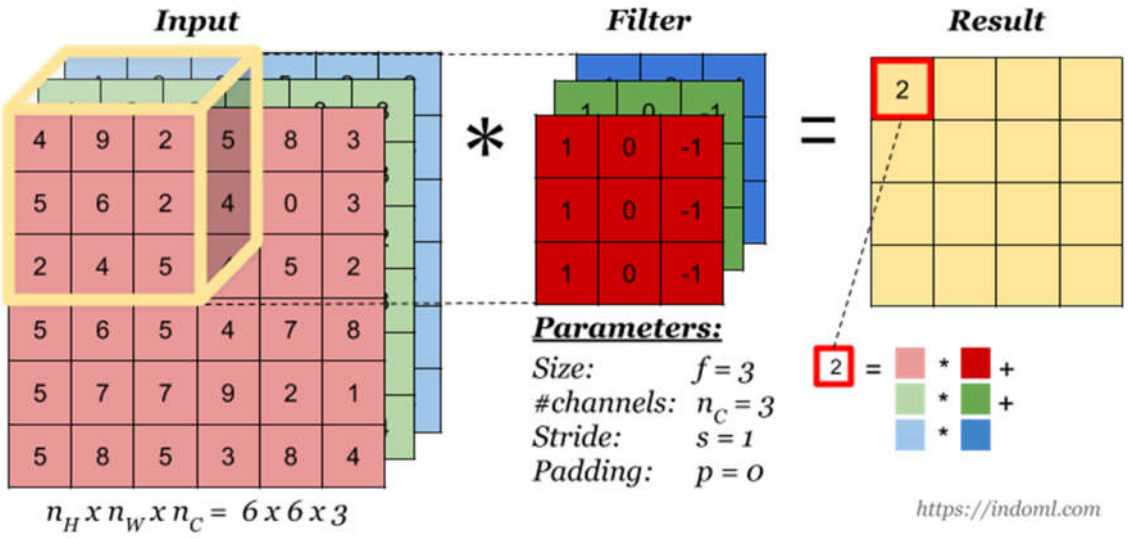

Figure 10. Performing convolution operation on RGB image [42]

Matrices in the filter stack are not necessarily the same. There are many filter options that can detect different features on different channels.

Additionally, multiple filters can be applied on the input image in the same time. For instance, a vertical edge detector and a horizontal edge detector can be applied in parallel. The input is convolved separately with the two filters resulting in two 2D matrices that are then stacked together forming a single output stack. A convolution can use as many filters as necessary so that the depth of the output of the convolution operation will be equal to the number of filters applied.

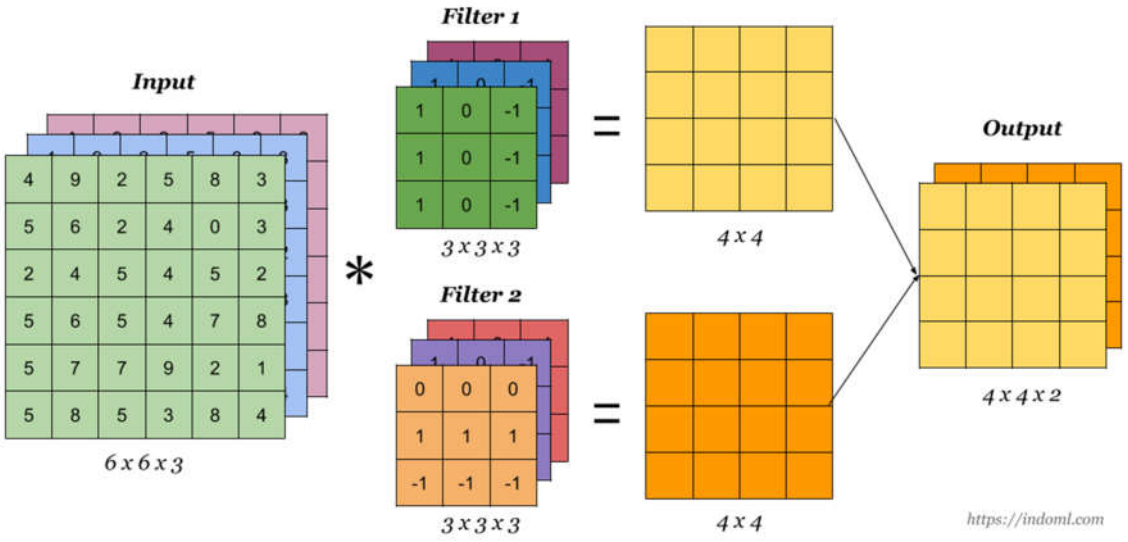

Figure 11. Applying multiple filters on a single RGB image [42]

2.1.5 Convolutional Layer

One of the main advantages of using convolutional layers is that the size of the input image is independent from the number of parameters in the layer.

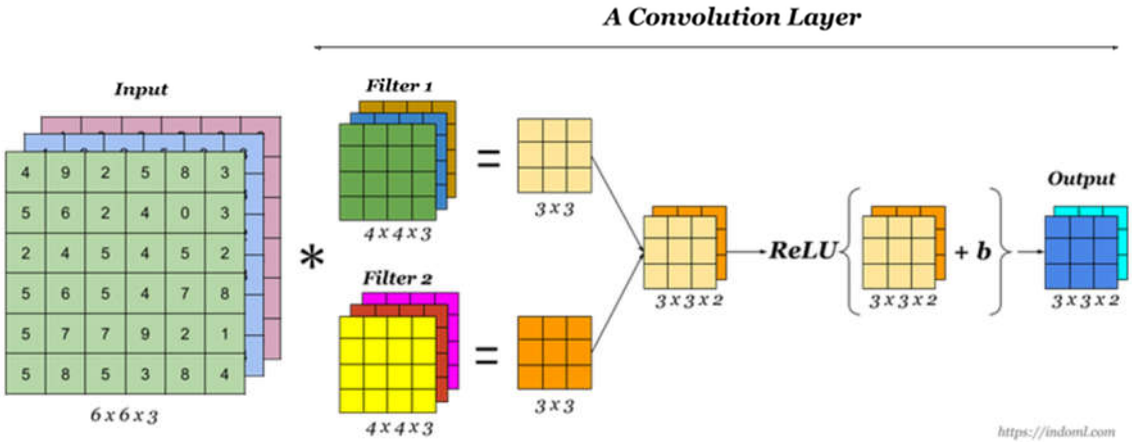

Figure 12. One convolutional layer in a CNN [42]

In classic Neural Network terms, the input image or matrix acts as the output of the previous activation ($a[0]$) and the filters act like weights ($W[1]$). After performing the convolution operation on input matrix $a[0]$ using filters $W[1]$, non-linearity e.g. ReLU is applied on each output matrix. Next, the matrices are stacked together forming the output of the layer, $a[1]$. Clearly, the number of parameters in the layer are fixed by the filters.

Example If you have 10 3x3x3 filters in one layer of a Convolutional Neural Network then the number of parameters in the layer is $3 * 3 * 3 * 10 + 10 = 280$ where +10 is the bias indicating 10 real numbers. For this reason, it can be applied to very large images even though the number of parameters will be still relatively small.

2.1.6 Pooling Layer

In CNNs there are generally three types of layers, convolution (CONV), pooling (POOL) and fully connected (FC) layers. We already saw how convolutional layers work. The next section describes the functionality and architecture of pooling layers.

Aim of a pooling layer is to reduce the size of the representation, speed up computation as well as make the detected features more robust. There are two commonly used pooling operations, max pooling and average pooling.

2.1.6.1 Max Pooling

In max pooling a fixed size window is slided over the input matrix with a given stride. It picks the maximum number from the region the window covers. A large number in the input matrix means that it detected a particular feature e.g. a vertical edge. Therefore, detected features are preserved in the output. Accordingly, the operation has two hyperparameters, filter size and stride. Padding is usually not used in pooling. Max pooling does not have parameters to learn, thus gradient descent does not affect the operation in any means.

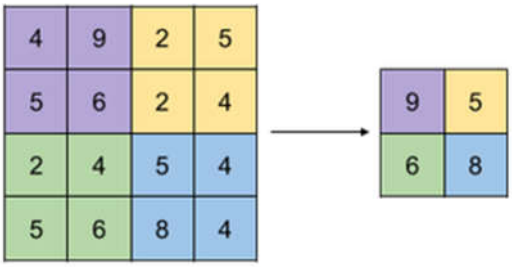

Figure 13. Applying max pooling on a 4x4x1 input tensor [42]

When 3D input is fed into max pooling, it performs the computation on each of the channels independently and outputs a matrix with the same amount of depth as the original input.

2.1.6.2 Average Pooling

Average pooling works in the same way as max pooling except that it calculates the average of the region rather than picking the maximum number. Although, average pooling is less popular than max pooling except some special cases e.g. very deep networks.

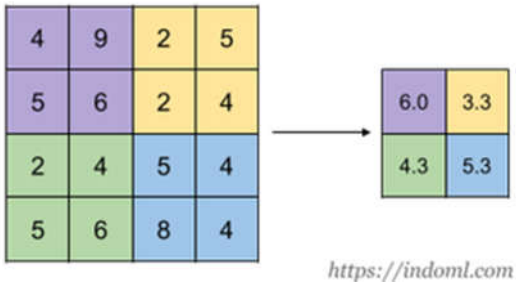

Figure 14. Applying max pooling on a 4x4x1 input tensor [42]

As described, the hyperparameters of pooling are filter size and stride. Generally padding is not used in average pooling operation either. Similarly, to max pooling, the operation does not have parameters to learn therefore not affected by gradient descent.

Remark Formulas for computing the size of the output in convolution also apply for pooling.

2.1.7 An Example CNN

All the necessary building blocks of a CNNs were examined in previous sections. Built on the above let's put together a CNN that is able to recognise handwritten digits on images.

Remark There is inconsistency in the ConvNet literature about what to call a layer. On one hand, a convention is to call CONV and POOL together a layer, on the other hand it is also common to call CONV and POOL separate layers. Generally, layers in Neural Networks has weights or parameters, however pooling layers does not have parameters. Therefore, from now on I am going to treat CONV and POOL together as a single layer.

Let's examine one of the classic networks, LeNet-5 [24] proposed by Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner in 1998. LeNet-5 was designed to recognise handwritten digits from 0 to 9 on grayscale images of size 32x32. It had an architecture CONV → POOL → CONV → POOL → FC → FC → output. This architecture has become quite common in later years. It is a small network with only 60,000 parameters.

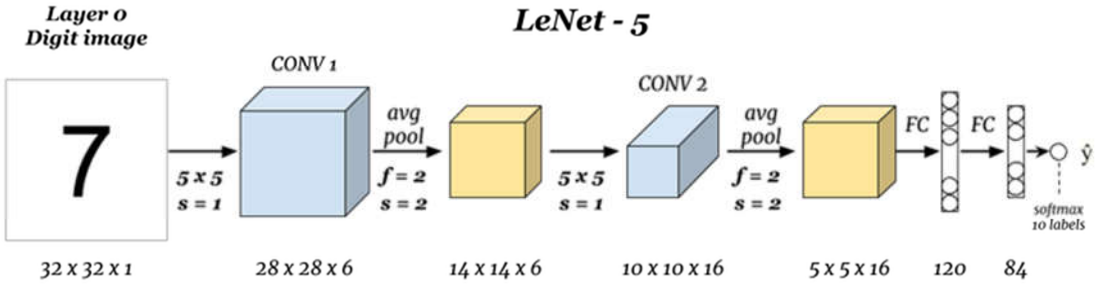

Figure 15. Architecture of LeNet-5 convolutional network [42]

LeNet-5 applies 6 5x5x1 filters on the first convolutional layer and 16 5x5x1 filters on the second convolutional layer. After the first convolution as well as the second convolution it performs average pooling with filters of size 2x2x1 and stride equals to 2. It closes the network with two fully connected layers and finally a softmax layer. Softmax is a generalisation of logistic regression to multiple classification that returns the probabilities of each target class.

In LeNet-5 and in most CNNs the activation size tends to decrease whereas the number of channels tends to increase as we go deeper in the network. Instead of ReLU, LeNet-5 used sigmoid non-linearity after convolution as well as pooling. Nowadays, it is more common to apply ReLU non-linearity after convolution, since it makes training faster and prevents vanishing gradients, and skip applying non-linearity after pooling. Also, LeNet-5 did not use padding at all, as it was not part of the convention in 1998.

Pooling layers do not have parameters to learn and convolutional layers tend to have relatively few number of parameters. Therefore, most of the parameters in a CNN tend to be in the fully connected layers.

Recently, CNNs tends to be much deeper than LeNet-5 and use new conventions like residual blocks [44] or inception layers [45] as we shall see later in Section 2.3.

2.1.8 Advantages of Using Convolutions

Convolutional layers have two main advantages over fully connected layers, namely parameter sharing and sparsity of connections.

Example Let's compare a convolutional layer and a fully connected layer given the same RGB input image of size 32x32x3. Consequently, the number of parameters in the image is $32 * 32 * 3 = 3072$. Let's assume that we apply 6 filters of size 5. Therefore, the number of parameters in the convolutional layer is $5 * 5 * 3 * 6 + 6 = 456$ where +6 is the bias. Clearly the number of parameters is quite small. As a result, the size of the output would be 28x28x6 containing $28 * 28 * 6 = 4704$ parameters. Alternatively, a fully connected layer that produces the same output would have $3072 * 4704 \approx 14M$ parameters. Considering the small size of the input image, these are a lot of parameters to train. Moreover, if the image would have a size 1000x1000x3 the weight matrix would become infeasibly large.

Both parameter sharing, and sparsity of connections describe how convolutions can result in fewer parameters thus make convolutional layers a more feasible solution in image processing compared to fully connected layers.

2.1.8.1 Parameter Sharing

Parameter sharing refers to the phenomena that in convolutions a filter such as a horizontal edge detector that can be used in one part of an image is probably useful in another part of the image. In other words, a feature detector can be used in lots of different positions in the image. Therefore, reusing a filter many times lead to fewer parameters.

2.1.8.2 Sparsity of Connections

Each output unit is connected to only a small number of inputs in each layer. It results in fewer parameters to be learned compared to FC layers since in FC layers each output is connected to each input value.

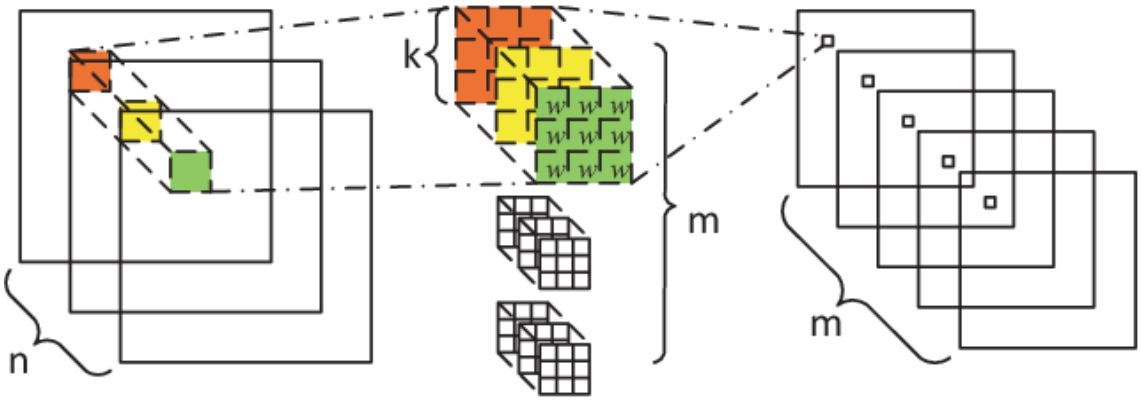

Figure 16. Visualisation of sparsity of connections [46]

Due to parameter sharing and sparsity of connections CNNs have a lot fewer parameters than fully connected networks that allows them to be trained with smaller training sets and are less prone to be overfitting. Furthermore, the convolutional architecture is very good at capturing translation invariance which is when an image is distorted with a couple of pixels. As a result, CNNs are able to decode that an image is shifted and should be assigned to the same output label.

Given the above, these characteristics justify the choice of using CNN as an approach to solve the object detection problem on the EPIC-KITCHENS dataset.

2.2 Convolutional Object Detection

As described in Section 1.1.1, object detection is a Computer Vision problem where the task is to classify and localise multiple objects on an image where objects can be from the same class or even from different classes. Classification networks like LeNet-5 [24], introduced in Section 2.1.7 can be transformed into networks that are capable of outputting classification along with localisation. In case of classification with localisation we assume that the image has a single object. In order to use classification networks for localisation we need to add more output units to the softmax target label. In particular, the target label needs to be extended with five parameters of which four parameters define a rectangle around the detected object and the remaining parameter is the probability of an object being detected in the rectangle. Therefore,

the parameters of the output label consist of the object classes, the bounding box coordinates and a confidence score.

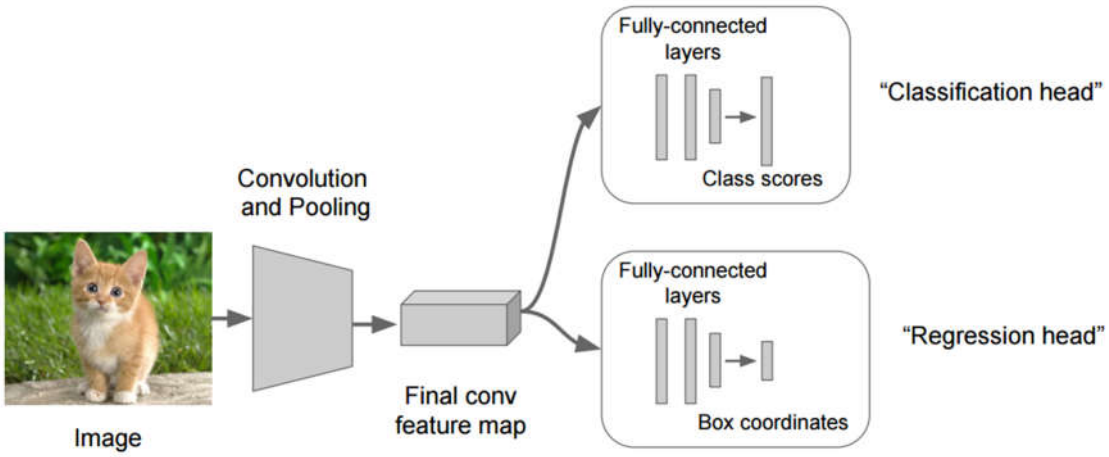

Figure 17. CNN used for classification with localisation [47]

At this point the algorithm is able to localise a single object on an image. However, it has not been explained how the algorithm computes the location of the object of interest. In order to find the location of a single object as well as multiple objects, sliding windows algorithm can be used.

2.2.1 Sliding Windows

Sliding windows is an early approach to object detection where a fixed size rectangular shape, called a window is slided over an input image. Each position of the window is fed into a convolutional classification network that outputs a prediction. If the window contains the object of interest, then the object is localised on the image by the coordinates of the window. While using sliding windows, it is sensible to use small stride in order to cover as many positions as possible. The process can be repeated with windows of different size until the window and location are found that localise the object of interest.

Sliding windows has a high computational cost as it takes large number of small crops from an image and independently runs them through a convolutional classification network. Using bigger stride would reduce the computational cost since there would be smaller number of crops however would hurt performance since less positions would be examined. Sliding windows worked well in the pre-Deep Learning era combined with hand-engineered features, however later became infeasibly slow for large images.

2.2.1.1 Convolutional Implementation

Sliding windows can be implemented convolutionally in a more efficient way. [48] showed that running the entire image through a Fully Convolutional Network (FCN) produces the same output as running the crops independently. The implementation requires the fully connected layers of the CNN (shown in Figure 18) to be replaced with convolutional layers. Note that mathematically none of the operations changes.

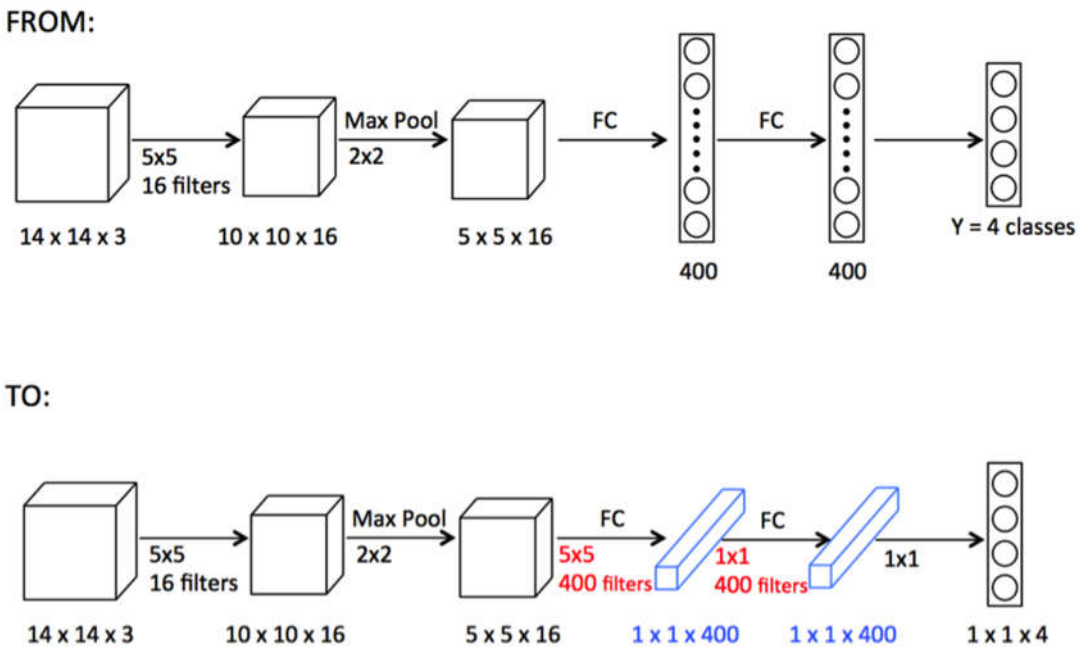

Figure 18. Convolutional implementation of sliding windows [49]

Example Let's assume that we have an image of 16x16x3, a window of size 14x14x3 and 4 class labels. We slide the window over the image with stride equals to 2 which produces 4 crops that are separately fed into a FCN. As a result, the FCN outputs a 1x1x4 tensor for each crop where each parameter corresponds to an object class. Alternatively, we could feed the entire image into the network and get a 2x2x4 output tensor where each 1x1x4 block is identical to the output that we would have got if we would have run the crops separately through the network. It turns out that the result is equivalent to feeding four 14x14x3 windows through the network.

This way many duplicate operations can be overcome resulting in a less computationally intensive process.

2.2.2 Bounding Box Predictions

Bounding boxes are rectangles drawn around objects of interest defined by four parameters b_x, b_y, b_h, b_w where b_x and b_y defines the midpoint of the box while b_h and b_w the height and width consequently.

Remark Bounding boxes can be encoded using different parameters, for instance some algorithms, e.g. RetinaNet [38] defines bounding boxes with the top left and bottom right coordinates.

Convolutional implementation of sliding windows is computationally more efficient but does not produce accurate bounding boxes around objects. In order to get more accurate predictions, a grid is placed on the image that divides the image into equal size grid squares. Then, the image is fed into the classification with localisation network that produces predictions for each of the grid cells.

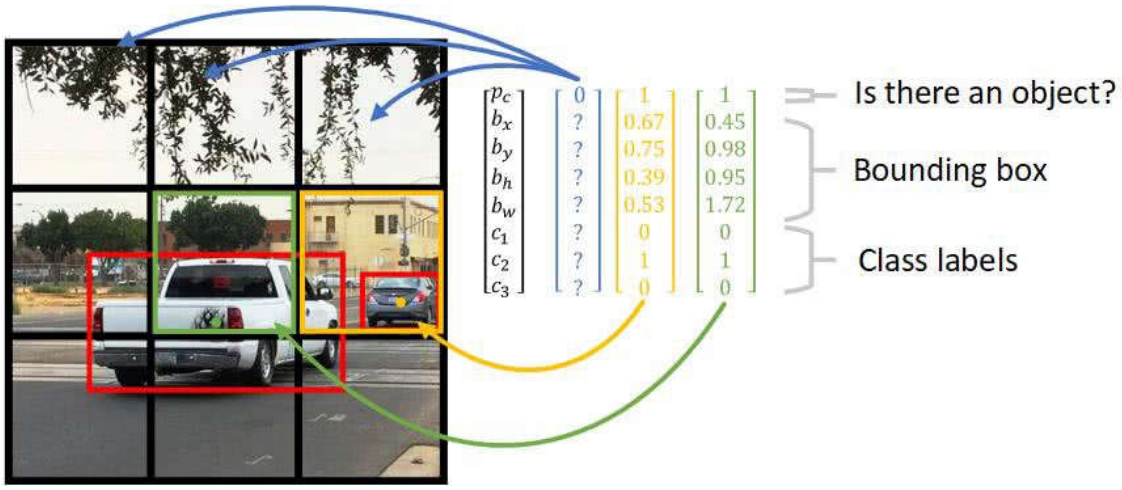

Figure 19. Applying a grid on the input image [50]

Example Let's assume that we have a 100x100x3 image where we would like to detect cars, pedestrians and bicycles. We place a 3x3x3 grid on the image then feed the image into a FCN that produces a 3x3x8 output where each 1x1x8 block corresponds to a grid square. Each 1x1x8 block contains a confidence score, four bounding box coordinates and three class labels.

Each bounding box is assigned to a single grid cell that contains its midpoint. This approach is very similar to the convolutional implementation of sliding windows however able to output bounding boxes of any size. It encodes bounding box coordinates as a fraction of the grid square therefore the midpoint coordinates must be between 0 and 1 although the width and height can be greater than 1 if the bounding box is bigger than the grid square containing its midpoint.

If there is only a single object in each grid square, then this approach should work well since the algorithm can only predict a single bounding box per grid cell. Having a finer grid can lower the chance of having multiple objects in the same grid square.

As seen, it is a convolutional implementation as the image is fed into the FCN as a whole rather than feeding each grid square separately, therefore it is highly efficient and fast due to shared computations. This approach is usually used in one stage networks [38][51][52] introduced in Section 2.3.

2.2.3 Intersection over Union

Intersection over Union (IoU) is used for evaluating object detection algorithms. It computes the intersection over union of two bounding boxes usually the ground-truth box and the predicted bounding box. Union is the area contained in either B_1 or B_2 while intersection is the area contained in both B_1 and B_2. In general, IoU measures the overlap between the two boxes.

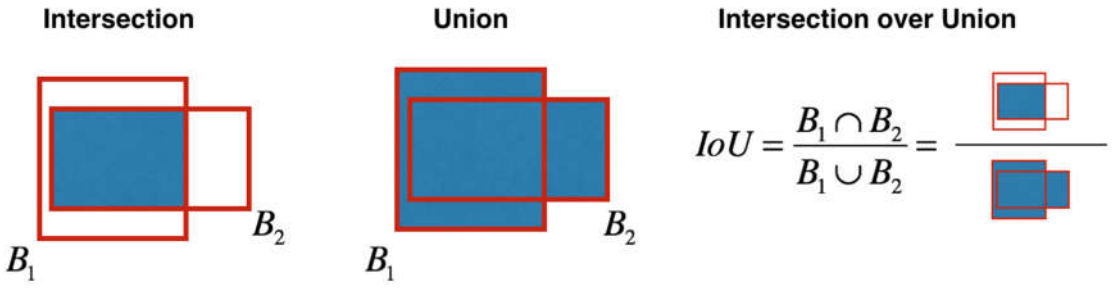

Figure 20. Visualisation of Intersection over Union (IoU) [53]

By convention a localisation is correct if IoU > 0.5 and incorrect if IoU ≤ 0.5. This threshold does not have theoretical base, applications where precise localisation is important may use IoU > 0.6 or 0.7.

2.2.4 Non-maximum Suppression

Rather than detecting an object once, algorithms tend to detect the same object multiple times. Non-max suppression [54] discards weak detections and proposes a single final detection. It looks at the probabilities associated with each detection, picks the one with the highest confidence and suppresses the others.

More precisely, first it discards all bounding boxes with lower confidence than 0.6 or other chosen threshold. Second, while there are any remaining boxes, it repeatedly picks the box with the highest probability and outputs that as prediction. It discards any remaining boxes with IoU ≥ 0.5 with the box output in the previous prediction.

If more than one object class need to be detected, then non-max suppression must be carried out independently on each of the output classes.

2.2.5 Anchor Boxes

So far each of the grid cells can detect only a single object. Anchor boxes are fixed size rectangles of different size that help detect multiple objects in the same grid cell. A certain number of anchor boxes are associated with each grid cell. Rather than making prediction for a single bounding box, predictions are made for each anchor box.

Example Let's assume that each grid square is associated with two anchor boxes of different size. As a result, a target label y corresponding to a particular grid cell changes as shown in Figure 21. Thus, target labels for each grid cell contain predictions associated with each anchor box.

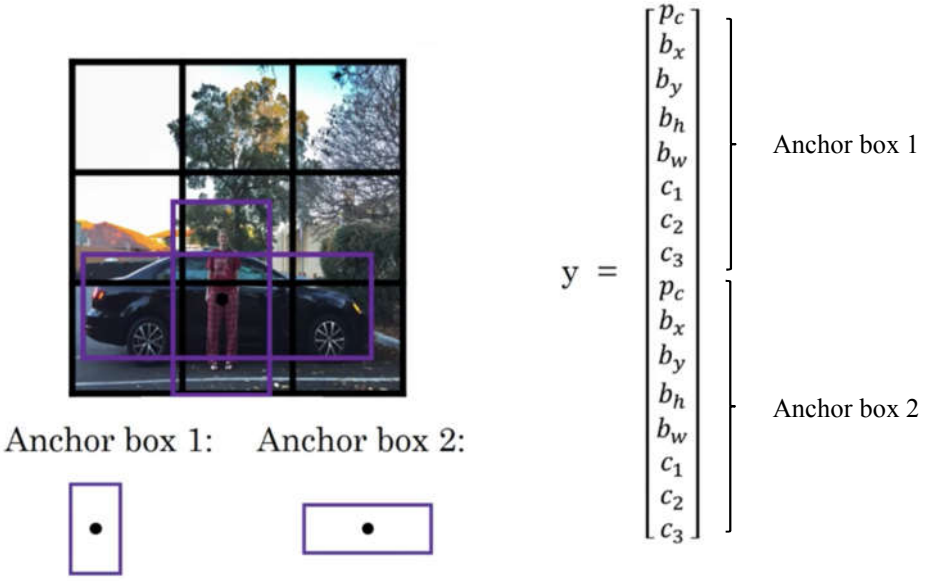

Figure 21. Detecting multiple objects in the same grid cell using anchor boxes [2]

Having anchor boxes of different size allows the algorithm to specialise better to detect objects of different shape. For instance, Anchor box 1 from Figure 21 is more suitable for detecting pedestrians than Anchor box 2. Therefore, some of the outputs can specialise in detecting flat and wide objects while others in tall and narrow objects.

Anchor boxes are usually handpicked imitating the shapes of objects the algorithm tries to detect. Alternatively, one could use k-means algorithm [55] for grouping together the types of bounding box shapes from the dataset and use that to select anchor boxes.

Disadvantages of using anchor boxes are that the algorithm cannot handle if there are more objects in a grid square than assigned anchor boxes, neither if two objects in the same grid square have similar shape and would be assigned to the same anchor box. However, these do not happen often in practice as having a fine grid, e.g. 19x19 would prevent having multiple objects in the same grid cell.

2.2.6 Evaluating Object Detectors

In order to be able to compare object detectors, we need a single metric that combines how well the models classify and localise objects. Mean average precision (mAP) is a commonly used metric that is calculated from precision, recall and Intersection over Union (IoU).

Intersection over Union (IoU) measures the overlap between the predicted bounding box and the ground-truth bounding box. In particular, it computes the area of intersection over the area of union of the two bounding boxes. A pre-set IoU threshold determines if a bounding box is accepted as final prediction or thrown away. Consequently, IoU determines the number of true positives and false positives when calculating precision and recall. By convention, if IoU > 0.5 then an object detection is considered to be true positive, while IoU ≤ 0.5 then false positive.

Both precision and recall are calculated for each object class. Recall shows when there is an object how often does the model predicts the object, hence $Recall = TP/(TP + FN)$. While precision shows when the model predicts an object how often is it correct, hence $Precision = TP/(TP + FP)$. Average precision (AP) is the average value of precision across a set of equally spaced recall values $\{0.0, 0.1, 0.2, \dots, 1.0\}$.

Accordingly, mean average precision (mAP) is the average of AP values across all class labels. For instance, PASCAL VOC 2007 challenge [34] used IoU threshold > 0.5, thus AP was averaged over all 20 object classes.

Remark Recently, papers tend to use the metric of MS COCO 2017 challenge [35] where AP is averaged over all 80 object categories and 10 IoU thresholds, namely IoU > $\{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$. Generally, this metric favours models that are better at precise localisation.

2.3 State-of-The-Art Object Detection Networks

There are several meta strategies to solve the object detection problem in Computer Vision. Two main approaches are the region-based approach and the single shot approach. Region-based methods select regions that are likely to contain objects from an image using different algorithms, e.g. Selective Search [56], Region Proposal Network (RPN) [57]. In contrast, single shot methods treat object detection as a regression problem and rather than performing independent processing on each potential region, they make all these predictions, i.e. bounding box coordinates predictions, class label predictions, all at once with a very deep Convolutional Neural Network.

Remark There is also a hybrid method [58] that combines ideas from both region-based and single shot approaches. However, we will not discuss hybrid methods in this dissertation due to the time constraints that apply to the project.

In this section four state-of-the-art object detection networks will be introduced and compared in order to explore and justify the network of choice for the EPIC-KITCHENS dataset object detection challenge. Baseline solution uses a region-based method, namely Faster R-CNN [57]. Therefore, further goal is to get an understanding of the differences between two-stage (region-based) and one-stage (single shot) methods in order to be able to compare the baseline solution and the chosen network.

2.3.1 Faster R-CNN

Faster R-CNN (Faster Region based Convolutional Neural Network) [57] is a region-based approach that generates Regions of Interest (RoI) to find areas on an image that are likely to contain objects. It is a third generational network that it is an upgraded version of R-CNN [59] and Fast R-CNN [60]. In order to see how Faster R-CNN came over the issues of R-CNN and Fast R-CNN, I introduce them below.

Remark Region proposals are built on the idea of sliding windows [22], but rather than running each possible position and scale through a ConvNet, R-CNNs only use those windows that are likely to contain objects of interest.

2.3.1.1 R-CNN

Given the input image, R-CNN (Region based Convolutional Neural Network) [59] generates Regions of Interest (RoI) using an external proposal method e.g. Selective Search [56], that is able to generate approximately 2,000 proposals of an image. Next, the generated region proposals are reshaped in order to fit the fixed size that the subsequent network expects. Then, each warped region proposals are passed through the ConvNet individually. Finally, a Support

Vector Machine (SVM) predicts label categories for each of the region proposals and a Linear Regressor predicts corrections for the bounding boxes (compared to the region proposal area).

R-CNN is a computationally expensive method as it runs separately on each region proposal, therefore its training time is slow and takes a lot of disk space. Inference time is also slow, it takes 47s per image to produce an inference [61]. Further disadvantage is that the parameters of the region proposal algorithm are fixed and not learned.

The main handicap of R-CNN is time, therefore later research was primarily focused on how to speed up the algorithm.

2.3.1.2 Fast R-CNN

Fast R-CNN (Fast Region based Convolutional Neural Network) [60] rather than processing each RoI separately, it runs the image through a ConvNet all at once in order to get a high resolution convolutional feature map corresponding to the entire image. Next, similarly to R-CNN it generates region proposals using an external method, e.g. Selective Search [56], but instead of cropping out regions from the image, it projects the region proposals onto the feature map and crops out regions from the feature map. Next, the crops from the feature map are reshaped by a RoI pooling layer in order to fit the fixed size that the subsequent fully connected layers expect. Finally, the reshaped crops are passed through the fully connected layers and class label predictions are made by a softmax classifier and bounding box predictions are given by linear regressors.

As a result of the changes, Faster R-CNN is 10 times faster to train compared to R-CNN [10]. It has a significantly shorter test time, it is actually so fast that the run times ends up being bottlenecked by producing the region proposals.

2.3.1.3 Faster R-CNN

Faster R-CNN (Faster Region based Convolutional Neural Network) [57] addresses the bottleneck problem of Fast R-CNN. Similarly, to Fast R-CNN, the image is run through a ConvNet to get the high-resolution feature map corresponding to the image. Next, instead of using an external proposal method, it uses a separate Region Proposal Network (RPN) that works on top of the convolutional features and predicts its own region proposals. After having the region proposals, it looks the same as Fast R-CNN, so the region proposals are projected onto the feature map. Crops from the feature map are reshaped by a RoI pooling layer and fed into a fully connected network in order to get the label categories and bounding box predictions.

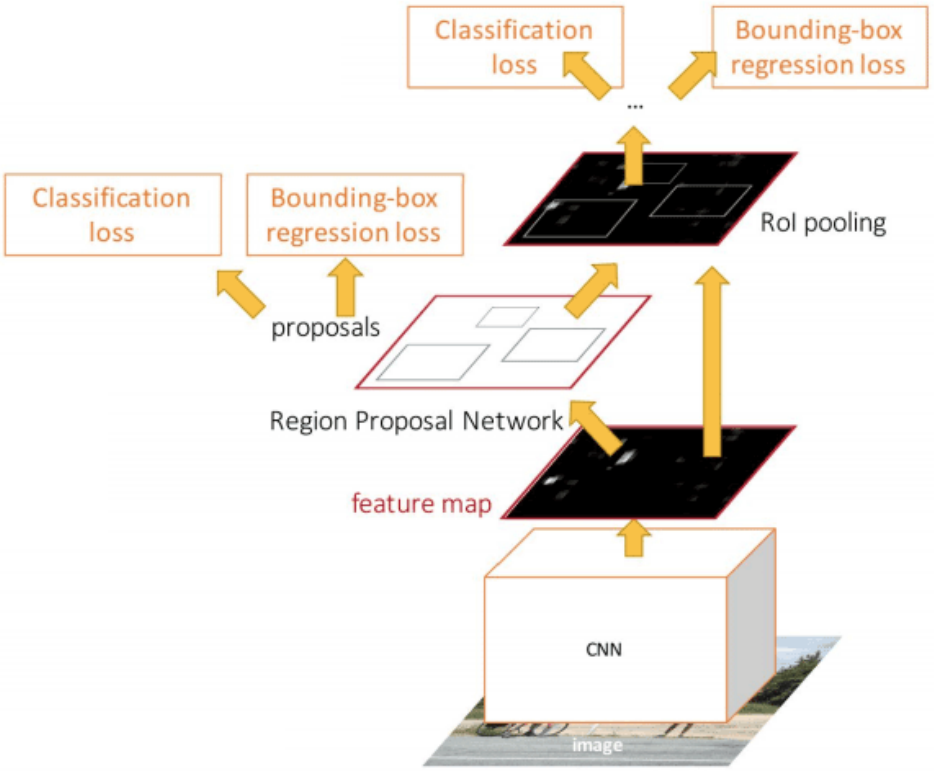

Figure 22. Faster R-CNN architecture [10]

Faster R-CNN is very fast [10] compared to its predecessors by using an internal network rather than an external fixed proposal method. Further advantage that it learns the region proposals as well, so a mismatch between the data and the region proposals cannot exist anymore.

Faster R-CNN tends to give higher accuracies but ends up being slower [10] than single shot methods since single shot methods do not require region processing.

2.3.2 YOLOv3

YOLOv3 (You Only Look Once v3) [62] is a single shot method that instead of running independently on each potential region of interest, requires only one forward propagation to make predictions.

Given an input image, YOLOv3 places a grid, e.g. 19x19, onto the image where each of the grid cells are responsible for predicting given number of bounding boxes. If a midpoint of an object falls into a grid cell, then that grid cell is responsible for detecting that object. For each bounding box in each grid cell YOLO outputs coordinates and a confidence score for the bounding box, and predictions for each class category.

Then, the confidence score for the bounding box and the class prediction are combined into a final score that tells the probability that a particular bounding box contains a specific type of object. Finally, low probability predictions are thrown away and non-max suppression [63] generates the final predictions.

By processing the grid cells all at once YOLOv3 takes advantage of shared computation that makes it a very efficient algorithm, even applicable for real time object detection.

Similarly, to Faster R-CNN [57], YOLOv3 is also a third generational method that addresses the issues of YOLO [52] and YOLOv2 [55]. Therefore, I introduce YOLO, YOLOv2 and YOLOv3 below.

2.3.2.1 YOLO

YOLO [52] was published in 2015 by J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. Shortly after YOLO got popular, since it is able to process streaming video real time with less than 25 milliseconds of latency. YOLO presented some major improvements compared to region-based networks, e.g. it makes less than half background errors compared to Faster R-CNN due to processing the image all at once and examining it globally. According to the authors, YOLO has a very good generalisation ability making it invariant when applied to new domains. It consists of 24 convolutional layers followed by 2 fully connected layers. YOLO optimises for the sum squared error which has some limitations, e.g. it does not maximise average precision (AP) and it handles equally errors from large and small boxes. Further limitations are that each grid cell can only have one class label and predict two bounding boxes. In addition, it struggles with detecting small objects in groups. Ultimately, the primary source of error is incorrect localisation.

2.3.2.2 YOLOv2

YOLOv2 [55] is focused on improving localisation error, since that was the main source of error in YOLO. Consequently, batch normalisation was added to each convolutional layer and dropout was removed from the model that resulted in 2% improvement in average precision (AP). Fully connected layers were removed from the end of the network. YOLOv2 uses anchor boxes to predict bounding boxes and instead of predicting bounding box coordinates, it only predicts the offsets. For detection prediction it uses a 13x13 feature map, larger than the 7x7 that YOLO used. In addition, a pass-through layer was also added to the network that combines lower and higher-level features.

2.3.2.3 YOLOv3

YOLOv3 [62] is able to process images at 30 FPS (frames per second) on a NVIDIA Pascal Titan X GPU and perform with mAP of 57.9% on the MS COCO dataset [35]. Instead of using mean square error, it uses cross-entropy loss for calculating the classification loss for each label. Rather than softmax, it uses independent logistic classifiers. Along with the bounding box predictions, it also predicts an objectness score for each bounding box that shows which anchor box overlaps the most the ground-truth object. Instead of predicting boxes at a single scale, YOLOv3 predicts across 3 different scales using the idea of Feature Pyramid Networks [64] similarly to RetinaNet [38]. Darknet-19 base network was replaced by Darknet-53 which is a 53-layer-long network with skip connections similar to ResNets (Residual Networks) [44] used for feature extraction.

As we shall see in Chapter 2.3.5, YOLOv3 overperforms other networks in speed however it is still behind RetinaNet [38] in terms of mean average precision (mAP) despite the improvements made to the original YOLO algorithm.

2.3.3 SSD

Similarly to YOLOv2 [55], SSD (Single Shot MultiBox Detector) [51] is a large fully convolutional network that uses different default bounding boxes (anchor boxes) and matches them with objects of interest. However rather than doing it once like YOLO [52], SSD performs it six times on six different feature maps of different scale using default bounding boxes of different aspect ratios.

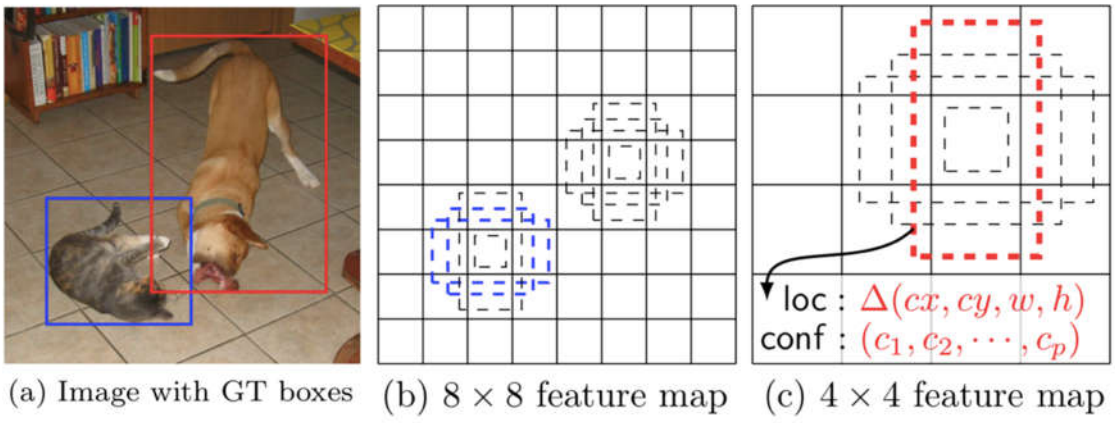

Figure 23. Using multiple feature maps of different scale [51]

SSD using a VGG-16 base network which is responsible for high quality image classification in the early layers of the network. Although, in theory SSD could be used with any deep learning base network, e.g. ResNet-101 [44]. Following convolutional layers are responsible for producing several feature maps of different resolutions. Each feature map specialises in detecting objects of a certain scale, i.e. finer feature maps are focused on detecting smaller objects. These layers are used for predicting the bounding box offsets.

SSD has two versions SSD300 and SSD512, taking input images of size 300x300 and 512x512 accordingly.

Training an SSD network is quite tricky as it involves choosing a set of default boxes and scales for the feature maps as well as the hard-negative mining and data augmentation strategies. Most default bounding boxes will not match any objects, therefore introduce an imbalance in the training examples. In order to reach a 3:1 balance between negative and positive examples, SSD sorts out most negative examples using hard negative mining.

As we shall see in the comparison, SSD is able to achieve higher accuracy then YOLOv2, however falls short of YOLOv3 in speed and accuracy.

Remark Note that there are newer variants of SSD [65][66][67] that we do not examine in this dissertation due to time constraints.

2.3.4 RetinaNet

RetinaNet [38] is a one-stage unified fully convolutional neural network consisting of a ResNet backbone combined with a Feature Pyramid Network (FPN) [64], a classification subnet and a box regression subnet. Accordingly, the backbone is responsible for computing the convolutional feature maps, the classification subnet carries out classification on the output of the backbone and finally the box regression subnet predicts bounding box coordinates.

RetinaNet addresses two problems from previous networks, (1) in Faster R-CNN the feature map is created after many convolutional layers resulting in a loss of low level information that makes the network unable to detect objects of small size; (2) the more anchor boxes a method use to find objects, the more imbalanced the training examples will be as most of the boxes will not match objects, causing the classifier to be biased and emphasize the background in order to minimize the loss. RetinaNet solves these problems by using a FPN on top of the ResNet backbone and adding a factor to the standard cross entropy loss function.

FPN is somewhat similar to the SSD architecture where different convolutional layers produce feature maps of different scale. However, SSD only produces feature maps in the upper layers while FPN creates feature maps both in lower and upper layers. Therefore, it combines low and high-level features using a top-down pathway and lateral connections. As a result, RetinaNet has an increased performance in detecting small objects as well as detecting objects in general.

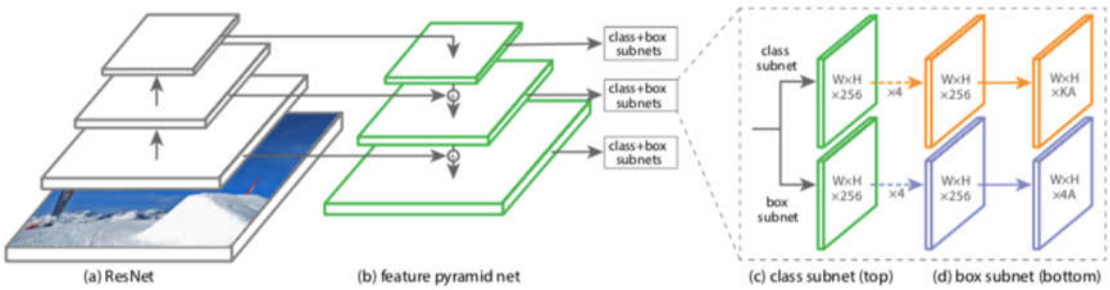

Figure 24. Architecture of RetinaNet [38]

RetinaNet proposes a new loss function called Focal loss which is a modified version of the cross-entropy loss function. As described, one shot methods have a high localisation error in general due to the overwhelming amount of anchor boxes that do not match objects. SSD tackles this problem by sorting out negative examples with hard negative mining [51]. RetinaNet takes a different route and down-weights the loss associated with correctly classified examples by changing the cost function. Therefore, Focal loss assigns minimal weight to correctly classified examples but more weight to incorrectly classified and hard examples. As a result, RetinaNet has an increased accuracy.

RetinaNet combines many ideas from the Deep Learning literature in one unified network that is able to achieve higher accuracy than two-stage methods and is a strong competitor to other one-shot methods in terms of speed.

2.3.5 Comparison of Object Detection Networks

There is a trade-off between speed and accuracy in object detection networks where two-stage methods prefer accuracy over speed, while one-stage methods prefer speed over accuracy. However, both RetinaNet and YOLOv3 seems to close the accuracy gap between one shot and region-based methods.

The EPIC-KITCHENS baseline challenge uses mAP metric from PASCAL VOC 2007 with 3 IoU thresholds, 0.05, 0.5 and 0.75. Therefore, I am going to compare Faster R-CNN, SSD, YOLOv3 and RetinaNet using mAP from PASCAL VOC 2007 with 2 IoU threshold, 0.5 and 0.75.

Table 2 below is taken from [62][38] showing mean average precision (mAP) for 2 IoU thresholds, 0.5 and 0.75 as well as for small, medium and large objects on the challenging COCO dataset [35]. Among others, COCO has several thousands of images picturing kitchen environments with objects like oven, refrigerator, microwave, bottle, bowl, etc., similar to the EPIC-KITCHENS dataset.

Object detector	Backbone	mAP					Time
		IoU > 0.5	IoU > 0.75	Small objects	Medium objects	Large objects	
Two-stage methods							
Faster R-CNN [64]	ResNet-101-FPN	59.1	39.0	18.2	39.0	48.2	-
Faster R-CNN [68]	Inception-ResNet-v2-TDM	57.7	39.2	16.2	39.8	**52.1**	-
One-stage methods							
SSD513 [51][65]	ResNet-101-SSD	50.4	33.3	10.2	34.5	49.8	125
RetinaNet-800* [38]	ResNeXt-101-FPN	**61.1**	**44.1**	**24.1**	**44.2**	51.2	-
RetinaNet-800 [38]	ResNet-101-FPN	57.5	40.8	20.2	41.1	49.2	198
RetinaNet-500 [38]	ResNet-101-FPN	53.1	36.8	14.7	38.5	49.1	90
YOLOv3-608 [62]	Darknet-53	57.9	34.4	18.3	35.4	41.9	**51**

* Trained with scale jitter and for 1.5× longer than the other RetinaNet models.

Table 2. Comparing state-of-the-art object detection networks [62][38]

As seen, RetinaNet brings the best results in most metrics except mAP for large objects where Faster R-CNN is the best, and time where YOLOv3 is the first, as expected. Both YOLOv3 and RetinaNet variants are above SSD. RetinaNet-800 with Res-NeXt-101 backbone is able to outperform two-stage methods in accuracy, however it needs 3.8x more time to process an image than YOLOv3. YOLOv3 achieves 57.9 mAP at IoU > 0.5 in 51ms while RetinaNet-800 with ResNet-101-FPN backbone achieves 57.5 mAP in 198ms. This suggests that YOLOv3 is able to show similar performance 3.8x faster. However, the performance of YOLOv3 drops when IoU threshold is increased to 0.75 showing that YOLOv3 is failing to get the boxes perfectly aligned.

A figure taken from [62] of the speed/accuracy trade-off shows that YOLOv3 is able to balance between speed and accuracy better than other networks.

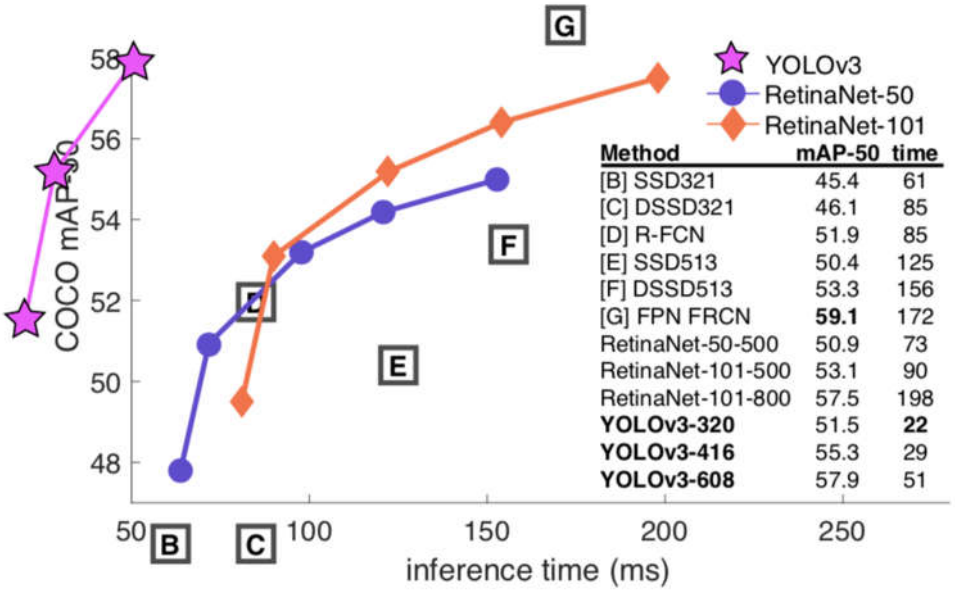

Figure 25. Mapping mAP (IoU > 0.5) to inference time (ms) on MS COCO dataset [62]

To summarise, RetinaNet is the most accurate object detector presented with comparable inference time to SSD. YOLOv3 is the fastest method that combines accuracy and speed efficiently, however falls short from RetinaNet in accuracy. Baseline challenge in the EPIC-KITCHENS dataset is focused on accuracy and does not consider inference time. Therefore, it justifies the choice to pick the most accurate method available, RetinaNet to solve the object detection challenge.

2.4 The EPIC-KITCHENS Dataset

The EPIC-KITCHENS dataset is a large-scale egocentric video dataset capturing various non-scripted activities in native kitchen environments which has been recently published (April 2018) by D. Damen *et al.* [1]. It features of 55 hours of video (11.5 million frames) that is labelled for 39.6 thousand action segments and 424.2 thousand object bounding boxes. After recording the videos, the 32 participants narrated their own material thus providing annotations in form of speech that were later processed and turned into action segment annotations with start/end times and active object bounding box annotations around objects subject to interaction. Primary aim of the research was to capture the natural multi-tasking and parallel-goal interactions that happen while participants perform different tasks. For this reason, the dataset is unique in both content and size.

2.4.1 Data Collection

While collecting data for the EPIC-KITCHENS dataset, 32 participants from 4 different cities recorded every kitchen activity using head-mounted GoPro cameras every day for three consecutive days. Participants were asked not to interact with other people while recording as human-to-human interaction falls out of the focus of the research.

Most recorded sequences have linear field of view, 59.94fps and Full HD resolution of 1920x1080, however few of them have different parameters as some participants used different settings. Each participant recorded 13.6 sequences on average with an average of 1.7h recording length however the latter widely varies.

Each participant narrated his/her own videos afterwards. Later, transcripts were created using Amazon Mechanical Turk (AMT) human workforce. In addition, YouTube's automatic

closed caption alignment tool was used to produce accurate timings. Disadvantages of using narrations are incompleteness, latency and the use of free language. As a solution, researchers only evaluated actions that have been narrated, adjusted the belated narrations using ground-truth action segments and grouped verbs and nouns into minimally overlapping clusters.

2.4.2 Annotation

Beyond creating transcripts, researchers used Amazon Mechanical Turk (AMT) to adjust start/end times for each action segment annotation. They introduced two constraints in order to reduce the amount of noisy annotations: (1) each action has to be at least 0.5 seconds; (2) action cannot start before the preceding action's start time. Using a combination of manual and automatic processes 39,564 ground-truth action segments were produced.

In addition, production of object bounding boxes was also crowdsourced via AMT. As a result, 454,158 object bounding boxes were created of objects relevant to actions.

Later, Part-of-speech (POS) tagging was used to find verbs and nouns in annotations. Namely, they applied spaCy's English core web model to produce the POS tags. Since several times multiple objects appeared in one sentence, the tagging process was not a straightforward task. Moreover, many nouns were absent or replaced with pronouns (e.g. 'dog' → 'it'). Next, researchers produced classes of verbs and nouns using manual and semi-automatic techniques. As a result, 125 verb classes, 352 noun classes and 19 super noun categories were created. In order to assure the quality of annotations 300 random samples were manually checked. They found that error rates are comparable to recently published datasets.

Remark Noun classes were divided into two: (1) multi-shot class group (those with ≥ 100 bounding boxes in training); and (2) few-shot class group (with 10 ≤ and < 100 bounding boxes in training) in order to see how the model deals with objects that appear only for a few times in the video.

2.4.3 Baseline Results

Since, the main motivation of the researchers was to see how the model performs in unseen environments, they created two different test sets (S1, S2) to evaluate the generalizability of the model. Seen Kitchens (S1) contain kitchens that are both in the train and test datasets. Unseen Kitchens (S2) contain sequences of kitchens that are not in the training set. Thus, all sequences of a kitchen are either in training or testing.

2.4.3.1 Method

Aim of the object detection challenge is to classify and localize multiple objects that are in interaction (pre, during, post interaction) using the predefined 352 noun classes and active object bounding boxes. Due to its state-of-the-art performance, researchers chose Faster R-CNN (Region based Convolutional Neural Network) [57] with a base architecture of ResNet-101 [44] pre-trained on MS COCO [35] to solve the object detection problem.

2.4.3.2 Implementation

8 NVIDIA Tesla P100 GPUs were used for training the model on the EPIC-KITCHENS dataset. Clearly, these GPUs represent enormous computing power which is not available for this project. In order to improve the performance of the model the following settings were used: (1) Learning rate was set to 0.0003 decaying by the factor of 10 after 30k and 40k iterations; (2) Mini-batch size of 4 was used on a single GPU; (3) Images were rescaled such as their shortest side was 600 pixels (aspect ratio was maintained); (4) Stride was set to 16 on the last convolutional layer for feature extraction; (5) Non-maximum suppression (NMS), a post-processing

algorithm responsible for merging all detections that belong to the same object, was used with IoU threshold of 0.7; finally (6) Number of proposals in the Region Proposal Network (RPN) was set to 300.

2.4.3.3 Testing

Lastly, annotated images were evaluated using Mean Average Precision (mAP) metric from PASCAL VOC [34] at IoU thresholds of 0.05, 0.5, and 0.75 similar to MS COCO [35]. Two test sets were used, one that contained kitchens that were already showed in training (S1) and one with kitchens only showed in testing (S2).

		15 Most Frequent Object Classes														Totals		
	mAP	pan	plate	bowl	onion	tap	pot	knife	spoon	meat	food	potato	cup	pasta	cupboard	lid	few-shot	many-shot	all
S1	IoU > 0.05	74.00	72.61	71.50	60.72	84.44	69.97	44.03	40.93	29.65	58.52	62.82	53.30	78.39	51.95	62.77	9.71	49.80	38.23
	IoU > 0.5	67.60	66.21	65.98	39.96	73.80	64.71	28.80	23.89	20.75	49.85	55.48	42.99	69.75	29.20	58.48	6.98	36.50	28.06
	IoU > 0.75	21.94	44.60	39.48	3.52	25.83	19.67	3.42	2.59	5.27	15.78	13.18	8.00	24.53	4.05	26.51	0.36	8.73	6.50
S2	IoU > 0.05	75.94	87.36	72.72	47.61	78.14	75.92	55.51	41.28	71.59	38.61	N/A	44.62	80.58	53.88	58.40	6.00	51.71	40.61
	IoU > 0.5	62.88	84.86	68.61	32.18	59.75	62.86	39.60	27.52	53.54	35.47	N/A	39.19	76.27	32.54	49.36	5.32	36.27	28.57
	IoU > 0.75	14.56	62.82	38.44	2.25	4.89	14.91	3.85	1.51	9.56	8.10	N/A	7.60	43.30	5.61	25.48	0.18	9.05	7.04

Table 3. Baseline results produced by Faster R-CNN from EPIC-KITCHENS [1]

Table 3 shows mAP for the 15 most frequent classes, also for few-shot and multi-shot class groups both in S1 and S2. Overall performance at standard IoU (> 0.5) is around 28% which shows that active objects are harder to detect than in most existing datasets. Performance for S1 and S2 are comparable that demonstrates good generalization capability. More challenging classes were “meat”, “knife”, and “spoon”. Finally, few-shot classes perform poorly compared to the many-shot class group.

It can be concluded that the object detection problem on the EPIC-KITCHENS dataset is more challenging than on other popular datasets (see Table 2) and requires more specialist approach than applied in the baseline solution.

2.5 RetinaNet Algorithm

Even though RetinaNet is a single unified network it consists of several subnets. In theory RetinaNet works with any classic backbone like ResNet [44], VGG [61] or DenseNet [69]. In this particular problem, I am going to use ResNet-50 since it is faster and less computationally intensive to train. It takes an input image and processes it through several layers where each layer outputs a feature map of different scale. Early layers capture higher level information, while layers deeper in the network produce more fine-grained feature maps. Feature maps are combined using a Feature Pyramid Network (FPN).

2.5.1 Feature Pyramid Network (FPN)

RetinaNet adopted the FPN design from [64]. FPN takes the feature maps produced by the third, fourth and fifth layers of ResNet as inputs. Smaller feature maps are upsampled with nearest neighbour method then 1x1 convolution is applied to uniformize the number of channels to 256. FPN constructs a rich multi-scale feature pyramid from the input image.

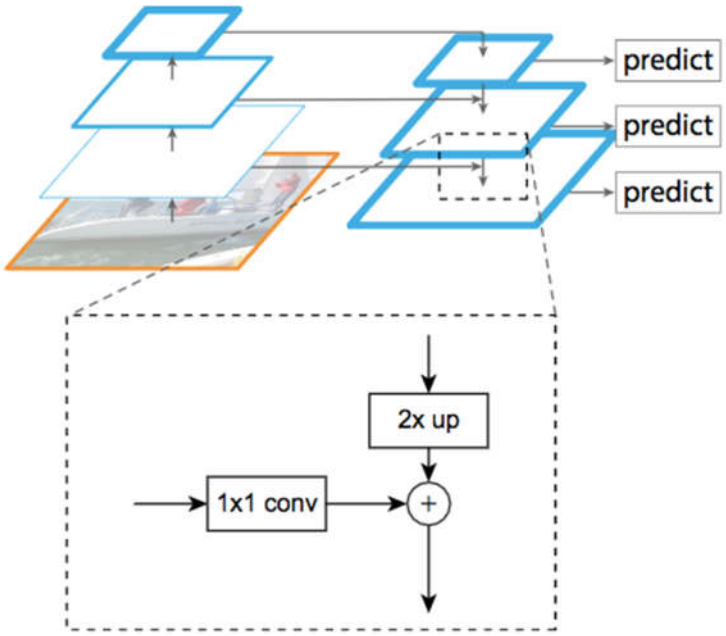

Figure 26. Visualisation of Feature Pyramid Network by A. Douillard [70]

2.5.2 Anchor Boxes

Anchors are fixed size bounding boxes used for localising objects on feature maps. FPN applies several anchor boxes of different size and ratio on each level. It uses 9 anchors per level in total and covers the scale range of 32-813 pixels across levels. Each anchor box is assigned to a vector of length 4 of box regression coordinates and a vector of length K where K is the number of classes. Furthermore, all anchors are assigned to maximum one ground-truth box using IoU threshold 0.5, thus the corresponding entry in the assigned vector of length K is set to 1 and all others are set to 0.

2.5.3 Classification and Regression

Two identical Fully Convolutional Neural Networks (FCN) are attached to the FPN where feature maps of each level are fed into the networks. Number of parameters of FCN are not dependent on the size of the input image therefore feature maps of different sizes can be fed into the same network. The first FCN is the classification subnet which is responsible for predicting the probability of object being present for each anchor box and object class. The second FCN is responsible for predicting bounding box coordinates. It predicts the relative offset between the anchor and predicted bounding box. Even though the classification and regression subnet are identical structure-wise, they do not share parameters.

After duplicate boxes of the same class are removed using non-max suppression, the bounding box corresponding to the highest scoring class is chosen as final prediction. If none of the boxes have higher score than a threshold then the object is considered to be part of the background.

2.5.4 Focal Loss

RetinaNet combines well established ideas from the Computer Vision literature, what makes RetinaNet novel is its loss function, Focal loss. Girshick *et al.* [38] identified extreme class imbalance as the main obstacle of single shot detectors to achieve high accuracy. Focal loss addresses the imbalance between foreground and background classes by assigning low weight

to well classified examples that are usually the background. Focal loss adds a modulating factor to the cross-entropy loss function for binary classification $CE(p_t) = -\log(p_t)$ where p_t is the confidence to be right in a binary classification.

$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

Modulating factor $(1-p_t)^{\gamma}$ with tuneable focusing parameter γ oscillating between 0 and 5 is able to down-weight easy examples and force the model to learn on hard negatives.

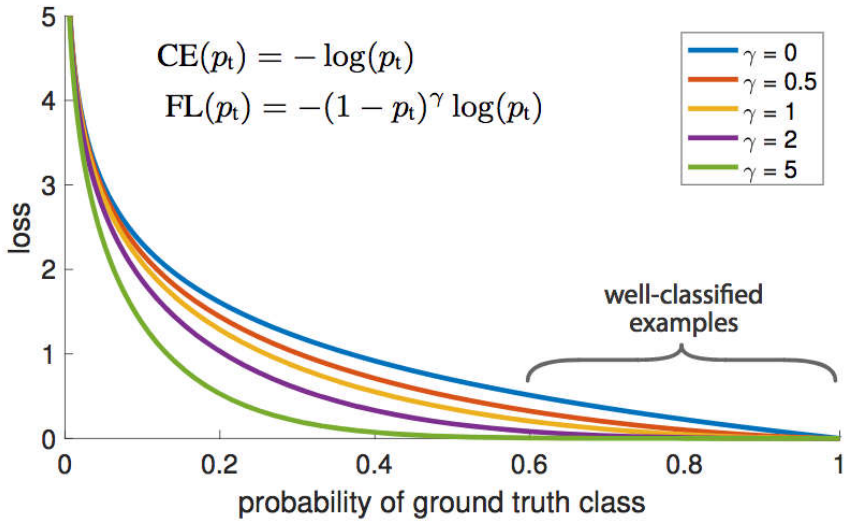

Figure 27. Focal loss under various modulating factors from Girshick *et al.* [38]

Girshick *et al.* [38] showed that $\gamma = 2$ and $\propto = 0.25$ work best in practice and RetinaNet is robust to $\gamma \in [0.5, 5]$. Focal loss is applied to each anchor box and calculated as the sum over all anchors normalised by the number of anchors assigned to a ground-truth box.

3 Methodology

When training a Deep Neural Network many decisions have to be made, for instance choosing the number of layers, the number of hidden units, the number of epochs, activation functions and many more. It is impossible to get all these hyperparameters right for the first time, therefore it is almost entirely sure that the first trained model will underfit or overfit the data. A model that underfits the data is too simple to capture the underlying function while a model that overfits the data is too complex and learned the characteristics of the training data so much that it is not able to generalise well. Either way the hyperparameters need to be readjusted and the model needs to be trained again. As described, building a Neural Network is an iterative process where one must come up with an idea of hyperparameters, code the network settings, and train and evaluate the model again and again until finding the desired model.

In these experiments we are looking for a model that is able to perform well on unseen data, namely has the same complexity as the underlying true function. Finding a model with good generalisation ability boils down to the *bias variance trade-off*. A model that is too simple has high bias and low variance, while a model that is too complex has high variance and low bias. Bias refers to the difference between the average prediction and the observed value. Variance describes the variability of the models, namely it is the average squared deviation of the predictions of the model from the average prediction.

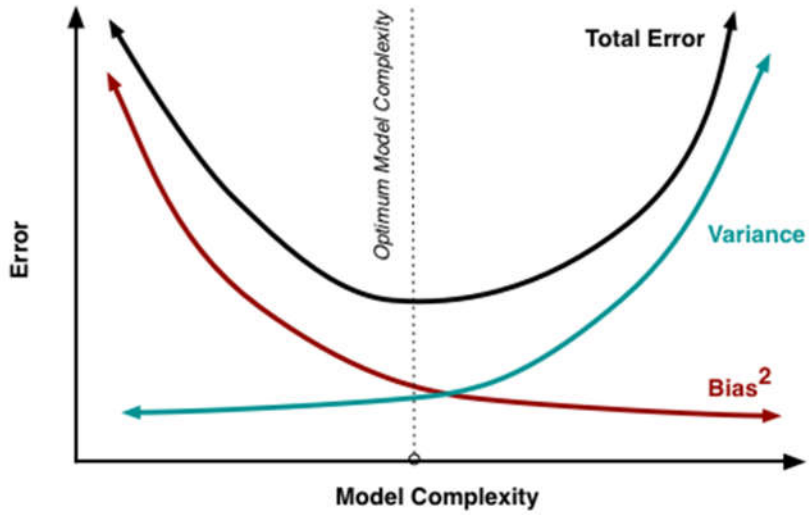

Figure 28. Visualisation of the bias variance trade-off [71]

Consequently, the goal of the experiments is to balance between bias and variance and find the model with the optimal level of model complexity, bias and variance that has the best generalisation ability.

3.1 Data Splits

In order to measure the level of bias and variance two main metrics are used, training error and validation error. High training error is the sign of high bias, while high validation error refers to high variance. More complex a model is the lower the training error, on the contrary validation error after reaching the optimal complexity starts to rise. Consequently, the correct model is the one with the lowest validation error. Therefore, setting up the training, validation and testing datasets correctly can very much help the experiments to find the desired model.

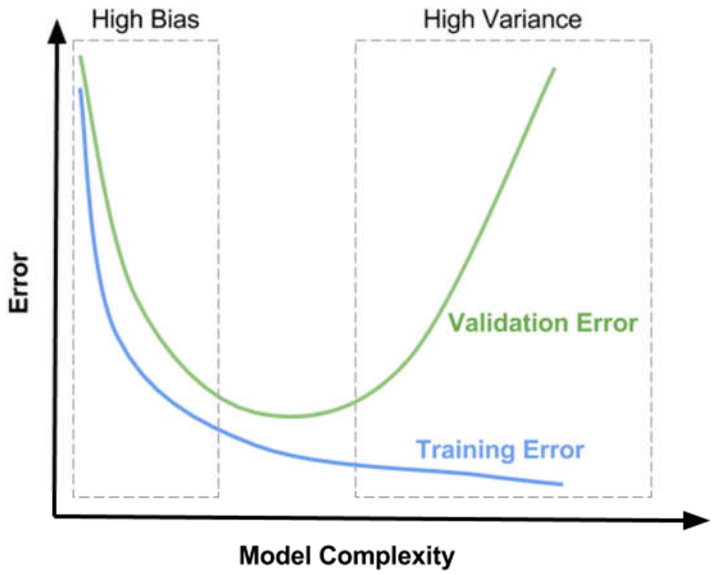

Figure 29. Relationship between error and the bias variance trade-off [72]

Accordingly, the available data is divided into three subsets, by name training, validation (also called development or dev) and testing sets. Logically, the model is trained on the training set then evaluated on the validation set. If the model needs to be readjusted, then the model is trained again on the training set and evaluated on the validation set until the desired model is found. After training, the best model found is evaluated again on the test set in order to have an independent unbiased final evaluation.

Previously, machine learning practitioners divided the data in the ratio 60:20:20 (train:dev:test) or 70:30 (train:test). However, earlier datasets only had 10,000 examples at most. Recently, much larger datasets are available with millions of examples, as result the ratio of training, validation and test sets changed. Size of the validation and test sets are much smaller as they are only necessary to evaluate the algorithms. Generally, 10,000 instances are enough for the validation and test sets from a dataset with a million examples. Consequently, the data is divided in the ratio 98:1:1 (training:dev:test).

Data splits for the EPIC-KITCHENS dataset and for this particular object detection problem will be discussed in Section 4.2 where implementation of the dataset will be presented.

3.1.1 Cross Validation

Beyond having a validation set to evaluate experiments, a commonly used approach to validate models is *k-fold cross-validation*. K-fold cross-validation divides the training data into k (usually 10) non-overlapping subsets where k-1 sets are used for training and 1 for validation. Training and validation processes are repeated until each subset has been used for validation exactly once. Cross-validation prevents the model from overfitting the validation set and gives a more accurate picture on the performance of the model.

However cross-validation has high computational expense compared to simple validation, therefore less often used for evaluating Deep Neural Networks. Use of cross-validation for this problem will be discussed in Section 4.4.3.

3.2 Hyperparameter Search

As said in the beginning training a model requires many decisions regarding choosing the value of hyperparameters. Hyperparameters are predefined aspects of the model that influence the process of learning thus the complexity of the model. For instance, important hyperparameters to set are learning rate, momentum, mini-batch size and number of hidden units.

Grid search approach is commonly used to optimise hyperparameters. Grid search involves picking a few values for each hyperparameter and trying all the combinations. It works well with few hyperparameters however Deep Neural Networks tend to have numerous hyperparameters. Therefore, in Deep Learning it is recommended to sample the hyperparameters at random in an appropriate range. Since, sampling random rather than using grid search ensures that the hyperparameter space is more richly explored. In addition, longer training time makes random search more sensible since grid search may produce many week models.

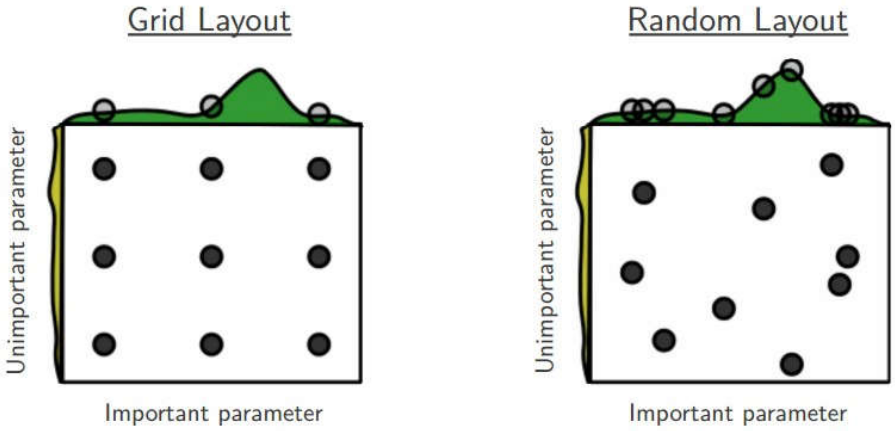

Figure 30. Illustration of grid search and random search by Bergstra and Bengio [73]

Random search can be further improved using coarse-to-fine search, namely after randomly sampling points from the hyperparameter space and finding the best model, zooming in and sampling more densely from the neighbourhood of the best model.

Given the above, random search will be implemented to optimise the hyperparameters. Details of the implementation will be discussed in Section 4.4.1.

3.3 Regularisation

Regularisation methods are able to control the level of bias in the model and its parameters. Bias can be introduced to the model by imposing penalty on the size of the weights as part of the cost function.

L2 regularisation, also called weight decay is commonly used to reduce the complexity of neural networks. Squared Euclidean norm, also called Frobenius norm along with a tuning parameter, λ is added to the cost function. By increasing the value of λ, the impact of many hidden units will be reduced which results in a simpler model. Accordingly, the tuning parameter controls how much regularisation is applied.

$$J(w^1, b^1, \dots, w^L, b^L) = \frac{1}{m}\sum_{i=1}^{m} \propto \left(\mathrm{y}^i, y^i\right) + \frac{\lambda}{2m}\sum_{l=1}^{L} ||w^l||_F^2 \qquad (2)$$

Where J is the cost function that the network tries to minimise, w is a parameter matrix and b is a real number denoting the bias for each layer l. w is an $n^l \; x \; n^{l-1}$ dimensional matrix where n is the number of units in layers l and $l-1$. Total number of layers is L. Sum of the squared Euclidian norm of w times λ over $2\mathrm{m}$ is added to the loss function as regularisation, where λ is the tuneable regularisation parameter and $2\mathrm{m}$ is a scaling constant.

RetinaNet uses weight decay of 0.0001 as part of the training loss which it the sum of the focal loss and the standard smooth L1 loss used for box regression.

3.4 Big Data Era

In the pre-Deep Learning era the bias variance trade-off was a highly discussed topic, however it is increasingly less important. Recently, many tools appeared that enabled Deep Learning practitioners to reduce bias or variance without hurting the other [74]. For instance, having a deeper network or training the network longer are able to reduce bias without increasing variance. Similarly, collecting more data or applying regularisation are able to reduce variance without increasing bias.

According to Andrew Ng [74] after training a network one should look at the training error to assess bias. If training error is high, then one should implement a deeper network or training the network for longer time. After fitting the training data, one should look at the validation error to assess variance. If validation error is high, one should collect more data or apply regularisation until fitting the validation set. It is beneficial to know the optimal error before examining the training and validation error, however it is not always available.

In this particular problem, I am going to follow the traditional way of evaluating algorithms, namely trying to find the model with the lowest validation error rather than fitting the training and development sets independently. Since the dataset size is small and optimal error for the problem is not available, training and validation error cannot be assessed by itself.

4 Implementation

This section details the implementation of EPIC-KITCHENS including data sampling and preprocessing and the RetinaNet object detection network including training and evaluation. It describes the hyperparameters investigated, furthermore the loss function and optimization algorithm used in this instance.

4.1 Environment

In order to make the solution easy to handle and scalable, first RetinaNet with ResNet-50 backbone was implemented using a smaller subset of the EPIC-KITCHENS dataset on CPU. Later, using a larger subset the environment was copied to a remote machine and the algorithm was implemented on GPU. Results presented in this paper are from experiments run on GPU.

First, the implementation environment was a MacBook Air laptop computer with an Intel Core i5 1.60 GHz CPU, 8 GBs of RAM and Intel HD Graphics 6000 GPU. The operating system was macOS High Sierra 10.13.4.

In order to set up the software environment a new Anaconda environment was created with Python 3.6, Keras 2.2.0, OpenCV 3.4.1, and Tensorflow 1.8.0 as specified in the RetinaNet Keras implementation [41]. The environment was tested with a RetinaNet model with ResNet-50 backbone pre-trained on MS COCO. Inference time was 23-29 seconds per image on CPU.

Second, the implementation environment was a Google Colab Notebook which backend was running on Google Cloud Compute Engine. Colab Notebook comes with Ubuntu Linux 17.10 operating system, Tesla K80 GPU with compute capability 3.7 and NVIDIA Driver version 384.111. Necessary libraries for running the Keras RetinaNet implementation, Keras 2.2.0, OpenCV 3.4.2, Numpy 1.14.5, keras-resnet 0.1.0, matplotlib 2.1.2, Cython 0.28.5, pillow 4.0.0, scipy 0.19.1 and h5py 2.8.0 were installed.

The object detection system and its related methods were implemented as a combination of pre-existing and self-programmed Python 3.6 tools.

4.2 Dataset Implementation

RetinaNet was trained, validated and tested on the EPIC-KITCHENS dataset. Due to time and computing power constraints only, a part of the data was used for solving the object detection problem. First, I decided to use the object detection images from the first video (P01_01) and train a model on CPU. However, I realised that having circa 3000 images and 300 classes is not feasible as each class appears only a few times which makes it very difficult for the model to learn the object classes. According to [1], learning from egocentric video is more difficult than from third person video, having a small dataset makes it impossible for a model to have good performance.

Therefore, I reduced the number of classes to one and decided to make a ‘pan’ detector. This way my RetinaNet implementation is still comparable with the baseline results [1] in a single class, moreover it is possible to have around 3000 images sampled from many kitchen environments and have higher accuracy compared to having 300 classes. Having a single class requires more data preprocessing. Annotations need to be assorted since only those annotations are necessary that describe object detection images with class ‘pan’. Consequently, object detection images need to be assorted as well.

Fortunately, negative examples are not necessary to include as Tensorflow detection models use parts of the image that do not belong to the annotated object as negatives [75].

4.2.1 Data Summary

Besides object detection images, the EPIC-KITCHENS dataset provides two CSV files [76] containing the noun classes and object labels for the images. In the first file 352 noun classes are described in three columns by a noun identifier, a class key and a list of subclasses. See an example line from the file below.

```
   1,pan,"['pan', 'pan:sauce', 'pan:frying', 'pan:cake', 'saucepan',
'saucepan:empty', 'saucepan:small', 'wok', 'wok:scour', 'pan:content']"
```

In the second file are 389,812 object labels described in six columns by a noun identifier, a noun, a participant identifier, a video identifier, a frame number, and four bounding box coordinates. Bounding box coordinates define the top left corner, height and width of boxes. In some cases, several bounding boxes are defined in one annotation describing multiple objects of the same class in the same image. Frame names are included in the names of object detection images that is how the model is supposed to find images based on annotations. See an example line from the file below.

```
20,bag,P01,P01_01,056971,"[(652, 954, 222, 284)]"
```

Object detection images are supplied in compressed folders per video. All object detection images have JPG format and 1080p resolution however there are some exceptions.

4.2.2 Data Preprocessing

RetinaNet [41] expects two CSV files, one with the class mapping and another with object annotations. Class mapping should contain the class name and class identifier consequently. Annotations should contain the path to each image along with the bounding box coordinates and the class name. It should contain a single annotation per line where the bounding box coordinates define the top left and the bottom right corners.

Accordingly, CSV files provided by the EPIC-KITCHENS dataset were preprocessed to meet the expectations of RetinaNet. First, the file containing noun classes was reduced to a single line where the class name was ‘pan’ and the class identifier was 0. There is no need to include background class. Second, the file containing annotations were preprocessed to contain a single annotation with a single bounding box per line. Annotations were assorted such that only those annotations were kept that describe ‘pan’ objects. Bounding box coordinates were transformed from top left corner, height and width parameters into parameters describing top left and bottom right corners. Some annotations do not have bounding box coordinates but are assigned to class labels, these annotations describe pre and post interactions with objects. Even though RetinaNet does not need negative examples, they still help the training of the algorithm to not to see objects of interest everywhere. Therefore, I included the annotations without bounding boxes as negative examples similarly to [70].

I sampled 3609 object detection images from fifteen different kitchen environments of which 366 images are negative examples. All images have 1080p resolution except object detection images corresponding to participant number 12 where images have 720p resolution.

Figure 31. Example object detection images

Similarly, to the EPIC-KITCHENS baseline implementation [1], images were rescaled so their minimum side was equal to 600 pixels. Following [38], I applied horizontal image flipping as the only form of data augmentation.

4.2.3 Data Splits

Similarly to [1], I created two test sets, S1 which includes images from kitchen environments seen both in training and testing and S2 which includes images from kitchen environments that were not used in training. Accordingly, I held out the complete sequences of two participants (P20, P26) for S2 testing. Next, I randomly sampled images from the remaining training set for S1. I cleaned both S1 and S2 images such that they do not include negative examples. As a result, S1 consists of 623 images and S2 consists of 203 images. All in all, the two test sets together form 23% of the dataset.

Keras RetinaNet implementation [41] requires a separate validation set that is used for evaluating models at the end of each epoch. Thus, I randomly sampled 628 images from the training examples. Similarly to the test sets, I excluded negative examples. Validation set forms 17% of the dataset.

As a result, 2138 images remained for training purposes. Hence, 60% of images were used for training.

Data splits	**Directory**	**# of images**	**% of all images**
Training	data/imgs/train	2138	60
Validation	data/imgs/validation	628	17
S1 testing	data/imgs/test/S1	623	17
S2 testing	data/imgs/test/S2	203	6
All images	-	**3592**	**100**

Table 4. Data splits described by size

4.3 Transfer Learning

Rather than starting the training process from randomly initialized weights, weights from a trained network can be downloaded and used as pretraining. This way, knowledge can be transferred from a large public dataset, e.g. MS COCO [35], ImageNet [33] or PASCAL VOC [34],

to the task of interest. Knowledge from a pre-trained network can be transferred to the current problem in several different ways.

For instance, a CNN that is pre-trained on a large dataset can be used as a fixed feature extractor. This idea is based on the insight that early layers of networks tend to extract more generic features, e.g. edges, colours. Naturally, it is not necessary to freeze the entire backbone, the number of frozen layers depends on the size of the dataset and the similarity between the original and the current datasets. Freezing only the early layers is used when the dataset is large and similar to the original dataset. Freezing the entire backbone is beneficial when the current dataset is small and similar to the original dataset.

Second option is to use the weights of a pretrained network as initialization but rather than freezing layers of the network, fine-tuning them using backpropagation. Fine-tuning is recommended when the dataset is large and very different from the original dataset.

By using transfer learning, a model is able to achieve better performance by having a higher initial skill, a steeper rate of improvement and better converged skill as shown on Figure 32.

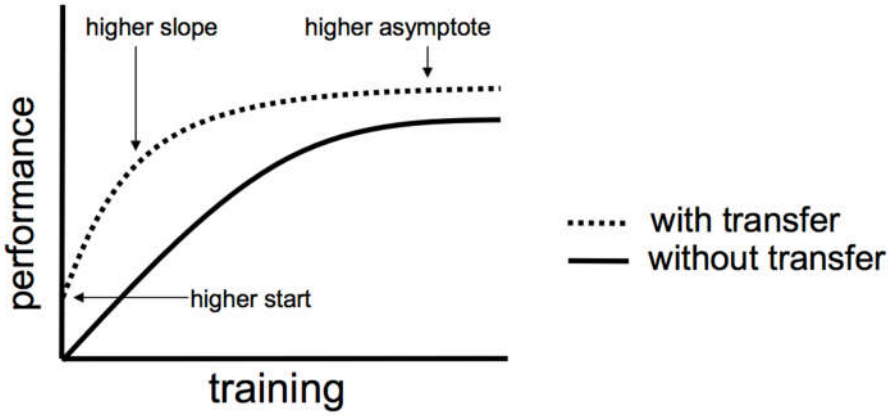

Figure 32. Benefits of using transfer learning by J. Brownlee [77]

Since the training set for EPIC-KITCHENS object detection problem only consists of roughly 2000 images and MS COCO has more than 7000 images of bowl objects which are a similar to frying pan, I used transfer learning method to achieve better performance along with shorter training time. A RetinaNet model with ResNet-50 backbone pretrained on MS COCO [35] is available to download from [78]. Keras implementation of RetinaNet offers the option to freeze the backbone, namely the ResNet and FPN in training. Consequently, while training I loaded the weights from the pretrained RetinaNet-50 network and froze the entire backbone. Therefore, training only affected the classification subnet and the regression subnet.

See a training script including the 'weights' and 'freeze-backbone' arguments below. The 'weights' argument expects the location of a model with weights that can be loaded as initialisation. The 'freeze-backbone' argument freezes the ResNet and the FPN in training.

```
!python3 keras_retinanet/bin/train.py
    --weights snapshots/resnet50_coco_best_v2.1.0.h5
    --batch-size 4
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --epochs 3
```

```
--steps 535
--snapshot-path snapshots/experiment_03_gpu
--freeze-backbone
--image-min-side 600
csv data/annotations_train.csv data/classes.csv
--val-annotations data/annotations_val.csv
```

Layers that shape of weights do not match with the pretrained model are skipped. As a result, loading weights to two layers in the classification subnet were skipped due to mismatch in shape.

4.4 Network Implementation

As described in Chapter 3, building on object detection and deep learning models in general requires to perform an iterative process where many different ideas, each with different hyperparameter settings are implemented and evaluated in some way. From several implementations the best performing model is chosen as the final object detector. Important parts of the iterative process are the three stages of model building, training, validation and testing, corresponding to the three different data splits described in Section 4.2.3. These data splits allow to efficiently measure bias and variance levels and quickly iterate through the different stages.

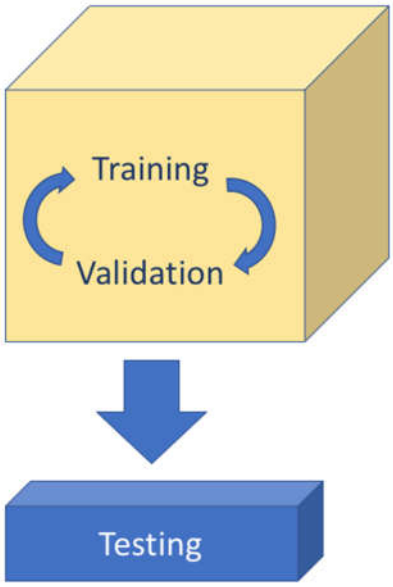

Figure 33. Stages of model building

Learning process of the model is done using the training split, while the validation set is used for assessing the performance of the model on unseen examples. Therefore, validation performance gives a grasp of the expected result on the test data. Finally, testing is the last independent test on unseen data of the best model found.

RetinaNet was implemented using the excellent Keras RetinaNet framework [41] which is an implementation of RetinaNet as described in [38]. It supplies ready-to-use functions and scripts for model training and evaluation. It is built with Python 3.6, OpenCV 3.4, Keras 2.2.0

and Tensorflow backend. Specific ways of applying the framework to build models are described next.

4.4.1 Training

Depending on the size of the dataset and the available computational resource, there are different training methods to choose from. If the dataset is large and not much computational resource is available such that only one model can be trained at a time, then the method called babysitting is recommended. In babysitting a single model is trained over a long time and hyperparameters are set continuously. If a large dataset and lot of computational resource are available, then several models can be trained in parallel. In this case many predefined hyperparameter settings are tried and at the end the best performing model is picked as the final model.

In this instance, babysitting method was used to train an object detector on the EPIC-KITCHENS dataset due to computing power constraints. Keras RetinaNet framework allows this type of training method as it can be configured to create and save a snapshot of the model after each epoch. Also, training can be resumed from a snapshot. Therefore, a single model is built over several iterations each with custom hyperparameter setting.

4.4.1.1 Loss Function

The training process is guided by a loss function where the goal is to find a model that minimises that loss function. In this instance, training loss is calculated as the sum of the focal loss [38] and the standard smooth L1 loss [60] used for box regression. Traditionally, gradient descent learning algorithm is used for minimising the loss function. In this case, an extension of stochastic gradient descent, the Adam optimization algorithm [79] was applied.

Focal loss is used for computing the classification loss for each minibatch of training examples. It is a modified version of cross-entropy loss that addresses the extreme imbalance of foreground and background classes in training. Focal loss was explained in detail in Section 2.5.4.

Smooth L1 loss is used for computing the regression loss similarly for each minibatch of training examples. It is a standard for box regression applied in several object detection systems [60][57][51]. According to [60][57][51], smooth L1 loss is less sensitive to outliers and prevents exploding gradients.

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i) \tag{3}$$

in which

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 5 & otherwise \end{cases} \tag{4}$$

Where u is the ground truth class and v is the ground truth box regression target. L_{loc} is defined over a tuple of true bounding-box regression targets for class $u, v = (v_x, v_y, v_w, v_h)$, and a predicted tuple $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ again, for class u. Smooth L1 computes the distance x between the two vectors. For background classes L_{loc} is ignored since there is no notion of a ground-truth bounding box.

4.4.1.2 Optimisation Algorithm

Right choice of optimization algorithm can accelerate the process of finding the minimum of the loss function. In this instance, Adam optimisation algorithm [79] is used which is an up-graded version of stochastic gradient descent (SGD). Rather than maintaining a constant learning rate like SGD, Adam keeps a running average of the gradient and the squared gradient, as well as includes a bias correction term. It performs adaptive moment estimation, hence the name.

According to [79], Adam requires little memory and is computationally efficient, also well suited for problems with large datasets, as well as with large number of parameters. Therefore, adaption of Adam has become widespread in Computer vision and Natural language processing (NLP).

Other than the learning rate, the implementation of Adam algorithm follows the default parameters provided by the original paper [79].

4.4.1.3 Hyperparameter Search

According to [38], most design parameters in RetinaNet are not sensitive to exact values. Therefore, parameters determining the architecture, e.g. number of hidden layers, are used as described in [38].

Due to the overwhelming number of hyperparameters random search is used to richly explore the search space. Coarse to fine sampling is ignored because of the small number of experiments performed and the style of training.

Other than the design parameters, hyperparameters can have significant effect on the learning process and consequently on the performance of the model. Hyperparameters to be tuned during training are as follows.

Learning rate - Learning rate controls how much the weights are adjusted with respect to the loss gradient. By convention, learning rate is decreased gradually in the training process in order to make the gradient descent converge. The lower the value of learning rate, the slower the gradient descent travels along the downward slope. If the learning rate is too high, gradient descent may overshoot the minimum and fail to converge. Therefore, the right configuration of learning rate is crucial to find the desired model. According to Y. Bengio [80] learning rate should be as high as possible until validation error goes up. The values investigated were 0.00001 and 0.0001 and 0.000001.

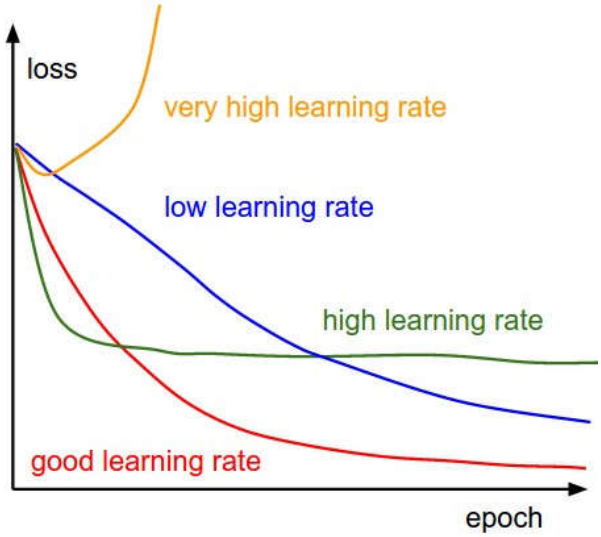

Figure 34. The effects of different learning rates on loss [81]

Momentum - Momentum encourages movement in the direction of previous weight changes therefore it leads to better convergence and faster learning. It gains knowledge from previous steps by keeping a running average of gradients and uses that average rather than the current direction of the data. Standard value 0.9 was implemented in each experiment.

Minibatch size - Minibatch size is the number of examples propagated through the network at each step. Weights are updated after each propagation. Smaller minibatch size allows updates to be made more frequently, while larger minibatches are better fitted for efficient parallel processing on GPUs. Using minibatches is beneficial since it requires less memory and faster than batch gradient descent as well as smoother than stochastic gradient descent. The values investigated were 2 and 4 following [38] and [1].

Number of epochs - Epoch, also called training iteration is one complete presentation of the data set to be learned to the network. Due to the style of training, a single model was trained through 12 epochs with different hyperparameter settings rather than many models in parallel through different number of epochs.

Steps per epoch - Steps per epoch is the number of propagations in a single epoch. Value of steps per epoch should be the number of training images divided by the minibatch size. Consequently, the steps per epoch used in experiments corresponding to minibatch size 4 was 535 and 1069 in experiments corresponding to minibatch size 2.

Sampling random from the hyperparameter space is not effective, an appropriate range needs to be set based on previous experiences and the literature. Furthermore, in case of particular hyperparameters, e.g. learning rate, sampling uniformly within a scale is not sensible. Rather than sampling uniformly, sampling on a log scale is more effective. It is necessary because learning rate has multiplicative effects on the training dynamics, it multiplies the gradient in the backpropagation update. Therefore, values for learning rate are sampled on a log scale.

4.4.2 Non-maximum Suppression

As described in Section 2.2.4, non-maximum suppression (NMS) is used for discarding weak detections and proposing a final detection.

In the Keras RetinaNet framework [41] the training process works with training models that only contain the necessary layers for training. In order to use a training model to perform object detection on an image, it needs to be converted into an inference model. Models used in validation and testing are inference models, both use NMS to do inference.

In inference models, after decoding box predictions from at most 1k top-scoring predictions per FPN level, NMS is applied with score threshold 0.05, IoU threshold 0.5 and max 300 detections in order to produce the final detection. Hyperparameters of NMS were not changed while doing inference.

4.4.3 Validation

Having a validation set enables faster iterations through models and hyperparameter settings, moreover it prevents overfitting the training set. Consequently, it leads to a model with better ability to generalise.

Even though cross-validation is a highly recommended method, it is not used in this particular project due to the design of Keras RetinaNet framework and long training times.

Keras RetinaNet framework allows to perform validation after each epoch. Mean average precision (mAP) was used to measure the performance of models on the validation set.

4.4.4 Testing

Testing is a final independent test on unseen data of the best model found. Test results give an estimation about the performance of the deployed model.

Following the testing protocol of [1], two different test sets were sampled. S1 is a set of images from kitchen environments already seen in training and S2 is a set of images from new, unseen kitchen environments. S2 aims to test the generalizability of the final model to novel environments.

The best model was evaluated and compared to the baseline results using mAP from PASCAL VOC [34] with IoU thresholds of 0.05, 0.5, 0.75.

Scripts used for evaluation and converting the training model to inference model are shown in Appendix 2 and 3.

5 Results & Analysis

This chapter discusses the results produced by the RetinaNet algorithm in training, validation and testing. Using random search various hyperparameter sets were fine-tuned to find the best performing configuration. Then, a testing protocol from [1] was employed to evaluate the performance of the final model and compare it to the baseline results.

5.1 Experiments

Four different experimental setups were defined in training with three different hyperparameter settings. Due to the nature of training, each experiment was resumed from the best model found so far.

Experiment 1 - Weights for Experiment 1 were loaded from a model pre-trained on MS COCO [35] supplied by the Keras RetinaNet framework [41]. As described in Section 4.3, the entire backbone of the pretrained model was frozen, therefore training only affected the classification and regression subnets. Similarly to [1], in the first experiment minibatch size of 4 was applied that required 535 steps in order to run the entire training set through the network. Learning rate of 0.00001 was used which is the default setting of the Keras RetinaNet framework. The model was trained through 3 epochs. Training and validation results from the first experiment are presented in Table 5.

Epochs	Minibatch size	Steps	Learning rate	Training				Validation
				Time (s)	Loss	Regression loss	Classification loss	mAP
1	4	535	0.00001	790	2.2532	1.4293	0.8239	0.4847
2				871	1.6751	1.1919	0.4832	0.6378
3				853	1.4688	1.0789	0.3899	0.7423

Table 5. Training and validation results of the first experiment

Loss, regression loss and classification loss were reported after each step. These three metrics from the first experiment are presented in Figure 35.

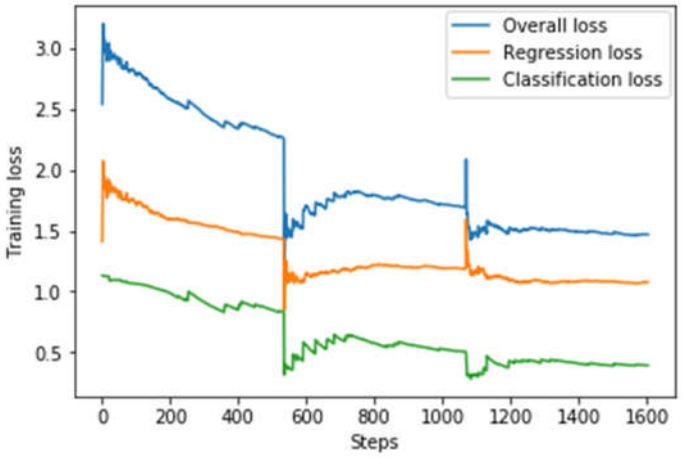

Figure 35. Training loss of the first experiment

Overall, both training loss and validation mAP dynamically improved. However, the loss curve after the first epoch did not significantly decrease. According to [81], the shape of the loss curve indicates that the learning rate is too high and the algorithm cannot converge.

Experiment 2 - Training for Experiment 2 was resumed from the best model which was gained after the third epoch in Experiment 1. In order to see if the algorithm is able to converge, learning rate was reduced from 0.00001 to 0.000001. Also, in transfer learning, use of lower learning rate is recommended since the original model has good weights already and they do not need to be altered that much. Following [38], minibatch size of 2 was applied in order to make updates more often in a single epoch. Due to the change in minibatch size, steps per epoch must be changed as well, since it should be equal to the number of training images divided by the minibatch size. Therefore, the new steps per epoch was 1069. The model was trained through 4 epochs in Experiment 2. Training and validation results from the second experiment are presented in Table 6.

Epochs	Minibatch size	Steps	Learning rate	Training				Validation
				Time (s)	Loss	Regression loss	Classification loss	mAP
1	2	1069	0.000001	862	1.2843	0.9706	0.3137	0.7961
2				891	1.1890	0.9005	0.2885	0.8163
3				1515	1.1053	0.8477	0.2576	0.8487
4				1523	1.0302	0.7905	0.2397	0.8527

Table 6. Training and validation results of the second experiment

As the model converges, I expected the validation results to stop improving. However, the validation results slowly improved which may mean that the algorithm is on a plateau, therefore instead of lower learning rate, higher learning rate may help to escape the plateau.

Interestingly, training time immediately increased after the second epoch resulting in a training time twice as much as previously. This may have been caused by distractions in the availability of the GPU or other connection problems between the Colab Notebook and the server.

Experiment 3 - Similarly to Experiment 2, training was resumed from the best performing model, gained after the fourth epoch in Experiment 2. Leaning rate was increased from 0.000001 to 0.0001 in order to see if the algorithm was stuck on a plateau or converging towards a local minimum. In case of a plateau, I expected slight improvement in both loss and validation rate. However, in case of the algorithm being in a valley and converging towards a local minimum, I expected the algorithm to perform poorer than previously. Also, according to Y. Bengio [80], learning rate should be as high as possible until validation error goes up. When the validation error goes up, the model starts to overfit the training data and loses its ability to generalise. Training and validation results from the third experiment are presented in Table 7.

Epochs	Minibatch size	Steps	Learning rate	Training				Validation
				Time (s)	Loss	Regression loss	Classification loss	mAP
1	2	1069	0.0001	1530	0.9769	0.7554	0.2215	0.8637
2				1856	0.9299	0.7159	0.2140	**0.8754**
3				1435	0.8739	0.6760	0.1979	0.8734
4				1432	0.8444	0.6524	0.1920	0.8658

Table 7. Training and validation results of the third experiment

Training loss continuously decreased throughout the four epochs, however validation mean average precision achieved a peak after the second epoch and then started decreasing. After the second epoch, the model probably started to overfit the training data therefore lost its ability to generalise on unseen data.

Experiment 4 - In order to see if the best model, gained after the second epoch in Experiment 3, is able to converge with smaller learning rate, I trained the model for one more epoch with learning rate 0.000001. Training was resumed from the best model from Experiment 3. Similarly to Experiment 3, other than learning rate, all hyperparameters remained unchanged. The model was trained through a single epoch. Training and validation results from the fourth experiment are presented in Table 8.

Epochs	Minibatch size	Steps	Learning rate	Training				Validation
				Time (s)	Loss	Regression loss	Classification loss	mAP
1	2	1069	0.000001	1257	0.8819	0.6789	0.2030	0.8738

Table 8. Training and validation results of the fourth experiment

Validation performance decreased while training loss showed better result. Even with smaller learning rate the model overfitted the training data, therefore performed poorer on the validation data.

Ultimately, the optimal configuration of RetinaNet was obtained in Experiment 3 after the second epoch. The final model was trained over 9 epochs using three different learning rates 0.0001, 0.00001 and 0.000001. Also, two different minibatch sizes were used 2 and 4 that required the usage of different steps per epoch, 1069 and 535 consequently. The model achieved 0.8754 mean average precision on the validation set.

5.2 Evaluation

The best model found was evaluated on two different test sets. Test set S1 consists of images from kitchen environments that were included in training, while test set S2 includes images from kitchen environments that were not seen in training. This testing protocol gives an insight into the generalizability of the model to unseen kitchen environments. The goal of the testing was to perform a last, independent check of the model that shows how it is expected to perform as a deployed system.

Similarly to [1], testing was performed with three different IoU thresholds, 0.05, 0.5 and 0.75. Metrics examined were the number of false positives, the number of true positives, recall, precision and average precision. Clearly, having one class results in average precision and

mean average precision being equal since mean average precision is the average of average precisions over all classes.

Results from S1 and S2 testing are presented in Table 9.

	S1 mAP	S2 mAP
IoU > 0.05	0.8957	0.8792
IoU > 0.5	0.8500	0.8246
IoU > 0.75	0.4508	0.1067

Table 9. mAP results of testing on S1 and S2 test sets

RetinaNet reached high mAP with IoU threshold 0.05 and 0.5, however its performance seriously dropped with IoU threshold 0.75. According to these results, RetinaNet struggled to get the boxes perfectly aligned with the objects. Seen and unseen environments are comparable with IoU thresholds 0.05 and 0.5, however performance on unseen split falls behind the performance on seen environments with IoU threshold 0.75. RetinaNet struggles to get accurate bounding boxes on unseen environments even more.

False positives, true positives, recall and precision are presented in Table 10.

	S1 (N = 636)				S2 (N =203)			
	FP	TP	Recall	Precision	FP	TP	Recall	Precision
IoU > 0.05	6962	629	0.9890	0.0829	1475	195	0.9606	0.1167
IoU > 0.5	6968	623	0.9795	0.0821	1479	191	0.9409	0.1143
IoU > 0.75	7164	427	0.6714	0.0562	1603	67	0.3300	0.0401

Table 10. Results of testing on S1 and S2 test sets

Precision and recall were calculated and summed after each detection in order to produce the necessary data for the precision-recall (PR) curve which is the base of calculating average precision (AP). Results shown in Table 10 were calculated after the last detection. In other words, they represent the last point of the PR curve. To calculate AP, precisions were averaged across equally spaced recall values.

High amounts of false positives can be explained by the low score threshold for detection during evaluation, 0.05. As a result, numerous false positives were produced with low confidence. Detections for the same object were merged into a single bounding box using non-maximum suppression.

Due to the high number of false negatives, precision does not provide valuable measure of the model performance. However, recall, which measures the percentage of positive results over the true number of positive examples, shows that RetinaNet was able to detect almost all objects with IoU threshold 0.05 and 0.5. RetinaNet failed to detect many objects with IoU threshold 0.75 which again shows that RetinaNet struggles to accurately position bounding boxes around objects.

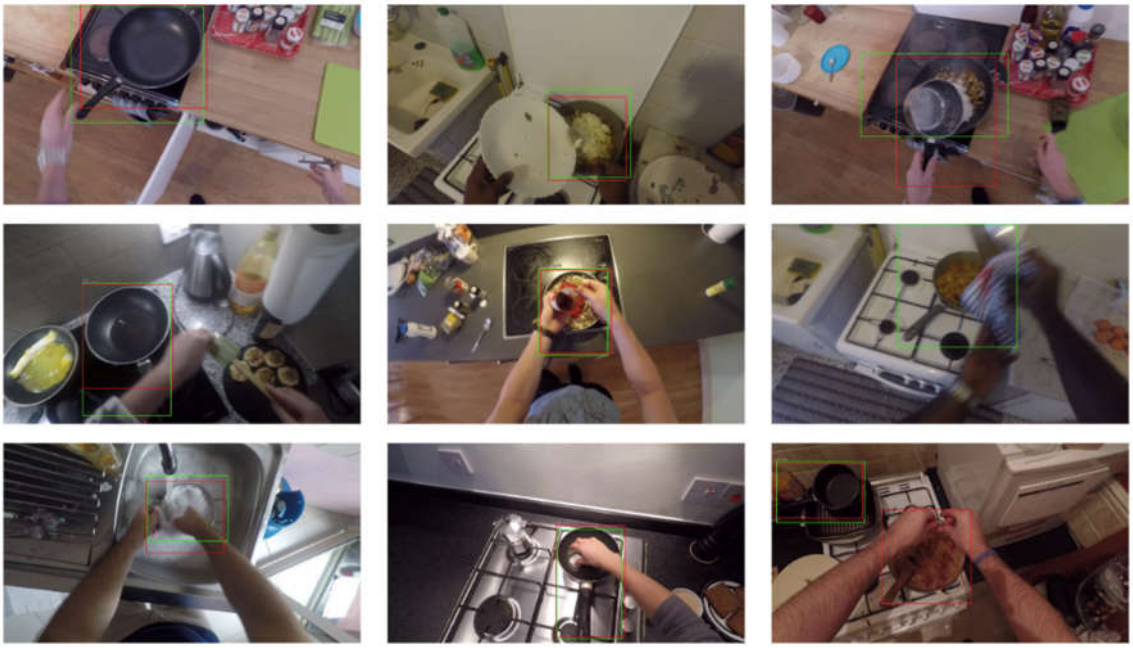

Figure 36. Qualitative results

Figure 36 shows qualitative results with ground truth bounding boxes shown in green and detections shown in red. Examples shown in the right-hand column are failure cases.

Given the above, RetinaNet is able to effectively localize 'pan' objects on the EPIC-KITCHENS dataset [1] with standard IoU, 0.5, however struggles to accurately position bounding boxes which is represented by the poor performance with IoU threshold, 0.75.

5.2.1 Comparing performance of RetinaNet and Faster R-CNN

Main goal of the dissertation was to compare baseline results produced by Faster R-CNN and results produced by RetinaNet on the EPIC-KITCHENS dataset [1]. Accordingly, compare the performance of state-of-the-art two-stage and one-stage object detectors in egocentric vision.

Both Faster R-CNN and RetinaNet were evaluated using the popular metric, mAP. The implementation of RetinaNet followed the testing protocol proposed by [1], as a result two test sets were used to evaluate the model. S1 test set included images from kitchen environments seen in training, while S2 test set included images from kitchen environments unseen in training. Both models were pretrained on MS COCO [35]. Results from S1 and S2 testing for both Faster R-CNN and RetinaNet are presented in Table 11. Note, that all results presented correspond to class 'pan'.

		Faster R-CNN mAP	RetinaNet mAP
S1	IoU > 0.05	0.7400	**0.8957**
	IoU > 0.5	0.6760	**0.8500**
	IoU > 0.75	0.2194	**0.4508**
S1	IoU > 0.05	0.7594	**0.8792**
	IoU > 0.5	0.6288	**0.8246**
	IoU > 0.75	**0.1456**	0.1067

Table 11. mAP figures for Faster R-CNN and RetinaNet at different levels of IoU for seen (S1) and unseen (S2) test environments

RetinaNet outperformed Faster R-CNN in most cases. Both Faster R-CNN and RetinaNet performed worse with IoU threshold, 0.75, however RetinaNet showed a more significant decrease in performance, especially on unseen environments. Both models show good ability to generalize across environments.

To summarize, RetinaNet is able to outperform Faster R-CNN in a single object class on the EPIC-KITCHENS dataset however this result does not confirm that RetinaNet generally performs better than Faster R-CNN on the EPIC-KITCHENS dataset.

6 Discussion

In this chapter, implementation and baseline results of Faster R-CNN by [1] will be compared to the implementation and results of RetinaNet presented in this dissertation.

Performance - In most cases, RetinaNet was able to outperform Faster R-CNN on both seen and unseen environments. Faster R-CNN performed better on unseen environments with IoU threshold 0.75. In other words, Faster R-CNN was able to more accurately position bounding boxes on unseen environments than RetinaNet. Even though, its performance is quite poor, meaning both Faster R-CNN and RetinaNet struggled to predict accurate bounding boxes on seen and unseen environments as well. RetinaNet has less significant decrease in performance on seen environments with IoU threshold 0.75. However, it shows more significant decrease than Faster R-CNN with IoU threshold 0.75 on unseen environments.

Generally, RetinaNet performs better than Faster R-CNN, however this may due to longer training time, different training style or the fewer number of classes, as we shall see later in this analysis.

Transfer learning - Both Faster R-CNN and RetinaNet were pre-trained on MS COCO [35]. Due to smaller dataset size, the base network of RetinaNet was frozen in training. In contrast, both the base architecture and subnetworks of Faster R-CNN were updated in training which required longer training time and more computing power.

MS COCO does not have the object class ‘pan’, however it has the object class ‘bowl’ which is similar in shape, size and colour. Furthermore, MS COCO includes images from diverse kitchen environments which could contribute to the high mAP of RetinaNet.

Size of the dataset - Due to time and computing power constraints, only a small part of the dataset, that included 15 different kitchen environments, was used for training and evaluating RetinaNet. In contrast, Faster R-CNN was trained and evaluated using the entire EPIC-KITCHENS dataset [1] which includes 32 kitchen environments. Given more time, RetinaNet could have been trained on a larger dataset sampled for two or more classes in addition to class ‘pan’. It would have been interesting to see how performance changes with additional classes. Given larger computing power, RetinaNet could have been trained on the entire dataset with 352 noun classes. It would have been interesting to see how RetinaNet deals with few shot classes which Faster R-CNN struggled with, as well as detecting small objects which is generally weakness of one shot networks.

Number of classes - Similarly to the dataset size, only a part of the classes was used to train RetinaNet. Due to smaller dataset size, it would have been very difficult for RetinaNet to learn all the 352 noun classes, therefore the number of classes were reduced to one. Consequently, the two models largely differ in the number of object classes they are able to detect. RetinaNet is a specialised model that is only able to detect a single class, ‘pan’, while Faster R-CNN is able to detect 351 different object classes in addition to ‘pan’.

At this point the comparison of the two networks can be questioned. However, if RetinaNet would be part of a large ensemble that is able to detect all 352 classes then comparing the performance of the ensemble with Faster R-CNN in each object class would be a valid com-

parison. Therefore, comparing the performance of RetinaNet and Faster R-CNN in class ‘pan’ is appropriate.

Generalization capability - Both Faster R-CNN and RetinaNet generalise well to unseen environments which is shown by the comparable performance on seen and unseen environments with IoU thresholds 0.05 and 0.5. However, neither of them is able to generalise well with IoU threshold 0.75. Poorer performance of RetinaNet on unseen environments with IoU threshold 0.75 might have been caused by fewer kitchen environments seen in training. In addition, one shot networks are generally less accurate than two shot networks which may have been contributed to poorer performance with IoU threshold 0.75 as well.

Training style - RetinaNet and Faster R-CNN differ in terms of training style. Due to limited GPU power, a single RetinaNet model was ‘babysitted’ through many epochs with different hyperparameter settings. In contrast, it is not stated in [1], however from the number of GPUs and the fixed hyperparameter setting, it can be assumed that many networks were trained in parallel and at the end the model with the best performance was picked as the final model.

Training style most likely did not affected performance significantly. However, babysitting gives more flexibility to navigate on the loss function than parallel training style. Therefore, babysitting may have contributed to better performance in case of RetinaNet.

Training time - RetinaNet was trained through 3 hours where an epoch took 1200s on average. Faster R-CNN was trained over 6.5 days [82]. Given 352 object classes, Faster R-CNN had 26.6 minutes to learn a single class on average. Consequently, RetinaNet had more time to learn a single object class which probably contributed to the better performance. However, RetinaNet was trained using a single GPU, while Faster R-CNN was trained using 8 GPUs which significantly accelerated the processing of images. It would be interesting to see how Faster R-CNN performs compared to RetinaNet given the same number of epochs per object class.

As detailed in this chapter, the implementation of Faster R-CNN and RetinaNet are significantly different as a result of different transfer learning protocol, dataset size, number of object classes, training style and time. Despite these differences, results produced by the networks are still comparable. However, one must keep in mind these differences when comparing the performance of the models.

7 Conclusion

7.1 Summary

Although due to mainly computing power constraints, some of the objectives had to be re-evaluated, the project as a whole was very successful. The final products are an object detection model that is capable of detecting frying pans on images as well as this technical report that details the implementation of the algorithm.

Almost all objectives described in Section 1.3 were met by the final product. RetinaNet, a one-stage object detector was trained and evaluated on the EPIC-KITCHENS dataset that is capable of detecting a single object class instead of 352 object classes as outlined in the objectives. In that single object class, the model was able to outperform the baseline results produced by a two-stage object detector, Faster R-CNN.

First, a profound literature overview was done that required quite a long time. The literature overview included learning about Computer Vision, Egocentric Vision, Convolutional Neural Networks and convolutional object detection. Furthermore, a selection of state-of-the-art object detectors were examined from different aspects. RetinaNet, a unified object detector with a fine-tuned loss function, was chosen to tackle the object detection challenge on EPIC-KITCHENS.

A single GPU was available for training that required the use only a part of the EPIC-KITCHENS dataset. In order to obtain good results, the number of classes were also reduced along with the dataset size. Around 3000 images were sampled from the dataset that contained the object of interest.

A Keras implementation with Tensorflow backend was selected to train RetinaNet. RetinaNet was trained through 9 epochs to obtain the optimal configuration of hyperparameters. The final model was evaluated on two test sets with seen and unseen kitchen environments.

The built object detector was capable of detecting frying pans with 0.8500 mAP with standard IoU threshold 0.5 on images of seen kitchen environments and with 0.8246 mAP with standard IoU threshold 0.5 on images of unseen kitchen environments. It was able to outperform Faster R-CNN that performed with 0.6760 mAP and 0.6288 mAP consequently.

RetinaNet model can be easily re-trained or replicated following this report to detect other objects of interest. RetinaNet showed promising results in detecting a single class on EPIC-KITCHENS, it may be capable of outperforming Faster R-CNN using additional or all the object classes.

Personally, I found my journey within this dissertation project fulfilling as well as challenging. I was pleased to work on a highly complex problem and was able to gradually understand the process of object detection and the behaviour of Convolutional Neural Networks. The project introduced me to a whole new world with numerous exciting challenges that I am looking forward to tackle in the future.

7.2 Critical Evaluation

Even though this project was a successful exploratory analysis into egocentric object detection, it has a lot of room for improvement. As mentioned previously, scope and objectives had to be re-evaluated while progressing with the project due to computing power and time constraints. In addition to these, this section provides a list of limitations for this work.

1. Background research and literature review were extremely time consuming and demanded a big slice of time available for this project. Instead of spending much time on reading research papers, I should have spent more time on the practical side of the project including data preparation and network implementation since both would have needed more time to meet the objectives.

2. Unlike planned, only a single object class, frying pan was used for training the network due to the few GPUs available for training. Similarly, only a part of the dataset that included the object of interest, 'frying pan' was sampled from EPIC-KITCHENS. 'Frying pan', is a relatively easy object, since it is large, has a characteristic shape and colour. Furthermore, sampled kitchen environments used for training and testing might have been presented an easier task for RetinaNet than the entire EPIC-KITCHENS dataset.

3. Reducing the number of classes to one caused the RetinaNet model to have a largely different implementation compared to Faster R-CNN. Thus, results could not be examined and compared in case of few-shot and many-shot classes; and large, medium and small object classes.

4. Only form of data augmentation used was horizontal flipping. Dataset size could have been increased by using random transform including rotation, translation, shearing and scaling.

5. Some ground-truth bounding boxes were not perfectly positioned in the dataset, moreover there were bounding boxes of "knife" object class annotated as pan. It might have been annotation or sampling error.

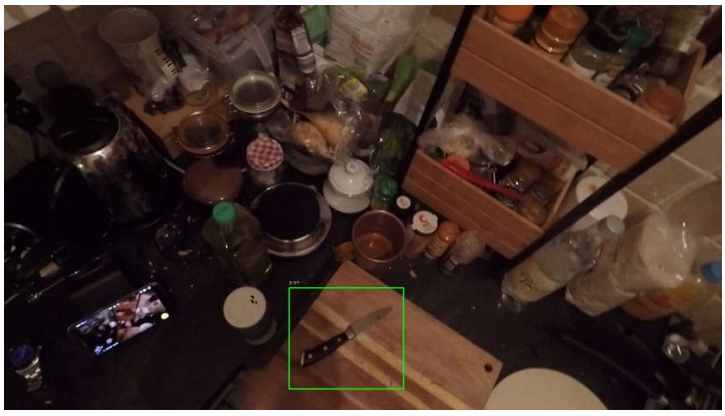

Figure 37. An example image where a knife was annotated as a frying pan

6. As detailed in Chapter 6, the implementations of RetinaNet and Faster R-CNN largely differ even though they are comparable in a single object class. Their implementations differ in the size of dataset used, the number of kitchen environments and object classes included in training and evaluation, and training time and style. Therefore, comparison of the two models can be criticised.

7. Cloud services provide easy access to multi-GPU training; however, these were not considered for this project even though the Keras RetinaNet implementation allows multi-GPU training with an additional argument that can be included in the training script.

8. Unlike suggested by Y. Bengio [80], learning rate was not the highest possible and gradually decreasing in training, instead it started with the default learning rate supplied by the Keras RetinaNet implementation.

7.3 Future Work

This section refers to the previous, Critical Evaluation section and proposes potential extensions to this work that could potentially overcome the limitations detailed above.

A natural future direction for this work would be to expand the number of object classes used to train the network. Training on more than one object classes would give valuable insight into the performance of RetinaNet. It would be interesting to see how the performance of RetinaNet changes when more object classes have to be learned. Given enough computing power, RetinaNet could be trained using all 352 object classes from EPIC-KITCHENS. According to [38], RetinaNet with ResNeXt-101-FPN backbone is able to outperform Faster R-CNN in small object classes on MS COCO [35]. It would be useful to see how RetinaNet compares to Faster R-CNN in detecting small and medium sized classes like knife and spoon which were challenging to Faster R-CNN on EPIC-KITCHENS.

Similarly, a future direction could be to experiment with the size of the dataset and the number of kitchen environments included in training.

More GPU power is a requirement for both increased number of object classes and dataset size, therefore it would be reasonable to explore different cloud solutions for multi-GPU training.

Using other forms of data augmentation could be a good way to further expand the size of the dataset and make the learning process more robust. Keras RetinaNet implementation has a random transform option that randomly rotates, flips, translates, scales and shears images providing a richer dataset for training.

Given more computing power, grid search would be a more scientific approach in hyperparameter optimization instead of random search. Even though grid search produces many weak models, it is easier to replicate and understand. Also, more computing power would allow to train many models in parallel instead of babysitting a single one.

This work could be extended by training and evaluating all networks investigated in Section 2.3 and comparing their results to each other as well as to their own performance on MS COCO.

Given more time, it would have been interesting to examine the dataset as well as the implementation from the aspect of egocentric challenges like moving cameras, disappearing and reappearing objects, blurring objects and central nature.

References

[1] D. Damen *et al.*, “Scaling Egocentric Vision: The EPIC-KITCHENS Dataset,” no. April, pp. 1–12, 2018.

[2] Deeplearning.ai, “Convolutional Neural Networks.” [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks. [Accessed: 29-Aug-2018].

[3] Cisco, “Cisco Visual Networking Index: Forecast and Methodology, 2016–2021,” 2017.

[4] Statista, “Snapchat: number of daily snaps created 2017 | Statistic,” 2018. [Online]. Available: https://www.statista.com/statistics/257128/number-of-photo-messages-sent-by-snapchat-users-every-day/. [Accessed: 20-Aug-2018].

[5] Statista, “YouTube: hours of video uploaded every minute 2015 | Statistic,” 2018. [Online]. Available: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/. [Accessed: 20-Aug-2018].

[6] W. Cueva, F. Munoz, G. Vasquez, and G. Delgado., “Detection of skin cancer ‘Melanoma’ through computer vision,” in *Proceedings of the 2017 IEEE 24th International Congress on Electronics, Electrical Engineering and Computing, INTERCON 2017*, 2017.

[7] C. Liu, H. Gomez, S. Narasimhan, A. Dubrawski, M. Pinsky, and B. Zuckerbraun, “Real-time visual analysis of microvascular blood flow for critical care,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] S. Kumar, P. Singhal, and V. Krovi, “Computer-Vision-Based Decision Support in Surgical Robotics,” *IEEE Des. Test*, vol. 32, no. 5, pp. 89–97, 2015.

[9] Mathworks.com, “Image Types - MATLAB Documentation.” [Online]. Available: https://www.mathworks.com/help/matlab/creating_plots/image-types.html. [Accessed: 20-Aug-2018].

[10] F. Li, J. Johnson, and S. Yeung, “CS231 Lecture 11 : Detection and Segmentation,” 2017.

[11] A. Betancourt, P. Morerio, C. S. Regazzoni, and M. Rauterberg, “The Evolution of First Person Vision Methods : A Survey,” *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1–17, 2015.

[12] D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperPoint: Self-Supervised Interest Point Detection and Description,” in *CVPR 2018 Deep Learning for Visual SLAM Workshop (DL4VSLAM2018)*, 2018.

[13] H. Liang, J. Yuan, D. Thalmann, and N. M. Thalmann, “AR in Hand: Egocentric Palm Pose Tracking and Gesture Recognition for Augmented Reality Applications,” *Proc. 23rd ACM Int. Conf. Multimed. - MM ’15*, no. October, pp. 743–744, 2015.

[14] H. Tilak and R. Cheng-yue, “Egocentric Hand Tracking for Virtual Reality,” 2007.

[15] Georgia Tech, “Extended GTEA Gaze+,” 2018. .

[16] A. Fathi, Y. Li, and J. Rehg, “Learning to recognize daily actions using gaze,” in *European Conference on Computer Vision*, 2012.

[17] D. Damen, T. Leelasawassuk, O. Haines, A. Calway, and W. Mayol-Cuevas, “You-do, I-learn: Discovering task relevant objects and their modes of interaction from multi-user egocentric video,” in *British Machine Vision Conference (BMVC)*, 2014.

[18] S. Bambach, “A Survey on Recent Advances of Computer Vision Algorithms for Egocentric Video,” 2015.

[19] X. Ren, M. Philipose, and Xiaofengren, “Egocentric recognition of handled objects: Benchmark and analysis,” *2009 IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2009*, pp. 49–56, 2009.

[20] D. G. Lowe, “Distinctive image features from scale invariant keypoints,” *Int. J. Comput. Vis.*, vol. 60, pp. 91–11020042, 2004.

[21] X. Ren and C. Gu, “Figure-ground segmentation improves handled object recognition in egocentric video,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, no. 1, pp. 3137–3144, 2010.

[22] N. Dalal and W. Triggs, “Histograms of Oriented Gradients for Human Detection,” *2005 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. CVPR05*, vol. 1, no. 3, pp. 886–893, 2004.

[23] A. Fathi, X. Ren, and J. M. Rehg, “Learning to recognize objects in egocentric activities,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3281–3288, 2011.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[25] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., “ImageNet Classification with Deep Convolutional Neural Networks,” *Adv. Neural Inf. Process. Syst. 25*, pp. 1–9, 2012.

[26] M. Ma, H. Fan, and K. M. Kitani, “Going Deeper into First-Person Activity Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] N. Rhinehart and K. M. Kitani, “Learning Action Maps of Large Environments via First-Person Vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[28] H. S. Park, J.-J. Hwang, Y. Niu, and J. Shi, “Egocentric Future Localization,” *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 4697–4705, 2016.

[29] H. S. Park, J.-J. Hwang, and J. Shi, “Force from Motion: Decoding Physical Sensation in a First Person Video,” *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3834–3842, 2016.

[30] Z. Ye, Y. Li, Y. Liu, C. Bridges, A. Rozga, and J. M. Rehg, “Detecting bids for eye contact using a wearable camera,” *2015 11th IEEE Int. Conf. Work. Autom. Face Gesture Recognition, FG 2015*, 2015.

[31] H. S. Park and J. Shi, “Social saliency prediction,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07–12–June, pp. 4777–4785, 2015.

[32] Y. Hoshen and S. Peleg, “An Egocentric Look at Video Photographer Identity,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016–Decem, pp. 4284–4292, 2016.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[34] M. Everingham *et al.*, “The PASCAL Visual Object Classes (VOC) Challenge,” *Int. J. Comput. Vis.*

[35] T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS,

no. PART 5, pp. 740–755, 2014.

[36] H. Pirsiavash and D. Ramanan, “Detecting activities of daily living in first-person camera views,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[37] F. D. La Torre *et al.*, “Guide to the Carnegie Mellon University Multimodal Activity (CMU-MMAC) database,” in *Robotics Institute*, 2008.

[38] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017–Octob, pp. 2999–3007, 2017.

[39] GitHub, “An implementation of RetinaNet in PyTorch.” [Online]. Available: https://github.com/c0nn3r/RetinaNet. [Accessed: 21-Aug-2018].

[40] GitHub, “RetinaNet on Cloud TPU.” [Online]. Available: https://github.com/tensorflow/tpu/tree/master/models/official/retinanet. [Accessed: 21-Aug-2018].

[41] GitHub, “Keras implementation of RetinaNet object detection.” [Online]. Available: https://github.com/fizyr/keras-retinanet. [Accessed: 21-Aug-2018].

[42] IndoML.com, “Student Notes: Convolutional Neural Networks (CNN) Introduction.” [Online]. Available: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/. [Accessed: 29-Aug-2018].

[43] I. J. V. Walker, M. P. Deisenroth, and A. Faisal, “Deep Convolutional Neural Networks for Brain Computer Interface using Motor Imagery,” 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 770–778, 2016.

[45] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *2015 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 1–9, 2014.

[46] C. Jin, H. Sun, and S. Kimura, “Sparse ternary connect: Convolutional neural networks using ternarized weights with enhanced sparsity,” *2018 23rd Asia South Pacific Des. Autom. Conf.*, pp. 190–195, 2018.

[47] “Object Localization and Detection.” [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html. [Accessed: 29-Aug-2018].

[48] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” in *International Conference on Learning Representations (ICLR)*, 2013.

[49] M. Cavaioni, “DeepLearning series: Objection detection and localization — YOLO algorithm, R-CNN.” [Online]. Available: https://medium.com/machine-learning-bites/deeplearning-series-objection-detection-and-localization-yolo-algorithm-r-cnn-71d4dfd07d5f. [Accessed: 29-Aug-2018].

[50] Stack Overflow, “YOLO object detection: how does the algorithm predict bounding boxes larger than a grid cell?” [Online]. Available: https://stackoverflow.com/questions/50575301/yolo-object-detection-how-does-the-algorithm-predict-bounding-boxes-larger-than/50576985. [Accessed: 31-Jul-2018].

[51] W. Liu *et al.*, “SSD: Single shot multibox detector,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016.

[52] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-

Time Object Detection," 2015.

[53] "Intro to Object Detection." [Online]. Available: https://lovesnowbest.site/2018/02/27/Intro-to-Object-Detection/. [Accessed: 31-Jul-2018].

[54] R. Girshick, P. Felzenszwalb, and D. Ramanan, "Non-maximum suppression," 2007. [Online]. Available: https://github.com/rbgirshick/voc-dpm/blob/master/test/nms.m. [Accessed: 22-Jul-2018].

[55] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 6517–6525, 2017.

[56] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, 2013.

[57] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

[58] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," 2016.

[59] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014.

[60] R. Girshick, "Fast R-CNN," *2015 Int. Conf. Comput. Vis.*, pp. 1440–1448.

[61] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," pp. 1–14, 2014.

[62] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[63] D. M. and D. R. Pedro F. Felzenszwalb, Ross B. Girshick, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010.

[64] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 936–944, 2017.

[65] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "DSSD : Deconvolutional Single Shot Detector," 2017.

[66] K. Lee, J. Choi, J. Jeong, and N. Kwak, "Residual Features and Unified Prediction Network for Single Stage Detection," 2017.

[67] J. Ren *et al.*, "Accurate single stage detector using recurrent rolling convolution," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 752–760, 2017.

[68] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta, "Beyond skip connections: Top-down modulation for object detection," 2016.

[69] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[70] A. Douillard, "Object Detection with Deep Learning on Aerial Imagery," 2018. [Online]. Available: https://medium.com/data-from-the-trenches/object-detection-with-deep-%0Alearning-on-aerial-imagery-2465078db8a9%0A. [Accessed: 13-Jul-2018].

[71] S. Fortmann-Roe, “Understanding the Bias-Variance Tradeoff,” 2012. [Online]. Available: http://scott.fortmann-roe.com/docs/BiasVariance.html. [Accessed: 30-Aug-2018].

[72] S. Mallick, “Bias-Variance Tradeoff in Machine Learning,” 2017. [Online]. Available: https://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning/. [Accessed: 30-Aug-2018].

[73] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[74] Deeplearning.ai, “Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.” [Online]. Available: https://www.coursera.org/learn/deep-neural-network. [Accessed: 12-Jul-2018].

[75] D. Lysukhin, “TensorFlow Object Detection API: basics of detection (1/2),” *Becoming Human: Artificial Intelligence Magazine*, 2018. [Online]. Available: https://becominghuman.ai/tensorflow-object-detection-api-basics-of-detection-7b134d689c75. [Accessed: 23-Jul-2018].

[76] D. Damen *et al.*, “EPIC Kitchens Dataset,” *GitHub*, 2018. [Online]. Available: https://github.com/epic-kitchens/annotations. [Accessed: 23-Jul-2018].

[77] J. Brownlee, “A Gentle Introduction to Transfer Learning for Deep Learning,” *Machine Learning Mastery*, 2017. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. [Accessed: 28-Jul-2018].

[78] “Releases,” *GitHub*. [Online]. Available: https://github.com/fizyr/keras-retinanet/releases. [Accessed: 28-Jul-2018].

[79] D. P. Kingma and J. L. Ba, “Adam: A Method for Stochastic Optimization,” in *ICLR 2015*, 2015.

[80] Y. Bengio, “Practical Recommendations for Gradient-Based Training of Deep Architectures,” 2012.

[81] A. Karpathy, “CS231n Convolutional Neural Networks for Visual Recognition,” *cs231n.github.io*. [Online]. Available: http://cs231n.github.io/neural-networks-3/. [Accessed: 01-Aug-2018].

[82] “Training time - Faster R-CNN · Issue #8 · epic-kitchens/annotations,” *GitHub*, 2018. [Online]. Available: https://github.com/epic-kitchens/annotations/issues/8. [Accessed: 14-Aug-2018].

Appendix 1 – Training scripts

```
!python3 keras_retinanet/bin/train.py
    --weights snapshots/resnet50_coco_best_v2.1.0.h5
    --batch-size 4
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --epochs 3
    --steps 535
    --snapshot-path snapshots/experiment_01_gpu
    --freeze-backbone
    --image-min-side 600
    csv data/annotations_train.csv data/classes.csv
    --val-annotations data/annotations_val.csv


!python3 keras_retinanet/bin/train.py
    --snapshot snapshots/experiment_01_gpu/resnet50_csv_03.h5
    --batch-size 2
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --epochs 4
    --steps 1069
    --snapshot-path snapshots/experiment_02_gpu
    --freeze-backbone
    --image-min-side 600
    csv data/annotations_train.csv data/classes.csv
    --val-annotations data/annotations_val.csv


!python3 keras_retinanet/bin/train.py
    --snapshot snapshots/experiment_02_gpu/resnet50_csv_04.h5
    --batch-size 4
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --epochs 4
    --steps 1069
    --snapshot-path snapshots/experiment_03_gpu
    --freeze-backbone
    --image-min-side 600
    csv data/annotations_train.csv data/classes.csv
    --val-annotations data/annotations_val.csv


!python3 keras_retinanet/bin/train.py
    --snapshot snapshots/experiment_03_gpu/resnet50_csv_02.h5
    --batch-size 2
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --epochs 1
    --steps 1069
    --snapshot-path snapshots/experiment_04_gpu
    --freeze-backbone
    --image-min-side 600
    csv data/annotations_train.csv data/classes.csv
    --val-annotations data/annotations_val.csv
```

Appendix 2 – Evaluation scrips

```
!python3 keras_retinanet/bin/evaluate.py
    --convert-model
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --save-path results/s1/IoU05
    --iou-threshold 0.5
    --image-min-side 600
    csv data/annotations_s1.csv data/classes.csv
    snapshots/experiment_03_gpu/resnet50_csv_02.h5


!python3 keras_retinanet/bin/evaluate.py
    --convert-model
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --save-path results/s1/IoU005
    --iou-threshold 0.05
    --image-min-side 600
    csv data/annotations_s1.csv data/classes.csv
    snapshots/experiment_03_gpu/resnet50_csv_02.h5


!python3 keras_retinanet/bin/evaluate.py
    --convert-model
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --save-path results/s1/IoU075
    --iou-threshold 0.75
    --image-min-side 600
    csv data/annotations_s1.csv data/classes.csv
    snapshots/experiment_03_gpu/resnet50_csv_02.h5


!python3 keras_retinanet/bin/evaluate.py
    --convert-model
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --save-path results/s2/IoU05
    --iou-threshold 0.5
    --image-min-side 600
    csv data/annotations_s2.csv data/classes.csv
    snapshots/experiment_03_gpu/resnet50_csv_02.h5


!python3 keras_retinanet/bin/evaluate.py
    --convert-model
    --gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
    --save-path results/s2/IoU005
    --iou-threshold 0.05
    --image-min-side 600
    csv data/annotations_s2.csv data/classes.csv
    snapshots/experiment_03_gpu/resnet50_csv_02.h5


!python3 keras_retinanet/bin/evaluate.py
```

```
--convert-model
--gpu GPU-6956aa44-5538-7062-ae50-a15af65b4db3
--save-path results/s1/IoU075
--iou-threshold 0.75
--image-min-side 600
csv data/annotations_s2.csv data/classes.csv
snapshots/experiment_03_gpu/resnet50_csv_02.h5
```

Appendix 3 – Convert training model to inference model

```
!python3 keras_retinanet/bin/convert_model.py
    snapshots/experiment_03_gpu/resnet50_csv_02.h5
    snapshots/inference_model/best_model.h5
```