Informatics

Detecting "Getting up" Behavior of a Person From a Bed

using Neural Network and Depth Data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Ali Amini Raoofpour, B.Sc.
Matrikelnummer 01227507

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Dr.techn. Martin Kampel
Mitwirkung: Projektass. Dipl.-Ing. Dipl.-Ing. Thomas Heitzinger

Wien, 16. November 2020

Ali Amini Raoofpour | Martin Kampel

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

Detecting "Getting up" Behavior of a Person From a Bed

using Neural Network and Depth Data

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Ali Amini Raoofpour, B.Sc.
Registration Number 01227507

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dipl.-Ing. Dr.techn. Martin Kampel
Assistance: Projektass. Dipl.-Ing. Dipl.-Ing. Thomas Heitzinger

Vienna, 16th November, 2020

Ali Amini Raoofpour   Martin Kampel

Erklärung zur Verfassung der Arbeit

Ali Amini Raoofpour, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16. November 2020

Ali Amini Raoofpour

Danksagung

Ich möchte meinem Betreuer, Martin Kampel, für seine Unterstützung danken. Ein großes Dankeschön auch an Christopher Pramerdorfer für seine Anleitung und Zeit sowie an die Firma Cogvis [1], die mir Hardware für die Tiefenkamera und Rohdaten zur Verfügung stellte. Ich möchte mich ganz herzlich bei meiner Partnerin Lena bedanken, die mich selbstlos unterstützte und mir in schwierigen Zeiten zur Seite stand. Zu guter Letzt möchte ich auch meinen Eltern, Mehdi und Mahnaz, danken, die mir immer den Rücken stärken und meine akademische Karriere ermöglicht haben. Danke euch allen.

[1] `https://www.cogvis.at/` (visited on 04.11.2020)

Acknowledgements

I thank my supervisor, Martin Kampel for his support. Also a big thank you to Christopher Pramerdorfer for his guidance and time and to the Cogvis[2] company who provided me with depth camera hardware and raw data. I would like to express my deepest gratitude toward my beloved partner Lena who selflessly supported me and stood by me through difficult times. Last but not least, I thank my parents, Mehdi and Mahnaz, who always have my back and enabled my academic career. Thank you all.

[2]`https://www.cogvis.at/` (visited on 04.11.2020)

Kurzfassung

Die Erkennung und Analyse menschlicher Aktivitäten ist ein Bereich der Computervision, in dem menschliche Bewegungen anhand von Kameras und/oder Sensoren erfasst und ausgewertet werden. Der Einsatz von Tiefensensoren in intelligenten Systemen in privaten Bereichen wie individuellen Haushalten oder Krankenhäusern nimmt aufgrund ihrer nicht-invasiven Natur stetig zu. Neben der Wahrung der Privatsphäre haben Tiefensensoren den Vorteil, dass sie kostengünstig sind und auch im Dunkeln genügend Informationen über den menschlichen Körper für verschiedene Anwendungen der Verhaltensanalyse erfassen. Der Aufbau intelligenter Systeme zur Verhaltensanalyse mithilfe von Tiefendaten bringt Herausforderungen wie Okklusion, Datenmangel und den im Vergleich zu RGB-Kameras kleinen Erfassungsbereich mit sich. Die Erkennung von Aufstehverhalten aus einem Bett mithilfe von Tiefensensoren ist ein Beispiel der Verhaltensanalyse in alltäglichen Situationen, die von diesen Herausforderungen betroffen sind. In der vorliegenden Arbeit beschreiben wir die Möglichkeit, das Aufstehverhalten aus einem Bett anhand verschiedener Deep-Learning-Modelle und Tiefendaten zu erfassen. Dies soll als Konzeptnachweis dienen, dass Systeme, die auf Tiefensensoren basieren, mit Hilfe von Deep Learning entwickelt werden können, um die oben genannten Hindernisse zu überwinden.

Da zu diesem Zeitpunkt kein ähnlicher Datensatz mit Tiefenbildern zur Erkennung des Aufstehverhaltens öffentlich verfügbar war, wurden die erforderlichen Daten selbst gesammelt, verarbeitet und kategorisiert, um so als ersten Schritt einen Datensatz zu erstellen. Anschließend wurden mithilfe eines Convolutional Neural Networks aufgabenabhängige Features aus Tiefendaten extrahiert, um die Analyse der Daten mithilfe von zwei unabhängigen Anwendungen durchzuführen: Klassifizierung und Objekterkennung. Anstatt das Netzwerk von Grund auf zu trainieren, haben wir das vortrainierte neuronale Netzwerk ResNet verwendet, um bereits vorhandenes Wissen von ResNet auf unser Modell zu übertragen und dadurch Probleme im Zusammenhang mit Datenmangel zu überwinden. Sowohl die angewendeten Klassifizierungs- als auch Objekterkennungsmethoden konnten das Aufstehverhalten zuverlässig erkennen. Unsere Ergebnisse zeigen, dass die Verwendung von vortrainierten Netzwerken den Hauptbeitrag zum Training leistet, obwohl das hierfür verwendete Netzwerk ursprünglich mit RGB-Daten trainiert wurde. Darüber hinaus demonstrieren unsere Ergebnisse, dass Transfer-Learning Techniken die Größe aufgabenspezifischer Daten reduzieren. Diese Ergebnisse zeigen, dass Convolutional Neural Network in der Lage sind, aufgabenabhängige Features aus Tiefendaten zu extrahieren, die bei der Entwicklung intelligenter Systeme verwendet werden können.

Abstract

A rapid increase in the development and integration of modern technology into society enables the modeling of human behavior in contact with new technologies. Detecting human activities is one of the areas of computer vision through which it detects and analyzes human movement behavior from data gathered through cameras and/or sensors. The application of depth-sensors in intelligent systems in privacy-sensitive areas such as homes or hospitals is steadily increasing, due to their non-invasive nature. Apart from preserving privacy, depth-sensors have the advantage of low cost and being able to operate in the dark, while still capturing enough information about the human body for various human behavior modeling applications. Building intelligent systems to be used in modeling human behavior based on depth data brings challenges such as occlusion, lack of data, and the shorter range of capture in comparison to RGB cameras. Detecting getting up behavior from a bed using depth-sensors represents one example of modeling human activity in real-life settings, which is affected by these challenges. We describe the possibility of detecting getting up behavior from a bed using different deep learning models and depth data as a proof of concept, that systems based on depth-sensors can indeed be developed with the help of deep learning to overcome the above-stated obstacles.

Because no similar dataset containing depth images for detecting getting up behavior was publicly available at the time of writing, all required data was gathered, preprocessed, and labeled to create a dataset as the first step. Then, a state-of-the-art convolutional neural network architecture was used to extract high-level task-dependent features from depth data to investigate the possibility of using them in two applications: classification and object detection. Instead of training the network from the ground up, we used the pretrained convolutional network ResNet to transfer knowledge of what the network has learned to overcome the problems associated with the lack of large data. Both classification and object detection methods were able to reliably detect getting up behavior. Our results show that using pretrained networks is the key contributor in training, despite the fact that the pretrained networks used here were initially trained on RGB data. Furthermore, our results show that transfer learning techniques reduce the size of task-specific data. These results also demonstrate that convolutional neural networks are capable of extracting high-level task-dependent features from depth data which can be utilized in developing ambient intelligent systems.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Humans are diverse in their abilities and knowledge. How a person behaves in any given situation varies based on psychology, sociology, or anthropology factors. Behavioral science, the combined study of these factors, investigates human actions and how decisions are made in real-world situations.[1] Particularly, it addresses the effect of human emotions, environment, human biology, and social elements on the decision making progress[2]. Analyzing and describing human behavior contribute to understanding human actions, detecting behavior patterns based on human actions, and providing assistance in dealing with them.

The rapidly increasing development and integration of computers and modern technology into society demands the study of human behavior in contact with new technologies [82]. One aspect of human and computer intersection is human behavior modeling in computer science. Modeling is a tool for achieving these objectives with three distinguishable purposes: simulation, prediction, and inference. Simulation models mimic real-world behavior whereas prediction models try to detect behavior with a certain probability. Inference models on the other hand aim to predict certain behavior as well as define the reason behind it [119]. Human behavior is modeled at the level of an individual because human behavior in a society can be modeled with statistics without any representation of individuals. Furthermore, because groups are modeled as if they were individuals, behavioral science at the level of an individual can also be applied to groups. [43].

Human behavior modeling in computer science (from now on we omit computer science) focuses on automatically recognizing human activities with the help of sensors and analyzing them. Human activities can be divided into categories of gestures, atomic

[1] `https://www.merriam-webster.com/dictionary/behavioral%20science` (visited on 20.09.2020)

actions, human-to-object or human-to-human interactions, group action, behaviors, and events[112]. Therefore, human behavior modeling aims to inquire and build systems that can provide services in various domains based on the complexity of the human action. Examples for human behavior modeling applications include predicting user behavior on Facebook [6], simulating (mental) health conditions for insurance companies [101, 65], human narrative modeling [81] and modeling human behavior in air combat simulations [118] .

Computer vision is a field of study that tries to address, how a computer can obtain a high-level of understanding from digital images or a sequence of digital images [8]. Detecting human activities is one of the areas of computer vision where it automatically detects and analyzes human movement behavior from gathered data from sources such as cameras or sensors [3]. Data for analyzing and modeling human behavior is classified into sensor-based and vision-based data which can be further classified into either wearable or non-wearable form [14]. Sensor-based devices are non-intrusive but provide only one-dimensional signal data. They cannot perform when the subject is out of bounds (in wearable sensor-based devices, this indicates that the subject is not wearing the sensor) [51] or an action performed by the subject is unidentified [13]. Vision-based devices are intrusive and are able to capture 2D or 3D data, images and videos [71].

There are commercial systems available in market that are based on four combinations of these categories: first, wearable sensor-based systems such as detecting fall based on a wearable sensor-based device [76], capturing individual and group behavior with a wearable sensor-based device named sociometer [78], and detecting natural disasters based on smartphone sensors [32]. Second, wearable vision-based systems such as Gait disorder rehabilitation based on wearable vision-based devices [4]. Third, non-wearable sensor-based systems such as recognizing human activity with the help of WIFI signals [29]. Fourth, non-wearable vision-based systems such as hospital bed video surveillance [46].

Non-wearable vision-based systems have gained a majority of interest in research and marketplace in the present time due to lower cost and complexity in terms of set-up and use [4, 71]. Furthermore, the advancement of machine learning, especially deep neural networks in computer vision, has given computers the ability to solve computer vision tasks comparable to human level [62]. However, there are drawbacks limiting the use of non-wearable vision-based systems such as occlusion of objects inside the frame of view, limited point of view, and lighting conditions [50, 71].

Human behavior modeling in non-wearable vision-based systems can be subdivided based on the data type, which includes two types of Red Green Blue (RGB) data [121] and depth data [85, 84]. Both RGB data and depth data are considered a form of data that are based on RGB images and depth images respectively, such as video or 2D/3D images. An RGB image is a digital image constructed from three matrices that have the same size (known as three-channel images), where each matrix represents one of three colors:

red, green, or blue. Each pixel in an RGB image is a combination of these three colors[2]. A depth image is a single-channel image (one matrix). Each element in this matrix is a pixel in the image which represents how far away an object is located from the camera lens in millimeters.[3] Systems that use machine learning and utilize RGB and depth data types are known as ambient intelligent systems. They are being used in various domains, such as video surveillance [63], object detection in public areas [115, 72], healthcare [4, 33] and fall detection [84, 48]. Human behavior detection systems trained based on Red Green Blue-Depth (RGB-D) data (both RGB and depth data) have shown to achieve higher accuracy in comparison to systems trained based on RGB alone, because depth data provides extra distance information [114]. However, it is not always possible to use both data types because in contrast to conventional RGB cameras, depth sensors are not widely available.

Due to the flexibility and feasibility of connecting devices to the internet, the advancement of artificial intelligence, mass data gathering and remotely controlling systems, humanity is facing new challenges in the 21st century[83]. These challenges include rapid advances in integration of computers into everyday life and increased application of neural networks. The rapid advancements and increased application spark distrust in accepting intelligent systems that can potentially harm or reveal private information. Hence, using RGB cameras in places where privacy can be violated such as personal homes or hospitals are problematic, because RGB images are intrusive and can capture personal information. Consequently, systems that are built upon using RGB data for human behavior modeling are considered unacceptable in privacy-sensitive settings [33]. One way to avoid these issues is to use depth images. The number of human behavior modeling systems that use depth images is increasing, because depth images preserve privacy and can operate without light. Depth sensors have proven to capture sufficient information about the human body in their field of view for human behavior modeling purposes [70] and produce better quality for 3D depth data than single-lens cameras [113]. Hence, reasons arguing in favor of using depth-sensor vision-based devices in privacy-sensitive areas include: to preserve privacy, decrease costs, and operate in darkness while still capturing sufficient information about human actions in a monitored environment.

While depth images are suitable for privacy-sensitive ambient intelligent systems, they bring challenges such as occlusion, lack of data, or shorter range of capture in comparison to RGB cameras, all of which need to be dealt with appropriately [42]. Depth images capture the shape of an object from its point of view with infrared light which makes the device sensitive to sunlight. Therefore, they are more suitable for indoor use where direct sunlight can be avoided. Furthermore, occlusion of an object with a human or interaction of a human with indoor objects adds complication to modeling human behavior based on depth images. Beds or sofas are common objects inside a home, and the interaction of a person with a bed while sleeping or lying while the person uses a blanket can be

[2]https://www.mathworks.com/help/matlab/creating_plots/image-types.html (visited on 07.10.2020)

[3]https://en.wikipedia.org/wiki/Depth_map (visited on 07.10.2020)

challenging for ambient intelligent systems. Getting up behavior is a human activity that involves a person to interact with a bed or sofa where the person is frequently occluded with a blanket. Detecting getting up brings all of these challenges together for modeling human behavior. In addition, it can be directly used as an extension in ambient detection systems such as fall detection where lying on a bed can be misclassified with falling since the person is in a horizontal position, or in hospital when a person in a Intensive Care Unit (ICU) is not supposed to move from a bed.

In this thesis project, we show that it is possible to detect getting up behavior with deep learning and depth images. This can be viewed as a proof of concept that ambient intelligent systems based on non-wearable depth-sensors can be developed to model human behavior with deep learning techniques in order to overcome the above mentioned obstacles. The problem can be formally described as follows:

Problem Definition:

Input:	A depth image x and set of class labels w that describe getting up behavior
Question:	Which function can be used to map a depth image to a class label $f : x \mapsto w$?

Common datasets such as [60, 17] contain tens of thousands of RGB images carefully annotated by humans. Such diverse and large datasets are not available for every problem. A large quantity of labeled data for problems in specific domains is often unavailable and too expensive to generate. This is problematic in deep learning applications because deep learning methods are data-oriented methods that need a large dataset for training [106]. Training based on a small amount of data can cause bias in the outcome of the trained model [94]. However, studies show that the training process and performance benefits from using pretrained neural networks that have been trained on a large general dataset [88, 80, 124, 79]. Hence, in order to address the need of large data for solving the task of detecting getting up behavior with depth images, we purpose to transfer the knowledge of a neural network model that has been trained on a large common dataset to the network that tackles the getting up behavior in order to improve the performance of the model. In machine learning, this technique is referred to as transfer learning [80].

1.2 State of the Art

Interaction and occlusion of humans with indoor objects like beds or sofas pose a challenge for ambient intelligent systems detecting human behavior. For instance, a person lying down on a bed is easily mistaken as a person falling to the ground by fall detection systems. Detecting getting up behavior from a bed with the help of a depth camera and neural network is a unique problem, because at the time of writing, there is no public record of such a system available with limited research published on this topic. However, state-of-the-art methods can be described and compared by considering characteristics of using a depth camera, neural network and transfer learning techniques to analyze similar real-life situations.

For example, Grimm *et al.* used a depth camera mounted in front of a hospital bed to classify four sleep positions (empty, left, right, supine) [28]. This system uses Bed Aligned Map (BAM) [45] to extract a low-resolution map from a depth map that is aligned to the bed position and then performs classification using Convolutional Neural Network (CNN). BAM uses depth images to localize the bed and estimate its surface. Then it divides the surface into 5 cm x 5 cm cells, each of which calculates the average height over the mattress and then feeds the encoded 2D cell-structure into CNN. This system achieves 94% accuracy and thereby outperforms non-spatially aware classifiers like Support Vector Machines or simple Multilayer Perceptrons. [28]. The data for this system was gathered during six months with 78 patients recorded over 94 nights (roughly 600 hours) which adds up to 65 millions images. Not all the images were used since manually labeling this amount of data is too time-consuming if at all possible. Hence, Grimm *et al*'s work proves that gathering and preparing large amounts of depth data for training a neural network from the bottom up is possible, but expensive and time-consuming.

In a different approach, Mithun *et al* tackled the problem of availability of depth data by using transfer learning techniques in detecting objects inside buildings and offices with depth images to preserve the privacy of people passing by the depth camera. Mithun *et al* used a fixed point of view for mounting the camera at the top of an entry or exit point in a room on the ceiling, to minimize occlusion of humans with other objects. The authors introduced an object detector network called (Object Detector using a Depth Sensor (ODDS) in which they used raw depth images to detect various objects such as a backpack, laptop, cup, etc. with the help of embedded GPUs [72] in real-time. The object detection network used was built upon the popular state-of-the-art method Single Shot Detector (SSD) [61] in which Visual Geometry Group (VGG)-16 [102] was utilized as the pretrained CNN backbone. Afterward, the network was trained with the help of curriculum learning and model pruning on three-channel depth images. Each of the depth images is a single-channel depth image, where the same single-channel image is copied over other channels to make a three-channel image. An overview of the approach by Mithun *et al* is depicted in Figure 1.1. This approach indicates that a real-time system for detecting daily objects was achieved by using only depth data with less powerful GPUs.

With the ODDS network, Mithun *et al* demonstrated the possibility of detecting objects with depth images while using a pretrained network to improve the accuracy of network and speed during the training procedure [72]. Thereby, they show the effectiveness of transferring knowledge from networks trained on RGB images to a network working with depth images. Gupta *et al* used the same idea and proposed a technique for transferring knowledge of what is learned from different formats of images, best described in their own words: "We use learned representations from a large labeled modality as a supervisory signal for training representations for a new unlabeled paired modality" [31]. Figure 1.2 demonstrate the idea behind this method. This approach has advantages concerning both image types, RGB and depth images. It shows that transferring what the network learned between different types of images leads to a better performance [31]. However, this

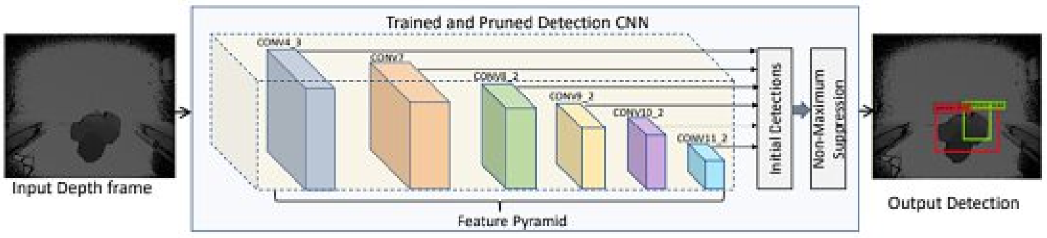


Figure 1.1: An overview of how ODDS works. In the middle of the figure, the VGG-16 used as CNN backbone to extract features at each level of the convolution and outputs them in a feature pyramid where each layer was used to detect objects in different sizes. The bigger the layer, the smaller its receptive field. On top of the feature pyramid layer, the initial detection was applied in order to detect the objects inside the convolutional layers and at last, a non-maximum suppression layer was used to combine similar detections for the same object into one final detection [72].

approach adds complications to training, because choosing a suitable layer for transferring the semantic is crucial for the network for it to be able to learn discriminating features. Also, different image modalities of the same point of view is necessary which requires multiple camera systems at the same spot.

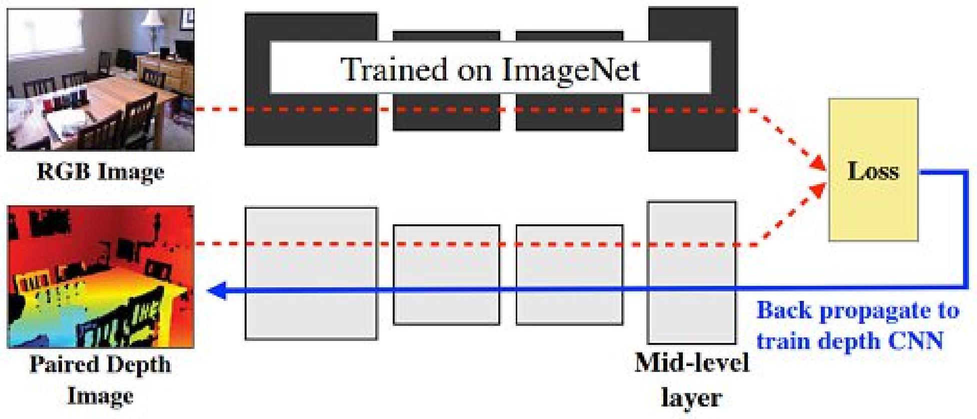


Figure 1.2: The figure demonstrates training a CNN network for new depth images by back propagating the semantic (loss) of what the network has learned from labeled RGB-images for the same paired up images [31].

The two publications mentioned above demonstrate the effectiveness of transfer learning techniques to increase performance and accuracy in comparison to using networks trained without transfer learning, when depth data is concerned. This advantage, along with low cost and preservation of privacy, make depth sensors favorable for ambient intelligent systems in privacy-sensitive areas. Nevertheless, it is possible to avoid using transfer learning techniques and utilize a simpler form of data (based on depth data) for modeling getting up behavior. Manzi *et al* used human skeleton joints extracted from kinect sensors in a simple Mulit-Layer Perceptron (MLP) to detect standing, lying, and sitting positions on a sofa and to detect falls [69]. In this case, the network detects the position with an accuracy of over 97 % in real-time. However, the authors made pre-assumptions about the camera position which helped in detecting positions and occurrence of a fall that may not be applicable in real-life situations. Furthermore, the test subject in this study was not occluded with blankets or similar objects. The camera was fixed in front of the sofa which gives the network the advantage of having less noise in the data. The authors used extracted joint data instead of depth images which increases the inference speed but makes the network reliant on the correctness of the extracted joint data.

In another approach, Zhao *et al* introduced a novel method for fall detection from a bed in real-time by characterizing human behavior via detection of the human head and upper body using random forest [123]. They solved the binary classification problem of falling by using a large margin nearest neighbor classification approach instead of a Neural Network (NN) which gives the advantage of real-time performance without any GPUs by detecting 21 frames per second. However, upper body detection with random forest suffers from noise present in the depth data and a person on the bed covered with a blanket increases errors in detection.

While the above described state-of-the-art techniques provide tools to facilitate the use of depth images in ambient intelligent systems, there is a need for further investigation of an end-to-end network that takes a depth image (without any pre-/post-processing) and analyzes human posture in interaction or occlusion with indoor objects. There also is a need for investigation whether the network is capable of not only recognizing the human action but also localizing it inside the depth image.

1.3 Aims

The main goal of this thesis is to provide a proof of concept for detecting a human action, in this case getting up behavior, based on depth data with the help of deep learning. This overall goal can be divided into three different aims:

Aim 1: To train a neural network that is able to detect getting up behavior of a person from a bed with or without a blanket by taking a single depth image regardless of the angle of movement, provided the whole area of interest is inside the field of view.

Aim 2: To examine the feasibility of detecting getting up behavior with two distinct machine learning applications, namely classification and object detection.

Aim 3: To exploit the effectiveness of using transfer learning techniques in order to facilitate data gathering, training, and assessing model performance.

Detecting getting up behavior of a person from a bed with or without a blanket represents an example for detection of human behavior in interaction with indoor objects, with (blanket) or without (no blanket) occlusion. The main challenge lies within a trade-off between the size and diversity of collected data and engineering a NN architecture. Therefore, the questions addressed in this thesis further include the following:

- Does transfer learning affect the size of gathered data?
- Is it possible to achieve the goal without transfer learning techniques?
- To what extent does the NN architecture affect the outcome?

1.4 Contribution

The contribution of this work is: 1) to gather diverse data, because there are no datasets publicly available for the purpose of detecting getting up behavior, 2) to engineer an End-to-End NN architecture to learn from gathered data in order to detect when a person gets up from a bed or sofa and, consequently, 3) to provide an indication of human behavior modeling based on depth data for ambient intelligent systems. Getting up behavior is detected with two approaches: classification, where each depth image is assigned to a class label, and object detection within the depth image in which the person and the bed are located. Each prediction is solely based on a single frame of depth image without any prior information about either the previous or next frames. Moreover, this thesis analyzes aspects of using a pretrained network on RGB images as a starting point for training a network on depth images.

1.5 Structure

The thesis is structured as follows. In chapter two, a theoretical approach to tackling the defined problem 1.1 is discussed, starting with goal definition and followed by how to achieve and evaluate the goal theoretically. Then, notion and theoretical definitions of the tools that are needed for achieving the goal are presented. In chapter three, the exact implementation of the solution is discussed, along with the presentation of exact methods for gathering and augmenting data. Moreover, the hardware specifications used for obtaining the examination result along with software requirements are presented. The results of training and evaluation of implemented methods are discussed in chapter four, with a focus on exploiting the effect of utilizing transfer learning techniques. The final chapter contains a conclusion and suggestions for future steps.

CHAPTER 2

Theoretical Background

To solve the defined problem 1.1, the computer needs to gain a high-level understanding of getting up motion from depth data. The use of depth images suggests that the problem lies within the field of computer vision. A depth camera set captures consecutive depth images from its field of view. Each image is a single-channel matrix where each element represents a pixel, its value being the distance between the object and the camera in millimeters. By considering that getting up from a bed contains a finite number of class labels, such as lying and standing, it is possible to reformulate the problem to assign a class label to each depth image. This is the definition of the classification task [21].

2.1 Theoretical analysis and historical background

Classification methods can be implemented in three different learning approaches: supervised[49, 108], unsupervised[16, 52] or semi-supervised[126]. Supervised learning is an approach where the data for training is labeled by humans whereas in unsupervised learning the data is not labeled at all. Semi-supervised is the combination of these two: the data is partly labeled and partly remains untouched[74].

Scientific Question 2.1:
Input: Set of single-channel depth images x and set of class labels c.
Question: Which learning approach, supervised, unsupervised, or semi-supervised, is suitable for assigning a class label from c to each element in x ?

The unsupervised learning approach is not suitable for this problem, because it mainly focuses on explanatory analysis from (big) data without any label for which the aim is to understand the underlying structure of the data. After all, the underlying structure/information in the data has not been cleared by humans. However, with the help of deep learning, new methods have been developed in unsupervised learning in order to reach or pass the result in a supervised learning approach [98]. Furthermore, there are methods that specifically use unlabeled data for training in the classification task. For example, Quoc V. Le *et al* built high-level features from a large unlabeled face dataset that detects and classifies faces[53]. Dosovitskiy *et al.* used discriminative unsupervised feature learning to train a network for classification with only unlabeled data [16]. Despite the fact that these methods were able to successfully use unsupervised learning in classification, they still were not able to outperform classification with labeled data [16].

The semi-supervised learning approach is used for building a model in presence of both, labeled and unlabeled data. Semi-supervised learning can be categorized into two slightly different settings: transductive learning and inductive learning [126]. In transductive learning, the concern is to predict an unlabeled example based on a model that is trained on labeled and unlabeled data. It cannot handle new or unseen examples. In inductive learning the goal is to build a model based on both, labeled and unlabeled data, to predict unseen examples such as the state-of-the-art model built by Xiaohua *et al* that used only 10% of the labels in the Imagenet Large-Scale Visual Recognition Challenge (ILSVRC)-2012 dataset to build a classification model by image rotation and visual representation learning [122]. the inductive learning is considered a traditional approach where the goal is to achieve a generic model that detects future unseen data. In other words, the goal of semi-supervised learning is to use the unlabeled data for automatically labeling a dataset or to build a model for more complex data representation where the manual labeling process is not possible or too expensive. None of these points are valid in detecting getting up behavior because depth cameras are inexpensive and can be used to systematically collect and label data. Furthermore, one of the aspects of the defined problem is to examine the effectiveness of transferring knowledge from pre-existing models instead of training a neural network from a large quantity of data. Hence, because unsupervised and semi-supervised approaches are not suitable, classical supervised learning is considered as the main approach for tackling the problem. Supervised learning currently is the main research field of machine learning and numerous state-of-the-art methods and architectures in deep learning have been developed [5].

Scientific Question 2.2:

Input:	A single-channel depth image x and set of predefined labels w
Question:	How to determine a suitable method in classification with supervised learning that can assign a label from set w to the image x?

A feature in machine learning is a distinct quality that is measurable and can be used to distinguish between class labels in classification tasks [11]. Choosing an informative feature is key to build an effective classifier. For instance, in this problem, recognizing a human body and posture is considered an informative and discriminative feature. However, such features are considered task-specific and high-level, because they are built on top lower-level features such as detecting edges or blobs in an image. These lower-level features are built on top of even lower-level feature until it reaches the color pixels of the image. Therefore, in order to assign a label to an image, a method should be able to extract high-level features from an image and utilize them to distinguish between the class labels. High-level features are unique to each problem and they are task-specific. In other words, a method is considered suitable for detecting getting up behavior, if there is a parametric model f $(w = f(x : \theta))$ that maps each image of a dataset to a class label in which the parameters θ of the model are set automatically by a training process that was able to extract task-specific high-level features from the data. Deep learning models with different architectures have shown that they are able to learn such features. They are known to learn non-linearity and high complex features and consist of multiple processing layers that can learn a representation of data with multiple levels of abstraction[5]. Nevertheless, they are considered to be a black box, because their understanding of data cannot be interpreted by humans at present.

The concept of deep learning has been around since the 1960s and the early idea of supervised learning in the neural network, which essentially is a variant of linear regression methods, goes back to the early 1800s [99]. Throughout the last century, neural networks have been developed but never became the main focus due to the high demand for data and computational power. However, one of the neural network milestones happened in 1989 when Lecun *et al* used a learning technique called backpropagation to learn the neural network parameters from hand-written images [56]. Lecun *et al* approach was based on the work of Rumelhart *et al* from 1986, who introduced backpropagation of errors as a learning representation for neural networks [96]. The excitement about deep learning was sparked by Hinton in 2006 [36] who produced a novel deep network called Deep Belief network (DBN). This led to a wave of rapid development in deep learning techniques in different fields such as computer vision. More detailed information on the background can be found in [55, 62, 37]. The flexibility in NN architecture has made it possible to have a variety of solutions depending on the data and the problems such as Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) that are suitable for the problems where the order or sequences of data is important like Natural Language Processing (NLP) [27].

One of the successful architecture in classification is called Convolutional Neural Network (CNN). It is a neural network model that uses a mathematical operation called **Convolution** [27]. Neural network models based on this architecture have achieved performance results comparable to humans[22]. This indicates that CNNs can extract high-level task-dependent understanding from data. Therefore, using CNN in classification is a suitable architecture for detection of getting up behavior. Using CNN is not the only way to tackle this problem. One may argue that RNN or LSTM could be more suitable for this task since they focus on sequence/order and getting up from a bed occurs throughout a sequence of frames. However, both of these approaches are computationally more expensive than CNN because they cannot be parallelized and each input of a network requires the previous output [27]. Furthermore, even though getting up behavior occurs in a sequence of events, a human being can easily detect the behavior by only looking at one frame without any knowledge about the previous or following frames. Therefore, it is necessary to investigate how to model human behavior without any information about the sequence of its occurrence.

A CNN is a black box function that is able to extract high-level features. To investigate its outcome, we expand the goal to use the extracted features from the CNN to not only detect getting up behavior inside an image but also to localize the bed and the person inside the image. These two approaches are referred to as "classification" and "object detection". Furthermore, the effectiveness of transfer learning techniques in both approaches is analyzed.

2.2 Architectures of methods

An artificial neural network is a directed graph where each node in the graph represents a linear or non-linear function and each edge defines the data flow between the nodes[27]. In other words, a neural network is a function that is composed of other functions (neurons). If the connections between the nodes do not form a cycle, the information or data only flows in one direction and never goes backward. Such a network is called a Feed-Forward (FF) neural network. A FF network has an input and output where the input nodes take data like a vector or an image and pass it to the next nodes. This continues until the data flow to the last nodes or the desired output is reached. The nodes or neurons can be grouped by their distance from the input. A group of these neurons/functions is called a layer. the architecture of a network is a construction of different arrangements and connections between the neurons and layers along with the mathematical characteristics of each neuron.

Each neuron is a parametric function that has one or more inputs and one output. It is formally defined as follows:

$$n(W, b; X) = f(\sum_i w_i * x_i + b) \tag{2.1}$$

where $X \in \{x_1, x_2, ...\}$ and $W \in \{w_1, w_2, ...\}$

Each neuron has three parameters of weights (w), bias (b), and activation function (f) beside its input that comes from the previous layer or the input data. Weight is associated with the input and determines the importance of the input, bias is the threshold for the neuron and activation function defines the output of the neuron[77]. A neuron takes every incoming data (x), multiples it by its weights (w) and sums them together. Afterward, the sum is added to the bias of the neuron and pass it through the activation function to produce the output of the neuron. One reason to use activation function as the output of a neuron is to add non-linearity to the model so that the network is able to learn multi-dimensional, complex data [15].

In classification, the output of the network is a vector of the size of class labels where its values represent a probability distribution for all the labels. The vector indicates how probable it is for the image to belong to each of the labels. This is achieved by a function called **softmax**. It takes a vector of discrete values (output of the network) and produces the same size vector so that the sum over all of its elements is equal to 1. Softmax has two advantages: First, it emphasizes the largest value and suppresses the small ones. Second, it is not scale-invariant [27, 11]. Since softmax outputs all values between 0 and 1, the values can be interpreted as probabilities. The output of an object detection method is the output of classification along with the exact position of a boundary box that shows the location of the objects inside the image. Each of these methods uses the notion of CNN and constructs a neural network model that fulfills the required functionality. Each of these components, along with state-of-the-art architectures, is described in the following subsections.

2.2.1 Convolutional Neural Network

CNNs are a form of FF networks that utilize a mathematical convolution. In mathematics, convolution is a mathematical procedure that takes two functions and constructs a new function which describes how one of the two functions is modified by the other one. In CNN terminology, the first argument is the input, the second argument is the kernel and the output is the combination of these two which is a single neuron that is referred to as feature maps[27]. The convolution and its receptive field are demonstrated in Figure 2.1. A convolution layer has two advantages: Firstly, using kernel makes it possible to look at a region of neurons instead of the entire previous layer. This is beneficial because spatially close pixels are highly correlated and correspond to the same object. Furthermore, going deeper by increasing the layers in the convolution layer, the receptive field increases. Secondly, using the same kernel to scan the input reduces the number of parameters. Each edge or data flow in a network has a parameter that needs to be set by the training process. If in a simple FF network, where every neuron in a layer is connected to all the neurons in the next layer (Such a network is called MLP), then the complexity of the computed functions (neurons) in the network increases exponentially as the network becomes deeper [86]. Hence, in CNN, by using and sharing a kernel, only the neurons close to each other are connected which reduces the total number of parameters in a network and consequently, makes it possible to have a deeper network.

Each layer executes a various number of convolutions and then applies an activation function on top to decide if a neuron should get activated or not. This is a replica of how neuronal signal transmission in the human brain works. An activation function can be linear or non-linear depending on the purpose of the network. However, a linear neural network can only model linear data. Therefore, they have a lack of capacity. On the other hand, non-linear activation functions can increase the capacity of the network and introduce non-linearity into the network. Furthermore, activation functions play an important role in training because in order to train a NN with backpropagation, the algorithms calculate the derivative of each function in the network. The derivative of neurons and convolutional layers with a linear function is constant. However, the derivative for non-linear functions is not constant and has an impact on training if the derivative small or zero dependent on the input of the function. Here, three well-known

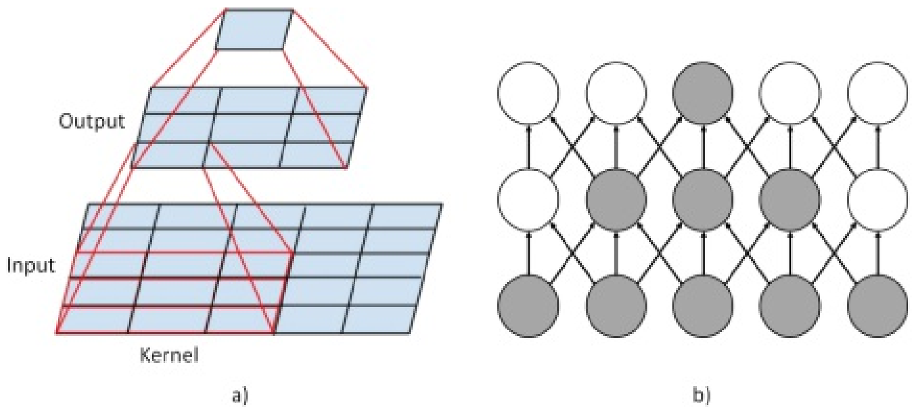


Figure 2.1: Example of convolutional layers. In a) each blue rectangle is a pixel and the kernel has a size of 3x3. The kernel scans the input and convolutes every 9 pixels to one output in the next layer. The last layer on top demonstrates the output of the convolution of the whole input in the next two steps.[1]In b) the sparse connectivity of the convolutional layer is shown. The receptive field of the neurons on deeper layers is bigger than the neurons on the shallow layers [27].

activation functions that are frequently used in the neural networks, are explained: Sigmoid, Tahn, and Rectified Linear Unit (ReLU).

Sigmoid

Sigmoid is an acitvation function that the input is $x \in (-\infty, +\infty)$ and the function output is always between zero and one $sig \in (0, 1)$. The characteristic of the Sigmoid function is that the output of the function almost stays constant either 0 or 1 as long as the input is bigger that -4 or 4 respectively. The definition of the formula is as follows:

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The formula (2.3) shows the derivative of the Sigmoid function and it is computationally cheap to calculate. Therefore, it is commonly used in MLP networks. However, the Sigmoid function has a downside. As demonstrated in Figure 2.2, the derivative of the Sigmoid function reaches zero gradient or saturates and becomes insensitive to training as it gets closer to either 0 or 1[30]. The saturation of neurons will reduce the update of the weights and biases to zero and the neurons on deeper levels cannot get updated. This problem is referred to as vanishing gradient[38]. Hence, due to this limitation, it is not possible to have a network deeper than 5 layers, and this function is usually used in the last layers of convolutional network[15].

[1]`https://redcatlabs.com/2017-06-24-BeginnersDay/#/13/3` (Visited on 22.10.2020)

$$sig'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{2.3}$$

Tahn

Tahn is very similar to Sigmoid but its output domains are slightly different and mostly average close to zero. The formula is as follows:

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

The input is $x \in (-\infty, +\infty)$ and the function output is always $sig \in (-1, 1)$. the difference between Sigmoid and Tahn is the derivative around zero. As shown in Figure 2.2, the Tahn function has a steeper derivative around zero than the Sigmoid function. However, both of these methods suffer from the vanishing gradient problem which makes them not suitable for deep convolutional layers. It is shown that the networks used the Tahn function converge faster than those with Sigmoid function[57]. The derivative formula of the Tahn function is as follows:

$$tanh'(x) = 2sigmoid'(2x) - 1 = \frac{4e^{-2x}}{(1+e^{-2x})^2} \tag{2.5}$$

ReLU

ReLU was proposed by Nair and Hinton in 2010 and has since become the most used activation function for convolutional layers[73, 87]. It is described by a very simple function:

$$g(x) = max(0, x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases} \tag{2.6}$$

ReLU is a threshold function that outputs the input based on each input, if the input value is positive, the function passes the input as the output without any change, if the input value is negative, the function outputs zero. The main advantage of this function is that the derivative of this function (demonstrated in formula 2.7) is cheap to compute since it is constructed by two linear functions. Furthermore, because of the simplicity of the function, the gradient is a constant if $x > 0$. Therefore, it eliminates the vanishing gradient problem mentioned above.

$$g'(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases} \tag{2.7}$$

Despite the advantages of the ReLU in facing vanishing gradient problem, it tends to overfit the network easier than the Sigmoid or Tahn function which its effect can be reduced by the technique introduced by Glorot *et al* ReLU[26] by randomly dropping some neuron during training. Moreover, ReLU has another disadvantage due to its gradient at zero. The zero gradient causes the neurons to die during training and the neuron gets be excluded from the rest of the training. This problem has led to the development of other activation functions such as Leaky ReLU[68], Exponential Linear Unit (ELU) [12] which overcome the dead neuron problem.

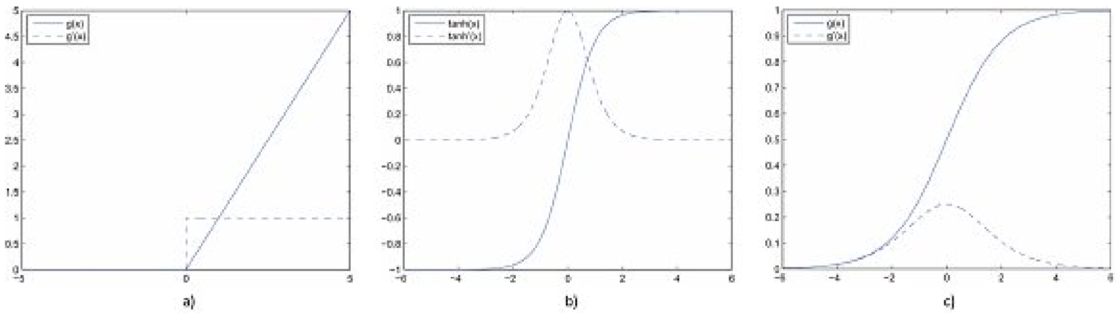


Figure 2.2: Each plot demonstrates a) ReLU b) Tahn c) Sigmoid activation function as straight lines and their derivatives as dotted-lines [15].

A common convolutional block contains three components: in the first layer, it carries out several stacked convolutional layers. Second, it uses non-linear activation functions to introduce non-linearity into the model. Finally, a pooling function stands on top of the last layer to modify the output in order to add invariance to small changes of the input. A pooling function is a tool that helps to make sure a feature is present regardless of its location and additionally reduces the feature maps size[27]. Max pooling is a type of pooling that was introduced by Zhou *et al* that outputs the maximum number within a rectangular area [125]. Max pooling has been extensively using in CNN layers to reduce the size of feature maps [49, 108].

In 2012 rizhevsky *et al* introduced an architecture design based on CNN called AlexNet that won ImageNet Large Scale Visual Recognition Challenge in that year. His paper is considered one of influential papers published in computer vision (as if now their paper is cited 72127 according to google scholar in 2020)[2]. This was the start of a new era of computer vision leading to the creation of a basic concept and simple recipe for designing a CNN. They used eight layers of convolutional layers to extract high-level features followed by three fully connected layers for classification which are trained by Compute Unified Device Architecture (CUDA) programming for parallel training.AlexNet was not the first of its kind that used the power of Graphics Processing Unit (GPU) for training. Dan C.*et al* also used GPU to accelerate training and implemented a similar network architecture.

[2]`https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=ImageNet+Classification+with+Deep+Convolutional+Neural+Networks&btnG=` (Visited on 23.10.2020)

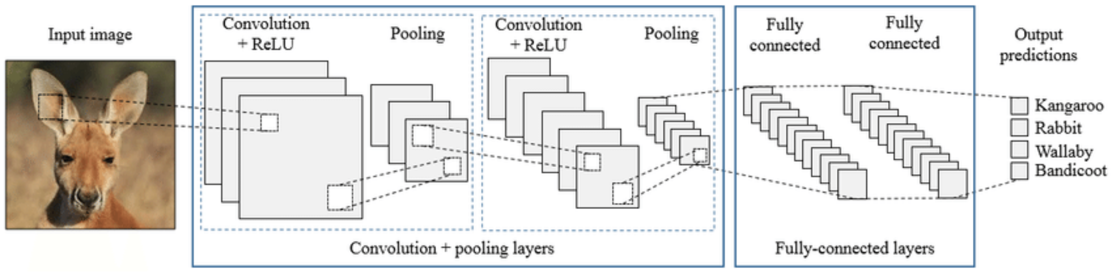


Figure 2.3: Demonstration of a convolutional neural network which contains convolution layer with ReLU and followed by pooling. This layer is called convolutional layer or in short conv-layer. Stacking these layers on top of each other can extract task-depend high-level features which are used in the fully connected layer at the end to classify the image [75].

As demonstrated in Figure 2.3, the basic concept of CNN contains three components: convolutional layer, fully connected layer and output. Various studies showed that deeper convolutional layers can result in better performance of a network because it can extract higher-lever features for classification [108, 109, 34, 35]. Going deeper in a neural network causes various problems such as the vanishing gradient problem, overfitting, or increasing the number of parameters in the network for training which increases the need for computational power. Therefore, CNN-based networks use different techniques to be able to train deeper networks for better performance result such as Residual Network (ResNet)[34], which uses identity function in order to preserve the gradient and avoid vanishing gradient problem while the network depth is increasing, or another design called inception [108] that uses auxiliary output in the middle of the network in order to overcome the problem of vanishing gradient descent in initial layers of the network while training.

2.2.2 Object detection

Object detection deals with identifying and locating multiple objects of certain classes inside one image. The Objects inside an image are located by creating a bounding box around the object. Object detection consists of two tasks: locating the area inside of an image that has a high probability of containing an object and classifying that area. Fundamentally, in each detection, the network proposes one or more regions that have a high probability to contain an object, and then this area market by a bounding box is passed through classification to assign a class label to it. If C is the number of class labels that, an object detection task, background class is also added to the labels $C + 1$ so that the classifier does not to forcefully classify a bounding box to an object of interest if there is a low possibility of the presence of any object. However, the background class will not contribute to the evaluation of the model and only the detection of actual objects contributes to how well object detection performs. This also implies that both parts of finding a suitable boundary box and its classification, contribute to its performance.

Because if a model proposes correct regions but does not classify correctly, it leads to unsatisfactory performance. On the other hand, if a model does classify correctly but the purposed regions are wrong and there is no object in it, it also leads to unsatisfactory performance. Hence, both parts of object detection are highly correlated. In the following, the two state-of-the-art methods name faster Region-Based Convolutional Neural Network (R-CNN) and You Only Look Once (YOLO) are introduced in detail.

Faster Regional-based Convolutional Neural Network (Faster R-CNN)

Convolutional neural networks are so powerful and robust in classifying images due to their ability to extract high-level features with convolutional layers. The first idea to create an object detection is to use the powerful CNN to classify only one object at a time within a bounding box and repeat the process for many other bounding boxes. This is what Girshick *et al* proposed in an approach called R-CNN [25]. They used a selective search algorithm based on colors, scales, and texture to propose regions. Then each region is warped to a fixed size and was fed into a CNN network for feature extraction. At last, they used the extracted features as the input for a Support Vector Machine (SVM) classifier to assign a label to each region. This approach is expensive and slow in terms of computation due to three main reasons as follows:

1. The region proposal algorithm extracts 2000 regions for each image.
2. Feature extraction is expensive and it needs to be done for every proposed region. If there is n number of images, the feature space would be $n * 2000$.
3. It uses three different methods instead of one end-to-end method. It also has a multi-stage training pipeline (CNN cannot improve based on SVM results).

These disadvantages are referred to by the same researcher in his subsequent work where he purposed a new approach to tackle these flaws. He proposed a new network that takes the whole image and set of proposed regions as the input and performs a single convolution network on the entire image to obtain the feature maps. Then it projects the bounding boxes onto the output of convolution layers, as demonstrated in Figure 2.4. With this approach, instead of extracting features for each purposed region, the whole input image goes through the convolution once, and afterward, the relevant features for each purposed region get extracted. Furthermore, he introduced a new layer called Region Of Interest (RoI) pooling layer in order to extract relevant features for each specific region in a fixed size. This layer takes the RoI projections which the projection of the region into the feature maps. Then divides the window from projection into a fixed size grid and max pool each cell inside the grid to achieve a fixed size volume. Afterward, it passes the result produced from RoI pooling layer to a MLP for classification instead of SVM because a MLP is also a neural network and brings the possibility to train the network as a whole. The MLP a is a fully connected network that has two sets of output:

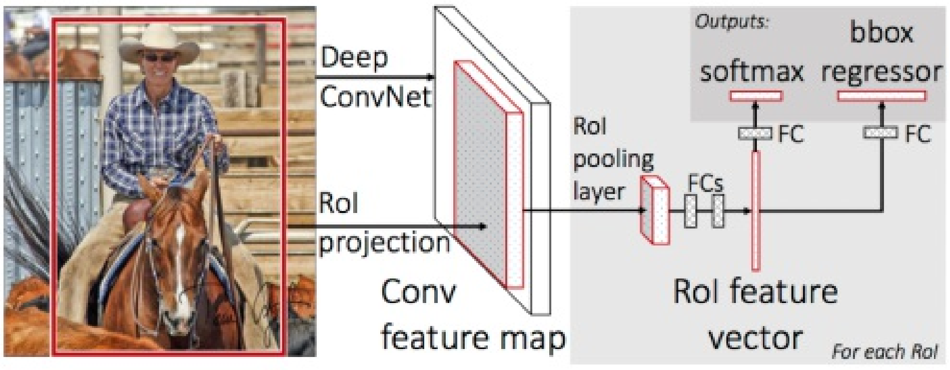


Figure 2.4: Demonstration of end-to-end architecture of fast R-CNN for object detection [24].

a softmax layer that outputs the probability over all class labels and a regressor vector of a size 4 that outputs a real-value number for each corner a bounding box[93].

The improvement of fast R-CNN reduced the running time and led to a slightly better performance in comparison to R-CNN [24]. These advancements exposed that region proposing computation with a selective search algorithm as a bottleneck. Finding Region proposals can still take up to 2 seconds [93]. Months after Girshick proposed Fast R-CNN, Shaoqing *et al* introduced an improvement to fast R-CNN that used a neural network called Regional Proposing Network (RPN) for proposing regions instead of selective search algorithm. This network is called faster R-CNN which is a single, unified end-to-end network for object detection.

Faster R-CNN contains two parts:

1. A deep fully convolutional network that proposes regions.
2. A Fast R-CNN network, as mentioned above.

RPN is a network that takes the feature space as input and outputs bounding box regressor with a binary classifier that clarifies if there exists an object inside the boundary box. The RPN function as a convolutional layer where the kernels of this layer are fixed anchors. As it is demonstrated in the red rectangle area in Figure 2.5, the RPN takes the feature maps and perform a convolution with the fixed size kernel and output a tensor of the size 1 (because the output of a kernel is a single neuron) for each dimension of the feature maps. Hence, the output of the convolutional layer of the RPN is an intermediate layer with the size $(1 \times 1 \times Dimension)$. For instance, in Figure 2.5, the feature maps have the 256-dimensions therefore, the intermediate layer of the RPN has the same number of dimensions. The intermediate layer features are fed to two siblings fully connected layers that output the bounding box layer and probability of existing an object inside the box. This process gets repeated for all k different sizes of anchor boxes. The RPN in essence is a MLP that is implemented with convolutional layers which makes it independent of input size. This is referred to as a fully convolutional network [64]. Hence, in the faster R-CNN, the RPN should function with every size image, since the RoI pooling is only placed after the regions are purposed. The RPN is also translation-invariant (the object should be found regardless of size and location) because of the different shapes of the anchor boxes.

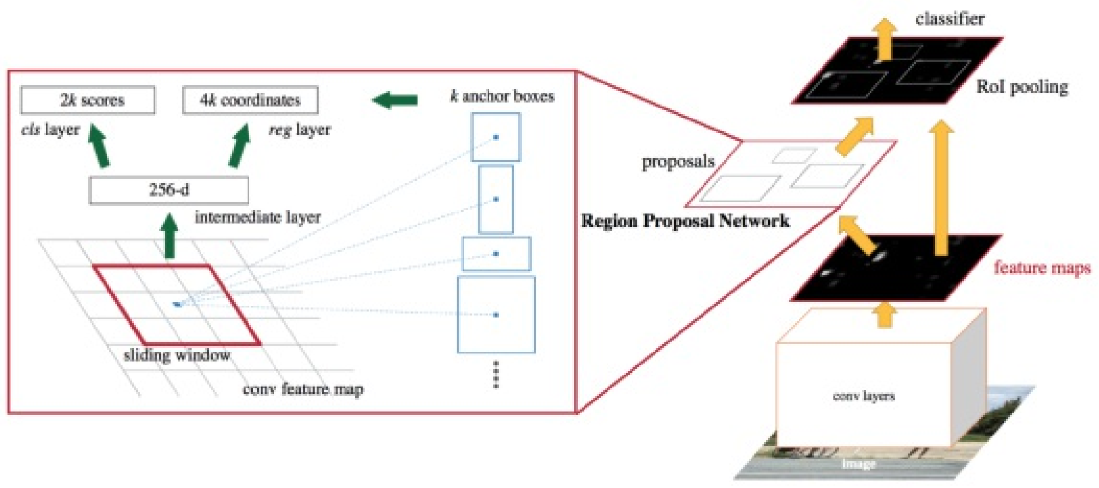


Figure 2.5: Illustration of the architecture of the RPN in faster R-CNN network [93].

The number of anchor boxes are dependent on two configurations: square box size and aspect ratio of the objects in the image. For example, for 5 sizes of (16×16), (32×32), (64×64), (128×128) ,(256×256) and 3 aspect ratios of $[1 : 1]$, $[2 : 1]$, $[1 : 2]$, there are k=15 anchor boxes for RPN. Furthermore, the output of the RPN is dependent on the number of anchor boxes, $2k$ for objectness scores and $4k$ coordinates for 4 different corners of a boundary box as illustrated on the left in Figure 2.5.

YOLO

Faster R-CNN tackles the object detection task by separating object localization and classification. Another approach for object detection was introduced by Redmon *et al* called YOLO. He treats the object detection as a regression problem where the localization and classification happen in one evaluation[90].

The YOLO divides the input image into a $S \times S$ grid and process each cell in the grid. Then it outputs a vector per each cell. Each of the vectors has four parts:

1. B number of anchor boxes
2. One element for box confidence

3. Four elements of (x, y, h, w) for each boundary box
4. C number of class scores (one per each class label)

The output vector has a shape of $(S, S, B \times ((4+1)+C))$. Similar to faster R-CNN the YOLO has been also developed in different versions. The first version YOLOv1, tuned for a challenge [17] and the network tuned to use a 7×7 grid, 2 anchor box per cell(B) and 20 classes (C) which make the output of the network to become $(7, 7, 30)$. The network uses a 24 convolutional layer as a feature extractor to reduce the input image to $(7, 7, 1024)$. Then it flattens the tensor and uses 2 fully connected networks to reshape and output a tensor of $(7, 7, 30)$. Since all the output in each grid is independent of one another, an object that belongs in two or more grid cells can make the YOLOv1 to make duplicate predictions for the same object. To refer to this problem, the YOLOv1 performs a non-maximal suppression per class which is a function that sort all the detections based on their confidence score and take the detections with the highest confidence score for each object.[90]. YOLOv1 treats the object detection problem as a regression problem rather than a classification problem and manages to perform object detection tasks in a single convolutional network. This unique approach brought the advantages of performing real-time, fewer background mistakes in comparison to faster R-CNN. However, YOLOv1 struggles behind Faster R-CNN in terms of localization of small objects or clusters of small objects and detecting the same object with different input sizes.

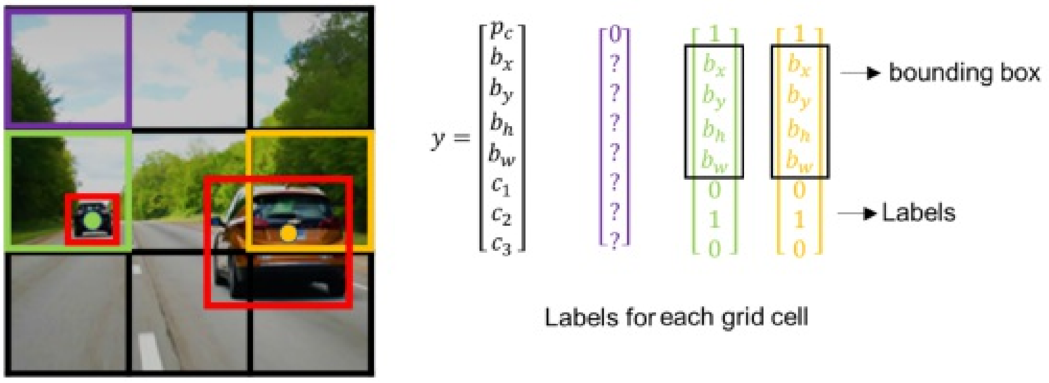

y =
p_c
b_x
b_y
b_h
b_w
c_1
c_2
c_3
0 ? ? ? ? ? ? ?
1 b_x b_y b_h b_w 0 1 0
1 b_x b_y b_h b_w 0 1 0
bounding box
Labels
Labels for each grid cell

Figure 2.6: Demonstration of how YOLOv1 divides the input image into a grid and outputs a vector y for each cell. p_c is the box confidence score that specifies if the object is in that cell. (b_x, b_y, b_h, b_w) defines the bounding box and c_1, c_2, c_3 are class scores for each class label (no background).[3]

To address the disadvantages of YOLO v1, Redmon *et al* proposed several improvements to their original work. One of the notable changes in YOLOv2 is proposing the anchor boxes similar to the faster R-CNN approach. They used k-means clustering to design k suitable anchor boxes with different aspect ratios and sizes on the given dataset then

[3] `http://datahacker.rs/deep-learning-bounding-boxes/` (visited on 22.10.2020)

used them for predicting bounding boxes. They also introduced Darknet19 [89] as the backbone CNN. Darknet19 enabled YOLOv2 to increase the input size from 224*224 to 448*448 while maintaining speed in terms of computation. The YOLOv2 addressed the problems of the first version by using two new tricks: fine-grained features and multi-scale training. Fine-grained features tackle the problem of finding the small objects by dividing the input image into a 13×13 grid. Multi-scale training tackles the problem of finding the same object with different sizes by augmenting the dataset in a way that each object is represented in multiple scales. More details on the improvements can be found in [91].

regardless of the robustness of YOLOv2 at the time, it was a trade-off between losing some accuracy and gaining inference speed and make it a real-time detection. However soon enough, the other approaches like RetinaNet [59] and SSD [61] outperformed the YOLOv2 in terms of accuracy. Hence, Redmon *et al* incorporates incremental improvements by adding skip connections [35], and up-sampling [92] in order to improve its accuracy. Furthermore, YOLO v3 also incorporates a complex and deeper variant of Darknet which is a 106 layer of the fully convolutional network as the backbone of the YOLOv3. More clearance on the development of these changes and other small alteration visit [92].

2.3 Learning process

By definition, a neural network is a parametric model which is a function f takes input x and depends on its parameters θ outputs y. Training of such a function is basically means to optimizing the parameters θ toward a certain machine learning goal like to predict a outcome. Therefore, the training process lies within the field of function optimization [27]. More specifically, the input x is a matrix representation of an depth image and the output is a vector of the size of the class labels. a parametric function is denoted as follows:

$$y = f(x; \theta) \tag{2.8}$$

with θ contains:

- W for the weights for each edge in the neural network.
- b for the biases for each neuron.

Training a parametric model includes finding suitable parameters θ from training data so that the function f can map the input to the output as desired by train data. After training is completed train data is discarded as it is no longer needed for future inference. For training a parametric model, two ingredients are necessary: a loss function (also known as cost or objective function) and an optimization algorithm. A loss function is a function denoted as $L(\theta)$ that measure the performance of the parametric model f with respect to parameters θ on a dataset. The loss function measures the performance of function f by determining the error between the current output of the network and the given target value in the dataset. The smaller the error, the more the function f represents the dataset. Thus, training a network or finding "good" parameters is the process of iteratively changing the parameters in order to reduce the error of the loss function to (almost) zero. This process is called iterative optimization.

Loss Function

In the classification task, the neural network is a valid probability mass function that outputs a vector that each element of it represent a class label, and its values are non-negative and their sum equals one. On the other hand, it is possible to represent the ground-truths in the dataset with the same characteristic as the output of the neural network by simply making a vector of the size of all the class labels with its all elements equal to zero except the label that image belongs to. In such a vector, all the elements are non-negative and their sum equals one. Hence, it is possible to use the principle of maximum likelihood between them under the assumption that labels are independent of each other. This is equivalently described as cross-entropy between ground-truth and the model output [27]. Given the two probability mass function u (ground-truth) and v(model output), the cross-entropy between the two is described as follows:

$$H(u, v) = -\sum_{t=1}^{T} u_t \ln v_t \tag{2.9}$$

The cross-entropy shown in the formula (2.9), measures the dissimilarity between ground-truth and the model output. The cross-entropy penalizes the difference. H is higher if u and v are more different. This formula determines how accurate the classifier performed on the a single sample from a dataset. To calculate the cross-entropy loss function, the cross-entropy between ground-truth and the model output for all sample data is needed. The loss function and softmax function (model output) for the parametric model f given the parameters θ over S number of data samples are as follows:

$$\text{softmax}_c(y) = \frac{e^{y_c}}{\sum_c e^{y_c}} \quad \text{c is for each class label} \tag{2.10}$$

$$L(\theta) = \frac{1}{S}\sum_{s=1}^{S} H(y_s,\ \text{softmax}(f(x_s; \theta))) \tag{2.11}$$

The cross-entropy loss function shown in formula 2.11 is mostly used in classification models where the assumption is the labels are independent of each other. In the object detection task, there are two outputs of only one of them required to use softmax and the other one is the real values of bounding boxes. Hence, a different loss function is required for object detection. As described by Goodfellow *et al* "The choice of cost function is tightly coupled with the choice of output unit."[27]. Because object detection outputs real-values for boundary boxes, it is possible to use loss functions suitable for regression problems. In regression the output of the model is real-values and it is possible to directly compare them with each other by simply looking at the difference between ground-truth value with the predicted value. two common loss functions in regression problems are called Mean Squared Error (MSE) or Mean Absolute Error (MAE) and their corresponding formulas are given below:

$$\text{MSE: } L(\theta) = \frac{1}{S}\sum_{s=1}^{S}(y_s - f(x_s;\theta))^2 \tag{2.12}$$

$$\text{MAE: } L(\theta) = \frac{1}{S}\sum_{s=1}^{S}|y_s - f(x_s;\theta)| \tag{2.13}$$

The MSE calculates the squared of the difference between ground-truth and predicted value. Therefore, the derivative of this loss is the difference between them and as the difference is high the derivative is bigger and vice versa. This is beneficial in the training process in terms of convergence and high accuracy of the model. However, the squared difference makes this loss function sensitive to outliers because as the error between ground-truth and model output increases, the loss function increases faster due to squared property. The MAE calculates the absolute difference between ground-truth and model output. In this loss function, the calculated error does not make the loss to increase rapidly. Therefore, the MAE is not sensitive with outliers but the derivative of this function is fixed and has nothing to do with loss value which makes the training process ineffective when the loss is decreasing.

Gradient Descent

Training a neural network includes optimizing the cross-entropy loss to reach zero. For this, an optimization algorithm is needed. In 1847, Cauchy introduced a gradient-based optimization technique called gradient descent[1]. This function and its variations are the popular choices for training neural networks. The idea behind it is that the derivative of any differentiable function at one precise point provides the slope of the function at that point. It clarifies, how to apply a change in input to obtain a related change in output at that point. Thus, by changing the input in small amounts in the opposite sign of the derivative, it is possible to decrease the function.

Functions with multiple inputs such as loss functions must use the concept of partial derivatives. The partial derivative of a function L with multiple inputs is the derivative of the function at one input where the rest of the inputs are constant in the calculation. Computing the partial derivative of function L based on all of the inputs as a vector is a vector of gradients. In other words, the notion of gradient rise from expanding the partial derivative on all the inputs[27]. The gradient points in the direction of the greatest increase in the loss function denoted as $\nabla L(\theta)$ and change the parameters in order to move in the opposite direction. Formally, it is defined as follows:

1. Compute gradient of loss function denoted: $\nabla L(x)$
2. Update parameters $\theta = \theta - \alpha \nabla L(x)$. where α is a positive scalar defining the step size.

These steps occur in an iterative procedure until $\nabla L(x) \approx 0$ and when the gradient of a function becomes zero, there is no information where to go next or how to change the parameter and the function thus is in a flat situation or the optimal parameters are found. These situations are called critical points. Basically, a function is in one of the possible states depicted in Figure 2.7 when its gradient is zero. If the optimization process reaches either of these points, it stops since no more changes are possible to be applied to the parameters. Among all of the critical points, the critical point b) in Figure 2.7 is unlikely to happen because the algorithm only makes changes in a way that the function value becomes smaller, not greater.

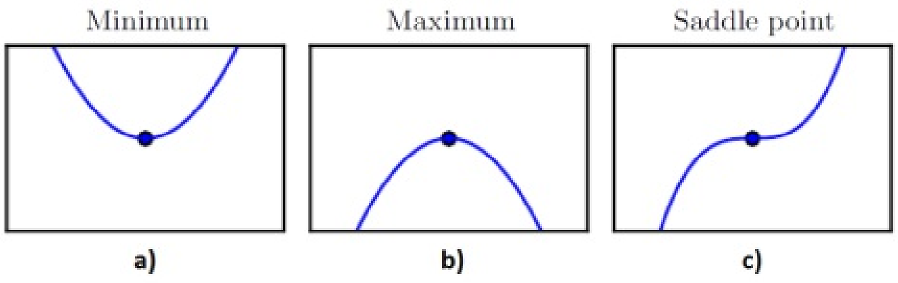


Figure 2.7: Demonstration of three types of critical points where the gradient is zero. a) the function reached a local minimum(or possibly global minimum) and its neighbouring points are higher than the central point. b) the function reached a local maximum and its neighbours are lower than the point. c) the gradient is zero and the neighbouring points are both higher and lower [27].

The absolute lowest value of a function is a global minimum point. A function can have multiple global minima as well as multiple local minima. However, if there is more than one global minimum, the values of the function in those points are equal to one another, whereas the value of the function in the local minimum could be different. Point a) in Figure 2.7 is a point lower than its neighboring points and it is not possible to decrease the function. This point is definitely a local minimum point and a possible global minimum. These types of points make the optimization difficult. However, in the context of deep learning, it is common to agree on a very low value of function even if it is not the global minimum. Also, if the step size α is large enough, there is a possibility for the optimization algorithm not to fall into these points [27].

The gradient descent first computes the gradient of the loss function $l(\theta)$ for the entire train set and then updates the parameters once. This process is called batch gradient descent. One of its advantages is that batch gradient descent guarantees to converge to the global minimum if the function has a convex shape and converge to a local minimum for non-convex surfaces. However, it has a disadvantage: the algorithm updates the parameters only once after calculating the gradient for the entire train set. Thus, the speed of calculating the gradient or training, in general, is highly dependent on the size of the train set. The bigger the train set, which is common in training a neural network, the slower the training phase. The problem is when the size of the train set is large and the batch gradient descent has to recompute the gradient for the same data for each iteration. To overcome this problem, the gradient can be estimated by calculating the gradient only on a single data sample instead of the entire dataset. This method is called Stochastic Gradient Descent (SGD). SGD is a faster method since calculating the gradient is independent of the size of the train set and it can update the parameters frequently. It also reduces memory usage of the GPU. The noisy estimation of the gradient leads to fluctuations in the SGD which gives it the ability to escape local minimum as well as overshooting over local/global minimum [95]. However, it has been shown that decreasing the learning rate indicates the same behavior as batch gradient descent when converging.

Another interesting variant of gradient descent is to take only a constant amount of samples from the dataset. This approach combines the benefits of the two methods described above: on the one hand, the number of samples is not dependent on the dataset size like the batch gradient descent, which makes it computationally cheap and on the other hand, the sample number is not one, so unlike the SGD it has a less noisy gradient estimation and thus it fluctuates less. Commonly, the number of samples varies between 1 and a few hundred, and due to data parallelism, it is two to the power of the number (2^n). So the training process updates the parameters after calculating the gradient for the defined constant number of samples and it continues until all of the samples in the train set are chosen. This is machine learning is referred to as **Epoch**. Epoch is the number that how many times a training process completely goes through the whole training set. For additional variations of gradient descent suitable for optimizing functions, Shilang *et al* explains and compares ten different types of first-order optimization algorithms in a survey [107].

Back-propagation

So far, calculating the gradient was presented as a black box. Computing an analytical expression for a gradient is straightforward, however, determining the exact value can be computationally expensive. A neural network is a computational graph. It contains functions that are composed of other functions. The computational graph is the graphical representation of functions where the edges represent an operation and each node in the graph represents a variable. An operation is a simple function that takes one or more variables and applies a simple operation such as plus, multiplication, or logarithm. Complex functions in a graph can be decomposed into multiple simple operations. The loss function is a complex function where its inputs are the outputs of the neural network. Therefore, loss functions can be represented as a computational graph where its inputs are the inputs of the network and the output is the value of the loss function.

In feed-forward neural networks, the input produces information that flows forward into the network until the output is produced. This is called **forward propagation**, while reversal of the flow of information or the propagation of the output value through a network until it reaches the input is called **backward propagation**. In 1986, Rumelhart *et al* introduced a back-propagation algorithm that uses both, the forward and backward propagation principle, to calculate the gradients based on computational graphs[97]. To compute the gradients, the authors used the inputs to evaluated the graph and stored each local result (Forward propagation). Then they aggregated the local gradients from the loss function to the inputs(backward propagation) by iteratively using the recursive application of the chain rule. For instance, if x is a real number and two functions of f and g map real number to real number, then a composition of the two could be denoted as $y = g(x)$ and $z = f(g(x)) = f(y)$. Derivatives in such a graph can be computed as follows: $y' = f'(g(x))g'(x)$. More details can be found in [97, 54].

Overfitting and UnderFitting

When training a neural network, the loss function only measures the training error, which is the error between ground truth in train data and network output. The training error iteratively decreases as the optimization algorithm continues. The main aspect of training is to increase the ability of a neural network to predict new unseen data correctly. This ability is called generalization. In machine learning, the goal is not only to reduce the training error but also to increase generalization. This can be measured by The generalization error which is defined by measuring the performance of the model on a different dataset referred to as test where the samples are drawn separately from the train set. Both datasets must have the same underlying probability distribution, but the samples must be different. Now the factor that determines whether a neural network model performs well is not only to make the training error smaller but also to keep the disparity between the training error and generalization error small. These factors contribute to two well-known challenges in machine learning: overfitting and underfitting. As explained in/by Goodfellow *et al*: "Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and generalization error is too large"[27].

An example of over/under-fitting in training is depicted in Figure 2.8, where the training error or training loss decreases constantly, but the test loss begins to rise at some point during the optimization. The reason is that in the training phase the optimization algorithm only looks at the training loss and tries to lower it. However, the test only decreases as long as the optimization algorithm is able to generalize well on unseen data. Therefore, in machine learning, a successful optimization ($loss \approx 0$) does not indicate good performance of the model. Monitoring the test loss is essential while training a model. This is considered to be the difference between two fields of machine learning and optimization[27].

In general, it is easier to avoid underfitting by simply reaching lower training loss or, to put it differently, if the training phase is long enough the underfitting should not appear to be the problem. On the other hand, Detecting overfitting is challenging. There are

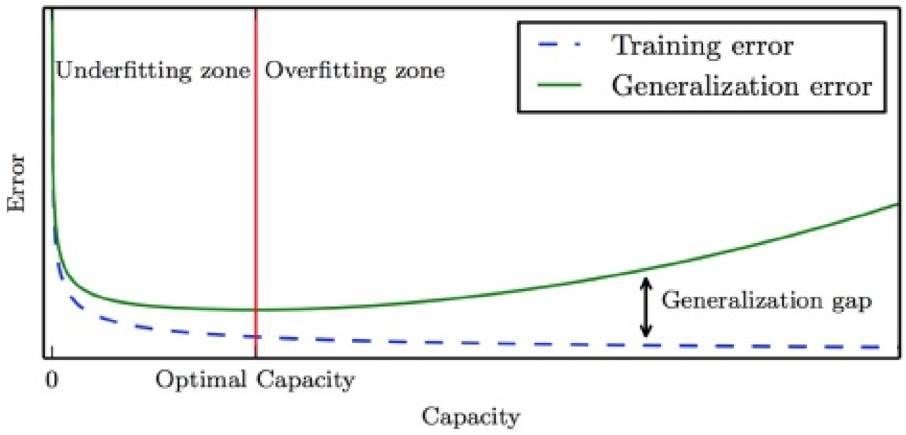


Figure 2.8: The plot demonstrates an example of the relationship between training time and error rate. The shape of the error for training and generalization (or test) has a U shape. On the left side of the graph, at the start of training, the error is high for both of the training and generalization errors. As the training iteratively reduces the training error, the generalization error drops simultaneously and the generalization gap between the two is relatively small. This area is called the underfitting regime. As the training continues, the training error continues to drop but the generalization error begins to rise and the generalization gap grows bigger. This area is referred to as the overfitting regime. The optimal solution is the red line between these two regimes [27].

techniques to combat overfitting, which are often referred to as **regularization**. The regularization techniques that used for solving the defined problem of getting up behavior modeling are listed below:

- **Early Stopping:** It is one of the natural ways of battling overfitting. The idea is to stop the training when the generalization gap starts to grow large. At each Epoch of training, the generalization error is compared with previous ones and if it is lower than all the others, a copy of the network's parameters is stored and the training is continued until there are not smaller generalization error can be achieved in e number of Epochs afterward. Then the training will stop and the stored parameters from the lowest achieved generalization error are used as the output of the training process. It is very simple to integrate this technique in the training process however, it adds another hyperparameter: finding the right number of Epochs can be challenging. If e is too small, the training stops before it actually reaches a local/global minimum and underfitting might happen. If it is too big, this will increase the training time.
- **Data Augmentation:** Another technique to fight overfitting is to use a larger amount of data. When there is more data to be trained on, the model generalizes better. However, it is not always possible to do so because in practice the available

data is often limited or expensive to make. On the other hand, there is a way to artificially increase the size of the train set by making "fake", new, meaningful data. This is called data augmentation. Images are highly dimensional and have many characteristics that can be artificially simulated by techniques like translation, rotation, resizing, cropping, zooming, illumination, mirroring, color spacing etc.[100]. Augmentation techniques are not applicable in every task, because they might change the class label after augmentation. For instance, a digit recognition application is required to detect "6" and "9" and if one applies a 180-degree rotation to them, each of them would be detected as the other. Generally, data augmentation is very effective in image classification and object detection and usually, it improves the generalization of the model. Occasionally, it has been shown that adding noise by augmentation in train data can improve the model [100]. The augmentation techniques used for creating datasets from depth images are illustrated in Figure 2.9.

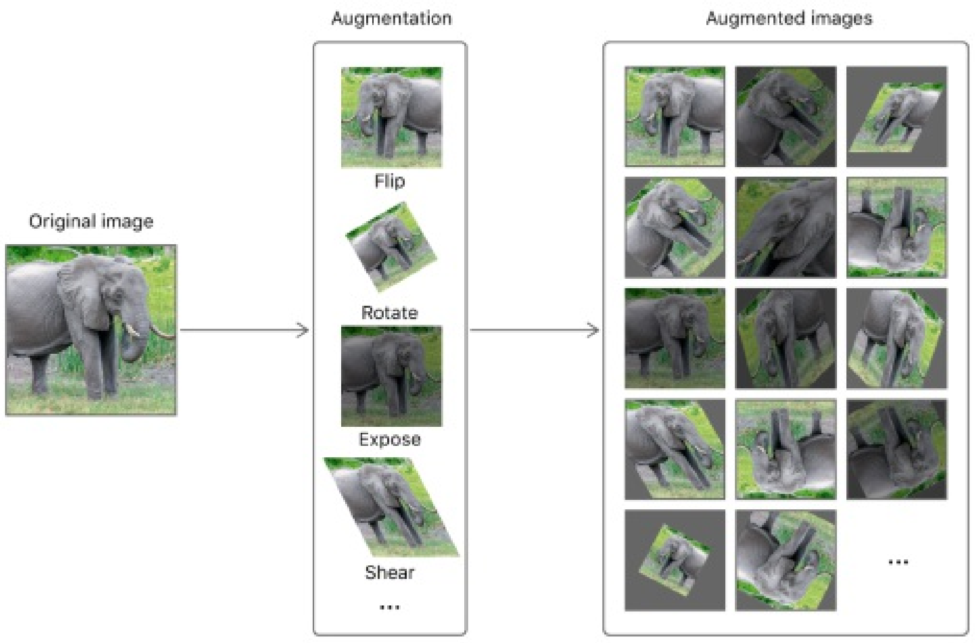


Figure 2.9: Illustration of augmenting an image.[4]

- **Weight Decay:** Another regularization approach is to limiting the capacity of the model by adding a parameter θ penalty to loss function. When the optimization algorithm decreases the loss function, it decreases the loss on train data and the size of parameters θ. It basically decreases the model variance which makes it insensitive to small changes. In deep learning, a common approach is to add only

[4]https://www.raywenderlich.com/5653-create-ml-tutorial-getting-started (Visited on 24.10.2020)

the weight parameters to loss function to penalize large weight parameters in order to prevent the certain inputs from dominating outputs. This is also called L2 regularization[27]. Loss function with weight decay has the following function:

$$L_{wd}(\theta) = \frac{\delta}{2} w^{\intercal} w + L(\theta) \tag{2.14}$$

where $\delta \in (0, 1)$ controls the effect of regularization. Calculated gradient of the loss function is described as:

$$\nabla L_{wd}(\theta) = \delta w + \nabla L(\theta) \tag{2.15}$$

Thus the gradient update for the optimization algorithm becomes:

$$w = w - \alpha(\delta w + \nabla L(\theta)) \implies w = (1 - \alpha\delta)w - \alpha\nabla L(\theta) \tag{2.16}$$

As described in formula (2.16), the weights are decreasing by a constant factor of $(1 - \alpha\delta)$ in each iteration. Therefore, the weights of the network decay to zero (hence, the name "weight decay"). The δ is yet another hyperparameter that needs to be set. If δ is too small, the weight would not change much. If it is too big, it could overrule the cross-entropy loss.

- **Dropout:** Dropout is a regularization technique that focuses on randomly discarding some non-output neurons to make the learning process not to rely on certain neurons. It is a layer where the output of some neurons is multiplied by zero to discard the neuron effect. The choice of discarding a neuron is random. An example of dropout is depicted in Figure 2.10. Another way of describing dropout: when a neuron is discarded in an iteration of training, the network is forced to learn from data with a slightly different and smaller version of the network. In simple terms, if a neuron is, for example, taught to detect edges in an image then removing it temporarily means, the network has to learn with the remaining neurons to detect an edge inside an image. Thus, the network is able to detect edges in different ways and becomes less sensitive to small changes, and generalizes better, which combats overfitting.

  Dropout applies randomly and if a network has n neurons then there are 2^n possibilities. In each iteration during the training phase, a different version of the neural network is trained due to dropout. One can think of training with dropout as training the same number of parameters with n number of neurons which instead of one single big network architecture, it is the collection of 2^n of possible different (smaller) neural networks. Dropout layers only take a place in the training phase. The generalization error is calculated without any dropout layer active. This is because the dropout layer is a regularization technique and battles overfitting in training. Also, randomly discarding some neurons affects the network output and consequently makes the generalization error invalid.

- **Mini-Batch Normalization:** Normalization is the act of adjusting the values of input to an output in a way that the mean of the inputs is zero and the standard

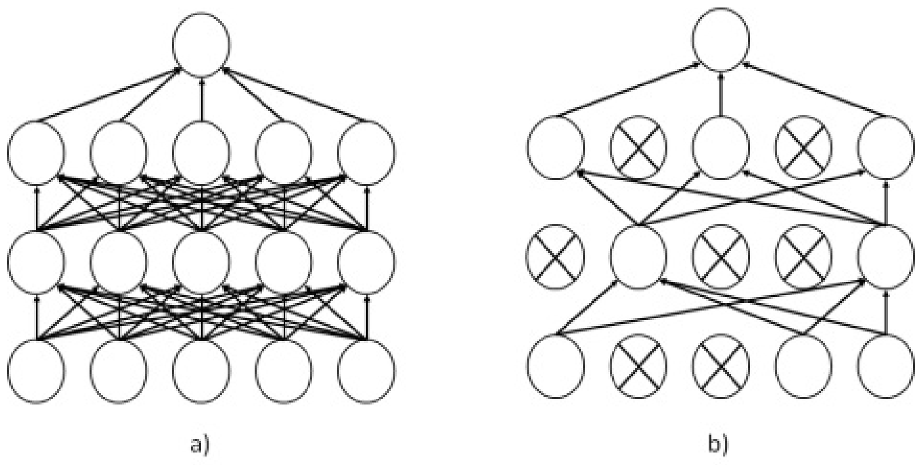


Figure 2.10: Illustration of dropout in a training phase where a network has two hidden layers, one input layer, and one output neuron. In a) the network is depicted before applying dropout. In b) an example of a network after dropout is shown which can be different each time [105].

deviation is one in which the values lies within a normal probability distribution[27]. In terms of deep learning for normalizing, each input image is converted to a three-dimensional Tensor (each dimension for RGB color channels) and the mean ($\mu_1 ... \mu_n$) and standard deviation ($\delta_1 ... \delta_n$) of each dimension over all training images are calculated. Afterwards, every dimension of each input image is normalized with the given mean and standard deviation for n channels as follows:

$$x_n^{norm} = \frac{(x_n - \mu_n)}{\delta_n} \tag{2.17}$$

Normalizing input speeds up the training process. The calculation of the gradient is dependent on input and if the input features are not on the same scale (not normalized), the gradient for weights for some inputs would be different (larger or smaller) between gradients and this would hinder the gradient descent to converge to a minimum. Normalizing inputs are helpful even if input features are on a similar scale. Weights and biases in deep neural networks iteratively change. Therefore, the distribution of input changes as it propagates forward into the network. As Ioffe *et al*describes this phenomenon in the paper: "Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization and makes it notoriously hard to train models with saturating nonlinearities. We refer to the change in the distributions of internal nodes of a deep network, in the course of training, as Internal Covariate Shift."[41]. To overcome this problem, the authors propose to normalized not only the input but also in between hidden layers before activation functions of neurons. Besides, the mean and standard deviation for normalization is calculated over a Mini-Batchs rather than all train images. This approach has two advantages: First, it maintains the input variance which increases the training speed, second, it has a slight regularization effect, since each is scaled by the mean and variation of that, therefore, it introduces some noises in each hidden layer. Increasing the batch size would reduce the regularization effect[41].

2.3.1 Transfer learning

Deep learning is a data-driven method in which the performance of a model is highly dependent on the size and quality of the dataset on which the model is training. The amount of data also varies based on the task. Finding a large dataset could be challenging in the areas that the data is sensitive or expensive to make such as medical data. This problem sparked the idea of enhanced a new task by transferring knowledge from another, similar, or related task that has been learned previously. The action of transferring knowledge is referred to as transfer learning in machine learning. Transfer learning is not a new idea and we as humans use it daily. The idea of transfer learning in machine learning is also not a new concept. The Neural Information Processing Systems (NIPS) 1995 workshop on learning to learn, is considered one of the early sparks of transfer learning for research[111]. Transfer learning can be applied by domain, task, and availability of data. More details on transfer learning, in general, can be found in a publication written by Pan *et al*[80]. Below, transfer learning is discussed in the context of deep learning with a focus on three main aspects: what, when, and how to transfer.

One aspect of transfer learning is to identify what part of one task is common with another task and can be used to improve the target task. Models in deep learning, each network consist of connected neurons that are treated as a black box. However, each network type has a characteristic that distinguishes them. For example, RNNs are suitable for the tasks that the sequence of data is important or MLPs are suitable for classification. There are many different state-of-the-art variants of these networks whose performance is even better than human performance in computer vision [35, 109, 92]. One of the robust network architecture is CNN, which is suitable to extract high-level features from data. Razavian*et al* introduced a new approach to utilize the trained CNN from state-of-the-art networks that are capable of extracting task-dependent features to tackle a similar problem with less amount of data[88]. The authors took a pre-trained network, cut the classifier, and attached a new, shallow classifier to train the modified network on a new dataset. In the training process, the parameter of the CNN part of the network is frozen and does not participate in training. This approach is referred to as **feature extraction** in transfer learning. They used this approach to different computer vision tasks and compared the results. Figure 2.11 shows the clear advantage of using this approach. If tasks between target and source are dissimilar and the dataset is small, it is also possible to extract more generic features (lower-level features) and use them in the source task. This can be done by taking the output of an intermediate layer instead of the last layer [88].

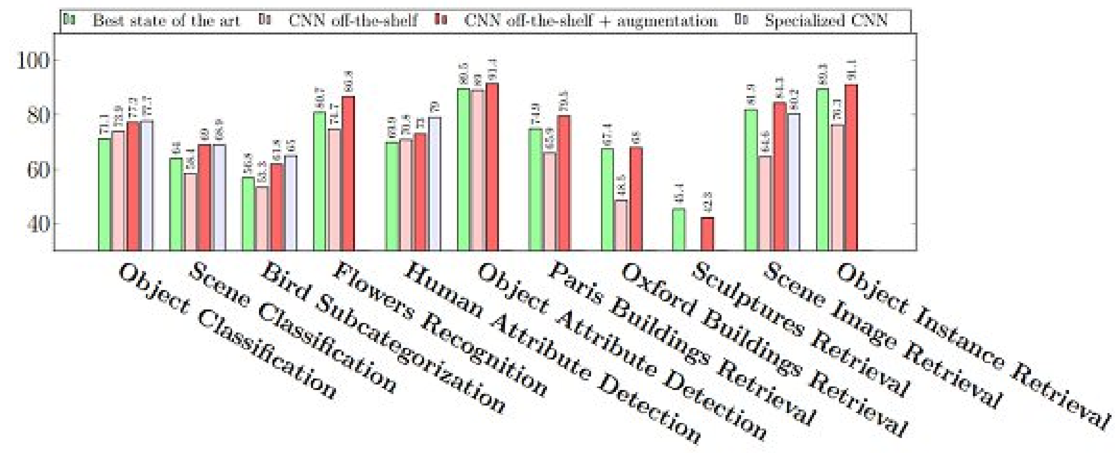


Figure 2.11: Performance of pre-trained models vs task-focused models. The y-axis shows the accuracy and the x-axis defines the computer vision task. In each task, CNN using transfer learning achieves better result than the best state-of-the art results [88].

On the other hand, if the target dataset is large enough, one can modify the network in the same way as before but instead of only train the newly added layer, the whole network is trained. This is commonly known as **fine-tuning**. It is not encouraged to use fine-tuning while the dataset is small because the parameters of such models are large and fine-tuning is prune to overfit the model[120]. Both approaches are developed and implemented for image classification and object detection for solving the lack of data for detecting a getting up.

Transferring the interesting part of a pre-trained CNN is straightforward. A neural network contains two parts: architecture and parameters θ. for transferring the knowledge, it is only needed to build a similar architecture of the interesting network and to initialize its parameters with the values of the same parameters from the pre-trained network[120]. Furthermore, transfer learning should occur before training starts.

2.4 Evaluation

The loss function in machine learning is a function that calculates dissimilarity between the model and the train data. The lower the loss function the closer the model is to the train data. This is pure optimization. However, model evaluation in machine learning in general aims to measure the accuracy of a model on future unseen samples. Loss function cannot provide this type of information. Hence, there is a need to estimate the generalization of a model and measure how well it will perform on unseen samples[27].

Evaluating a model is dependent on the task itself, as well as the data. A model is trained on a train set and uses a test set to calculate the generalization error or model performance. These two datasets are distinct from one another, because if the test set and train set use share data then it is not possible to accurately estimate model performance. There are variables such as learning rate, batch size, number of Epochss etc. in defining a machine learning model that controls the model capacity and performance. These variables are called hyperparameters and should be set before training starts. Setting the best possible hyperparameters is task specific and it is different for each model. The objective of setting the hyperparameters is to increase model performance. Therefore, they need to be optimized on a the data set. However, by tuning the hyperparameters on training set, the model overfits and if they are optimized on test set, the evaluation of the model becomes invalid. This problem can be solved by introducing a new dataset named validation set. A validation set is a distinct subset of train data, which is only used for optimizing hyperparameters. If n is the number of data samples available for a task, then usually it is divided into three distinct subsets: Train set, Validation set and Test set. As depicted in Figure 2.12, the train set takes most of the sample (commonly above 80 % of data samples) and is used for training a model. Validation set and test set each take 10 %, whereas the validation set is used for tuning hyperparameters and the test set is used for final estimation of the performance.

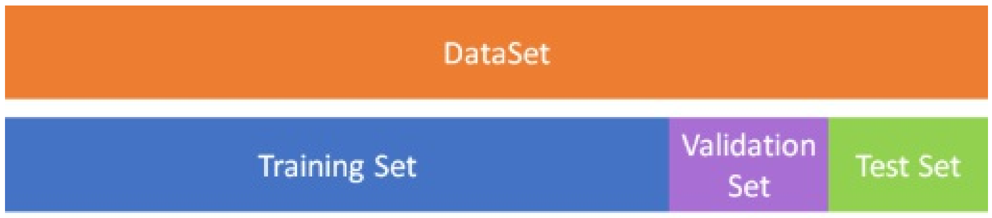


Figure 2.12: Dividing n number of samples available for a task into three distinct subsets of train set(green), validation set(blue) and test set(red)

Dividing the data can also be challenging. For instance, small test set after dividing the data, can lead to statistical uncertainty or if the data is ordered, then dividing each dataset would be biased towards its order. To overcome this, methods for dividing the data such as k-fold cross-validation [20] and bootstrapping [47] has been developed in which they tackle the problems of dividing data and gaining a unbiased evaluation. Each method divides the data into three subset of training, validation and test sets. Training set is only for training the network, validation set is used to tune the hyperparameters and test set is only used to calculate final performance of the model.

Testing all possible combinations of hyperparameters for deep learning algorithms is not possible, simply because each training can take a long time and since there are more than one hyperparameter, then the number of combinations for testing is each value for each hyperparameter becomes large and consequently makes it unpractical and unrealistic (if possible) to test each combination. the first idea to tackle this problem is to find a similar task and use their hyperparameter values as initial values. At last, they can be tuned to fit more in the current task. The second approach is to use an approximative search strategy such as Grid search [57] or Random Search [10] to find suitable values. If there are H

hyperparameters, regardless if they are continuous or discrete, their values can be divided into search intervals $I_1, I_2, ..., I_H$. In grid search, the combination of hyperparameters are sampled uniformly from all intervals I_h for $h \in 1, 2, ..., H]$. In random search, the combination is sampled randomly from all intervals I_h for $h \in \{1, 2, ..., H\}$. In practice, random search has shown better results [10]. In Figure 2.13, the difference between grid and random search are depicted.

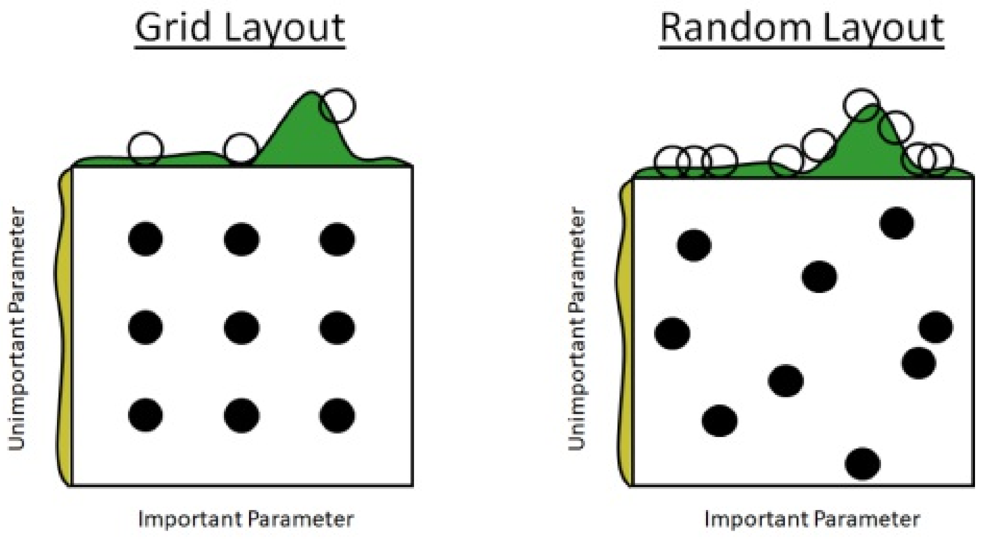


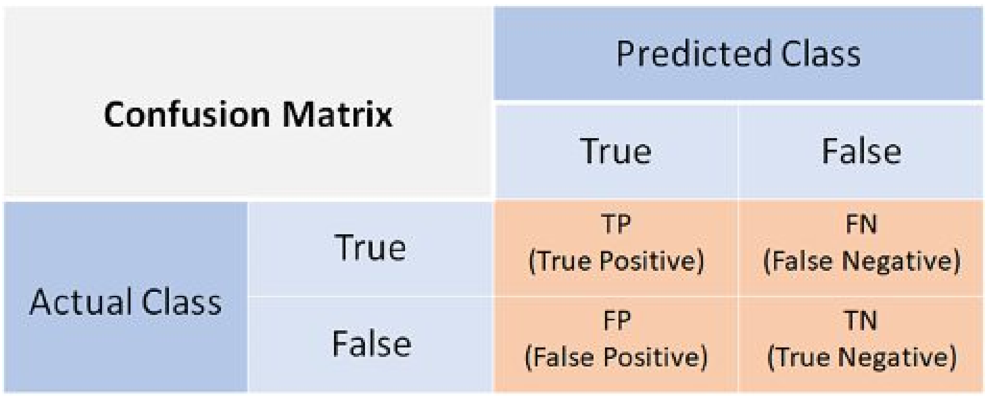
Figure 2.13: The plot demonstrates the difference between random and grid search for nine combinations of parameters. The left side of each plot represents the less effective parameters on the outcome and on the top in green, the more important parameters are depicted. In the grid search plot, the search algorithm explores only three trials on both important and unimportant parameters and skips over the optimal. The random search on the other hand can explore 9 different positions in the space on both sides. It finds the most optimal values for hyperparameters [10].

Evaluating a model is highly dependent on the model output and can vary a lot depending on the task it is solving. Different competitions use different metrics for evaluation. Therefore, In the following subsections, the evaluation metrics that were used for analyzing the performance of the getting up detection model on both tasks of classification and object detection.

Evaluation in classification

A form evaluation in classification is to simply check if the assigned labels to the input are correct. This is called accuracy. Calculating Accuracy in classification is simple and it can be calculated on all three datasets. Basically, the output of a classification network is a vector of the size of all class labels and each element value is a probability that represents how likely the current input of the network belongs to that class. To calculate accuracy, the biggest probability is taken as the assigned label from the network for each input and a network generates an output for all data samples in each dataset. The accuracy is the percentage describing how many data samples are assigned correctly in that particular dataset.

Accuracy only represents how many of the inputs are assigned correctly in total. However, there are situations where the network can only detect one class well because the training data is biased regarding that label. This can lead to high accuracy even though the network has not learned how to detect all the class labels or in another situation where the accuracy is low and one wants to investigate which input is misassigned to which class label. Therefore, the confusion matrix is used to evaluate the output of the network for each class label. A confusion matrix is a matrix in which the number of correct and incorrect predictions are summarized and divided into each class. A confusion matrix has the size of $n \times n$ where n is the number of class labels. Each row in the matrix represents a ground truth and each column represents predicted values. The fundamental of a multi-class confusion matrix is based on a confusion matrix for one class label, where rows and columns show if an input belongs to that label or not. Such a matrix has only four elements as depicted in Figure 2.14.



Confusion Matrix		Predicted Class	
		True	False
Actual Class	True	TP (True Positive)	FN (False Negative)
	False	FP (False Positive)	TN (True Negative)

Figure 2.14: The plot demonstrates a confusion matrix for a class label. The matrix has only 4 elements depicted in orange called TP, FN, FP, and TN that their values are counter for that category. For each sample in the dataset and based on the ground truth and the prediction by model the count number in one of the elements in the matrix increases by one. sum over all elements of matrix is equal to the size of the dataset (TP+FN+FP+TN=#DataSamples).

The definition of each element in the matrix is explained as follow:

- **True Positive (TP):** When an input belongs to the label and that label is predicted correctly.
- **False Negative (FN):** When an input belongs to the label but the model did not predict it correctly.

- **False Positive (FP):** When an input does not belong to the label but the model falsely predicted that it does.
- **True Negative (TN):** When an input does not belong to the label and the model correctly does not assign it to the label.

Based on this confusion matrix, it is possible to calculate accuracy, precision, recall, and F1-measure metric for a class label. Each of them is defined as follows:

- **Accuracy:** indicates the accuracy of the label.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

- **Precision:** indicates the correctness of positive labels. The precision specifies how many data samples that were labeled positive were identified correctly. Hence, high precision indicates less false positives or in other words, with high precision, a model is less likely to misclassify data as positive for a certain label.

$$Precision = \frac{TP}{TP + FP} \tag{2.19}$$

- **Recall:** indicates how many data samples that belong to this class label are recognized. High recall specifies that false-negative detection is low, which implies that the model most likely is able to recognize the true positive samples.

$$Recall = \frac{TP}{TP + FN} \tag{2.20}$$

- **F1-measure:** represents both recall and precision. F1-measure is the harmonic mean between recall and precision and punishes extreme values. Therefore, F1-measure is always near the smaller value of precision or recall. High F1-measure indicates high precision and high recall which consequently means a low number of false positives and false negatives.

$$F1 - measure = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \tag{2.21}$$

Interpreting these measurements can be helpful in evaluation. For example, if a classifier has high precision but low recall, this implies that the model misses many positive examples but if it predicts something, there is a high chance that the prediction is correct.

This binary confusion matrix is expandable to a multi-class confusion matrix by changing the matrix dimension from two to the number of class labels. In the multi-class matrix, if an input belongs to class i and the model predicts that it belongs to class j, then the value of the element in $M_{i \times j}$ increases by one. The elements on the main diagonal are true positives, the elements in the triangle above the main diagonal are false negatives and the elements below the main diagonal are considered to be false positives. In the multi-class confusion matrix no elements are representing true negatives, because, in a classification task, a model cannot assign no label to an input.

Evaluation in Object detection

In object detection, the output of a model does not only contains classification but also has one or more bounding boxes coordinates that localize the object inside the input images. Therefore, classification metrics are not applicable in object detection because using the accuracy or confusion matrix only considers the output label and does not consider if the boundary box is predicted correctly. The evaluation metrics used for the object detection in this project is based on two popular competitions called: Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes (PASCAL VOC)[5] and Common Object in COntext (COCO) detection challenge[6]. Both competitions use the same principle to establish a metric based on a curve called Precision x Recall Curve (PR-Curve). Before defining the PR-Curve, The following notions need to be explained:

- **Intersection Over Union (IoU):** intersection over union is a measurement that evaluates the overlap between two different bounding boxes. If BB_{gt} is the bounding box for ground truth and BB_p is the predicted bounding box, IoU is the area of overlap divided by the area of union. The formula and an illustration of the IoU is depicted in figure 2.15.

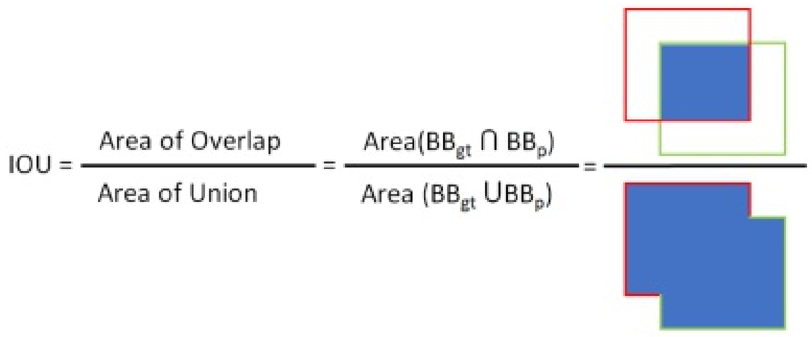


Figure 2.15: Illustration of calculation of IoU between predicted bounding box and the ground truth. The IoU is the area of overlap divided by area of union. In the right most division in the plot, the red box demonstrates a predicted bounding box and the green box is the ground truth.[7]

[5]`http://host.robots.ox.ac.uk/pascal/VOC/` (visited on 04.11.2020)
[6]`http://cocodataset.org/#home` (visited on 04.11.2020)
[7]`https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detecti` (visited on 31.10.2020)

- **Precision and Recall:** Object detection metrics uses a similar principle than the one used for calculating the confusion matrix in classification to calculate precision and recall. However, the terms have a slight different definition.
  - **True Positive (TP):** A detection that it is classified correctly and the IoU>= *threshold*[8].
  - **False Positive (FP):** A detection that it is classified correctly and the IoU<= *threshold*[8].
  - **False Negative (FN):** a ground truth is not detected.
  - **True Negative (TN):** It is not related because it indicates if an object is not detected correctly and an object detection model only outputs found prediction.
  - **Precision:** Precision in object detection determines the ability of the model to identify just relevant objects. It is the percentage of correct positive predictions:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{All Detections}} \tag{2.22}$$

  - **Recall:** Recall in object detection determines the ability of the model to find all relevant objects. It is the percentage of how many of the ground truths are detected:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{All Ground Truths}} \tag{2.23}$$

The IoU threshold used for determining true positives and false positives is dependent on the metric. In PASCAL VOC, it is set to 50 % [18], while in COCO it is set to various values between 50% to 90% to average the estimation[60]. The higher the IoU threshold, the more precise the model detects the bounding boxes. Hence, the threshold has a direct effect on precision. Lowering the threshold increases the number of false positives which consequently decreases the precision and increases the recall. On contrary, raising the threshold increases the number of false negatives that leads to an increase in precision and a decrease in recall. Other than the IoU threshold, there is also a confidence threshold. An object detection model also outputs a score that indicates, how confident the model is to find a label in the predicted bounding box. Usually, predictions, where the confidence score is lower than the confidence threshold, are discarded and are not counted in evaluation.

Based on the metrics, an object detection model has higher performance if it can find all the ground-truth objects (zero false negatives) and identify the relevant object (zero

[8]There is only one true positive per ground truth. If there are more than one detection for same ground truth, the detection with the biggest IoU is considered a TP and the rest FP.

false positives), which implies high in recall and precision respectively. On the other hand, a model considered to achieve less in performance, it increases the number of objects detected in order to have a high recall, which in turn leads to a high number of false positives and lower precision. Or, other way around, to achieve high precision, the model cannot detect all the ground-truth objects which leads to a high number of false positives and consequently low recall. The trade-off between these two is a metric for evaluating an object detection model and a Precision-Recall (PR) curve is a way to demonstrate the trade-off for the model. A PR-Curve is a plot that indicates the association between the two metrics in which the x-axis represents the recall values and the y-axis represents precision values. Each pair of precision and recall is a point in the PR-Curve. By setting different confidence thresholds, one can get different pairs of precision and recall. The higher the confidence, the more precise the model (highest precision), and the fewer objects are detected (lowest recall). Therefore, PR-Curves tends to start with high precision which decreases as the confidence threshold decreases. The reason is that it is harder to detect more of the ground truth objects correctly. On the other hand, the recall increases monotonically, because as the confidence threshold drops, more objects are being detected. An example of such a plot is depicted in Figure 2.16. In practice, instead of setting different confidence thresholds, the table of all detections is sorted by confidence score and the pair of precision-recall is calculated by accumulated TP and FP for each row in the table. For more detailed information on this topic, visit [18, 60].

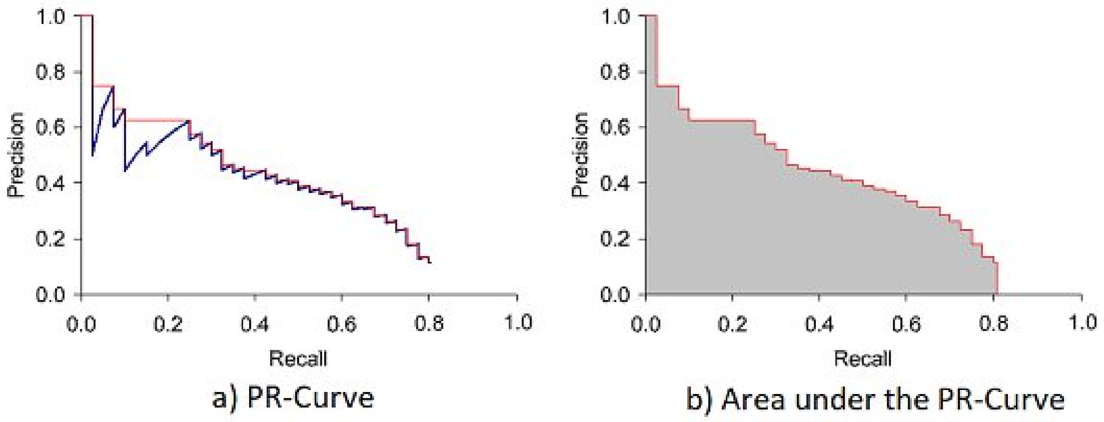


Figure 2.16: an Example of PR-Curve.[9]

To precisely evaluate and compare PR-Curves, the area under the plot is calculated as an evaluation metric for a single class in the object detection model at one threshold. Both precision and recall values are bounded between $[0, 1]$. Hence, the whole area of the plot is equal to 1 and makes the area under the curve to be less than or equal to one. The area under the curve is a numeric metric that is called **Average Precision (AP)**. It is

[9]https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html (visited on 04.11.2020)

the precision averaged across all unique recalls. In order to reduce the impact of zigzags in the plot, the precision is interpolated at all the recall levels before actually calculating the AP. Originally in the PASCAL VOC challenges the recall levels are interpolated at only 11 equally spaced points[18]. Nowadays, the interpolation point is chosen by each unique recall presented by data. The interpolated precision p_{interp} at a unique recall level r is defined as highest precision found for any recall level r' that is bigger than r. The formula for calculating the AP is as follows:

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{2.24}$$

$$AP = \sum_{i=1}^{n-1} (r_{i+1} + r_i) p_{interp}(r_{i+1}) \tag{2.25}$$

The calculated AP is based on a single class label. However, in object detection, usually, there is more than one class label. To have a single value as the metric, the AP must be calculated for each class with the same IoU threshold and be averaged over all APs. This metric is referred to **mean Average Precision (mAP)** which is a single metric for evaluating an object detection model at one exact IoU threshold. The mAP formula is defined as follows:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{2.26}$$

As previously mentioned, the PASCAL VOC challenge used a single IoU threshold of 50% for calculating the mAP in 2012 [18]. However, this threshold is not suitable for all tasks. For example, if an object is small in an image, it needs a higher overlap threshold since a small misplacement of a bounding box might refer to another small object in the image. To have a better evaluation of a model, the COCO challenge defines several IoU thresholds for evaluating a model. They calculate mAP averaged over 10 IoU thresholds in a interval [0.5-0.95] (i.e. 0.50, 0.55, 060, ... , 0.95) as primary metric for the challenge. Furthermore, they also used different object scales, different numbers of detection per class, for evaluation of an object detection model. All the metrics are explained in detail in [60].

CHAPTER 3
Methodology

To build a model that detects getting up behavior, two components are needed: a dataset and a neural network architecture. A dataset for supervised learning contains data representing getting up behavior with manually assigned labels for classification and object detection. It consists of getting up scenes recorded at different angles and single images extracted from these recordings. Each extracted image is labeled to create the final dataset. For each dataset, data augmentation is applied to increase the quantity, diversity, and generalization of the data.

Based on the structure of the depth images and desired outputs, a compatible neural network architecture is constructed which is capable of receiving knowledge from other trained networks. To achieve this instead of building a new architecture, a state-of-the-art network that has been trained on a large RGB datasets is used and modified to be compatible with the new dataset and class labels. Both data collection and network architecture used for detecting getting up behavior are described in detail in this chapter.

3.1 Creating Datasets

To create a dataset for the purpose of modeling getting up behavior, at first the required data is collected in the form video sequences. Second, the gathered data needs to be processed and labeled to the desired dataset for classification and object detection. Each dataset undergoes data augmentation to increase the performance of the model.

3.1.1 Data Gathering

All data are collected in two different formats: depth images and conventional RGB images. RGB images are recorded as an auxiliary source for labeling as the depth images represent distance in their pixels, not colors. Therefore, it is not possible to correctly label them with only depth images. However, RGB images are not used in datasets and training and evaluation are performed only with depth images. Each depth image captures a scene in a one-dimensional matrix in which every pixel represents how far the object is located from the camera in millimeters. Depth images record no information about the color and pattern of the object and the lighting of the recorded scene does not affect the image as long as the lighting reflects the sunlight. Thanks to these attributes, depth image cameras protect the privacy of the recorded person which makes them suitable for use in places like hospitals or nursing homes where the privacy of a patient is a major concern.

Camera and principle of depth imaging

The Astra pro RGB-D camera depicted in Figure 3.1 is used for recording video sequences. This device consists of 3 main parts: an RGB camera, an Infra-Red (IR)-projector and an infrared camera. Video sequences are recorded in (640 × 480) (VGA) quality at 30 Frames Per Second (FPS) with a field of view of 60 H x 49.5 V x 73 D.[1]

Figure 3.1: Camera Astra RGB-D sensor.[2]

A depth camera set measures the distance by using infrared light, an electromagnetic radiation with a wavelength between 700 NanoMeters (nm) to 1 MilliMeters (mm)[67]. These wavelengths are longer than visible light and therefore not visible to the human eye. The visible spectrum of wavelengths lies between 400 nm and 700 nm[67]. Infrared light can be visualized with night vision cameras.

Taking a single frame of a depth image is based on the invention of Freedman *et al* [19] that involves three steps: first, the device projects an irregular pattern of infrared dots (around 300,000 dots) into the field of view[45]. Second, the infrared camera captures the infrared light that bounces back from objects in the field of view. The projected dot

[1]`https://orbbec3d.com/product-astra-pro/` (visited on 01.11.2020)

[2]`https://orbbec3d.com/wp-content/uploads/2018/10/astra-pro-for-slider.png` (Visited on 01.11.2020)

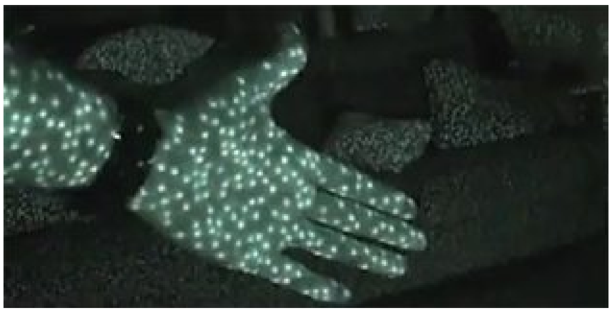


Figure 3.2: Illustration of projected dot pattern from an IR-projector into its field of view [103].

pattern is a reference dot pattern that is captured in a calibration process at a known distance from the sensor and is stored in the memory of the sensor. As a third and final step, the captured image is compared to the projected dot pattern in order to calculate the distance between object and camera. Figure 3.3 illustrates a dot observed in both projected and observed pattern. When a dot is projected at an object and bounces back to the camera, it will be either in the same place that it was projected or it will be shifted in directions of left or right. If the distance between the object and camera is smaller than the reference pattern, then the dot in the observed pattern is located on the right side of the same dot in the projected pattern with a disparity and if the dot is located on the left side, then the distance is larger than the distance between projected pattern and camera.

The distance between IR-camera and IR-projector is B, the distance from B to the projected pattern is Z_0, and the disparity between the projected dot and observed dot is D. In this case, it is possible to calculate the distance Z for each dot from the similarity of triangles as follows:

$$\frac{D}{B} = \frac{(Z_0 - Z)}{Z_0} \implies Z = \frac{Z_0(B - D)}{B} \tag{3.1}$$

In this formula, B is known and remains constant. Z_0 can be determined by calibration and the disparity D is the displacement of a dot or a group of neighboring dots between the projected and observed patterns. More detailed information about the geometrical calculation and deeper insight into the different models of estimation can be found in [45, 66, 104, 23].

Recording settings

The video sequences are recorded in different bedrooms and living rooms where different actors (male and female) are simulating getting up behavior. Each field of view is set to represent a possible real-world use case of positioning the camera setup. The camera is positioned to record scenes from the right, left, and center. Generally, no specific

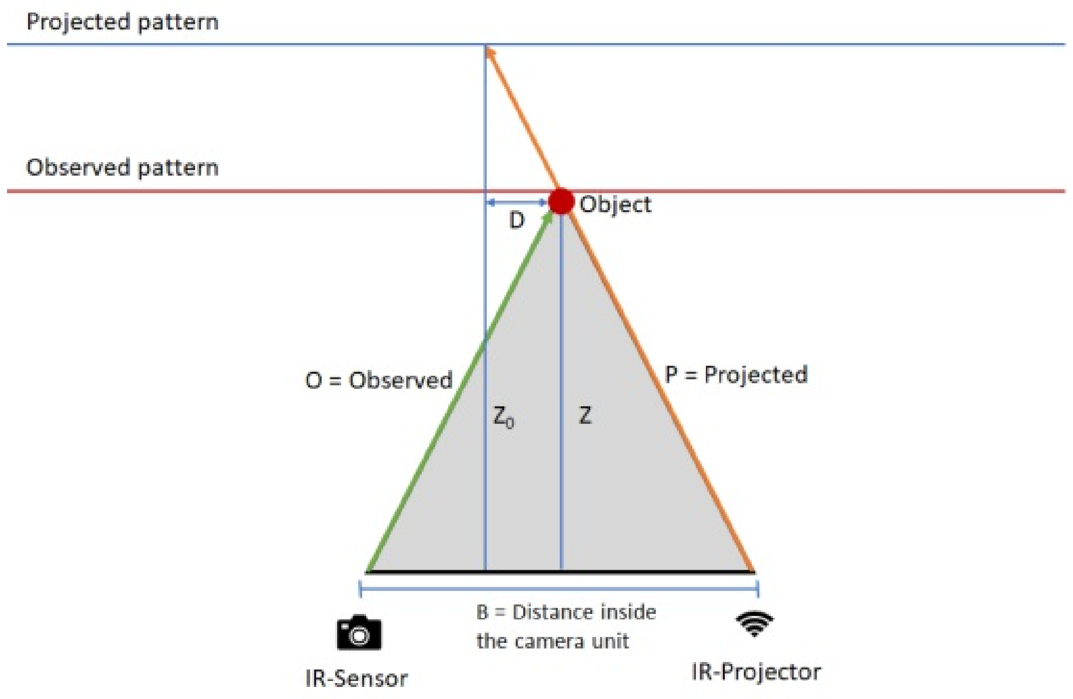


Figure 3.3: Demonstration of an example of projected and observed dot pattern for calculating the distance z when the object is closer to the camera than the projected pattern. The disparity D indicates the displacement of a same dot in both of the projected and observed patterns. (inspired by [45])

restrictions are considered on where and in which position the camera should be placed inside a room. The only rule being that bed and person should be completely inside the field of view of the camera. Each actor in each recording simulates a variety of getting up behaviors with and without using a blanket. The getting up behavior varies as each person gets up from a bed dependent on the characteristics of their physical body and habits. Getting up behaviors like jumping on the bed or standing on the bed before getting up etc. are not considered and not examined during evaluation. Examples of recording situations are depicted in Figure 3.4.

3.1.2 Data pre-processing

All the video sequences are recorded with the maximum possible quality (640 × 480 resolution @30 FPS) with the Astra RGB-D camera unit. Hence, from each second of the depth stream, 30 depth images with the same resolution are extracted and stored separately as Portable Network Graphics (PNG) image file. The PNG file format is chosen because it uses a non-lossy compression algorithm that avoids loss of information when storing on a hard-drive. Furthermore, this format does not degrade image quality after editing the image which is crucial for Data Augmentation.

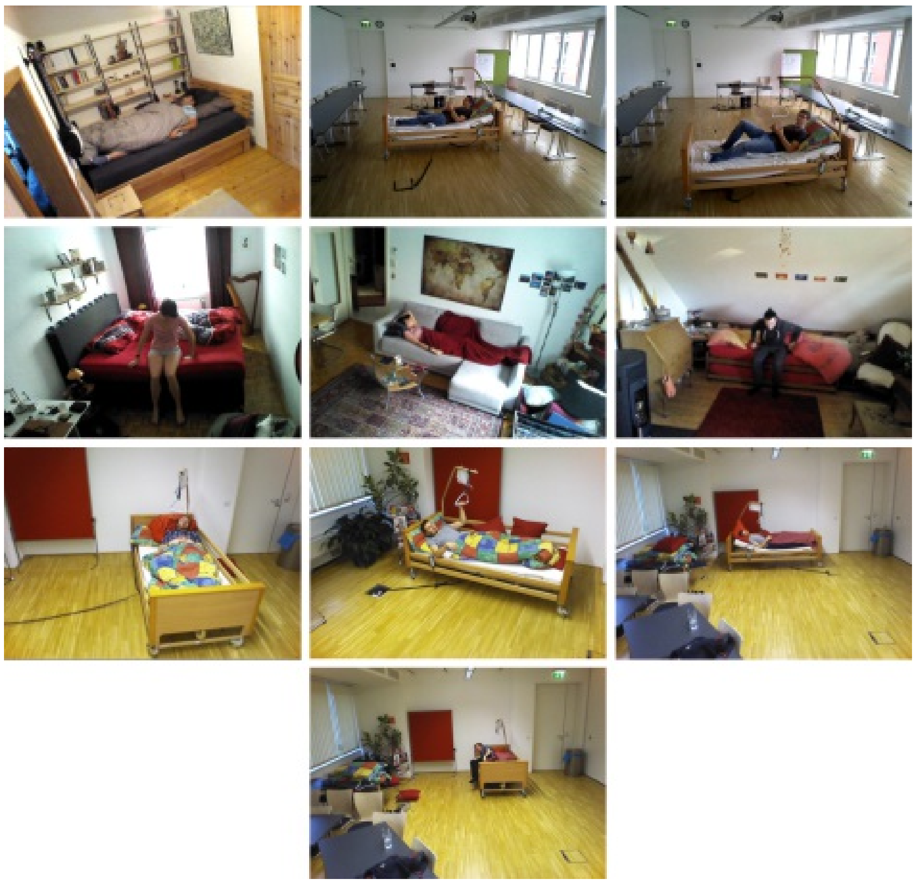


Figure 3.4: Demonstration of camera position and actors who are getting up in different settings. RGB images are used in this figure for demonstration purpose only.

There is a structural difference between depth and RGB images that would cause an incompatibility problem when transferring the knowledge from a network that is trained on RGB images. A depth image is a single-channel image where each pixel represents how far an object is from the camera. RGB images on the other hand have three-channels where each pixel is the combination of three colors of red, green, and blue. Due to this difference, depth images are not compatible for training/testing on networks that have been pre-trained on RGB images. There are two possible solutions of this issue: the first option is to change the network so that it becomes compatible with single-channel images. This, however, is not a suitable solution because changing the layers (especially early layers) could have a ripple effect on what the network has learned and damage the transfer learning process. The second option is to convert the depth images from single-channel images to three-channel images in order to have the same input structure as RGB images. One way to achieve this is to convert each pixel of a depth image into an RGB color. This is not suitable because it could lead to information loss since the recorded pixels are distances, not colors. Another way to convert the depth images is to take the single recorded channel of depth image, copy it twice and combine the copies to produce a three-channel image. Thus, the depth image as the input of the network becomes a three-dimensional tensor mimicking an RGB image. The three dimensions of the tensor are equal to each other and represent a single-channel depth image.

Labeling

Labeling the extracted images is done differently, depending on the classification or object detection method. In classification, the whole image is simply assigned to a label. In object detection, an object inside the image is specified by drawing a bounding box around it and assigning a label to it. In order to distinguish the images from one another and label them correctly, it is necessary to formally define getting up behavior of a single person to determine what is considered to be a sitting or standing position and what is considered as a lying position. Each complete getting up motion contains consecutive motions that are captured in a series of consecutive frames. Humans could recognize a getting up motion by looking at each individual frame without any prior knowledge about the motion. Therefore, each frame is labeled individually and the datasets contain no information about the order of frames.

Getting up is defined as the motion of the human body that transfers the body from a horizontal position on the bed to a vertical position to maintain body balance for standing up. In order to recognize a person getting up, a network has to be able to distinguish between lying and standing or sitting. Hence, each frame is labeled with one of two labels: "lying" or "standing". An image is labeled "lying" when a person is present on the bed in horizontal resting mode. Once a person passes a critical point in which the feet are off the bed on a downward-moving trajectory and simultaneously the upper body part moves from the mattress in an upward-moving trajectory, the image is labeled "standing". Both labeled situations and the critical point are demonstrated in Figure3.5.

For training a model and evaluating its generalization for unseen future data, the data should be divided into three smaller datasets: training set, validation set, and test set. The training set contains all images used for training in forward pass and backward pass in every Epoch. The validation set is used to estimate how well the network is generalized in the same Epochs and tunes the hyperparameters, whereas the test set is used to estimate how well the network performs after being trained. Noteworthy, all images in each of the three datasets must be distinct from one another. However, all the frames in a depth video sequence are similar, because they are acquired with 30 frames per second, which means 30 depth images per second are generated and within a second the setting of a room stays mostly the same. Also, an activity like lying on a bed is a long term activity in which the person mostly does not move. Therefore, taking all frames would overfit the network with the repetition of similar images. To avoid overfitting, instead of extracting all the frames, only every 5th frame is extracted (6 frames per second instead of 30). In order to make the datasets distinct, the images are should be divided based on their the field of views.

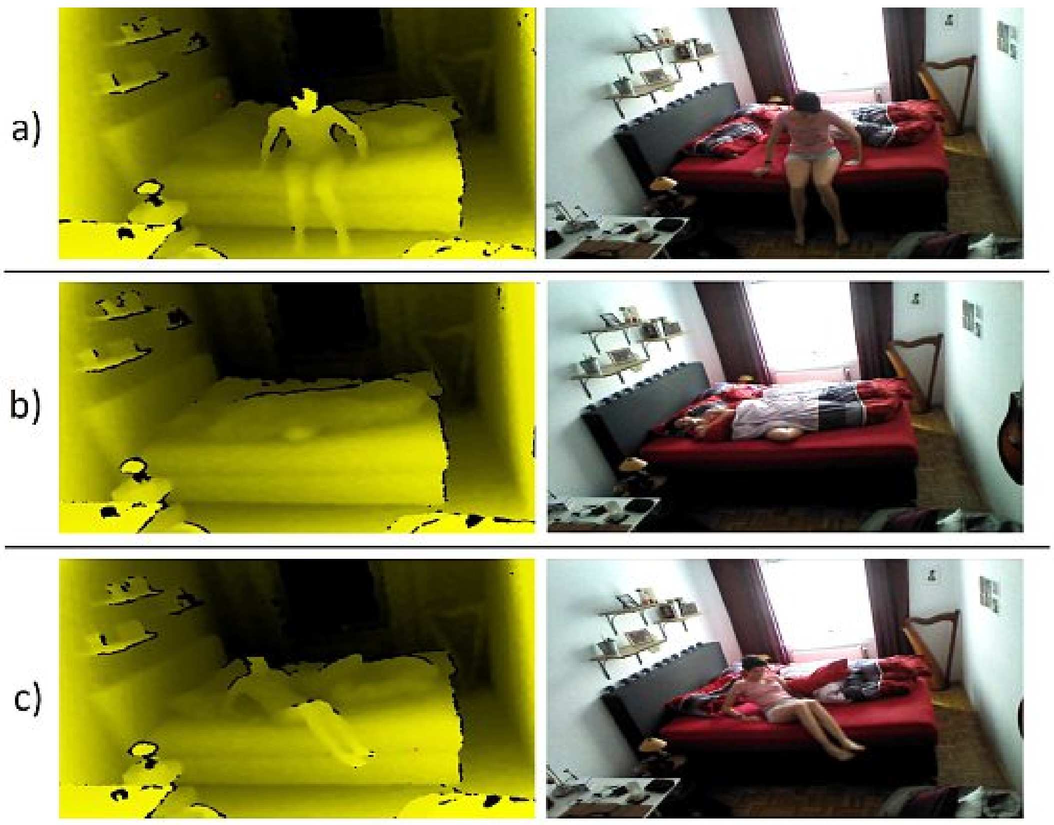


Figure 3.5: Example of an image which is a) labeled "standing" where a person is sitting on the edge of the bed and b) labeled "lying" where a person is lying on a bed; c) shows the critical point at which the position changes from lying to standing.

Because the amount of data available for getting up behavior modeling is limited, the data is divided only into training and validation sets and the test set is omitted. The reason behind this is that the test set should ideally simulate unseen future data and if the size of the test set is fairly small, it makes the evaluation of the model uncertain. Plus, increasing the size of validation and train sets benefits the model performance.

Each of the training and validation sets is prepared differently based on the classification or object detection task they are utilized to solve. In classification, each label is assigned to a folder and the images of each dataset are labeled simply by copying each extracted frame into the corresponding folder. For object detection, each depth is labeled with bounding boxes. Each bounding box is a rectangular shape that contains information about the position and the object inside it. An image can have more than one bounding box, each of which points to different objects inside the image. However, here, each scene contains one bed within a room with one person lying on it. Therefore, there is one bounding box per image. The labeling is done with the help of RGB images of the same recording. As depicted in figure 3.6, for labeling a depth image, first the bounding box with its label is manually added to the RGB image of a frame, and then the Extensible Markup Language (XML) information about the bounding box is produced and extracted. Next, the RGB image is replaced by the depth image of the same frame. Each XML file contains the following items: filename of the depth image, path to the image, coordinations for the bounding box and its label. Labeling and setting bounding boxes is performed by the program "LabelImg"[3] in PASCAL VOC format.

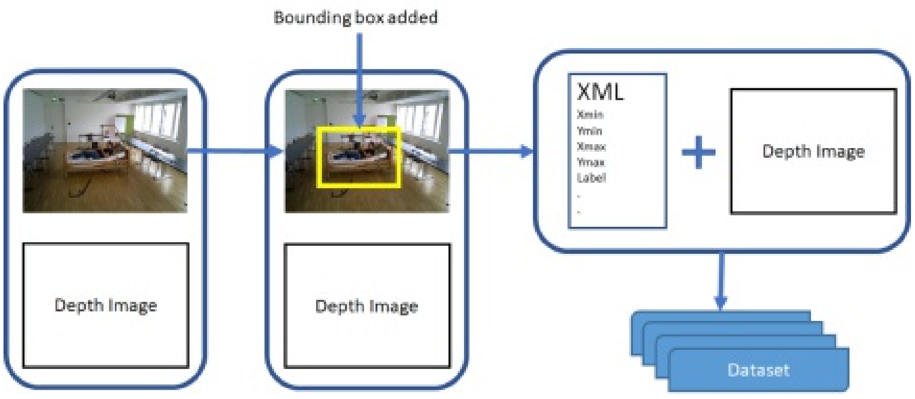


Figure 3.6: Demonstration of labeling bounding boxes of depth images with the help of RGB images. At each frame the RGB and depth images are taken and the bounding box is added to the RGB image. Then, information about the bounding box is stored in an XML file. The pair of XML and depth image is used as input for object detection.

Data Augmentation

In total, 12 different rooms are captured and Data Augmentation techniques are used to artificially enlarge the datasets.The augmentation techniques used for classification and object detection are listed below:

- **Classification:**
  - Training set:
    - Random crop: Cropping random size of 0.08 to 1.0 from the original size with the same aspect ratio. This crop at the end is resized to the size of 224×224 pixel.[4]
    - Random horizontal flip: Randomly horizontally flipping a defined proportion of the training set.
    - Random Rotation (45): Randomly rotating the whole image in 45 degrees in clockwise or counterclockwise direction.

[3]`https://github.com/tzutalin/labelImg` (visited on 04.11.2020)

[4]`https://pytorch.org/docs/stable/_modules/torchvision/transforms/transforms.html#RandomResizedCrop` (Visited on 04.11.2020)

- Validation set:
  - Resize: resizing the images to the size of 224×224 pixel.
  - Center crop: Cropping the image from center to the desired output size.

- **Object detection:**
  - Train set:
    - Resize: resizing the images to the size of 640×480 pixel.
    - Rotate: Rotating the image and its bounding box by 45 and -45 degrees.
    - Translation: Shifting pixels inside the image in two different axis of X and Y.
    - Crop: Cropping pixels at the sides of the image.

Data augmentation is not required for the validation set, because this dataset represents real unseen future situations. The resizing that is used in data augmentation is based on network architecture. In classification, the ResNet is used that accepts a certain input size due to the existence of a fully connected network as a classifier. Therefore, validation and train sets use resizing as a form of data augmentation which can be seen as data preparation. In object detection where the faster R-CNN is used, images do not need to have a certain input size even though the network contains two fully connected layers as a classifier in the last layers. The faster R-CNN uses a special pooling layer that resizes every feature map tensor to a specific size before passing it to the fully connected layer. However, all the images are resized to a fixed size of 640×480 so that the network performs on the unified input size.

In order to normalize the data, the mean and standard deviation of the entire dataset (either train set or validation set) for each channel needs to be calculated. One way to calculate them is based on the images sizes. If the images have the same size, the mean and standard deviation is equal to the mean of each image means and standard deviation. If the images have different sizes, the same principle but with a weighted mean can be used. Another way to calculate them per mini batch instead of per image. The mean and standard deviation is calculated per mini batch of the size 2048 (The bigger the size of mini batch, the more accurate the standard deviation) and averaged over throughout the entire dataset. The calculated standard deviation and mean differs from what they actually are. However, this difference is small that does not have an impact in practice.

3.2 Methods

For each method of classification and object detection, a network architecture that is compatible with depth images and has the desired output is required. This is achieved by taking a network trained on RGB images and modifying it in order to be compatible with the output desired for detecting getting up behavior for both methods of classification and object detection. Then, for each network, the training process needs to be defined in order to not only tackle the task of detecting getting up behavior but to also exploit the effectiveness of using transfer learning techniques in this context.

3.2.1 Classification

Architecture

In classification, there are state-of-the-art CNN-based architectures available such as ResNet [34], inception [108, 109], Alex Network [49], VGG [102], dense network [39] and squeeze network [40]. Each of these architectures has been trained on publicly available datasets such as ImageNet. Adjusting and modifying the neural networks are not automatic tasks and they are unique to each model.

Any of the aforementioned networks could have been a possible choice for this experiment. All of these networks can be used in both classification and object detection and they use a different technique to overcome the vanishing gradient descent problem [44]. This is a problem that occurs during training when each neuron receives an update based on a partial derivative of loss function backpropagated to the neuron. When a network is deep enough the gradient update becomes so small that it has almost zero effect to change the values in the earlier layers of the network. Out of these networks, ResNet is chosen as the backbone CNN because apart from the mentioned advantages, the ResNet also solves the degradation problem which is a problem caused by the network depth. As the networks increases in depth, the accuracy gets saturated which then causes the accuracy to degrades. Basically, adding more layers to the suitable models leads to higher training errors [34]. In addition, the ResNet is available in various number of layers from ResNet18 to ResNet152.

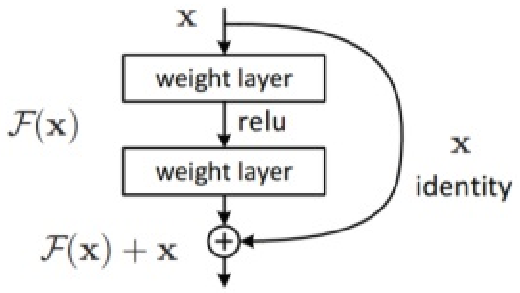

$\mathbf{x}$
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$

Figure 3.7: Illustration of a single residual block that is proposed in ResNet [34].

ResNet introduces a layer called **residual blocks** to overcome the issues. As depicted in Figure 3.7, a residual block is a layer that takes the input activation and fast-forwards to the deeper layer (skip connection) or, in mathematical terms, it uses an identity function which always returns the input as the output to preserve the gradient, since the gradient of the identity function is 1. Each residual block has a connection that can skip layers and prevent the vanishing gradient problem. Therefore, it is possible to stack residual blocks as needed to go deeper into complex problems without facing the vanishing gradient problem. More detailed information about identity function and residual blocks can be found in publications written by *Kaiming He* [34, 35].

Among the available depths of ResNets, the ResNet18 is chosen as the backbone for this experiment because it is lightest among all ResNet networks while the overall performance difference is less than 5% according to [110]. ResNet18 is trained on the 1000-class ImageNet dataset and has exactly one output neuron for each of the classes. Since the depth images are converted to three-channel images in preprocessing, the input of the network does not need to be modified. The output layer of the ResNet18 (classification part) on the other hand needs to be adjusted. It takes the extracted feature from CNN part as input and generates "lying" and "standing" classes as output. The ResNet18 architecture contains a convolutional layer with a max pool, 8 residual blocks that are divided into 4 different layers followed by an average pooling, and 1000 neurons as output layer that are fully connected to the previous layer. The network architecture is depicted in figure 3.8. To modify the network, the current fully connected output layer of the trained network that is directly connected to the extracted features by CNN is removed and two new neurons as the output with random weights and biases are attached as demonstrated in Figure 3.8.

The network ResNet uses a fully connected network at the last layer that only takes a fixed-size feature map. The average pooling before this layer outputs the needed fixed-size feature map and it is indifferent to image size. However, because in this experiment, the pretrained network is used, the input depth images must have the same input size that the ResNet was originally trained on which is a squared image of the size 224×224 pixels. Thus, all the depth images in the dataset which have the size 640×480 pixel are resized before being fed into the network as a part of data augmentation. The input tensor for the network is a $3 \times 224 \times 224$ and the size of the input tensor changes as it is passing through the network. The output sizes after each layer are listed below (the softmax layer is omitted):

- First Convolution and max pooling: $64 \times 112 \times 112$
- Layer 1: $64 \times 56 \times 56$
- Layer 2: $128 \times 28 \times 28$
- Layer 3: $256 \times 14 \times 14$
- Layer 4: $512 \times 7 \times 7$
- Average Pooling: $512 \times 1 \times 1$
- Output Layer: 2

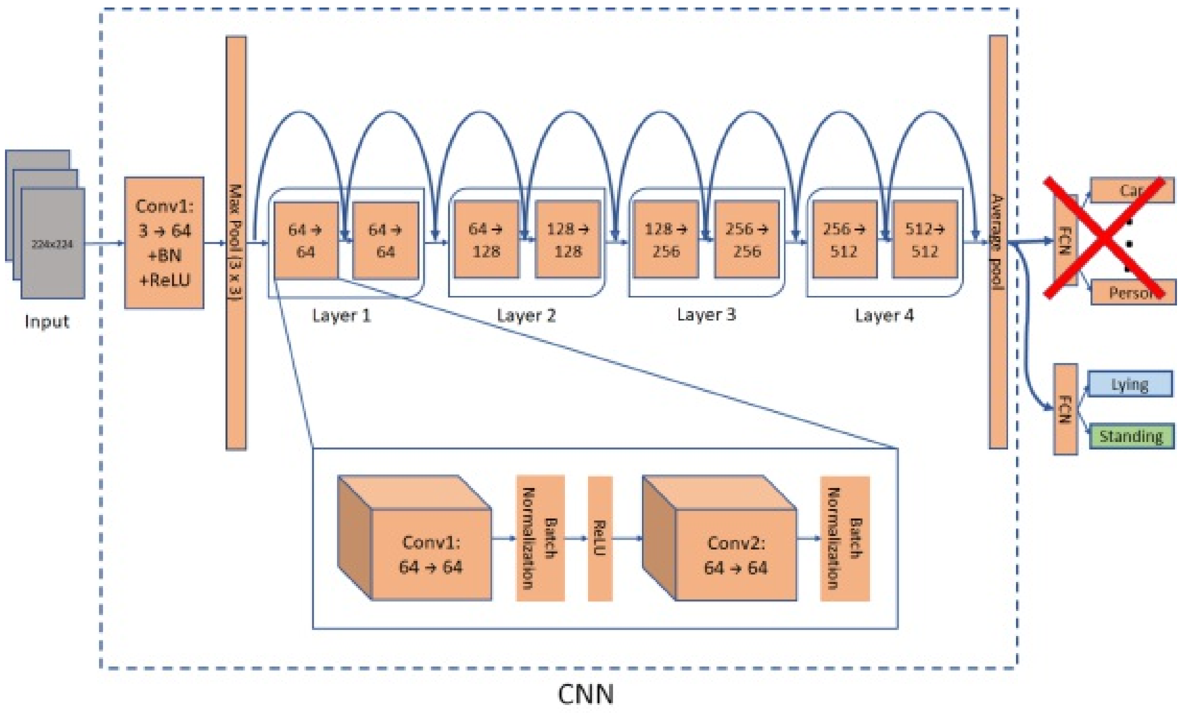


Figure 3.8: The architecture of modified ResNet18 that is used for classification. It contains two parts:CNN and a fully connected network. The CNN part of the ResNet18 includes 1 convolutional layer followed by 4 layers of residual blocks where each of them has two convolutional layers that are followed by a batch normalization layer and ReLU. After the fourth layer, the CNN outputs the features to the fully connected network with an average pooling in order to have a fixed output size. The last layer takes the feature maps and outputs a vector of size 2 where it uses a softmax layer to output the probability of the two class labels.

The abstract way of looking at this network is that the network takes an image and extracts features from the original image. As it propagates forward in the network layer by layer, it tries to extract higher-level features until it reaches a tensor of the size 512. At last, based on the extracted features, a simple fully connected network classifies if a person is standing or lying on a bed.

Training

The modified classification network is trained in three different manners to exploit the effectiveness of the training process and transfer learning:

1. Freeze and fine-tune
2. No Transfer Learning

3. Only fine-tune

In both fine-tuning and transfer learning approaches, the whole modified network is trained without any restriction. In the fine-tuning, all of the parameters inCNN part are initialized with the values transferred from the trained ResNet18 whereas in the no transfer learning approach, all of the network parameters are initialized randomly. The freeze and fine-tune approach has a two-phase training: In the first phase, the transferred part of the network is frozen in order to avoid changes during training and only the newly added part is trained (feature extraction mode). In the second phase, the frozen part of the network is unblocked from training and the whole network participates in training without any restriction (fine-tuning mode). The idea behind this approach is that the newly added neurons and connections have random weights and biases and this makes the network not knowing how the output should look like while the CNN part is extracting features. Thus, in the first phase, the newly added output layer learns how the output of the network should look before the whole network starts to get trained. In other words, the new output layer has no understanding of how to classify based on the extracted features due to its random initialization. In training, this might lead to unwanted alterations in the transferred part of the network and/or induce loss of what CNN has previously learned.

For training of all the models, a backpropagation algorithm with a cross-entropy loss function is used. Also, instead of using a constant learning rate during the training process, its value is reduced by *gamma=0.1* at every *n=10* Epoch. This is known as step-wise decay or adaptive learning rate which helps the optimizer to converge faster with a higher success rate as previously suggested by Bengio *et al.* [9].

In training, at the end of every Epoch, the validation loss obtained from the current Epoch is compared to the best previous validation loss. If it is smaller, then the weights and biases of the network from the current Epoch will be saved and the best previous validation loss will be replaced by the current Epoch validation loss. If it is bigger, then nothing happens and the training continues. This functionality leads the training process to return the best-achieved model parameters of the entire training instead of the parameters of the last training Epoch. In addition, this feature enables the use of early stopping, which is used to avoid overfitting during training. The threshold for early stopping is defined by the step size of the learning rate because if the network is not generalizing and starting to overfit, the learning rate would get reduced at least one more time and gives the optimizer another chance to converge if possible before the training stops by early stopping. In the fine-tuning and no transfer learning approaches, the hyperparameters are set before the training process. However in the freeze and fine-tune approach, before the start of the second phase, all the hyperparameters that are changed during the last phase such as learning rate, early stopping counter, optimizer are reset to its original value after the first phase is finished. The exact values for hyperparameters used are listed in the next chapter 4.1.

3.2.2 Object detection

Architecture

For object detection, Faster R-CNN with ResNet18 as its feature extractor is used. This network has previously been trained on the COCO dataset and is flexible in choosing different CNN networks as feature extractors [93]. For the sake of simplifying comparison between classification and object detection, ResNet18 is used as backbone CNN for both tasks.

The general idea behind Faster R-CNN is to take the feature maps from the CNN part and use it as the input for the RPN network that outputs/proposes boundary boxes (around 2000 proposals). For each proposed boundary box, the parts of the feature maps that correlate to the proposed boundary box are then extracted with the help of RoI pooling. Afterward, the classifier assigns a label to the proposed boundary box based on the extracted features. RPN works with predefined anchor boxes in order to propose the boundary boxes and has two tensors as output: the first is the likelihood score which indicates how likely an object (regardless of the class labels) exists in this boundary box and the second is the tensor that contains the offset coordinates for the boundary boxes which will be applied to anchors to get the final coordinates. The output of the classifier layer is similar to the output of RPN except that one of the tensors assigns a class label to each box with the probability instead of the likelihood of the existence of an object.

To modify the network to fit the purpose of object detection in detecting "standing" and "lying" in the context of getting up behavior, the changes do not only need to be made on the output of the classifier layer but also on the output of the RPN network. This is because the predefined anchor boxes from RPN are different than what it is suitable for this experiment and the RPN is trained to detect objects that are in the COCO dataset. Thus, in order to build the network, three parts of Faster R-CNN have to be built/modified, namely: backbone CNN, RPN and the classifier. The architecture of the network and the three parts are illustrated in Figure 3.10.

Part I: ResNet18 is chosen as the backbone for this network. The feature map produced by ResNet18 is $512 \times 1 \times 1$ for each single image. This is a single vector of the size 512 which is not suitable for Faster R-CNN, because the RoI pooling layer is used to find the region of interest inside the feature map. Thus, if each dimension of the feature map has only one element 1×1, then all the proposed boundary boxes have the same set of features regardless of where each box is located. One solution is to cut the network before the average pooling where the output tensor of the CNN part is $512 \times 7 \times 7$ so that the RoI pooling layer can find the region of interest inside the feature map. However, this approach introduces another problem with predefined anchor boxes. One of the advantages of using CNN is that each neuron at a deeper level has a larger receptive field. By calculating the layers for ResNet18, each neuron at the feature maps has a receptive field of a 435×435 [34]. The receptive field can be larger than the input because the network uses residual connections [7]. This implies that every anchor size (even the size of 1×1) applied to these feature maps still defines a boundary box as big as the input image. Therefore, training such a network requires multiple sizes of an object for training so that the network can gain an understanding of various sizes of the object. Again, this would borderline the problem with the problems of gathering data, especially in detecting getting up behavior where there is no dataset publicly available. Still, there is a possibility to cut further into the CNN until the size of the feature map and its receptive field is suitable for applying different anchor sizes where anchor boxes can search through the image at various window sizes. However, cutting deeper into the CNN reduces the capacity of the network because going deeper is the key to extracting high-level features.

Tsung-Yi Lin *et al* introduced a novel network called Feature Pyramid Network (FPN) where it maintains the capacity of the CNN while using different anchor size at various scales[58]. The FPN uses multi-scale feature maps to detect different sizes of an object. Figure 3.9 depicts how RPN works. The image data flows in two directions: Bottom to top and top to bottom. On the left side of the figure, the image data flows through the CNN and gets smaller in resolution but higher in dimension (high in terms of semantics). On the right of the figure, the FPN layer takes the last feature map from CNN with high semantics to solving a task and increases the resolution. There is a lateral connection between FPN and feature maps that merges the higher resolution of feature maps with the current feature maps. This helps to improve the detection and also functions as a skip connection similar to what is used in ResNets. More detailed information about FPN can be found in [58]

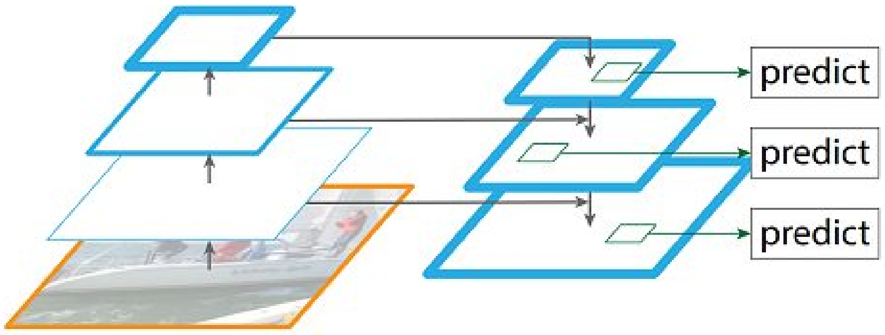


Figure 3.9: The network illustrates the architecture of FPN. On the left-hand side, the image data flows from bottom to top in order to extract the features (thicker borders represent higher semantics), and on the right-hand side, it uses the extracted feature to construct a higher resolution with higher semantics [58].

Figure 3.10 demonstrates the whole modified network for detecting getting up behavior. The left side of the figure shows the architecture of the first part. ResNet18 is used as the CNN for extracting high-level features for the FPN and it has four residual blocks that are color-coded in Figure 3.10 where each is named as C2, C3, C4, and C5 for the convolutional layers 2, 3, 4, and 5 respectively. Each of these layers has a corresponding layer in the top to bottom part of FPN where they are called M2, M3, M4, and M5 respectively. Furthermore, there is an extra layer called M6 that is the same as M5 with a max pool layer on top. Each of the M2, M3, M4, M5, M6 layers outputs a feature map called P2, P3, P4, P5, P6 respectively that have different sizes but equal semantics.

The last layer of ResNet18 outputs a tensor of the size of $(512 \times 15 \times 20)$. The actual output size of the ResNet18 is different than what is described in the corresponding publication [34]. This is because the input of the depth images has the size of (640×480) and is not similar to the fixed size of (224×224) that is assumed in the publication. After passing the last layer of ResNet18, the extracted feature map is passed on to a 1×1 convolution to reduce the $C5$ from 512-d to 256-d without changing the resolution of the feature map. Hence, the size of the feature map is $(256 \times 15 \times 20)$ at $M5$. This is the first feature map (P5) that is passed to the RPN as an output. Then, as it goes to the next layer, it upsamples the tensor by the factor of 2 using nearest neighbor upsampling. Then it merges element-wise with the $C4$ (the related feature map) which is also passed through a 1×1 convolution filtering to reduce its dimension to 256-d. In other words, the tensor $(256 \times 15 \times 20)$ is upsampled to $(256 \times 30 \times 40)$ and then is simply added element-wise with the output of lateral connection which has the same size. This is the second output of the network (P4) that is passed to RPN. This is repeated twice more until there are four feature maps of P2, P3, P4, and P5 that correspond to four layers of ResNet18. In addition, there is an extra output that is the result of a simple 2D max-pooling (downsample by stride two) applied on the $M5$. This feature map is created, for covering larger anchors sizes such as 512-pixel and higher [58].

In all the layers inside the FPN, frozen batch normalization is used instead of batch normalization, because in object detection the size of mini-batches is small. Using normal batch normalization leads to poor mini-batch statistics which degrades the performance [34].

Part II: In this layer the RPN takes all the feature maps and proposes RoIs with the help of predefined anchors. Originally without FPN, the RPN takes a single feature map and perform a 3×3 convolutional layer succeeded by two neighboring 1×1 convolutional layers that are used for classification and boundary box regression with the respect to predefined anchors. The anchors are defined based on their size and aspect ratio. By default Faster R-CNN uses three sizes and three aspect ratios which yields $k = 9$ anchors. The anchor sizes and aspect ratio are fixed and should be set before training or inference. Therefore, they are considered hyperparameters that can affect the outcome of the network.

For detecting getting up behavior, it is assumed that the size and position of the bed inside the frame is not fixed. Therefore, the size of the boundary box can be varied and the network should be able to detect them all. Because the size of depth images is fixed (640×480) pixels, the maximum possible anchor size is (512×512) pixels. The sizes bigger than 512 pixels would be 1024 pixels that contain the whole image which is a classification task by considering the size of depth images. Figure 3.11 depicts the statistics for all boundary boxes in both the train and validation sets. Plot a) in the figure demonstrates the histogram of aspect ratios in which it is clear that more than 95 % of the aspect ratios lie within the interval of $[1, 2.5]$. Plot b) demonstrates the

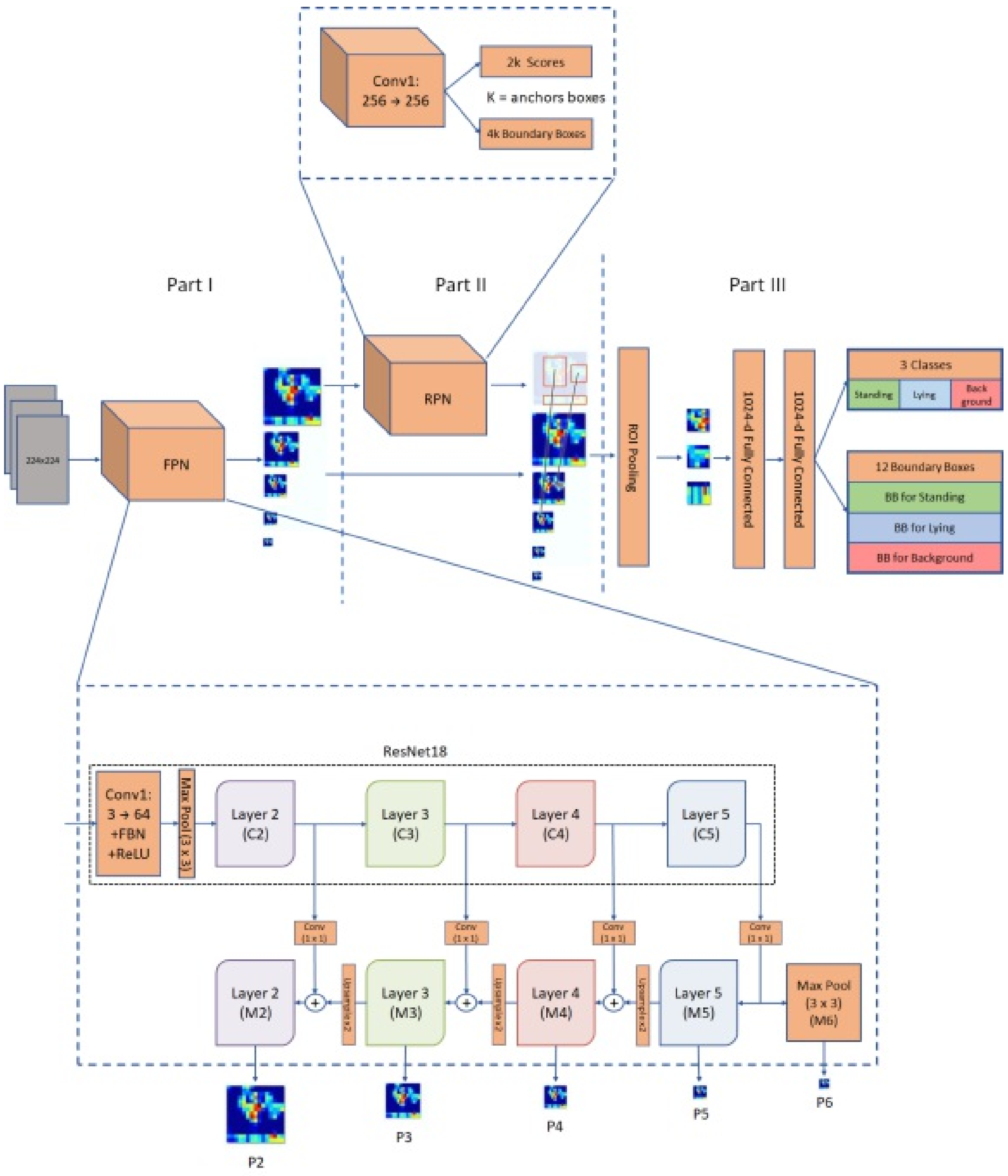


Figure 3.10: The plot illustrates the modified object detection network which contains three different parts: FPN as the backbone, RPN, and the classifier. The backbone of the network uses ResNet18 with FPN that outputs multi-scale feature maps where each feature map has high semantics (high extracted features) but in different sizes [58]. The second part includes the RPN that takes each of the feature maps and proposes boundary boxes. In the third part, with the help of RoI pooling, the relevant feature for each proposed box is extracted in a fixed size and passed forward to a fully connected layer for classification. The output of the network has two tensors: classes of size 3 (one for each class label + one for the background) and offset to boundary boxes for each class label.

histogram of the square root of the area of the boundary boxes in which more than 97 % of the boundary box areas lie within the interval $[127, 527]$. Hence, it is sensible to assume that the anchors in future situations would also lie within these intervals. Plot c) in the figure demonstrates a scatter plot of the aspect ratio and boundary box area and the red rectangle inside the plot shows the intervals mentioned above. It is clear that the chosen intervals do not cover every possibility of occurred boundary boxes and possibly every new situation/room might have different sizes of boundary boxes. However, they cover more than 95.0 % and 97.2 % of all the aspect ratios and sizes available in the datasets. Therefore, the values of ($Size = \{32^2, 64^2, 128^2, 256^2, 512^2\}$ and $AspectRatio = \{1:2, 1:1, 2:1, 2.5:1\}$) are set as the anchor boxes values based on the mentioned intervals. Hence, there are 20 anchors per spatial area.

RPN is adapted to be able to work with FPN. The RPN is applied to all levels of the FPN with 3×3 convolution followed by two 3×3 convolutional layers. However, because the 3×3 slides over all areas of all levels of the pyramid, it is not necessary to use all different scales of anchors in terms of size over every level of the FPN output. Each anchor size is assigned to a single level of FPN output. Formally, the anchors are defined to have areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ pixels covered on $\{P_2, P_3, P_4, P_5, P_6\}$ and all sizes are used in multiple aspect ratios of $\{1:2, 1:1, 2:1, 2.5:1\}$in each pyramid level. The reason to assign the smallest area to the largest feature map is that the higher (smaller) the size of the feature map, the bigger is its receptive field. The output of the RPN is a set of proposed boundary boxes per each level of pyramids and passes to the next part of the network.

Part III: This part takes the RoI proposals from RPN and the outputs from FPN then produces the final results. Originally, the RoI pooling is used to extract features from a single feature map in a fixed size. The RoI for this network is needed to extract the related features from different feature maps. So it is necessary to expand the capability of the RoI so it can work with multi-scale feature maps. Therefore, the RoI adapts a new strategy to decide which feature map is suitable for extracting the features for each proposed boundary box. The strategy starts from the feature map $M6$ (smallest in terms of resolution) and takes a proposed RoI. Then the RoI is scaled to the input image if the proposed RoI becomes smaller than half of the original image size (640×480), it uses a finer resolution level (or bigger in terms of feature map resolution) until the RoI scale becomes bigger than half of the input image size. The exact formula can be found in [58].

After the feature map is chosen, the RoI pooling is used to extract 7×7 relevant features and passes them to the next two fully connected network layers as depicted in Figure 3.10. In the original ResNet18, the last convolutional layer (conv5) is adopted to extract a 7×7 feature but in the FPN this is already being used to construct the feature pyramid. Hence, the RoI pooling outputs the same size features to attach two hidden 1024-d fully connected layers. These layers are initialized randomly. Afterward, these layers are connected to two sibling layers of classification (softmax) and boundary box regressor that finalize the output. Unlike the similar layers in RPN, these last two layers are specifically designed to output prediction for detecting getting up behavior. The classifier

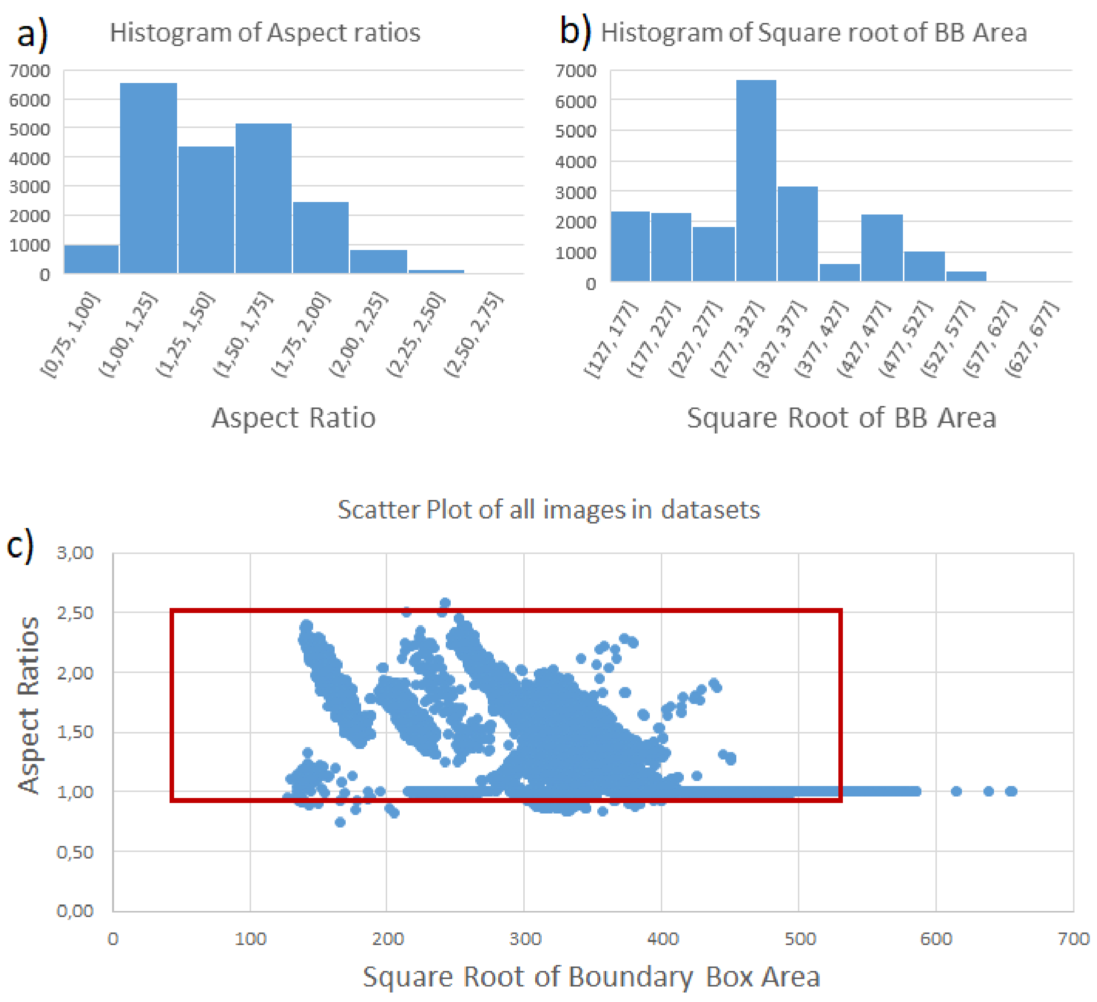


Figure 3.11: This figure demonstrates statistics of boundary boxes available in train and validation sets. In a), a histogram of all aspect ratios of all boundary boxes is depicted. b) depicts the histogram of all square root of all boundary boxes. The square root of the area is chosen because it is easier to understand the area of a boundary box based on the number of pixels that exist on the side instead of the number of pixels inside the area. c) demonstrates the scatter plot between the area of boundary box and aspect ratio. The red rectangle inside this plot shows the area that the chosen anchors ($Size = \{32^2, 64^2, 128^2, 256^2, 512^2\}$ and $AspectRatio = \{1:2, 1:1, 2:1, 2.5:1\}$) for the modified Faster R-CNN network is able to cover.

head has the size of three, one for each class label of "standing", "lying" and "background". The boundary box regressor has a size of 12 which is three times boundary box locations of size four, one for each class label.

After building these three parts, they are assembled to build an End-to-End network that is trained all at once. This network is a collection of different parts that are designed separately and combined. Unlike the classification network which needs a modification of the original pretrained ResNet18 to be used for this experiment. In this network, part I has the possibility to be used as a pretrained network and the other two parts are built based on the requirements of this project and initialized randomly.

Training

The modified Faster R-CNN network is trained in three different approaches similar to classification: Freeze and fine-tune, No transfer learning, and Only fine-tune. However, the non-CNN part of the network is larger since it has the classifier and RPN. This network is trained with different hyperparameter values.

The only difference between classification and object detection in terms of the training process is how the loss of the network is calculated. The object detection network has four different losses, two of which are for RPN network and the remaining two are for the final output of the network. Each uses a different loss function, as stated below:

- RPN Objectness: Binary Cross Entropy with logits
- RPN Boundary Box Regressor: L1-loss
- Classifier: Cross Entropy
- Boundary Box Regressor: Smooth L1-loss

The final loss for the training is the sum of all four. However, in this project, the loss used for training is a weighted loss where the losses are multiplied by a constant before summing them up. This is, because the sum of all losses drops below 0.01 after one Epoch which renders the training process unable to affect any changes in the network and consequently the network does not learn/change to reveal any increase in validation accuracy. Therefore, weighted loss is used to magnify the error rate (loss) for the backpropagation to be able to make changes especially in the early layers of the network. Each of the four mentioned losses is multiplied by a constant. Therefore, they are treated as hyperparameters and selected based on the grid search. The weighted loss that the object detection network trained with is calculated as follows:

$$loss = (\text{loss classifier} * 10) + (\text{loss box reg} * 3) + (\text{loss objectness} * 5) + (\text{loss rpn box reg} * 1) \tag{3.2}$$

3.3 Workflow

In machine learning, it is common that the developer/researcher repeatedly makes changes to a model until it reaches the desired outcome[116]. This iterative workflow or human-in-the-loop approach is used for producing the desired results for this experiment. An iterative workflow is a method in which all processes, data, and decisions inside the workflow cycle can be executed multiple times until a satisfactory result is obtained. This is useful in deep learning since training and evaluating a model requires repetitive optimization on hyperparameters and the stochastic behavior in the learning process might lead to different results [117].

Figure 3.12 shows the customized iterative workflow for this project. It starts by formulating the problem where the goal of the project, the process of how to reach the goal, and the tools that are required are defined. Then the cycle inside the workflow starts by collecting data. Afterward, all the recorded data along with data received from Cogvis Co. are processed and divided into two datasets: train and validation. Then both datasets are used to construct a model. In the next step, the model is evaluated .

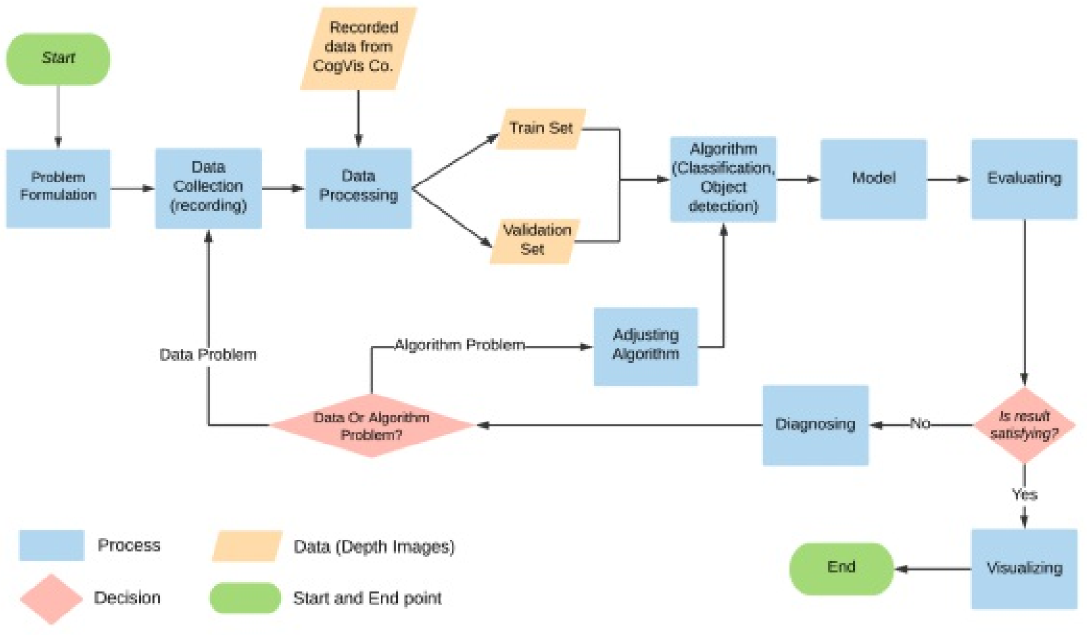


Figure 3.12: Iterative workflow is used in this experiment. The workflow includes the steps that are taken from the point of setting the goal to the point where the outcome of the project is evaluated and visualized. The graph contains a cycle that iteratively goes through different steps and can be broken if the obtained results are satisfactory.

The evaluation process is different from machine learning evaluation since there is no test set available for this experiment. The evaluation here provides information such as the correctness of the algorithm or data, hyperparameter settings, etc. With the information provided by the evaluation process, it is possible to decide if the result is satisfying or not. If yes, then the obtained results are visualized for analyzing the model. If no, it is necessary to investigate if there is a problem in the development of the algorithm or data preparation/collection. After diagnosis, the flow is continued in either collecting a larger amount of data, preprocessing the data or even refining the algorithm.

3.3.1 Software

All the programs are written in Python programming language version 3.6.4 with the help of Pycharm[5]as IDE in which a variety of packages is used for development. The video sequences are recorded with Open Natural Interaction (OpenNI) which records in two formats of RGB and depth and stores the video sequences as Open Natural Interactions (ONI) file. This file can be opened and modified in Python with the same package. All Python packages used are listed and briefly described below:

- **Data collection and cleaning:**
  - PrimeSense (v2.2.0.30.post5)[6]: An OpenNI python package for working with depth cameras.
  - OpenCV (v4.1.1.26)[7]: Unofficial pre-built OpenCV packages for Python.
  - Pillow (v6.2.1)[8]: Python Imaging Library.
  - Matplotlib (v3.1.1)[9]: Python 2D plotting library.
- **Deep Learning:**
  - Pytorch (v1.3.0)[10]: Deep learning research platform based on torch.
  - Torchvision (v0.4.1)[11]: a package consisting of popular datasets, model architectures, and common image transformations for computer vision.
  - TensorBoard (v2.0.2)[12]: Visualization and tooling needed for machine learning experimentation.
- **Evaluation and saving results:**
  - Pandas (v0.25.3)[13]: Python Data Analysis Library.
  - Numpy (v1.7.17)[14]: fundamental package for scientific computing with Python.

[5]`https://www.jetbrains.com/pycharm/` (visited on 04.11.2020)
[6]`https://structure.io/openni` (visited on 04.11.2020)
[7]`https://opencv.org/` (visited on 04.11.2020)
[8]`https://github.com/python-pillow/Pillow` (visited on 04.11.2020)
[9]`https://matplotlib.org/` (visited on 04.11.2020)
[10]`https://pytorch.org/` (visited on 04.11.2020)
[11]`https://pytorch.org/docs/stable/torchvision/index.html` (visited on 04.11.2020)
[12]`https://www.tensorflow.org/tensorboard` (visited on 04.11.2020)
[13]`https://pandas.pydata.org/` (visited on 04.11.2020)
[14]`https://numpy.org/` (visited on 04.11.2020)

– OS[15]: Miscellaneous operating system interfaces.

3.3.2 Hardware

The computational power utilized in all parts of the project was provided by Vienna Scientific Cluster (VSC)[16], a scientific cluster that provides supercomputer resources and corresponding services to Austrian universities. It contains over 2800 Linux compute nodes in which additional hardware such as GPUs is available. The nodes this project is computed on are listed in 3.1

Component	Model
Processor(CPU)	Intel Xeon
Working Memory(RAM)	32 Gb
Graphic processor(GPU)	NVIDIA V100 TENSOR CORE GPU with 16 Gb
Operation System	Centos Linux 7

Table 3.1: Hardware specifications

[15]`https://docs.python.org/3/library/os.html` (visited on 04.11.2020)
[16]`http://www.vsc.ac.at/` (visited on 04.11.2020)

CHAPTER 4

Results and Discussion

The modified networks of classification and object detection are trained on the created train set and analyzed on validation set during and after training in order to gain an understanding of the model's performance. For each model, the output performance on validation sets is shown to determine 1) the behavior of the model during training and 2) at which Epoch the network has the highest performance.

Following training, each model is analyzed based on its task: in classification, the model output is examined with a confusion matrix to visualize the performance of the model for each class label. All incorrect detections of the model are analyzed based on the room situations in the validation set. In object detection, the model performance is analyzed based on the two mAPs for a variety of IoUs along with incorrect detection in reference to the room situations.

Overview of the Experiments

Classification and object detection tasks have different hyperparameters and are distinct in terms of evaluation. Each task of classification and object detection is trained until its optimal hyperparameter values are found. Afterward, each model is trained multiple times with the same hyperparameters to find the model which performs best on the validation set. This is due to the element of randomness that exists in training caused by the optimization algorithm and online data augmentation which can lead to different results in each training run.

After determination of the optimal hyperparameters, the models are trained with and without the help of transfer learning techniques with these hyperparameter values in order to exploit the effectiveness of applying transfer learning techniques.

4.1 Classification

In the classification task, the modified neural networks explained in the previous chapter are trained in three different ways as follows:

1. Freeze and fine-tune
2. No transfer learning
3. Only fine-tuning.

In freeze and fine-tune, the model is trained in two phases: First, weights and biases are blocked (frozen) from participation in training as long as the optimizer algorithm is still able to converge. Second, all weights and biases are unblocked (unfrozen) and the training of the whole network with weights and biases gained from the first phase is continued. This model is referred to as **Freeze and Fine Tune** throughout this sub-chapter. In the second approach, the network is initialized with random weights and biases, and no transfer learning technique is used for training the model. This model is referred to as the **No Transfer learning** model from here on. In the third training approach, the whole network is fine-tuned without any restrictions. This approach is referred to as **Fine Tune**.

All of the aforementioned models are compared with each other and their advantages and disadvantages are discussed in the following subsections. The aim of training the same neural network architecture in different manners is to exploit the effects of using the transfer learning technique and how it can affect the model outcome.

Hyperparameters

All the hyperparameters used in classification are listed in Table 4.1 along with their values for the best result. The table does not contain all hyperparameters available for a neural network model. hyperparameters related to architectural design are excluded. The values are selected based on the grid search technique and are kept constant throughout the different training models in the classification task. The number of Epochs are omitted from the table because the early stopping technique is used.

Model 1: Freeze and Fine-tune

In Figure 4.1, the learning curve obtained from training the freeze and fine-tune model is depicted. The entire training took about 40 minutes in 80 Epochs. The accuracy of the training and validation set starts from around 52 % and continue rising to 70 % in Epoch 11. The network is not able to converge further. Training continued without further improvement in the accuracy or loss until Epoch 49. Interestingly, during training until Epoch 49 the validation set achieved approximately 2 % higher accuracy and approximately 0.02 higher loss compared to the training set in Figure 4.1. This effect can

Hyperparameter	Value
Learning Rate	0.001
Batch Size	256
Optimizer	SGD
Momentum	0.9
Weight Decay	0.0001
Learning Rate Step Size	Every 10th Epoch
Learning Rate Change Rate (gamma)	0.1
Loss Function	Cross Entropy
Early Stopping	True
Data Augmentation	True
Normalization	True

Table 4.1: Hyperparameter values applied for training of the classification task

be explained in various ways: Firstly, the size of the validation set is smaller than the training set and the majority of the network is blocked from training. So the network is performing on two datasets, one of which is smaller than the other, but both are derived from the same distribution. Secondly, the training loss has regularization losses such as dropout, weight decay, etc. during training, but none of these is used while calculating for the validation set. Thirdly, even though both training accuracy and loss are calculated during one Epoch, the validation set accuracy and loss are calculated after the training is concluded. Thus, validation accuracy and loss are calculated after the Epoch is done with the training set.

Following the freeze phase, the whole network is unblocked and participates in the training. As depicted in Figure 4.1, at the start of phase two rapid improvements occur in both accuracy and loss and it continues until Epoch 55, where the validation loss and accuracy start to depart from one another. This is considered the optimal point as further training past this point leads the model to overfit. The network saves the weights and biases at Epoch 55 as the most suitable parameters achieved up to this point. However, as no better local optimum for accuracy and loss in the validation set is achieved past this point, the training stopped and returned the parameters saved from Epoch 55.

Model 2: Fine-tune

The modified ResNet18 and its learned parameters are fine-tuned without any restrictions for this model. The fine-tune training encompassed approximately 19 minutes and 34 Epoch in total. As shown in Figure 4.2, the learning curve begins to improve from the start for both validation and training set. At Epoch 8, validation accuracy reaches its maximum value while the training accuracy is still rising. After Epoch 8, the validation set accuracy and loss starts to worsen until the optimization algorithm stops working. The training returns the parameters for the final model at Epoch 8 which is determined

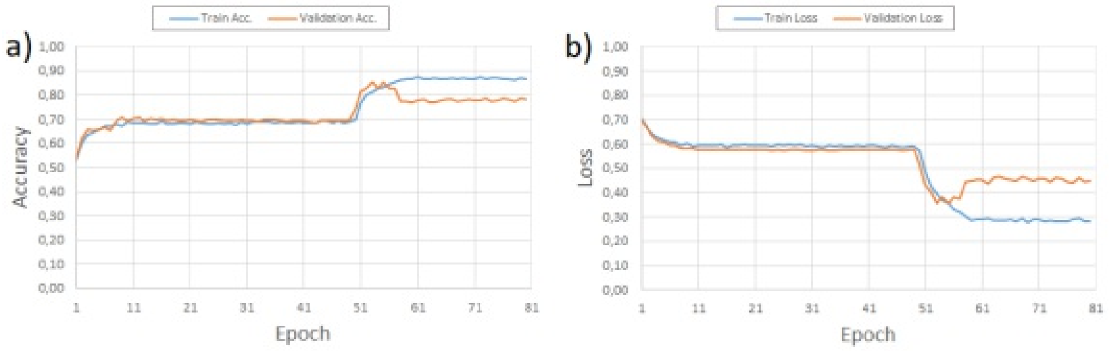


Figure 4.1: The plots illustrate the learning curve of the network during training on ResNet18. The plot demonstrates a) the accuracy and b) the loss of train and validation sets at the end of each Epochs during the training phase.

as the optimal point for training. This model and training process were able to converge at Epoch 8 which is 2 times faster in comparison with the freeze and fine-tune model that reached the local optimum at Epoch 55. However, the freeze and fine-tune model has approximately 4 % better validation accuracy and loss compared to fine-tune.

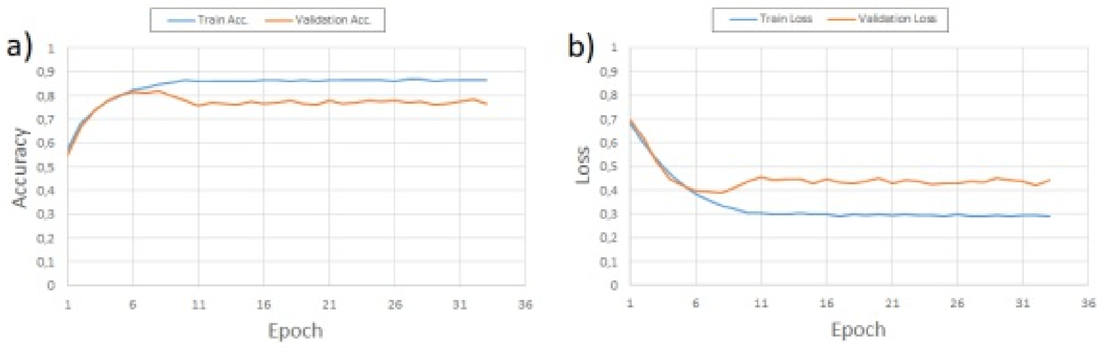


Figure 4.2: Lines in the plot demonstrate the learning curve a) accuracy and b) loss of training and validation sets at the end of each Epochs during training in which the whole modified network is fine-tuned in training.

Model 3: No Transfer Learning

The network used for this model is similar to the fine-tune model but with the difference that its parameters are initialized randomly. Hence, no transfer learning techniques are applied. The training took about 40 minutes and 69 Epochs in total. The learning curve achieved for this model is depicted in Figure 4.3. The curve at the beginning of training improves until Epoch 11 even though the improvement is less than 5 %. After that, the curve becomes almost a straight line throughout the rest of the training. The train and validation curves are close to each other and neither diverge from nor converge with one another. This indicates that the optimization algorithm is not able to converge further. The accuracy for the training and validation sets from the beginning until the end is between 50 and 60 % which is considered almost random for a binary classifier. In other words, the curve shows that the model either is not able to learn how to detect getting up behavior from depth data alone, or the provided data is simply not informative enough for the task it should solve. From these observations, it can be concluded that without the transfer learning, the network is not able to converge solely based on the currently available data.

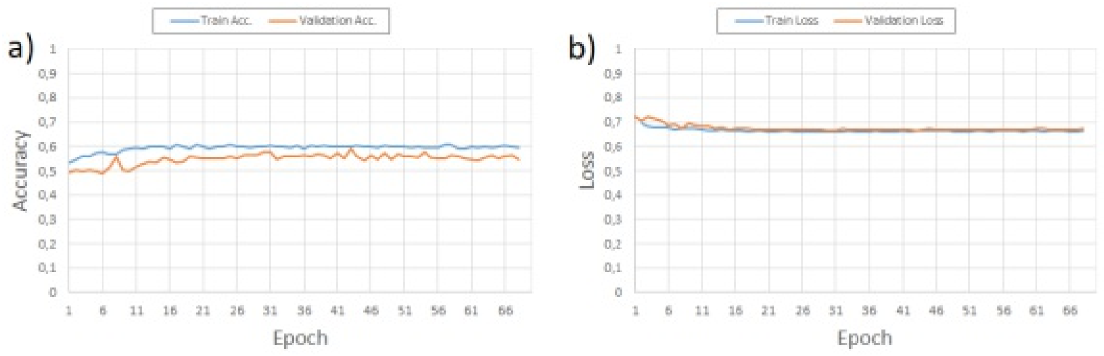
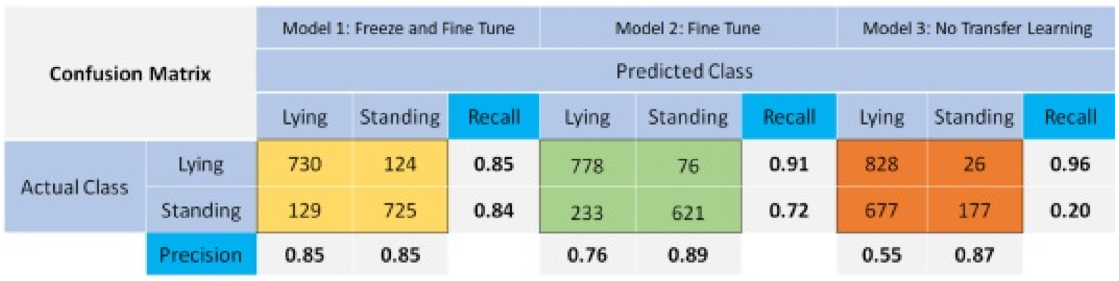


Figure 4.3: The plots demonstrating a) the accuracy and b) the loss of the train and validation sets at the end of each Epoch during the training phase in which the network's weights and biases are not transferred from a trained network and are initiated randomly.

Confusion Matrix

The confusion matrix is a suitable approach to dive deeper into evaluating the three models introduced above based on the class labels. In Table 4.2, three binary confusion matrices for each of the three models along with their precision and recall are depicted. The validation set contains exactly 854 images for each label.

For the freeze and fine-tune model, 85 % of all images labeled „lying“ and 84 % of images labeled “standing” are correctly detected (recall) and the precision achieved for lying and standing labels is 85 % for both. In the fine-tuning model, 91 % of lying labels and 72 % of standing labels are detected correctly and accuracy for lying and standing reached 76 % and 89 % respectively, which indicates more than 15 % difference between the two labels for this model. Besides, the recall for lying and the precision for standing are almost 5 % higher than the same values achieved by the Freeze and fine-tune model. The difference gets higher up to 10 % when it comes to the precision for lying and recall for standing between the two models. In other words, the difference between these two models is due to the fact that the Freeze and fine-tune model is able to detect both labels equally but the fine-tune model is almost 5 % higher at detecting lying labels and higher precision at detecting standing labels. In model 3, where no transfer learning techniques are used, the precision reaches approximately 55 % for lying and 87 % for standing, which indicates that the model is biased towards higher precision in detecting standing situations. However, the recall value of 20 % for standing and 96 % for lying shows a 70 % difference between the labels in the same model that implies the model is biased toward detecting lying positions rather than standing positions.



Confusion Matrix		Model 1: Freeze and Fine Tune			Model 2: Fine Tune			Model 3: No Transfer Learning		
		Predicted Class								
		Lying	Standing	Recall	Lying	Standing	Recall	Lying	Standing	Recall
Actual Class	Lying	730	124	0.85	778	76	0.91	828	26	0.96
	Standing	129	725	0.84	233	621	0.72	677	177	0.20
	Precision	0.85	0.85		0.76	0.89		0.55	0.87	

Table 4.2: Three binary confusion matrix for all three models of classification are shown, where the three models (Freeze and fine-tune, No transfer learning, and fine-tune) are demonstrated in colors of yellow, orange, and green respectively. The rows in the matrix represent actual class labels and columns represent the predicted class. The diagonal of each matrix demonstrates the true positives of each class. The precision and recall for each label and model is calculated and is depicted next to each matrix.

Incorrect detections

In order to distinguish between situations, it is possible to divide the images of the validation set into their field of view and evaluate the models based on the field of view, because each field of view has its own characteristics which can help to evaluate the models. In the validation set, there are four different situations. Four examples of the images from each situation are depicted in Figure 4.4 and their characteristics are described as follows:

- **Couch**: An L-shaped couch is used instead of a normal bed
- **Bedroom 1**: The bed is positioned far from the camera and in a way that only the bottom of the bed is visible to the camera set.
- **Bedroom 2**: King size bed, only half of which is used and the camera set is positioned on the side.
- **Bedroom 3**: The edges of a single bed (especially the bottom of the bed) are occluded with blankets and the camera is positioned on the side of the bed in a lower position.

In Table 4.3, the incorrect detections are divided into different rooms contained in the validation set where the images were originally taken. The field of views of bedroom 1 and the couch with the average of 31.6% and 34.6 % respectively are the highest incorrect detections among of all the field of views. The reason for this is, in the couch field of view, the network should detect a getting up from a couch based on the data that only represents getting up behavior from beds. In bedroom 1 field of view, by looking closely at Figure 4.4, it is clear that the bed inside the field of view in bedroom 1 is positioned in a way that the bottom of the bed occludes the rest of the bed and the highest wrong detection is for detecting standing position in the field of view of the bedroom 1. In the field of view of bedrooms 2 and 3, even though the rooms and beds are considered to be new situations for the model, the least wrong detections happen in comparison to other situations.

Field of View	Incorrectly detected (%)											
	Model 1: Freeze and fine-tune			Model 2: fine-tune			Model 3: No Transfer Learning			Average Over all models		
	Total	Lying	Standing	Total	Lying	Standing	Total	Lying	Standing	Total	Lying	Standing
Couch	16.9	16.3	**17.2**	**27.5**	**41.8**	20.9	**59.6**	0	86.8	**34.6**	19.3	41.6
Bedroom 1	**41.7**	**54.3**	8.1	25.1	12.9	**57.3**	28.2	**11.7**	72.2	31.6	**26.3**	**45.8**
Bedroom 2	8.8	3.2	13.7	17.6	0	33.2	35.5	2.3	65.0	20.6	1.8	37.3
Bedroom 3	8.2	2.2	16.9	7.1	1.5	15.3	39.6	0	**97.2**	18.3	1.2	43.1
Average	18.9	19.0	13.9	19.3	14.5	31.6	40.7	3.5	80.3			

Table 4.3: Summary of incorrect detections in percentage based on the field of view of the camera(rooms) for each model in total and per label. In the validation set, the data per label is not balanced since it is representing real-world situations and that is the reason why in some cases, the total percentage of incorrectly detected is not equal to the average over the incorrect percentage of the labels.

By looking at Table 4.3, in the model freeze and fine-tune column, the error in detecting lying positions in bedroom 1 is much higher than in the fine-tuning model and vice versa. This can be explained by the randomness of the learning algorithm which indicates there are local minimums in training that could lead to the same overall accuracy but in fact, they are different in detection. Besides, the no transfer learning model is skewed toward lying labels. Therefore the incorrectly detected standing in all the situations is higher than for other models.

By carefully examining the detected images, some of which are depicted in Figure 4.4, the situations that the models were not able to detect correctly fall into two categories: Firstly, where the images labeled wrongly as standing. these situations contain a position when a person is in the progress of changing position from lying to sitting and/or the legs of the person inside the frame are covered with a blanket. In this situation, the freeze and fine-tune model achieved the lowest incorrect detection for standing. Secondly, where the images are labeled lying incorrectly. This has three reasons: First, the sitting position is different than usual, especially in the couch images, where one person can sit at the front, center, or back side of the couch after getting up. Second, not both of the legs are visible in a sitting position. This is where one leg of one person is either hidden under the blanket or the person is sitting with one leg tucked under the other. The third reason is similar to the previous scenario, where a person is in the middle of the getting up process. In this situation, the fine-tuning model achieved the lowest incorrect detection rate at the lying label. The incorrect detection score on average for the freeze and fine-tune model is the lowest in comparison to other models.

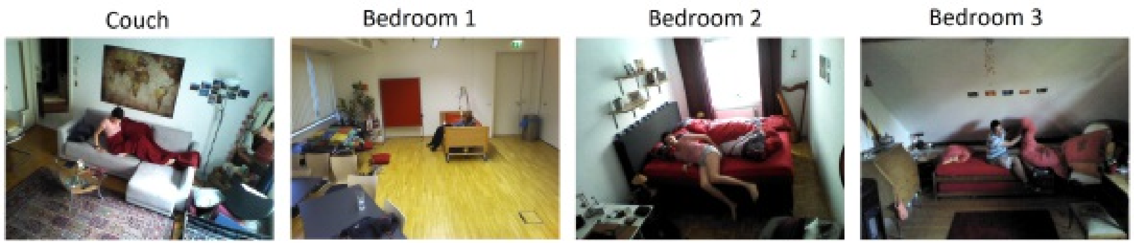


Figure 4.4: Examples of images for each field of view used in the validation set that the models had difficulty detecting them correctly.

Summary

Table 4.4 shows an overview of all the measurements obtained for all three models. The model Freeze and fine-tune is able to reach the highest validation accuracy and lowest validation loss among all the models. However, the fine-tune model comes as the close second and the difference to the first model is less than 3 %. By looking at the F1-measurements, lying on average is 13 % higher in comparison to standing. This indicates that all models are able to detect lying images better than standing images.

Overview of all the classification experiments									
Model	Precision		Recall		F1-Measurement		Accuracy	Loss	Time
	L	S	L	S	L	S	(Validation)	(Validation)	(Minute))
Model 1: Freeze and fine-tune	**0.85**	0.85	0.85	**0.84**	**0.85**	**0.84**	**85.2 %**	**0.355**	39
Model 2: fine-tune	0.76	**0.89**	0.91	0.72	0.82	0.79	81.9 %	0.389	19
Model 3: No Transfer Learning	0.55	0.87	**0.96**	0.20	0.69	0.32	58.8 %	0.666	40
Average	0.72	0.87	0.90	0.58	0.78	0.65			

Table 4.4: Summary of all classification models

By comparing the accuracy of models 1 and 3, it becomes clear that using transfer learning techniques has an positive impact on solving the problem. Even though, the knowledge that transferred is from the ResNet18 network which is only trained on the-channel RGB images, and this experiment is based on single-channel depth images.

4.2 Object detection

The object detection network, similar to classification, was trained in three different ways: Model 1 freeze and fine-tune, Model 2 fine-tune, and Model 3 no transfer learning. The main reason for training the same network in different ways is not only to achieve the best results but also to exploit the effect of the transfer learning technique. The concept of three different training processes is similar to classification. All of the three models are compared with each other and their advantages and disadvantages are discussed in the following subsections.

Hyperparameters

All hyperparameters in object detection are listed in Table 4.1 along with their values for the best achieved result. The table does not contain all hyperparameters available for the neural network model. Hyperparameters related to architectural design are excluded. The values are selected based on the grid search technique and are kept constant throughout all three models in training of the the object detection task. The number of Epochs are omitted in the table, since the early stopping technique is used.

Hyperparameter	Value
Learning Rate	0.001
Batch Size	8
Optimizer	SGD
Momentum	0.9
Weight Decay	0.0001
Learning Rate Step Size	Every 4^{th} Epochs
Learning Rate Change Rate (gamma)	0.1
Loss Function	Cross Entropy
Early Stopping	True
Data Augmentation	True
Normalization	True

Table 4.5: Hyperparameter values used in training of the object detection task

Model 1: Freeze and Fine-tune

Figure 4.5 depicts the curves for the freeze and fine-tune model. The left plot in the figure shows the mAP curves in which the mAP(COCO) indicates that the mAP is calculated for IoU $[0.50 - 0.95]$ at each 0.05 interval and the mAP(PASCAL VOC) indicates that the mAP is calculated for $IoU > 0.5$ for the validation set during training. The right plot demonstrates the different losses of train data during training. In the first phase of training, where only the newly added parameters participate in training, the curve starts from mAP(COCO)= 0.05 % and mAP(PASCAL VOC)= 25 % and goes up to 7 % and 3 % respectively until Epoch 3. Afterward, both curves start to fluctuate and reach higher average precision until Epoch 12, where phase two of the training begins. In the second phase of training where the entire network participates in the training, the mAP steeply increases at Epoch 13. However, interestingly enough, in the right plot, the loss at Epoch 13 increases when the training phase changed. Despite the higher loss in at Epoch 13, the network with all of its parameters in training achieved approximately 15 % higher mAP(PASCAL VOC) compared to Epoch 12, where the loss is smaller but only part of the network is taking part in the training. This could be because of RPN network and fully connected layers of the network which are initialized randomly, actually had a worsening effect in training after the FPN part of the network is unblocked in the second phase of training. In other words, the layers must have unlearned what they learned during the first phase of training. Nevertheless, After Epoch 13 the network started fluctuating again and did not improve even though the loss was still decreasing. After 9 Epochs, the training stopped and the states of all parameters from the best mAP(COCO) at Epoch 13 is saved as the final model. mAP(COCO) is considered the final single evaluation metric during training because it is a metric that covers different IoU thresholds from 0.5 to 0.95 and it represents an overall fitness of the model.

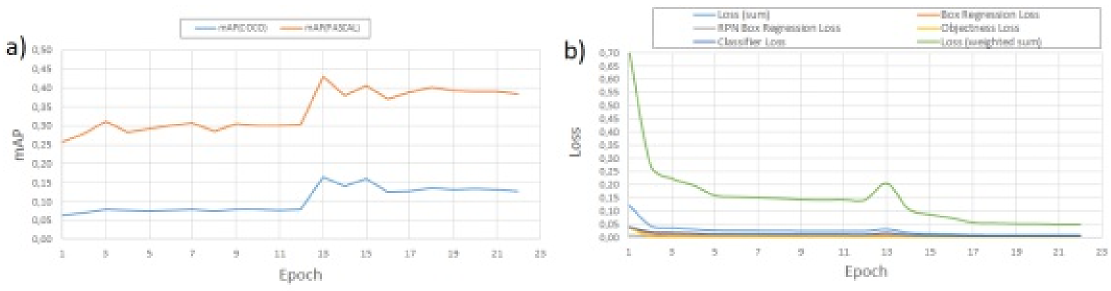


Figure 4.5: plots demonstrating the training curve for the freeze and fine-tune model. Plot a) indicates mAP for two different IoU thresholds of [0.5 – 0.95] for mAP(COCO) and $IoU > 0.50$ for mAP(PASCAL VOC). Plot b) demonstrates the loss values of the training set during training. The loss of the training contains four different losses that are calculated directly from outputs of the network, loss(sum) is the sum of all four, and loss (weighted sum) is the sum of the magnified all of the four loss values.

Figure 4.6 demonstrates PR-Curve for two different IoU thresholds for the trained model at Epoch 13. In plot a), the curve starts from the top left, where precision is equal to 1 and recall is equal to about 0.08. The curve slowly comes down to where the precision equals 0.20 and recall is near 0.70. This shows that at first the model is able to find approximately 8 % of all the ground truths with 100 % positive prediction and as the recall started to rise, the model is less likely to detect them correctly until the point where 70 % of the ground truths are detected with only 20 % confidence. In plot b), the model becomes less confident as the IoU threshold is raised to 0.70. The precision starts around 0.40 and goes up slightly and then slowly drops until 0.10 and recall reaches the highest point below 0.40. The area under the curve is an approximation of the mAP for its respective IoU. In Figure 4.6, the left plot has a mAP of 42.9 % which is equal to the value of mAP(PASCAL VOC) at Epoch 13 in the left plot of Figure 4.5. The right plot has a mAP equal to 14.9 %. The difference between the two mAPs suggests that the model is less accurate in finding more precise bounding boxes.

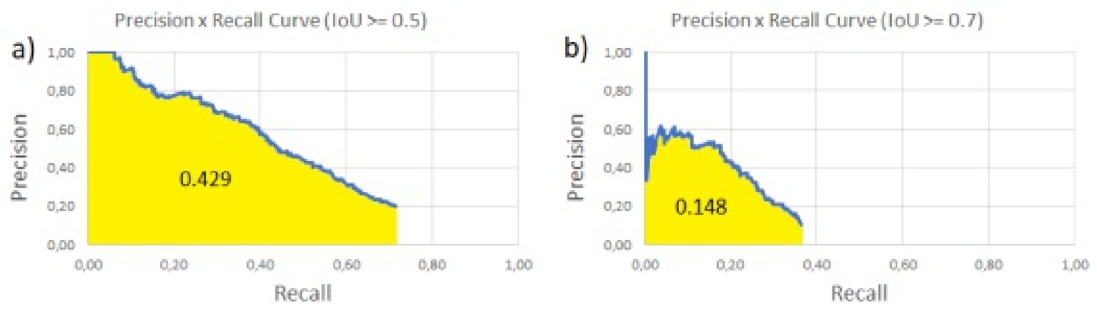


Figure 4.6: plots demonstrating precision recall curve for a) $IoU > 0.5$ and b) $IoU > 0.7$ for the freeze and fine-tune model. The area under the curve approximates the mean average precision of the model at the defined IoU threshold.

Model 2: Fine-tune

The network in this model is trained as a whole without any restrictions from the beginning. Figure 4.7 depicts the training curve for this model. In the left plot, the curve starts at 0.80 and 0.30 for mAP(COCO) and mAP(PASCAL VOC) respectively. The mAP(PASCAL VOC) starts to increase until 0.90 at Epoch 5 where it reaches its optimum point. On the other hand, the mAP(COCO) does not change much during training. It increases slightly from 0.30 at Epoch 3 to 0.31 at Epoch 4. In the right plot, loss values starts at 0.40 and decreases sharply until it is 0.10 atEpoch 2 and continues to decrease until Epoch 5. Afterward, it does not change and stops improving. The training process stopped at Epoch 13, where no more improvements are achieved and returned the parameter values at Epoch 4 as the final model parameters. An interesting fact about the training curve is that both of the mAPs did not increase more than 10% during training but they started at a point where it is twice as high as the previous model. At the start of training, the network has no understanding of the data and the task, because the network before training at Epoch 0 is initialized with random values. Therefore, the network either learned most of the data in the first Epoch and/or it started at a point close to the local optimum with a steep gradient.

Figure 4.8 depicts PR-Curve for the fine-tune model. In the left plot, precision has a value of 1.0 at the top left corner for the recall between $[0, 0.4]$. Thereafter, as the recall increases from 0.4 onwards, the precision starts to decrease to the point where recall equals almost 1.0, and precision drops to 0.35. As shown in the figure, the precision dropped drastically in the last 20 % of recall. In other words, the model is able to detect 80 % of all ground truths with a precision of more than 80 % but close to where recall equals almost 1, the model is able to detect the entire ground truths with only 35 %

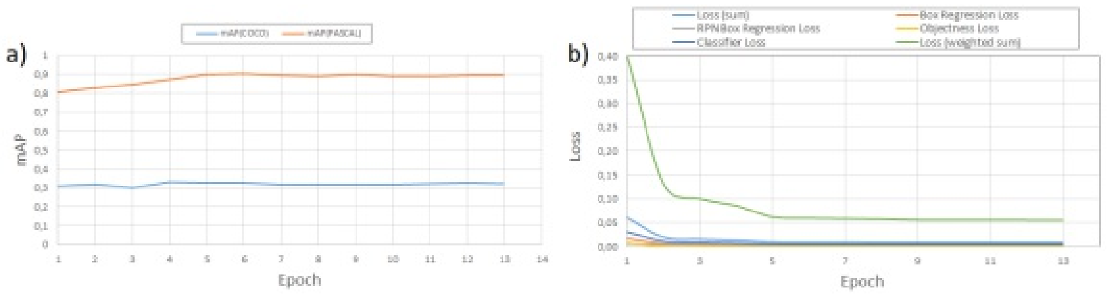


Figure 4.7: Plots demonstrating training curve for the fine-tune model. Plot a) indicates mAP for two different IoU thresholds of [0.5 – 0.95] for mAP(COCO) and $IoU > 0.50$ for mAP(PASCAL VOC). Plot b) demonstrates the loss values of the training set during training. The loss of the training contains four different losses that are calculated directly from outputs of the network, the loss(sum) is the sum of all four, and loss (weighted sum) is the sum of the magnified all of the four loss values.

precision, so with less than half of what it is at recall 80 %. This indicates that the model is facing difficulties to detect 20 % of the ground truths. This plot also shows that the model is able to detect all of the ground truths and to detect more than 80 % of them with 80 % precision.

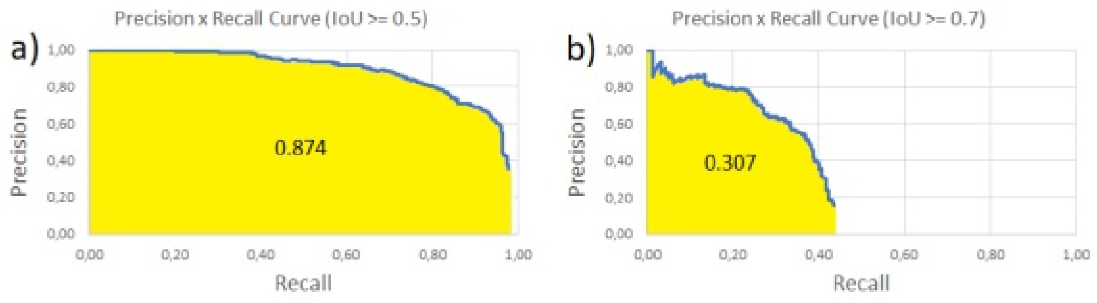


Figure 4.8: Plots demonstrating precision recall curve for a) $IoU > 0.5$ and b) $IoU > 0.7$ for the fine-tune model. The area under the curve approximates the mean average precision of the model at the defined IoU threshold.

In the right plot of Figure 4.8, the model starts at the top left corner with a precision equal to 1.0 but it decreases drastically as the recall increases to the point where it is approximately 0.45. This plot demonstrates that the model, similar to the previous model is not able to detect all the ground truths when the more accurate IoU threshold is considered.

Model 3: No Transfer Learning

The network in this model is trained with all of its parameters initialized randomly and participating in training without any restrictions. Figure 4.9 shows the training curve for this model. In the left plot, the curve for the mAP(COCO) starts at 0.12 and for the mAP(PASCAL VOC) starts at 0.37. Similar to the previous model, mAP(COCO) does not change much during training. It only increases slightly from 0.12 at Epoch 3 to 0.15 at Epoch 5. However, mAP(PASCAL VOC) increases from 0.37 to 0.5 at Epoch 6. In the right plot, despite the fact that loss values are less than one at the Epoch 1, it manages to decrease considerably from 0.65 at the beginning to 0.11 Epoch 5. The training algorithm stopped after Epoch 14 where it could not converge further and returned the parameter values at Epoch 5 as the final output.

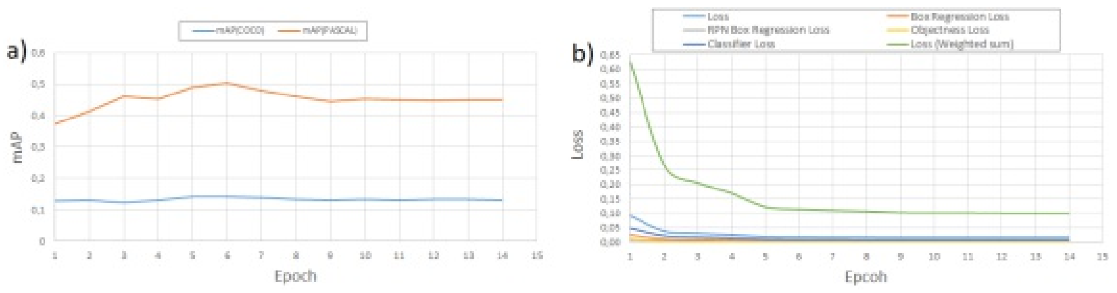


Figure 4.9: Plots demonstrating training curve for the no transfer learning model. Plot a) indicates mAP for two different IoU thresholds of [0.5 – 0.95] for mAP(COCO) and $IoU > 0.50$ for mAP(PASCAL VOC). Plot b) demonstrates the loss values of the training set during training. The loss of the training contains four different losses that are calculated directly from outputs of the network, loss(sum) is the sum of all four, and loss (weighted sum) is the sum of the magnified all of the four loss values.

In Figure 4.10, PR-Curve demonstrates the evaluation of the no transfer learning model. In plot a), on the top left corner, precision equals 1.0 at its maximum. As the recall or the total number of positive detection increases, the precision starts to drop about 10 % to the point where the recall is near 0.28. Afterward, the precision starts to decrease rapidly until the precision is dropped down almost 40 % where recall equals 0.38. At this point, the decrease of the precision slows down but continues until 0.23, where the recall is 0.82. This plot suggests that the network is able to learn some getting up positions better than others. Therefore, there is a 50 % change in precision as the recall increases only 20 %. In the right plot of Figure 4.10, the area under the curve is less than 0.5. Precision starts at 0.4 and drops down immediately to almost zero at recall 0.2. The area under the curve in plot b) is equal to only 4.2 % which indicates that the network is not able to detect precise boundary boxes at all.

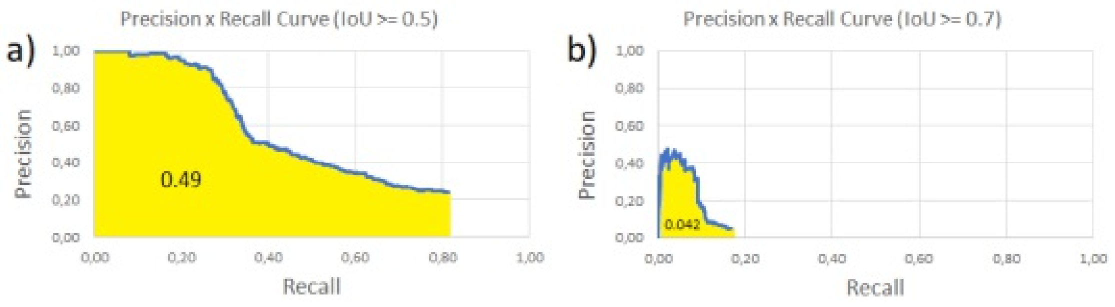


Figure 4.10: Plots demonstrating precision recall curve for a) $IoU > 0.5$ and b) $IoU > 0.7$ for the no transfer learning model. The area under the curve approximates the mean average precision of the model at the defined IoU threshold.

Analyzing incorrect detections

An approach to evaluate the three models is to dive deeper into understanding which situations are not detected. In order to distinguish between situations, the images of the validation set can be divided into and evaluated based on their field of view, because each field of view has specific characteristics which can help to better evaluate the models. In the validation set, there are four different situations. Eight examples of the images from each situation are depicted in Figure 4.11 and have the following characteristics:

- **Couch**: An L-shaped couch is used instead of a normal bed
- **Bedroom 1**: The bed is positioned far from the camera and in a way that only the bottom of the bed is visible to the camera set.
- **Bedroom 2**: King size bed, only half of which is used and the camera set is positioned on the side.
- **Bedroom 3**: The edges of a single bed (especially the bottom of the bed) are occluded with blankets and the camera is positioned on the side of the bed in a lower position.

In Table 4.6, calculated mAPs for the three models based on field of view for two different IoU thresholds of 0.5 and 0.7 are represented. As shown in the left part of the table, where IoU > 0.5, the fine-tune model achieved the highest mAP in all room settings. In the couch field of view, the freeze and fine-tune model (model 1) and fine-tune model (model 2) could achieve higher mAP than the no transfer learning model (model 3). However, in the bedroom 1 field of view, model 1 did not perform well and achieved only a mAP equal to 4.6 %. On the other hand, in the field of view of bedroom 2, all of the models achieved an average of 94 % which indicates that all models correctly detected most of the getting up positions in this field of view. In bedroom 3, model 2 is able to achieve the highest mAP score of 93.9 % which is three times larger than the second-highest 28.3 %.

Model 1 was able to detect the couch and the bedroom 2 field of views with 68.3 % and 86.9 % mAP respectively whereas the mAP for bedroom 1 and 3 is less than 5 %. Model 2 achieved 99.4 % and 93.9 % mAP scores in bedroom 2 and 3 respectively which is at most 20 % higher than the scores for the couch and bedroom 1. The interesting difference is in the performance between model 1 and model 2 (especially in bedroom 1 and 3) even though, in training, both used the same network, training data, and fine-tuning technique. The only difference is that RPN and classification parts of model 1 are trained before the fine-tuning training phase. Freezing the network before fine-tuning had a reverse effect on the learning process in model 1. In other words, the network forgot/unlearned what it had learned/transferred before.

By looking at the average of all mAPs per field of view overall models in Table 4.6, it becomes is clear that all of the models detected all of the images belonging to bedroom 2 with more than 94 % accuracy. However, in other situations, the models achieved on average a mAP between $[43\% - 55\%]$. This indicates that Bedroom 2 is the most suitable field of view for all the models.

Models	mAP (%)							
	IoU > 0.5				*IoU* > 0.7			
	Couch	Bedroom 1	Bedroom 2	Bedroom 3	Couch	Bedroom 1	Bedroom 2	Bedroom 3
Model 1: Freeze and fine-tune	68.3	4.6	86.9	8.3	**37.9**	0	23.6	0.84
Model 2: fine-tuning	**84.5**	**71.5**	**99.4**	**93.9**	20.5	**0.004**	**81.4**	**14.6**
Model 3: No Transfer Learning	12.7	54.7	97.6	28.3	0.02	0	12.5	5.7
Average Over All Models	55.1	43.6	94.6	43.5	19.47	0	39.1	7.1

Table 4.6: Summary of incorrect detections in percentage based on the field of view of the camera(rooms) for each model in total and per label

On the right side of the Table 4.6, where the IoU > 0.7, model 2 achieved the highest mAP values in the bedrooms 1,2 and 3 field of view and model 1 was able to achieve the highest mAP in the couch field of view. Model 3 was barely able to detect anything in the couch field of view. In bedroom 1, surprisingly (almost) no model could detect any boundary boxes. This indicates that all the models have problems precisely locate smaller objects because the bed is located further away from the camera and consequently makes the boundary boxes smaller in comparison to other fields of view (See figure 4.11). In bedroom 2, model 2 achieved a mAP equal to 81.4 % which is approximately three times better than the mAPs in the other two models for the same field of view. In bedroom 3, all three models, similar to bedroom 1, achieved low mAP scores where all of them are below 15 %. By looking at the average mAPs on the bottom, bedroom 2 again has the highest value among other situations and the couch takes second place, whereas bedroom 1 and 3 come in last. It is noteworthy that the average mAP in each situation where IoU > 0.7 is smaller than the average mAPs where IoU > 0.5.

Figure 4.11 demonstrates eight incorrectly detected examples from all field of views. The examples for bedroom 1 and the couch field of view show that finding a correct boundary box is crucial for faster R-CNN. Basically, the network finds the region of interest, then cuts inside the box and classifies it. Therefore, both parts of cropping and classifying are essential for the faster R-CNN object detection network. If the box is placed in an incorrect position, no matter how good the classifier, the classifier cannot detect it correctly. On the other hand, if the box is placed in the correct position but the classifier makes a mistake and does not classify it correctly, the network also performs badly as demonstrated in the examples for bedroom 2 and 3 in Figure 4.11. By looking closely at these pictures, the proposed regions do contain a bed and a person (even though the proposed boundary boxes are bigger than the ground truths) even though the classifier cannot assign the correct class label to them. Yet, at the bedroom 1 bottom image in the figure, the proposed green region does contain the bed, but not the person and the classifier correctly assigned lying since the person is not in the box and to the classifier, it appeared that the person is occluded by the blanket.

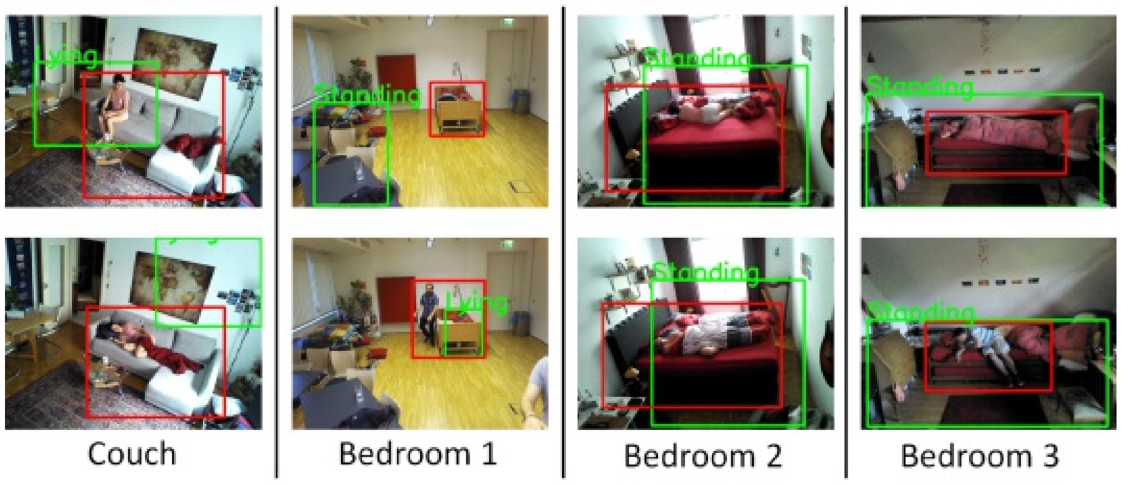


Figure 4.11: Examples of the wrong detections by object detection models in different rooms available in the validation set. Red boxes inside the images indicate the ground truth and green boxes depict- the detections made by the models.

Summary

Table 4.7 shows an overview of all measurements obtained for the three models. The fine-tuning model achieved the highest mAPs, mean Average Recall (mAR), and the smallest loss in comparison to the other models. By looking at mAP where the IoU [0.50-0.95] is, the models 1 and 3 achieved similar values equal to 16.5 % and 14.1 % respectively. This implies that the two-phase training in this situation not only did not benefit the performance of model 1 but also made the network perform as good as if it did not use any transfer learning techniques. Model 2 achieved mAP 33.1 % for the IoU [0.50-0.95] which is almost double in value in comparison to models 1 and 3. The same situation applies to mAR, where models 1 and 3 are close to one another, and model 2 has the highest value among them in the Table.

Models	mAP (%)							mAR (%)	Loss (Weighted)	Training Time (Minutes)
	IoU > 0.5			*IoU* > 0.7			IoU	IoU		
	Total	Lying	Standing	Total	Lying	Standing	[0.50-0.95]	[0.50-0.95]		
Model 1: Freeze and fine-tune	42.9	41.6	44.2	14.8	12.35	17.25	16.5	27.4	0.204	340
Model 2: fine-tuning	**87.4**	**88.75**	**86.06**	**30.7**	**27.1**	**34.3**	**33.1**	**41.1**	**0.084**	**249**
Model 3: No Transfer Learning	49.0	44.65	53.35	4.2	1.5	6.9	14.1	25.7	0.122	277
Average over all models	59.7	58.3	61.8	16.5	13.6	19.4				

Table 4.7: Summary of all object detection models

In the Table 4.7, results for mAP at IoU > 0.5 and IoU> 0.7 are calculated per each label. By comparing the average of standing and lying mAPs, the standing label has achieved a 3.0 % to 6.0 % higher score than lying on average which indicates that detecting standing situations is easier than lying even though the size of the lying and standing images are balanced in the validation set. A standing position is easier to detect since the person is visible and not occluded with blankets in comparison to lying. Among all three models, the IoU > 0.5 has a much better score than IoU > 0.7 on average which implies that the models faced difficulties in detecting precise boundary boxes.

Model 1, which uses a pretrained backbone in training, not only did not perform equal to model 2 but also model 3 outperformed it at calculated mAP where IoU > 0.5 although model 3 did not even use the pretrained backbone. This suggests, that the training process can hinder the possible positive effect of transfer learning techniques on model performance. Model 2, which also used the pretrained network, outperformed the other two models in all metrics. This emphasizes how much transfer learning techniques benefit the performance of model 2 as well as training time (model 2 took almost 30 minutes less than model 3).

16 of the correctly detected situations with IoU > 0.5 performed by model 2 are demonstrated in Figure 4.12. In all images, the model was able to propose a boundary box that contained a person and a bed and classify them correctly. An interesting point is that in part d) of the figure, the model detected the extra blanket and pillows at the bottom of the bed as a part of the bed, which possibly indicates that the model has gained an understanding of blankets.

4.3 Comparison

It is not possible to directly compare models used for classification and object detection. Each model is evaluated with different metrics and generally they all have a different purpose in computer vision. Therefore, in order to compare the models of the two methods, it is necessary to define an objective goal and analyze their advantages and disadvantages towards the defined objective. In this context, the defined goal is to detect the getting up behavior of one person from a bed inside a field of view while keeping the computational cost (speed) in mind.

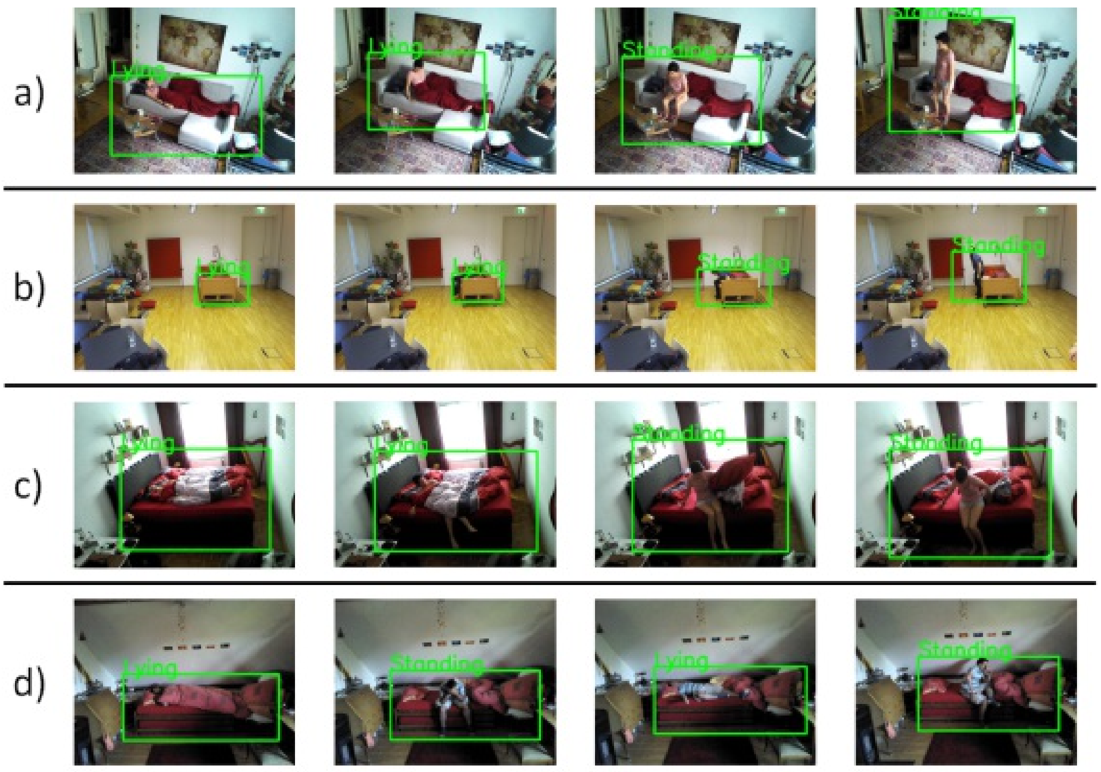


Figure 4.12: Examples of correctly predicted frames in all different scenes available in validation set.

One should keep in mind that the comparison of the two methods is based on the best models obtained during training with regard to the defined goal and it is not a general comparison between classification and object detection.

- **Classification** (Model 1: Freeze and fine-tune):
  - **Advantages**:
    - ∗ Smaller network and computationally more efficient in comparison to the object detection network.
    - ∗ Faster detection.
  - **Disadvantages**:
    - ∗ Single detection per frame.
    - ∗ Assigns a label to a frame regardless of what the input images are.
- **Object Detection** (Model 2: fine-tune):
  - **Advantages**:

    * Ability to locate getting up behavior inside the frame.
    * Ability to detect multiple people getting up in the same frame.

- **Disadvantages**:
    * Bigger network and consequently less computationally efficient in comparison to classification network
    * Less accurate when the bed is positioned further from the camera set
    * Longer training time.
    * Only the backbone is able to benefit from pretrained networks and the RPN and classifier parts are initialized randomly.
    * the entire detection is based on the boundary boxes

The classification method proved to be reliable in various situations, since the classifier does not rely on boundary boxes. However, if the correct boundary box is proposed, the object detection is less likely to make mistakes compared to classification. Classification uses a smaller network which makes it computationally more efficient than the object detection network. This makes classification suitable for less powerful computers such as Raspberry PI where the model can be deployed on the device and detects getting up behavior without any latency on demand. On the other hand, object detection brings the possibility of detecting multiple getting up motions in settings such as using one camera for detecting multiple beds in a hospital setting, where post-processing can be used.

CHAPTER 5

Conclusion and Outlook

This thesis project contributes to the research field of human behavior modeling by introducing an example of using depth images to detect a specific human action (getting up from a bed) indoors. The method is applicable even when the person is partially occluded with indoor objects without any restriction on the angle of the camera. Furthermore, the work described here demonstrates how to gather and preprocess depth images to produce a dataset for training networks that are built and pretrained based on RGB images.

In conclusion, both the classification and the object detection method are able to reliably detect getting up behavior. The results presented here imply that the CNN is capable of extracting high-level task-dependent features from depth data regardless of the architectures for classification or object detection. Also, using pretrained networks is the key contributor to correctly detect getting up behavior even though the pretrained networks used here were trained on RGB data and not on depth data. Transfer learning techniques reduce the size of task-specific data because the data used for training the networks in this project are considerably small in comparison to well-known datasets such as COCO or ImageNet.

Going forward, future work will have to focus on reducing the errors in detecting getting up where a person is in transition from lying to standing or vice versa. One possible solution is to introduce a new class label to classify transition points. The practicality of the network can then be changed or expanded from getting up from a bed to sitting and walking inside the room, based on the purpose of the real-life application. The combination of depth data and CNN for modeling human behavior gives the flexibility to be used in intelligent systems such as detection of falls or sleep positions. Depending on the goal of the application (for example high inference speed or high performance), this combination can be applied to a variety of state-of-the-art convolutional neural networks.

List of Figures

List of Tables

Glossary

Data Augmentation a strategy to significantly increase the diversity of data without actually collecting the actual data. 46, 50

Epoch A complete loop over all training images with mini-batches. 26, 28, 34, 48, 55, 62, 67–71, 75–77, 79

ImageNet ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. 52, 53, 87

Mini-Batch a subset of dataset which its size is 2 to power of n like 32,64,128,256, etc.. 30, 32

Tensor a form of generalized matrices and can be represented in multidimensional array. 31

Acronyms

AP	Average Precision. 40, 41
BAM	Bed Aligned Map. 5
CNN	Convolutional Neural Network. 5, 6, 11–13, 16–18, 22, 32, 33, 52–57, 62, 87
COCO	Common Object in COntext. 38, 39, 41, 56, 75–79, 87
CUDA	Compute Unified Device Architecture. 16
DBN	Deep Belief network. 11
ELU	Exponential Linear Unit. 16
FF	Feed-Forward. 12, 13
FPN	Feature Pyramid Network. 57–60, 76
FPS	Frames Per Second. 44, 46
GPU	Graphics Processing Unit. 16, 26
ICU	Intensive Care Unit. 4
ILSVRC	Imagenet Large-Scale Visual Recognition Challenge. 10
IoU	Intersection Over Union. 38, 39, 41, 67, 75–83
IR	Infra-Red. 44, 45
LSTM	Long Short-Term Memory. 11
MAE	Mean Absolute Error. 24
mAP	mean Average Precision. 41, 67, 75–83

mAR mean Average Recall. 82

MLP Mulit-Layer Perceptron. 7, 13, 14, 18, 20, 32

mm MilliMeters. 44

MSE Mean Squared Error. 24

NIPS Neural Information Processing Systems. 32

NLP Natural Language Processing. 11

nm NanoMeters. 44

NN Neural Network. 7, 8, 11, 13

ODDS (Object Detector using a Depth Sensor. 5, 6, 89

ONI Open Natural Interactions. 64

OpenNI Open Natural Interaction. 64

PASCAL VOC Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes. 38, 39, 41, 50, 75–79

PNG Portable Network Graphics. 46

PR-Curve Precision x Recall Curve. 38, 40, 76, 77, 79

R-CNN Region-Based Convolutional Neural Network. 18–21, 51, 56, 58, 61, 62, 82

ReLU Rectified Linear Unit. 14–17, 54

ResNet Residual Network. 17, 51–60, 62, 69, 70, 74

RGB Red Green Blue. 2–6, 8, 31, 43, 47–51, 64, 74, 87

RGB-D Red Green Blue-Depth. 3

RNN Recurrent Neural Network. 11, 32

RoI Region Of Interest. 18, 20, 56, 58–60

RPN Regional Proposing Network. 19, 20, 56–60, 62, 76, 81, 85

SGD Stochastic Gradient Descent. 26, 69, 75

SSD Single Shot Detector. 5, 22

SVM Support Vector Machine. 18

VGG	Visual Geometry Group. 5, 6, 52
VSC	Vienna Scientific Cluster. 65
XML	Extensible Markup Language. 50
YOLO	You Only Look Once. 18, 20–22

Bibliography

[1] *Méthode générale pour la résolution de systèmes d'équations simultanées.* 1847.

[2] Klemke, e.d., robert hollinger, and a. david kline, eds. introductory readings in the philosophy of science. buffalo, ny: Prometheus books, 1980. xii + 373 pp.; $10.95 (pb); isbn 0-87975-134-7. *Science, Technology, & Human Values*, 5(4):66–66, 1980.

[3] J. Aggarwal and L. Xia. Human activity recognition from 3d data: A review. *Pattern Recognition Letters*, 48:70 – 80, 2014. Celebrating the life and work of Maria Petrou.

[4] A. Ali, K. Sundaraj, B. Ahmad, N. Ahamed, and A. Islam. Gait disorder rehabilitation using vision and non-vision based sensors: a systematic review. *Bosnian journal of basic medical sciences*, 12(3):193–202, Aug 2012. 22938548[pmid].

[5] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019.

[6] H. Ameur, S. Jamoussi, and A. B. Hamadou. A deep neural network model for predicting user behavior on facebook. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[7] A. Araujo, W. Norris, and J. Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. https://distill.pub/2019/computing-receptive-fields.

[8] D. H. Ballard and C. M. Brown. *Computer Vision.* Prentice Hall Professional Technical Reference, 1st edition, 1982.

[9] Y. Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

[10] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[11] C. M. Bishop. *Pattern recognition and machine learning.* Information science and statistics. Springer, New York, NY, 2006. Softcover published in 2016.

[12] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[13] M. Cornacchia, K. Ozcan, Y. Zheng, and S. Velipasalar. A survey on activity detection and classification using wearable sensors. *IEEE Sensors Journal*, 17(2):386–403, 2017.

[14] S. Deep and X. Zheng. Leveraging cnn and transfer learning for vision-based human activity recognition. In *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–4, 2019.

[15] B. Ding, H. Qian, and J. Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1836–1841, June 2018.

[16] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 766–774. Curran Associates, Inc., 2014.

[17] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111:98–136, 2014.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[19] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns, 05 2010.

[20] T. Fushiki. Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2):137–146, 2011.

[21] P. Gavali and J. S. Banu. Chapter 6 - deep convolutional neural network for image classification on cuda platform. In A. K. Sangaiah, editor, *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pages 99 – 122. Academic Press, 2019.

[22] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker, 2018.

[23] J. Geng. Structured-light 3d surface imaging: a tutorial. *Adv. Opt. Photon.*, 3(2):128–160, Jun 2011.

[24] R. Girshick. Fast r-cnn, 2015.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[26] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

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

[28] T. Grimm, M. Martinez, A. Benz, and R. Stiefelhagen. Sleep position classification from a depth camera using bed aligned maps. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 319–324, Dec 2016.

[29] Y. Gu, F. Ren, and J. Li. Paws: Passive human activity recognition based on wifi ambient signals. *IEEE Internet of Things Journal*, 3:1–1, 01 2015.

[30] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio. Noisy activation functions, 2016.

[31] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer, 2015.

[32] Y. Han and Y. Choi. Human action recognition based on lstm model using smartphone sensor. pages 748–750, 07 2019.

[33] A. Haque, A. Milstein, and L. Fei-Fei. Illuminating the dark spaces of healthcare with ambient intelligence. *Nature*, 585(7824):193–202, Sep 2020.

[34] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks, 2016.

[36] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. PMID: 16764513.

[37] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[38] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[39] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks, 2016.

[40] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.

[41] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[42] A. Kadambi, A. Bhandari, and R. Raskar. *3D Depth Cameras in Vision: Benefits and Limitations of the Hardware*, pages 3–26. 07 2014.

[43] W. Kennedy. *Modelling Human Behaviour in Agent-Based Models*, pages 167–179. 01 2012.

[44] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, Apr 2020.

[45] K. Khoshelham and S. Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors (Basel, Switzerland)*, 12:1437–54, 12 2012.

[46] P. Kittipanya-Ngam, O. S. Guat, and E. H. Lung. Computer vision applications for patients monitoring system. In *2012 15th International Conference on Information Fusion*, pages 2201–2208, July 2012.

[47] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[48] X. Kong, L. Chen, Z. Wang, Y. Chen, L. Meng, and H. Tomiyama. Robust self-adaptation fall-detection system based on camera height. *Sensors (Basel, Switzerland)*, 19(17):3768, Aug 2019. 31480384[pmid].

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[50] M. Kyrarini, X. Wang, and A. Gräser. Comparison of vision-based and sensor-based systems for joint angle gait analysis. In *2015 IEEE International Symposium on Medical Measurements and Applications (MeMeA) Proceedings*, pages 375–379, 2015.

[51] O. D. Lara and M. A. Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys Tutorials*, 15(3):1192–1209, 2013.

[52] Q. V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8595–8598, May 2013.

[53] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning, 2011.

[54] Y. Lecun. A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21–28. Morgan Kaufmann, 1988.

[55] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[56] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[57] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[58] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2016.

[59] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection, 2017.

[60] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2014.

[61] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[62] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017. Cited By :478.

[63] W. Liu, T. Xia, J. Wan, Y. Zhang, and J. Li. Rgb-d based multi-attribute people search in intelligent visual surveillance. In K. Schoeffmann, B. Merialdo, A. G. Hauptmann, C.-W. Ngo, Y. Andreopoulos, and C. Breiteneder, editors, *Advances in Multimedia Modeling*, pages 750–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[64] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[65] L. Luceri, T. Braun, and S. Giordano. Analyzing and inferring human real-life behavior through online social networks with social influence deep learning. *Applied Network Science*, 4(1):34, Jun 2019.

[66] R. Lun and W. Zhao. A survey of applications and human motion recognition with microsoft kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, 29:150330235202000, 03 2015.

[67] D. Lynch, W. Livingston, and W. Livingston. *Color and Light in Nature*. Cambridge University Press, 2001.

[68] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[69] A. Manzi, F. Cavallo, and P. Dario. A neural network approach to human posture classification and fall detection using rgb-d camera. In F. Cavallo, V. Marletta, A. Monteriù, and P. Siciliano, editors, *Ambient Assisted Living*, pages 127–139, Cham, 2017. Springer International Publishing.

[70] A. Martinez-Gonzalez, M. Villamizar, O. Canevet, and J.-M. Odobez. Efficient convolutional neural networks for depth-based multi-person pose estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, page 1–1, 2020.

[71] L. Minh Dang, K. Min, H. Wang, M. Jalil Piran, C. Hee Lee, and H. Moon. Sensor-based and vision-based human activity recognition: A comprehensive survey. *Pattern Recognition*, 108:107561, 2020.

[72] N. C. Mithun, S. Munir, K. Guo, and C. Shelton. Odds: Real-time object detection using depth sensors on embedded gpus. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 230–241, 2018.

[73] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

[74] S. S. Nath, G. Mishra, J. Kar, S. Chakraborty, and N.Dey. A survey of image classification methods and techniques. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pages 554–557, 2014.

[75] H. Nguyen, S. Maclagan, T. Nguyen, T. Nguyen, P. Flemons, K. Andrews, E. Ritchie, and D. Phung. Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring. 10 2017.

[76] T. Nguyen Gia, V. K. Sarker, I. Tcarenko, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen. Energy efficient wearable sensor node for iot-based fall detection systems. *Microprocessors and Microsystems*, 56:34 – 46, 2018.

[77] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[78] D. Olguín, P. Gloor, and A. Pentland. Capturing individual and group behavior with wearable sensors. pages 68–74, 01 2009.

[79] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[80] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, Oct. 2010.

[81] R. Parker. Human behavior modeling: The necessity of narrative. In L. Wang and L. Yu, editors, *Computer Architecture in Industrial, Biomechanical and Biomedical Engineering*, chapter 6. IntechOpen, Rijeka, 2019.

[82] T. Pedersen, C. Johansen, and A. Jøsang. Behavioural computer science: an agenda for combining modelling of human and system behaviours. *Human-centric Computing and Information Sciences*, 8(1):7, Mar 2018.

[83] M. Perc, M. Ozer, and J. Hojnik. Social and juristic challenges of artificial intelligence. *Palgrave Communications*, 5(1):61, Jun 2019.

[84] R. Planinc and M. Kampel. Introducing the use of depth data for fall detection. *Personal and Ubiquitous Computing*, 17, 08 2012.

[85] C. Pramerdorfer. *Depth data analysis for fall detection*. Wien, Techn. Univ., Dipl.-Arb., 20130, 2013.

[86] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the expressive power of deep neural networks, 2017.

[87] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2017.

[88] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition, 2014.

[89] J. Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[90] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2015.

[91] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger, 2016.

[92] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

[93] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[94] A. S. Rich and T. M. Gureckis. Lessons for artificial intelligence from the study of natural stupidity. *Nature Machine Intelligence*, 1(4):174–180, Apr 2019.

[95] S. Ruder. An overview of gradient descent optimization algorithms, 2016.

[96] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[97] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[98] L. Schmarje, M. Santarossa, S.-M. Schröder, and R. Koch. A survey on semi-, self- and unsupervised learning for image classification, 2020.

[99] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[100] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[101] B. G. Silverman, N. Hanrahan, L. Huang, E. F. Rabinowitz, and S. Lim. Chapter 7 - artificial intelligence and human behavior modeling and simulation for mental health conditions. In D. D. Luxton, editor, *Artificial Intelligence in Behavioral and Mental Health Care*, pages 163 – 183. Academic Press, San Diego, 2016.

[102] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[103] H. Singh, T. Parhar, and S. Malla. Gesture control interface using machine learning algorithms. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5:898, 10 2015.

[104] J. Smisek, M. Jancosek, and T. Pajdla. *3D with Kinect*, pages 3–25. 01 2013.

[105] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[106] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era, 2017.

[107] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A survey of optimization methods from a machine learning perspective, 2019.

[108] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[109] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.

[110] M. Talo, O. yildirim, U. Baloglu, G. Aydin, and U. R. Acharya. Convolutional neural networks for multi-class brain disease detection using mri images. *Computerized Medical Imaging and Graphics*, 78:101673, 10 2019.

[111] S. Thrun and L. Pratt. *Learning to Learn: Introduction and Overview*, pages 3–17. Springer US, Boston, MA, 1998.

[112] M. Vrigkas, C. Nikou, and I. A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015.

[113] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Learning actionlet ensemble for 3d human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):914–927, 2014.

[114] P. Wang, W. Li, P. Ogunbona, J. Wan, and S. Escalera. Rgb-d-based human motion recognition with deep learning: A survey. *Computer Vision and Image Understanding*, 171:118 – 139, 2018.

[115] I. R. Ward, H. Laga, and M. Bennamoun. Rgb-d image-based object detection: from traditional methods to deep learning techniques, 2019.

[116] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. Parameswaran. Accelerating human-in-the-loop machine learning: Challenges and opportunities. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18, New York, NY, USA, 2018. Association for Computing Machinery.

[117] D. Xin, L. Ma, S. Song, and A. Parameswaran. How developers iterate on machine learning workflows – a survey of the applied machine learning literature, 2018.

[118] J. Yao, Q. Huang, and W. Wang. Adaptive human behavior modeling for air combat simulation. In *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 100–103, 2015.

[119] K. Yordanova. Toward a unified human behaviour modelling approach. *Technical report University of Rostock, ISSN 0944-5900*, 01 2011.

[120] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks?, 2014.

[121] N. Zerrouki, F. Harrou, Y. Sun, and A. Houacine. Vision-based human action classification using adaptive boosting algorithm. *IEEE Sensors Journal*, 18(12):5115–5121, 2018.

[122] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning, 2019.

[123] F. Zhao, Z. Cao, Y. Xiao, J. Mao, and J. Yuan. Real-time detection of fall from bed using a single depth camera. *IEEE Transactions on Automation Science and Engineering*, 16(3):1018–1032, 2019.

[124] W. Zhao. Research on the deep learning of the small sample data based on transfer learning. *AIP Conference Proceedings*, 1864(1):020018, 2017.

[125] Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, July 1988.

[126] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. http://www.cs.wisc.edu/∼jerryzhu/pub/ssl_survey.pdf.