DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Multitask Deep Learning models for real-time deployment in embedded systems

MIQUEL MARTÍ I RABADÁN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Multitask Deep Learning models for real-time deployment in embedded systems

MIQUEL MARTÍ I RABADÁN

Degree Programme in Electrical Engineering
Master's degree in Telecommunications Engineering
Date: May 26, 2017
Supervisors: Atsuto Maki, Montse Pardàs
Examiners: Hedvig Kjellström, Elisa Sayrol
Swedish title: Deep Learning-modeller för multitaskproblem, anpassade för inbyggda system i realtidsapplikationer
Catalan title: Models multi-tasca basats en xarxes neuronals d'aprenentage profund per al desplegament en sistemes encastats i en temps real
KTH School of Electrical Engineering
UPC Barcelona School of Telecommunications Engineering

Abstract

Multitask Learning (MTL) was conceived as an approach to improve the generalization ability of machine learning models. When applied to neural networks, multitask models take advantage of sharing resources for reducing the total inference time, memory footprint and model size. We propose MTL as a way to speed up deep learning models for applications in which multiple tasks need to be solved simultaneously, which is particularly useful in embedded, real-time systems such as the ones found in autonomous cars or UAVs.

In order to study this approach, we apply MTL to a Computer Vision problem in which both Object Detection and Semantic Segmentation tasks are solved based on the Single Shot Multibox Detector and Fully Convolutional Networks with skip connections respectively, using a ResNet-50 as the base network. We train multitask models for two different datasets, Pascal VOC, which is used to validate the decisions made, and a combination of datasets with Aerial View images captured from UAVs.

Finally, we analyse the challenges that appear during the process of training multitask networks and try to overcome them. However, these hinder the capacity of our multitask models to reach the performance of the best single-task ones trained without the limitations imposed by applying MTL. Nevertheless, multitask networks benefit from sharing resources and are 1.6x faster, lighter and use less memory compared to deploying the single-task models in parallel, which turns essential when running them on a Jetson TX1 SoC as the parallel approach does not fit into memory. We conclude that MTL has the potential to give superior performance as far as the object detection and semantic segmentation tasks are concerned in exchange of a more complex training process that requires overcoming challenges not present in the training of single-task models.

Contents

Chapter 1

Introduction

This thesis lies in the area of Deep Learning (DL) within the wider field of Machine Learning (ML). It focuses on using Multitask Learning (MTL) for the deployment of DL models in embedded systems for real-time applications in cases where more than one task needs to be solved simultaneously. In this section the motivation for carrying out this thesis is explained, the approach that will be followed is justified and the contributions specified.

1.1 Background

Typically, deep learning models (and machine learning models in general) focus on solving one task at a time. However, applications often require more than one task to be performed simultaneously and if that is the case the naive solution is to deploy different models running in parallel, one for each task. In fact, [1] claims that most real-world problems can be seen as multitask problems, and that, although few of the test problems in machine learning repositories are seen as multitask ones, this is because most have often been broken to fit into smaller subproblems beforehand.

Examples of applications in which more than one task needs to be solved are easy to find in multiple domains: in Natural Language Processing one might want to translate from one language to multiple languages [2], in Medical Imaging multiple clinical variables can be estimated from a same brain image [3] and in Computer Vision (CV) a self-driving car might

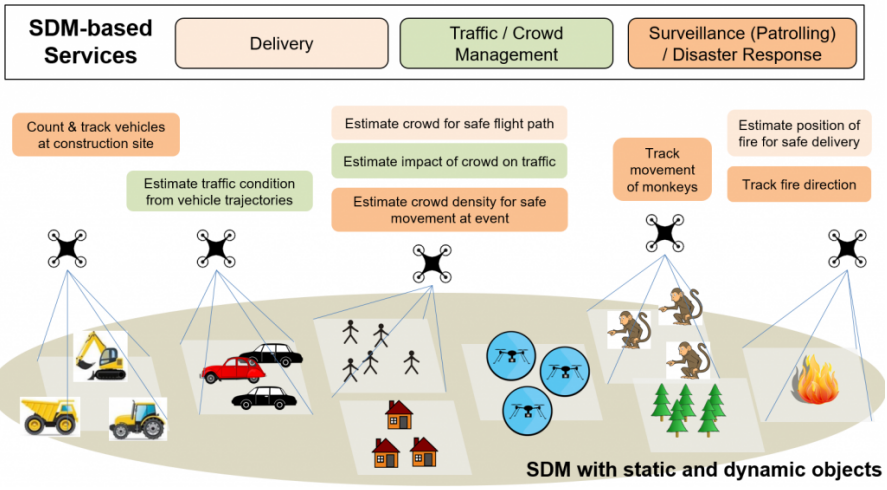


Figure 1.1: The Shared Dynamic Map (SDM) presents all static and time-varying (dynamic) objects that are captured by the UAVs. In this way, the SDM is a foundational technology for applications and services including delivery, traffic/crowd management, surveillance, disaster response, and so on. In this study, we do not consider the shared aspect of SDMs so we refer to them as dynamic maps. From [4].

need to do semantic segmentation to differentiate between *road* and *tree* while at the same time estimating the depth of the different objects in the image to know when to brake.

Another example, which we will use as a study case in this thesis, is that of creating dynamic maps from Unmanned Aerial Vehicle (UAV) imagery, see Figure 1.1. The dynamic map is built by doing semantic segmentation and Multiple Object Tracking (MOT) on aerial view images for capturing the static, larger objects and the smaller, dynamic objects respectively and later representing them on a traditional map over time. A number of applications can benefit from such a map: disaster response, traffic or crowd management or surveillance.

The creation of a dynamic map is another example of an application that needs multiple tasks to be solved simultaneously. [5] developed a DL model for semantic segmentation from UAV imagery to get the static, larger objects present in UAV imagery such as *roads*, *buildings* or *water bodies*. In order to obtain the dynamic objects, MOT[6] needs to be introduced. Following the common *tracking-by-detection*[7] approach to MOT, an object detection model needs to be developed before for

obtaining the detections for each frame that would later be used to do the data association step across them with each identified object.

Solving multitask problems becomes most relevant when it happens in embedded systems - which can be found on autonomous systems or mobile devices such as UAVs, robots, smartphones, wearables, virtual / augmented reality headsets or portable game consoles. Such systems typically run on low power, which in turn limits the computing power they can provide, and are limited on terms of memory available, making the lack of resources an important concern. Moreover, their applications often have the extra requirement of having to run in real-time, meaning small inference times are essential and off-board computation is not possible.

Nevertheless, DL solutions deployed in desktop or datacenter systems without the previously mentioned constraints could also benefit of models that allow more efficient computation and have smaller memory footprints, being able to process more information in a quicker and/or cheaper way.

1.2 Challenges

Embedded systems as the ones mentioned in the previous section can be based on a Central Processing Unit (CPU) and include a Graphics Processing Unit (GPU) or can be based on a Field-programmable Gate Array (FPGA) or Application-specific Integrated Circuit (ASIC). Each of these approaches serves different applications and poses different challenges, e.g. a typical FPGA has little on-chip memory[1] but is much more efficient than a CPU.

Here the focus is on the more flexible but slower CPU or GPU-based embedded systems. In those, the available resources range from very limited to moderate but are never close to the systems in which deep learning models are trained and where their performance is usually evaluated. As an example, Table 1.1 shows a comparison between the resources available in two of the most powerful System-on-a-chip (SoC) including a GPU available on the market, the Jetson TX1 and TX2, and the most

[1] As an example a Xilinx Spartan-7 FPGA has at most 0.5 MB of on-chip memory.

powerful consumer-grade GPU, the Titan X Pascal, among other GPUs in between.

Table 1.1: GPU models comparison. Tegra X1 and Tegra P1 are the chips in the Jetson TX1 and TX2 respectively. *: memory is shared with the system. Values from [8], [9].

GPU model	FP32 Processing power (GFLOPS)	Memory (GB)
Tegra X1	512	4*
Tegra P1	750	8*
GTX 980	4612	4
GTX Titan X	6144	12
Titan X Pascal	11366	12

This poses three main challenges from the point of view of deployment of deep learning models for inference, which are nothing but accentuated when it is not one but multiple models that have to be deployed simultaneously. The challenges are in terms of memory requirement, both in disk and physical memory, and computational resources needed to achieve small inference times that allow high enough throughput to give real-time performance.

First, the model weights need to be stored on disk to be later used. Embedded systems are typically constrained in the size of their storage and the weights of the models can take hundreds of MB[2], which can mean a large share of the total available storage space. Moreover, imagine a smartphone app that runs AlexNet for classifying images in the camera album and create nicely ordered albums depending on the content of the image. Such an app would need to download the weights from the Internet and store them, making the size of the app too large to be practical. If we need more than one task and have one model for each, we have to store the weights for each model, even if they are not to run simultaneously.

Second, the model has to fit into memory. Not only the parameters have to be loaded into memory but the outputs of intermediate layers must be stored as well, whose size depends on the size of the input data and

[2]For example, the file containing the weights of the reference AlexNet model for Image Classification is 233 MB [10] when using the Caffe framework[11]. Find it at: https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

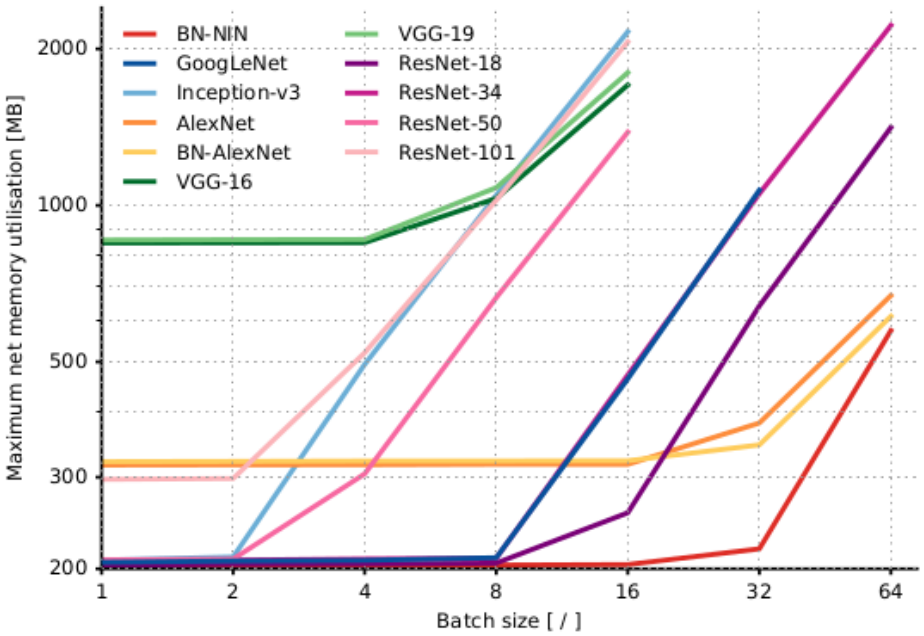


Figure 1.2: Memory vs. batch size. Maximum system memory utilization for batches of different sizes. Memory usage shows a knee graph, due to the network model memory static allocation and the variable memory used by batch size. From [12].

the batch size. Figure 1.2 shows the relation between batch size and memory utilization for common convolutional networks used for Image Classification. This might not be a problem for models running in data centers but clearly compromises models running in embedded systems with low memory available, both if they run on CPU or on GPU with dedicated memory. If two models are needed in parallel - assuming they are the same size - the memory footprint is doubled.

Third, the model has to be run to get the inference results, i.e. a forward pass of the model has to be computed for each sample of the input data. Focusing now on the GPU-accelerated inference, the best throughput is achieved typically by making full use of the parallelism capabilities of the GPU using a batch size larger than one, which introduces some latency. When its utilization gets close to the maximum the inference time is proportional to the computational complexity or number of operations that needs to be performed [12]. See Figure 1.3. In fully-connected and convolutional networks, the computational complexity is dominated

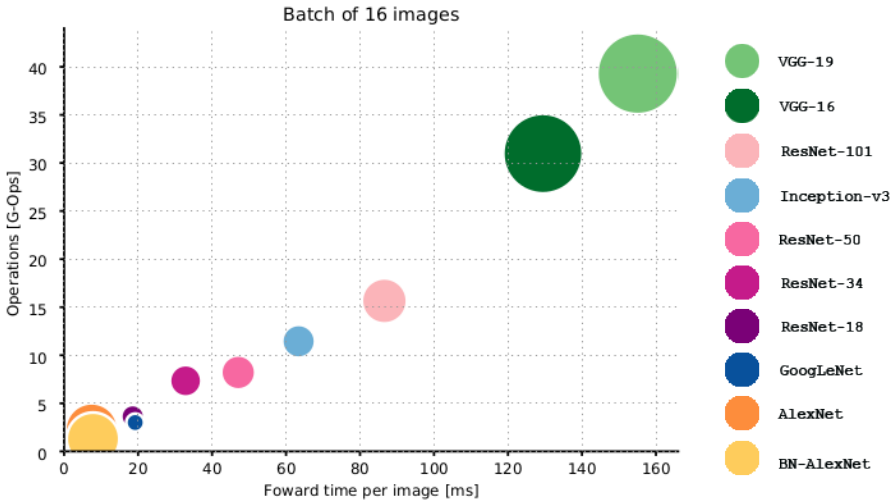


Figure 1.3: Operations vs. inference time, size proportional to # parameters. Relationship between operations and inference time for batch size 16, where resources are fully used and, therefore, the operations count represents an optimal estimation of inference time. From [12].

by multiply-adds, typically measured in FLoating-point Operations Per Second (FLOPS). With two models in parallel we will have double the number of operations, i.e. double the inference time (assuming they reach the maximum utilization of the GPU or are directly running sequentially on a single CPU).

1.3 Approach

Multitask learning learns to solve tasks in parallel using a shared representation of the same input data. One straightforward way of implementing MTL is with Neural Networks (NN) which share a base trunk or set of hidden layers that learns a common representation for the multiple tasks and from which a number of task-specific branches emerge, giving multiple and different outputs related to each task.

Using a part of the network to learn a shared representation also means computing this representation in a shared manner, sharing the resources both in terms of memory and computation cost. In this way, only the branches emerging from the shared representation must be stored and

computed in parallel, amounting to the extra cost over having a single task network. Depending on the relative size of these task-specific branches over the shared trunk the multitask network can be much larger than the single-task network or almost negligible.

In practice, networks solving multiple, different tasks are already based on the same networks. For example, most of the lately proposed networks for CV tasks are based on VGG [13] or ResNet [14] networks previously trained for Image Classification. The non-task-specific layers of these networks amount for most of the weights and computational complexity. These are perfect candidates to be used as the shared base trunks for multitask networks as they have already been proved useful for solving many different tasks in CV related problems, as previously investigated in the literature [15]–[17]; in [18] features extracted from the OverFeat network [19] are used for solving up to 5 different tasks.

Moreover, previous research proves that by exploiting synergies between related tasks the models can improve their generalization performance, as the extra information in the training signals of other tasks works as an inductive bias towards the more general solution that gives better performance for the multiple tasks [1].

MTL seems thus to be a win-win approach for deployment of models in real-world problems, and specifically in those constrained by real-time requirements and by the limited resources available in embedded systems. As a downside, the training process turns out to be more complex as has already been analyzed in previous research [1].

The study of the viability of MTL for this purpose will focus on the proposed study case. The semantic segmentation and object detection tasks are known to be related and have been previously solved simultaneously with multitask networks [20]–[22], although never with the goals pursued here.

The multitask network approach opens up the possibility of creating a swiss-knife for different tasks which share the same input data as done in [21]. In this thesis the focus is on two tasks, but it is worth mentioning that any other number of tasks can be added as long as they have the same input, are related and are based on a convolutional trunk. For the case study conducted here, the natural task to add would be action recognition as it would directly extend the features and uses of the dynamic map for surveillance or disaster scenarios.

1.4 Contribution

This thesis focuses on the study of how the inference times, memory footprint and size of deep learning models can be reduced without compromising their prediction accuracy in applications in which more than one task needs to be solved. In particular, if multitask networks can keep the prediction accuracy at reasonable levels or even improve it while, by construction, reducing the required resources for the deployment of the models, thus achieving smaller inference times, memory usage and model size compared to the deployment of multiple single-task models at the same time.

As a case study, we train and evaluate multitask models for both semantic segmentation and object detection. First, we train on Pascal VOC, a subset of which has ground-truth annotations for both tasks, and observe that for some models they perform better than the single-task baselines trained with the same data on one of the tasks, showing that indeed MTL can be used to boost the accuracy of the models in some cases. However, we cannot match the performance of the models trained with more data.

Second, we train on a collection of datasets composed of aerial view images in which some have annotations of one task and some of the other, by interleaving samples of each in the same mini-batch. The models perform reasonably well for the semantic segmentation task but fail to produce results similar to the ones achieved with the single-task models for object detection.

Observing the training process, we realize that the semantic segmentation task learns at a much higher pace and starts overfitting well before the object detection task peaks so we conjecture that this affects negatively the training process and hinders the final performance of the model. We highlight this and other challenges imposed by applying MTL, explain how they affect the training of our models and summarize them in a number of choices that need to be made when defining the multitask models that will affect their learning process.

Third, we evaluate how our multitask models compare in terms of inference time, memory usage and model size against deploying a different model per task in parallel, without sharing resources and prove that multitask models are superior in this regard, being 1.6x faster, lighter and

less memory hungry.

1.5 Summary of chapters

This thesis is organized as follows. **Chapter 2** discusses related work available in the literature. **Chapter 3** explains the details of how the experimental work of the thesis is carried out. **Chapter 4** shows the results of training the different models and analyzes the challenges found during the process. **Chapter 5** concludes the report by summarizing the key learning points and outlining future areas for research.

Chapter 2

Background

This chapter reviews the most relevant work found in the literature related to Multitask Learning, explains how it relates to Transfer Learning and analyses the latest work on the CV tasks that serve as a case study for the problem. In addition, it reviews other approaches that aim at reducing the inference speed, memory usage or model size, that are of interest for its use in embedded systems.

2.1 Multitask Learning

Multitask Learning [1] is an approach to *inductive transfer* or transfer learning that learns to solve tasks in *parallel* using a shared representation of a common input, improving the generalization of the model for the multiple tasks by using the training signals of the different tasks as an inductive bias for the others towards a more general representation that gives better performance. MTL can be applied to different ML techniques, such as k-Nearest Neighbors, Decision Trees or Support Vector Machines [23]. It is, however, with NN and DL that it has been most used. Multitask, DL models learn a common representation by sharing a set of hidden layers between the different tasks. Each task is then attached to this shared base as a branch to have a network that finally gives multiple outputs, each related to a task. Such a network can still be trained end-to-end by defining a global objective function as the sum of the loss functions of each task. In fact, many modern networks operate

in this manner even if not formally defining multiple tasks but only multiple loss functions [20], [24] and, as pointed out by [1], most real-world Machine Learning problems are in fact multitask problems.

Multiple challenges appear in the training of MTL networks when compared to their single task training: differences in the *learning speed* of the different tasks - which can be compensated with early stopping (although this does not allow for simultaneous inference with the shared layers) and/or defining different learning rates so the different tasks achieve peak performance at the same time, *computational cost* - which is increased when compared to their single task counterparts if only one task is to be learnt but that is decreased if all tasks need to eventually be trained, decisions on the network *architecture* and specifically which and how many layers to share and how to evaluate the *relatedness* of the different tasks, which in many cases is difficult to determine beforehand [1]. Early work using MTL includes NETtalk [25], a network that learns phonemes and stresses simultaneously although again the authors saw it as a single task with multiple outputs.

More recent work is divided in research on the theory of MTL and on applied MTL to solve problems from multiple domains, both multitask problems that arise naturally and problems that integrate auxiliary tasks with the purpose of achieving better performance. Multinet[26] presents a framework for integrated perception that by recurrently encoding both the input and the output of the different tasks can learn a shared data representation or integration space to solve different sub-problems of interest simultaneously, improving the performance of the individual tasks. This approach reduces to conventional MTL when there is no recurrence. One issue when designing multitask architectures is how to decide which layers to share and where to start the task specific part of the networks. In [27] applying tensor factorization techniques allows to automatically learn the sharing structure. In [28] a deep relationship network jointly leans transferable features and task relationships to circumvent *under-transfer* due to not sharing upper layers and *negative-transfer* when sharing too many features for tasks that are not related enough. In [29] a *cross-stitch* unit for convolutional neural networks is introduced to explicitly learn what and how much to share between layers of two or more networks. However, this last approach is not useful from our perspective as there is no shared computation, which is the feature from MTL we are after. The approach taken by [30] is the most promising for our pur-

pose as keeps an eye on having a low-memory footprint and low latency during inference while learning the multitask architecture and sharing among the different tasks. It achieves so by growing from a thin network during the training phase with a criterion that promotes grouping of related tasks while penalizing model complexity.

2.2 Multitask learning in Computer Vision

MTL has been applied for many different problems within CV. Here we focus on the case of MTL with convolutional networks. It has been used for face analysis tasks to improve the performance [31], [32] or to simultaneously solve them [33]–[35], for achieving better performance in object detection[22], [24], [36]–[38], instance segmentation[20], [39], salient object detection[40] and action recognition [41], for jointly learning pose estimation and action Detection[42] or for simultaneously solving up to seven different CV tasks ranging from low to high level[21].

The approaches that aim at simultaneous prediction of all tasks are the ones that are of interest as they care about having good performance for all of them, while in the ones that just want to boost the performance of the main task the performance of the extra tasks is often not even evaluated. A comprehensive method for the training of multitask network is explained in [21], even when the training involves using multiple partial datasets containing only labels for some of the tasks in each of their samples by creating a loss function that shuns the losses for the tasks with no ground-truth for a given sample and updating the parameters of a task branch only after having observed a number of samples with ground-truth for it.

2.3 Transfer learning

Work on *sequential MTL* [43]–[45] is now more commonly called *transfer learning* or more colloquially *fine-tuning* in which previously learnt domain knowledge in one task is used to initialize the training of a similar model in another task, while we will exclusively use the term MTL to refer to its parallel counterpart as previously defined. Both techniques

can be simultaneously used as is the case for many of the examples that will be shown next and are closely related. Transfer learning has been extensively studied over the last two decades in all of its forms and usually under different names - *inductive transfer*, *bias learning* or *MTL*. A survey is available in [46].

More recently, transfer learning has gained increasing popularity for CV problems after the success of [15], [18], [19], [47] to the point of being a must for most approaches based on convolutional neural networks. [47] define a way of quantifying how transferable are features from a given layer between tasks and show two causes of *negative-transfer*, i.e. when transferring features causes performance drops, are feature specificity and optimization difficulties due to the splitting of the original network that leads to fragile co-adaption between units in consecutive layers. [48] goes one step further and studies how can the performance of a generic convolutional network representation be maximized when applied to a new task, identifying the most relevant factors and giving advice on how to set them according to the characteristics of the new task, which greatly depends, again, on the similarity or relatedness between the original and the target tasks. We believe that the decisions on where to split a source task network for transfer learning and where to split a multitask network into its task-specific branches are deeply related.

2.4 Focus on practical applications

Most of the best performing convolutional neural networks have been getting more and more expensive both in terms of memory and computation. Most of them have been created for Image Classification and then applied to other tasks via transfer learning. Different approaches have been proposed to reduce computational complexity and/or memory footprint.

2.4.1 Task-specific approaches

The approaches to make the networks faster and lighter have often appeared in the task specific part of the network, with different ideas such as integrating region proposal networks into the architecture [24] for ob-

taining nearly cost-free region proposals or directly removing the need for generating proposals by discretizing the output space [49], [50] for the case of object detection or substituting the fully-connected layers for more efficient convolutional ones [51] for dense prediction problems such as semantic segmentation.

2.4.2 Lighter base networks

However, more interesting is to find solutions that tackle the large computational and memory cost of some of the base networks, those originally trained for Image Classification, which would effectively speed up and make lighter all models depending on them. Such task-agnostic approaches focus on achieving low latency or inference times and small memory footprints, requirements absolutely essential for real-time applications and specially when the deployment is in low memory devices such as the ones found in embedded systems. [12] studies the deployment of the most common Image Classification models on a GPU-enabled SoC, the NVIDIA Jetson TX1, to come to the conclusion that there is a clear trade-off between accuracy and throughput appears when comparing the models. Power consumption is surprisingly not related to model size, number of operations or accuracy. Some network architectures were already designed having their efficiency in mind, as GoogleNet[52] with the Inception module, or ResNets[14], in which by learning residual functions in the layers with reference to their inputs instead of learning unreferenced functions deeper networks can be trained while keeping a lower computational complexity. Moreover, the modular design allows to tune the complexity by having different numbers of layers and play with the speed/accuracy trade-off. Thus, some versions of ResNet [14], namely Resnet-18, Resnet-34 or ResNet-50, and GoogleNet, have a good trade-off between accuracy and speed plus a good accuracy density.

More recent work proposes even more efficient networks using the latest concepts of convolutional network design clearly having in mind the deployment in real-time applications and low-memory systems: Darknet19[53] achieves 76.5% top-1 accuracy on ImageNet Image Classification dataset [54] with a complexity of only 5.58 G-FLOPS while [55] proposes an extremely light network for real-time semantic segmentation but which can potentially be modified and fine-tuned to perform other tasks, achieving 68.4% top-1 accuracy with 1.6 G-FLOPS and a model size of

only 0.7 MB. MobileNets[56] focus on the creation of lightweight networks for mobile and embedded vision applications by introducing depth-wise separable convolutions and hyper-parameters for choosing the appropriate model size according to the application requirements in terms of speed and accuracy.

2.4.3 Distillation, compression and acceleration

Other solutions have arisen from different perspectives but have in common that take already existing networks such as the reference models AlexNet[10], VGG[13] or Resnets[14] as a starting point. In knowledge distillation [57]–[61] a lighter, faster *student* model learns to mimic a *teacher* model which can consist of an ensemble of models, ideally maintaining the same accuracy. [62] prunes the network to learn only the important connections, quantizes the weights to enforce weight sharing and encodes the weights with Huffman codes[63], a known lossless data compression algorithm, to achieve relative model size reduction by 35x to 49x and speedups of 3x to 4x. Other approaches include low-rank-factorization, structured matrices or dynamic capacity networks. It is worth mentioning that these techniques are complementary to reducing the complexity of the task-specific parts of the networks so by combining both even better performance can be achieved or the networks can run on systems with lower requirements. SqueezeNet[64] combines a custom architecture design with the compression technique from [62] to achieve a model with the accuracy of AlexNet[10] but 500x lighter. ZynqNet[65] modifies it to deploy it on a custom FPGA accelerator that optimizes the convolution and pooling operations. In [66], SqueezeNet is accelerated to run on Android devices using heterogeneous computing for distributing during runtime the workload between CPU and GPU cores.

2.5 Semantic Segmentation

Semantic segmentation aims at partitioning parts of images belonging to the same semantic class. Different approaches lead to this goal, for example, pixel-wise classification classifies each pixel rather than segmenting first and classifying the resulting segment. Semantic segmentation is

used, for example, for robot vision, autonomous driving applications and medical imaging.

Fully convolutional networks (FCN)[51] have allowed improved both accuracy performance and speed for dense prediction problems. By removing fully-connected layers or transforming them into convolutional ones, the input can take arbitrary sizes and the computation is much more efficient than patch-wise inference, by adding in-network upsampling layers pixel-wise prediction with the same image size as the input can be achieved. This approach does not use any pre- or post-processing step, which can still complement it like done in [67] using Conditional Random Fields. The fixed upsampling layers can be changed for learnable *deconvolution* layers as in [68], [69]. In ParseNet[70], global context is added to FCN by appending global context pooled from a feature map to the feature map itself, after scaling both appropriately via a normalization layer, which increases the accuracy without any post-processing.

Semantic segmentation needs to be distinguished from the more complex Instance Segmentation problem in which not only each pixel has to be classified according to its category but also according to the instance of that category that it belongs to, the object. This is important, for example, in the case of segmenting a street image with a number of pedestrians very close together. Semantic segmentation would assign a large area of the image to the category pedestrian while Instance Segmentation would separate the areas belonging to different pedestrians as different segments.

2.6 Object Detection

Object detection aims at finding in an image all instances of objects from a number of known classes, while also giving their class. The detection is usually given in the form of a class and a bounding box defining the location and scale of the detected object within the image. Object detection is many times used within higher level CV pipelines, e.g. in action detection[71].

Approaches to object detection have typically been divided between sliding windows[72] and region proposal classification approaches [73], [74]. Since the accuracy jump provided by R-CNN[73] the second method

has gained much attention and been improved to the extent that Faster R-CNN [24] was able to run at 7 frames per second (fps) and achieve 73.2 mean Average Precision (mAP) by integrating the Region Proposal part of the pipeline into the network. YOLO[50] takes another direction and sees the object detection task as a regression problem where the regressed variables are bounding boxes coordinates and class probabilities, obtained directly from a single forward pass of a convolutional network. By integrating all steps in the same network the inference times achieved are extremely fast (45 frames per second), although the accuracy lags behind the state-of-the-art. The Single Shot MultiBox Detector (SSD) [49] is similar to YOLO but instead of pulling features only from the top of the network it takes features at different levels to predict offsets and confidence scores for default bounding boxes at multiple scales and aspect ratios. It proves to be faster and more accurate than YOLO. Following the region proposal classification approach, R-FCN[75] converts the network in a fully convolutional one by making the Region Proposal Network convolutional and adding a position-sensitive score maps to be able to counteract the translation invariance introduced by the convolutional layers, which undermines the final task of object detection that intrinsically requires translation variance. It achieves a higher accuracy than previous approaches but its speed is far from that of the single-network ones. Since the start of this thesis three relevant architectures have appeared which are both faster and more accurate[38], [39], [53].

In [76], the speed versus accuracy trade-off is studied for different object detection systems following the architectures of Faster R-CNN, R-FCN and SSD using different feature extractors.

2.7 Multiple Object Tracking

MOT consists in locating all the targets of interest in a video and maintaining their identity across frames. One of the most common approaches is *tracking-by-detection*, in which objects are detected at every frame and then a patch of the image taken as a measurement that needs to be associated with a target ID. Challenges include discarding false detections and handling processes of birth and death of undetermined numbers of targets while being robust to clutter and occlusion.

In [77], a simple implementation of a MOT algorithm based on a Kalman

filter for the motion prediction and the Hungarian algorithm for data association achieves top-3 performance at 10 fps in the MOT Benchmark[78] by obtaining high quality detections with Faster R-CNN and extracting appearance features for computing the affinity value for data association with a deep convolutional feature extractor based on GoogleNet. The success of this approach highlights the importance that the detector quality has within *tracking-by-detection* schemes.

Chapter 3

Methods

In this chapter the methodology followed for obtaining the experimental results from training the different models is explained. In particular, the details of how the models are developed, the data used to train them and how it is handled, the training strategies followed, how the models are evaluated and the metrics used, and the details about the devices and software used.

3.1 Model details

In order to construct and evaluate multitask models for both semantic segmentation and object detection a choice has to be made on the base models that will be used for each task and the common convolutional trunk that will serve as the feature extractor. Ideally, one would choose these via cross-validation, however the resources and time available fall short to study which models are more adequate for MTL. The choice is made in terms of accuracy, speed, simplicity and prior success in the literature when using the most promising models.

3.1.1 Feature extractor: Residual Network, 50 layers

The common convolutional trunk will be selected based on memory and computational complexity requirements from the typical architectures used nowadays as the base for these tasks.

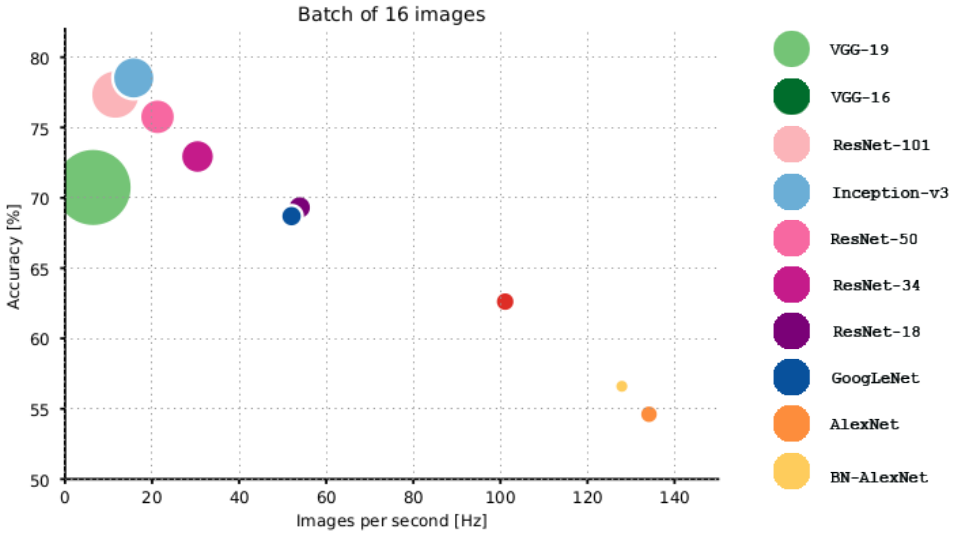


Figure 3.1: Accuracy vs. throughput, size proportional to number of operations. A non trivial linear upper bound can be seen in this scatter plot, illustrating the relationship between prediction accuracy and throughput of all examined architectures. From [12]

Figure 3.1 plots the accuracy of the most known architectures for Image Classification versus their throughput for batch size 16 on the Jetson TX1 GPU.

While no architecture offers a *free lunch* and gives high throughput and high accuracy, we expect that the heaviest networks are not suited for deployment in embedded systems due to both bigger memory footprint and larger inference times and the lightest and that fastest models generalize poorly to other tasks compared to the heaviest ones due to their lower capability of learning general enough features. While ResNet-18 or GoogleNet seem to be interesting candidates, preliminary exploration with ResNet-18 leads to bad results for semantic segmentation and [51] reported that GoogleNet underperformed compared to other architectures.

Finally, we choose ResNet-50 for its reasonable accuracy at a small enough inference time and successful prior uses in the literature[5] for the tasks at hand. The chosen common convolutional trunk for our multitask model is shown in the top of Figure 3.2. It consists of a first convolutional layer and pooling layer, and then 4 stages of residual blocks, with different number of blocks in each (3, 4, 6 and 3) for a total of 50 layers.

The difference from the original ResNet architecture meant for the Image Classification task is that the last pooling, fully-connected and softmax layers are removed.

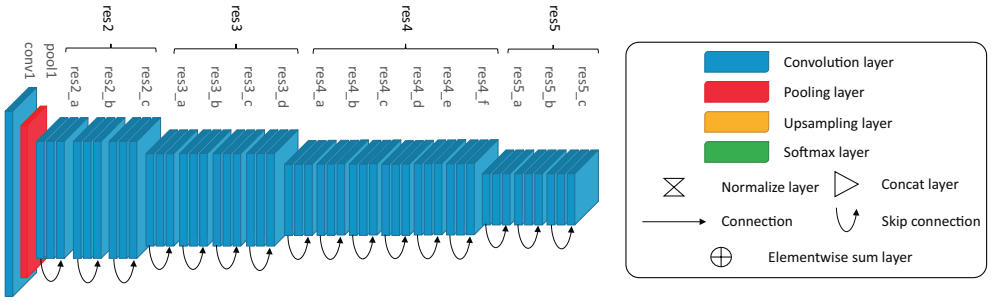
3.1.2 Semantic Segmentation: Fully Convolutional Networks

As commented in Section 2.5, most modern approaches for semantic segmentation and other dense-labeling tasks are based the Fully Convolutional Network paradigm proposed in [51]. Here, we keep it simple and stick with its basic implementation only adding skip connections to enable finer details after a fixed upsampling layer. In this way extra complexity is not added to the comparison between the single-task model and the multitask one and the inference time is also kept to a minimum. Extra post-processing steps could be added later if needed.

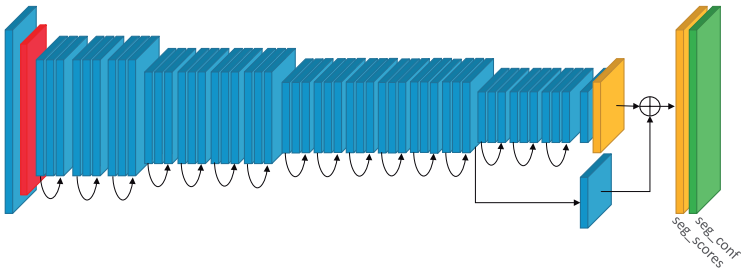
The chosen model is depicted in Figure 3.2. It consists of two 1x1 convolutional layers with as many filters as number of classes to predict. The output of this layer gives a score heatmap for each class the same size of the feature map it had as an input. One of the *score* layers pulls features from the top of the network, *res5c*, while the other from deeper inside the network, *res4f*, right before the next convolutional layer with stride 2 that effectively halves the feature map size. The second forms the skip connection and the predictions are thus made at a higher resolution although with less deep features. After the lower resolution convolutional layer, a 2x upsampling layer is attached before merging the two score heatmaps by element-wise sum to finally upsample them again to match the original size of the input and give the final segmentation map, named *seg_scores*. A *softmax* layer is finally added for training, but can also be used during deployment to get a confidence map *seg_conf* of the predicted classes.

3.1.3 Object Detection: Single Shot MultiBox Detector

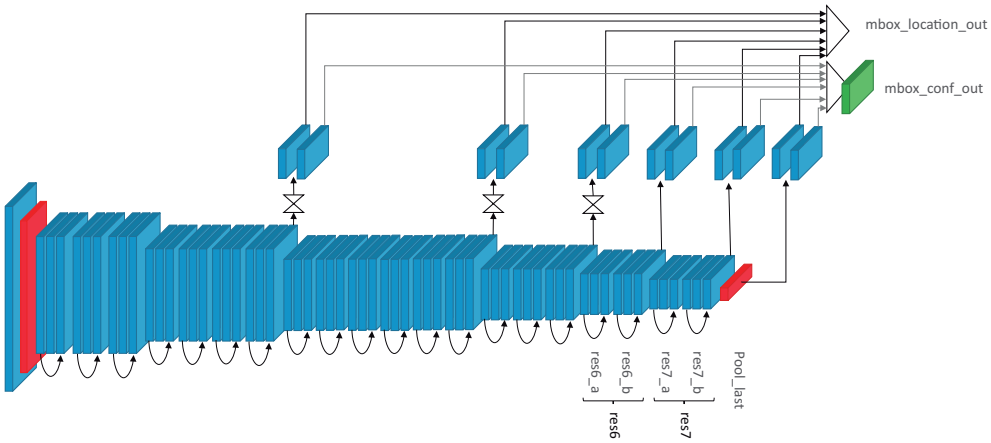
The considered object detection architectures are explained in Section 2.6. In [76], the authors identify the most interesting models as those



(a)



(b)



(c)

Figure 3.2: **(a)** Resnet-50 common base network and legend. **(b)** With FCN branch added. Crops layers not shown. Adapted from [5]. **(c)** With SSD branch added. Reshape, flatten and permute layers before not shown for a simple visualization.

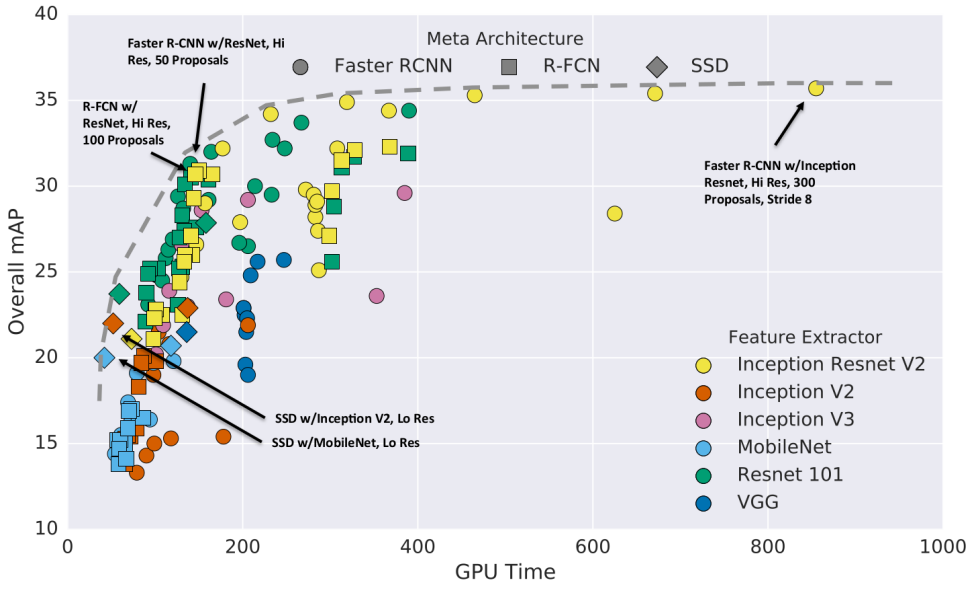


Figure 3.3: Accuracy vs. inference time for different object detector architectures and different feature extractors. From [76].

in the optimality frontier of the speed versus accuracy trade-off that we are after, see Figure 3.3. The *sweet-spot* models are defined as those in the elbow of the optimality frontier. According to this study, having fixed the feature extractor to ResNet-50, the best would have been to use R-FCN or Faster R-CNN, which together with ResNet-101, the most similar feature extractor to our choice, fall on this *sweet-spot*. However, this study was not available at the start of the thesis, so the decision was made without this knowledge. Instead, SSD is used as it promises interesting speed ups compared to the others according to the reported results, seems simple to integrate into a multitask model and is trainable end-to-end with no intermediate steps.

The SSD branch of the multitask network, shown in the bottom of Figure 3.2, is composed of extra convolutional layers added on top of the base network and a number of detection heads that are attached at different levels. The extra layers are attached after *res5c* and decrease in size progressively, allowing the prediction at multiple, larger scales, with a final pooling layer. In contrast with the original implementation, the extra layers that we attach are residual blocks as the ones used for the base network, each new stage having two blocks and 1024 outputs. We

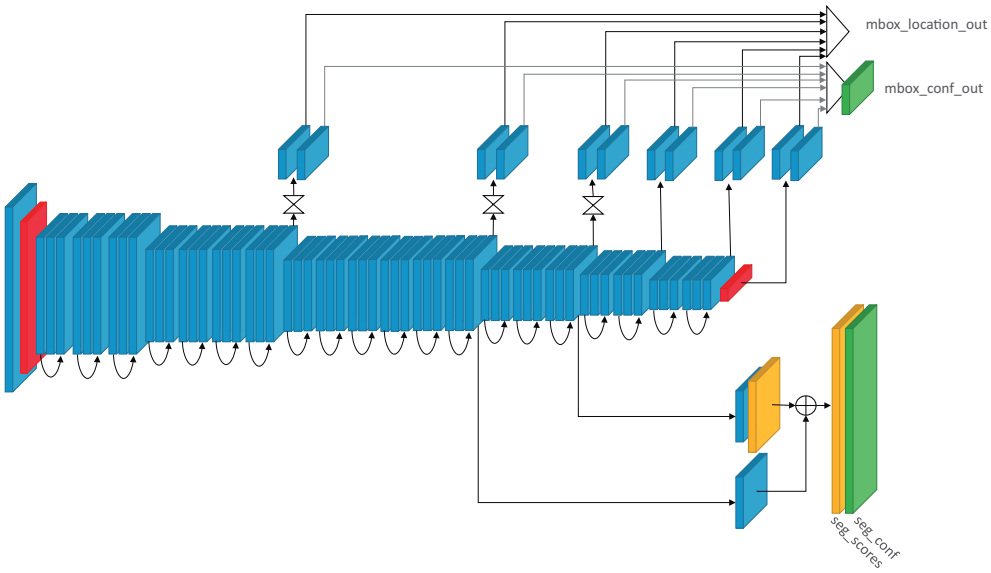


Figure 3.4: The multitask architecture based on ResNet-50 with FCN and SSD branches attached.

use Batch normalization[80] whenever the training is done with a big enough batch size, i.e. 6 or larger. The detection heads are composed of two small 3x3 convolutional layers to predict one the scores for each class and the other the offsets respect to the default box and aspect ratios for each feature map. They are attached after *res3d*, *res4f*, *res5c*, *res6b*, *res7b* and *pool_last*. Normalization layers are used for the first three stages. The outputs are named *mbox_conf_out* and *mbox_location_out* respectively.

3.1.4 Multitask model

Finally, we attach both task-specific branches to the base network to create the multitask network, as shown in Figure 3.4. In general, no change needs to be made from an architecture perspective, as we do not try to evaluate the different possibilities about which layers to share and which not to avoid *under-transfer* or *negative-transfer*. We do this only once by splitting the network after *res4f* and having the 5th stage of the base network twice, so it is not shared by both tasks. Unlike [21], we do not have any multi-scale scheme.

3.2 Input data

Having properly annotated datasets is essential for training discriminative models as the ones in which this work focuses. It is common that each task has a set of datasets with different characteristics on which the models are trained, tested and compared. Many models have elements to tackle specifically not only elements of the task they want to solve themselves but of the dataset for which they are being trained, making the dataset part of the task that is to be solved which means at the same time making the models less general, many times in order to achieve better performance scores on the dataset that is used. When a model is thought to generalize properly, it is also common to prove so by training it on multiple datasets with minimal modifications or even to see how it performs on another dataset than the one it has been trained for. However, some datasets have been growing to include multiple tasks and characteristics, in part to challenge the researchers to create more general models.

Pascal VOC [81], [82] and Microsoft COCO[83] are two good examples of such datasets and include (partial) annotations for both object detection and semantic segmentation among others through some additional annotations. Having a dataset which includes annotations for both tasks makes the work of creating a multitask model much easier, although with the method proposed in [21] this limitation is alleviated.

For aerial view, however, there is no dataset available that has annotations for both tasks. Datasets including one of the two tasks of interest include Stanford Drone Dataset[84] and UAV123[85] for object detection and tracking and Swiss [86], Okutama[5], Aerial-KITTI[87], Vaihingen[88] and TorontoCity[89] for semantic segmentation.

First, we use Pascal VOC dataset consisting of natural RGB images from hand-held cameras combining both the 2007 and 2012 versions for the validation of different training strategies. Finally, we train the best models on a dataset consisting of aerial view images that matches the target application. This dataset is a combination of Swiss and Okutama datasets for semantic segmentation and Stanford Drone Dataset (SDD) for object detection.

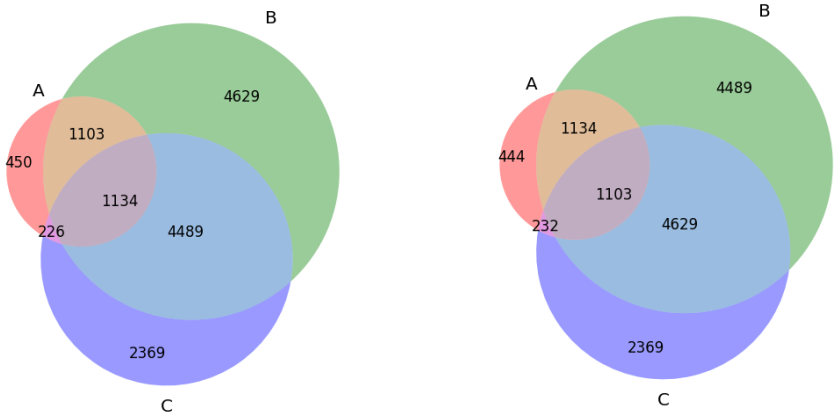
3.2.1 Pascal VOC dataset

Pascal VOC includes a total of 20 classes plus background annotated for both object detection and semantic segmentation tasks. Images were retrieved from *flickr.com* website. Object sizes range from small to larger objects and include classes like *Person*, *Car*, *Table* or *TV Monitor*. Both versions of Pascal VOC - 2007[81] and 2012[82] - are combined to get a larger dataset. In order to make it even larger, the extended annotation set for semantic segmentation from the Semantic Boundaries Dataset (SBD)[90] is also used. As pointed out by the authors, some data points in their training set are part of the validation set of the original VOC2012, which has to be taken into account for a correct evaluation of the models.

Ground truth annotations for both tasks are required so only those data points with both annotations are used. The intersection between the sets with available annotations for object detection from VOC2007 and VOC2012 and the sets with available annotations for semantic segmentation from VOC2007, VOC2012 and SBD is used to create three sets. The original training and validation sets for VOC2012 detection task are respected. Figure 3.5 shows the different sets and how they intersect for both training and validation/test sets. The final size for the training set is 5489 images, considerably smaller than if the limitation of having both annotations was not there as there would be 8218 images with object annotations and 10582 images with segmentation annotations[67]. As Pascal VOC has a private test set, the validation set is randomly split in two in order to have a test set that can be used to report final results. We get two equally sized validation and test sets of 2982 images each. Some example images with both annotations can be found in Figure 3.6.

3.2.2 Aerial View datasets

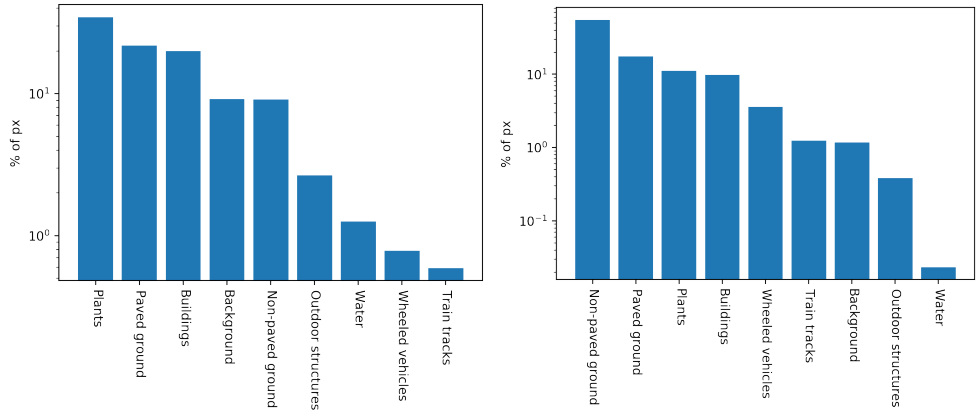
As already mentioned, there is no dataset that has both annotations for the tasks of interest from an aerial view perspective. Luckily, Swiss and Okutama datasets have ground truth annotations for semantic segmentation and SDD for object detection. By combining them, a multitask model can be trained using a trick inspired by [21], interleaving samples of each dataset inside each mini-batch during training.



(a) (b)

Figure 3.5: **(a)** Train set. **(b)** Validation set. **A**(red) is the set of images with semantic segmentation available for VOC2007 and VOC2012. **B**(green) is the set of images with semantic segmentation annotation from SBD. **C**(blue) is the set of images with object detection annotations for VOC2007 and VOC2012. The intersection between C and either A or B or both forms the desired set with both annotations for object detection and semantic segmentation.

Figure 3.6: Pascal VOC dataset example images with bounding box ground-truth annotations for object detection and overlaid semantic segmentation ground-truth annotation.



(a) (b)

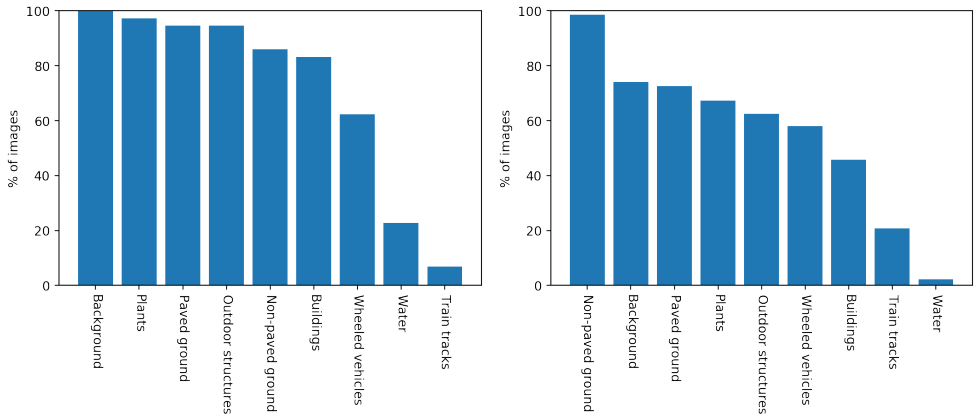
Figure 3.7: Percentage of pixels per class in **(a)** Okutama dataset and **(b)** Swiss dataset.

3.2.2.1 Swiss and Okutama datasets

Swiss and Okutama datasets are two small size datasets consisting of top-down images taken from a UAV in Japan and Switzerland at a height of either 50 or 90m above the ground. They are annotated for semantic segmentation with 10 classes including *Water*, *Plants*, *Train tracks* or *Buildings*. Each image has 4k resolution and there are 100 and 92 images for each dataset respectively. However, in order to handle it more easily, we crop the images in a 2x2 fashion to get 4 times the number of images with 1080p resolution. As any semantic segmentation dataset, intrinsic imbalance in the number of pixels belonging to each class is present, as can be seen in Figure 3.7.

Due to these facts, it is important that at least the sets in which the dataset is split are balanced in terms of number of images having at least some pixels belonging to each class. Figure 3.8 shows the percentage of images having at least one pixel of each class.

Thus, dataset balancing needs to be done when splitting into training, validation and testing sets so each set has a good representation of every class. Images are assigned to a set so that all classes appear in about the desired percentage of images in each set. Swiss dataset is balanced in a similar fashion. Figure 3.9 shows how the images per class percentage looks like after balancing for each set of the datasets. The target was



(a) (b)

Figure 3.8: Percentage of images containing each class in **(a)** Okutama dataset and **(b)** Swiss dataset.

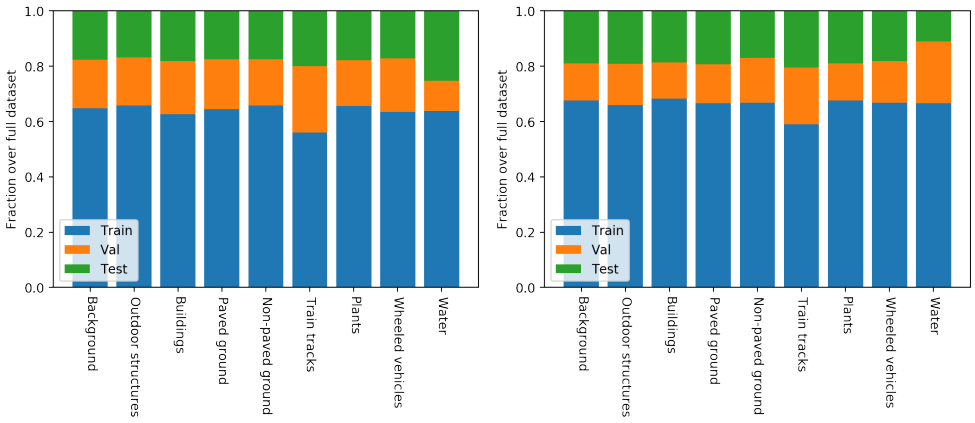
to split the dataset in 60% of images for training, and 20% for both validation and test.

Finally, some sample images with semantic segmentation annotation can be found in Figure 3.10.

3.2.2.2 Stanford Drone Dataset

SDD is a dataset compiled with the goal of enabling the development of algorithms for both trajectory forecasting and target tracking. However, given the fact that the per-frame annotations are bounding boxes of a number of target classes, it can also be used for object detection. The classes are *Bicyclist*, *Pedestrian*, *Skateboarder*, *Cart*, *Car* and *Bus*. It is collected from a hovering UAV with a camera facing top-down in 8 different locations of a university campus environment and includes almost a million frames and more than 19k unique targets or object instances. Although they were originally collected at 4k resolution, after distortion correction and stabilization the videos have a resolution close to 1080p.

For practical reasons and the fact that consecutive frames are really similar, only each 20th frame is used so that after randomly splitting the dataset a total of 31565 frames are used for training and 10522 each for both validation and test sets. Example frames can be found in Figure 3.11.



(a) (b)

Figure 3.9: Fraction of images per class for each set over full dataset. **(a)** Okutama. **(b)** Swiss.

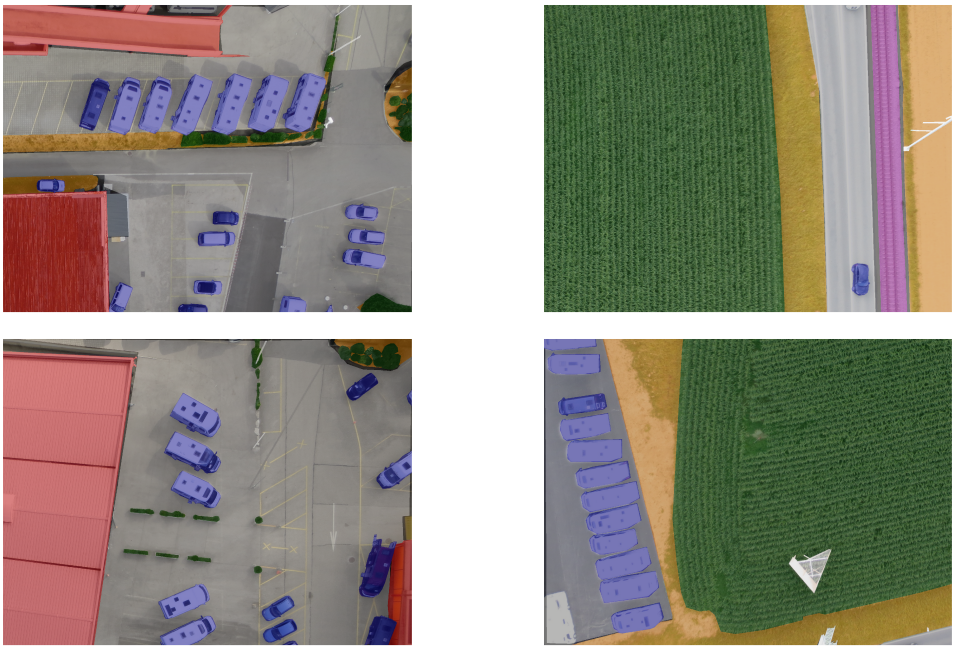
3.2.3 Data augmentation

In order to fight overfitting, we apply different data augmentation techniques in an online fashion during training. We use either random crops or resizing when using a batch size bigger than one so that all images in the batch have the same size. We apply color distortion by randomly altering channel values in HSV color space. We also randomly flip images, horizontally for Pascal VOC dataset and both horizontally and vertically for the aerial view datasets.

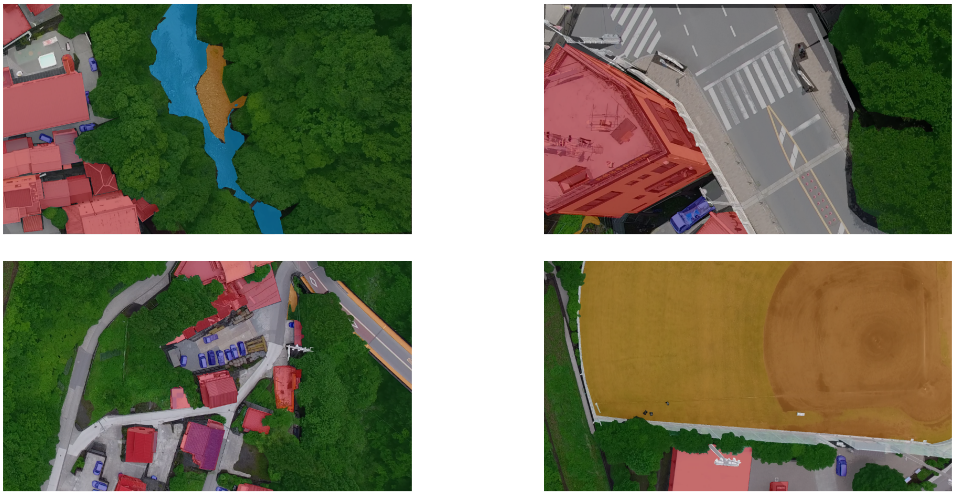
3.2.4 Feeding the networks for MTL

For Pascal VOC dataset, the network can be fed with either only one image, resized or cropped or with the original size, or with a batch of multiple images, in which case all of them must have the same size so they need to be resized or cropped. Annotation for both tasks is also available for each data point so the loss can be computed for training without problems.

For the aerial view datasets, as each dataset has only annotations for one of the tasks, we interleave one image from each dataset to form a batch in which there are samples from both datasets with annotations for both tasks. In order for the annotations that are missing not to affect the loss



(a)



(b)

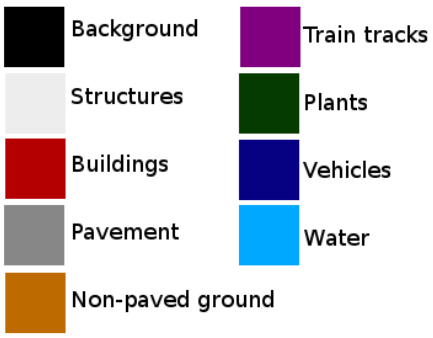


Figure 3.10: **(a)** Swiss dataset example images with overlaid semantic segmentation ground-truth annotations. **(b)** Okutama dataset example images with ground-truth.

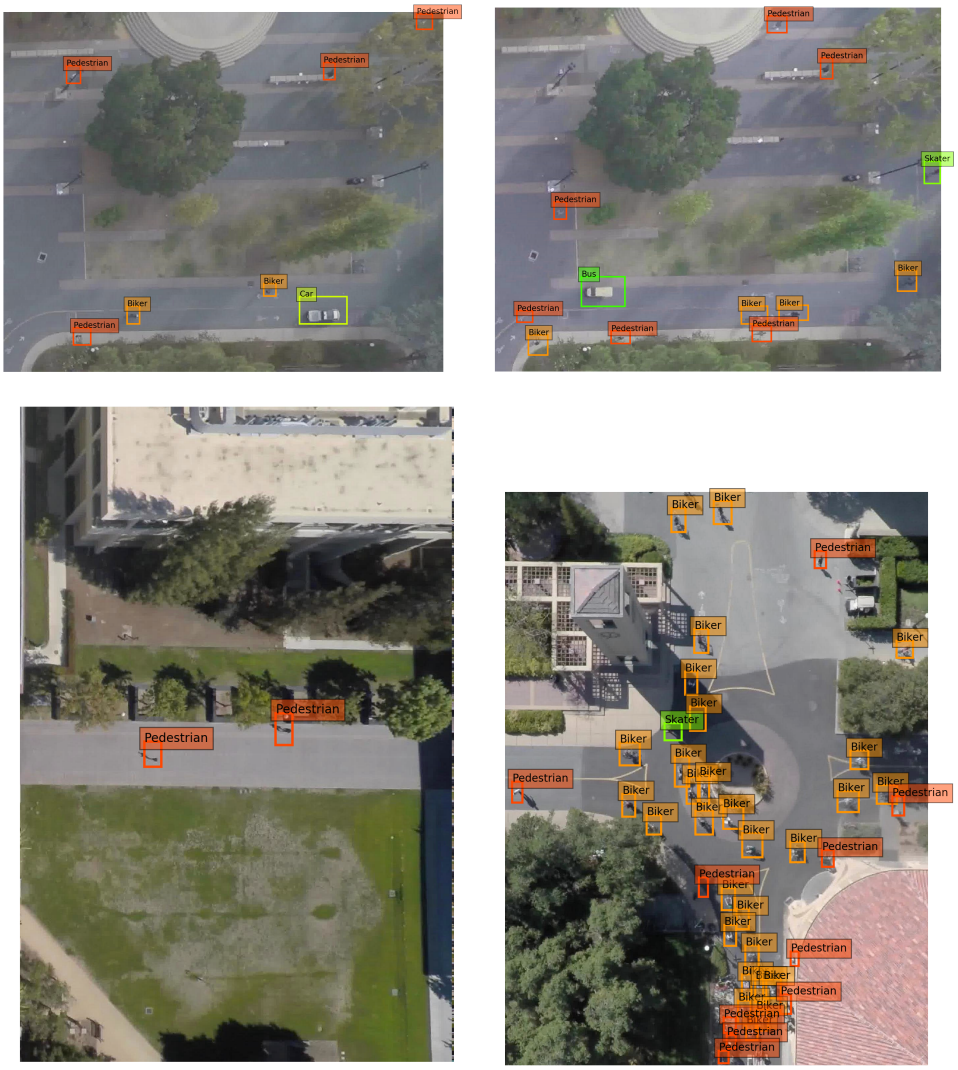


Figure 3.11: SDD dataset example frames with bounding box ground-truth annotations for object detection.

of the corresponding task, these take a known value that makes them be ignored while computing the loss. In this way the corresponding losses for that image take zero value and the gradients are also zero, meaning there is no back propagation into the network for the loss corresponding to the missing ground-truth annotation. As for each batch there are both cases, the weight updates are computed with the average of both gradients in the common layers and only with the ones for the task for which there was annotation for the task-specific layers. This is similar to the strategy described in [21] but less flexible as the updates for the common trunk are always twice more often than for the task-specific branches, which always happen with the same frequency.

3.3 Training strategies

Different training strategies are evaluated on the validation set to determine the best one, both for the single-task and multitask models. The strategies followed for the single-task models in previous works is described but it is expected that it is not necessarily the best for the multitask models, starting from the fact that they greatly differ for both tasks.

3.3.1 Single-task models

Here we explain how the models were originally trained and the details that are common to all models. The exact details used for each of the different models trained are explained in Section 4.1.

3.3.1.1 FCN model

In the original paper, FCN models are trained via Stochastic Gradient Descent (SGD) in an online fashion (batch size of 1 image) with images of variable sizes, large momentum of 0.99 (vs. the standard 0.9), fixed learning rate of 10^{-10}[51]. The loss function is the classical multinomial logistic or cross-entropy loss for multi-class, single label problems but with no normalization over the number of pixels. In [70], better results are achieved by using standard momentum, higher base learning rate of 10^{-9}

and polynomial decay learning rate policy ($base\_lr * (1 - \frac{iter}{max\_iter})^{power}$) with power parameter set to 0.9 and 80000 iterations. No data augmentation is used but [5] reports some performance gains by doing random flips and crops on Swiss dataset, most likely due to its small size. The weights of the base network are initialized with those of a model pre-trained on ImageNet[54] and the new layers of the fully convolutional branch with MSRA[91] initialization.

We try using both the original and improved training strategies, introducing normalized loss over the number of pixels, resizing of images and mini-batch training. We also adapt the number of iterations for which we train to the dataset size, which for our subset of Pascal VOC dataset is roughly half, and if we apply or not data augmentation techniques, in which case we also increase the number of iterations.

3.3.1.2 SSD model

The training of SSD models is significantly more complex. To start with, ground truth annotation has to be assigned to a specific output of the detector for then computing the loss function and applying back propagation. This matching is done by pairing the default and ground truth annotation bounding boxes with highest jaccard overlap ($\frac{Area_{intersection}}{Area_{union}}$) but also those with a jaccard overlap higher than 0.5 in order to ease the learning as overlapping bounding boxes can all have high scores. With the matched default boxes and annotations the loss is computed as the weighted sum of a localisation loss, computed as a Smooth L1 loss between the bounding box parameters, and a confidence loss, computed as a cross-entropy loss. The number, scale and aspect ratio of the default bounding boxes is a parameter that can be tuned according to the characteristics of the dataset, the default parameters are kept for all experiments. Hard negative mining is used to balance the number of positive and negative training examples. Extensive data augmentation is also used and greatly affects the performance of the model. It includes sampling patches of multiple sizes and aspect ratios according to the jaccard overlap with the ground truth boxes or randomly, random flips and photo-metric distortions. In the original paper, SSD is only trained for Pascal VOC with VGGNet[13] as a base network but in [79] SSD is trained with ResNet-50 instead. The VGG model is trained via SGD with a base learning rate of 10^{-3}, batch size 32 and standard values for

the rest of parameters. The learning rate is divided by 10 after 40k and 50k iterations until reaching 60k. In the second case, the first *conv1* and *res2* blocks are frozen but batch normalization statistics are computed again for all other layers. The learning is divided by 10 after 40k iterations. As done for the FCN model, the initialization of the weights of the base network takes values pre-trained on ImageNet and the rest use MSRA initialization.

We apply similar flipping and color distortion techniques but use a simpler patch sampling strategy with only random crops. We use the same base learning rate of 10^{-3} but instead of using a *step* policy we use the same polynomial decay learning rate policy that we use for FCN and train for different numbers of iterations depending on dataset and batch size.

3.3.2 Multitask models

For the multitask models, a common training strategy needs to be followed. Given the fact that the strategies reported as best greatly differ between both tasks, exploration over the hyper-parameters and data augmentation techniques is needed. The hyper-parameters that we explore are the base learning rate and policy, using batch normalization or not and the batch size - which, for practical reasons, cannot reach 32 as in SSD due to the fact that the Caffe version used does not support multi-GPU training with Python layers. Also, we evaluate the effect that the data augmentation techniques have on the final accuracy. Unlike [21], where the weights are initialized with those of networks pre-trained on Microsoft COCO [83] for both tasks, we initialize the base network with ImageNet weights and the task-specific branches with MSRA initialization.

3.4 Evaluation

We compre the resulting multitask models to their single-task counterparts in terms of accuracy, inference time, memory usage and model size. In terms of resources, the results are compared to the naïve approach of running one model for each task in parallel but on the same device. We

test the models both on a desktop GPU and an embedded device with GPU.

The accuracy metrics used for evaluating each task are explained next. Per-class accuracy scores are also a very interesting metric to take into account in the evaluation of models, particularly when different classes appear in different scales and number of instances in the datasets.

3.4.1 Metrics

3.4.1.1 Semantic Segmentation metrics

Semantic segmentation is typically evaluated in terms of mean Intersection-over-Union (mIoU), the per-class average of the Intersection-over-Union (IoU):

$$mIoU = \sum_{i=0}^{N_c} IoU(i)$$

$$IoU(i) = \frac{TP(i)}{TP(i) + FP(i) + FN(i)}$$

Where N_c is the number of classes, TP is the number of pixels correctly classified as belonging to class i, FP is the number of pixels wrongly classified as class i and FN the number of pixels wrongly not classified as class i. The frequency-weighted IoU takes into account class imbalance in the dataset by weighting the sum for computing the average over the different classes with the frequency with which the class appears in the dataset.

$$fwIoU = \sum_{i=0}^{N_c} freq(i) \cdot IoU(i)$$

As the model has to perform properly for all classes even when they are not balanced in the dataset, we believe mIoU is a better summary of its performance and we use it instead.

The overall (OP) or per-class mean (PC) pixel accuracies are often used as well, although they are more biased towards classes with

a much greater presence in imbalanced datasets[92], such as Pascal VOC (with around 70% of all pixels belonging to background class) or Okutama/Swiss datasets, as explained previously.

$$OP = \frac{\sum_{i=0}^{N_c} TP(i)}{\sum_{i=0}^{N_c} TP(i) + FP(i)}$$

$$PC = \frac{1}{N_c} \sum_{i=0}^{N_c} \frac{TP(i)}{TP(i) + FP(i)}$$

3.4.1.2 Object Detection metrics

For the object detection task, the common evaluation metric is the Average Precision (AP), used for example for Pascal VOC challenge since 2007[93] in its 11-point interpolated version:

$$AP(i) = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{interp,i}(r)$$

Where $p_{interp}(r)$ is an interpolated precision defined as the maximum precision over all recalls greater than r. This precision is defined as:

$$p_{interp,i}(r) = \max_{\hat{r}:\hat{r} \geq r} p_i(\hat{r}_i)$$

$p_i(\hat{r}_i)$ is the precision at recall $\hat{r}_i$, which can take different values depending on the set of detections taken from the list of detection outputs for class i ranked according to their confidence scores. The recall at K or $r_i@K$ for class i is defined as taking the top-K detections from that list. The first K that gives the given recall is used. Thus, $p_i(\hat{r}_i)$ is the precision using the same set of top-K detections that gives recall $\hat{r}_i$.

$$r_i@K = \frac{TP_i(K)}{TP_i(K) + FN_i(K)}$$

$$p_i(\hat{r}_i) = p_i@K_{K:r_i@K=\hat{r}_i} = \frac{TP_i(K)}{TP_i(K) + FP_i(K)}$$

TP_i is the number of objects correctly detected and classified as belonging to class i, the true positives; FP_i is the number of objects wrongly detected and classified as class i, the false positives; and FN_i the number of objects wrongly not detected and classified as class i, the false negatives. The matching strategy between detection and annotation bounding boxes is the same as explained in Section 3.1.3, measuring the bounding box overlap and assigning as TP those with jaccard overlap or Intersection-over-Union higher than 0.5. Multiple detections of the same object in an image are considered false detections. The mean Average Precision is the average of the AP of each class over N_c classes:

$$mAP = \sum_{i=0}^{N_c} AP(i)$$

3.4.2 Test devices

The accuracy of each model should be independent of the system used. The memory usage and specially the inference times can be different however. In order to see how both vary between desktop-level systems and embedded ones we do this evaluation in two different systems, one belonging to each type.

First, we use the system used for training also for evaluating the performance of the models as our desktop-level system. The model runs on a NVIDIA GTX Titan X GPU with CUDA 7.5 and cuDNN 5 support. It has 3072 CUDA cores that run at 1GHz for a maximum processing power for single precision operations of 6144 GFLOPS. It has 12 GB of memory and a memory bandwidth of 336 GB/s.

Second, we use a Jetson TX1 development kit as our embedded system, which incorporates a NVIDIA Tegra X1 GPU and has CUDA 8 and cuDNN 5.1 support. This GPU has 256 CUDA cores that run at 1 GHz for a maximum processing power of 512 GFLOPS with single-precision floating-point format. Its memory is shared between the system and the GPU and has a total of 4GB and a memory bandwidth of 25.6 GB/s.

In numbers, the systems are clearly different in terms of resources as the desktop one has roughly 12 times more processing power, 3 times the amount of memory (which is dedicated) and 13 times more memory

bandwidth.

3.5 Software and frameworks

All experiments are performed on Ubuntu Linux 14.04 using the *ssd* fork of Caffe framework from [49] with an NVIDIA GTX Titan X GPU and CUDA 7.5 and cuDNN 5 support. We do loading and data augmentation operations via a custom Python (2.7) layer which uses NumPy, PIL, OpenCV and SciPy libraries for image processing and tensor handling.

We use Jupyter notebooks for tracking training progress and test results parsed from the log files and to analyze and manage the different datasets. In order to test the network during its training we adapted another custom Python layer from the test code of [51] for computing the metrics related to the semantic segmentation task. Such a layer is already available for obtaining the object detection metrics in the Caffe fork used. For testing the performance of the models on an embedded system, we use an NVIDIA Jetson TX1 which has the same software but CUDA 8 and cuDNN 5.1.

Chapter 4

Results

This chapter presents and analyses the results of the different models for both Pascal VOC and Aerial View datasets, in terms of accuracy, inference times, memory footprint and model size; and discusses the challenges found during the training process.

4.1 Pascal VOC

Evaluating the proposed method on Pascal VOC allows us to validate our ideas and strategies on a smaller dataset which is easier to handle and faster to train for. As we have to find a common strategy for both tasks on Pascal VOC, we start by evaluating on each task the effects of using features of the strategy used for the other task. All models are trained via SGD with momentum unless specified.

First, we decide to investigate which batch size gives better results for the training of the SSD model for object detection. Here, we test batch sizes 6, 12 and 24, although for the latter via gradient accumulation due to memory limitations so the batch normalization parameters are computed with half the batch. The total number of iterations is 30000 for the batch size 6 model, and divided proportionally when augmenting the batch size. When trying with the larger base learning rate used in [79], however, the gradients explode so the original base learning rate is used instead. We did not use data augmentation here except for random horizontal flips.

Table 4.1: Single-task SSD model results compared to those in the literature.

	Batch size	mAP (%)
VGGNet[50]	32	74.3
ResNet-50[88]	32	70.4
Ours	6	**51.3**
	12	48.2
	24	45.7

Table 4.1 shows the results compared to the reported values of previous work. Unlike our models, the reference ones use VOC2007 and VOC2012 `trainval` for training and test on VOC2007 `test`, and, most importantly, use data augmentation with color distortion and an extensive sampling strategy to generate more samples at different scales, both small and large, via random crops and random expansions[49].

The batch size that gives best results is 6 but these are still far from the reference models, a performance gap which we attribute to the extensive data augmentation and patch sampling used in them.

Second, we try using the modified training strategy from [70] for semantic segmentation and achieve the same result than using the strategy of the original paper. As the loss used in FCN is not normalized but the one in SDD is, a normalized loss is mandatory for the multitask training in order to have losses of similar magnitudes. When doing so the learning rate has to be incremented.

For the object detection task, SSD uses mini-batch training, which means that all images in the batch must have the same size. We evaluate the effect of warping the images to train with equally sized images, also the effect of using mini-batch training instead than online. As the dataset used is roughly half the size of the one used in other works[67], [70], we train for half the number of iterations, 40000. Table 4.2 summarises the different changes introduced to the training of FCN models evaluated on the validation set of Pascal VOC. The result reported in [5] uses also the images that do not have object detection annotations.

The training strategy for ParseNet achieves the same result with less iterations as after 30000 the model accuracy does not improve. However, all other modifications decrease the final accuracy of the model. The

Table 4.2: Mean Intersection-over-Union (mIoU) and per-class pixel accuracy (PC) results for the different single-task semantic segmentation models on the validation set of Pascal VOC after the different modifications introduced to the training strategy.

	Laurmaa[5]	FCN-ResNet-50				
ParseNet training			✓	✓	✓	✓
Normalized loss				✓		✓
Warping, 300x300					✓	✓
Mini-batch, $bs = 6$						✓
mIoU (%)	62.5	**58.9**	58.8	54.6	57.0	56.4
PC (%)	-	68.3	**68.7**	65.6	66.5	66.8

Table 4.3: Multitask model results for different batch sizes and including data augmentation on our validation set for Pascal VOC dataset.

	Multitask-ResNet-50					
Batch size	6	12	24	6	6	6
Random crops				✓		✓
Color distortion					✓	✓
mIoU (%)	54.4	53.3	53.8	54.2	**55.2**	54.5
PC (%)	65.4	62.9	64.5	**67.1**	66.8	67.0
mAP (%)	51.8	47.3	47.8	51.8	**52.6**	50.9

most severe seems to be normalizing the loss function. For it different learning rates are tested but 10^{-4} proves to be the best choice. In the mini-batch training case, using batch normalization allows a larger base learning rate of 10^{-3}.

Finally, with the FCN training strategy adapted to match that of SSD, we train our multitask model. Again, we evaluate for batch sizes 6, 12 and 24 and later introduce more data augmentation: color distortion and random sampling of patches via random crops instead of warping. We train all models with *poly* learning rate policy, base learning rate 10^{-3} and 30000 iterations. Table 4.3 shows the results for the different models.

Batch size 6 gives the best results among the evaluated ones, improving 0.5% mAP for object detection over the single-task model trained in similar conditions but with a 2% drop in mIoU for semantic segmentation. We see that random crops without resizing do not improve accuracy but

color distortion does by 0.8% mAP for object detection and 0.7% mIoU, 1.2% PC for semantic segmentation. Strangely, when combining both techniques the performance drops, we hypothesize that it needs to be trained for more iterations due to a greater variability in the training samples. Duplicating the last stage of the base network so that each task can learn it separately does not lead to good results for the object detection task.

Some example detection and segmentation outputs given by the best model can be found in Figure 4.1.

The results for semantic segmentation seem to be better when the object is clearly salient in the image, otherwise there is more confusion and patches of different classes appear on a same object. For object detection there is a significant number of false negatives and some cases with double detections or two bounding boxes covering different parts of a same object. In many cases, the SSD model does not seem to agree with the semantic segmentation output, such as in the correct segmentation of the *Sheep* that is wrongly detected as both a *Dog* and a *Horse* or the *Cow* that is also classified as a *Horse*.

4.2 Aerial view

For the aerial view datasets, we first train single-task models without the limitations of the MTL training. For Okutama/Swiss datasets, we train an FCN model based on ResNet-50 for 80000 iterations on the training and validation sets in an online manner but with gradient accumulation of 8 samples, no loss normalization, standard 0.9 momentum and base learning of 10^{-10}. For the Stanford Drone Dataset, we train an SSD model with batch size 32, resized images to 300x300 and all the original parameters and data augmentation strategies for 90000 iterations.

Finally, we evaluate our multitask model with the new datasets with both random crops and color distortions, batch size 12 for the shared layers and 6 for the task-specific ones, base learning rate 10^{-3}, learning rate policy *poly* and crop size of 300. Table 4.4 summarizes the results of the single-task and multitask models trained.

As the SDD dataset includes mostly small scale objects, the achieved mAP is not as high as the reference model as we are not using the exten-

Figure 4.1: Example detection and segmentation outputs from our best multitask model for images of Pascal VOC dataset test set.

Table 4.4: Object detection and semantic segmentation results for aerial view dataset baseline and multitask models.

	Image size	mIoU (%)	PC (%)	mAP (%)
FCN	Variable	70.9	79.8	-
SSD	500	-	-	54.3
Multitask	300	64.1	72.0	17.7
	500	**65.3**	**73.6**	**28.2**

sive patch sampling techniques from [49] in our multitask models. For the 300x300 model, the AP of the most common classes *Bicyclist* and *Pedestrian* reaches 47.0% and 44.0% respectively, 18.7% for *Cars* and all others below. We hypothesize that the imbalance between classes in the dataset affects negatively the result. In the reference model, this happens as well in the beginning but training for more iterations the unbalanced classes are finally learnt properly and the final per-class AP are more similar. In addition, with the multitask model the FCN branch learns at a much faster pace and starts overfitting soon after, most likely affecting the ability of the model to continue learning for the SSD branch. A larger image size positively affects the performance of the model. For semantic segmentation, the best performing classes are *Plants* and *Non-paved ground* around 90% IoU and are also the most frequent, followed by *Wheeled vehicles*, *Buildings* and *Paved ground* all above 75% IoU. The other less frequent classes are all below 30%.

Some example detection and segmentation frames using the multitask model can be found in Figure 4.2 for Swiss and Okutama datasets and in Figure 4.3 for SDD.

The results look better when the image quality is good. For the Swiss and Okutama images, which did not have ground-truth for object detection, the detection of *Cars* is surprisingly good and even the few *Pedestrians* that appear are also recognized. For SDD, however, the semantic segmentation output looks coarser and with more confusion between classes than in Okutama and Swiss. The blurriness in all SDD images, however, makes it a difficult task even for the human eye.

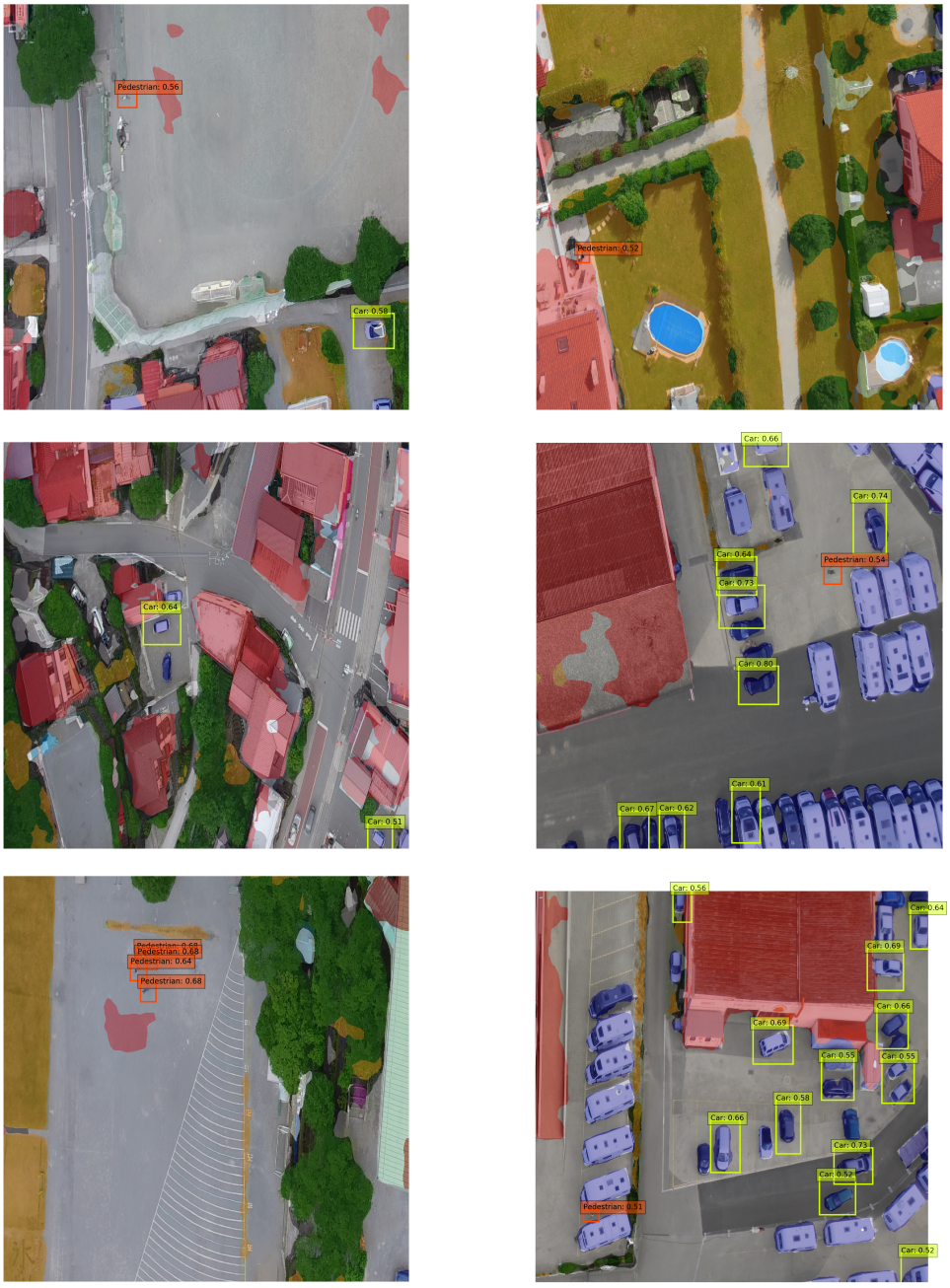


Figure 4.2: Example detection and segmentation outputs from our best multitask model for images of Okutama(left) and Swiss(right) test sets.

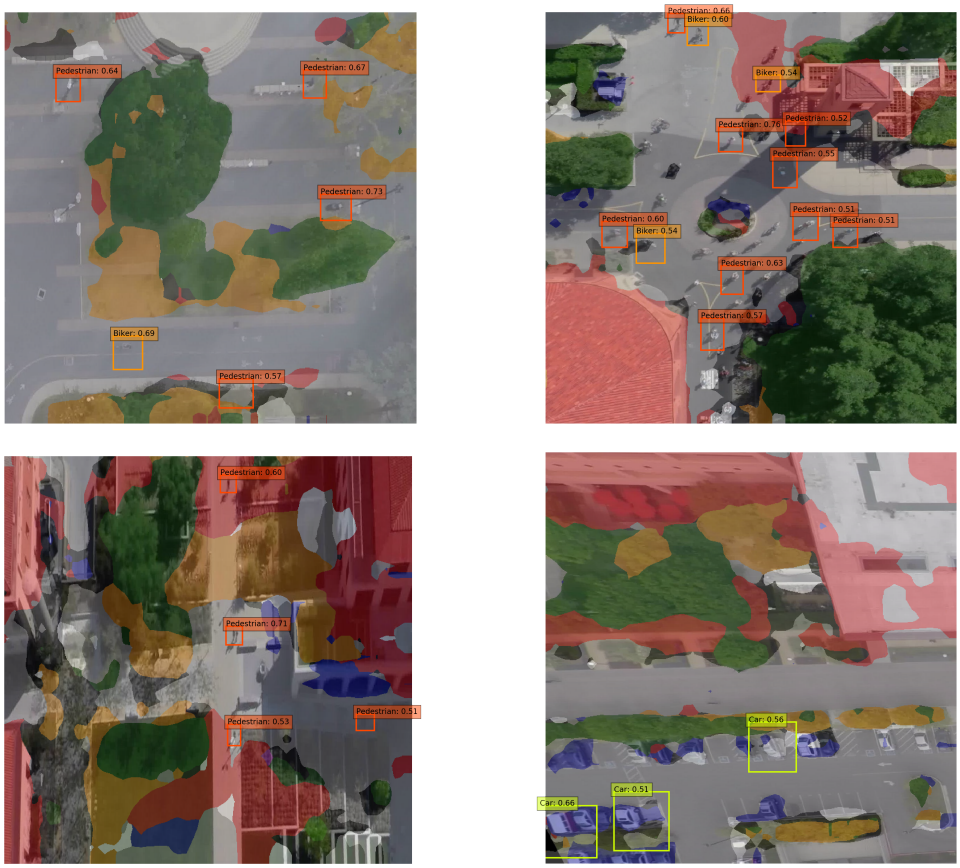


Figure 4.3: Example detection and segmentation outputs from our best multitask model outputs for images of SDD test set.

4.3 Inference time, memory usage and model sizes

As the goal of this work is to develop models that are not only accurate but usable in embedded applications the resources used by it are evaluated. Both memory usage, which summarizes both number of parameters and memory taken by the intermediate layers, and inference time, which relates to the number of operations but is also dependent on the memory bandwidth and architecture of the GPU. We choose these two values because are the ones that best define the usability of the model. The inverse of the inference time will indicate the throughput, the number of frames per second that can be processed. The total memory usage indicates if the model can fit into the memory of the GPU. For simplicity, we evaluate images with size 300x300 pixels and use a batch of a single image. This values are close to the limit of what can be deployed in an embedded system such as the Jetson TX1, which as analysed previously has a total of 4GB of shared memory. While doing the experiments, the operating system was using already 1575MB. For other systems with dedicated and/or more memory available the batch size can be increased to take advantage of matrix-matrix product acceleration in GPUs and increase the throughput at the cost of introducing some extra latency. With higher image sizes both the inference time and memory usage go up. Table 4.5 summarises the inference times, memory used and model sizes for the different cases.

The advantage in terms of resource sharing of multitask models is clear. Deploying the two single-task models simultaneously without sharing is 1.5x slower, uses 1.6x more memory and the model size is 1.6x larger than the multitask model for the Pascal VOC models deployed on the GTX Titan X. For the aerial view models, it is 1.6x slower, uses 1.6x more memory and the model size is 1.7x larger. We cannot test the two-single task models simultaneously on the Jetson TX1 as the memory gets full, so it is not even an option to deploy them.

Table 4.5: Inference times, memory usage and model weights size when running the base, single-task, multitask models and both single-task models simultaneously on the Jetson TX1 and GTX Titan X. All values are averaged over 50 forward iterations using a single image per batch.

Jetson TX1		**Inf. time(ms)**	**Mem.(MB)**	**Weights(MB)**
Base network		175	1193	95
Pascal VOC	FCN	202	1276	95
	SSD	224	1702	158
	MT	249	1780	158
Aerial View	FCN	185	1225	95
	SSD	210	1616	140
	MT	219	1644	140

GTX Titan X		**Inf. time(ms)**	**Mem.(MB)**	**Weights(MB)**
Base network		19	1203	95
Pascal VOC	FCN	31	1276	95
	SSD	29	1594	158
	MT	39	1668	158
	Both	58	2641	253
Aerial View	FCN	24	1233	95
	SSD	27	1525	140
	MT	30	1552	140
	Both	49	2511	235

4.4 MTL effects on the training process

Although multitask models offer a clear advantage in the use of resources and speed, the process of training them is more cumbersome and performance obtained in the single-task models following the optimal training strategies is difficult to achieve. This is due to a number of reasons related to the use of MTL that are explained next.

First, the choices when stitching together different models become a hyper-parameter that needs to be chosen, i.e. the layers that are shared from the base network and the ones that are used by a specific task only etc. We have opted for choosing it the natural way and we share all the base network but it definitely takes an important role in the final model performance, memory used and inference time.

Second, the meta-architecture used for each task (SSD, Faster R-CNN, etc.) or the base network are another choice that will affect the training process. Both [21] and [22] used VGGNet as the base network, Faster R-CNN for object detection and DeepLab[67] for semantic segmentation with success. Faster R-CNN only uses features from the top of the network, so the path that the gradients follow when they back-propagate into the network is always the same and independent of the dataset characteristics themselves. In contrast, as SSD pulls features from different levels of the network for the different prior bounding box sizes, the gradients get into the network at multiple locations affecting the shared layers in different manners depending finally on the statistics of the dataset. For Pascal VOC, where there are images of all sizes, this might not be that important, for SDD, where most objects appear small, most will take prior boxes that pull features from the first layers, so the gradients that they will produce will affect only those below. We conclude that the way the different approaches used to solve each task interact and affect the learning process of the shared layers and the branches of other tasks in the network is something that has to be taken into consideration in the design phase.

Third, different tasks peak in performance at different times. We see this for the VOC models although it does not seem to affect the final performance. However, for the aerial view models the semantic segmentation task peaks way before the object detection task, most likely due to the large difference in dataset sizes, 504 training images for Okutama/Swiss

dataset versus 31565 training images for SDD. [1] explains that not only this can lead to obtaining no benefit from the MTL approach, but the opposite, it can undermine the performance of the slower learning task if the fastest starts to overfit as it can make the shared layers overfit as well. To overcome this, it proposes using per-task learning rates, which are fine-tuned in an iterative, trial and error way; to make all tasks reach best performance nearly simultaneously. We did not try this solution in this thesis and we leave it for future work. We tried using different optimization methods, such as AdaDelta[94] or Adam[95] that adapt different learning rates for the different parameters, but they did not lead to good results.

Finally, the optimal strategies for training the models typically differ between tasks and datasets as they have been empirically been found to work best after fine-tuning the hyper-parameters for a particular task. This forces to compromise between the different training strategies to have something in between which works well for all tasks. In [22] this was considered but the second task was typically used to regularize the first and increase its accuracy so the goal was different as the performance only had to be good for the main task. UberNet[21] takes this fact into consideration with the same purpose than here and proposes asynchronous updates with different batch sizes for the different task branches and shared base network. In the end, this means that for a MTL model the parameters that were once carefully chosen for each task need to be tuned again for a good, common performance on all tasks.

Chapter 5

Conclusion

In this thesis, we proposed MTL as a way to speed up Deep Learning models for applications in which multiple tasks need to be solved simultaneously, which is particularly useful in real-time, embedded applications with limited resources. We replaced the naïve approach of deploying a model for each task simultaneously by a single jointly trained, multitask model which takes advantage of shared resources for reducing the total inference time, memory footprint and model size.

We applied MTL to a Computer Vision problem in which we aim at solving two tasks: object detection and semantic segmentation. We adapted and trained state-of-the-art models for each task, with the limitations that the MTL training introduced. We chose the Single Shot Multibox Detector for object detection and simple Fully Convolutional Networks with skip connections for semantic segmentation. The base network was replaced with ResNet-50 in order to achieve lower inference times. We developed MTL models combining both tasks and trained and evaluated them for two different datasets with different characteristics, mimicking different applications: Pascal VOC, for hand-held, mobile applications; and a combination of Okutama, Swiss and SDD datasets for creating a multitask dataset for aerial view applications.

The challenges found during the training of multitask models included the lack of datasets with multitask annotation, which were dealt with by combining different datasets; the increased number of choices that need to be made when combining architectures to create multitask models, the different paces with which different tasks learn, peaking at different

times; and the difference in training strategies of the different models for the different tasks forming the multitask model, which have to be trained jointly in the end.

The compromises made, the particularities of the MTL training process itself and especially the lack of a strong data augmentation to handle multiple scales caused the final accuracies of our multitask models to clearly lag behind that of the single-task ones trained without limitations although they clearly improved in terms of speed and usage of resources, being 1.6x faster, lighter and consuming less memory than deploying the two single-task models simultaneously. Compared to the single-task models trained with the same limitations, the multitask models matched or outperformed their accuracy for one of the tasks.

We conclude that MTL has the potential to give superior performance in accuracy versus speed over using multiple single-task models simultaneously as far as the two tasks of object detection and semantic segmentation are concerned. This is in exchange of a training process that is more complex as it requires extra choices to be made, a new tuning of the parameters that jointly works well for each task and overcoming new challenges that were not present in the training of single-task models.

There are several potential directions for extending this thesis. First, most interestingly, investigating how to overcome the challenges found here and finding strategies for making easier the process of defining and training multitask models, for example following directions similar to [30], where the architecture of the multitask network itself is modified during training taking into consideration accuracy but also resource sharing, combining the approach with guidelines from the meta-architectures that are known to perform best for each task according to the literature. Other future work that is of interest is to evaluate how MTL works when applied to other domains. Also, in order to use the models for real applications, further tuning the parameters and choices made here to achieve better results or adding more tasks of interest such as action detection, that can be of interest from an application perspective.

References

[1] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, Jul. 1997.

[2] D. Dong, H. Wu, W. He, D. Yu, and H. Wang, "Multi-task learning for multiple language translation," in *ACL*, 2015.

[3] D. Zhang, D. Shen, A. D. N. Initiative, and others, "Multi-modal multi-task learning for joint prediction of multiple regression and classification variables in alzheimer's disease," *Neuroimage*, vol. 59, no. 2, pp. 895–907, 2012.

[4] "DRONET," *DRONET – We Develop the DRONET Open Platform to Transform the Unprecedented Opportunity of Designing and Safely Operating Low-Altitude Airspace into the Next IT Revolution.* [Online]. Available: http://siliconmountain.jp/en/frontpage/

[5] J. Laurmaa, "A deep learning model for scene segmentation of images captured by drones," Master's thesis, EPFL, Switzerland, 2016.

[6] Z. W. Pylyshyn and R. W. Storm, "Tracking multiple independent targets: Evidence for a parallel tracking mechanism," *Spatial Vision*, vol. 3, no. 3, pp. 179–197, 1988.

[7] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I. Reid, and S. Roth, "Tracking the trackers: An analysis of the state of the art in multiple object tracking," *ArXiv Preprint ArXiv:1704.02781*, 2017.

[8] Wikipedia, "List of nvidia graphics processing units — wikipedia, the free encyclopedia." 2017 [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_Nvidia_graphics_processing_units&oldid=779901473

[9] Wikipedia, "Tegra — wikipedia, the free encyclopedia." 2017 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Tegra&oldid=779810508

[10] 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.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadar-

rama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *ArXiv Preprint ArXiv:1408.5093*, 2014.

[12] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678, 2016.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *ArXiv Preprint ArXiv:1512.03385*, 2015.

[15] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A deep convolutional activation feature for generic visual recognition," in *International Conference in Machine Learning (ICML)*, 2014.

[16] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1717–1724.

[17] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 818–833.

[18] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2014.

[19] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, vol. abs/1312.6229, 2013.

[20] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," in *CVPR*, 2016.

[21] I. Kokkinos, "UberNet: Training a Universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory," *CoRR*, vol. abs/1609.02132, 2016.

[22] S. Brahmbhatt, H. I. Christensen, and J. Hays, "StuffNet: Using 'stuff' to improve object detection," *CoRR*, vol. abs/1610.05861, 2016.

[23] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 109–117.

[24] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[25] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce english text," *Complex Systems*, vol. 1, no. 1, pp. 145–168, 1987.

[26] H. Bilen and A. Vedaldi, "Integrated perception with recurrent multi-task neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M.

Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 235–243.

[27] Y. Yang and T. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," *ArXiv Preprint ArXiv:1605.06391*, 2016.

[28] M. Long and J. Wang, "Learning multiple tasks with deep relationship networks," *ArXiv Preprint ArXiv:1506.02117*, 2015.

[29] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3994–4003.

[30] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," *ArXiv Preprint ArXiv:1611.05377*, 2016.

[31] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 94–108.

[32] J. Yim, H. Jung, B. Yoo, C. Choi, D. Park, and J. Kim, "Rotating your face using multi-task deep neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 676–684.

[33] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[34] R. Ranjan, V. M. Patel, and R. Chellappa, "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition," *ArXiv Preprint ArXiv:1603.01249*, 2016.

[35] R. Ranjan, S. Sankaranarayanan, C. D. Castillo, and R. Chellappa, "An all-in-one convolutional neural network for face analysis," *ArXiv Preprint ArXiv:1611.00851*, 2016.

[36] Y. Zhu, R. Urtasun, R. Salakhutdinov, and S. Fidler, "Segdeepm: Exploiting segmentation and context in deep neural networks for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4703–4711.

[37] S. Gidaris and N. Komodakis, "Object detection via a multi-region and semantic segmentation-aware cnn model," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1134–1142.

[38] A. R. Cheng-Yang Fu Wei Liu, "DSSD: Deconvolutional single shot detector," *ArXiv Preprint ArXiv:1701.06659*, 2017.

[39] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[40] X. Li, L. Zhao, L. Wei, M.-H. Yang, F. Wu, Y. Zhuang, H. Ling, and J. Wang,

"DeepSaliency: Multi-task deep neural network model for salient object detection," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3919–3930, 2016.

[41] G. Gkioxari, R. Girshick, and J. Malik, "Contextual action recognition with r*CNN," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[42] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik, "R-cnns for pose estimation and action detection," *ArXiv Preprint ArXiv:1406.5212*, 2014.

[43] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[44] J. Baxter and others, "A model of inductive bias learning."

[45] T. M. Mitchell, S. B. Thrun, and others, "Explanation-based neural network learning for robot control."

[46] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[47] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[48] H. Azizpour, A. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of transferability for a generic convnet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 9, pp. 1790–1802, 2016.

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[50] 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.

[51] 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.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[53] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *ArXiv E-Prints*, Dec. 2016.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[55] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," *CoRR*, vol. abs/1606.02147,

2016.

[56] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv Preprint ArXiv:1704.04861*, 2017.

[57] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 535–541.

[58] L. J. Ba and R. Caruana, "Do deep nets really need to be deep?" *CoRR*, vol. abs/1312.6184, 2013.

[59] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *ArXiv E-Prints*, Mar. 2015.

[60] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *ArXiv Preprint ArXiv:1412.6550*, 2014.

[61] J. Laurmaa, A. Holliday, C. Kandaswamy, and H. Prendinger, "Speedup of deep learning ensembles for semantic segmentation using a model compression technique," in *Preprint Submitted to Journal on Computer Vision and Image Understanding*, 2016.

[62] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[63] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[64] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *ArXiv:1602.07360*, 2016.

[65] D. Gschwend, "ZynqNet: An fpga-accelerated embedded convolutional neural network." https://github.com/dgschwend/zynqnet.

[66] M. Motamedi, D. Fong, and S. Ghiasi, "Fast and energy-efficient CNN inference on iot devices," *CoRR*, vol. abs/1611.07151, 2016.

[67] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *ArXiv Preprint ArXiv:1412.7062*, 2014.

[68] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *ArXiv Preprint ArXiv:1511.00561*, 2015.

[69] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *The IEEE International Conference on Computer Vision (ICCV)*,

2015.

[70] W. Liu, A. Rabinovich, and A. C. Berg, "ParseNet: Looking wider to see better," *CoRR*, vol. abs/1506.04579, 2015.

[71] S. Saha, G. Singh, M. Sapienza, P. H. Torr, and F. Cuzzolin, "Deep learning for detecting multiple space-time action tubes in videos," *ArXiv Preprint ArXiv:1608.01529*, 2016.

[72] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[73] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition*, 2014.

[74] R. Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[75] K. H. Jifeng Dai Yi Li, "R-FCN: Object detection via region-based fully convolutional networks," *ArXiv Preprint ArXiv:1605.06409*, 2016.

[76] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and others, "Speed/accuracy trade-offs for modern convolutional object detectors," *ArXiv Preprint ArXiv:1611.10012*, 2016.

[77] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, "POI: multiple object tracking with high performance detection and appearance feature," *CoRR*, vol. abs/1610.06136, 2016.

[78] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *ArXiv:1603.00831 [Cs]*, Mar. 2016.

[79] J. Mahadeokar, "PyNetBuilder," *GitHub Repository*. https://github.com/jay-mahadeokar/pynetbuilder; GitHub, 2016.

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

[81] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[82] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[83] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[84] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning social eti-

quette: Human trajectory understanding in crowded scenes," in *European Conference on Computer Vision*, 2016, pp. 549–565.

[85] M. Mueller, N. Smith, and B. Ghanem, "A benchmark and simulator for uav tracking," in *Proc. of the European Conference on Computer Vision (ECCV)*, 2016.

[86] Sensefly, "Swiss dataset - example datasets," *Example Datasets: SenseFly SA*. [Online]. Available: https://www.sensefly.com/drones/example-datasets.html

[87] G. Mattyus, S. Wang, S. Fidler, and R. Urtasun, "Hd maps: Fine-grained road segmentation by parsing ground and aerial images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3611–3619.

[88] ISPRS, "2D semantic labeling - vaihingen dataset," *2D Semantic Labeling - Vaihingen - ISPRS*. [Online]. Available: http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-vaihingen.html

[89] S. Wang, M. Bai, G. Mattyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun, "TorontoCity: Seeing the world with a million eyes," *ArXiv Preprint ArXiv:1612.00423*, 2016.

[90] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *International Conference on Computer Vision (ICCV)*, 2011.

[91] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision (ICCV) 2015*, 2015.

[92] G. Csurka, D. Larlus, F. Perronnin, and F. Meylan, "What is a good evaluation measure for semantic segmentation?." in *BMVC*, 2013, vol. 27, p. 2013.

[93] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015.

[94] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *ArXiv Preprint ArXiv:1212.5701*, 2012.

[95] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.