Escuela
Politécnica
Superior

Interaction estimation in egocentric videos via simultaneous hand-object recognition

Bachelor's Degree in Computer
Engineering

Bachelor's Thesis

Author:

Manuel Benavent-Lledó

Supervisors:

José García-Rodríguez

Sergiu Oprea

June 2021

Universitat d'Alacant
Universidad de Alicante

Interaction estimation in egocentric videos via simultaneous hand-object recognition

Author

Manuel Benavent-Lledó

Supervisors

José García-Rodríguez

Sergiu Oprea

Department of Information Technologies and Computing

Bachelor's Degree in Computer Engineering

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, June 2021

Abstract

In this project, we propose an action estimation pipeline based on the simultaneous recognition of the hands and the objects in the scene from an egocentric perspective. To this end, research on the state of the art in this field has been conducted. From the latest approaches, we have come to the conclusion that the hands are a key element from this point of view. An action consists of the interactions of the hands with the different objects in the scene therefore, the 2D positions of hands and objects are used to compute the object which is more likely to be used.

The architecture used for achieving this goal is YOLO, as its prediction speed allows us to predict the actions fluently with good accuracy on the detected objects and hands. After reviewing the available datasets and generators for hand and object detection, different experiments have been conducted. The best results determined by PascalVOC metric have been used in the proposed pipeline.

Resumen

En este proyecto proponemos una arquitectura de estimación de acciones basado en el reconocimiento y localización simultánea de manos y objetos en primera persona. Para ello, se ha realizado un estudio del estado del arte, a partir del cual hemos concluido que las manos son un elemento clave desde esta perspectiva. En nuestro caso de estudio, una acción consiste en las interacciones de las manos con los diferentes objetos en la escena, por lo tanto, las posiciones 2D de las manos y los objetos se utilizan para calcular el objeto que tiene más probabilidades de ser utilizado. Además, se muestran las diferentes acciones que se pueden realizar con los objetos más cercanos a las manos.

La arquitectura utilizada para lograr este objetivo es YOLO, debido a que su velocidad de predicción nos permite predecir las acciones con fluidez y buena precisión sobre los objetos y manos detectados. Después de revisar los conjuntos de datos y generadores disponibles para la detección de objetos y manos, se han realizado diferentes experimentos. La configuración del sistema con mejores resultados, determinados por la métrica PascalVOC, ha sido utilizada en el pipeline propuesto.

Acknowledgements

I would like to thank my Thesis advisors, Jose García-Rodríguez and Sergiu Oprea, as well as the rest of 3D Perception Lab team, Pablo Martínez-González and John Alejandro Castro-Vargas, for their motivation and guidance during the development of this Thesis.

Also, to my family and friends for their support during this Bachelor's Degree.

Contents

List of Figures	xiii
List of Tables	xv
List of Acronyms	xvii
1. Introduction	1
1.1. Overview	1
1.2. Motivation	1
1.3. Proposal and Goals	2
1.4. Schedule	2
1.4.1. Learning stage: September - January	2
1.4.2. Setup stage: January - Mid February	3
1.4.3. Experimentation stage: February - June	3
1.5. Outline	4
2. State of the art	5
2.1. Introduction	5
2.2. Datasets	5
2.2.1. EPIC-KITCHENS	6
2.2.2. ADL Dataset	7
2.2.3. EgoDaily	7
2.2.4. EGTEA Gaze+	8
2.2.5. EgoRoutine	9
2.2.6. EgoHands	9
2.2.7. The MECCANO dataset	9
2.2.8. THU-READ	10
2.2.9. Summary	10
2.3. Object and hand recognition approaches	11
2.3.1. Object Detection methods	13
2.3.1.1. YOLO: You Only Look Once	13
2.3.1.1.1. YOLOv4	14
2.3.1.2. Other architectures	15
3. Materials and Methods	17
3.1. Introduction	17
3.2. Hardware	17
3.3. Docker	18

3.4. Frameworks	19
3.4.1. TensorFlow	19
3.4.2. Keras	20
3.4.3. PyTorch	20
3.4.4. Darknet	21
4. Proposal	23
4.1. Introduction	23
4.2. Architecture: YOLOv3	23
4.3. Datasets	25
4.3.1. Hands dataset: EgoDaily	25
4.3.1.1. Hand recognition and transfer learning	26
4.3.2. Object dataset: EPIC-KITCHENS	26
4.4. Recognition pipeline	27
5. Experiments and Results	31
5.1. Introduction: training and evaluation	31
5.1.1. Training	31
5.1.2. Evaluation	31
5.1.2.1. PascalVOC evaluation metric	31
5.1.2.2. COCO evaluation metric	32
5.2. Hand Detection	32
5.2.1. Hand disambiguation	33
5.2.1.1. Model from scratch	33
5.2.1.2. Transfer learning model	33
5.2.2. Generic hand recognition	34
5.2.2.1. EfficientNet backbone	34
5.2.2.2. Darknet backbone	35
5.3. Object Detection	35
5.3.0.1. Darknet backbone	36
5.3.0.2. EfficientNet backbone	37
5.3.0.3. CSPDarknet53: YOLOv4	37
5.4. Interaction estimation results	38
6. Conclusions	43
6.1. Conclusions	43
6.2. Future Works	43
Bibliography	45
A. Appendix I: Pipeline results	49

List of Figures

1.1. Schedule summary.	3
2.1. EPIC-KITCHENS dataset sample.	6
2.2. ADL dataset sample.	7
2.3. Frame samples from EgoDaily dataset.	8
2.4. EGTEA Gaze+ dataset sample.	8
2.5. EgoRoutine dataset.	9
2.6. EgoHands sample frames.	10
2.7. Meccano dataset frames with object bounding boxes and action labels.	10
2.8. THU-READ RGB frames on the top with its depth image at the bottom.	11
2.9. Object detection with scene identification filtering.	12
2.10. Overview of the egocentric action prediction framework EgoACO – Egocentric Action-Context-Object Net.	12
2.11. YOLO architecture from [25].	13
2.12. YOLO pipeline and evaluation.	14
2.13. Object detector.	14
3.1. Asimov’s hardware specifications	18
3.2. Docker Containers vs. Virtual Machines	19
3.3. PyTorch tensors (equivalent to NumPy ndarrays).	20
4.1. Convolutional Neural Network (CNN)	24
4.2. YOLOv3 architecture.	25
4.3. EPIC-KITCHENS selected classes with the total number of annotations.	28
4.4. Proposed recognition pipeline.	29
5.1. EgoDaily ground-truth with hand disambiguation.	33
5.2. YOLOv3 evaluation results when trained from scratch with EgoDaily.	34
5.3. YOLOv3 evaluation results when trained from Darknet53 with EgoDaily.	35
5.4. YOLOv3 evaluation results when trained from EfficientNet with EgoDaily (without hand disambiguation).	36
5.5. YOLOv3 evaluation results when trained from crosshands with EgoDaily (with- out hand disambiguation).	37
5.6. EPIC-KITCHENS ground-truth.	38
5.7. YOLOv3 evaluation results with Darknet53 backbone when trained using a downsampled version of EPIC-KITCHENS dataset.	39
5.8. YOLOv3 evaluation results with EfficientNet backbone when trained using a downsampled version of EPIC-KITCHENS dataset.	41
5.9. YOLOv4 evaluation results with CSPDarknet53 backbone when trained using a downsampled version of EPIC-KITCHENS dataset.	42

A.1. Results when 2 hands are detected and 2 or more objects are detected.	49
A.2. Results when 1 hand is detected and 2 or more objects are detected.	50
A.3. Results when no hand or object is detected.	51

List of Tables

2.1. Dataset summary	11
3.1. NAS specification	19
4.1. Epic-Kitchens train object labels.	27
5.1. Train arguments.	40

List of Acronyms

ADL	Activities of Daily Living.
AP	Average Precision.
API	Application Program Interface.
AR	Average Recall.
ASIC	Application-Specific Integrated Circuit.
CNN	Convolutional Neural Network.
CNTK	Microsoft Cognitive Toolkit.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
DPM	Deformable Parts Model.
DTIC	Departamento de Tecnología Informática y Computación.
EGTEA	Extended Georgia Tech Egocentric Activity.
FAIR	Facebook Artificial Intelligence Group.
GPU	Graphics Processing Unit.
IOU	Intersection Over Union.
LSTM	Long Short-Term Memory.
mAP	mean Average Precision.
mAR	mean Average Recall.
MLP	Multilayer perceptron.
MoDeAsS	Monitoring and Detection of human behaviors for personalized assistance and early disease detection, A2HUMPA in Spanish.
NAS	Network Attached Storage.
OOP	Object Oriented Programming.
RAID	Redundant Array of Independent Disks.
RPN	Region Proposal Network.
SSH	Secure Shell.
TPU	Tensor Processing Unit.
VMs	Virtual Machines.
YOLO	You Only Look Once.

1. Introduction

This first chapter introduces the main topic of this Thesis. It is divided into 5 sections as follows. First, Section 1.1 synthesizes the work carried out during this Thesis. In Section 1.2 the motivation of this work is explained, followed by Section 1.3 where the goals of this work are detailed. Thesis' timeline is presented in Section 1.4 and finally, Section 1.5 details the structure of the Thesis.

1.1. Overview

This Bachelor's Thesis presents the research and experiments carried out for the task of plausible interaction estimation in RGB egocentric videos from simultaneous objects and hands recognition. We aim to determine a set of possible actions that a person could perform with an object in a given scenario. To this end, deep learning approaches and datasets for this task have been revised. From the most relevant ones, a pipeline is proposed to estimate actions from the 2D relative positions of objects regarding to the hands. Finally, conclusions are extracted and the future lines of work introduced.

1.2. Motivation

This Thesis represents the knowledge learned during the four years (September 2017 - June 2021) of Bachelor's Degree in Computer Engineering at the University of Alicante. The motivation of this work comes from different interests acquired in the past years.

On the one hand, the collaboration with the Department of Information Technologies and Computing (in Spanish Departamento de Tecnología Informática y Computación (DTIC)), specially with 3D Perception Lab group whose main research topics are Deep Learning, 3D Computer Vision and GPU computing. In particular, the project Monitoring and Detection of human behaviors for personalized assistance and early disease detection, A2HUMPA in Spanish (MoDeAsS), funded by the Spanish Government and whose main researchers are Jose Garcia-Rodriguez and Miguel Angel Cazorla-Quevedo, both professors at the University of Alicante. In this project we aim to provide personalized assistance to dependent people, such as the elderly ones. By identifying changes in their normal behaviour, anomalous situations (*e.g.* memory loss or disorientation) can be detected and the quality of life of patients improved. This project does not focus in a specific group of people, in fact, all the developments in this project can be applied to numerous groups of people with different diseases. For example, declination of cognitive abilities caused by dementia, assistance during recovery after an accident or even early disease detection in apparently healthy people.

On the other hand, from a personal perspective, computer vision has always caught my attention since my interest in computer science started due to its many applications in mul-

multiple areas, specially robotics. Hence, this is an excellent opportunity to learn some of the latest methods and datasets used in this field, alongside the research process.

1.3. Proposal and Goals

The main goal of this Thesis is to develop a system capable of inferring activities from the simultaneous recognition of objects and hands. An egocentric point of view is used as it provides a close perspective to the way a person perceives the surrounding scene. The intention is to estimate the possible actions that could be performed by dependant people with brain injuries or under-recovery from an accident. After analyzing the latest approaches and datasets, the most promising ones will be used to train and evaluate deep learning models. Furthermore, we will empirically determine the best parameters for a robust object and hand recognition. The most promising results will be used to present a pipeline that estimates the interactions between hands and objects.

1.4. Schedule

The development of this project to accomplish the previous goals has been distributed during the 9 months an academic year takes. The schedule has been divided into 3 different stages from late September 2019 until the 3rd of June, the submission date for this Thesis. Some tasks will be distributed along the whole project development, while others need to be finished before starting the next stage. Figure 1.1 shows the schedule summary distributed along the different phases. Figure 1.1a shows the Gantt's diagram where the precedence order of the tasks is detailed, whilst figure 1.1b shows the timeline followed.

1.4.1. Learning stage: September - January

The first stage in the process consists of a formative period in which the fulfilment of 2 tasks is necessary before starting the second phase:

- State of the art: every week some of the latest related papers recommended by the supervisors are read and summarized, so as to determine an appropriate architecture and datasets to train the object recognition models. Then, the summary of each paper is used as a reference to write the state of the art of this Thesis.
- Materials: there are several materials needed for the development of the experiments that will be described afterwards, in this phase a deep learning framework is chosen after reviewing some of the most popular ones. Tutorials and guidelines from the supervisors for the necessary materials are used to improve the skills, so the lack of them does not become a bottleneck in the development of the project.

During this stage, this document is started and some of the chapters are completed, this task is not required before the start of the next stages and therefore it will be completed alongside other tasks.

(a) Gantt's Diagram.

(b) Timeline.

Figure 1.1: Schedule summary.

1.4.2. Setup stage: January - Mid February

After acquiring the necessary knowledge and before the start of the final stage, some time is spent to prepare the workspace with the materials, datasets and implementations for the selected architecture. Also, the annotations are normalized so they fit the required training format. With this information a proposal for experimentation in the next phase is detailed.

1.4.3. Experimentation stage: February - June

In order to obtain representative results, different experiments have been carried out to evaluate the capabilities of our proposal. This experimentation has been performed from the previous analysis and theoretical proposal we have made. We have evaluated different deep learning models whose characteristics are adjusted by modifying the architecture, training data and/or hyper-parameters in order to obtain the model that best approximates our

problem.

1.5. Outline

This Thesis consists of 6 chapters: chapter 1 introduces the project and its motivations with the main goal to be achieved, as well as the planning details followed to achieve it. In chapter 2, the state of the art datasets and approaches in the object recognition field are reviewed in order to choose some of them for experimentation. Chapter 3 describes the different materials used for the development of the project. From the state of the art review, a proposal is made in chapter 4 and the experiments carried out with their results are detailed in chapter 5. Finally, in chapter 6 conclusions from the work described in previous chapters are extracted, alongside the future lines of work.

2. State of the art

This chapter presents the state of the art in egocentric human behaviour recognition and object detection. Section 2.1 contains an introduction to this topic, followed by Section 2.2 presenting most used datasets. Finally, in Section 2.3 most related approaches to this work are reviewed.

2.1. Introduction

During the last years, researchers have been developing different approaches for the action recognition problem based on various types of data sources. The first person perspective is obtained from a forward-facing camera placed on the chest or the head of a person. This egocentric point of view contains relevant information such as interactive objects in the foreground, and frequently the camera wearer’s hands. Some approaches analyze the video sequences whereas others are based on detecting the elements in front of the person. This refers to the surrounding area and its contents, usually consisting of objects, hands, other people and the scene background. To this end, a set of new challenges arise such as: (1) the quantification of interactions between the hands, (2) their proximity and relation with objects and (3) the scene identification. In the following sections, datasets designed for this purpose are presented as well as different deep learning-based approaches.

2.2. Datasets

Deep learning-based approaches are well known to be data-hungry. This has motivated researchers to focus on large-scale datasets and synthetic data generators. Some of the latest generators in the action recognition field are ElderSim [14] and VirtualHome [22]. The first one is aimed to be used by robots to help elderly people. The second one allows us to reproduce daily actions by specifying the atomic actions. Although synthetic datasets provide full annotations at a large scale, they are not representative of the underlying distribution of real-world data. First, it is challenging, if not impossible, to simulate real-world properties in a synthetic environment. Second, a deep model trained exclusively on synthetic data exhibits generalization issues in the real domain. This is due to the inter-domain distribution discrepancy between synthetic and real data, fact addressed in the context of domain adaptation. Despite this drawback, they are still a good option to test and propose new solutions for the problem to which they are oriented. From the problem present in most generators, their lack of realism, emerges UnrealROX [20]. An alternative to the generators previously mentioned, which addresses this situation by using Unreal Engine 4 to generate photo realistic datasets for a wide variety of applications (semantic segmentation, object detection, visual grasping...). We can also find some datasets whose actions have been performed by different

actors such as the Ikea dataset [2], MMAct [17], Toyota Smarthome Untrimmed [5] and Toyota Smarthome [8], all of them used in the action recognition field. However, since they only provide third person camera views they are just worth mentioning in this Thesis. From this point, several egocentric action recognition datasets, based on the concept of recognizing the surrounding elements, will be reviewed.

2.2.1. EPIC-KITCHENS

The Epic-Kitchens dataset [7] addresses the action recognition and object detection problem. With 55 hours of RGB video, it contains people recording themselves with only one person in the frame at all times performing fine-grained actions in a kitchen, *e.g. put, take, open, and close various objects*. This dataset contains annotations of activity segments and generic object bounding boxes for each of the over 11.5 millions of frames. The annotations of this first version, that was the biggest dataset for action recognition at the time, contains 39596 action segments classified in 149 classes in the field of action recognition and 454158 bounding boxes classifying 323 different types of objects. An example of the video information and its labels can be observed in figure 2.1.

An extension of this dataset was included in their 2020 version [6], with 100 recording hours, 20 million frames and 90000 actions belonging to 700 variable-length videos captured in 45 different environments. The annotations have been improved from the previous version (54% more actions per minute and 128% more action segments) and it benchmarks action recognition, weakly supervised action recognition, action detection, action anticipation, unsupervised domain adaptation and multi-instance retrieval. Moreover, automatic annotations for hands and objects are provided in form of masks and bounding boxes.

Figure 2.1: EPIC-KITCHENS dataset sample.

On the top we can see the frames, with the action timeline in the middle and the objects detected on the bottom.

2.2.2. ADL Dataset

The ADL dataset [21] was presented for medical applications purposes such as rehabilitation or life-logging tasks for memory loss. It also focuses on object detection and action recognition from an egocentric point of view, with 18 different action recognition classes: make up, washing dishes, making coffee/tea or doing laundry are some of them. The most remarkable characteristics of this dataset is that it includes bounding boxes for hands as we can see in figure 2.2, unlike EPIC-KITCHENS 100 [6], no masks or binary data is used in this case. And it includes some long-scale temporal actions, like *making coffee*, that can take several minutes in the field of action recognition, and some complex object interactions in the field of object detection, *e.g. difference between fridge open or closed (active or inactive)*.

Figure 2.2: ADL dataset sample.
Video frames including bounding boxes for objects and hands.

2.2.3. EgoDaily

The EgoDaily dataset [4] is presented as an alternative for some other egocentric datasets focusing on detecting the hands. It contains 50 thousand 1080p images, with over 95.000 hand annotations recorded by 10 different subjects from different gender and cultural ethnicity in order to add variability, some sample frames can be observed in figure 2.3. Frames belong to 5 different activities (biking, eating, kitchen, office and running) recorded at different locations, on top of that, the annotations distinguish between left and right hand. This dataset emerges from the lack of an egocentric dataset for hand disambiguation since for most of them, either it is not the main objective, the environments are restricted or their small quantity of hand annotations causes the necessity to include other datasets to complete the training. That is the case for EPIC-KITCHENS [7] [6], ADL dataset [21], EDSH [18] or EGTEA [19].

Figure 2.3: Frame samples from EgoDaily dataset.

2.2.4. EGTEA Gaze+

The Extended Georgia Tech Egocentric Activity (EGTEA) Gaze+ dataset [19] is a larger version of its predecessors GTEA, GTEA Gaze and GTEA Gaze+, it is composed of 28 hours of HD videos (1280x960) recorded by 32 different subjects in 86 different sessions of cooking activities. The authors aim to use the information from the gaze, together with a head-mounted camera to determine the areas where the user is centering the most of its attention to, in order to determine fine-grained actions, such as "Cut bell pepper" or "Pour condiment (from) condiment container into salad". Action (human-object interactions) and hand masks are provided for the videos, audios and gaze tracking. In figure 2.4, we can observe the importance of the gaze from some frames extracted from this dataset.

Figure 2.4: EGTEA Gaze+ dataset sample.

For each of the action types, four rows of boxes are shown, each selected from one instance of the action. The actions are (a) spread peanut-butter on bread using knife, (b) scoop jam using knife and (c) close milk. Since the information is not equally relevant in all subareas, the information from both the hands and the gaze can help determine the ones with higher bearing.

2.2.5. EgoRoutine

The authors of EgoRoutine [30] present an egocentric dataset describing the daily life of the camera wearers with a novel automatic discovery of routine days. Routine is defined as the usual things that a person does daily and helps describe the individual's lifestyle. The proposal of this dataset is to determine the routines followed by a user on a daily basis, since each person has a different routine, behaviour patterns are detected. From the alteration of routine actions, health problems in Activities of Daily Living (ADL) can be detected. It is composed of a total of 100.000 images, from 104 days recorded by 7 different users. Figure 2.5 shows some sample frames from the recordings.

Figure 2.5: EgoRoutine dataset.

2.2.6. EgoHands

The proposed dataset called “EgoHands” [1] addresses the hand detection from egocentric perspective with an object detection dataset. This dataset focuses on human interaction and has two people in the video facing one another, and interacting by playing board games: playing cards, playing chess, solving a jigsaw puzzle and playing Jenga. Although some different locations and clothes were used to add variability in terms of background and illumination, video capture was static since actors never moved from the table. It contains 48 different videos with 4800 annotated frames with pixel-level masks gathering 15053 hand annotations. The recordings were captured using Google Glasses’ camera therefore, a head view is provided. This type of vision limits hand pose analysis, as most hand annotations do not belong to the wearer, as dataset sample in figure 2.6 shows. On the contrary, head movements are also recorded resulting in a more relevant scene than static chest-mounted cameras.

2.2.7. The MECCANO dataset

The MECCANO dataset [23] has been explicitly labelled for the task of recognizing human-object interactions from an egocentric perspective in an industrial-like domain. The 20 participants recorded over 299 thousand video frames while building a meccano motorbike model which were classified in 61 different action classes and 20 objects that are labelled using 64349 bounding boxes. Figure 2.7 contains a sample of this dataset whose main disadvantage is its

Figure 2.6: EgoHands sample frames.

reduced range of applications.

Figure 2.7: Meccano dataset frames with object bounding boxes and action labels.

2.2.8. THU-READ

The THU-READ dataset [32] authors provide a dataset that not only includes RGB videos that have been growing in the past years with the development of wearable cameras such as GoPro, Pivthead or Google glasses. They used a RGB-D camera mounted on a bike helmet, which provided an egocentric depth image with a focus on using the 3D structural information of the scene. The recordings from 8 subjects form this dataset with almost 2000 thousand videos with 346,626 frames, classified in 40 different action classes, separated in different modalities according to 2 hand use criteria: hand-object/non-hand-object and single-handed/double-handed, for example *bounce a ball* is one-handed and hand-object whilst *wash hands* is double-handed and non-hand-object. Figure 2.8 contains RGB-D samples from this dataset.

2.2.9. Summary

After analyzing some of the state-of-the-art datasets. A summary table is presented with their scope of applications, locations, number of different classes and the number of video frames and annotations with the ones that are more relevant in the object and hand detection task.

Figure 2.8: THU-READ RGB frames on the top with its depth image at the bottom.

Name	Type of objects	Location	Object classes	Data	Annotations
EPIC-KITCHENS	Kitchenware, food and hands	Kitchen	352	200,390 frames	389,813
ADL	Daily Activities Objects	Indoor (house)	-	10 hours of video	-
EgoDaily	Hands	Indoor and outdoor	2	Over 50,000 frames	95,000
EgoHands	Hands	Indoor and outdoor	4	4,800 frames	15,053
MECCANO	Meccano pieces	Indoor	20	299,376 frames	64,349

Table 2.1: Dataset summary

2.3. Object and hand recognition approaches

Many approaches have been developed in the action recognition field using different data sources as input (RGB videos and frames, 3D skeletons, depth images...). The recognition of objects can provide a set of possible actions that can be filtered if the scene is detected (it is not common to cook in a bedroom). Moreover, as emphasized by the authors in [16], hand detection is a key feature for activity recognition. Their position can help determine which object is more likely to be used. A hand can be detected as an object and therefore, some deep learning architectures for this task are introduced towards choosing one of them for experimentation.

As we said before, the prominent characteristic of the egocentric point of view is the more relevant information provided for the camera wearer than, for example, third person views. From the hands, objects and scene location in an image, a detection pipeline can determine the conducted action. Location could be inferred from the surrounding objects, as the method presented by the authors in [35], due to the inaccurate estimations of the scene, they present

a new pipeline which takes as an input a single image and estimates its layout and the 3D object poses. In [33], authors state that taking the temporal context into consideration can help determine the scene which implies an improvement in the objects detected. Also, a framework using Long Short-Term Memory (LSTM) is presented where no scene labelling is necessary but as we can see in figure 2.9, objects are filtered depending on the scene.

Figure 2.9: Object detection with scene identification filtering.

Finally, before introducing the object detection methods, it is worth mentioning EgoACO [29] where authors present a deep learning architecture to recognize actions on objects in egocentric videos. From class activation pooling using attention dictionaries the most relevant feature regions are used to decode object and scene descriptors. Figure 2.10 shows an overview of this prediction framework.

Figure 2.10: Overview of the egocentric action prediction framework EgoACO – Egocentric Action-Context-Object Net.

2.3.1. Object Detection methods

As we aim to detect actions from the surrounding information in a video frame, we need an architecture that is capable of detecting objects and hands. There are 2 problems in this type of pipelines that need to be solved: (1) bounding boxes for the objects need to be generated and (2) then classified. In this subsection different approaches will be reviewed based on different objectives from their authors such as detection speed or accuracy.

2.3.1.1. YOLO: You Only Look Once

YOLO is one of the most popular object recognition methods, from its first version [25] it has provided an outstanding speed compared to previous methods such as Deformable Parts Model (DPM) using sliding windows or R-CNN [12] which generates more bounding boxes slowing down the prediction result. YOLO authors developed a unified architecture of 24 layers (9 in a tiny version) with 2 fully connected layers as we can see in figure 2.11, since the whole detection pipeline is a single network is extremely fast (45 fps) and more accurate than other methods. The main disadvantage presented by this architecture is the bad detection results when small objects appear in the image since most times those are not recognized. Nevertheless, YOLO authors have made improvements in both speed and accuracy of their network and in the reduction of localization errors over the past years, as we can see in their second [26] version. In the latest version of YOLO proposed by the same authors YOLOv3 [27] detection of smaller objects has been improved at the expense of reducing the prediction speed to 29 ms with an input size of 416 x 416, 35 fps on a Titan X.

Figure 2.11: YOLO architecture from [25].

The detection pipeline (figure 2.12a) consists of dividing the image in several regions, then bounding boxes are generated in each cell of the grid. Class predictions are computed for each bounding box and the accuracy is determined using Intersection Over Union (IOU). The confidence for the bounding boxes is calculated as $Pr(Object) * IOU$, $Pr(Object)$ refers to the object class detected and IOU is the accuracy of the bounding box using Intersection

over Union which can be described as an evaluation metric that measures the area of the predicted bounding box and the real bounding box, figure 2.12b represents some examples of good and bad results using this metric.

Figure 2.12: YOLO pipeline and evaluation.

2.3.1.1.1. YOLOv4 From the contributions of the authors of the previous YOLO versions, a different group of researchers developed YOLOv4 [3]. On the basis of modern object detectors (figure 2.13), which are composed of 2 main parts: a backbone which is pre-trained on large datasets such as ImageNet, and head in which bounding boxes and classes are predicted. By combining some of the state of the art backbones and heads they have developed YOLOv4 aiming to obtain good accuracy and speed results on regular GPUs. YOLO backbones have been improved as well as their heads, they adopted Darknet 53 in YOLOv3 [27] as the results were better than Darknet 13 used in YOLO9000 [26], their second version. In YOLOv4, the backbone used is CSPDarknet53 [34] an improved version of Darknet53 using cross-state-partial-connections.

Figure 2.13: Object detector.

2.3.1.2. Other architectures

In spite of the good results provided by YOLO compared to other methods, other approaches have been reviewed. The authors of R-CNN [12] proposed an object detection pipeline combining region proposal extraction and image classification. The main disadvantage of this approach is the computation time of region proposal generation. With different architectures Fast R-CNN [10] and SPPnet [13] have improved significantly the detection speed exposing region proposal extraction as a bottleneck. From the previous improvements emerges Faster R-CNN [28] a Region Proposal Network (RPN). An end-to-end model generates region proposals which are classified with Fast R-CNN [10].

Region proposals have been reduced from around 2000 in R-CNN [12] to 300 in Faster R-CNN [28]. The quality of the generated regions results in better accuracy than YOLO, specially when with small objects. Still, the average detection speed is 5 fps.

3. Materials and Methods

In this chapter the materials and methods used during the development of this Bachelor's Thesis are detailed, in essence, the hardware and software. The chapter is divided in section 3.1, which contains an overview of the materials and methods. Section 3.2 where the hardware used is described in detail. Section 3.3, where Docker is briefly described and its use is explained. And section 3.4, in which some deep learning frameworks are studied in order to choose one.

3.1. Introduction

The development of an artificial intelligence based research project has technological requirements and requires the use of different tools in order to help us focus on the main tasks. From the bottom, firstly we will need appropriate hardware in order to train and test deep learning models. On top of it an isolated environment is needed so that its conditions can be reproduced by others later, to do that Docker will be used. Once the environment is ready, a programming framework is needed in order to focus on the deep learning architecture and its parameters avoiding low-level implementation details.

3.2. Hardware

It is well known that deep learning based applications need high computation resources in order to obtain results in a decent amount of time, mainly due to the large size of the datasets used to train the models. For this purpose, 3D Perception Lab research group has provided access to one of their servers, *Asimov*. The server is running Ubuntu 20.04 with an Intel(R) Core(TM) i7 on top of an Asus X99-A motherboard, but where the server capabilities rely most is on the 3 GPUs: a NVIDIA GeForce Titan X for deep learning with 12 GiB of GDDR5 Video Memory and 3072 CUDA cores, a NVIDIA GeForce Tesla K40c with 12 GiB of GDDR5 Video Memory and 2880 CUDA cores for computation and finally a NVIDIA GeForce GT730 for visualization, so that the other 2 are completely available for research purposes. The complete configuration is specified in figure 3.1.

The server is accessed remotely using Secure Shell (SSH) with public/private key pair authentication so only authorized users can access. Apart from computation power, storage is needed for the huge datasets used for training deep learning models. *Asimov* is connected to a Network Attached Storage (NAS) where a Redundant Array of Independent Disks (RAID) type 5 is mounted with a total of 25.3 usable TB. Table 3.1 contains all the details.

Asimov	
Motherboard	Asus X99-A Intel X99 Chipset 4 × PCIe 3.0/2.0 × 16(×16, ×16/ × 16, ×16/ × 16/ × 8)
CPU	Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz 3.3 GHz (3.6 GHz Turbo Boost) 6 cores (12 threads) 140 W TDP
GPU (visualization)	NVIDIA GeForce GT730 96 CUDA cores 1024 MiB of DDR3 Video Memory PCIe 2.0 49 W TDP
GPU (deep learning)	NVIDIA GeForce Titan X 3072 CUDA cores 12 GiB of GDDR5 Video Memory PCIe 3.0 250 W TDP
GPU (compute)	NVIDIA Tesla K40c 2880 CUDA cores 12 GiB of GDDR5 Video Memory PCIe 3.0 235 W TDP
RAM	4 × 8 GiB Kingston Hyper X DDR4 2666 MHz CL13
Storage (Data)	(RAID 1) Seagate Barracuda 7200rpm 3TiB SATA III HDD
Storage (OS)	Samsung 850 EVO 500GiB SATA III SSD

Figure 3.1: Asimov’s hardware specifications

3.3. Docker

Since an isolated environment is needed and both (1) the server processes and programs, and (2) the other users of the server may be an issue, Docker is used by every user in order to obtain an adequate environment to work. Docker is based on the concepts of containers, unlike virtual machines such as Virtual Box or HyperV that are managed by a hypervisor that runs on top of the operating system, Docker runs as any other program in the computer being therefore able to access the kernel resulting in a performance improvement as we can see in figure 3.2. Moreover, compatibility problems due to, for example, different versions can be avoided as well.

While Virtual Machines (VMs) are an abstraction of the physical hardware, a container is an abstraction at the application layer that packages code and its dependencies together. Although both of them allow us to run several instances in the same OS, VMs include a full copy of an operating system, applications... using unnecessary space and in most cases, slowing down the boot of the system. Since docker images, typically, use less space more applications can be handled at the same time.

In order to use docker, we first need to create an image using a Dockerfile and a container will be created when the image is executed. We can either create our own images or download them from Docker Hub and complete them if needed. For example, we can download a

Model name	DS1819+
CPU	INTEL Atom C3538
CPU Clock Speed	2.1 GHz
CPU Cores	4
RAM	4096 MB
Storage	8x4TB disks RAID 5 (25.3TB usable)

Table 3.1: NAS specification **Figure 3.2:** Docker Containers vs. Virtual Machines

TensorFlow image and add on top of it the necessary Python packages that we need. Finally, it is worth mentioning that NVIDIA Docker is used in order to build and run Graphics Processing Unit (GPU) accelerated containers, an essential aspect in deep learning research.

3.4. Frameworks

The implementation of a neural network from scratch is a task that requires time, for this reason and for some years now, easier ways of writing the code have been developed. Frameworks allow us to abstract from implementation details and focus on the architecture and its parameters, so they have become an essential tool among researchers. The list of frameworks grows every year, standing in the way of researchers when they need to choose the most adequate framework for their task. In this case, we will introduce and choose among some of the most popular frameworks nowadays in the scope of object and action detection: Tensorflow, Keras, PyTorch and Darknet.

3.4.1. TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning developed by Google in order to fulfil its needs of building and training neural networks. The name derives from the operations performed on the multi-dimensional data arrays called tensors and its computations are expressed as stateful dataflow graphs. From a usability perspective, it provides a flexible ecosystem of tools, libraries and community resources that allows both researchers

and developers to achieve their goals. Written in Python, C++ and Compute Unified Device Architecture (CUDA). Officially, Python and C++ are the languages where its Application Program Interface (API) are stable, even so some other languages such as JavaScript, Java or Swift also have backward compatibility although it is not guaranteed. Its architectural design allows to easily build and deploy systems seamlessly whether it is on Central Processing Unit (CPU), GPU or Tensor Processing Unit (TPU), an AI accelerator Application-Specific Integrated Circuit (ASIC) developed by Google specifically for neural network machine learning, particularly using Google's own TensorFlow software.

3.4.2. Keras

Keras is a high-level deep learning framework that runs on top of TensorFlow, Microsoft Cognitive Toolkit (CNTK) or Theano developed with the aim of fast experimentation. The common front-end allows us to change between the 3 libraries in an easy manner, moreover, it automatically uses CPU or GPU depending on the system's availability. This framework allows us to create models by adding layers as a sequence (Sequential API), producing linear models, in essence, one path from entrance to exit. More complex models can be easily developed using the functional API, models are created as a graph of layers where each layer is treated as a node and therefore multiple paths are possible.

3.4.3. PyTorch

PyTorch is an open-source machine learning framework for Python, Java and C++ programmed in C++, CUDA and Python. Its development has been mainly carried out by Facebook Artificial Intelligence Group (FAIR) together with many contributors in GitHub. PyTorch is used either as a replacement for NumPy library to use the power of GPU or as a deep learning research platform that provides flexibility and speed, due to the deep Python integration many Python popular packages can be used to code neural networks or process data. It provides a GPU-Ready Tensor library, these tensors (see figure 3.3) can be executed either on CPU or GPU and computing is accelerated by a huge amount when using them, together with their routines to fit many computation needs such as indexing, slicing or linear algebra.

Figure 3.3: PyTorch tensors (equivalent to NumPy ndarrays).

When used for building neural networks it provides a different approach from other frameworks. Instead of using static graphs and reuse the same structure that can't be modified while executing, dynamic graphs are used, that is, the graph is created at runtime. This provides several benefits for different architectures, for example, the number of hidden layers could be modified depending on the epoch counter, or certain parts could be re-executed

until certain conditions for the output are met. On the contrary, certain layers could be skipped as well. Finally, its object-oriented approach provides extensibility and modularity, meeting OOP principles such as inversion of control. This allows us to write architectures in a similar way to Keras (functional or sequential approach), since the parent class is the one handling the logic of the layers.

3.4.4. Darknet

Darknet is an open-source neural network framework written in C and CUDA. The specialized architecture features for Object Detection may reduce its scope compared to other frameworks. However, its performance proved in the different YOLO versions makes it an interesting choice when working with bounding boxes for objects and both accuracy and speed are the key aspects. The developers of this framework also provide some popular architecture pre-trained models using their classification backbones.

4. Proposal

This chapter presents a detailed description of the proposal of this Thesis. Section 4.1 contextualizes the proposal based on the previous chapters. Section 4.2 details YOLOv3, the selected architecture. In Section 4.3 we detail the datasets used in the experiments and the data preprocessing. Finally, Section 4.4 sums up the proposal and the recognition pipeline.

4.1. Introduction

This section describes a recognition pipeline and its components built to achieve the main goal of the project using the knowledge acquired in the related works review made in previous chapters. Due to COVID-19 pandemic restrictions, we have not been able to record our own dataset using UnrealROX framework [20]. This is because it requires the use of virtual reality headsets. Consequently, datasets for hand and object recognition will be selected among the available public ones. For the recognition pipeline, we have used YOLOv3 [27] architecture as its performance is a good balance between speed and accuracy. Previous YOLO versions provide higher framerates (45 FPS against 30 FPS of the third version) but results with smaller objects do not meet our criteria. Despite the fact higher accuracy is obtained with other reviewed architectures, as Faster R-CNN [28], low processing speed has prevented us from using them. For the implementation, Tensorflow-Keras framework was used. Even though some other alternatives may provide better results, the simplicity of this framework enables fast prototyping of new architectures. This allowed us to focus in more important details for this Thesis.

4.2. Architecture: YOLOv3

YOLO, as many architectures designed for image classification, is a Convolutional Neural Network (CNN). Generally speaking, Neural Networks are universal function optimizers trying to mimic the functioning of the human brain and its neurons' interconnection. Specifically, CNNs consists of two differentiated parts: (1) a group of hidden layers used for feature extraction and (2) a set of fully connected layers for classification. Each hidden layer includes a convolutional filter, which essentially is the scalar product between the input and a kernel as we can see in figure 4.1a. The resulting values are fed to a non-linear function, as its name indicates, it is used to reduce the linearity during the learning process. The previous two layers compute the image features that are extracted subsequently by a pooling layer, whose function is to downsample the matrix keeping only the most relevant values. The second part of the process, classification, starts by flattening the matrix from the last pooling layer into a vector. The fully-connected layer and finally, a softmax activation work as a Multilayer perceptron (MLP) and predicts a probability for each class. The more accurate the feature extraction is, the better the classification layers will perform. During the training process,

weights associated to the connections between two layers are updated depending on the difference between the output and target values. The process of updating the weights used to predict the classes is called backpropagation. Figure 4.1b represents an overview of a CNN architecture.

(a) A convolutional operation.

Notice how the output values are computed based on a series of parameters such as, stride and padding.

(b) CNN architecture.

Figure 4.1: Convolutional Neural Network (CNN)¹

Figure 4.2 represents YOLOv3 head with its backbone Darknet-53, an image classification architecture whose purpose is the feature extraction in this pipeline. A model trained with ImageNet is provided by its researchers so as to use it for prediction or transfer learning, further details on this concept are explained later in this section. The genuine version was implemented using the framework Darknet, among the existing Keras versions two of them have been reviewed. The first one² is a basic implementation able to train and execute a YOLOv3 model. An extension of this implementation³ will be used instead as it provides evaluation of the trained models using intersection over union. Moreover, it has been programmed as YOLOv4, the backbone can be modified among other parameters aiming to obtain the best results.

The selected architecture needs some specific annotations and therefore datasets should be normalized using the described format for both train and test labels. The file containing the bounding boxes labels is required to have in each of its lines the image path and N bounding

¹Source <https://medium.com/analytics-vidhya/yolov3-real-time-object-detection-54e69037b6d0> (visited 22/05/2021).

²<https://github.com/qqwweee/keras-yolo3>

³<https://github.com/david8862/keras-YOLOv3-model-set>

Figure 4.2: YOLOv3 architecture.

boxes *image_file_path box₁ box₂ box_n*. Bounding boxes must have the following format: $x_{min}, x_{max}, y_{min}, y_{max}, class_id$ with no spaces, the 4 corners of the bounding box followed by the class id.

In order to reduce the training time transfer learning is applied. This technique consists of initializing the model with previously trained weights during a vast amount of time with a huge dataset, such as ImageNet in the object classification scope. The resulting model is capable of generalising really well and it is used as start point for our network. During some epochs only some layers of the network are updated, usually the fully connected ones, while the rest remains frozen, that is, no weights are updated. In the case of YOLO, the backbone is frozen and only the head is updated with the new input images. Once the transfer learning stage has finished, the resulting model is used as initialization for training a new model where no layers are frozen, this technique is called fine tuning. Together with transfer learning, results are better than training a model from scratch in a shorter amount of time.

4.3. Datasets

Different datasets were reviewed in the state of the art, chapter 2. As the objective is to develop a pipeline that can recognize hands and objects, a dataset for each model is needed. EgoDaily [4] and EPIC-KITCHENS 55 (2018 version) [7] are the selected ones for hand and object detection, respectively.

4.3.1. Hands dataset: EgoDaily

The EgoDaily dataset [4] provides variability of subjects performing different activities at different locations. Besides the provided distinction between left and right hand, resulting in an excellent choice for a hand recognition model. Download instructions and labels can be found in their GitHub repository ⁴, the provided labels have the following format:

```
Path
im_width im_height
```

⁴<https://github.com/sercruzg/EgoDaily>

N

x_min y_min x_max y_max class [repeated for N annotations]

Using Python the labels have been converted to the expected text format. This dataset contains frames in 5 different locations, since we only aim to detect hands indoors the outdoor activities (run and bike) have been removed. Resulting in a dataset of 30 thousand images from 3 different activities (eat, kitchen and office). A balanced subset with 20% of the images has been separated and used only for evaluation.

4.3.1.1. Hand recognition and transfer learning

Transfer learning and fine tuning are key aspects when training YOLO models. Most models are initialized using the generic Darknet ImageNet weights, from those weights a YOLO hand detection model ⁵ has already been trained. YOLO hand detection model from *cansik* was trained with EgoHands [1] and cross-hands, a set of images from people running labelled by the author to train this model. This model cannot be used directly since most locations are outside or not egocentric but it is a better initialization than general object detection weights. EgoDaily annotations have been modified and hand disambiguation removed to evaluate the results using a unique hand class and using cross-hands weights as initialization.

4.3.2. Object dataset: EPIC-KITCHENS

Among the datasets for objects recognition, the choice for experimentation is the EPIC-KITCHENS 55 dataset (known as 2018 version) [7]. Despite the restricted location to the kitchen and the existence of a newer version with more frames and improved labels, this version meets our requirements. Unlike the 2020 version, EPIC-KITCHENS 100 [6], bounding boxes in text files are provided to be used by any architecture. Moreover, the fact that object-action relation is provided, makes this dataset an excellent option to be able to infer the actions from the recognized objects. Annotations can be found in their GitHub repository⁶.

EPIC_train_object_labels.csv contains bounding boxes for the objects in the format detailed in table 4.1. Labels have also been normalized to meet the expected format by YOLO. From the participant_id, the video_id and the frame normalized to a total of 10 digits, we can create a file location. For example /datasets/EPIC-KITCHENS/P01/P01_01/0000056581.jpg refers to the frame 56581 from the first video of the first participant. The bounding boxes values refer to the top corner and the size, since YOLO requires 2 corners (top left and bottom right), from the given values the bottom right corner is calculated. Regarding the class, noun_class will be used, the unique id for each class can be easily segregated and then transformed into text. The equivalences between id and class name are stored in the file EPIC_noun_classes.csv, which contains only three columns: the id, the noun and synonyms for the noun used during the labelling process.

After normalizing all the different values, their bounding boxes were grouped per frame and saved into a text file, resulting in almost 390 thousand labels for objects from 352 classes. With the intention of abbreviating the training time, annotations have been downsampled.

⁵<https://github.com/cansik/yolo-hand-detection>

⁶<https://github.com/epic-kitchens/epic-kitchens-55-annotations>

Column Name	Type	Example	Description
noun_class	int	20	Integer value representing the class in noun-classes.csv.
noun	string	bag	Original string name for the object.
participant_id	string	P01	ID of participant.
video_id	string	P01_01	Video the object was annotated in.
frame	int	056581	Frame number of the annotated object.
bounding_boxes	list of 4-tuple (0 or more)	"[(76, 1260, 462, 186)]"	Annotated boxes with format (<top:int>,<left:int>,<height:int>,<width:int>).

Table 4.1: Epic-Kitchens train object labels.

The first filter affected the number of actors: the first 4 are used for training and the following 3 for evaluation. Secondly, the number of classes have been reduced using the files `EPIC_many_shot_nouns.csv` and `EPIC_many_shot_actions.csv` which contain id for nouns or verbs and the corresponding text value. From the action annotations, the authors have computed the most frequent nouns and verbs. The code in the accompanying notebook⁷ shows that verbs, actions and nouns are considered as *many shot* if they appear over 100 times. Therefore, the second filter only includes these nouns, additionally the file `EPIC_many_shot_actions.csv` includes verb-noun relation, in essence, the possible actions per object. Nouns in this file are considered *many shot* and interactive, as at least one action is associated to them. This improvement in the second filter removes the object with no possible interaction, in addition, providing annotations for adequate activity inference. Finally, from the remaining classes, the 11 with the most annotations compose the final dataset: bag, cupboard, drawer, knife, onion, pan, plate, spoon, tap, box and food. The resulting size includes over 25 thousand images for training and nearly 6 thousand for evaluation, figure 4.3 shows the selected classes with the number of annotations per class. From the previously mentioned noun-verb association, actions per object are grouped and used in the recognition pipeline to infer possible actions.

4.4. Recognition pipeline

The selected architecture and datasets are used to train models for the proposed recognition pipeline. Figure 4.4 contains a representation of the proposal which uses 2 different YOLOv3 models for detection. The first one is exclusively trained for hand detection, because of the significance of the hands from an egocentric perspective. The second one is trained for object detection in the scene.

If a hand is detected, the closest object to it will include the possible actions which can be performed, otherwise, all objects will include them. The closest object to a hand is

⁷<https://github.com/epic-kitchens/many-shot-classes>

Figure 4.3: EPIC-KITCHENS selected classes with the total number of annotations.

determined as the Euclidean distance from the center of hand to the center of object. The prediction results provide 4 different values: x_{min} , x_{max} , y_{min} and y_{max} , the center coordinate is determined as:

$$c = \left(x_{min} + \frac{(x_{max} - x_{min})}{2}, y_{min} + \frac{(y_{max} - y_{min})}{2} \right)$$

And the distance between 2 centers is obtained from:

$$d(c_1, c_2) = \sqrt{(c_1 - c_2)^2}$$

Figure 4.4: Proposed recognition pipeline.

5. Experiments and Results

This chapter contains the experimentation carried out with the obtained results. Section 5.1 describes the training and evaluation process for the pipeline. Section 5.2 describes the experiments and results for the hand detection model, whilst section 5.3 does for objects. Finally, in section 5.4 the results of the pipeline using the most promising models are included.

5.1. Introduction: training and evaluation

From the proposal detailed in the previous chapter different experiments have been conducted and their results evaluated. This section describes the training and evaluation process, then further details are given for each specific experiment.

5.1.1. Training

The training process is started by executing the file `train.py` which allow us to configure different parameters by passing arguments on the command line. Table 5.1 details the different arguments used to train a model. Described parameters and used values represent the default values for most training experiments. It is worth mentioning that the resulting files are not models but weight files, they need to be dumped into a model so that they can be used for prediction and evaluation. The file `yolo.py` with the flag `--dump_model` performs this operation, this file can also be used for prediction using a single model among other tasks.

5.1.2. Evaluation

Similarly to how training experiments are executed, if the results are not evaluated at run-time, the file `eval.py` is used. The arguments in this case are more simple, the majority of them refer to the paths of classes, model, test annotations or anchors. Consequently, only the key arguments are described, these are dependant on the type of evaluation chosen (`--eval_type`): PascalVOC or COCO. For both metrics, a confidence threshold of 0.001 (`--conf_threshold`) is set to cast aside results with low confidence.

5.1.2.1. PascalVOC evaluation metric

PascalVOC [9] evaluation metric computes Average Precision (AP) and Average Recall (AR) for each class. Before defining them, the concept of true and false positives or negatives needs to be introduced. Positives are the detected bounding boxes while negatives are the missing ones. True or false means they were or were not in the annotations, respectively. Precision computes the number of true positives over the sum of true and false positives, in essence, it provides a value of how relevant the obtained results are. Recall measures how good the

positives are regarding the ground-truth, it is calculated as true positives over the sum of true positives and false negatives. In a mathematical format:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Average precision is computed as the area under the interpolated precision-recall curve:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) * p_{interp}(r_{i+1})$$

And average recall as two times the area under the recall-IOU curve, that is the recall averaged over all $IoU \in [0.5, 1.0]$:

$$AR = 2 \int_{0.5}^1 recall(o) do$$

As a whole for the model, mean Average Precision (mAP) and mean Average Recall (mAR) are calculated, they represent the previously described values for all classes:

$$mAP = \frac{\sum_{i=1}^{N_{classes}} AP_i}{N_{classes}}$$

$$mAR = \frac{\sum_{i=1}^{N_{classes}} AR_i}{N_{classes}}$$

Intersection Over Union (IOU) threshold can be modified using the argument `--iou_threshold`, default value of 0.5 is used as a matching surface of 50% is consider good enough for this type of problem. This is the evaluation metric that has been used in this Thesis.

5.1.2.2. COCO evaluation metric

COCO evaluation metric¹ is calculated as PascalVOC, but instead of using one IOU threshold for AP, 10 values between 0.5 and 1.0 are used, by adding 0.05 to the previous threshold. This is the primary metric used in this evaluation although PascalVOC and a strict metric of 0.75 are also computed. Besides, across scales are calculated for: small objects ($area < 32^2$), medium objects ($32^2 < area < 96^2$) and large objects ($area > 96^2$). AR across scales are calculated with the same values, while for the number of objects there are 3 thresholds depending on the maximum number of detections in an image: 1, 10 and 100.

5.2. Hand Detection

In this section, the experimentation conducted for the hand detection model will be explained discussing the obtained results. As described in chapter 4, two approaches have been followed aiming to obtain the best results.

¹<https://cocodataset.org/#detection-eval>

5.2.1. Hand disambiguation

For the first approach, we have used the provided classes from EgoDaily, containing left and right-hand differentiation. Two models have been trained, both of them using Darknet53 backbone, YOLOv3 and the default configuration described before. The first one has been trained from scratch without using transfer learning. The second one has been initialized to the default YOLO Darknet53 weights. For both of them, the same ground-truth is used, see figure 5.1 for size details per class.

Figure 5.1: EgoDaily ground-truth with hand disambiguation.
The value in each class represents the number of bounding boxes for it.

5.2.1.1. Model from scratch

The results obtained training a model from scratch, fundamentally a model trained without transfer learning stage, are grouped in figure 5.2. They prove that this part is necessary for training in a reasonable time, as precision is 7.26% and to improve it more epochs would be needed. The mean Average Precision (mAP) obtained value of 2.19% means that model accuracy decreases if hand disambiguation is used, as left and right hand are mixed up.

5.2.1.2. Transfer learning model

As the previous experiment results demonstrate transfer learning is a basic function when training this type of architectures. The experiment has been repeated with 10 epochs of transfer learning before fine tuning. By initializing the model with Darknet53 weights, results in figure 5.3 show that mAP has improved. Nevertheless, precision has drastically decreased and the number of false positives (figure 5.3d) increased. This is because even though transfer learning is applied Darknet53 weights were trained using COCO classes for object detection. In consequence, to obtain better results training a hand model different weights for initialization or more epochs would be needed.

Figure 5.2: YOLOv3 evaluation results when trained from scratch with EgoDaily.

5.2.2. Generic hand recognition

As results do not meet our requirements, localization errors have been reviewed and left and right hands are recognized. Still, the model confuses them due to the many similarities. Despite the fact this problem could be easily solved by removing hand disambiguation, it would not be enough to improve the precision. Thereupon, two new experiments aim to solve this problem. Firstly, a new backbone was tested removing hand disambiguation. Secondly, Darknet53 ImageNet default weights were replaced by *crosshand* weights. The ground-truth for this experiments consists of the sum of left and right hand in figure 5.1, 11480 bounding boxes for 6000 images belonging to one class.

5.2.2.1. EfficientNet backbone

EfficientNet [31] backbone has been used for experimental purposes. Despite the fact improvements were made for AR (83%) and mAP (53%), precision result worsened to 1.58%. In addition, prediction speed is slower than Darknet backbone. Figure 5.4 contains all the results.

Figure 5.3: YOLOv3 evaluation results when trained from Darknet53 with EgoDaily.

5.2.2.2. Darknet backbone

The use of Darknet backbone replacing default weights with *crosshands* has significantly improved the results. Average precision is at 45% whilst recall and mAP are over 94%. Figure 5.5 groups the charts, subfigure 5.5d stands out as many false predictions have been made. The reason why is the confidence threshold set for evaluation is low (0.01) in comparison to the one used for prediction, a minimum of 0.1. So most of these false positives would not be detected in a prediction environment. As this model was the most promising one from a theoretical point of view, the number of transfer learning and fine tuning epochs were doubled.

5.3. Object Detection

The experiments carried out for object detection consist of testing different backbones with a variable number of epochs. The same backbones used for hand detection have been tested, in addition, CSPDarknet53 has been included. Figure 5.6 contains the ground-truth for the evaluation of object detection models. In spite of choosing the most common interactive objects for training, some of them are not as frequent in the selected test participants, therefore

Figure 5.4: YOLOv3 evaluation results when trained from EfficientNet with EgoDaily (without hand disambiguation).

some of the classes are not balanced. On top of that, the different kitchens used by the selected participants for evaluation, present variability in terms of luminosity or background colour among other factors. This is one of the reasons why the accuracy during evaluation is lower than expected.

5.3.0.1. Darknet backbone

As for the hand detection, the default YOLOv3 backbone has been evaluated and the results are detailed in figure 5.7. Default parameters are used for training this model, as explained before, results for most classes are not as good as expected. Recall value of 41% indicates that almost half of the annotated bounding boxes were detected. However, low precision results show the generalisation issues of the model because of the variability of the new kitchens in comparison to the train ones.

Figure 5.5: YOLOv3 evaluation results when trained from crosshands with EgoDaily (without hand disambiguation).

5.3.0.2. EfficientNet backbone

Results for EfficientNet backbone, figure 5.8, have worsened as well as speed. More bounding boxes are generated using this backbone for most classes, resulting in a reduction of the speed compared to Darknet. Moreover, these bounding boxes are not properly generated as localization errors are made, resulting in a value of mAP of under 6%.

5.3.0.3. CSPDarknet53: YOLOv4

CSPDarknet53 is an improved version of Darknet53 backbone, as tested by the authors of YOLOv4, YOLOv3 can be used as head but adding a neck between backbone and head improves the results. Therefore this model has been tested using YOLOv4 architecture, figure 5.9 contains the results. The reduction in the number of false predictions has improved the overall AP and AR. In spite of the bad evaluation results, this model has performed better than the others. As a result, we can conclude that with an adequate evaluation dataset and the necessary training time this model would perform as expected for object detection.

Figure 5.6: EPIC-KITCHENS ground-truth.
Object classes with the number of test bounding boxes.

5.4. Interaction estimation results

From the conducted experiments, the best models have been used in the described pipeline. It has been programmed as an extension of the file `yolo.py` but in this case, two models are expected, as well as weights and classes for each of them. The implementation can be found in the GitHub repository under 3D Perception Lab’s organization². Annotations and code used are included, with a link to download the selected models for the pipeline and a demo video of the pipeline.

For the object detection task our pipeline uses YOLOv4, the improvements on Darknet53 backbone alongside the insertion of a neck, while maintaining YOLOv3’s head has performed better during evaluation. Hand detection model uses YOLOv3, as the initialization of Darknet53 weights with the previously trained weights for hands (*crosshands*) during transfer learning, has provided outstanding results compared to the other experiments. Figure A.1 includes some sample frames when both hands are detected and two objects are detected. In case there was a third object or as figure A.2 shows, a hand is missing, only the closest objects is the most likely to be used and therefore actions are estimated for it. Finally, some results when no hand or objects are detected in the scene are included in figure A.3.

²https://github.com/3dperceptionlab/tfg_mbenavent

Figure 5.7: YOLOv3 evaluation results with Darknet53 backbone when trained using a downsampled version of EPIC-KITCHENS dataset.

Option	Details
<code>-h, --help</code>	Shows arguments menu.
<code>--model_type</code>	YOLO model type with format backbone_head. Implemented backbones: CSPDarknet53, Darknet53, Darknet19, MobilenetV1, MobileNetV2, MobileNetV3, PeleeNet, EfficientNet, Xception, VGG16; Heads: YOLOv4 (lite), YOLOv3, YOLOv3 Nano, YOLOv2 and the tiny versions for all of them. YOLOv3 is always used as head, Darknet53 is the most used backbone.
<code>--anchors_path</code>	Path to the anchor size, YOLOv3 default anchors will be used.
<code>--model_image_size</code>	Input image size for the network, 416x416 is used.
<code>--weights_path</code>	Pretrained weights for transfer learning.
<code>--annotation_file</code>	Train annotation file path.
<code>--val_annotation_file</code>	Test annotation file path.
<code>--val_split</code>	Validation data percentage used to test during training.
<code>--classes_path</code>	Classes file path.
<code>--batch_size</code>	Batch size for transfer learning train, half is used during fine tuning to optimize memory usage. 16 is used (8 during fine tuning).
<code>--optimizer</code>	Optimizer for training: adam, rmsprop or sgd. adam is used.
<code>--learning_rate</code>	Initial learning rate, 0.001 for Darknet53, 0.000001 for EfficientNet to ensure loss does not get NaN value.
<code>--transfer_epoch</code>	Transfer training stage epochs: 15 is used.
<code>--freeze_level</code>	Freeze level of the model during transfer learning: 0 (None), 1 (Backbone), 2 (Only open prediction layer). Backbone level is used.
<code>--init_epoch</code>	Initial training epochs for fine tune training, 0 is used.
<code>--total_epoch</code>	Total number of epochs, 100 is used.
<code>--gpu_num</code>	Number of GPU to use, 1 used.
<code>--eval_online</code>	Whether to do evaluation on validation dataset during training.
<code>--eval_epoch_interval</code>	Number of iteration(epochs) interval to do evaluation, default=10.
<code>--save_eval_checkpoint</code>	Whether to save checkpoint with best evaluation result.

Table 5.1: Train arguments.

Figure 5.8: YOLOv3 evaluation results with EfficientNet backbone when trained using a downsampled version of EPIC-KITCHENS dataset.

Figure 5.9: YOLOv4 evaluation results with CSPDarknet53 backbone when trained using a down-sampled version of EPIC-KITCHENS dataset.

6. Conclusions

This chapter sums up the conclusions drawn from the work done in this Thesis. Firstly, section 6.1 summarizes the proposal and the obtained results. Followed by section 6.2, where possible improvements are discussed as well as future lines of work in this research area.

6.1. Conclusions

In this Thesis, we have proposed an action estimation pipeline based on simultaneous detection of the hands and objects from an egocentric perspective. Aiming to provide assistance to dependant people, we have started by making an extensive review of the state of the art. From the latest approaches, we have concluded that this point of view provides a unique perspective in which, the study of hand-object interaction can be used to estimate the set of possible actions that may be conducted by a patient. Hence, our pipeline recognizes the objects and the hands separately, for this purpose the best architecture is YOLO. Its third and fourth versions provide a great balance between accuracy and detection speed in comparison to other methods, whose detection speed is 7 times lower. Datasets and generators have been reviewed in the action and object detection scope. A selection of the most appropriate ones for object and hand detection has been made based on our objective.

In order to train and evaluate deep learning detection models a proper implementation is needed, the most popular frameworks and the existing implementations have been studied. Finally, TensorFlow-Keras has been selected and an existing implementation used as the start point for training and evaluation of the models using suitable metrics. The selected datasets have been normalized to fit the architecture input and their size adjusted. The conducted experiments have remarked the importance of transfer learning and fine tuning to reduce training time and improve the results, and how the initialization weights affect the final results. Finally, the most promising models have been used in the prediction pipeline in which the possible actions are shown for the most promising object, the closest to a hand, computed from the 2D position of the object in respect to the hand and the other objects in the scene.

6.2. Future Works

As the development of this project is restricted to an academic year, several improvements and ideas have been left out for future works:

- Improvements on the object detection model. At the moment the pipeline's use is reduced to the kitchen because of the selected dataset. By generating our own detection synthetic dataset, different locations could be added and the set of classes to be used

would be more balanced. Improving the generalisation of the model would improve the evaluation results and in consequence the results when applied to the real world.

If different scenes locations are included, scene identification should be added to the pipeline. Depending on the room of the house, actions of the detected objects could be filtered.

- Study different approaches based on 6D pose estimation of the hands and the objects. Assistance in how to grab an object can be provided using grasping methods if the 6D pose of an object is known. In addition, if both hand and object poses are known, the system can tell the user how to exactly move his hand so as to grab the object properly.
 - Explore different architectures using recurrent neural networks to take into consideration the timeline. As most actions do not take place in a single frame, if the previous frames are known the set of possible actions is reduced. Furthermore, fine-grained actions, such as the different steps for preparing a coffee, can be detected and provide instructions on how to complete the whole activity or a part of it, depending on the patient needs.
-

Bibliography

- [1] S. Bambach, S. Lee, D. J. Crandall, and C. Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.
- [2] Y. Ben-Shabat, X. Yu, F. S. Saleh, D. Campbell, C. Rodriguez-Opazo, H. Li, and S. Gould. The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose, 2020.
- [3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.
- [4] S. Cruz and A. Chan. Is that my hand? an egocentric dataset for hand disambiguation. *Image and Vision Computing*, 89:131 – 143, 2019.
- [5] R. Dai, S. Das, S. Sharma, L. Minciullo, L. Garattoni, F. Bremond, and G. Francesca. Toyota smarthome untrimmed : Real-world untrimmed videos for activity detection. pages 1–7, 2020.
- [6] D. Damen, H. Doughty, G. M. Farinella, , A. Furnari, J. Ma, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. Rescaling egocentric vision. *CoRR*, abs/2006.13256, 2020.
- [7] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018.
- [8] S. Das, R. Dai, M. Koperski, L. Minciullo, L. Garattoni, F. Bremond, G. Francesca, A. Toyota Smarthome, and F. Bremond. Toyota smarthome: Real-world activities of daily living. Technical report, 2019.
- [9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [10] R. Girshick. Fast r-cnn, 2015.
- [11] R. Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.
- [12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

-
- [13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, page 346–361, 2014.
 - [14] H. Hwang, C. Jang, G. Park, J. Cho, and I.-J. Kim. Eldersim: A synthetic data generation platform for human action recognition in eldercare applications, 2020.
 - [15] J. Jang, D. Kim, C. Park, M. Jang, J. Lee, and J. Kim. Etri-activity3d: A large-scale rgb-d dataset for robots to recognize daily activities of the elderly, 2020.
 - [16] G. Kapidis, R. Poppe, E. van Dam, L. P. J. J. Noldus, and R. C. Veltkamp. Egocentric hand track and object-based human action recognition, 2019.
 - [17] Q. Kong, Z. Wu, Z. Deng, M. Klinkigt, B. Tong, and T. Murakami. Mmact: A large-scale dataset for cross modal human action understanding. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.
 - [18] C. Li and K. M. Kitani. Pixel-level hand detection in ego-centric videos. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3570–3577, 2013.
 - [19] Y. Li, M. Liu, and J. Rehg. In the Eye of the Beholder: Gaze and Actions in First Person Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.
 - [20] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez. Unrealrox: An extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation, 2019.
 - [21] H. Pirsiavash and D. Ramanan. Detecting activities of daily living in first-person camera views. Technical report, 2012.
 - [22] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs, 2018.
 - [23] F. Ragusa, A. Furnari, S. Livatino, and G. M. Farinella. The meccano dataset: Understanding human-object interactions from egocentric videos in an industrial-like domain, 2020.
 - [24] J. Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>, 2013–2016.
 - [25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.
 - [26] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger, 2016.
 - [27] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.
 - [28] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.
 - [29] S. Sudhakaran, S. Escalera, and O. Lanz. Learning to recognize actions on objects in egocentric video with attention dictionaries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.
-

-
- [30] E. Talavera, C. Wuerich, N. Petkov, and P. Radeva. Topic modelling for routine discovery from egocentric photo-streams. *Pattern Recognition*, 104:107330, 2020.
 - [31] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.
 - [32] Y. Tang, Y. Tian, J. Lu, J. Feng, and J. Zhou. Action recognition in rgb-d egocentric videos. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3410–3414, 2017.
 - [33] G. Vaca-Castano, S. Das, J. P. Sousa, N. D. Lobo, and M. Shah. Improved scene identification and object detection on egocentric vision of daily activities. *Computer Vision and Image Understanding*, 156:92–103, 2017. Image and Video Understanding in Big Data.
 - [34] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.
 - [35] C. Zhang, Z. Cui, Y. Zhang, B. Zeng, M. Pollefeys, and S. Liu. Holistic 3d scene understanding from a single image with implicit representation, 2021.
-

A. Appendix I: Pipeline results

Figure A.1: Results when 2 hands are detected and 2 or more objects are detected. As each hand is closer to one of the objects, actions for all objects are estimated.

Figure A.2: Results when 1 hand is detected and 2 or more objects are detected.
Notice how only the closest object to the hand includes the actions.

Figure A.3: Results when no hand or object is detected.
If no hand is detected, every object includes the possible actions.