POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Thesis

The RGB-D Triathlon Challenge: Towards Agile Visual Toolboxes for Robots

Supervisors
doct. Mancini Massimiliano
prof. Caputo Barbara
prof. Lamberti Fabrizio
prof. Montuschi Paolo

Fabio Cermelli

December 2018

The RGB-D Triathlon Challenge: Towards Agile Visual Toolboxes for Robots Master thesis. Politecnico di Torino, Turin.

The work in this thesis will be submitted to the International Conference on Computer Vision Systems (ICVS) 2019.

Acknowledgments

In this part, I want to thank all the people who have been close to me during these years. For this reason, I would like to apologize to who does not speak Italian, but the following part does not cover academic arguments and it is intended to be read by relatives and friends that are not very comfortable to understand English language.

Vorrei ringraziare i miei genitori, mio fratello Andrea, i miei zii, i miei nonni e tutta la mia famiglia, che non mi ha mai fatto mancare nulla e che mi ha sempre supportato in ogni decisione. Un ringraziamento speciale va a Chiara, a cui devo moltissimo. Lei mi ha aiutato a superare ogni difficoltà con il sorriso e rende speciale ogni giorno passato insieme.

Vorrei ringraziare la Professoressa Barbara Caputo e Massimiliano Mancini per avermi dato questa opportunità. Non era facile scommettere su un ragazzo che non conoscevano, di un'università lontana dalla propria e che non aveva forti conoscenze in machine learning. Ma loro lo hanno fatto, e spero di essere stato una scommessa vinta. Senza il loro aiuto questa tesi non avrebbe mai preso corpo e probabilmente il mio destino avrebbe preso una piega diversa. Vorrei anche ringraziare il Professor Fabrizio Lamberti, che si è dimostrato sempre disponibile durante il corso della tesi, sia per fornirmi le risorse necessarie, che per aiutarmi a superare le difficoltà lungo il cammino.

Vorrei dedicare un ringraziamento speciale al Professor Paolo Montuschi. Questa tesi è nata grazie al suo contributo, infatti, è grazie a lui che ho potuto conoscere la professoressa Caputo. Inoltre, lo ringrazio per aver speso sempre una buona parola su di me. Infine, vorrei ringraziarlo per quello che ha fatto e sta facendo per una associazione studentesca, il Mu Nu chapter of IEEE-HKN. Vorrei ringraziare il professore e tutti i membri dell'associazione, non solo per aver creato una comunità virtuosa all'interno del Politecnico, ma soprattutto per avermi fatto conoscere delle persone meravigliose, che mi hanno accompagnato in questi anni in numerose esperienze che hanno lasciato il segno nella mia memoria personale e professionale.

Infine, vorrei ringraziare tutti i miei amici, coloro che non si sono mai tirati indietro nel fare qualche follia con me, coloro che mi hanno accompagnato in tante serate di divertimento e in progetti strampalati. In particolare vorrei ringraziare Lorenzo e Pietro che sono diventati la mia seconda famiglia nell'ultimo anno, Marco che è stato il mio compagno di merende al Politecnico, i miei ex-coinquilini e tutti i miei amici di sempre, coloro con cui sono cresciuto e con cui ho passato i miei anni migliori.

Abstract

Convolutional Neural Networks have brought in the last years a significant advance in the perceptual visual abilities of intelligent autonomous systems like robots. Still, these algorithms need to 'overspecialize' in order to achieve performances robust enough to be employed in realistic settings. While this is a suitable strategy for methods design to perform a single visual task, it is clearly suboptimal in the case of a robot system, where multiple and varying visual tasks are requested in order to act and interact intelligently with the environment. This thesis studies the problem of learning without forgetting in the robot vision context, developing the RGB-D Triathlon Challenge, a benchmark evaluation protocol to assess the advantages and weaknesses of possible approaches, and evaluate the current state of the art in computer vision in this new, challenging scenario.

Contents

Chapter 1

Introduction

Some years ago it was often said that computer vision could not compete with human abilities. They were wrong. The use of deep learning algorithms leads to incredible results: now computers can recognize images as well as humans do [19], there are self-driving cars on the road that drive themselves more safely than the average humans do, and applications on the medical sector show that computers could save our lives, for example, by predicting cancer [89]. However, there are still very few applications that use deep neural networks in the real world. These algorithms, in fact, achieve very high results only when they have thousands of parameters and are specialized in performing a single task in a specific domain. Clearly, this strategy could not be used to deploy autonomous systems, such as robots, in the real world because they need to perform multiple and various tasks to effectively interact with the environment.

Computer vision is a long-standing field that deals with how computers can understand and interpret images. Its core tasks are simple visual recognition tasks such as image classification, recognition, and detection. The first outstanding result in computer vision obtained through the application of deep learning was reached in 2012 at the Neural Information Processing System (NIPS) conference, when Geoffrey Hinton and two students, Alex Krizhevsky and Ilya Sutskever, submitted a paper showing that their convolutional neural network, AlexNet [33], was able to classify objects on the ImageNet [70] dataset with an error rate of 16 percent, outperforming the previous best method by 9 percent. Since then, the approaches that take advantage of deep learning have dominated the computer vision field, consistently outperforming solutions based on alternative learning paradigms. For example, [65] and [40] are able to perform object detection in real time, [86] is able to estimate the human pose, [56] performs semantic segmentation, [14] can tracks the motions, and [60] is able to recognize human actions.

The sudden growth of deep learning in computer vision motivated the robotics community to investigate how to take advantage of these algorithms. For an autonomous robot is fundamental the ability to understand and interpret the world. For example, a self-driving car must be able to perceive what there is in front of it and it must detect pedestrians, animals, or cyclist to avoid them. It must comprehend where it is to regulate its speed, and it must interpret the traffic signs to respect the driving rules. Doing all of this without artificial vision might be unfeasible, and certainly very costly. Through this example can be caught the main differences between computer vision and robot vision.

First of all, while computer vision aims to extract information from images, robot vision aims to use that information to make decisions and perform actions. Therefore, making mistakes while extracting information from data is much more dangerous in the robotic context, because it can lead to bad decisions or actions and compromise the ability of a robot to complete its task, with possible dangerous consequences for humans. Thinking again to the self-driving car example, if the system makes a mistake detecting pedestrians, it may run over them with catastrophic consequences.

Another important difference is that a robot must take actions depending on its context. It is important that the car understands where it is and that it regulates its behavior accordingly. If the car is in an urban context, it should slow down and be very careful because pedestrians can appear suddenly.

A robot is an agent that acts in the space, therefore, the depth is very important for the artificial sight. The most common approach of computer vision is to analyze only flat colored images (RGB images), that project the three-dimensional world into two-dimensional images, leading to the loss of the depth information. Robotic vision can exploit the diffusion of low-cost RGB-D cameras that add to the standard RGB images the depth information, i.e. the distance between the scene and the camera pixel by pixel. The depth allows robots to better understand the geometry of the scene, increasing their ability of estimating the distance of the objects. The importance of the depth became even clearer in the self-driving car example. By analyzing the distance between the car and the objects, the car is able to estimate perfectly the distance of objects in the scene, a crucial ability to avoid pedestrians and accidents.

In computer vision, the state of the art deep neural networks achieve very high accuracy on many tasks but they require millions of parameters and a lot of computations both for training and inference. Often these algorithms are executed by clusters with many GPUs, therefore, with a lot of computational resources, and with terabytes of memory (RAM). Obviously, this is not applicable in the robotic context. Robots must make decisions in real time but they can neither mount a cluster to perform the computation due to both spatial and energy constraints, nor connect to a cluster through th Internet due to the latency that it will generate. Thus, it is necessary to use models with fewer parameters than those used in computer vision.

To deal with this issue, multi-task learning is considered a very promising approach. Multi-task learning in computer vision refers to models that perform different tasks at the same time, by sharing parameters and computation among them. Therefore, this approach reduces the number of parameters needed by sharing them between different tasks while decreasing the time needed for training and inference.

The ability to add new tasks sequentially to an existing model is crucial [51], [85]. Early methods such as [7] assumes that all tasks that would be performed are known before the training. This assumption is unrealistic, because applications evolve over time and the need of introducing novel tasks can arise when the model is already deployed. Therefore, is essential the ability to add tasks sequentially without modifying the performance over tasks that are already optimized. A well-known problem of sequential learning is that it typically suffers from forgetting or degrading the performances on the previous tasks. This phenomenon, referred to as *catastrophic forgetting* [15], [50], must be kept into account by developing models able to avoid it or limit its effect. Moreover, training data of old tasks may be unavailable or outdated, thus, to be as much as possible realistic, the old training data should not be used while training for a new task.

For example, suppose to have a self driving car that is able to perfectly detect pedestrians and see traffic lights. The car was initially trained on a standard dataset and next the neural network's parameters were optimized while making real experience in cities, but now the company wants to extend its market and update the car to drive also in the countryside. Thus, the car needs to perform new tasks, such as detecting animals, to be deployed in this new scenario. However, training again the car from the initial dataset will degrade the performance of the car, because the experience done in the city would be lost. Then, the novel tasks must be learned sequentially, using an approach that leverages only on the data of the new tasks. Summarizing, the car should be able to learn new tasks without increasing too much the number of parameters of the model, without suffering catastrophic forgetting, and without accessing the old training data. This thesis will focus on this setting that is related to multi-task, sequential, life-long and incremental learning.

Many mainstream approaches for adapting deep models to novel tasks do not respect the constraints of this setting. For example, fine-tuning the network parameters to the new task produces a powerful model on the novel task but it degrades the performances on the old ones. To avoid this problem, one can think to replicate the network and train a separate copy for each new task. This approach preserves exactly the performances on the old tasks at the expense of introducing as many copy of the network's parameters as the number of the tasks to be performed, and indeed breaking the constraint of adding only a few parameters to learn the new task.

An interesting approach is designing models that introduce task-specific network parameters [46]–[48], [62], [63]. This approach consists in adding few extra task-specific parameters that are optimized during training on that task, while the other network's parameters are kept frozen. Interestingly, these methods obtain performances comparable to fine tuning the whole network on the new tasks without suffering catastrophic forgetting and at the cost of introducing only a small fraction of the network's parameters per task (between 3% and 15%).

Robots can also make use of multi-task learning to exploit the relations between tasks and improve the overall performance. For instance, robots can exploit the relation between object classification and scene recognition. The real world presents many semantic regularities and robots should reason about the semantic relations between the scene and the objects in it. For example, it is very common to found cups in the kitchen or on the dining table, but it is not probable to find them in a bathroom. Currently, state of the art detection systems [65], [66] do not exploit this relationship, losing a big opportunity. If robots are able to understand their context, i.e. the scene they are on, they can use semantic relationships as prior to detect objects and, vice-versa, they can use the information of the detected object to improve the performances on scene recognition. Robots can also exploit the relation between object classification and pose estimation. When a robot must estimate the rotation of an object, identify the class of the object is really helpful as a prior for the estimation. On the other hand, to classify an object can be useful to know how it is orientated in the image an the pose of the object can be extracted and used as a prior for the classification task. Combining two or more tasks allows robots to reason about the geometries and semantics of their surroundings, an ability necessary to make a step toward the general artificial intelligent.

Contribution of this thesis This thesis proposes a benchmark for multi-task learning in robotic context: the RGB-D triathlon challenge. It aims to motivate researchers in developing new models able to perform different visual tasks on the constrained, realistic robotic setting. It is crucial for these models to share many parameters among different tasks and to be able to learn sequentially new tasks, without accessing the training data of the other tasks and suffering catastrophic forgetting.

The challenge proposes three fundamental tasks: object classification on the RGB-D Object Dataset (ROD) [34], pose estimation on LineMOD dataset [22], and scene recognition on NYU Depth V2 dataset [53]. These datasets provide RGB-D images taken by robots in real environments. Thus, three settings are proposed for the challenge: the RGB setting, in which can be used only RGB images, the D setting, in which are only available depth images, and RGB-D setting, where both RGB and depth images are provided.

It is defined a simple accuracy metric for every task, and, to assess the performances of methods on the multi-task benchmark, five different metrics that bind the results of the three tasks with the number of parameters used to perform them, and provide the overall score of the challenge are discussed. As it is shown in the experiments, the linear score demonstrated to be the fairest metric and it is proposed as the metric for the RGB-D triathlon challenge.

This work evaluates against the challenge some methods that have shown good performances in sequential and multi-task learning: serial [62] and parallel [63] residual adapter, Piggyback [47], and binarized affine fransformation (BAT) [48]. These methods are compared with two baseline methods: fine-tuning the whole neural network on the new task and considering the network as a feature extractor by training only the task-specific classifier. Fine-tuning the network parameters can be seen as an upper bound for the performances of the other methods.

The methods showed an outstanding ability to adapt to novel task. This demonstrates that neural networks, if well designed, can share a lot of parameters among different tasks without degrading the performances on them. Moreover, in some cases, the performances of these methods are better than those obtained by fine-tuning the network's weights, demonstrating that performing multiple tasks at the same time can even lead to better results.

However, the tasks that have been chosen require a similar neural network architecture. To obtain a real agile toolbox for robotic vision more complex tasks shall be included, such as object detection, semantic segmentation, or grasp prediction, that require different neural network architectures to be performed.

The remaining part of the thesis discusses the related works (chapter 2) that emphasizes the importance of the problem for the community. Then, the technical foundations to understand this work are introduced in chapter 3, explaining the tasks presented by the challenge, presenting the basic concepts related to Neural Networks and Convolutional Neural Networks, and summarizing the methods that are compared in the challenge. In chapter 4 it is introduced the challenge, diving into the details of the tasks to be performed and discussing the evaluation metrics. Chapter 5 reports the results of the methods and explains how they was implemented. Finally, in chapter 6, the fundamental points of this work and its future possible developments are discussed.

Chapter 2

Related Works

The competition pushes humans into achieving increasingly higher performances since the Olympics of ancient Greece. In the same way that competition is good for sports, it is also good in pushing forward the state of the art in artificial intelligence and machine learning. In recent years, the computer vision community has proposed challenges and competitions such as ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [70], Common objects in context (COCO) [39], Places [98]. These competitions have encouraged the researchers in developing new methods that led to improvements in object recognition, object detection, semantic segmentation, place recognition, and visual question answering. The challenges motivated the researchers with interesting problems, provided new datasets and evaluation metrics to establish which method solved best the problem. The most popular challenge is ILSVRC [70] that evaluates algorithms for object detection and image classification at a large scale. It is considered the standard benchmark to monitor the convolutional neural networks performances in image classification. It introduced a very large dataset made of more than one million images taken from the Internet, divided into 1000 classes. The benefit of this challenge can be seen by considering the growth of the performances: in 2010, when the challenge was launch, a good error rate was around 25%; 7 years later, the best methods in classifying objects reached error rates of nearly 3%, a performance comparable to human abilities.

The robotics community has seen a great opportunity in the growth of the computer vision and especially of the deep learning methods. In the last few years, researchers were very engaged in discussing how deep learning could be applied and what are the opportunities that it offers in the robotic field [82]. For instance, [36], [74] demonstrate the use of deep learning for grasping and manipulating objects. [36] describes an approach that learns eye-hand coordination for robotic grasping using a monocular camera and a convolutional neural network that predicts the probability of success of the grip. [74] presents a deep learning approach that combines

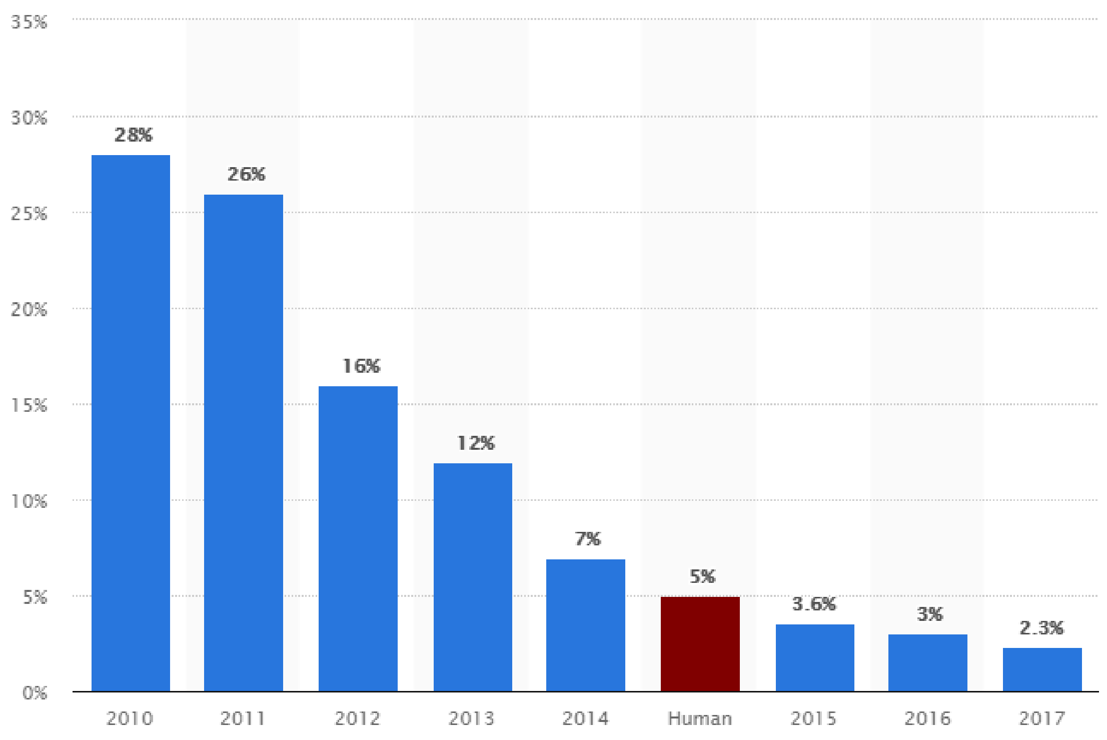


Figure 2.1: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [70] winning algorithm's classification error rate in chronological order.

object detection and semantic segmentation to perform autonomous robotic manipulation in cluttered scenes. Moreover, 10% of the papers submitted to ICRA 2018 used as keyword *Deep learning in robotics and automation*, and it was initiated a new Conference on Robot Learning (CoRL). However, even if robotic context can take advantage of the progress in computer vision, it presents a different scenario. Differently, from computer vision, where the agent is static and has the goal to extract information over images, a robot is a dynamic agent that acts in, and interacts with, the world with the goal to make decisions and perform actions. It introduces many more challenges such as the deployment in open-set conditions, incremental learning, semantic reasoning, active learning, active vision, and transfer learning.

Researchers still lack benchmarks to evaluate the performance of the deep learning methods they develop in this new context. This theme is very important for the robotics community, in fact, it was discussed recently in the main conferences. In CVPR 2018 it was held a workshop named *Real-World Challenges and New Benchmarks for Deep Learning in Robotic Vision* [91] in which were discussed new benchmarks with the aim of recreating the success of the computer vision challenges also in robotic vision. In this conference were presented many new challenges: Paris-Lille-3D [69] proposes a new benchmark for scene segmentation and classification, [17] discusses new metrics and experimental paradigms for continual learning, Falling Things [87] is a synthetic dataset for object detection and pose estimation, the active vision dataset [2] is a benchmark to evaluate performances of active learning.

However, none of these challenges focus on a very important aspect: *multi-task learning*. Multi-task learning has been proven to be effective in deep learning by Caruana [7]. In its standard approach [7], it is implemented in neural networks by sharing the early layers of the architecture among different tasks. The tasks are jointly learned by these layers by means of back-propagation. The final layers of the network instead are task-specific and trained only on the relative task. The main advantage is that it offers a way to learn more general representations by implicitly adding an inductive bias caused by the constraint on the learned representation. It has demonstrated to be effective in many different fields such as natural language processing [9], speech recognition [12], drug discovery [61], reinforcement learning [21], and computer vision [31], [97].

Some challenges were proposed to deal with multi-task learning. The Robust Vision Challenge [67] aims to encourage researchers to develop visual systems that are robust and can perform several tasks (reconstruction, optical flow, semantic/instance segmentation, single image depth prediction) across benchmarks with different characteristics. The Natural Language Decathlon [49] is a challenge that spans ten tasks in the context of natural language processing: question answering, machine translation, summarization, natural language inference, sentiment analysis, semantic role labeling, relation extraction, goal-oriented dialog, semantic parsing, and commonsense pronoun resolution. The Visual Decathlon challenge [62] is a challenge on multiple domain learning, a problem related to multi-task learning. Instead of trying to learn different tasks, it aims to learn the same task, image classification, within different domains. The challenge is a benchmark that evaluates how well neural network models are able to learn simultaneously ten very different visual domains.

Multi-task learning is also useful in robotics. In [38] it is presented a method that tries to understand scenes by exploiting several semantic relations between objects and the overall scene using conditional random fields. Place categorization and improved object detection have been reached utilizing learned scene-object priors, as demonstrated in [83]. In [96] the authors developed a method to perform holistic scene understanding extracting information from the context using a deep neural network. However, in these works, multi-task learning is only a tool to achieve better performances toward a primary goal. The secondary tasks are auxiliary and the method tries to improve their performances only for achieving better results on the primary task. This thesis aims to motivate the community to develop methods able to perform multiple tasks at the same time, without differentiating between auxiliary and primary tasks. This is useful for two reasons: (i) the neural network can learn better representations by the effect of the inductive bias and by exploiting the semantic relationship between tasks, (ii) by sharing the network parameters we can use few parameters per task, reducing the amount of memory and computation required to perform the various tasks.

This thesis proposes a new benchmark in the robotic vision context to evaluate the performances in multi-task learning. To the best of the candidate's knowledge, this is the first multi-task benchmark in the robotic vision context.

Chapter 3

The Landscape

This chapter introduces the technical foundation on which this thesis is based. Section 3.1 describes the tasks chosen as target of the challenge. Section 3.2 briefly explains the concept of Neural Networks, focusing on their convolutional version (CNN). Lastly, Section 3.3 describes the methods used as baselines for the challenge.

3.1 Defining the robotic vision challenge

A robot is an active agent that interacts with the world. It perceives the world with a set of sensors and it should exploit all the available information to decide, plan, and execute various tasks. Mistakes can lead to catastrophic results that not only can harm the robot itself, but even the humans around it. In such scenario the ability to see is essential. Since a robot must perceive the environment around it and take actions based on what it sees, thus its perception capabilities must be highly accurate.

For this reason the proposed challenge should tackle tasks which better allow the robot to understand the scene it has in front. Under this consideration, multiple tasks are essential (e.g. object classification and localization, instance segmentation, scene recognition). As a starting point the challenge proposes three fundamental tasks for a visual system:

- *Object recognition* concerns the ability of identifying the semantic category of an object present in an image. This task is fundamental for a robot and it is the foundation to perform more complex tasks (e.g. object manipulation).
- *Pose estimation* aims to define the orientation of an object relative to the agent. This task allows a robot to handle objects and understand their role in the scene. Even this task is the base to perform complex tasks such as grasping and manipulation.

- *Scene classification* concerns the ability to recognize where the agent is semantically (e.g. in what type of environment among dining room, outdoor, etc). This task is necessary for spatial competencies, robot navigation and human robot interaction.

Even if the tasks are very different, both object recognition and scene classification can be reduced to image classification. In fact, both tasks aim to process an image producing a class label which denotes the global semantic content of the image itself.

Finally, notice that while the tasks considered are three, the challenge can be easily extended to include other different tasks (e.g. action recognition), even involving different output representations (e.g. object detection).

3.1.1 Image classification

Image classification aims to assign to an input image one label from a fixed set of categories. This is one of the classical problems in computer vision that, despite its simplicity, has a large variety of practical applications and it is the basic building block of more complex tasks such as object detection [66] and semantic segmentation [41].

To better explain this task, let us suppose that we want to classify an image in one of 4 categories: *dog*, *hat*, *mug* and *cat*. Given the image in figure 3.1, the classification task aims to predict the correct label of the image (in this example, a *cat*). The algorithm receives as input the image as a 3-dimensional matrix of size $w \times h \times 3$, where w is the width in pixel, h is the height in pixel, and 3 is given by the fact that images are represented by three color channels: red, green and blue (or RGB for short). Each value of the matrix is an integer with a value that ranges between 0 (black) and 255 (white). The cat image of the example is 248 pixels wide and 400 pixels tall, for a total of 297600 integer values. The image classification task consists in converting these integer values into a single label, the class of the image. The output of the algorithm is a probability vector which denotes the probability of the image to belong to each of the known classes, taking as predicted label the one with maximum probability. We can see that, in this example, the algorithm is quite confident (82 %) that the image portrays a *cat* and not the other classes, thus the final prediction of the algorithm would be correct.

Although the task of recognizing concepts in images is trivial for a human to perform, it presents many challenges for a computer or robot vision system among which:

- *Viewpoint variation.* An instance of an object can be oriented in many ways with respect to the camera.

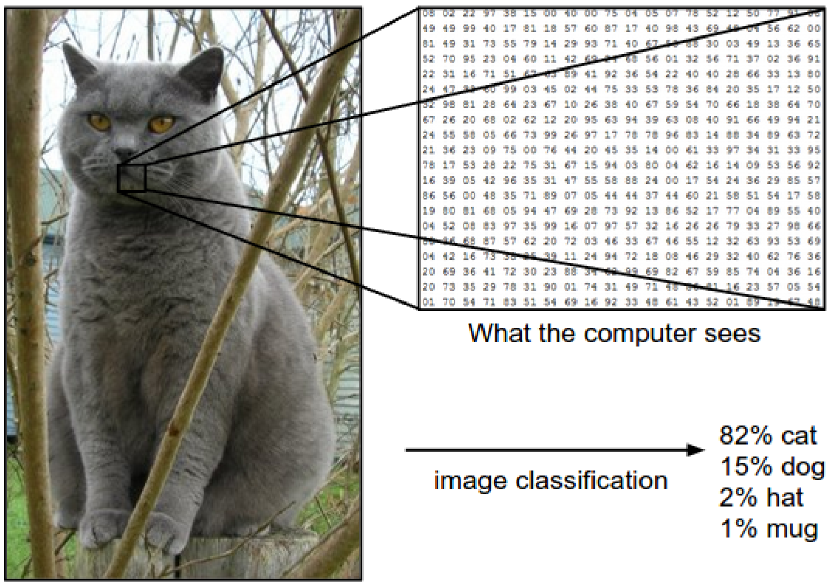


Figure 3.1: The image is taken from [28]. The goal of image classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. The algorithm sees the images as a 3-dimensional matrix of size $Width \times Height \times 3$. Each value of the matrix is an integer that ranges from 0 to 255 that represents the intensity of the color channel (the most common representation is given by the three colors red, green, blue and known as RGB).

- *Scale variation.* Objects often exhibit variation in their size.
- *Deformation.* Objects that are not rigid bodies can be deformed in many ways.
- *Occlusion.* The objects in the image can be overlapped by other objects and only a small part can be visible.
- *Illumination conditions.* The effects of illumination can change the integer values drastically, even if for humans the difference is imperceptible.
- *Background clutter.* The objects to be recognized may camouflage into the environment, making them hard to identify.
- *Intra-class variation.* The object classes can be relatively broad. There are various instances of these objects, each with their own appearance.

An image classification model must consider all these possible variations and must be robust enough to do not suffer a decrease in performance under different input conditions.

Many approaches have shown to be successful in addressing this task such as building K-Nearest Neighbor or Support Vector Machine classifiers on top of image feature representation such as histogram of oriented gradients (HOG) [11], speeded up robust features (SURF) [3], scale-invariant feature transform (SIFT) [42], [43]. However, all these approaches have been recently outperformed by *Convolutional Neural Networks* (CNNs) [19], [33], [75], [84], [95]. These architectures replace precomputed features with trained ones, merging the feature extraction and classification phases in a single architecture, trained end-to-end.

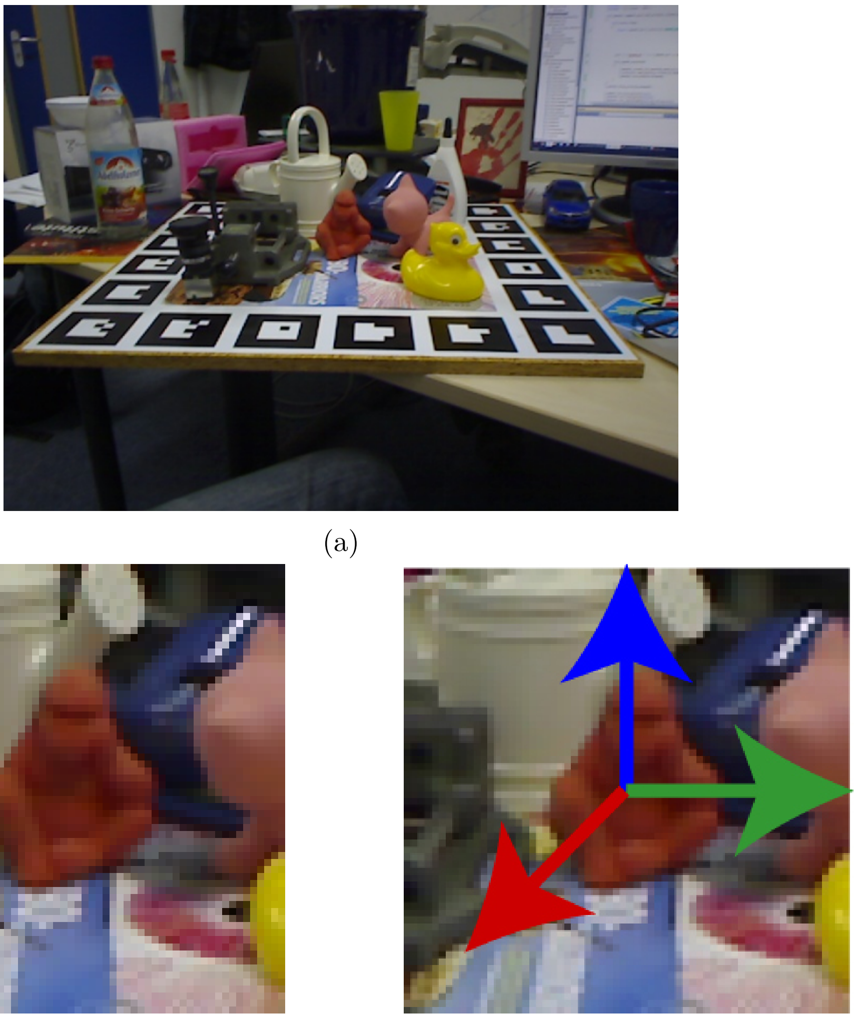
The improvement that these algorithms brought to this task can be noted from the top performances on the Imagenet Large Scale Visual Recognition Challenge (ILSVRC) [70], the de-facto standard benchmark for image classification. Imagenet is a dataset made of more than one million images taken from the internet, divided into 1000 categories. In 2010, the first year of the challenge, a good error rate was about 25%. In 2012, Geoffrey Hinton, Ilya Sutskever, and Alex Krizhevsky submitted the first model of a deep convolutional neural network called AlexNet [33] that reached an incredible result, achieving an error rate of 16%. In 2017, after seven years, the top-5 accuracy in classifying objects within the dataset rose from 71.8% to 97.3%, comparable with human abilities [71].

3.1.2 Pose estimation

A fundamental task of computer vision and robotics is to determine the object's position and orientation relative to some coordinate system. This information is important to allow, for example, a robot to interact with the object and manipulate it.

The term pose refers to the transformation between the object and the camera and it is often captured using six parameters that represent both orientation and position: azimuth a, elevation e, camera-tilt c, distance d to the camera and image translation made of two parameters, x and y. The pose estimation task can be decomposed in two sub-tasks: the localization of the object in the image, which is represented by the last three parameters (d, x, y), and the estimation of the rotation matrix R between the object and the camera, captured by the first three parameters (a, e, c).

Even though many works try to perform both sub-tasks at the same time [59], [93], this work assumes that the object is always centered in the input image, implicitly solving the first task and reducing the pose estimation task only to the latter sub-task as in [44], [90]. This is motivated by the fact that object localization is a well-known task in computer vision and can be performed by an object detection system, for which exists many implementations such as [40], [65] that can do it even in real-time. So, for us, the pose estimation task consists in converting an image into a rotation matrix representation, such as the one made by the three angles



(a)

(b) (c)

Figure 3.2: The images report an example of the pose estimation task. The image (a), taken from the linemod dataset [22], is given as input to the object detection system, that localizes the ape in the center (the object of interest) and outputs the figure (b). Finally, the rotation of the object is estimated as we can see from the arrow represented in (c).

azimuth, elevation and camera tilt, or axis-angle representation, or quaternions.

In the literature there are many works that address this task, from [20], [58], [73] to more recent methods that use Convolutional Neural Networks (CNN) [44], [57], [81], [88], [90], [92]. The CNN-based methods use very different approaches. For instance, there are methods that treat pose estimation as a classification problem, fragmenting the pose space into bins. These methods can be divided into two groups: the first contains those that predict 2D keypoints from the image and then use a 3D model to evaluate the 3D pose given these keypoints [57], [92]; the second group contains those methods that predict directly the pose given the image [81], [88]. An interesting method is the one in [90]. It uses a CNN to map the image space to a descriptor space and then searches, through the nearest neighbor algorithm, the most similar descriptors to predict the pose and the object class.

There exist also CNN-based methods that consider pose estimation as a regression problem. The challenge is that 3D pose space is a non-Euclidean space, hence, CNN algorithms need to take it into account. To this extent, in [44] the authors try to solve the problem by designing a suitable representation and a loss function that respects the non-linear structure of the pose space. They specifically study two representations of the rotation, axis-angle and quaternions, and model their constraints using non-linearity in the output layer. Moreover, they propose to (i) use the geodesic distance on the space of rotation matrices as a loss function; (ii) exploit the knowledge about the object, proposing a network architecture that is made by part shared between all objects (called feature network) and a part that is object-specific called pose network.

This work takes inspiration from [44] and treats the pose estimation task as a regression problem. Similarly to that work, this thesis represents the rotation using quaternions and uses the geodesic distance as loss function. Differently from it, it is proposed a network architecture that both predicts the object class and regresses the pose of the object.

3.1.3 From RGB to RGB-D: the importance of the third dimension

A robot is an agent that acts in a 3D environment, thus, it is important to consider not only the 2D representation of the sight but also the depth to effectively act in the real world. The diffusion of low-cost RGB-D cameras gives an incredible opportunity to robotics, because they add to the RGB image also the distance from the various parts of scene to the camera. Obviously, exploiting this information can lead to build more robust models, achieving better results in various tasks.

This work proposes three settings: the setting where only RGB data are available, the setting where only depth information is provided and the setting where both RGB and depth data are given. Moreover, it is provided a simple way to combine depth and RGB information on a pretrained architecture and it is studied the effect of this additional input to the final performances of the employed models.

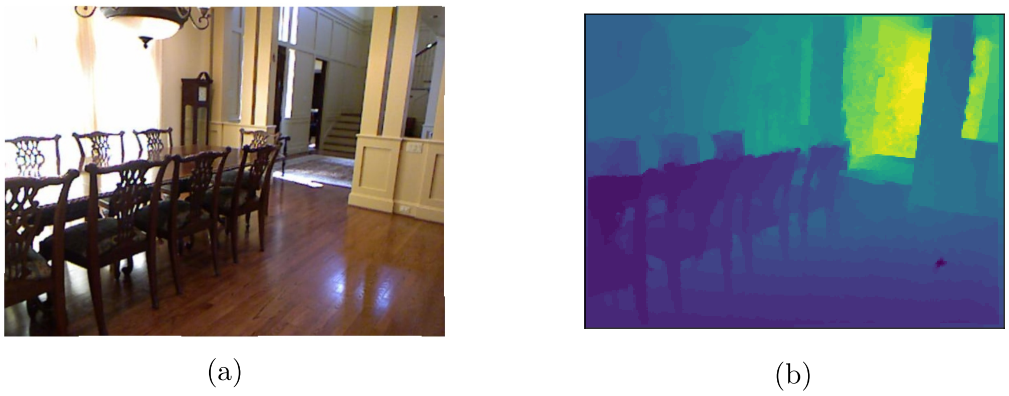


Figure 3.3: The images show a dining room. In the left (a) there is the RGB image, while in the right (b) there is the depth image that was colored for better visualization.

3.2 Convolutional neural networks

3.2.1 Neural networks

Neural networks are a data processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. They are composed by a number of highly interconnected processing elements, called neurons. Each neuron has multiple input connections and, to each connection, a weight is assigned. Every neuron performs a function: it takes as input a vector $x \in \mathbb{R}^m$ and performs a dot-product between the input and its weights $w \in \mathbb{R}^m$, adding the bias b and applying an activation function to it:

$$y = f(\sum_{i=1}^{m} w_i x_i + b). \tag{3.1}$$

In the formula, f indicates the activation function, which is typically non-linear. The most used functions are the *sigmoid*, the *tanh*, the *ReLU*, or the *step function*. The weights and the bias term are values that each neuron stores and these are typically learned in the training phase.

A neural network is structured in layers $[l_1, \ldots, l_n] \in L$, that are a collection of neurons. Layers are interconnected in a way that the output of the l_i is the input of the following layer l_{i+1}. The input of the first layer l_1 is the input data, while the output of the last layer l_n is the result. In the most common layer type, the fully-connected, the neurons within a layer are not inter-connected, but each neuron is connected to all neurons of the previous and following layers.

Training a neural network with a supervised approach means learning the weights

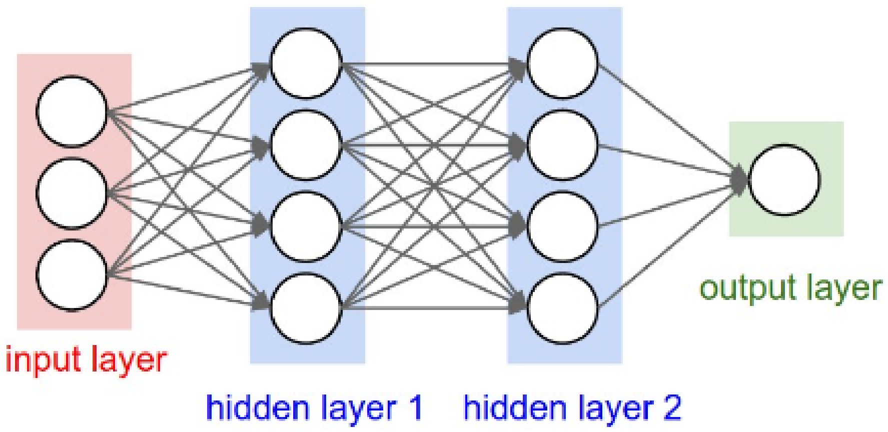


Figure 3.4: 3-layer neural network with three inputs, two hidden fully-connected layers of 4 neurons each and one output layer. Notice that in both cases there are connections between neurons across layers, but not within a layer. Image taken from [28]

and the bias term for each neuron that minimize the difference between the output produced by the network and the desired output. Before training, the neuron's weights are typically initialized with random values close to zero. Training is then made by alternating two phases: the forward propagation and the back-propagation. In the first phase, the input is passed to the first layer and the output is produced by computing values layer by layer until the last one. The produced output is evaluated by a loss function that estimates how far it is from the desired one. After calculating the loss value, the back-propagation phase begins. In this phase the loss value (or error) is propagated backward i.e. it is passed to the output layer and passes through the network until the first layer, and the neuron's weights and bias are updated according to some policy. The loss-function estimates the error between the output produced by the current set of weights and the desired output. The goal of the training is to find the set of weights that minimize the loss function. The most common update policy is the *Gradient Descent*, that updates the weights by descending the gradient of the loss function computed with respect to the weights themselves. Given a loss function with this form:

$$C(W, b, x^i, y^i) = \frac{1}{2}|f_{W,b}(x^i) - y^i| \tag{3.2}$$

where x^i is the input, y^i is the output, W, b are the weights and the bias term, the neuron's weights are updated using the gradient of the loss function, with the following update rule:

$$W_{i,j} = W_{i,j} - \alpha \frac{\partial C(W,b)}{\partial W_{i,j}} \tag{3.3}$$

$$b_i = b_i - \alpha \frac{\partial C(W,b)}{\partial b_i} \tag{3.4}$$

The parameter α takes the name of *learning rate* and specifies the degree of the learning. This hyper-parameter is very important because it sets the step size of the update in the direction pointed by the gradient.

Usually, α is higher in the first iterations of the training, to perform larger steps, and it decreases during the training to perform smaller steps when we are in proximity of a local minimum to facilitate convergence. The main drawback of this technique is given by the fact that the shape of the loss function is often non-convex and there can be a lot of local minima where the algorithm can remain blocked, preventing the weights and the biases to find the optimal value.

There are many strategies to apply this optimization policy that mainly differs in the amount of input data processed by the network for the calculation of the gradient. The *Vanilla Gradient Descent* (or *Batch Gradient Descent*) is the simplest version: the entire dataset is used to compute the gradient and to update the weights. In large scale application, where the training dataset contains millions of example, it is not efficient to perform a single update to the weights after the computation of the loss function over the whole dataset. Thus, in these cases, the Vanilla Gradient-Descent is replaced by the *Stochastic Gradient-Descent (SGD)* that computes the gradient after processing batches of data. The batch size is an important hyper-parameter that affects the performances of the algorithm and typical value ranges between 16 and 256 for classification tasks. The SGD works well because the examples in the training data are correlated and the gradient of a batch is a good approximation of the gradient computed over the full dataset. Therefore, using it increases efficiency because it performs more frequent parameter updates, reducing the time needed for the training.

Neural networks have a high representational power, in fact, they are universal function approximators [10]. Increasing the size and the number of layers in a neural network increases the capacity, i.e. the space of representable functions. While increasing the capacity allows the representation of more complicated functions, this has also a drawback: the network is more exposed to overfitting. *Overfitting* occurs when the model fits the noise of the data, decreasing the overall performance. There are many regularization techniques preventing overfitting such as L2 regularization, early stopping and dropout [78]. L2 regularization penalizes the squared magnitude of all parameters adding a term $\frac{1}{2}\lambda w^2$ directly in the loss function. It can be interpreted as a way to limit peaky weight values, encouraging the network to use all of its input rather than some of its input a lot. Dropout is a very effective

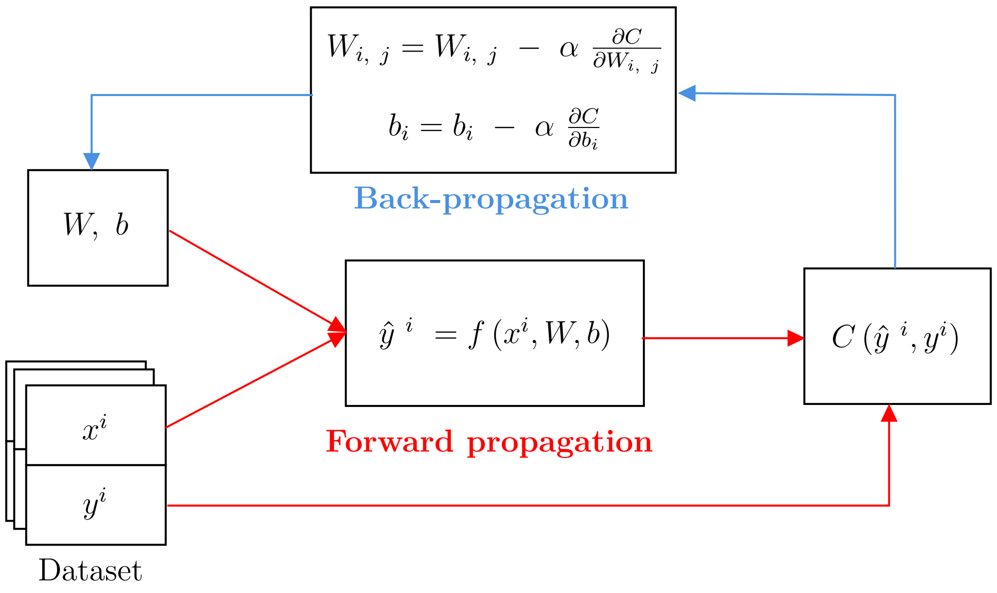

$W_{i,\ j} = W_{i,\ j} \ - \ \alpha \ \frac{\partial C}{\partial W_{i,\ j}}$
$b_i = b_i \ - \ \alpha \ \frac{\partial C}{\partial b_i}$
Back-propagation
$W,\ b$
$\hat{y}^{\ i} \ = f\,(x^i, W, b)$
$C\,(\hat{y}^{\ i}, y^i)$
x^i
y^i
Forward propagation
Dataset

Figure 3.5: Illustration of the steps required to train a neural network with a supervisioned approach. In the forward propagation the input x^i, the weights W and the biases b are used to compute the output $\hat{y}^i$. The loss function is then computed comparing $\hat{y}^i$ and y^i. Then, the back-propagation starts: the gradient is computed and the weights and the biases are updated.

and simple regularization technique that complements the other methods. This technique left out randomly some neurons during the forward phase. It assigns a probability p to the neurons to be switched off and this probability can even differ between layers. Using these techniques improves the network ability to generalize over example data and boost the performance.

3.2.2 Convolutional neural networks

Convolutional Neural Networks (CNN) are similar to a standard neural network: they are made by neurons that have weights and biases that should be optimized to minimize a loss function. However, CNNs make an explicit assumption: that input points in different locations may need the same features (e.g. as for different parts of an image) and this allows to make an efficient architecture that simplifies the computation of the forward function and reduces the number of parameters within the network.

First of all, Convolutional Neural Networks exploits the dimensionality of the input images, that are represented as integer matrices of dimension $Width \times Height \times 3$ (see Section 3.1.1) and, as shown in figure 3.6, they arrange neurons in three dimensions: width, height and depth. Thus, every layer of a CNN transforms a

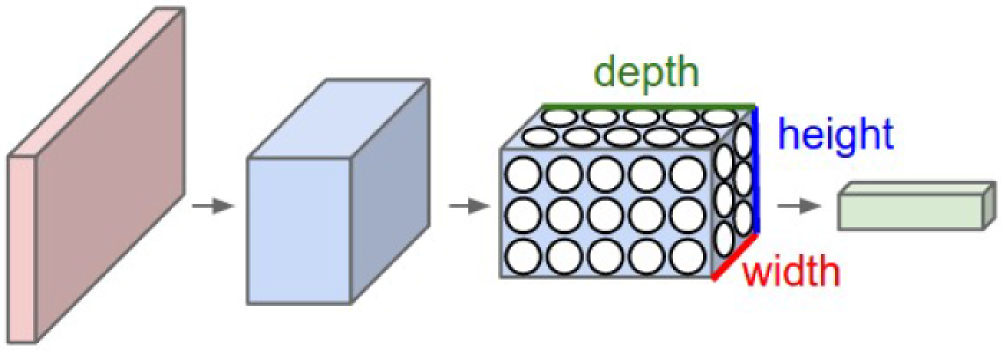


Figure 3.6: The structure of a convolutional neural network. Image taken from [28]

volume of activations $W \times H \times D$ to another through a differentiable function.

A standard CNN is a sequence of different components. In the following some of them will be described.

Convolutional Layers are inspired by the structure of the visual cortex in the animal world. In this part of the brain every neuron looks at only a small region of observation, called receptive field, responding to precise stimuli. The receptive fields of different neurons are overlapped in a way that covers the entire field of view.

The convolutional layer architecture emulates it by applying the convolution operation to a small region of activation volume. A convolutional layer is made by a set of learnable filters that are 3-dimensional matrices with small width and height but with a depth equal to the depth of the input volume. For example, given an input volume of $32 \times 32 \times 3$, the filters have the form $w \times h \times 3$, where typical values for h and w range between 1 and 7. Moreover, the filters are typically squared ($h = w$). h and w are important hyper-parameters known as kernel size and represent the *receptive field* of the neuron.

During the forward pass each filter is convoluted with the input volume. It means that, for each region of the input volume, it computes the dot product between the filter and the input values in any position. The result of the convolution between one filter and the input volume gives a 2-dimensional activation map that represents the responses of that filter at every position of the input volume. Intuitively, we can see this operation as looking for patterns in the input image where each filter searches for a different pattern and, when the filter founds it in a region, a positive value is stored in the activation map for that specific region.

This mechanism is an efficient *parameter sharing scheme*, allowing to reduce the number of parameters. It starts from this assumption: if a feature is useful in position (x_1, y_1) then it is also useful in another position (x_2, y_2). From this assumption it turns out that it can be wasteful to train many filters to search for the same pattern in many positions: in fact, this would require to use as many filters as the input surface pixels. It is more efficient to reduce the number of parameters by training a single filter to search for a pattern in the whole input surface. Anyway, the assumption may not make sense in some specific cases. If we know that the input images have a structure where we should expect to have some patterns in different specific positions, we should learn different filters and apply them only in the relative position. A common way to solve this problem is to relax the parameter sharing scheme and use a so-called *Locally-Connected Layer*.

The output volume of a convolutional layer has size $w \times h \times d$ and it is controlled by three hyper-parameters: depth, stride and padding. The *depth* controls the third dimension of the output volume d and it is equal to the number of filters that we use. The filters elaborate the same input volume but they look for different patterns and produce different 2D activation maps. Thus, after the convolution layer we obtain d 2D activation maps where d is the depth or the number of filters used. The two dimensions w and h, that represent the size of the activation maps, are controlled by the other two parameters. The *stride* is the step that each filter takes sliding the input volume. When the stride is one, we apply the convolution at all coordinates of the input volume; when the stride is two, the convolution is performed one pixel every two. The same principle applies for all values: if the stride is n the convolution is performed one pixel every n. Anyway, it is uncommon to find stride values grater than two because, as we will see later, the stride reduces the output volume proportionally to it. The *padding* adds zeros to the edge of the receptive field. It is useful because it allows to control the output volume, especially to keep it at the same dimension of the input volume.

Given the dimension of the input volume, the kernel size, the padding, and the stride, the dimension of the activation maps can be obtained using the following formula:

$$w = \frac{W_i - F + 2P}{S} + 1 \tag{3.5}$$

where w is the resulting width, W_i is the width of the input volume, F is the kernel size, P is the padding, and S the stride. To obtain the resulting height, it is sufficient to replace the width of the input volume with the relative height.

To summarize, the output volume size is $w \times h \times d$ where w and h are obtained with the previous formula and d is the depth.

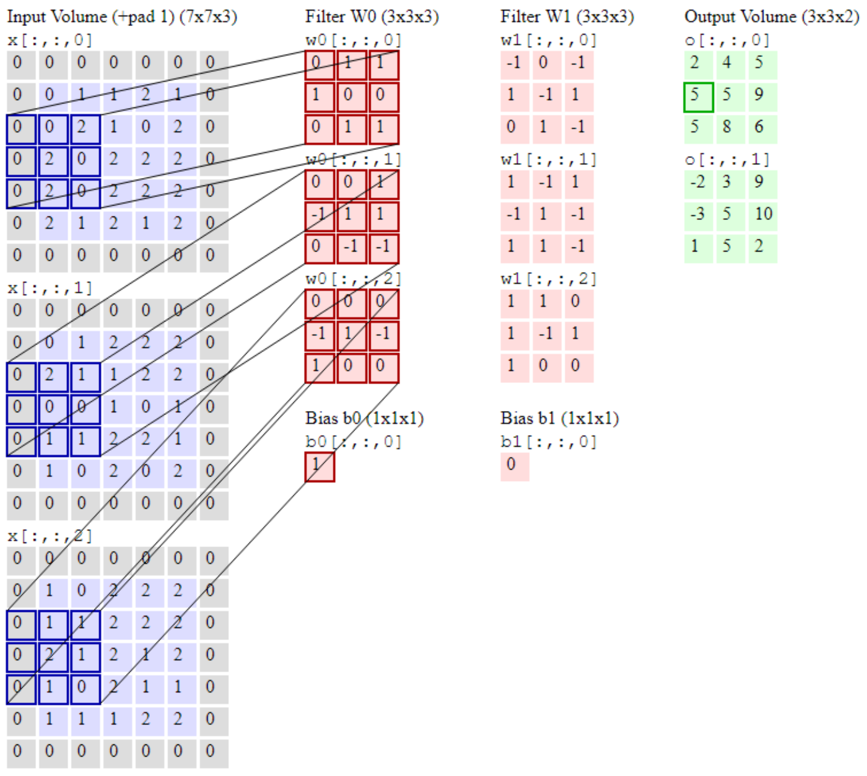


Figure 3.7: Example of a convolution operation between 2 different filters with the input volume. The input volume is $7 \times 7 \times 3$, the kernel size is 3×3, the stride is 2 and the padding 1. Therefore, the output volume is $3 \times 3 \times 2$. The image is taken from [28].

Pooling layer is another common layer that can be found in a CNN architecture. Its main goal is to decrease the surface (height and width) of the volume of activations, obtaining a reduction in the number of parameters and in the computation time. It accepts a volume size of $W_i \times H_i \times D_i$ and requires two hyper-parameters: the spatial extent F and the stride S. The pooling operation is performed on each depth slice (i.e. the single 2-dimensional section along the depth axis of the activations volume) sliding a simple function, such as maximum or the average, over the receptive field. The dimension of the receptive field is controlled by the parameter F, in fact, it contains a region of activations of dimension $F \times F$. The stride parameter controls the step of the sliding: as in the convolutional layer, with a stride of one it is computed for each pixel (x,y) of the depth slice, with a step of 2 it is computed once every two pixels, and so on. The output width is given by the following formula:

$$w = \frac{W_i - F}{S} + 1. \tag{3.6}$$

The output height h can be obtained in the same way, substituting H_i to W_i. The resulting output volume has size $w \times h \times D_i$, where w and h are computed with the previous formula, and D_i is the input depth.

It is important to note that the pooling layer does not have any learnable parameter. Once the hyper-parameters are set, it can apply its function over and over. For this reason the pooling layer does not perform any update operation in the back-propagation but it only propagates the errors backwards.

It is worth to note that recently, many researchers proposed to discard the pooling layers [77] in favor of an architecture that uses larger stride in the convolutional layers.

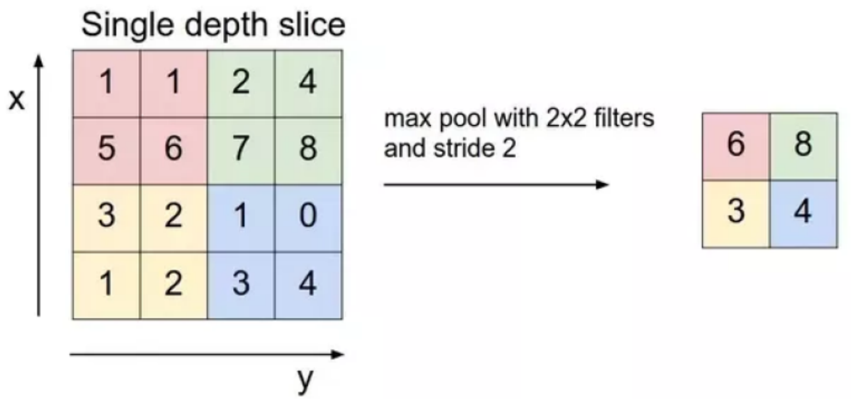


Figure 3.8: Example of a pooling operation with spacial extent and stride equal 2. The pooling function used is the maximum. Image taken from [28]

Non-linear activations are the functions which are commonly found after each convolutional layer. Their purpose is to apply a non-linear transformation to the output of a convolutional layer in order to increase the capacity of the network: in fact, it can be mathematically shown that a series of layers with linear activation functions is equivalent to a single linear layer. Adding non-linear activations allows to increase performance without introducing additional parameters. In fact, common non-linear activation performs only a fixed function, without requiring any learnable parameter. The most popular functions that are applied in a non-linear activations layer are:

- Sigmoid or logistic function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

- Hyperbolic Tangent function (Tanh)

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.8}$$

- Rectified Linear Units (ReLU)

$$f(x) = max(0, x) \tag{3.9}$$

- Leaky ReLU

$$f(x) = \mathbf{1}(x < 0)(\alpha x) + \mathbf{1}(x \geq 0)(x) \tag{3.10}$$

  where α is a small constant.

The first two methods have seen frequent use historically but they have fallen out of favor because they suffers the *vanishing gradient problem* [5]. This means that when the neurons' activation saturates the gradient became almost zero, making the training of the network impossible. ReLU and Leaky ReLU instead do not suffer the vanishing gradient problem, because they never saturate, and accelerate the convergence of stochastic gradient descent as showed in [33].

Fully Connected (FC) layer is the classical layer of a neural network wherein the neurons are connected to all neurons in the previous layer. The activations of an FC layer are computed as the dot product between the input and the weight matrix, plus a bias term (see equation 3.1). It is common to find fully-connected layers used one or more times in the final part of a convolutional architecture, especially for classification and regression tasks.

To connect the FC layer to the previous layers, the input volume with size $w \times h \times d$ must be flattened to obtain a $1 \times 1 \times n$ volume so that the FC neurons can be connected to all input values. The fully-connected layer has only one hyper-parameter to be set: the number of neurons K. This parameter controls the output dimension that is a single vector with dimension $1 \times 1 \times K$. The main function of the FC layer is to summarize the information extracted by the convolutional layers and convert it in a way that can be used to complete the task. For example, suppose to solve an image classification task, the last FC layer will contain as many neurons K as the number of classes present in the dataset. The computed $1 \times 1 \times K$ vector is then fed to a probabilistic function that will perform the final classification.

Batch Normalization (BN) layer [26] is a layer frequently used after each convolutional layer. Being introduced recently, it does not appear in classical architecture but recent works make extensive use of it. It aims to simplify the training of the neural network reducing the effect of the *internal covariate shift*. This phenomenon is due to the fact that, during training, each layer's input changes because of the change of the parameters in the previous layers. This forces each layer of the network to continuously adapt to a new input distribution. Typically, a low learning rate and a careful initialization of the network parameters are adopted to limit the effect of covariate shift. However, it is still hard training the network and having a low learning rate increases the time needed for training.

Batch normalization comes from the observation that network parameters converge faster during training if its input is whitened (linearly transformed to obtain zero means and unit variance) and decorrelated, then it should be also advantageous apply the same operation to the input of each layer. Batch Normalization addresses the problem by adding to the architecture a new layer responsible for the normalization of its input. The normalization is performed during the training of each batch (see Stochastic Gradient-Descent in the previous section) by subtracting from each channel the mean and dividing by the standard deviation computed over the current batch. Given the values of x over a batch $B = \{x_i, ..., x_m\}$, the normalized output $\hat{x}$ is obtained through the following formula:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{3.11}$$

$$\delta_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2, \tag{3.12}$$

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\delta_B^2 + \epsilon}}, \tag{3.13}$$

where ϵ is a small constant that prevents the division by zero.

Unfortunately, this operation may change what the layer is able to represent. Thus, an additional transformation is applied which scales and shifts the normalized value through two trainable parameters: γ and β. The result of the batch normalization y is given by the following formula:

$$y = \gamma\hat{x} + \beta. \tag{3.14}$$

It is worth noting that, neglecting ϵ, setting $\gamma = \sqrt{\delta_B^2}$ and $\beta = \mu_B$ recovers the identity function $y = x$.

Despite the normalization dependent of batches is very efficient during training, it is neither necessary or desirable during inference, since we want to determine the output deterministically, depending solely on the current input. For this reason, the BN layer uses the following formula in the inference to compute the output y:

$$y = \gamma \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} + \beta. \tag{3.15}$$

$E[x]$ and $Var[x]$ use all input data rather than the examples sampled in the batches. Since the means and the variances are fixed during inference, the normalization is a simple linear transformation applied to each activation.

Since this transformation is applied to each depth slice/channel of the input, independently of other slices, given an activation volume with depth size of d, a BN layer requires $2d$ additional parameters, that is γ and β for each depth slice.

3.2.3 Standard architectures

The most common architecture [33] is composed of few blocks made of convolutional and ReLU layers followed by a pooling layer until the activation volume reaches a small size. After these blocks, the activation volume is flattened and there are applied some fully-connected layers, until the last one that computes the output.

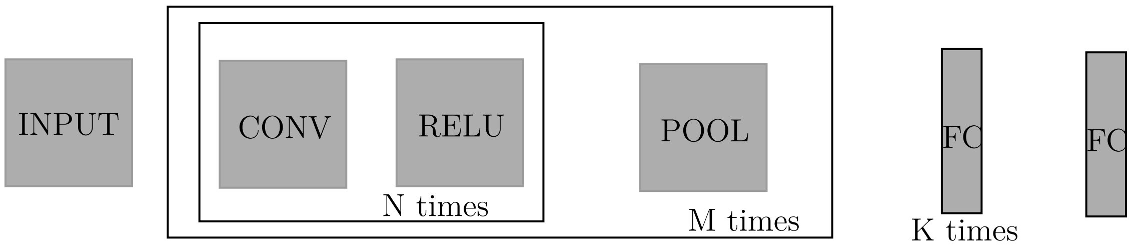


Figure 3.9: Classical pattern for a convolutional neural network.

Sizing the convolutional layer is tricky. There are two strategies to reach large receptive fields: stacking small filters with non-linearities in between or using large filters. Despite being more simple, using a large filter has many disadvantages. The output of the single filter is a linear combination of input while more layers add non-linearity, increasing the expressiveness of the network. Moreover, using large filters introduce more parameters than stacking small filters. The downside of using small filters is that they need more memory to hold the intermediate results while training the model with back-propagation.

In the literature there are many proposed architectures: AlexNet [33] was the first model able to outperform the ImageNet ILSVRC challenge [70]; ZF Net [95] won the ILSVRC 2013, improving AlexNet by tweaking architecture hyper-parameters, especially the size of the middle convolutional layers; GoogLeNet [84] won ILSVRC

2014 proposing an Inception Module that reduced drastically the number of parameters and replacing the fully-connected layers at the top with average pooling layers; another important submission in ILSVRC 2014, VGGNet [75], showed that the depth of the network (the number of layers) is critical to get good performance, and proposed a depth architecture containing 16 layers. The downside of VGGNet is that is very expensive, both in term of memory and time, containing 140 million parameters (AlexNet contained 60 million parameters, GoogLeNet only 4 million). In 2015, the ILSVRC challenge was win by a novel architecture called ResNet[19]. This architecture solves many problems of standard ones, such as vanishing gradient [5], and it is currently the building block in many state of the art models [18], [24], [94]. In the following, this architecture will be briefly described, since it will be the one on which are built the baselines for the challenge.

Residual Networks (ResNet) [19] are convolutional neural networks which were mainly built to address the *vanishing gradient* [5] problem. In very deep networks the gradient is back-propagated through many layers and thus, it can become infinitely small (or even explode) making the training very difficult. As a result, increasing the depth of a neural network does not improve or even decreases performance.

The solution proposed in the ResNet to avoid vanishing gradient is the **residual block**. It is an architectural component that, instead of trying to learn some unreferenced function, tries to learn a residual function starting from an identity mapping. The authors hypothesize that it is easier to optimize a residual mapping than to optimize an unreferenced function. They started from the intuition that, taking a shallow model and stacking identity mapping layers to the end, the resulting network should keep the performances of the shallow model without suffering the degradation problem, because the additional layers do not change the output. Thus, if we add some trainable parameters to the identity mapping layers, we can evaluate a function over the input and then compute the output as the sum of the result of that function and the input itself, that is the residual function. If the identity mapping is optimal, the additional parameters can be driven towards zero and the identity function is recovered. More formally, given an input x, they obtain a function $H(x)$ using parameters θ:

$$H(x) = F(x, \theta) + x. \tag{3.16}$$

To implement it in a neural network, we can fit few stacked layers to approximate the residual function $F(x) = H(x) - x$ with x denoting the input of the first layer. The operation $F(x)+x$ can be implemented with shortcut connections, that are able to connect directly the input with the output, implementing the identity mapping without introducing any computation or additional parameters. The dimension of x and $F(x)$ must be equal in the previous equation. If it is not the case, it can be performed a linear projection W_s by a so called *shortcut connection*, obtaining

$$H(x) = F(x,\theta) + W_s x. \tag{3.17}$$

The union of the few stacked layers and the shortcut connection is called *residual block* and is represented in Figure 3.10. A common choice for the layers is to use a convolutional layer followed by the ReLU operation and another convolutional layer. Moreover, it is common to use batch normalization after each convolutional layer.

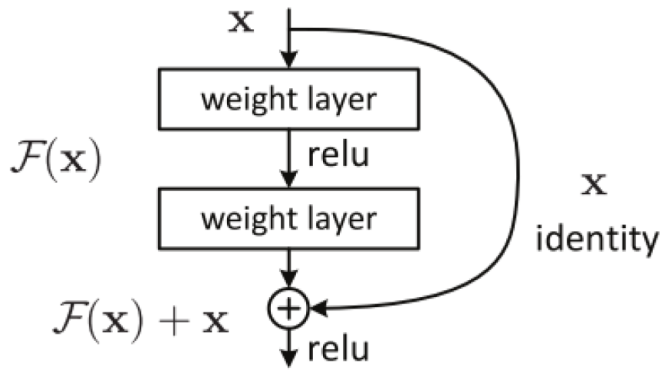

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.10: A residual block. Image taken from [19]

3.3 Multi-task learning

Multi-task learning (MTL) in computer vision looks at models that can accomplish different tasks for a given image (such as image-level labels, semantic segments, object bounding boxes, object contours, occluding boundaries, vanishing points, etc.) while exploring commonalities and sharing computation among them. This is crucial in settings where there are few computation capabilities and limited memory, such as in realistic robotic settings.

Early methods in this area [7] focused on deep neural networks that share weights in the early layers and define specialized ones in the later layers. The network parameters are jointly learned by interleaving samples from each task. As explained in [7], sharing parameters is useful because tends to regularize better the network, improving generalization and performances. This approach requires additional parameters per task and to manually design the network deciding which layers should be shared across tasks. Moreover adding a novel task when the model has been already optimized requires to train again the whole network from scratch by accessing the old training data. This can be a problem, because very often training data of old tasks are unavailable, proprietary or outdated.

An important research area in multi-task learning deals with networks able to learn sequentially new tasks. Works in this area present approaches in which are developed deep neural networks able to learn incrementally and life-long [1], [64]. It is crucial, while learning a new task or domain, to keep the same performances on the already learned tasks. Ideally, every time a new task has to be learned, it should share the parameters with the old tasks, adding as few task-specific parameters as possible, without suffering *catastrophic forgetting* [15], [50] and accessing the old training data. We can identify two strategies to avoid catastrophic forgetting: one is to preserve the knowledge of the old tasks without adding extra parameters, the other is to freeze the weights of the old tasks while learning additional task-specific parameters. Methods that follow the first strategy are for example [37] and [30]. In [37], the authors ensure the preservation of performances on previous tasks by using initial network responses on new data as regularization targets during the new task training. [30] considers updating the network parameters based on the importance for previously seen tasks. While these methods do not increase the network size maintaining the same number of parameters as the original network, performance drops as many tasks are added to the network, limiting the number of learned tasks and suffering catastrophic forgetting. Also, for [37], a large domain shift for a new task causes a significant drop in prior task performance.

To overcome these issues, methods that follow the second strategy keep the previous weights frozen while adding additional per task parameters, leaving untouched the performances on previous tasks and avoiding catastrophic forgetting. The extreme interpretation of this strategy can be found in [72]. It instantiates a parallel neural network for each new task, connected to networks of other tasks through parallel connections. The main drawback of this method is clearly the significant growth of parameters for every task added.

However, recently were introduced methods that follow the second approach capable of learning a new task by introducing only few task-specific parameters. These methods were first proposed in the context of *multiple domain learning* (MDL).

While multi-task learning aims at learning multiple related tasks, multiple domain learning focuses into learning a single network to perform image classification tasks in a diverse set of domains. In other words, it aims to perform only one task (image classification) for many domains (or datasets). The main goal is to learn a single network that can represent compactly all the domains with a minimal number of task-specific parameters. Bilen and Vedaldi [6] proposed to share all core network parameters except normalization layers to model different domains in a single neural network, [46] introduced an approach that uses weight-based pruning to free up redundant parameters across all layers of deep neural network with minimal loss in accuracy, [68] proposed a parameter efficient architecture that enables learning new domains sequentially without forgetting.

Recently, in [62] was introduced the Visual Decathlon Challenge that has motivated researchers to develop methods in this context. The challenge has lead to the proposal of interesting methods that will be analyzed in the following: Piggyback [47], binarized affine fransformation (BAT) [48], series [62] and parallel [63] residual adapters.

Unlike multiple domain learning, this thesis focuses on learning different *tasks* sequentially, while it inherits from that setting the additional goal of requiring a minimal number of task-specific parameters. For this reason, this work evaluates against the proposed challenge models originally developed in the multiple domain learning context and excludes the ones that require a lot of specific parameters per additional task, or that need to access training data of old tasks while learning a new one.

3.3.1 Visual decathlon challenge

The visual decathlon challenge [62] is a new benchmark that aims to evaluate performances of algorithms in the multiple domain learning scenario. Every method is tested on several different domains at the same time and the overall performance is computed to measure its capability on addressing all tasks together. The decathlon challenge proposes to use ten well-known classification datasets from multiple visual domains: FGVC-Aircraft Benchmark [45] contains 10,000 images of aircraft, with 100 images for each of 100 different aircraft model variants. CIFAR100 [32] contains 60,000 colour images for 100 object categories. Daimler Mono Pedestrian Classification Benchmark [52] consists of 50,000 grayscale pedestrian and non-pedestrian images. Describable Texture Dataset [8] is a texture database, consisting of 5640 images, organized according to a list of 47 categories. The German Traffic Sign Recognition Benchmark [79] contains cropped images for 43 common traffic sign categories. Flowers102 [55] is a fine-grained classification task which contains 102 flower categories, each consisting of between 40 and 258 images. ILSVRC12 [70] is the largest dataset in the benchmark and contains 1000 categories and 1.2 million images. Omniglot [35] consists of 1623 different handwritten characters from 50 different alphabets. The Street View House Numbers [54] is a real-world digit recognition dataset with around 70,000 images. UCF101 [76] is an action recognition dataset of realistic human action videos, collected from YouTube. It contains 13,320 videos for 101 action categories converted into images.

Methods are evaluated with a metric that gives a single scalar score, inspired by the decathlon sport discipline. This metric encourages algorithms to perform well in every task rather than be optimal only in few. Given the datasets D_d, $d = 1, ..., 10$ formed of pairs $(x, y) \in D_d$, where x is an image and $y \in \{1, ..., C_d\}$ is a label, divided into training, validation and test sets; given a model Φ that predicts the label given an image, the metric is computed as follow:

$$S = \sum_{d=1}^{10} \alpha_d \, max\{0, E_d^{max} - E_d\}^{\gamma_d}, \tag{3.18}$$

$$E_d = \frac{1}{|D_d^{test}|} \sum_{(x,y) \in D_d^{max}} \mathbf{1}_{\{y \neq \Phi(x,d)\}}, \tag{3.19}$$

where E_d is the average test error for each domain, E_d^{max} the baseline error above which no point is scored, the exponent $\gamma_d \geq 1$ rewards more reductions of the classification error as this becomes close to zero and is set to $\gamma_d = 2$ for all domains. The coefficient α_d is set to $1000(E_d^{max})^{-\gamma_d}$ so that the perfect result receives a score of 1000 for each dataset (10,000 in total).

This work takes inspiration from this setting, defining a new challenge for sequential multi-task learning, considering additionally RGB-D input data. Moreover, the baseline methods that are evaluated in this work had already been evaluated on the visual decathlon challenge. In the following, the baselines methods tested on the proposed challenge will described.

3.3.2 Classical methods: fine-tuning and feature extraction

Fine-tuning is a common method of transfer learning in visual recognition. Often, when a visual task has to be learned, the network is not trained from scratch but it is initialized with weights taken from another network trying to solve a similar task on a large amount of data (e.g. ImageNet [70]). Training on the new task modifies the parameters of the existing model: usually, the classification layer is trained from scratch and initialized with random weights (this is because two different domains have often different set of classes) and a low learning rate is used to tune all the parameters of the network, minimizing the loss function on the new task. Fine-tuning a given architecture achieves good results on the new task at the cost of degraded performance on the old ones because it suffers the catastrophic forgetting problem. For this reason, fine-tuning can be used to learn a new task, but to keep the same performance on the previous is necessary to duplicate the weights and fine-tune the copy. So, we need to have an additional copy of the whole model for each task, resulting in an explosion of parameters. For example, tackling three tasks within the same model requires three copies of the parameters, one for each task.

Feature extraction is another common method. It uses a neural network to extract features from the input image. The features extracted in this way are then fed as input to one or more classification layers. One can think of it as composed by a first network that extracts useful information of an image and a classification layer that takes in input the extracted data and that is specialized for a specific domain or task. As in the previous case, it is often used starting from a network already trained for a specific task (for example, it is common in visual recognition tasks to use a network trained to classify the Imagenet [70] dataset) to whom is replaced the classification layer with a task specific one. The main advantage of this method is that it does not modify the original network and allows the new task to benefit from complex features learned by it. However, if the new task (or domain) is very different from the original one, then the results are poor, far from the performances reached by fine-tuning the model.

3.3.3 Residual adapters

Residual adapters were first proposed in [62] by Rebuffi, Bilen, and Vedaldi. They aimed at finding a deep learning technique able to learn a universal representation of the images, i.e. a neural network able to work well in different domains. The knowledge should be shared between domains allowing to learn a compact multivalent representation. Sharing is the key to obtain a model that reaches better performances in every domain than models tuned only in a specific domain.

Their goal is to develop a parametric family of neural networks $\phi_\alpha : X \to V$ indexed by parameters α, where $X \subset \mathbb{R}^{H \times W \times 3}$ is the space of RGB images and $V \subset \mathbb{R}^{H_v \times W_v \times C_v}$ is the space of feature tensors. ϕ_α can be seen as a parametrized neural network in which the classification layer is removed, therefore it is like a feature extractor. The final parametric neural network is made by the feature extractor ϕ_α in which are defined the parameters α_d and the linear classification layers $V \to Y_d$ for each domain d.

The feature extractor parameters are partitioned in the universal vector w which is fixed and shared among all domains, and a set α of parameters which contains the domain-specific parameters. If α is chosen to contain a lot of parameters, the network needs to learn, for each domain, millions of parameters, and this can slow down the training phase and be computationally unfeasible for a large number of tasks. Furthermore, if α contains only a small subset of the network parameters, then the vast majority of parameters can be shared between different domains. For these reasons, α must be chosen to contain few parameters. The ideal resulting architecture is capable of sharing the vast majority of the parameters and has the ability to learn a new α for a new domain from very few training samples.

The residual adapters were proposed in their series form in [62]. This implementation modifies a standard residual network (see Section 3.2.3) adding domain-specific convolutional layers. Each additional convolutional layer has filters with a kernel size of 1×1 to keep small the number of domain-specific parameters. In this way, they got a network in which there were many shared domain-agnostic layers with fixed parameters and few small convolutional layers with domain-specific parameters. The residual adapter modules were added into the residual block as can be

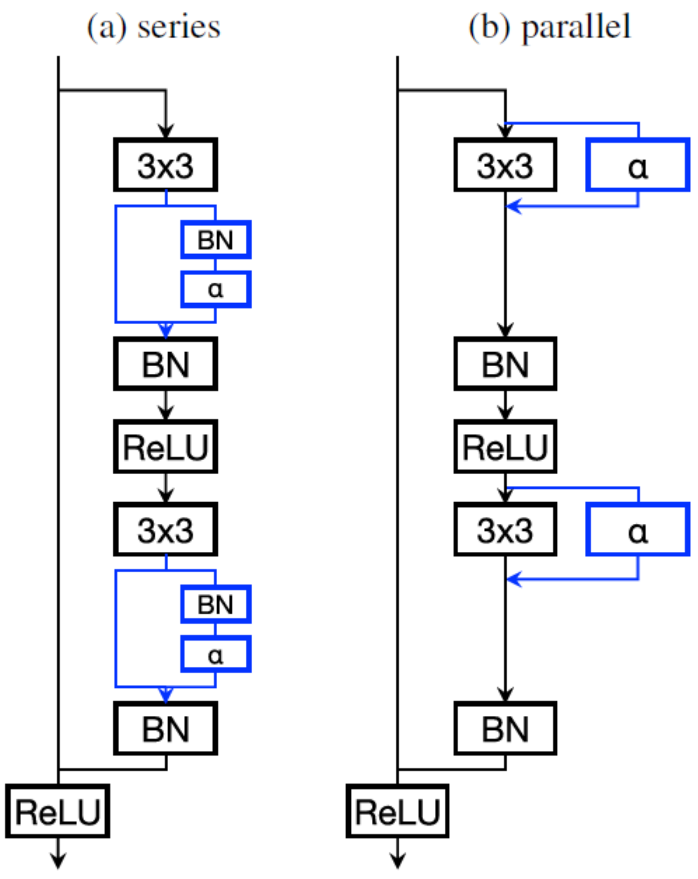


Figure 3.11: The figure, taken from [63] shows the series (a) and parallel (b) residual adapter modules (in blue) embedded in the standard residual module. Both residual adapters contain the batch normalization layers and the residual adapter modules but in different configurations. In (a) there are two series residual adapter modules that are made of a batch normalization layer and a 1×1 convolution, followed by a batch normalization layers. In (b) there are two parallel residual adapter modules followed by a batch normalization layer. It is important to note that the batch normalization layers are domain specific in both (a) and (b) even if they are not blue.

seen in figure 3.11. They implement the following function:

$$g(x;\alpha) = x + \alpha * x. \tag{3.20}$$

It is important to note that in the formula we are ignoring the batch normalization layers effect. These layers are added after each convolutional layer and they are important to normalize the output and facilitate the learning. The normalization operation is followed by a shift and scaling of the output $\gamma x + \beta$, where both γ and β are domain-specific parameters (see Batch normalization in Section 3.2.2). Learning domain-specific batch normalization layers is very effective because it provides more model adaptation and improves performances at the small cost of two parameters per layer.

One of the advantages of using a residual structure for the adapter modules is that the original network output is unchanged if the adapter parameters are zeros. This is extremely important for small domains because this mechanism can prevent overfitting and get better performances.

Moreover, in the adapter modules is impossible to suffer catastrophic forgetting because the shared parameters are fixed and they are never modified while the extra parameters are domain specific, so they are not related to other domains. This is crucial in life-long learning for two reasons: the performance on already learned domains keeps untouched while learning new domains and, furthermore, there is no limit on how many domains a model can learn.

The residual adapters were then proposed in their parallel form in [63]. As the series residual adapters, they are added in a standard residual module, but, as it can be seen in figure 3.11, instead of putting them sequentially to the convolution layers, they are added in parallel to them. They implement the following function:

$$y = f * x + \alpha * x. \tag{3.21}$$

Similarly to the series residual adapters, parallel ones have convolutional layers made of 1×1 filters. So, parallel and series residual modules have a similar number of parameters (the exact same number excluding the additional batch normalization layer in the series residual adapters). The parallel residual adapters also have the nice property of unchanging the output of the network if they are set to zero. As in the series residual adapters, this is useful to avoid over-fitting and increases performance.

The experiment in [63] shows that the parallel residual adapter modules give better results than serials. Anyway, these methods gave similar performances in the experiments and both will be evaluated against the proposed challenge.

3.3.4 Piggyback

Piggyback [47] is a powerful approach that allows solving many tasks adding very few additional parameters to a fixed deep neural network without affecting the performances on any old task.

The intuition behind it relies on the fact that to learn a new task it is not necessary to change the network parameters, but it is enough selectively masking them, or setting certain weights to zero, while keeping the rest the same as before.

Therefore, the key idea of this method is to apply task-specific binary-valued (0,1) masks per task, which will be element-wise multiplied with the network parameters, enabling or disabling them for the specific task, as illustrated in figure 3.12. The base network (called also backbone network) is then shared between multiple tasks and its weights are kept fixed for all tasks. The binary masks are additional parameters that are task-dependent, but, as we will see, they introduce only a little overhead (i.e. 1 bit per network parameter per task).

This approach is very powerful because, even though the backbone network is kept unchanged, it allows to create 2^N different networks, where N is the number of network parameters. For example consider an array of parameter $v = [0.9, 0.37, 0.42, 1]$. Applying masks to v can give rise to many filters such as $v_1 = [0.9, 0.37, 0, 1]$, $v_2 = [0, 0.37, 0.42, 1]$ or $v_3 = [0.9, 0, 0, 1]$.

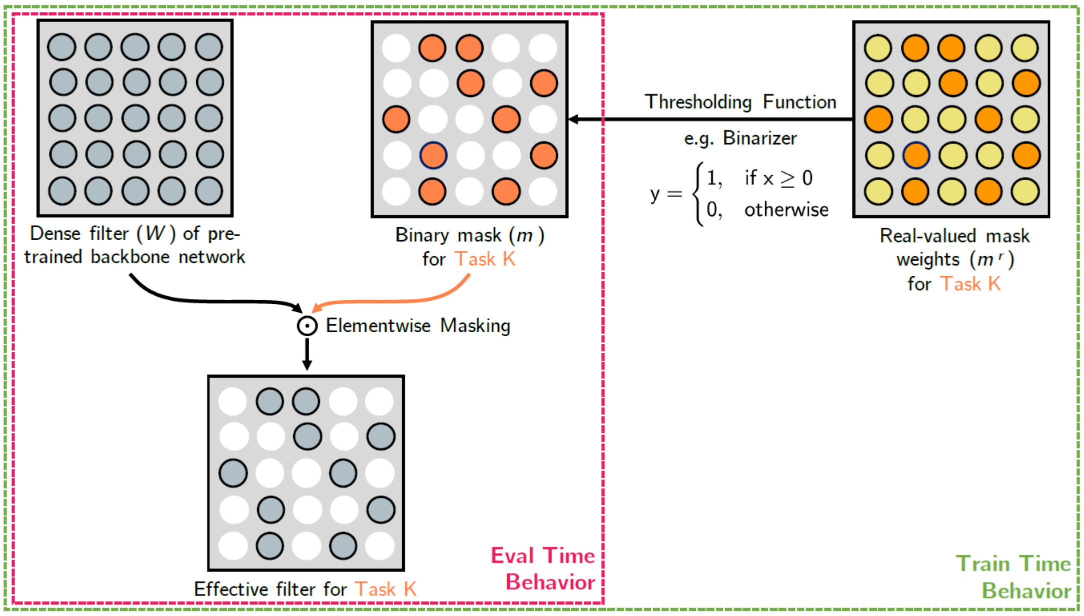

Dense filter (W) of pre-trained backbone network
Binary mask (m) for Task K
Thresholding Function
e.g. Binarizer
$y = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$
Real-valued mask weights (m^r) for Task K
⊙ Elementwise Masking
Effective filter for Task K
Eval Time Behavior
Train Time Behavior

Figure 3.12: The image, taken from [47], is an overview of the piggyback method. The set of real-valued weights mr passes through the thresholding function giving the binary masks m. The binary masks are element-wise multiplied to the parameters W of the base network, which is the same of keeping active or not the individual weights. In the evaluation time, the binary masks are sufficient to evaluate the output, so the real-valued masks can be discarded, obtaining an overhead of one network mask per task.

In practice, the method selects a pre-trained network (for example a ResNet model trained on ImageNet [70]) as the backbone network and associates a real-valued mask to each convolutional and linear layer (except classification layer). Let us consider a fully connected layer (for simplicity, the same description holds for convolutional layers as well) with input vector $x = (x_1, x_2, ...X_m)^T$ of size $m \times 1$, an output vector $y = (y_1, y_2, ...y_n)^T$ of size $n \times 1$ and a weight matrix $W = [w]_{ji}$ of size $n \times m$. The input-output relationship is given by $y = Wx$. The bias term is ignored for ease of notation.

Piggyback associates to the weight matrix W a matrix of real-valued mask weights m^r having the same size as W $(n \times m)$. The binary matrix m is obtained by passing the real-value matrix m^r through a thresholding function:

$$m_{ji} = \begin{cases} 1, & if\, m^r_{ji} \geq 0 \\ 0, & otherwise. \end{cases} \tag{3.22}$$

The binary matrix m enable or disable the weights of W depending on whether the particular value of m_{ji} is, respectively, 1 or 0. The input-output relationship of the masked fully connected layer becomes

$$y = (W \odot m) \cdot x, \tag{3.23}$$

where $\odot$ indicates the element-wise multiplication. The training is done by freezing the weights W while only training the real-valued mask weights m^r. Even though the thresholding function is non-differentiable, the gradients of the binary mask weights can be used as a noisy estimator of the gradients of m^r. Therefore, the real-valued parameters m^r are updated using the gradient computed for m, following the straight-through estimator approach [4], [23].

After the training, the real-valued masks are no longer required and can be discarded. The binary masks instead are stored and will be used in the evaluation time to mask the weights of the model for the task for which they were trained. The binary mask requires only 1-bit per parameter and these bits are the only extra parameters to store for each task. Thus, the overhead per task is approximately 3.12% of the backbone network size. This is extremely effective to represent compactly multiple domains or tasks.

There are two important aspects to note of this method. First, this method requires attention in how the real-valued masks are initialized and optimized because the mask gradients would have different magnitude at different layers. The authors propose two approaches. The first is to initialize m^r with values proportional to the weight matrix W of the corresponding layer and use a constant learning rate for all layers. The latter is to initialize m^r with a constant value while using an adaptive optimizer such as the Adam [29] algorithm. The latter approach produces the best result and will be used also in this work.

The second aspect concerns the initialization of the backbone network. A study performed in [47] shows that a good initialization is critical to obtain good performances. The authors tried different settings and it is shown that the pre-trained model that works best is the one trained on ImageNet [70]. This work follows their suggestion and it uses a model pre-trained on Imagenet.

3.3.5 Binarized affine fransformation (BAT)

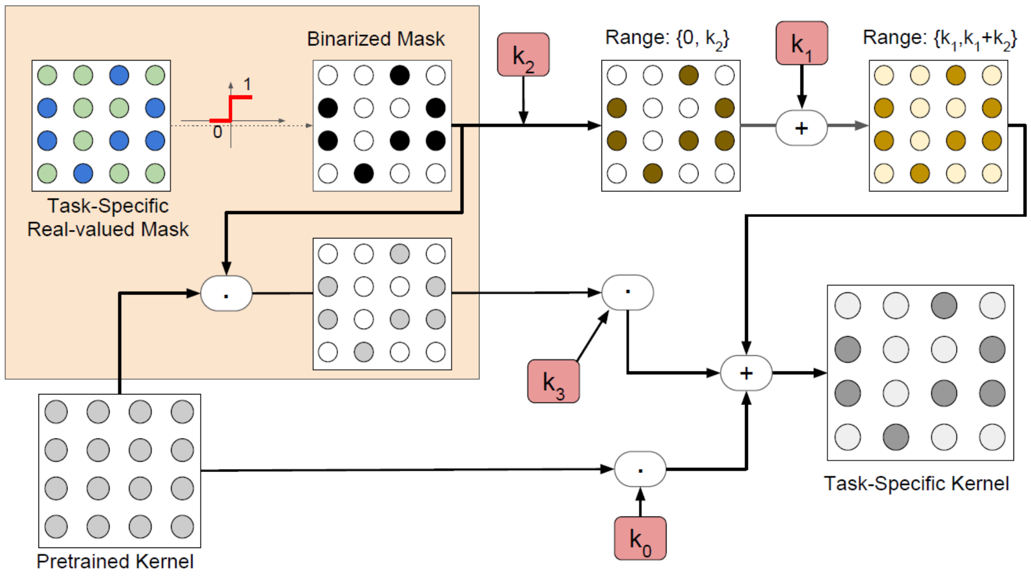


Figure 3.13: The image, taken from [48], is an overview of the binarized affine fransformation (BAT) method. Similarly to Piggyback, each convolutional filter is masked by a task-specific binary-valued mask that is obtained passing a real-valued mask through a thresholding function (orange part). However, while Piggyback uses only the binary masks as task-specific parameters, BAT generalizes the approach applying an additional affine transformation directly to the binary masks, which scales (through the parameter k_2) and shift (through the parameter k_1) the mask values. Moreover, both the original filter and the masked filter are multiplied by, respectively, the scale factors k_3 and k_0. Finally, the different masks obtained are summed to produce the final task-specific kernel.

This method, proposed in [48], exploits the same key idea of Piggyback (see Section 3.3.4). Thus it aims at learning incrementally new tasks without changing the weights of the neural network but combining its parameters, obtaining a new model at the cost of adding a few task-specific parameters.

This work can be seen as a generalization of Piggyback, in fact, instead of pursuing the construction of task-specific kernels by simply multiplying element-wise the convolutional filters and the binary masks, it is implemented a more elaborated parametric affine transformation that mixes the model parameters with task-specific binary masks, as it is represented in figure 3.13.

This method starts from a pre-trained network (for example, a Resnet model trained on Imagenet [70]), and associates a real-valued mask to each convolutional and linear layer (except classification layer). Consider a fully connected layer (for simplicity, the same description holds for convolutional layers as well) with input vector $x = (x_1, x_2, ...X_m)^T$ of size $m \times 1$, an output vector $y = (y_1, y_2, ...y_n)^T$ of size $n \times 1$ and a weight matrix $W = [w]_{ji}$ of size $n \times m$. The input-output relationship is given by $y = Wx$. The bias term is ignored for ease of notation.

BAT associates to the weight matrix W a matrix of real-valued mask weights m^r having the same size as W $(n \times m)$. The binary matrix m is obtained by passing the real-value matrix m^r through a thresholding function

$$m_{ji} = \begin{cases} 1, & if\, m^r_{ji} \geq 0 \\ 0, & otherwise. \end{cases} \tag{3.24}$$

Once the binary matrix has been computed, we can proceed to combine it with the weight matrix W. The general input-output relationship of the masked fully connected layer becomes

$$y = \widetilde{W} \cdot x, \tag{3.25}$$

$$\widetilde{W} = k_0 W + k_1 \mathbf{1} + k_2 M + k_3 W \odot M, \tag{3.26}$$

where $\odot$ indicates the element-wise multiplication, k_i are additional task-specific parameters that are learned along with the binary mask m and $\mathbf{1}$ is a matrix of $n \times m$ ones. If $k_{1,2,3}$ are set to zero and k_3 is set to one, this method behaves equally to Piggyback.

The authors proposed also a second and simpler version of BAT. They found that the term correspondent to Piggyback's multiplicative transformation does not help to achieve higher performances. For this reason, they got rid of that component putting $k_3 = 0$ and obtained the simple version, that implements the following input-output relationship.

$$y = \widetilde{W} \cdot x, \tag{3.27}$$

$$\widetilde{W} = k_0 W + k_1 \mathbf{1} + k_2 M, \tag{3.28}$$

In addition to learning the binary masks and the parameters k_i, the authors suggest learning also task-specific batch normalization parameters. In the cases where the batch normalization layer follows the convolutional one, the parameter k_0 can be fixed to 1, because the output of the batch normalization is invariant to the scale of the convolutional weights.

While the training of k_i parameters is straightforward, the optimization strategy of the binary mask is the same followed by Piggyback (see Section 3.3.4). Moreover, the authors initialized the m^r values with a constant value and adopted an adaptive learning strategy, specifically Adam [29].

BAT is a very effective strategy. It is able to obtain high performance by leveraging the high degree of freedom in perturbing the base neural network while keeping low the per-task overhead in terms of additional parameters. In fact, it adds slightly more than 1 bit per parameter per task, that is nearly the 3% of the network parameters.

This work is comparable, both in terms of performances and number of parameters, with Piggyback, so, in the proposed triathlon challenge both methods will be analyzed.

Chapter 4

The RGB-D triathlon challenge

This thesis proposes the RGB-D triathlon challenge to motivate researchers in developing models able to perform different tasks and at the same time using a neural network architecture that minimizes the number of additional parameters required.

The challenge requires to perform three fundamental tasks: object classification, pose estimation and scene recognition. The datasets chosen offer both RGB and depth images because the challenge wants to let researchers to exploit the data they prefer, so, it provides three different settings: use only RGB data, use only depth information, use both RGB and depth. Each setting will be evaluated separately allowing researchers to focus on what they prefer.

The first Section of this chapter provides the details of the tasks to be performed, in particular, the dataset chosen and how to compute the accuracy for the task. The second Section discusses five different evaluation metrics that combine the accuracy of the tasks to establish the score on the challenge.

4.1 Task and datasets

4.1.1 Object classification

Object classification is a common task that consists in assigning to an input image containing an object a semantic class label. For this task it is chosen the RGB-D Object Dataset [34]. It is a large dataset containing 300 common household objects organized in 51 categories. The dataset was recorded using a Kinect style 3D camera that records 30 synchronized and aligned 640×480 RGB and depth images per second. Each object was placed on a turntable during the recording and it was

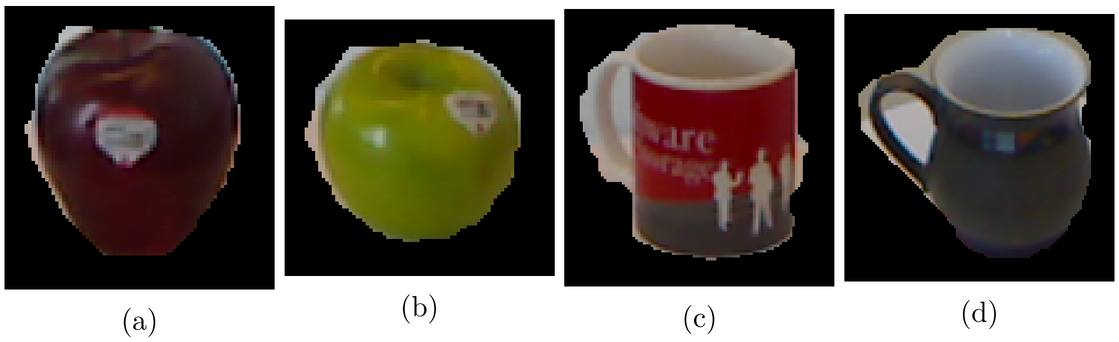


Figure 4.1: Some examples of dataset images. There are shown two apples, the first (a) is in the training set and the second (b) belongs to the test set. Images (c) and (d) represent two coffee mug and, as before, the left one is in the training set, the right one belongs to the test set. It can be noted that they have different dimensions and they are masked in order to exclude the background.

captured the whole rotation. For each object, there are three video sequences, each one with the camera mounted at a different height so that the object is seen from different angles, approximately 30, 45, and 60 degrees relative to the horizon. In this dataset, objects are organized into both categories and instances. For example the class *apple* is divided into physically unique instances like a red apple and a yellow apple (see figure 4.1).

This work will use the evaluation version of the dataset, the same used by its authors in their work [34]. It contains a subset of the original dataset, picking only one image every fifth video frame. The images are also cropped to tightly include the object in the frame and masked to exclude the background. Thus, the evaluation dataset contains nearly 45000 RGB-D images with different width and height divided into 51 categories. As in [34], the model is trained only on a subset of the available objects (physical instances). The test dataset contains one instance for each category that was left out from the training dataset. At test time, the system is queried with an image that contains an object that was not present in training data and the task is to assign a category label to that image. For ease of comparison, the instances are not put into the test dataset randomly for every trial as done in [34] but it is used a fixed set of instances, that is the first split proposed by the authors in their trials.

The accuracy of the model represents the number of correct label assigned to the examples of the test dataset. It can be computed using the following formula:

$$a_{OC} = \frac{1}{|D_{TEST}|} \sum_{(x_i,y_i)\in D_{TEST}} \mathbf{1}(\phi(x_i) = y_i), \tag{4.1}$$

where D_{TEST} is the test dataset, ϕ is the neural network function and $\phi(x)$ the predicted class.

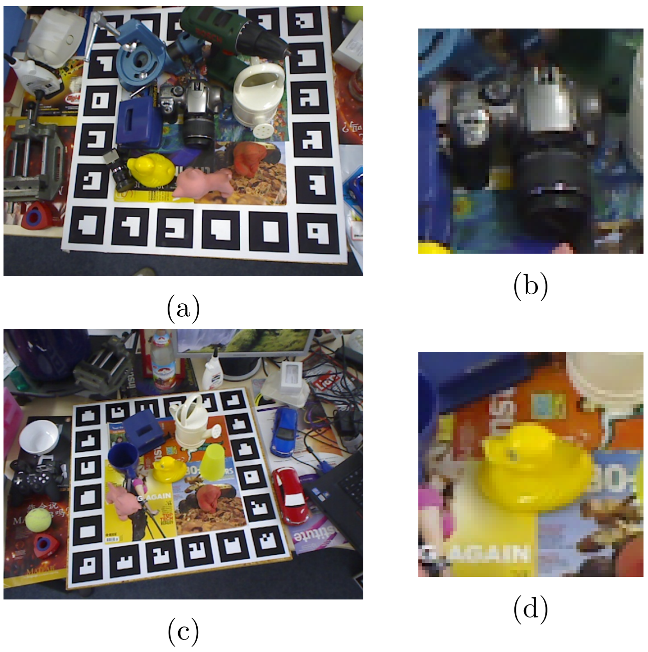


Figure 4.2: Example of LineMOD dataset images. On the left, images (a) and (c) are taken from the original set and it is visible the marker-board. Images (b) and (d) are to the cropped version of, respectively, (a) and (c). It can be noted that they have the same pose but they do not have the surrounding marker-board.

4.1.2 Pose estimation

Pose estimation consists in determining the pose, i.e. the orientation, of an object in the image (see Section 3.1.2). For this task, it was chosen the LineMOD dataset [22]. It contains 18000 RGB-D images with 15 different objects classes. The original version of the dataset contained the objects centered in a marker-board. Making some experiments we noted that the neural network was able to predict the pose without considering the object but looking only to the marker board. Thus, we decided to pick a cropped version of the dataset that was proposed in [90] and can be found here: www.tugraz.at/institute/icg/research/team-lepetit/research-projects/object-detection-and-3d-pose-estimation. In the cropped version all the images are squared with size 64×64 pixels and they contain the objects centered in the scene.

The dataset was divided into two parts: the training set and the test set. The full dataset was split by picking one image every five and putting it into the test dataset. The other four images belong to the training set. The ground truth of the pose is given in the form of 3×3 rotation matrix that maps the camera world coordinate into camera coordinates.

In order to explain the accuracy metric over this task, we need to define how to represent the rotation. Taking inspiration from [44] this work uses the unit quaternion representation. It provides some advantages: it is a compact representation, using only 4 numbers instead of the 9 parameters of the rotation matrix, it is numerically stable and avoids the *gimbal lock* phenomenon.

Quaternions are a number system that extends the complex numbers. This work will focus on unit quaternions that can be expressed as $q = s+v_1i+v_2j+v_3k = (s, v)$ where i, j, k are fundamental quaternion units, and that have unit norm $|q|_2 = 1$. According to Euler's rotation theorem, any rotation about a fixed point can be expressed as a single rotation of angle θ around a fixed axis v, called Euler axis, that runs through the fixed point. The Euler axis can be represented as a three-dimensional vector with unit norm: $v = v_1i + v_2j + v_3k, |v|_2 = 1$. Therefore, any rotation in three-dimensional space can be described through a combination of a vector v and an angle θ, providing the axis-angle representation θv. The axis-angle representation can be represented by a quaternion using the extension of Euler's formula:

$$q = e^{\frac{\theta}{2}(v_1i+v_2j+v_3k)} = cos\frac{\theta}{2} + sin\frac{\theta}{2}(v_1i + v_2j + v_3k) \tag{4.2}$$

Quaternions can be also derived directly from the rotation matrix but it requires particular care when the trace of the matrix is zero or very small.

Before defining the distance between quaternions is important to define some basic algebraic operations. Let $p = p_1 + p_2i + p_3j + p_4k$ and $q = q_1 + q_2i + q_3j + q_4k$ be unit quaternions representing two rotations in the same basis.

- *Addition* $p + q = (p_1 + q_1) + (p_2 + q_2)i + (p_3 + q_3)j + (p_4 + q_4)k$
- *Multiplication by a scalar* $\lambda q = \lambda q_1 + \lambda q_2i + \lambda q_3j + \lambda q_4k$
- *Hamilton product* $pq =$
  $(p_1q_1 - p_2q_2 - p_3q_3 - p_4q_4) +$
  $(p_1q_2 + p_2q_1 + p_3q_4 - p_4q_3)i +$
  $(p_1q_3 - p_2q_4 + p_3q_1 + p_4q_2)j +$
  $(p_1q_4 + p_2q_3 - p_3q_2 + p_4q_1)k$
- *Conjugation* $q^* = q1 - q_2i - q_3j - q_4k.$

Performing a rotation denoted by the multiplication between two rotation quaternions pq is equivalent to perform the rotation represented by q and next, the rotation represented by p. The conjugate quaternion q^* represents the same rotation of q but performed on the opposite axis. In fact, recalling the extension of Euler's formula (Equation 4.2) it can be seen that the conjugation operation is equivalent to deriving the quaternion from the axis-angle representation $-\theta v$.

Thus, the difference rotation quaternion that represent the difference rotation is defined as $r \doteq pq^*$. The distance between rotations represented by p and q is the angle of this difference rotation quaternion r:

$$cos(\frac{\theta}{2}) = p_1q_1 + p_2q_2 + p_3q_3 + p_4q_4, \tag{4.3}$$

$$\theta = 2\arccos(|p_1q_1 + p_2q_2 + p_3q_3 + p_4q_4|). \tag{4.4}$$

The equation 4.4 define the geodesic distance between rotation represented by quaternions. This is the optimal way to represent the distance in this non-Euclidean space. Other distances, such as the L2 distance (or Euclidean distance), do not represent well the distance in this space and should be avoided.

The metric used for this challenge to define accuracy considers both the classification task and the pose estimation task. The first is computed by comparing the predicted label with the ground-truth class, while the latter is evaluated by computing the geodesic distance between the predicted quaternion and the ground-truth rotation represented by a quaternion. In particular, the accuracy is computed using the following function:

$$a_{PE} = \frac{1}{|D_{TEST}|} \sum_{(x_i,y_i,r_i)\in D_{TEST}} \mathbf{1}(\phi(x_i) = y_i,\ \theta(\psi(x_i), r_i) < 20), \tag{4.5}$$

where $\phi(x_i)$ is the label predicted by the model, y_i is the ground truth label, $\psi(x_i)$ is the quaternion predicted by the model, r_i is the ground truth rotation, and $\theta(\cdot)$ is a function that computes the geodesic distance between quaternions expressed in degrees.

4.1.3 Scene recognition

Scene recognition consists into assigning to an image a class label that corresponds to the place where the picture was taken. For this task it is proposed the NYU-Depth V2 dataset [53]. The dataset contains 1449 pair of synchronized RGB and depth images, gathered from a wide range of commercial and residential buildings in three different US cities, comprising 464 different indoor scenes across 27 scene classes. Anyway, most of these scene classes are not well represented, thus, following the approach used in [16] the 27 categories were divided into 10: 9 most represented categories and the rest. As in the other tasks, the dataset was split in training data and test data. The images are divided according to the split proposed in [80], where is given a mapping between the image numbers and the dataset to which they belong.

The accuracy of the model represents the number of correct labels assigned to the examples of the test dataset. It can be computed using the following formula:

$$a_{SR} = \frac{1}{|D_{TEST}|} \sum_{(x_i,y_i)\in D_{TEST}} \mathbf{1}(\phi(x_i) = y_i), \tag{4.6}$$

where D_{TEST} is the test dataset, and $\phi(x)$ the predicted class.

Figure 4.3: Some examples of NYU-Depth V2 dataset images. Images (a) and (b) represent two different classrooms, while images (c) and (d) represent two living-rooms.

4.2 Metrics for evaluation

Defining the accuracy on each dataset is not enough, being a multi-task challenge, the accuracies of the three tasks needs to be joined through single metric that take them into account.

In particular, the metric must consider two factors: the accuracy achieved in each task and the number of additional parameters required. For example, fine-tuning achieves a very good result but at the cost of using a lot of additional parameters, thus, it should be assigned a low score. The metric must assign a high score when the accuracy is good for all tasks and the additional parameters are few.

For the challenge, different metrics have been evaluated for computing the score. Each one has its own advantages and disadvantages, so each of them will be briefly discussed, concluding with the metric chosen for the proposed challenge.

In the discussion it will be used this notation:

- a_i^m is the accuracy of the model m for the task i.
- a_i^e is the accuracy of the feature extractor for the task i.
- a_i^f is the accuracy of the fine-tuning for the task i.
- p_i^m is the size of the additional parameters (in bit) added by the model m to perform task i.

- p_0 is the size of the base parameters of the model (in bit); these parameters are the ones shared between all tasks.

4.2.1 Average accuracy

This score is the simplest. It computes the average of the accuracy over the three tasks. It does not consider the additional parameters.

$$S = \frac{1}{N}\sum_{i=1}^{N} a_i^m \tag{4.7}$$

It presents many disadvantages. First of all, it does not consider the number of parameters, ignoring one of the judging criterion. Being so trivial it also ignores the fact that a task can be more difficult than others.

4.2.2 Average accuracy - parameters ratio

This score is an extension of the previous that considers also the parameters. It computes the average accuracy over the tasks and divides it by the number of parameters. To prevent a very tiny score, the parameter ratio is normalized by adding p_0 as follows:

$$S = \frac{1}{N}\sum_{i=1}^{N} \frac{p_0}{p_0 + p_i^m} \cdot a_i^m. \tag{4.8}$$

Even if it considers the number of parameters, it presents the same disadvantages of the previous score, being ineffective in judging tasks with different difficulties.

4.2.3 Decathlon score

This metric takes inspiration from the score used in the Visual Decathlon Challenge [62] (see Section 3.3.1). This score does not consider the number of parameters but tries to encourage the achievement of a good score on all tasks instead of focusing only on some.

To define the metric they do not consider the accuracy but the error E_i. These are obviously bound by the relation $E_i = 100 - a_i$. Given $\Phi_i(x)$ the function of the neural network for the task i, D_i^{test} the dataset to be evaluated for the task i, the error E_i of the dataset can be computed as follow:

$$E_i = \frac{1}{|D_i^{test}|} \sum_{(x,y)\in D_i^{test}} 1(y \neq \Phi_i(x)). \tag{4.9}$$

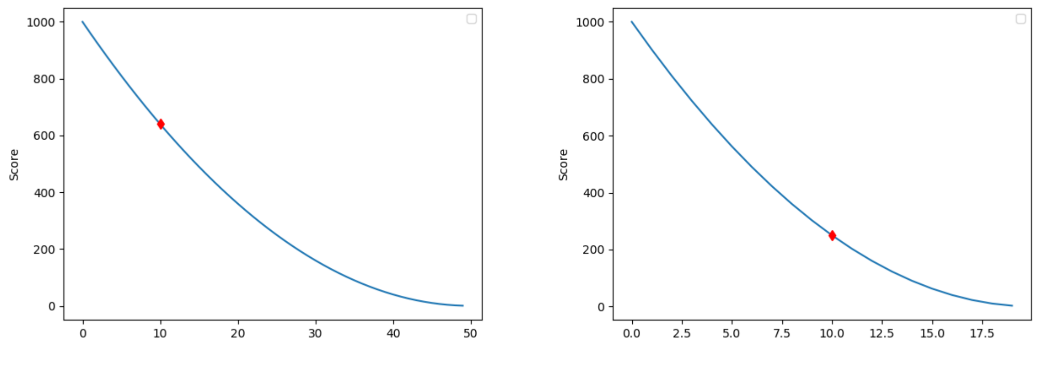
They define a maximal error that can be committed for each task E_i^{max}. If E_i is bigger than E_i^{max} the score assigned to the task i is zero. The overall score is computed as follows.

$$S = \sum_{d=1}^{N} \alpha_i \; max(0, E_i^{max} - E_i)^{\gamma_i}, \tag{4.10}$$

$$\alpha_i = 1000(E_i^{max})^{-\gamma_i} \tag{4.11}$$

It encourages getting a good score on all tasks by raising the difference between errors by a coefficient γ_i. In the Visual Decathlon Challenge [62] it is suggested to set $\gamma_i = 2$ for all tasks. It can be noted that, if a model focuses only on some tasks, it will get a high score for them, but a very low score for the others, obtaining a low final score. The score of each dataset is normalized by the parameter α_i that force it to be in the range [0,1000].

Anyway there are two drawbacks of using this score. First, it does not consider the number of parameters, a judging criterion. Second, it is hard to set the E_i^{max} for each dataset. Its value strongly impacts the score for each task and should be carefully tuned. In [62] the authors propose to set $E_i^{max} = 2E_i^f$, where E_i^f is the error of the fine-tuning method.



(a) (b)

Figure 4.4: The plot in figure (a) reports the Decathlon score for $E_i^{max} = 50$. The plot figure (b) reports the same score with $E_i^{max} = 20$. The red dot indicates the result that a model get making an error of 10 % in both (a) and (b). It can be noted the score gets by a model making an error of 10% changes a lot by changing the parameter E_i^{max}.

4.2.4 Revised decathlon score

This new metric is built upon the decathlon score. It considers the number of additional parameters used while keeping the advantages of the previous metric. First of all, it takes into account the accuracy and not the error as done before. Thus, it is introduced a parameter A_i^{min} that represents the minimal accuracy that a model must get to have a score higher than zero for task i, and a parameter A_i^{max} that represent the accuracy threshold that guarantees the highest score for task i. Then, it is introduced another parameter: the minimal number of parameters required by the model P_i for the task i.

The parametric function to compute the score is the following:

$$S = \sum_{i=1}^{N} \alpha_i \cdot \lambda^{\frac{P_i}{p_i^m} - 1} \cdot max(0, a_i - A_i^{min})^{\gamma_i}, \tag{4.12}$$

$$\alpha_i = 1000(A_i^{max} - A_i^{min})^{-\gamma_i} \tag{4.13}$$

As it can be noted, the function above contains the hyper-parameter λ that did not appear in the decathlon score. λ affects the impact of the number of additional parameters. If it is big, then the ratio $\frac{P}{p_i^m}$ will be very important for the final score. Otherwise, if it is nearly one, the ratio has only a little effect. γ_i and α_i have the same meaning as in the decathlon challenge, and as it was done there, γ_i is set to 2 for all the tasks.

This metric considers both the parameters and the accuracy, motivating the models to achieve good results in all datasets instead of focusing only on some. Anyway, it requires a lot of parameters to be set: A_i^{min}, A_i^{max}, P_i, and λ. We evaluated some different settings and we propose the following values.

- $A_i^{min} = 100 - 2(100 - a_i^f)$. It is the minimal accuracy that corresponds to an error equals to the double of the fine-tuning error, as described in the previous score but expressed in term of accuracy.

- $A_i^{max} = 100$. As adopted in the decathlon score.

- $P_i^{max} = p_0$. We decided to set the value to the number of parameters shared by the network between tasks.

- $\lambda = 10$. We decided to strongly penalize the use of additional parameters. A comparison with other values is reported in figure 4.5

4.2.5 Linear score

This work proposes an additional metric that considers both the number of parameters and the accuracy. It is computed through the following formula:

$$S = \frac{1}{N}\sum_{i=1}^{N} 2 \cdot (\frac{p_0}{p_0 + p_i^m} - \frac{1}{2}) \cdot \frac{max(0, a_i^m - A_i^{min})}{A_i^{max} - A_i^{min}} \tag{4.14}$$

where A_i^{max} and A_i^{min} are arbitrary values.

This metric penalizes the addition of extra parameters, in fact, it decreases the score when the additional parameter size increases. The maximal size of the additional parameters to get a score is two times the size of shared parameters. For example, the fine-tuning, that copies the original network introducing a number of parameters equals to the original network, will have a score equals zero.

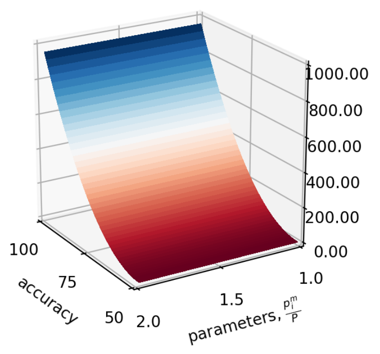
The accuracy is bound linearly to the score. When the accuracy reaches A_i^{max}, the model gets an optimal score for that task: 1000. the maximal score is kept unbounded, giving a bonus to the methods that perform better than fine-tuning. On the other hand, if the accuracy is less than or equal to A_i^{min}, the model will score zero.

As in the previous metrics, the parameters need to set carefully. We propose to set A_i^{max} equals to the fine-tuning accuracy to best evaluate the method. This is very important for a lot of methods, such as piggyback (see Section 3.3.4), where the initialization of the backbone network is crucial for the performances of the method itself. We want to avoid that a simple tuning of the backbone network weights increases the score, otherwise methods will be not comparable. This requires that each submission to the challenge must compute the fine-tuning accuracy of its backbone network. For the same reason, we set the A_i^{min} equals to the feature extractor accuracy. The Equation 4.14 thus becomes:

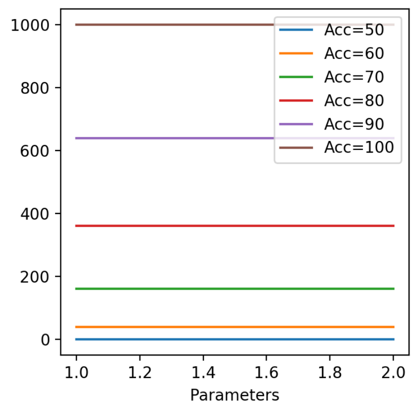
$$S = \frac{1}{N}\sum_{i=1}^{N} 2 \cdot (\frac{p_0}{p_0 + p_i^m} - \frac{1}{2}) \cdot \frac{max(0, a_i^m - a_i^e)}{a_i^f - a_i^e} \tag{4.15}$$

where a_i^e is the accuracy of the feature extractor and a_i^f the accuracy of the fine-tuning for the i-th task.

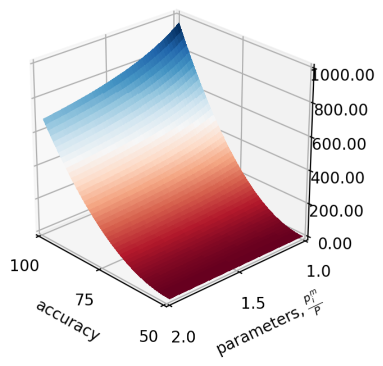
This is the metric that is applied to the RGB-D Triathlon challenge. It is more adaptive to various models and it is the best way to evaluate different methods with different baseline networks as it will be shown in Section 5.2.



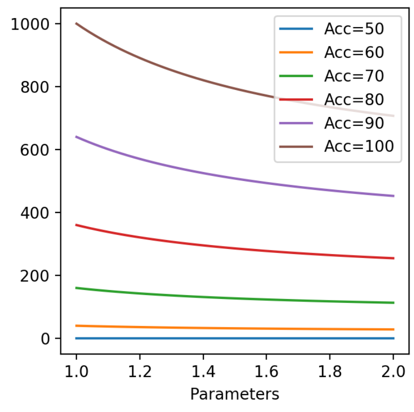
(a) $\lambda = 1$



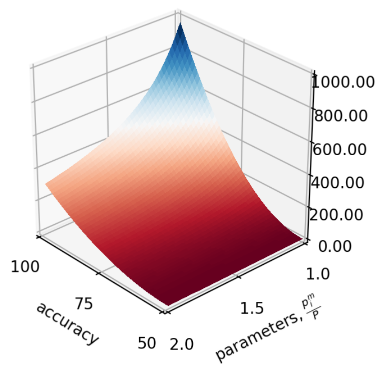
(b) $\lambda = 1$



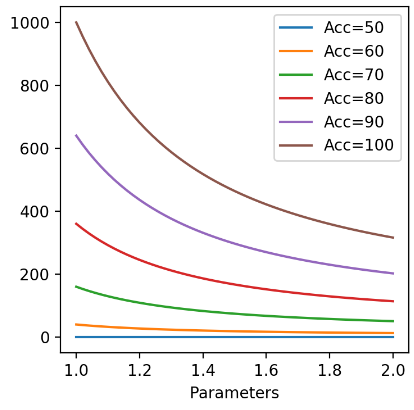
(c) $\lambda = 2$



(d) $\lambda = 2$

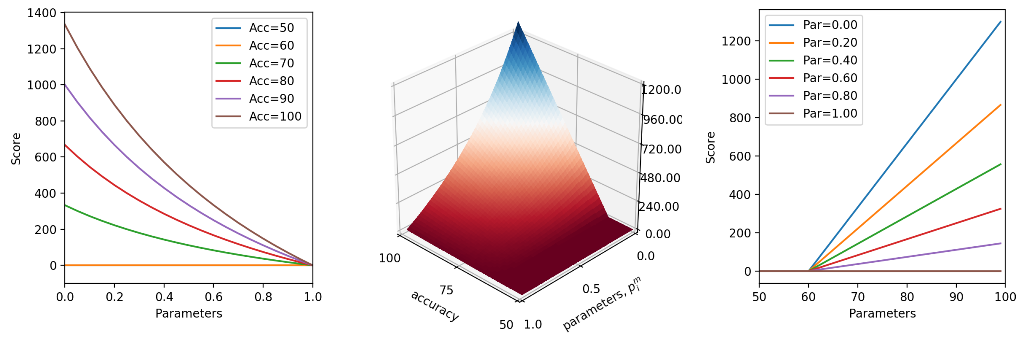


(e) $\lambda = 10$



(f) $\lambda = 10$

Figure 4.5: Effect of λ in computing the revised decathlon score with $A_i^{min} = 50$.



(a) (b) (c)

Figure 4.6: Plot of the Linear score with $a_i^e = 60$, $a_i^f = 90$. The parameter scale reports the ratio $\frac{p_i^m}{p_0}$

Chapter 5

Experiments and Results

This chapter discusses the implementation and the performances against the challenge of these methods: *fine-tuning*, *feature extraction*, *series residual adapter*, *parallel residual adapter*, *Piggyback*, and *binarized affine fransformation (BAT)*. The first section describes the implementation of the methods, the data pre-processing steps and the hyper-parameters used for the training. The second section discusses the results on the challenge, providing the results using different score metrics, showing the advantages and disadvantages of each one.

5.1 Implementation

The various methods evaluated against the challenge in this work require a backbone network. We decided to use the *ResNet-18* pre-trained on ImageNet [70] dataset. It is a cheap standard residual architecture which consists of four blocks of two residual units. Each residual unit contains a convolutional layer with 3×3 kernel size, batch normalization and ReLU (see Section 3.2.3). The network accepts 224×224 images as input, downscale the spatial dimension by two at each block and ends with a classification layer. The number of filters is set to 64, 128, 256, 512 for these blocks, respectively. This network because is the best trade-off between accuracy and speed. Moreover, being a very common choice, many frameworks provide their own implementation and even some pre-trained models, and this can help other researchers to replicate the results reported in this work or to build over them.

The *fine-tuning* is the simplest method to be implemented. It requires to instantiate a new ResNet-18 model pre-trained on the ImageNet dataset for each task and each of them is fed with the images of the respective dataset. Each network is independent of the others, both in training and evaluation phases, de-facto producing three different models, one for each task. The *feature extraction* instead requires only one backbone neural network but three different classification layers. The weights of the network are frozen, i.e. they are not optimized, besides the task-specific classification layer, for which parameters are updated through stochastic gradient descent with momentum. The *series residual adapter* implementation follows the method given in [62]. A series residual adapter module is added after the convolutional layer of each residual unit. It is made by a convolutional layer with kernel size 1×1 and a batch normalization layer. While the weights of the backbone networks are taken from the pre-trained model on ImageNet dataset, the series adapter modules' convolutional layers are initialized randomly and follow the normal distribution $\mathcal{N}(0, 0.0001)$ and the batch normalization layers are initialized to have scale equals 1 and bias equals zero. Each task requires a few additional parameters that are learned independently from the other tasks with stochastic gradient descent with momentum. All the batch normalization layers are considered task-specific. The *parallel residual adapter* method is similar to the previous one and follows the method given in [63]. Instead of adding serial residual adapter modules, it adds task-specific residual adapter modules parallel to the convolutional layers of each residual blocks. Each parallel residual adapter module consists in a small convolutional filter with kernel size 1×1 whose parameters are initialized randomly and follows the distribution $\mathcal{N}(0, 0.0001)$. The other parameters are taken from the pre-trained model. Also for this method, the batch normalization layers are considered task-specific. *Piggyback* is implemented as showed in [47]. The binary masks are added to each convolutional layer and are task-specific. The real-valued masks parameters are initialized to 0.01 and the threshold value is zero. The other parameters are taken from the pre-trained model and the batch normalization layers are considered task-specific. *Binarized affine fransformation (BAT)* is implemented as showed in [48]. As in the previous method, the binary masks are added to each convolutional layer and are task-specific. The real-valued masks parameters are initialized randomly and follow a uniform distribution $\mathcal{U}(0.00001, 0.00002)$. The parameters k_1, k_2, k_3 are task-specific and are optimized while training for the specific task. They are all initialized to zero. The parameter k_0 is set to 1 and it is not learnable.

To implement these methods it was used the PyTorch framework that offers an open-source implementation of the ResNet-18 and a pre-trained model on the ImageNet dataset. The code of the networks and the training procedure can be found here: github.com/fcdl94/RobotChallenge.

All the methods have been evaluated on the three data settings: using only RGB images (called RGB), using only Depth images (D), using both RGB and depth images (RGB+D). While the RGB setting is trivial to use because the ResNet architecture is designed to process RGB images, the other two settings require to be adapted. First of all, the backbone neural network accepts RGB images that have three color channels but a depth image has only one channel. To adapt the D setting it was used a naive approach. Instead of using the depth as single channel image, the depth channel was copied into other two, obtaining a three channels depth image where all the channels have the same values. In this way, the standard ResNet can be fed with a 3-channel depth image and produce an output. After some tests, we decided to process the depth images using the Resnet-18 pre-trained over ImageNet as the backbone network, as done in the RGB setting. The RGB+D settings present a similar problem. It should process 4-channels images: 3 channels from the RGB image and one channel from the depth image. Instead of trying to combine these channels, the naive approach was preferred again, using the depth images as three-channels images. The RGB and depth images separately were processed and then to combine with a classification layer the features extracted from the two different sub-networks. Intuitively, the resulting neural network elaborates both images at the same time extracting different features from each one, and then combines these features with a fully-connected layer that is able to select which are the most relevant and to give a prediction based on that. In this way, RGB+D setting can be adapted the without changing the backbone network. It needs only to instantiate two different networks of the selected method and to combine the features with an additional fully-connected layer. It is important to note that we call features the activation volume that is produced by the last layer before the classification one.

To provide comparable results between the baselines the same set of hyper-parameters was used for all the methods. The network parameters were trained using Stochastic Gradient Descent with momentum except for Piggyback and BAT where it is used SGD with momentum for the classification layer and Adam [29] as optimizer for the rest of the network as suggested by their authors [47],[48]. The various methods were trained in the RGB and D settings for 30 epochs with a batch-size of 32, an initial learning rate for SGD of 0.005 with momentum of 0.9, and an initial learning rate of 0.00001 for Adam. Both the learning rates are decayed by a factor of 10 after 20 epochs. It is also added a regularization factor or weight decay of 0.000005 to both the optimization policies. The only difference that has been introduced in the RGB+D setting was training the methods for twice the epochs, that is for 60 epochs with the step of the learning rate after 45 epochs. After few tests, Piggyback shows difficulties to use the same Adam learning rate for all the tasks, so two different learning rates have been used: 0.0001 for pose estimation and scene recognition, and 0.00001 for object classification, keeping the decay of the learning rates by a factor of 10 after 20 epochs in RGB and D settings, after 45 in the RGB+D setting. Any method submitted to the challenge should adopt this set of hyper-parameters in order to be comparable with the baselines.

The three datasets offer RGB-D images of different sizes: the RGB-D object dataset contains images of size nearly 100×100 pixels, LineMOD cropped dataset contains images of size 64×64 pixels, and NYU Depth V2 Dataset contains images of size

640 × 480 pixels. To make use of them with the neural network it is necessary to rescale the images all to the same size: 224 × 224 pixels, that is the minimal dimension accepted by the implemented ResNet. The data were normalized by subtracting the mean of the ImageNet dataset to each channel. To increase the generality of the dataset, data augmentation was performed: the images of RGB-D object dataset are cropped randomly, while images of NYU Depth V2 Dataset are cropped randomly and mirrored (i.e. flipped horizontally).

5.2 Results

This section reports the results obtained by some well-known methods on the RGB-D Triathlon challenge. The results in term of accuracy for each task are reported in table 5.1. The accuracy in the tables refers to the top-1 accuracy (as defined for each task in Section 4.1) averaged between multiple runs to reduce the variance.

The results show, as one could image, that the highest accuracies are reached through the RGB+D setting. Anyway, RGB and RGB+D settings are comparable, denoting that all the methods find more useful the RGB images rather than the Depth images. This may be due to the naive approach adopted in handling the depth information. In only one case the result achieved on the RGB+D setting is less than the one reached on the RGB setting. This is the case of Piggyback in the object recognition task over the RGB-D object dataset. As can be noted in the table, it reaches 79.29 % in the RGB setting and only 78.13 % in the RGB+D one. This is mainly due to overfitting on the RGB+D setting.

Some methods achieve comparable and even higher results than fine-tuning the base network in the object recognition task. The series residual adapters are able to outperform fine-tuning in each setting, especially in RGB+D where it achieves an accuracy of 82.55 %, 2 % more than fine-tuning. However, the best method in the RGB+D setting is BAT that reaches 82.85 %, 0.3 % more than the series residual adapter. It is worth noting that ROD is a challenging dataset, in fact, many methods in the experiments suffered overfitting and it has been hard to mitigate this problem. Overfitting appears mainly due to the way the images are split into training and test dataset. The test dataset contains instances of objects that do not appear in the training set and, in many cases, the variance between physical objects of the same class is high. This scenario is challenging but also real. For example consider the mug class, in the world there are many instances of mugs with different shapes and colors and the system cannot be trained on every existing instance but must be able to recognize them when operating.

In pose estimation, there is a large gap between the feature extractor and the other methods. This is due to the fact that pose estimation is very different from the task on which the pre-trained model was trained (image classification on ImageNet dataset). Thus, the feature extraction is not able to extract interesting features and it leads to poor performances. Moreover, the behavior of the feature extraction underlines the ability of the other methods, especially Piggyback and BAT to adapt to a new task very different from the original one, reaching an accuracy similar to the one achieved by fine-tuning the backbone network. Differently from object recognition, in this task, no method outperforms fine-tuning the base network parameters but all the methods, except feature extraction, achieve comparable results, especially on the RGB+D setting.

Similarly to object recognition, in scene classification some methods outperform fine-tuning the base network, achieving higher accuracy. In the RGB setting, the series residual adapter reaches an accuracy of 70.02 %, nearly 0.5 % higher than the fine-tuning, while Piggyback reaches approximately the same accuracy of the fine-tuning. In the D setting, the series residual adapter method is still the best one, while the other methods do not converge to an optimal accuracy, reaching results approximately lower by 3 % than the fine-tuning. In the RGB+D setting, there is a tie for the best method between Piggyback and BAT, in fact, both reach an accuracy of 71.46 %, a result 0.07 % better than fine-tuning.

Overall, the best results are achieved by the series residual adapter method and BAT. Both show a great ability to adapt to new tasks, often reaching results comparable to or better than fine-tuning the base network. Also Piggyback showed this ability but it requires to set carefully the hyper-parameters; recall that the learning rate has been adapted to each task to obtain competitive results.

In the following it is analyzed how well these methods perform according to the scores defined in Section 4.2: average accuracy (AvgA), average accuracy - parameters ratio (AAPR), decathlon (Deca), revised decathlon (RevD), and linear. The hyper-parameters used to compute the metrics are the one discussed in Section 4.2. The tables 5.2, 5.3, 5.4 report the scores for each method and metric, respectively for RGB, D, and RGB+D settings. The tables report an additional column (*LiWP*) that represents the linear score neglecting the number of parameters.

In the tables, the *Par.* column reports the sum of parameters used for all the tasks respect the parameters of the base network. For example, Piggyback uses $1.10 \cdot p_0$ parameters and that the number of additional parameters is $0.10 \cdot p_0$, where p_0 is the number of parameters of the backbone network. The total number of parameters is computed by calculating the total size of parameters in bits excluding the classification layers.

On the *average accuracy* metric the best methods are fine-tuning over RGB and D settings, and BAT on the RGB+D setting. However, this metric does not keep into account the number of parameters and, moreover, it is not capable to consider the difficulties on the tasks. For example, on the RGB setting, even if the serial residual adapter method is better than BAT in both object classification and scene recognition, it is penalized in the overall score because it loses many points on pose

Methods for each dataset	**RGB**	**D**	**RGB+D**
RGB-D object dataset [34]			
Fine-tuning	79,44	64,03	80,92
Feature extractor	70,44	49,47	74,65
Piggyback [47]	79,29	60,68	78,13
BAT [48]	79,29	62,62	82,85
R.a. series [62]	79,84	64,20	82,55
R.a. parallel [63]	78,50	63,05	81,27
LineMOD dataset [22]			
Fine-tuning	96,86	86,14	98,43
Feature extractor	14,00	10,11	17,39
Piggyback [47]	95,33	84,88	96,93
BAT [48]	95,83	85,41	96,66
R.a. series [62]	92,76	80,10	97,29
R.a. parallel [63]	91,77	82,60	95,27
NYU Depth V2 dataset [53]			
Fine-tuning	69,59	61,16	71,39
Feature extractor	58,81	48,59	63,58
Piggyback [47]	69,60	57,21	71,46
BAT [48]	68,91	57,73	71,46
R.a. series [62]	70,02	61,34	70,70
R.a. parallel [63]	68,61	58,52	70,94

Table 5.1: Results in term of accuracy for the RGB-D Triathlon challenge.

estimation, that is a complex task, as indicated by the result of the feature extraction method. The *average accuracy - parameter ratio* improves the previous score penalizing the methods that adds extra parameters. In this metric fine-tuning is strongly penalized and achieves the last score. The best methods are Piggyback and BAT that adds very few parameters while keeping good performances. However, this metric, as the previous, does not keep into consideration the difficulties of the tasks. The *decathlon score* takes into account the different complexity among tasks by assigning to each task a score between 0 and 1000. However, in the attempt of penalizing the methods that performs well in only some task and poorly in others, the definition of the maximal error as the double of the error committed by fine-tuning the base network weights lead to the same problem of the previous metrics. Few percentage points lost on pose estimation leads to a poor score. This aspect will be further analyzed later comparing the scores of Piggyback and BAT. The *revised decathlon score* improves the previous metric by penalizing the methods that uses additional parameters but, having the maximal error defined in the same way, it suffers the same problem. Finally, the *linear score* tries to solve the issues of the previous methods by both considering the number of additional parameters and the complexity of the tasks. The number of parameters is considered by scaling the result for a factor proportional to the number of additional parameters. As it can be seen, methods are strongly penalized for using additional parameters. For example, the series residual adapter has accuracies comparable to Piggyback or BAT, but it introduces more per-task parameters than these methods and so, the score is lower. It considers also the complexity of the task, in fact, neglecting the number of parameters and considering the RGB setting (see *LiWP* column of table 5.2), the series residual adapter method achieves 1011 points, even more than the fine-tuning (1000) and BAT (969). This is due to the fact that the linear score compares the accuracies with both the fine-tuning and the feature extractor methods, producing better estimates of the complexity of the tasks. For this reason the linear score is considered the best way to evaluate the RGB-D Triathlon challenge.

In the scores that penalize the use of additional parameters the fine-tuning method presents the worst results. Fine-tuning introduces a copy of the network for each new task and so, it is strongly penalized by the number of parameters in these metrics: in the average accuracy - parameter ratio score its result is one third of the one obtained without penalizing the parameters; in the revised Decathlon score it loses nearly 600 points with respect to the decathlon score; in the linear score it achieves zero and this is due to the design of the score itself that assigns a score of zero for the task in which the method adds as many parameters as the number of parameters of the base network.

Feature extractor, instead, does not add any extra parameters. It achieves the same score in metrics that consider parameters and in the ones that do not consider it. The average accuracy and average accuracy - parameter ratio score are in fact equals and the same applies to the decathlon and the revised decathlon scores. However, even if it is not penalized by adding extra parameters, its results are poor, especially on pose estimation, and the score is low compared to the other methods. In the linear score, it achieves zero by the design of the score itself.

Series residual adapter method outperforms the parallel residual adapter method. However, the latter adds less parameters and so the scores in some settings are comparable. For example, considering the revised decathlon score, the parallel residual adapter method reaches a score nearly equal to the series in the RGB setting and a higher score in both D and RGB+D settings. However, in the linear score, series achieves a higher result in every setting, even if they are nearly equal in the RGB+D setting. This is due to the fact that the revised decathlon score penalizes exponentially the use of additional parameters.

Even if the accuracies between BAT and series residual adapters are similar, the number of parameters changes a lot. BAT introduces only a few parameters per task, nearly 3.3 %, while the serial residual adapter introduces nearly 15.6 % for each new task. For this reason, the scores of the series residual adapter method in the metrics that keep the parameters into account are lower than the one of BAT.

Piggyback and BAT use the same number of parameters. Piggyback achieves the best linear score in the RGB setting while BAT achieves a better decathlon and revised decathlon score. This difference can be explained considering how decathlon score and linear score handle the pose estimation accuracy. Decathlon score encourages methods to get a good score in all tasks by computing the score of each task raising by two the difference between the maximal error, set to the double of the fine-tuning error, and the method's error. This means that the decathlon score maps the 1000 points available for this task in very few percentage points. Considering the RGB case, a score of zero is assigned for methods that achieve accuracy below 93.76 %, and 1000 points for methods that achieve 100%. In linear score instead adopts a completely different approach. Recalling what is defined in Section 4.2 this score computes the fraction between the difference of the accuracy of the method and the accuracy of the feature extraction, and the difference between the accuracy of the fine-tuning and the feature extraction. For pose estimation in the RGB setting, the gap at the denominator is large, approximately 80, then, methods that achieve similar accuracy, i.e. few percentage points of difference, will achieve a similar score. Note that this does not apply to other tasks, for example object recognition, because the accuracies of the feature extraction and the fine-tuning methods are similar, producing a small denominator. In the other settings BAT outperforms Piggyback, especially in the RGB+D setting.

It is worth noting that BAT reaches a linear score in the RGB+D setting (1030.61) that is greater than the optimal score achieved by a method that does not introduce additional parameters and perform as the fine-tuning (1000). BAT, in fact, outperforms fine-tuning in object classification and scene recognition while introducing very few parameters.

Metric	Par.	AvgA	AAPR	Deca	RevD	LiWP	**Linear**
Fine-tuning	3	**81.96**	27.32	**750.00**	161.58	<u>1000.00</u>	0.00
Feature extractor	1	47.75	47.75	183.20	183.20	0.00	0.00
Piggyback [47]	1.10	<u>81.41</u>	**74.32**	562.27	<u>460.11</u>	988.62	**927.69**
BAT [48]	1.10	81.34	<u>74.26</u>	<u>597.85</u>	**489.21**	969.29	<u>909.54</u>
R.a. series [62]	1.47	80.87	55.00	516.87	247.43	**1011.46**	737.25
R.a. parallel [63]	1.38	79.63	57.89	461.83	246.31	914.40	710.94

Table 5.2: Results in term of accuracy for the RGB-D Triathlon challenge in the RGB setting. The best method is bold, the second best is underlined.

Metric	Par.	AvgA	AAPR	Deca	RevD	LiWP	**Linear**
Fine-tuning	3	**70.44**	23.48	**750.00**	161.58	**1000.00**	0.00
Feature extractor	1	36.06	36.06	202.99	202.99	0.00	0.00
Piggyback [47]	1.10	67.59	<u>61.70</u>	614.00	<u>502.45</u>	813.09	<u>762.98</u>
BAT [48]	1.10	<u>68.59</u>	**62.61**	<u>663.08</u>	**542.59**	873.59	**819.74**
R.a. series [62]	1.47	68.55	46.62	584.38	279.75	<u>982.30</u>	715.99
R.a. parallel [63]	1.38	68.06	49.48	592.45	315.97	892.11	693.61

Table 5.3: Results in term of accuracy for the RGB-D Triathlon challenge in the D setting. The best method is bold, the second best is underlined.

Metric	Par.	AvgA	AAPR	Deca	RevD	LiWP	**Linear**
Fine-tuning	3	<u>83.58</u>	27.86	**750.00**	161.58	1000.00	0.00
Feature extractor	1	51.87	51.87	244.75	244.75	0.00	0.00
Piggyback [47]	1.10	82.17	<u>75.02</u>	433.78	<u>354.97</u>	848.09	<u>795.82</u>
BAT [48]	1.10	**83.66**	**76.37**	<u>554.37</u>	**453.63**	**1098.31**	**1030.61**
R.a. series [62]	1.47	83.51	56.80	551.39	263.96	<u>1052.22</u>	766.96
R.a. parallel [63]	1.38	82.50	59.97	501.40	267.41	986.32	766.86

Table 5.4: Results in term of accuracy for the RGB-D Triathlon challenge in the RGB+D setting. The best method is bold, the second best is underlined.

Chapter 6

Conclusion and future works

In this thesis a new benchmark, the RGB-D Triathlon challenge, was proposed. It aims to push forward the state of the art in multi-task life-long learning in robotics context by encouraging researchers to develop methods able to perform multiple tasks while using as few parameters as possible. The RGB-D Triathlon challenge presents three fundamental tasks: object classification, pose estimation, and scene recognition. Object classification and scene recognition can be seen as a specialization of a more general task: image classification. Object classification task is evaluated on the RGB-D object dataset [34] and scene recognition on the NYU Depth V2 dataset [53]. Pose estimation is a complex problem that requires to define the representation of the rotation and the solution space. Despite many works consider it a classification problem where the pose space is discretized into bins, this work takes the approach of [44] where the pose estimation task is considered a regression problem in a non-Euclidean space. This task has to be performed on the LineMOD dataset [22].

This work presented three settings for the challenge that differ on the data that are considered: the RGB setting considers RGB images, the D setting considers depth images, and the RGB+D setting considers both RGB and depth images. Researchers interested in submitting methods to this challenge can chose the setting they care most because each setting has its own leader-board and it is independent from others. This work proposed a simple naive approach to handle depth images that does not require to change the backbone network between different settings. Each depth image is a single-channel image in which each pixel represents the distance between that point and the camera. Every depth image is transformed into a three channel image by simply duplicating the depth channel many times, allowing well-known networks designed to process RGB images to elaborate it. This strategy does not exploit at best the depth information but can be easily implemented, allowing researchers to develop their own methods by using well-known architectures, such as a ResNet-18, for which implementations and pre-trained models can be easily found on-line.

For each method submitted to the challenge is assigned a score. Chapter 4 analyzed many possible scores aiming to find the best metric that keeps in consideration two factors: the accuracy achieved by the method on each task and the number of additional task-specific parameters introduced. The metric must assign an higher score to method that performs well on each task by using a minimal number of additional parameters. The metric must consider also the complexity of the tasks and must assign scores without preferring any task to others. The chapter started by evaluating trivial scores: average accuracy score is computed by averaging the performances obtained on each task, and average accuracy - parameter ratio score is computed in the same way but adds a factor that penalize the use of additional parameters for solving the tasks. Then it was discussed the metric defined in the context of the Visual Decathlon Challenge [62] that is further extended by proposing a revised version of it that takes into account the number of parameter used. These metrics, however, do not estimate well enough the complexity of the tasks. Thus, it is introduced an additional metric called *linear score*. It considers both the number of additional parameters and the accuracy achieved and compares the accuracy of each task with both the accuracies of the feature extractor and the fine-tuning methods. By doing this it is able to perform a better estimate of the complexity of the tasks.

Four methods were evaluated against the RGB-D Triathlon challenge. These methods were developed in the context of multiple domain learning: series residual adapters [62], parallel residual adapter [63], piggyback [47] and binarized affine fransformation (BAT) [48]. The results show that Piggyback and BAT achieve the best scores by far, even if the series residual adapter achieves comparable performances in term of accuracy. This is due to the fact that these methods use less parameters, thus, the score penalizes more the series residual adapter method than the other two methods.

We are interested to test the RGB-D Triathlon challenge with other methods. For example, we are considering to test different backbone networks such as VGGNet [75], a deeper Resnet [19] (Resnet-50 or Resnet-101), or a DenseNet [25]. This is useful for two reasons: (i) it allows us to consider different quantities in the number of parameters of the backbone network and, (ii) we provide additional baselines with which researchers can compare their own methods. Moreover, we are considering to evaluate other methods that have proven to be effective in sequential and life-long learning such as Domain adaptation network [68] and Life-Long Learning [37].

This work encourages the community to develop methods able to face many different tasks in robotics by providing an evaluation benchmark. Obviously, the RGB-D

Triathlon challenge is only a starting point to provide a much more complex benchmark, where must be included more complex tasks such as semantic segmentation [41], grasp prediction [13], [27] and object detection [65], [66]. Methods that will achieve a high score on this extended challenge will really be an agile visual toolbox for robots and a big step toward general artificial intelligence.

Bibliography

[1] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, “Memory Aware Synapses: Learning what (not) to forget”, *arXiv preprint arXiv:1711.09601*, 2017.

[2] P. Ammirato, A. C. Berg, and J. Košecká, “Active Vision Dataset Benchmark”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2046–2049.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (SURF)”, *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[4] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation”, *arXiv preprint arXiv:1308.3432*, 2013.

[5] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult”, *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[6] H. Bilen and A. Vedaldi, “Universal representations: The missing link between faces, text, planktons, and cat breeds”, *arXiv preprint arXiv:1701.07275*, 2017.

[7] R. Caruana, “Multitask Learning”, *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997, ISSN: 1573-0565. DOI: `10.1023/A:1007379606734`. [Online]. Available: `https://doi.org/10.1023/A:1007379606734`.

[8] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, “Describing textures in the wild”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3606–3613.

[9] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning”, in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 160–167.

[10] G. Cybenko, “Approximation by superpositions of a sigmoidal function”, *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[11] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection”, in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 886–893.

[12] L. Deng, G. Hinton, and B. Kingsbury, “New types of deep neural network learning for speech recognition and related applications: An overview”, in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, IEEE, 2013, pp. 8599–8603.

[13] A. Depierre, E. Dellandréa, and L. Chen, “Jacquard: A Large Scale Dataset for Robotic Grasp Detection”, *arXiv preprint arXiv:1803.11469*, 2018.

[14] N. Doulamis and A. Voulodimos, “FAST-MDL: Fast Adaptive Supervised Training of multi-layered deep learning models for consistent object tracking and classification”, in *Imaging Systems and Techniques (IST), 2016 IEEE International Conference on*, IEEE, 2016, pp. 318–323.

[15] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, “An empirical investigation of catastrophic forgetting in gradient-based neural networks”, *arXiv preprint arXiv:1312.6211*, 2013.

[16] S. Gupta, P. Arbelaez, and J. Malik, “Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images”, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[17] T. L. Hayes, R. Kemker, N. D. Cahill, and C. Kanan, “New Metrics and Experimental Paradigms for Continual Learning”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2031–2034.

[18] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn”, in *Computer Vision (ICCV), 2017 IEEE International Conference on*, IEEE, 2017, pp. 2980–2988.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] M. Hejrati and D. Ramanan, “Analysis by synthesis: 3d object recognition by object reconstruction”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2449–2456.

[21] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, “Multi-task deep reinforcement learning with popart”, *arXiv preprint arXiv:1809.04474*, 2018.

[22] S. Hinterstoisser S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes", 2011.

[23] G. Hinton. (2012). Neural networks for machine learning, [Online]. Available: `https://www.coursera.org/learn/neural-networks` (visited on 10/31/2018).

[24] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks", *arXiv preprint arXiv:1709.01507*, vol. 7, 2017.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks.", in *CVPR*, vol. 1, 2017, p. 3.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *arXiv preprint arXiv:1502.03167*, 2015.

[27] E. Johns, S. Leutenegger, and A. J. Davison, "Deep learning a grasp function for grasping under gripper pose uncertainty", in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, IEEE, 2016, pp. 4461–4468.

[28] A. Karpathy. (2018). Convolutional Neural Networks (CNN / ConvNets), [Online]. Available: `http://cs231n.github.io/` (visited on 10/15/2018).

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[30] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks", *Proceedings of the national academy of sciences*, p. 201 611 835, 2017.

[31] I. Kokkinos, "UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory.", in *CVPR*, vol. 2, 2017, p. 8.

[32] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images", Citeseer, Tech. Rep., 2009.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[34] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset", in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1817–1824. DOI: `10.1109/ICRA.2011.5980382`.

[35] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction", *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[36] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection", *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[37] Z. Li and D. Hoiem, "Learning without forgetting", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[38] D. Lin, S. Fidler, and R. Urtasun, "Holistic Scene Understanding for 3D Object Detection with RGBD Cameras", in *The IEEE International Conference on Computer Vision (ICCV)*, 2013.

[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context", in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector", in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[41] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[42] D. G. Lowe, "Object recognition from local scale-invariant features", in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Ieee, vol. 2, 1999, pp. 1150–1157.

[43] ——, "Distinctive image features from scale-invariant keypoints", *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[44] S. Mahendran, H. Ali, and R. Vidal, "3D pose regression using convolutional neural networks", in *IEEE International Conference on Computer Vision*, vol. 1, 2017, p. 4.

[45] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft", *arXiv preprint arXiv:1306.5151*, 2013.

[46] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning", *arXiv preprint arXiv:1711.05769*, vol. 1, no. 2, p. 3, 2017.

[47] ——, "Piggyback: Adding Multiple Tasks to a Single, Fixed Network by Learning to Mask", *arXiv preprint arXiv:1801.06519*, 2018.

[48] M. Mancini, E. Ricci, B. Caputo, and S. R. Bulò, "Adding New Tasks to a Single Network with Weight Trasformations using Binary Masks", *CoRR*, vol. abs/1805.11119, 2018.

[49] B. McCann, N. S. Keskar, C. Xiong, and R. Socher, “The natural language decathlon: Multitask learning as question answering”, *arXiv preprint arXiv:1806.08730*, 2018.

[50] M. McCloskey and N. Cohen, “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem”, English (US), *Psychology of Learning and Motivation - Advances in Research and Theory*, vol. 24, no. C, pp. 109–165, Jan. 1989, ISSN: 0079-7421. DOI: `10.1016/S0079-7421(08)60536-8`.

[51] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B Dalvi, M. Gardner, B. Kisiel, *et al.*, “Never-ending learning”, *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, 2018.

[52] S. Munder and D. M. Gavrila, “An experimental study on pedestrian classification”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.

[53] P. K. Nathan Silberman Derek Hoiem and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images”, in *ECCV*, 2012.

[54] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning”, in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, 2011, p. 5.

[55] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes”, in *Computer Vision, Graphics & Image Processing, 2008. ICVGIP’08. Sixth Indian Conference on*, IEEE, 2008, pp. 722–729.

[56] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation”, in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[57] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, “6-dof object pose from semantic keypoints”, in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 2011–2018.

[58] B. Pepik, M. Stark, P. Gehler, and B. Schiele, “Teaching 3d geometry to deformable part models”, in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, IEEE, 2012, pp. 3362–3369.

[59] M. Rad and V. Lepetit, “BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth”, in *International Conference on Computer Vision*, vol. 1, 2017, p. 5.

[60] H. Rahmani, A. Mian, and M. Shah, “Learning a deep model for human action recognition from novel viewpoints”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 667–681, 2018.

[61] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, “Massively multitask networks for drug discovery”, *arXiv preprint arXiv:1502.02072*, 2015.

[62] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, “Learning multiple visual domains with residual adapters”, in *Advances in Neural Information Processing Systems*, 2017, pp. 506–516.

[63] ——, “Efficient parametrization of multi-domain deep neural networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8119–8127.

[64] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “icarl: Incremental classifier and representation learning”, in *Proc. CVPR*, 2017.

[65] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[66] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, in *Advances in neural information processing systems*, 2015, pp. 91–99.

[67] (2018). Robust Vision Challenge, [Online]. Available: `http://www.robustvision.net` (visited on 11/01/2018).

[68] A. Rosenfeld and J. K. Tsotsos, “Incremental learning through deep adaptation”, *arXiv preprint arXiv:1705.04228*, 2017.

[69] X. Roynard, J.-E. Deschaud, and F. Goulette, “Paris-Lille-3D: A Point Cloud Dataset for Urban Scene Segmentation and Classification”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 2027–2030.

[70] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: `10.1007/s11263-015-0816-y`.

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge”, *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[72] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks”, *arXiv preprint arXiv:1606.04671*, 2016.

[73] H. Schneiderman and T. Kanade, “A statistical method for 3D object detection applied to faces and cars”, in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, IEEE, vol. 1, 2000, pp. 746–751.

[74] M. Schwarz, A. Milan, A. S. Periyasamy, and S. Behnke, “RGB-D object detection and semantic segmentation for autonomous manipulation in clutter”, *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 437–451, 2018.

[75] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *CoRR*, vol. abs/1409.1556, 2014.

[76] K. Soomro, A. R. Zamir, and M. Shah, “UCF101: A dataset of 101 human actions classes from videos in the wild”, *arXiv preprint arXiv:1212.0402*, 2012.

[77] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net”, *arXiv preprint arXiv:1412.6806*, 2014.

[78] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting”, *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[79] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”, *Neural networks*, vol. 32, pp. 323–332, 2012.

[80] D. Stutz, “Superpixel segmentation: an evaluation”, in *German Conference on Pattern Recognition*, Springer, 2015, pp. 555–562.

[81] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, “Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2686–2694.

[82] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, *et al.*, “The limits and potentials of deep learning for robotics”, *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[83] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft, and M. Milford, “Place categorization and semantic mapping on a mobile robot”, in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, IEEE, 2016, pp. 5729–5736.

[84] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[85] S. Thrun, “Lifelong learning algorithms”, in *Learning to learn*, Springer, 1998, pp. 181–209.

[86] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660.

[87] J. Tremblay, T. To, and S. Birchfield, “Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation”, *arXiv preprint arXiv:1804.06534*, 2018.

[88] S. Tulsiani and J. Malik, “Viewpoints and keypoints”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1510–1519.

[89] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, “Deep learning for identifying metastatic breast cancer”, *arXiv preprint arXiv:1606.05718*, 2016.

[90] P. Wohlhart and V. Lepetit, “Learning descriptors for object recognition and 3d pose estimation”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3109–3118.

[91] C. Workshop. (2018). Real-World Challenges and New Benchmarks for Deep Learning in Robotic Vision, [Online]. Available: `https://sites.google.com/view/cvpr2018-robotic-vision` (visited on 11/01/2018).

[92] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman, “Single image 3d interpreter network”, in *European Conference on Computer Vision*, Springer, 2016, pp. 365–382.

[93] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes”, *arXiv preprint arXiv:1711.00199*, 2017.

[94] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated Residual Transformations for Deep Neural Networks”, *arXiv preprint arXiv:1611.05431*, 2016.

[95] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks”, in *European conference on computer vision*, Springer, 2014, pp. 818–833.

[96] Y. Zhang, M. Bai, P. Kohli, S. Izadi, and J. Xiao, “Deepcontext: Context-encoding neural pathways for 3d holistic scene understanding”, *arXiv preprint arXiv:1603.04922*, 2016.

[97] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, “Facial landmark detection by deep multi-task learning”, in *European Conference on Computer Vision*, Springer, 2014, pp. 94–108.

[98] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, “Places: A 10 million image database for scene recognition”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 6, pp. 1452–1464, 2018.