Gathering a Machine Learning dataset for object detection from a satellite-platform

Version 1

F. van Veelen

On bandwidth-efficient gathering of a Machine Learning dataset for Object Detection with Faster-RCNN from a satellite-platform

TU Delft

Glossary

Adam Adaptive Moment Estimation

AI Artificial Intelligence

AP Average Precision

ASIC Application-Specific Integrated Circuit

CNN Convolutional Neural Network

CPU Central Processing Unit

CV Computer Vision

DCT Discrete Cosine Transform

DL Deep Learning

EO Earth Observation

FC Fully Connected

FPGA Field Programmable Gate Array

GAN Generative Adversarial Network

GPU Graphical Processing Unit

IoU Intersection over Union

JPEG Joint Photographic Experts Group

KNN K-Nearest Neighbours

LEO Low Earth Orbit

ML Machine Learning

MOS Mean Opinion Score

NAG Nesterov Accelerated Gradient

NLP Natural Language Processing

NN Neural Network

PSNR Peak Signal-to-Noise Ratio

R-CNN Regional Convolutional Neural Network

ReLU Rectified Linear Unit

RoI Regions of Interest

RPN Regional Proposal Network

SIFT Scale-Invariant Feature Transform

SSD Single Shot MultiBox Detector

SVM Support Vector Machine

TOPS Tera Operations Per Second

YCoCg Luma Chrominance orange Chrominance green

YOLO You Only Look Once

Gathering a Machine Learning dataset for object detection from a satellite-platform

Version 1

by

F. van Veelen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on April 8, 2021 at 11:00.

Student number:	4441818	
Project duration:	13/04/2020 – 08/04/2021	
Thesis committee:	Ir. Dr. C.J.M. Verhoeven,	TU Delft, supervisor
	Dr. R.T. Rajan,	TU Delft
	Dr. Ir. A.J. van Genderen,	TU Delft
	Dr. Ir. Bert Monna,	Hyperion Technologies

This thesis is confidential and cannot be made public until April 8, 2022.

TU Delft

Preface

Before you lies the Thesis "Gathering a Machine Learning dataset for object detection from a satellite-platform", which revolves around the question: how can machine learning be implemented on a satellite platform in a practical way. It is a first step in realising the end goal. It was written as part of the Embedded Systems program at the Delft University of Technology (TUDelft). This thesis was written between April 2020 and March 2021.

This thesis was done in cooperation with Hyperion Technologies, who proposed the research. The research question was formulated together with Hyperion Technologies. It is in part an exploratory work on the problems of using machine learning on-board of satellites and in part in-depth research how to gather a dataset in a bandwidth-efficient manner.

I would like to thank everyone at Hyperion Technologies for the support, both in guidance/support and financially. Furthermore I would like to thank my supervisors for their guidance and support. This work has made use of the experimental systems on the Dutch national e-infrastructure with the support of the SURF Cooperative.

I hope you enjoy reading this work,

F. van Veelen
March 23, 2021

Contents

1 Introduction

In the past years, small Earth Observation (EO) satellites have become increasingly capable of taking high-resolution images at high sample rates. These images contain valuable information for different sectors, such as the agricultural and military sector. Furthermore they can contain important information about the climate and climate change. Sending these images to earth requires a large amount of down-link bandwidth. This results in heavy, large power modules and communication modules, resulting in larger, more expensive (in terms of launch cost as well as in terms of production cost) satellites. This phenomenon already results in satellites not sending all information they gather, with examples of being able to send 2 minutes worth of data per orbit (approx. 90 minutes) not being out of the ordinary. As more and more satellites are transmitting data towards earth the communication is also expected to become even more power-intensive (or even more limited), since the (theoretically) available bandwidth per satellite is reduced. Therefore a shift towards a different approach is necessary. Smarter ways to get the relevant information to earth have to be developed. The goal is to develop a widely applicable method of extracting that relevant information on-board of the satellite.

Since these satellites often have missions with one specific goal, where the images are eventually only used to extract information using object detection algorithms on earth, it is a logical step to consider extracting this information on board of the satellite itself, such that only the extracted information has to be transmitted, reducing the power and bandwidth waste on transmission. Object detection algorithms are usually either machine learning based or deep learning based. Machine learning approaches require manual definition of "features" to describe what kind of object one is interested in. Deep learning approaches are end-to-end approaches, where the features are self-learned. For this project it is chosen to use a deep learning approach, as this is the more adaptable approach to different problems, since no manual definition of features is needed for each problem. This is also the direction in which literature has gone in recent years, with a large increase in Deep Learning research since AlexNet[31] (2012). Trained features (such as in Deep Learning based approaches) tend to perform better than hand-picked ones on complex problems such as object detection in images. Deep learning approaches are based on Convolutional Neural Networks (CNNs), which are explained in appendix A.1.4. Non-deep learning based approaches are also briefly discussed in A.2

Apart from being bandwidth-restricted satellites are also power-restricted, as needing more power means needing larger solar panels as well as larger batteries. Both those factors result in increasing mass as well as physical size, resulting in a more expensive satellite. Therefor the power usage of the proposed solution to the bandwidth-restriction should be less than or equal to the status quo. It is expected that this is the case, the power budgets of the different approaches are worked out in appendix 2. This means that if one would have a trained CNN, it would save power on-board of the satellite. It should be noted that the total power consumption of the approach that uses an object detection algorithm depends on the exact approach taken, which is also described in appendix 2.

Our goal is thus to be able to deploy an object detection CNN on a satellite. However, it first has to be trained. The training can be done in different ways, which can be classified as supervised, unsupervised and semi-supervised approaches. This thesis will focus on supervised approaches. While an unsupervised approach combined with in-orbit training is also an interesting research path, it is not considered in this thesis, as it leads to a different problem with different challenges, where the main challenge is the power consumption of the training in orbit.

Supervised algorithms need annotated data in order to train the model. In general having more data results in a better algorithm, since the data is more representative for the input-space. Since our goal is to reduce the bandwidth usage of a satellite, this results in a controversy. On one hand as many images as possible are wanted to train, on the other hand the amount of data transmission should be kept as low as possible.

Gathering such a dataset is traditionally done by gathering as many high-quality (in terms of resolution, colour depth, dynamic range etc.) samples as possible. These images are then annotated (most often by humans) to then be used for the training. The most expensive step of this process is often the annotation of the images, as this can cost a lot of man-hours. Reducing the amount of man-hours is done by reducing the images needed (thus having as high-quality images as possible). For the application described above the trade-off is different, as the cost of satellites is high and the lifetime is limited. The problem is thus more about using the satellite as efficiently as possible, while being bound by a limited amount of bandwidth.

As such, the object of this thesis is to provide a new method of gathering such a dataset more efficiently. The accuracy reached by a given object detection CNN should be maximized with a minimal amount of training data transmission needed. This will be called data-efficient training from now on.

Of course, using an existing dataset is an obvious solution for many situations where these are available. That, however, does not mean that this thesis is not relevant for those situations. In those situations the same approach can still be used, albeit from a different starting point. The approach given in this thesis can then be used for efficient continuous learning during the mission. Furthermore, existing datasets are almost never exactly the same, thus one usually wants to "fine-tune" the model on data from the satellite.

To reduce the amount of data transfer from space needed to gather the training dataset, the effect of using classical image compression techniques on the training set (on-board of the satellite) will be researched. While other methods to reduce the size of the training set exist, such as converting images to grey-scale, using auto-encoders or reducing the resolution of the images, classical image compression is chosen for this work. Classical image compression algorithms are designed to maintain the information in an image while reducing the data size of said image, which is the goal in this case as well. Auto-encoders are also a promising approach to use with this work, but could not be tested due to time constraints. By compressing images the spread over the input-space is not reduced, while the amount of data transmission is reduced. It is expected that this will increase the data-efficiency of the training. Compressing the images too far is expected to result in the loss of the needed information within the image. The optimum herein has to be found. While this work focuses on using CNNs as the object detection method, the method of using image compression to build the training set might extend to different object detection algorithms.

The mission approach proposed in this thesis is shown graphically in figure 1.1. This approach assumes one satellite, however it could easily be extended to multiple satellites. By assuming one satellite, the least favourable situation is considered, as sharing the trained model within a constellation splits the training cost between the satellites. It can be seen that the lifetime of the satellite can be split into two major parts, the initial phase and the main phase. In the initial phase the goal is to get an initial classifier that is then improved in an iterative way in the main phase. This results in a lifelong-learning approach, where there is continuous improvement during the lifetime of the satellite. The focus of this thesis will lay on the block "Send complete images with the bandwidth that is left", where the question whether those images should be compressed or not before sending them, as well as how far they should be compressed will be answered.

If one considers the initial model trained on external data (data not produced by the satellite) already satisfactory, one might consider drastically reducing the cost of the satellite by reducing the communication capabilities of the satellite, resulting in smaller photo-voltaic panels as well as a lighter/smaller communication module. This can be done since no complete images have to be send to earth, instead only annotations are send to earth. Taking this approach one does have to take care of the differences between imagers of the external data and the satellite, which might result in accuracy loss.

The data-flows during the mission are shown graphically in figure 1.2. It can be seen that two CNNs exist, one on the satellite and one within the training software on ground. The CNN on the ground is updated when new data arrives from the satellite. The models are kept synchronised by updating the model on-board of the satellite when the accuracy improvement of the updated version on the ground is large enough. The external training data in the diagram is optional, if available, as seen in figure 1.1. The control algorithm shown has to decide which images to send using the left-over bandwidth, as well as at what compression level to send said image. This thesis goes into how to decide what compression level the images have to be send at. The human classifier is needed to annotate the training images. It should be noted that depending on the exact problems that the object detection should solve, other sources of annotation might be available, such as

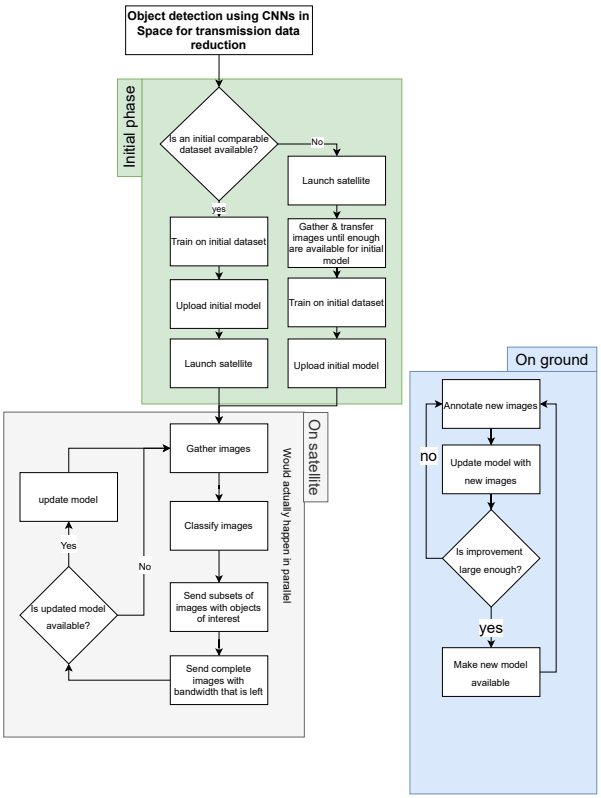


Figure 1.1: Graphic overview of mission

radar data for ships. This data can be used to reduce the cost of the human annotation. It can be seen that the annotations that the human produces also go to the client of the data, overwriting the predictions of the CNN if any differences exist.

The training will first be analysed for the static case, where a fixed amount of images is available. It will be researched what the effect of image compression is on the accuracy of the trained network. This will then be extended to the time-varying case, where the next image that the satellite sends to earth has to be chosen. To make this choice it will be researched what the most efficient use of the available bandwidth is in terms of image quality. Finally predictions will be made on what the final possible accuracy of models trained on different compressed training sets is.

For this research, a dataset will be used to get empirical results. The chosen dataset is the Airbus Ship Detection Challenge dataset, which is discussed below. The focus of this thesis is not on this specific dataset, but to develop a general method of gathering a dataset for object detection in a bandwidth-efficient manner. The dataset described below is merely an example used to get empirical results.

1.1. Overview of airbus dataset

First an overview of the dataset used for this thesis will be given, to provide context about the problem. It should be noted that the problem is used as an example of a problem and the solution should be as general to other object detection problems as possible. The dataset, as described in the introduction, is taken from a Kaggle challenge by Airbus. It is a dataset consisting of (parts of) images taken by the SPOT constellation. These images have a ground resolution of 1.5 meters and consist of land, coast and open sea images. The challenge posted was about detecting ships within the images. The relevance of ship detection is great. Multiple parties are interested in tracking ships, such as Airbus and different Departments of Defense. The importance lies in the tracking of a variety of human activities on the sea, such as fishing, whaling, oil drilling and trafficking on the sea. Furthermore, piracy can be prevented if the pirate ships can be tracked. Piracy is estimated to cost $16 billion per year.[1]

[1] https://web.archive.org/web/20071214040613/http://www.foreignaffairs.org/20041101faessay83606/gal-luft-anne-korin/terrorism-goes-to-sea.html

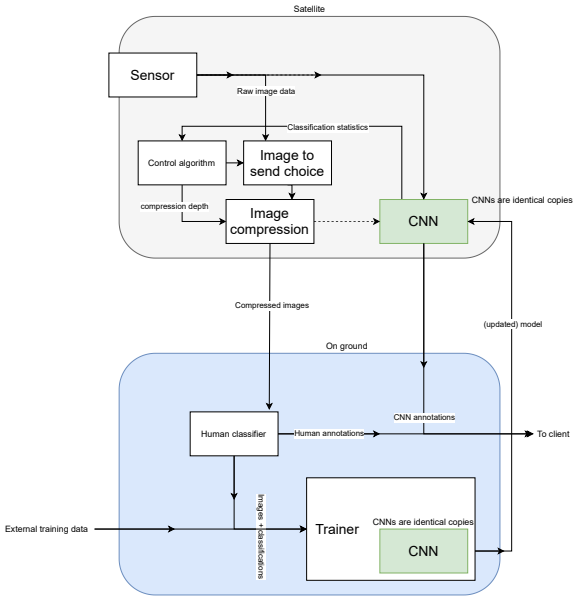


Figure 1.2: Dataflows during mission

The ships are designated with a bit-mask. While the challenge was semantic image segmentation (see appendix E), for this thesis the goal is object detection (see appendix E).

Some example images can be seen in figure 1.3a, with the ships highlighted in figure 1.3b. It can be seen that the amount of ships per image varies drastically. In figure 1.4 the distribution is shown. It is immediately clear that the most images have no ships, while the largest amount of ships within an image is 15. The differences in the amount of ships per image is further illustrated in figure 1.5. It can be seen that there are land or clouded images with no visible ships (figure 1.5a), while images with a large amount of ships are often harbours (1.5d). It can also be seen that not all images are full size, some are filled with a blue rectangle (figure 1.5b, 1.5c).

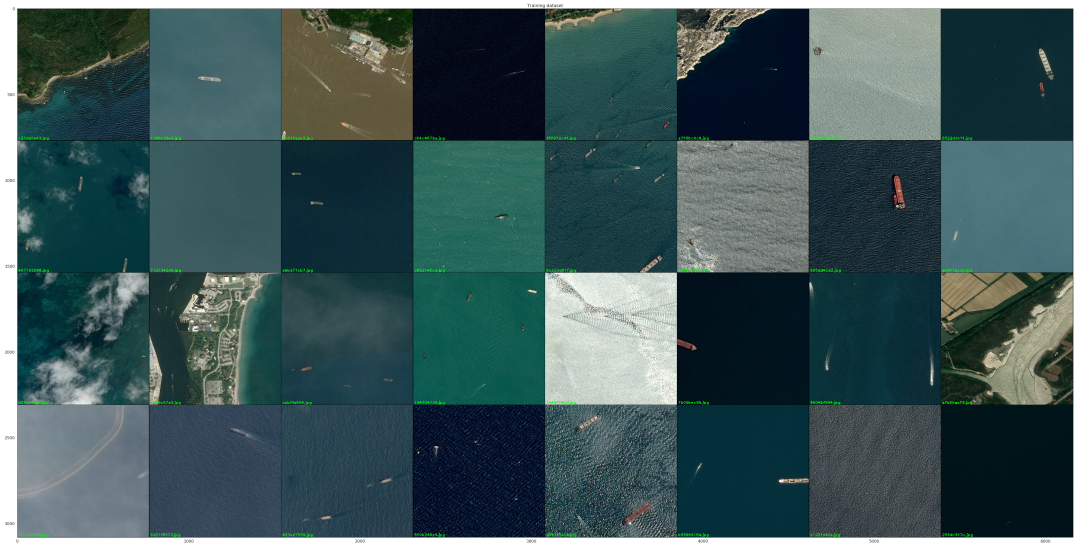
The size of the ships is also not always the same. The area of the ships within the dataset are calculated from their masks. The results are shown in figure 1.6. There are two clear peaks around 100 and 1100 m^2, most likely because of different classes of ships in the dataset. It is probable that the peak at around 1100 m^2 is because of oil tankers and cargo ships, whereas the peak at 100 m^2 consists of smaller ships such as recreational ships and smaller passenger ships.

All images are provided in JPEG-format, with an unknown amount of compression. For this thesis it is not expected that the results are influenced by this fact, since the quality is relative to the original image (the JPEG image provided in the dataset) and all results are, in principle, only valid for this specific dataset.

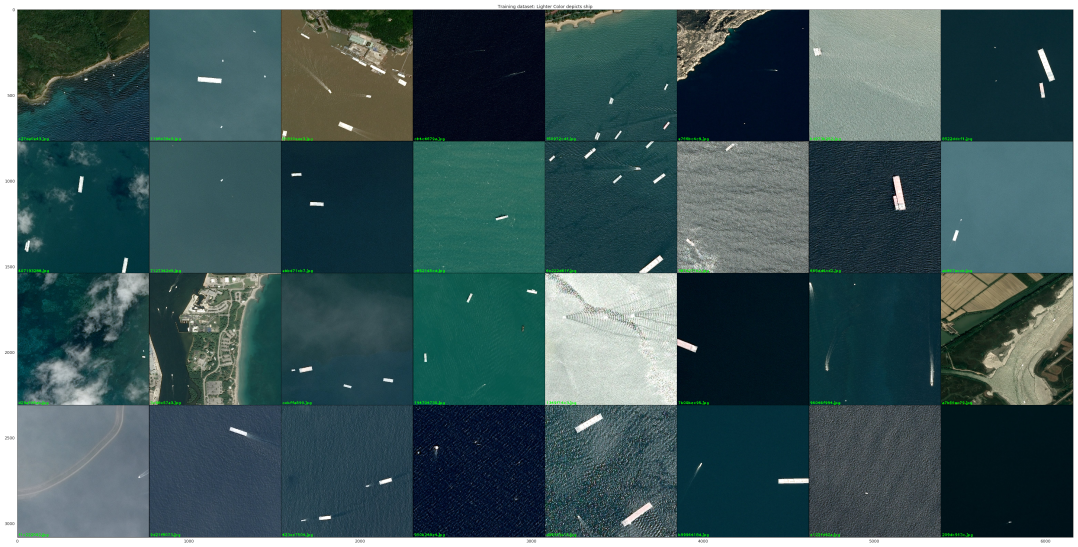
1.2. Conclusion

To conclude this chapter, to make more effective use of satellites a more efficient manner of transmitting useful information from the satellite to earth has to be developed. One technology that can be used to do this are CNNs, however to use this technology a dataset with a large amount of diverse, relevant images is needed. Classically, such a dataset would be large, thus one would need a large amount of data transmission to get the training images on earth, resulting in the problem that should be solved in the first place. The research question can thus be formulated as: What is the most effective method of gathering a training dataset for object detection on-board of a satellite, given a bandwidth restriction on the communication between the satellite and earth? The approach to solve this proposed in this work is using classical image compression on the images of the training set before sending them to earth, resulting in a more data-efficient method of building a training dataset.

With the goal and approach explained, the literature on this subject will be discussed. After discussing the literature this document will continue analysing the data-efficient training approach proposed.



(a) Randomly selected images from dataset



(b) Randomly selected images from dataset with their mask highlighted

Figure 1.3: Randomly selected images from dataset

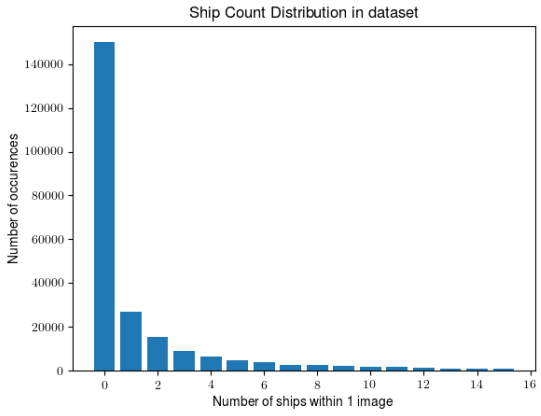
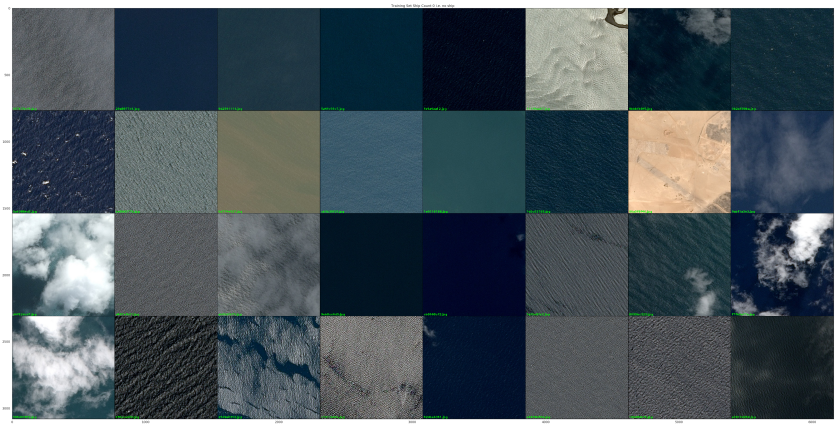
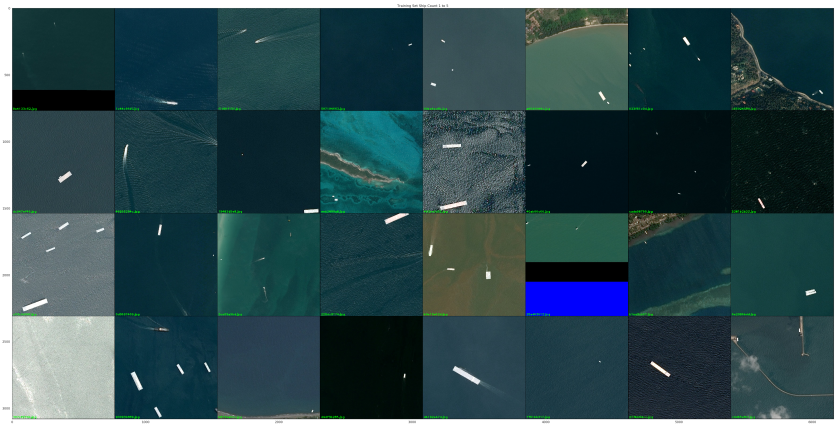


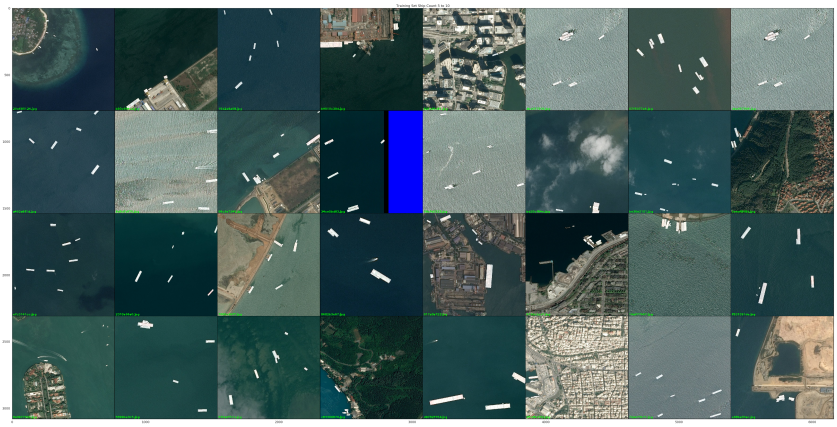
Figure 1.4: Ships per image



(a) Randomly selected images with no ships



(b) Randomly selected images with 1 to 5 ships



(c) Randomly selected images with 5 to 10 ships

(d) Randomly selected images with more than 10 ships

Figure 1.5: Randomly selected images from dataset with different amounts of ships

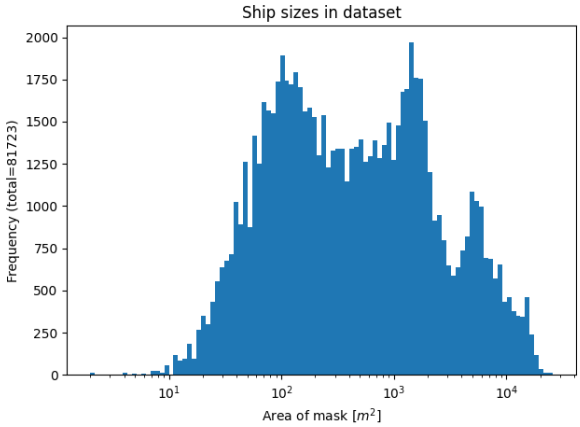


Figure 1.6: Sizes of ships within the dataset

2

Feasibility of using CNNs in orbit

When starting a new project, the feasibility of this project should always be researched. This is why in this chapter an estimation of the power savings will be made, compared between two cases: one case where JPEG compression is used, to then transmit the compressed images to earth, with the other case being that an object detection CNN is used to detect ships in a part of the image, to then only send that part of the image to earth. In this chapter some assumptions will be made, as it is an exploratory investigation. If the proposed approach is feasible in the chosen scenario, it is considered to be worth the further research. When considering using the approach described in this thesis, one should first reevaluate the assumptions made in this chapter.

In this chapter two different scenarios are assumed. Both assume an Earth Observation satellite. The first one (the currently used approach) is that after taking an image, it is compressed using an efficient Joint Photographic Experts Group (JPEG) algorithm, after which the compressed image is send to earth. The second scenario is that, after taking a picture, it is analysed by an object detection CNN and only the parts of the image that contain the objects that the mission is interested in are send to earth. These parts are compressed using the same efficient JPEG algorithm.

Since the main benefit of the proposed scenario would be reducing the bandwidth needed to transmit data to earth, the solution is considered feasible if the power budget of the solution using the object detection algorithm is comparable to the power usage of the scenario where all data is transmitted using a State-of-the-Art radio downlink.

Another possible scenario could be thought of, where no image is send to earth but only a message that says object X is in location Y, but for the proof-of-concept this will not be assumed, as it is hard to proof the working of the object detection CNN to a client this way. If other methods of gathering (parts of) the data are available, these sources can also be used to validate the working of the object detection, meaning this scenario can be used.

2.1. Data transmission

The main difference between the solutions is what the power budget is spent on, where for the object detection CNN solution more power is needed in the processing, while for the "normal" solution more power is needed in transmission. In this section the differences will be analysed nominally.

The amount of data transmission saved by the object detection CNN solution is highly dependent on the amount of positives. For example, if the CNN is looking for container ships, there are 50000 ships in the world [1], which in total cover approximately 750 km^2 [2], or $1.5 * 10^{-4}$% of the earths' surface, while forests cover $\approx$ 8% of the earths' surface [3]. This means that the theoretical maximum saving on power, assuming an oracle that provides the location of the objects in an image that uses no power is 92% for a forest detector, while it is 99.9997% for a ship detector.

If no images of the object found are needed to be send to earth, this consideration is way less relevant, since the amount of meta-data per object found is negligible compared to one full-sized image. It is therefore

[1] https://www.ics-shipping.org/shipping-facts/shipping-and-world-trade
[2] https://en.wikipedia.org/wiki/Container_ship
[3] https://data.worldbank.org/indicator/AG.LND.FRST.K2

not overly optimistic to choose a use case such as detecting ships, where almost no positives are found.

The cost of transmission of course depends on the means for transmission. For example, the Hyperion CubeCat laser communication module can handle 1Gbps at 15W. This means it uses 15 nJ/bit. A State-of-the-Art radio downlink is capable of transmitting 150 Mb/s with a Tx power (the actual power put into the antenna) of 2W.[4] With a typical efficiency this would results in approximately 10W of input power, which results in using 8.33 nJ/bit. The more typical solution, especially for lower transmission rates, is the radio downlink, since the CubeCat is a relatively heavy module (1.33kg) and requires precise attitude control, resulting in higher mission costs. This is why for the comparison the radio transmission metric is used.

2.2. Processing hardware

For the processing hardware the edgeTPU is chosen to research the feasibility. In appendix F further details about different hardware platforms are given. It is assumed that the edgeTPU at normal speed will be enough to perform the object detection. This means that the added power for object detection is ≈ 2.5W at maximum [5]. It can perform at 400 fps for an object detection CNNs such as mobilenet. This means it uses $\frac{2.5W}{400\text{ frames}} = 6.25 * 10^{-3}$ J per frame.

2.3. Comparison

JPEG encoding costs approximately 0.8-1 J per frame on general hardware (specifically a Raspberry Pi) [3] (resolution unknown). On dedicated hardware however, a JPEG operation (compressing a 8x8 pixel piece of an image) costs 1.3 pJ [44]. This means that an image that mobilenet can handle (224x224px) consists of 784 operations and thus costs approximately 1 µJ/frame to encode to JPEG.

It is assumed that a 224x224px frame the size after $\text{JPEG}_{90\%}$ is approximately 5.12kB [6]. With the assumptions made in section 2.1, this results in $5.12kB * (1 - 0.999997) = 0.12$ bit/frame on average that has to be transmitted for the AI with image chip scenario (for only the image, no metadata). The metadata size is assumed to be negligible compared to the image chip size.

The different power costs are summarised in table 2.1.

(estimated) Power costs per frame	JPEG90% compression	Transmission	AI Cost	Total cost
Non-AI Scenario	1µJ	~5.12kB →0.34mJ	None	0.34mJ
AI with image chip	Negligible	Negligible	6.3mJ	6.3mJ
AI with only location	None	Negligible	6.3mJ	6.3mJ

Table 2.1: Estimated power costs per frame for the different scenarios

2.4. Conclusion

With the power usage increase of approximately 20 times, it can be concluded that the scenario using the object detection can be feasible, as they are comparable. The possible increase in information gathering from the satellite means a larger power budget for the processing of the data is acceptable. It is also expected that the improvement in efficient object detection hardware will be larger than the improvements in data transmission hardware, as the object detection Application-Specific Integrated Circuits (ASICs) are a new development and knows no hard physical limit, where efficient data transmission is a more mature field and is bound by physical limits.

Thus we can conclude that, at least for the use case of detecting ships, using CNNs in orbit is a feasible option.

This does not mean that every Earth Observation mission can use this approach, however a large amount could. It is expected that scientific missions generally want unprocessed data to perform experiments on, while commercial parties have less need for the unprocessed data.

Now that the theoretical feasibility of the project is validated, this work will continue with a review of the existing literature.

[4] https://www.endurosat.com/cubesat-store/cubesat-communication-modules/x-band-transmitter/
[5] https://coral.ai/static/files/Coral-USB-Accelerator-datasheet.pdf
[6] https://toolstud.io/photo/megapixel.php

3

Literature review

As the idea of using CNNs in orbit is only feasible (see appendix 2) as of recently, with both great progress with image recognition algorithms and (low-power) CNN specific hardware, as well as increasing problems in data transmission, no literature on the problem of this thesis (data-efficient training) could be found. There is however research on different parts of the solution, such as image compression techniques and prediction of the number of samples needed to train CNNs. It should be noted that while in this work it was chosen to use CNNs as the object detection algorithm, the idea of using image compression on-board of the satellite to build a training dataset might carry over to other object detection algorithms that have to be trained on large datasets. This research will be discussed in this chapter. First the image compression techniques are discussed, after which the sample number prediction is discussed.

3.1. Image Compression Techniques

As discussed in chapter 1, to maximize the training accuracy for the minimum amount of data transfer from space, the effect of image compression on the training process should be researched. As no direct research on the effect itself could be found at the time of writing, image compression techniques will be discussed.

Usually, the Peak Signal-to-Noise Ratio (PSNR) is used as a metric to analyze the effectiveness of a compression algorithm. For training CNNs, it is unknown whether this metric relates to the training effectiveness (the amount of data needed to reach a wanted accuracy) of the compressed images. It might be the case that certain artifacts created by the compression algorithms have larger or smaller effects, resulting in better training effectiveness. The PSNR does, however, relate to the amount of distortion caused by an image compression algorithm.

Another metric that is often used is the perceived quality by humans (Mean Opinion Score (MOS)), this is however harder to gather. It might be that this metric is a better estimate for the training effectiveness since CNNs are (loosely) based on human vision. This is outside the scope of this work, in which PSNR will be used.

First, an overview of image compression is given. Compression algorithms can be split into two branches: lossless and lossy compression. Lossless compression can be reversed, resulting in the exact same image. Lossy compression is not reversible, meaning the exact image can not be gotten back. This, however, does come with the advantage that the image can be further compressed. In very high level terms, this is done by taking a Fourier transform of the different channels, discarding the highest frequencies and quantizing the lower frequency parameters. The image (in a worse quality) is then gotten back by taking the reverse of the Fourier transform. Further details are described later.

Different compression algorithms were compared specifically for satellite images. [14] The results of this study are shown in figure 3.1. It can be seen that for high compression ratios JPEG-XR and CCSDS-IDC are Pareto optimal (meaning no objectively better algorithm was considered and a trade-off has to be made between compression ratio and PSNR), where DPCM (Differential Pulse Code Modulation) performs worse in both compression ratio and PSNR. For a very high PSNRs JPEG-LS is optimal, as it is near-lossless, while still providing a compression ratio of almost 3. JPEG-XR has lower computational and memory requirements [14], meaning the compression will cost less power.

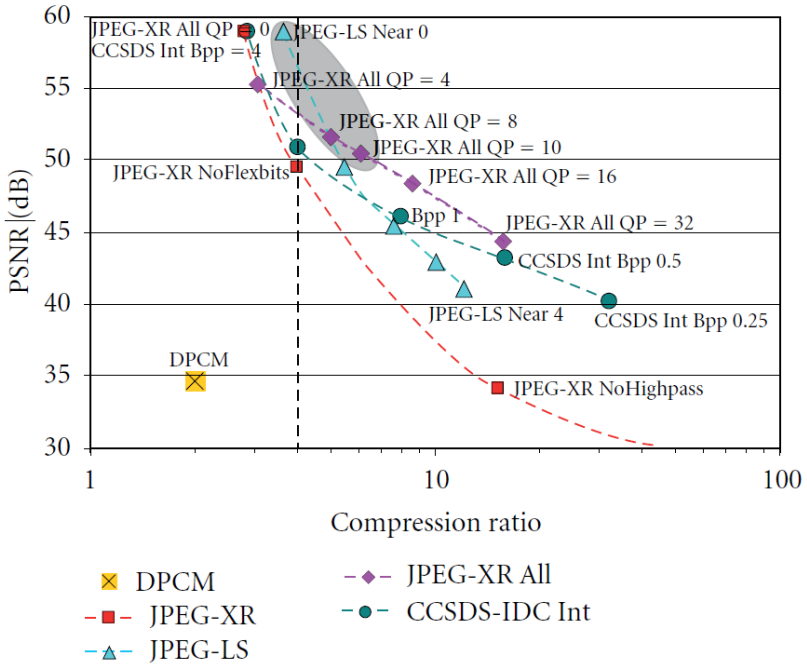
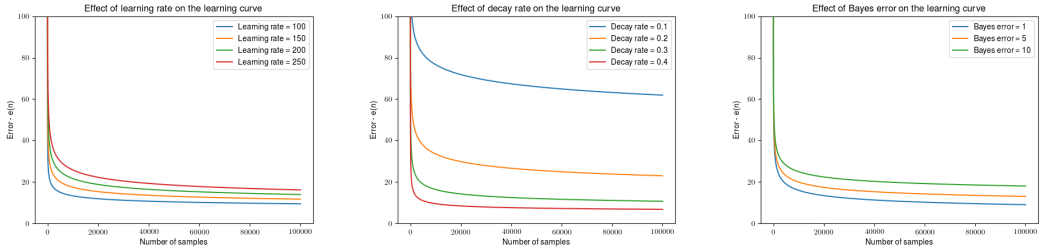


Figure 3.1: State of the Art data compression algorithms[14]

JPEG-LS is not further considered for this thesis, as the compression ratio that can be reached is low, since it is a (near-)lossless standard. DPCM works by encoding the difference between samples and predicted samples (based on previous samples) instead of samples themselves. These differences are also quantized, resulting in rounding errors and therefor lossy compression.[53]

In contrast with this approaches the JPEG-XR algorithm for images work based on a Discrete Cosine Transform (DCT). This works by converting images into blocks, which are then compressed individually. For each block, all pixels are put in a specific sequence, which is then transformed into the frequency domain similar to a Fourier transform. The compression then works by removing the higher frequencies from the transformed image. After this, the coefficients are compressed using a lossless algorithm. Further details are different for different implementations and will not be focused on in this thesis. It should be noted that because the image is split into blocks before the compression, single event upsets can only affect one block, which means at most one block of an image is lost by a single event upset, where a single event upset during encoding with DPCM would result in corruption of the rest of the image.

Therefor this thesis the JPEG-XR algorithm will be used to test the hypothesis that the amount of data transfer needed to reach a wanted (validation) accuracy can be reduced using image compression, since it has the highest PSNR at the compression ratio's wanted. It can not be concluded that this is the best compression algorithm for training effectiveness, since it is not known what the exact effect of the different compression algorithms on CNN training is, however the research can easily be extended to different compression algorithms to analyse the differences. The exact implementation and further background detail on the JPEG-XR algorithm can be found in appendix D.



(a) The effect of the learning rate on the learning curve

(b) The effect of the decay rate on the learning curve

(c) The effect of the Bayes error on the learning curve

Figure 3.2: The effect of the different parameters on the learning curve

3.2. Prediction of Number of Samples Needed

While the prediction of the number of samples that are needed to train a CNN model is an important topic for the field to become a mature engineering field [27, 28], it is often not taken into account during the development of a model, whereas the size of the validation set needed is done using the power law. [8, 21]

This might be because a lot of models are developed using existing datasets, such as ImageNet, or the data is cheap enough to gather and the expensive part (annotating the data) is mitigated by using semi-supervised learning techniques. Therefore the attempts to solve the same problem found are in the medical field, where a limited number of independent samples are available, due to the limited number of patients. This will be discussed in this section.

3.2.1. Learning curve fitting

As discussed in [11] it generally holds that the expected error rate of a trained model depending on the number of samples, for a fixed setup, follows the inverse power law, given in equation 3.1.

$$e(n) = an^{-\alpha} + b \tag{3.1}$$

With e the error, n the number of samples, b the Bayes error (the minimum error rate reachable with an infinite amount of samples), a the learning rate, and α the decay rate. [41] This means that the three parameters a, b, α have to be estimated. It should be noted that these parameters are specific to the classifier, dataset, and training approach.

The Bayes error predicts the maximum attainable accuracy of a classifier for a certain problem, thus generally is low (a few %). Having a high Bayes error means a classifier is not suited for the problem or the dataset is not consistent. The learning rate and decay rate are both positive. They both heavily depend on the problem and classifier. The different effects on the learning curve of the different parameters are shown in figure 3.2. It can be seen that both the decay rate and the learning rate have similar effects on the learning curve (determining the "learning speed"), while the Bayes error determines the maximum accuracy achievable (minimal error).

The estimation can be done using intermediate points on the learning curves for subsets of the training dataset. One such approach is discussed and tested in [4, 41], where a least-squares approach is used. One problem that was encountered was the fact that the points on the learning curve for a very small number of samples are statistically insignificant (take for example 1 sample with a dataset with 1000 labels, the algorithm could only ever predict 1 out of the 1000 classes accurately). Therefore a significance test was performed. Only points on the curve that are significant according to that test are then used.

Another approach is using a weighted least squares function where points based on more samples weighed more heavily. [9, 16] (The weights 1, 1, 1, 1, 100, 150 were used for training set sizes of 5, 10, 20, 50, 100, 200 in [9], which means the sample with n=200 is deemed to be 150 times more relevant than the sample with n=5). Having more samples means a more repeatable training (smaller variance), resulting in the sample being more reliable. Thus giving the sample a larger weightThis approach resulted in a lower prediction error than with all weights equal. In [16] a confidence interval was also calculated, by using n-fold validation.

Being able to estimate the parameters accurately with a small number of samples results in more predictability of the cost of developing an object detection CNN, estimating the final accuracy reachable, but also in faster development, as it becomes clear which classifier is more accurate with less training. This result can be used to reduce the amount of training to be done to research the effect of image compression on

training as well. Furthermore, it can extend the results to larger datasets that may not be available during research. With this theoretical background established it makes sense to analyse the theoretical feasibility of using Deep Learning (DL) based object detection algorithms on-board of small satellites. The effect of compression of the training set using image compression will then be analysed in the chapter after that. Fundamental object detection concepts will also be explained.

4

The effect of image compression on the training of a object detection net

In order to minimize the amount of data that has to be transferred from orbit to earth to build a training dataset, the usage of lossy image compression before transmission will be analyzed. This will be done for one specific dataset using a specific architecture (Faster-RCNN [50]) and a specific compression algorithm (JPEG-XR[1]). The choice of JPEG-XR is explained in chapter 3.1. The choice for the Faster-RCNN is made because it is widely implemented in different software packages and produces state-of-the-art results. While many different architectures could be used, it is expected that the results of this research carry over (qualitatively) between architectures, such that it becomes largely irrelevant which of the specific architectures is used specifically.

To compare two different models some method of scoring them needs to be defined. This is done in chapter 4.1.

4.1. Scoring method of trained models

In the field of object detection various methods of scoring a prediction exist. Most of these methods are based on a score used by a competition. The method used in this thesis is called Average Precision (AP) based on the Intersection over Union (IoU). This is a method of scoring a bounding-box prediction[2], where both the annotation and the output of a model are a square around the (predicted) object. The method is based on the area of overlap divided by the area of the union of these boxes. In the ideal situation one would want the boxes to be completely equal but for various reasons this almost never happens. The Intersection over Union (IoU) can be seen graphically in figure 4.2[3].

A threshold is then defined to decide how high the IoU has to be before a prediction is considered correct. This threshold is 0.5 for this thesis. A false positive is a predicted bounding box which does not correspond to any existing bounding box. A false negative is an annotated bounding box that does not correspond with any prediction.

Precision can then be defined as $\frac{\#\text{True positive}}{\#\text{True positive}+\#\text{False positive}}$ and recall as $\frac{\#\text{True positive}}{\#\text{True positive}+\#\text{False negative}}$, or in other words: precision is the chance that a predicted box corresponds to a ground truth box, while recall (also referred to as sensitivity, true positive rate (TPR)) is the chance that a predicted negative corresponds to a ground truth negative. Note that there is a trade-off between scoring well on these measures. Precision could easily be perfect by predicting the object in question is not in the dataset at all, but this would result in a recall of 0 because of the false negatives ($\frac{0}{0+\#Falsenegative}$). On the other hand, predicting everything is a positive, results in a perfect recall, but a low precision because of the large amount of false positives.

Important to note is that none of these metrics include the true negatives. This is because the amount of true negatives is very large, as each subset of the image with no object in more than half of the image (in case an IoU threshold of 0.5 is used) is a true negative. Since an image can be split into nearly infinite subsets (in

[1] `https://jpeg.org/jpegxr/index.html`
[2] The definition also holds for segmentation networks
[3] Image source: `https://towardsdatascience.com/confusion-matrix-and-object-detection-f0cbcb634157`

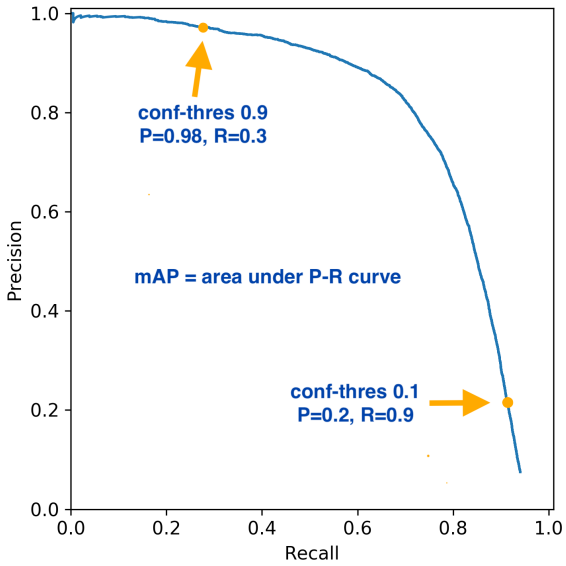


Figure 4.1: An example Precision-Recall curve

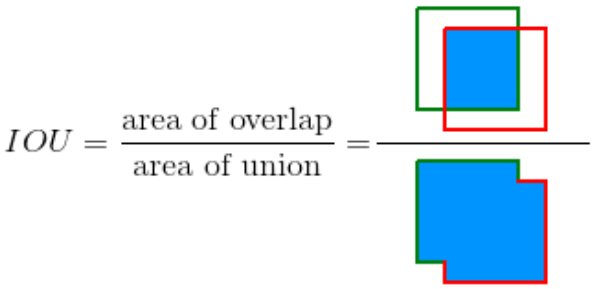


Figure 4.2: A diagram of what is meant by Intersection over Union

subsets of $1px \times 1px$, $2px \times 2px$ etc.) this makes the metric unusable, in part because of the computation needed to calculate the amount of true negatives.

The model also gives a confidence with its predictions. A confidence threshold is applied, above which a prediction is taken into account. Below this confidence interval the prediction is not considered. By varying this confidence threshold a curve is created on which the trade-off between precision and recall is created. An example curve is shown in figure 4.1[4]. A single metric is wanted, where the results of one model are objectively better, equal or worse than another model. This is done by taking the area below the curve. The area below this curve is called the Average Precision (AP). The average of this precision over all different classes considered is called the mean AP, or mAP. Since this thesis considers one class (ships) these are interchangeable. The AP is used as the metric to compare two models.

4.2. Method

Having defined a scoring method, the research method will now be discussed. The method for this research will be as follows: first, the model will be trained on the full training dataset without any compression. This model will be used as the baseline and is expected to be the best performing model since it uses the largest amount of data. Using this training the model parameters will also be fine-tuned to improve the performance. These fine-tuned parameters will then be used for all training.

Following, the datasets are compressed to different compression levels (for an exact definition of the compression levels see appendix D) (20, 10, 6, 4, 3, 2) to then be used for training different models. For each compression level, a separate model will be trained. This model is trained until the validation accuracy plateaus.

[4]Image source: `https://github.com/ultralytics/yolov3/issues/898`

The APs of the models are then compared. It is expected to see some loss in accuracy with a lower quality level, however, since the amount of data is also lower it is expected to be more data-efficient.

After this, a test will be done where all quality levels get an equal amount of "bandwidth", which means lower quality level models are trained using more images. It is expected that this results in a curve, where the optimum is somewhere in the middle-quality levels, since very low-quality levels result in loss of detail in the image, while high-quality levels result in little images being available. This test is done for multiple bandwidth sizes (the equivalent of {1000, 10000, 30000} Q_2 images), to analyze the scaling with respect to the bandwidth available of the effect. It is expected that with increasing the bandwidth available for the training set, the performance of models trained on compressed images decreases with respect to the models trained on non-compressed images, since at some point the amount of images is "enough" and the detail of the images becomes more important.

4.3. Software implementation

As mentioned, the Faster-RCNN [23] architecture will be used. This architecture was chosen because it is widely known and can perform well on different image-scales, due to its' anchors. This architecture is implemented for Keras within Tensorpack [59]. This implementation is used because of its' focus on fast training speeds due to its' efficient dataflow. Having a fast training speed is relevant since 50 different models have to be trained.

The backbone used is ResNet-50 [22], which consists, as the name implies, of 50 layers. These layers are visualised in `http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006`. It has around 25 million trainable parameters. ResNet is based on the idea that adding layers to a CNN should never make its' prediction performance worse (but does increase the computational power needed). This is realized by "shorting" layers with an identity bypass, effectively making those layers inactive if they do not improve the accuracy. The model takes 3.8×10^9 FLOPs to compute.

To use this project on this specific dataset changes had to be made. Additional scripts also had to be written. These can all be found on GitHub[5], together with documentation on the software. The training was done on two different hardware setups. A desktop with an NVidia 2080Ti and a server (by Surf[6]) with a NVidia Tesla V-100. On both those setups, the training is done at approximately 10 steps/second, with the GPU being the bottleneck. For faster training one could add more Graphical Processing Units (GPUs), which is supported by tensorpack.

The JPEG-XR (de-)compression is done using the imagecodecs python module[7]. The compression is done offline, meaning multiple copies (with different quality levels) of the dataset are stored, to reduce the compute power needed during training. The total amount of storage needed for the data is 37GB.

4.4. Baseline

First, the baseline was trained, in order to have a comparison point. The hyper-parameters are also tuned using this baseline, after which they are kept constant.

For all following training, the training set consists of 34k images, while the validation and the test set both consist of 4.2k images. While the original training dataset consists of more images, negative examples (images without any ships) were not taken into account, since it is more computationally efficient to "mine" negative examples from images with ships (taking parts of the image without a ship as a negative example), which is implemented by tensorpack in the Faster-RCNN example which is used.

For the first run, the precision-recall curve is shown in figure 4.3. The AP (as mentioned, the area below the precision-recall curve) of this run is 75%.

[5] `https://github.com/FrankvVeelen/AI4SatCombined`
[6] `https://www.surf.nl/en/research-ict`
[7] `https://pypi.org/project/imagecodecs/`

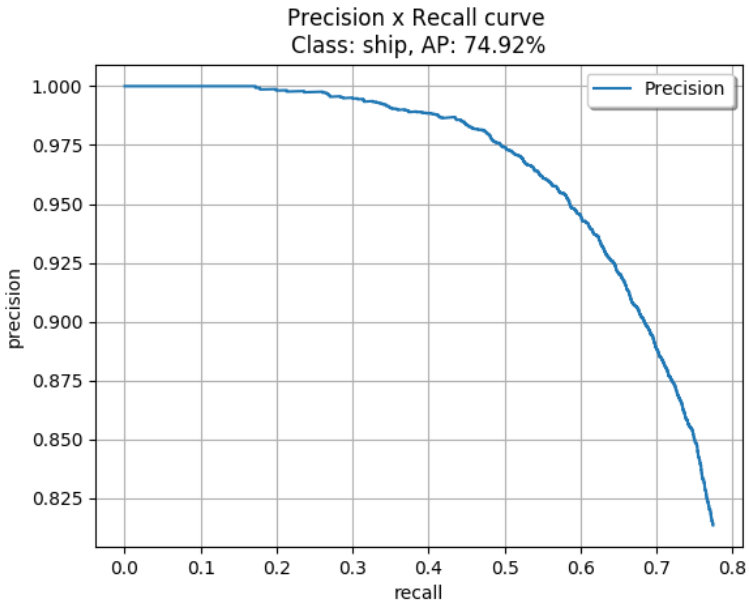


Figure 4.3: The first run of the training on the full training set

By manually looking at the inference results on the test set it was found that the model produced better results for larger ships than smaller ships, where the smallest ships were never found. It was therefore tried to reduce the size of the anchors (see appendix A.1.8), which resulted in the results shown in figure 4.4. The improvement is obvious, both by the precision-recall curve and by the AP, which increased to 85%.

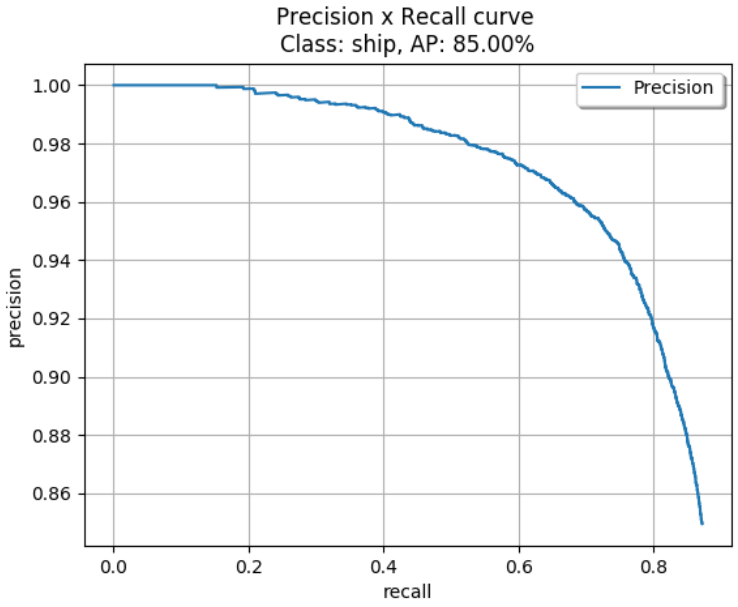


Figure 4.4: The second run of the training on the full training set (with smaller anchors)

It should be noted that both previous runs were without data augmentation. For the third run data augmentation in the form of horizontal flip (P=0.5) and vertical flip (P=0.5) (see appendix A.1.13 were added, since there is no inherent orientation in satellite images. This once again results in an obvious improvement with the AP increasing to 95%, as shown in figure 4.5. To give some perspective on this results: with a precision of 0.9 and a recall of 0.9, which is one of the points on the precision-recall curve, 90% of all ships in the test set were found and of all predicted bounding boxes 90% corresponded to a ship in the ground truth.

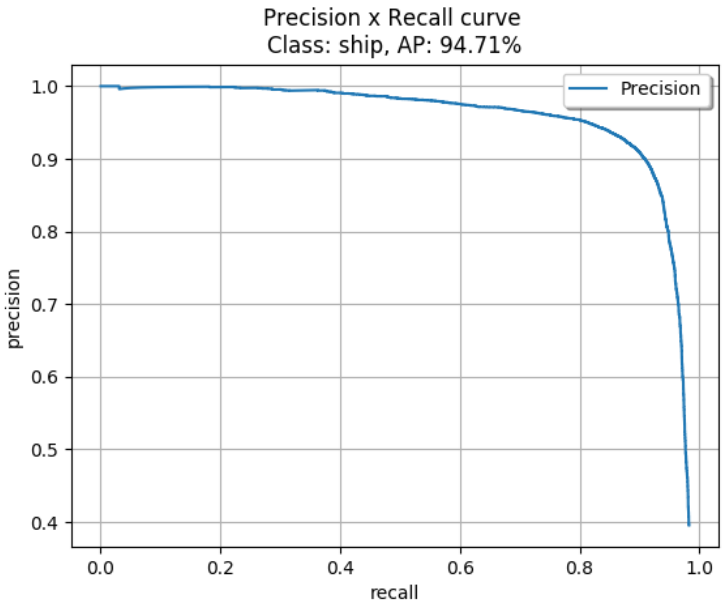


Figure 4.5: The third run of the training on the full training set (with augmentation)

This is the final model that is used as a baseline for the rest of this thesis.

4.5. Influence of the compression

Now that the baseline is established, the influence of compression on the training will be analyzed. This is done for a training set of 1000 images, to both speed up the training process as well as to be able to add extra images to the trained model later, to research the scaling of the dataset.

The training is done for 200.000 steps (200 epochs) for all compression quality levels. The training is also repeated three times, with three completely separated sets of 1000 images, in order to analyze the repeatability. The 95% confidence intervals for the precision-recall curves are shown per compression quality level in figure 4.6. From these precision-recall curves it can be seen that the training is repeatable with different parts of the dataset, as the precision-recall curves within one compression level have a small confidence interval. The difference between the (extremes of the) precision-recall curves can be seen in figure 4.7a. It can be seen that the better the image quality, the larger the area below the precision-recall curve. The AP (with the 95% confidence interval) of these results is also shown in figure 4.7b.

From figure 4.7b it can clearly be seen that image compression does indeed have a negative effect on the training result and a larger compression percentage results in a worse result. It can however also be seen that the compression to 20% has a negligible effect, while it can be seen from figure D.2 that the compressed file size is 5 times as small. Furthermore, it can be seen that the training results are repeatable, even for completely separate sets of images (from the same dataset), as all confidence intervals are small.

4.6. Static case

After confirming the expected effect and quantifying the effect of image compression on the training, the logical follow up is to analyze the static case, where a set amount of bandwidth has been available to send images to earth. The main research question to be answered is what compression quality level should have been used for sending those images. This is done by analysing the accuracy of models trained on different datasets with an equal amount of bandwidth usage (sum of file sizes). In table 4.1 the different compression levels tested are shown, as well as how many images are used for each dataset.

Models were trained on three completely separate sets of images of the sizes in table 4.1. The models trained on these datasets were then tested using the test dataset, which consists of 4255 separate images. The precision with an IoU threshold of 0.5 is then calculated. This gives us the results shown in figure 4.8a.

It can clearly be seen that the greater the compression ratio (i.e. the worse the image quality) the better the performance of the model, up to a certain point where the performance drops off again (compression level 3 scores the highest). The question that logically follows from this result is how this result scales with

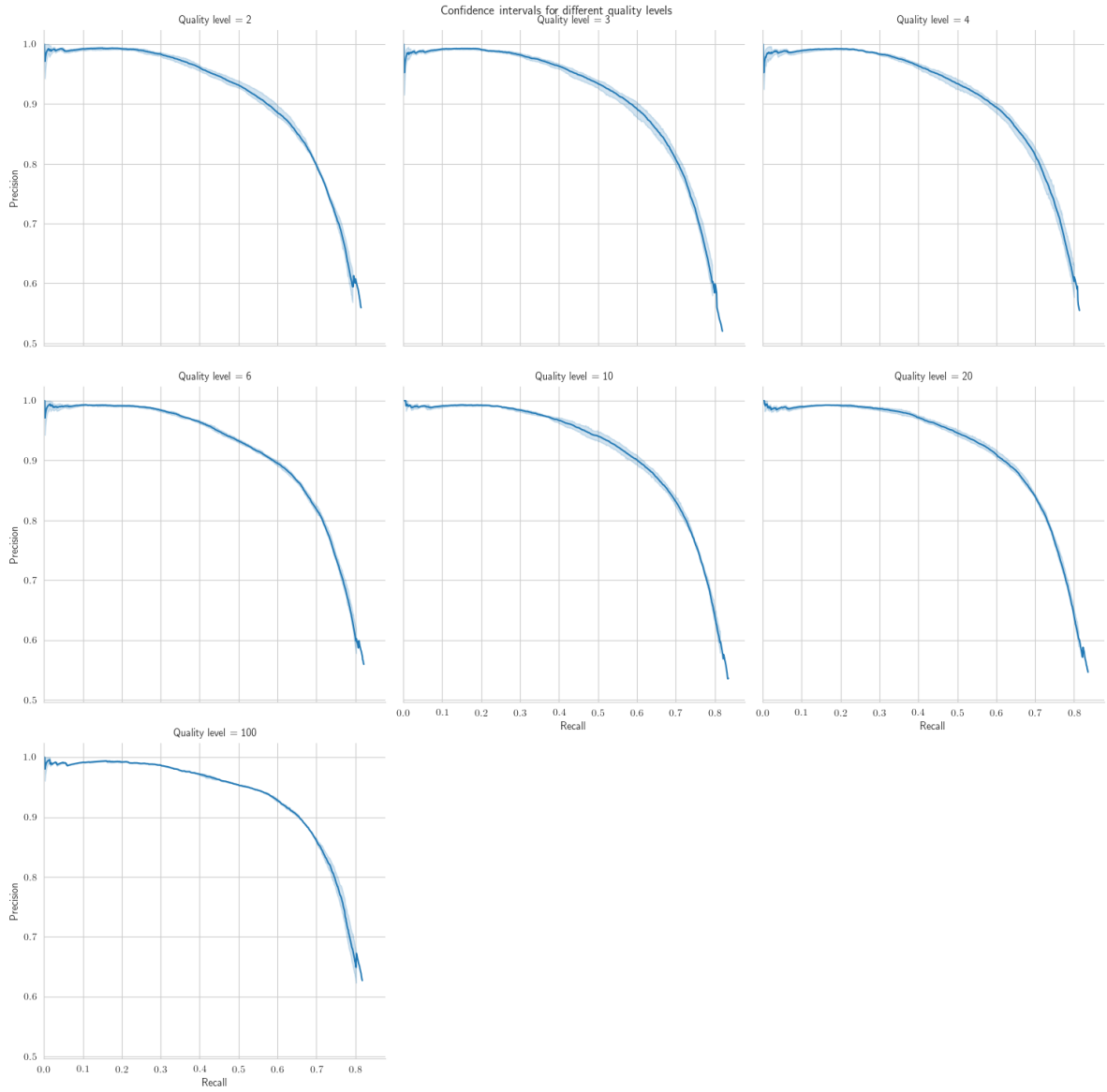
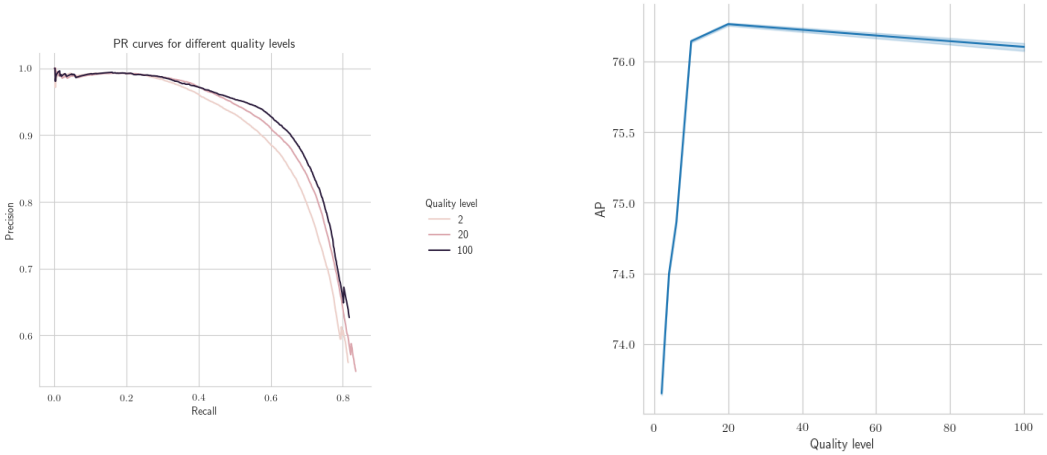


Figure 4.6: The confidence intervals of the precision-recall curves

Data quality	Avg file size [kB]	# images
original	147.17	89
20	24.75	530
10	18.18	722
6	15.38	854
4	14.11	930
3	13.46	975
2	13.13	1,000

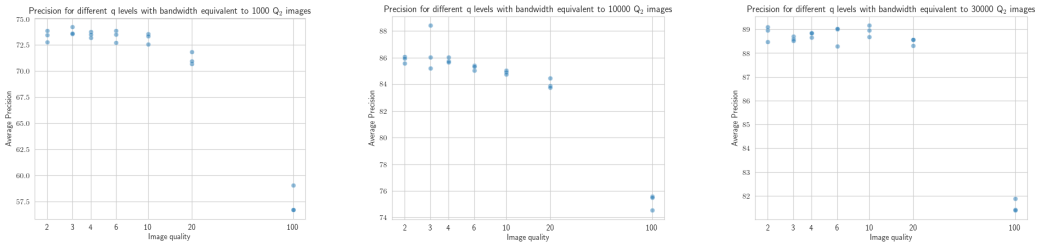
Table 4.1: Amounts of images for equal bandwidth



(a) The precision-recall curves for different compression quality levels

(b) The AP for different compression quality levels

Figure 4.7: The effect of image compression on training performance



(a) The AP for different quality levels with equal bandwidth usage (1000 Q_2 images)

(b) The AP for different quality levels with equal bandwidth usage (10000 Q_2 images)

(c) The AP for different quality levels with equal bandwidth usage (30000 Q_2 images)

Figure 4.8: The effect of image compression on training performance at different scales

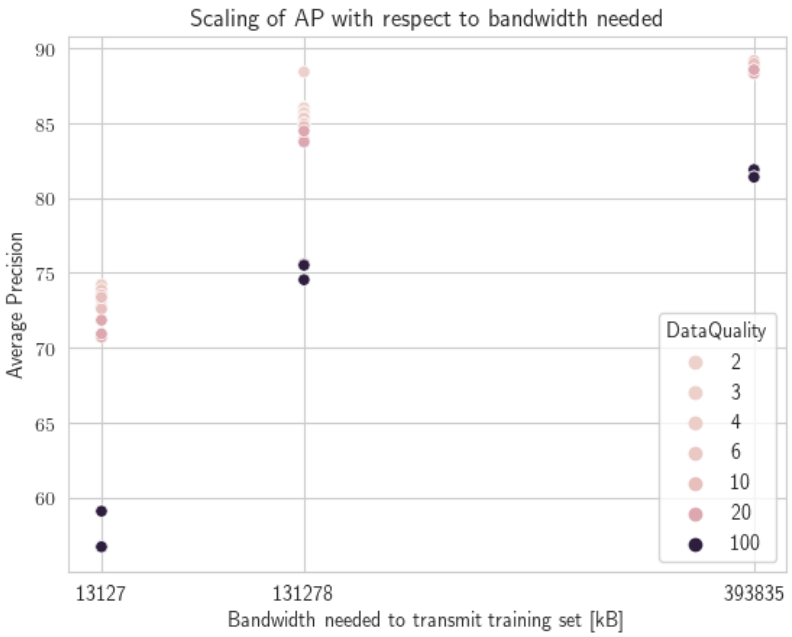


Figure 4.9: Overview of the scaling with respect to bandwidth

Data quality	Avg file size [kB]	# images eq. 1000	# images eq. 10000	# images eq. 30000
original	147.17	89	892	2,676
20	24.75	530	5,304	15,911
10	18.18	722	7,222	21,667
6	15.38	854	8,538	25,615
4	14.11	930	9,301	27,904
3	13.46	975	9,750	29,249
2	13.13	1,000	10,000	30,000

Table 4.2: Amounts of images for equal bandwidth (with larger datasets)

respect to the bandwidth available, or in mission terms: what happens when the mission has gone on longer and more images have been sent? Is the best approach still to send the images with quality level 3, or does the optimum shift?

This is why the same experiment[8] was also done for different amounts of images, as shown in table 4.2. The results of these experiments are shown in figure 4.8. Furthermore, the average of all experiments is shown in figure 4.9.

4.7. Prediction of further training

With these results, as well as with the works described in 3.2, conclusions can now be made about both the performance of models trained on compressed images with more images than available, as well as the number of images needed to reach wanted results. Furthermore the final accuracy of a model (if infinite annotated images would be available) can be predicted.

In order to do this a model for the learning curve has to be derived for each quality level (as the quality level influences this curve, as seen in this chapter). Instead of using the error, as in equation 3.1, the AP will be used, resulting in equation 4.1.

[8]The image sets with amounts of images larger than 11k could not be done using completely separate sets, due to limited availability of training data. Instead, they were separated as far as possible. For the actual images used for every experiment see appendix G

$$\begin{aligned} AP(n) &= (100-a)+bn^c \\ &s.t. \\ 0 \le a &\le 100 \\ c &< 0 \\ b &> 0 \end{aligned} \tag{4.1}$$

Where AP is the expected AP of a model, n is the number of samples in the training set, and a, b, c are the parameters. It can be seen that for $n \to \infty$, $AP(n) \to (100-a)$, meaning that $100-a$ is the maximum accuracy of a model. This is why from now on a will also be called intrinsic error (also referred to as Bayes error in literature).

To fit the sample points of each quality level on the model in equation 4.1, non-linear least squares was used, as described in [4, 9, 41]. This was done using the `scipy.optimize` python module. This module implements different curve fitting and optimization tools. In order to use this module the equation given in 3.1 is rewritten in the standard from. This standard form requires the function to give residuals, or the error in the fit of a sample. Furthermore it requires constraints on the parameters (a, b, c) to be given in the form $lb < p < ub$, with lb being the lower bound, ub the upper bound and p the parameters. Since one of the constraints in 4.1 is $0 \le a \le 100$
a numerical trick is used where because of limited resolution
If $p = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, then the bounds can be defined as $lb = \begin{bmatrix} 0 \\ -\epsilon \\ -\infty \end{bmatrix}$, $ub = \begin{bmatrix} 100 \\ \infty \\ \epsilon \end{bmatrix}$, with $lb < p < ub$ and ϵ the machine error (since the calculations are done in float64, with a precision of 53 bits, $\epsilon = 2^{-53} = 1.1 \times 10^{-16}$.

The curve is then fitted using non-linear least squares, resulting in the optimization problem given in 4.2, with $y(n)$ being the measured points.

$$\begin{aligned} &\min_p \left(y(n) - \left((100-a) + bn^c \right) \right)^2 \\ &s.t. \\ &lb < p < ub \end{aligned} \tag{4.2}$$

This problem is then solved, as said using the `scipy.optimize` module, for all the aforementioned quality levels. The error in the model fit, given by equation 4.2 is zero for all the learning curves, since the number of parameters is equal to the number of observations (3). Since each quality level is trained on 3 times, this results in 3 different values for $p = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ for each quality level considered, which give an estimation of the learning curve of the Faster-RCNN algorithm using images of that quality level. Using these results, assuming multivariate Gaussian noise for the parameters (a, b, c), the co-variance (since the parameters are not independent) matrix of the parameters is estimated, with which the 95% confidence band for the predicted curves is calculated. The confidence bands are shown in figure 4.10. It can be seen that the further the prediction is extended, the wider the confidence interval gets. This, of course, makes sense as an error in a parameter results in a larger error in the $AP(n)$ as n increases. It can also be seen that up to at least the equivalent bandwidth of 90000 Q_2 images, uncompressed images results in a worse performing model. For Q_3 it can be seen that there is a large variance around the 10000 Q_2 image equivalent, which is caused by an outlier in the measurements.

In figure 4.11 it can be seen that, as expected, there is a crossover point where higher quality images result in a better model. This crossover point is shown for Q_2 with Q_{20} images. While it is expected that at a further point, with n increasing, the same happens for Q_{100} with Q_{20} images, more samples are needed to conclude this.

From the intrinsic error and its' variance, the final accuracy of a model can be found, with a confidence interval. This final accuracy is shown in figure 4.12. While it can be seen that the best fit intrinsic error decreases with the quality level going up, it can also be seen that the confidence in the estimate decreases, with an extreme error bar for the original quality images, which means that no conclusive results can be

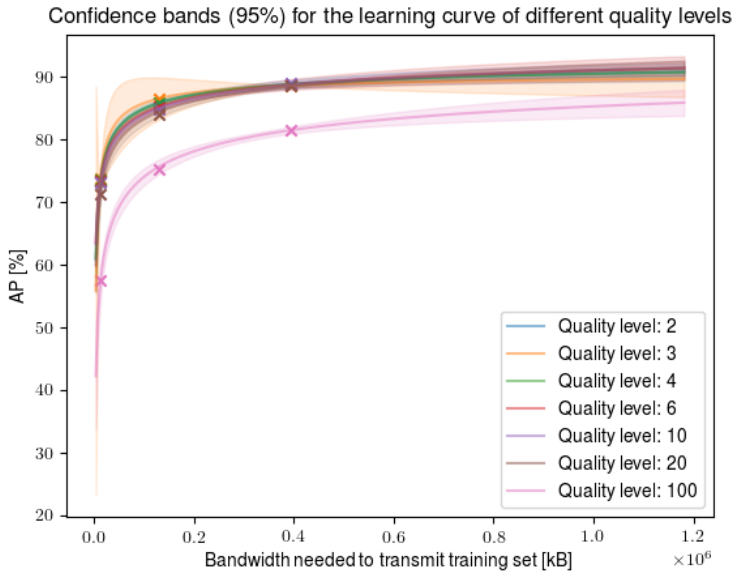


Figure 4.10: An overview of the 95% confidence bands of the predicted learning curves

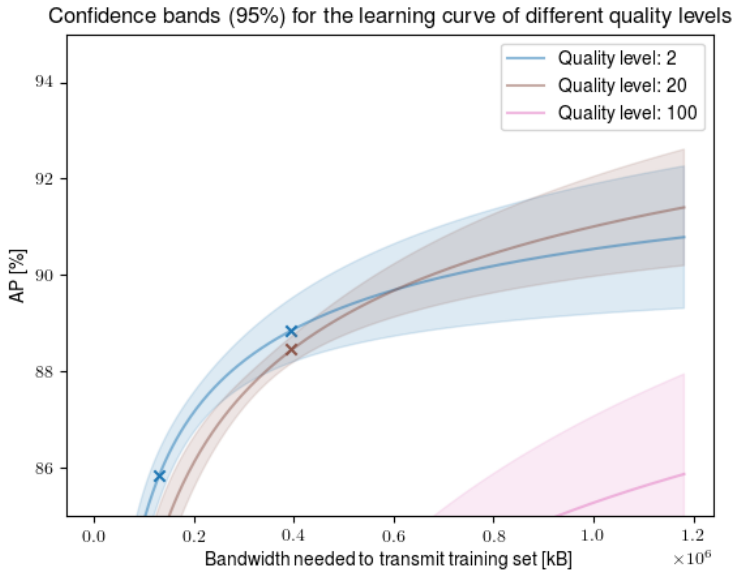


Figure 4.11: A zoomed view of the 95% confidence bands of the predicted learning curves

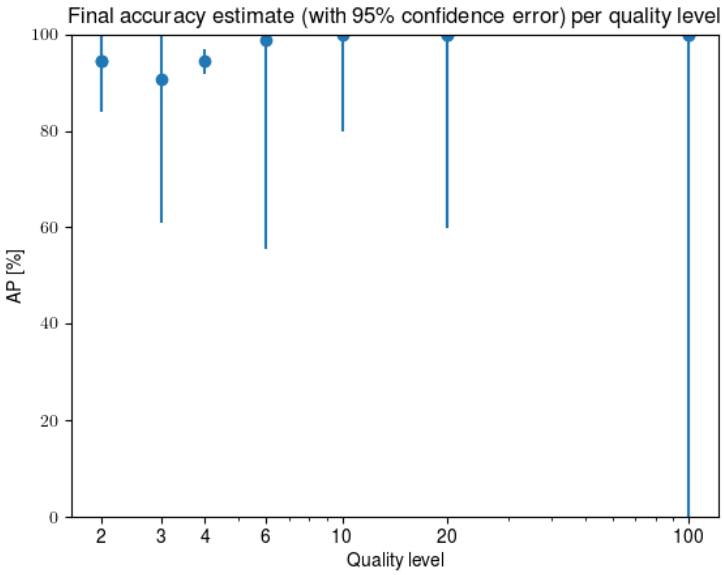


Figure 4.12: Final accuracy estimate

gotten from this. To get conclusive results more samples would be needed, especially for the higher quality levels.

The conclusions that can be drawn from these results will be discussed in the next chapter.

5

Conclusion

In this thesis an efficient manner of gathering a object detection dataset for training a CNN was developed. This was done using image compression techniques to reduce the size of training images.

In contrast with the "common knowledge" that is often applied in the field of object detection, using the highest possible image quality does not transfer to the best trained network when gathering a dataset of satellite images, since the main constraint is the bandwidth available for transmitting images, where "normally" the largest constraint is the amount of man-hours spent on annotation. Compression of training images with JPEG-XR quality level 2 during the gathering of training images results in a better "bandwidth-efficiency", in the dataset used for this research at least up to 30000 images. It was also found that when more bandwidth is available and thus more images can be added to the training set, the optimal amount of compression tends to decrease. This results also lies in line with the result that the "final accuracy" (the predicted accuracy of a model trained on an infinite amount of images) of the models tend to improve with better image quality. From this it can be concluded that for the optimal approach the training images should be compressed as far as possible at the start of training, to then decrease the amount of compression as the mission progresses and more cumulative bandwidth is available.

While the results are all based on the dataset used in this thesis - the airbus ship detection challenge dataset - the results are likely comparable for different problems, especially comparable problems.

After concluding the qualitative effects of image compression, the question remains what this would hypothetically save on a mission. To do this an pretend mission is created. In this mission the goal is to detect all ships within the images taken by a satellite with a 90% accuracy[1]. Of course, a more realistic scenario would be having multiple satellites to cover the whole earth, but this results in complex mission planning and overlap between their paths. The scenario sketched is always less optimal for one satellite as the training costs can not be shared between multiple satellites.

Three approaches are considered:

- not using a CNN on the satellite, thus sending all images to earth
- training a CNN using non-compressed images
- training a CNN using compressed images with quality level 2

Since in table 2.1 it was already concluded that the largest amount of power is spent on transmission, the amount of transmission needed is used to "score" the approaches. The score will be calculated for different mission durations. Some assumptions will be made: the satellite is assumed to be in a 90-minute orbit, with a ground resolution of 1.5m/px[2]. It will also be assumed that there is no overlap between images. Furthermore it will be assumed that the imager produces 768x768 px images to be able to use the results from the dataset used in this thesis. It will be assumed that the CNN approaches only send the location of the ships. For simplicity it will be assumed that transmitting such a location uses a negligible amount of data, compared to gathering the dataset. It will also be assumed that 20% of images contain a ship.

[1] For the accuracy AP will be used as a metric

[2] Since the dataset used in this thesis has that resolution

Lifetime	No CNN	CNN uncompressed	CNN compressed
1 orbit	4.88GB	Still training	Still training
1 day (16 orbits)	78.1GB	23GB	3.4GB
31 days	2.36TB	23GB	3.4GB
1 year	27.8TB	23GB	3.4GB

Table 5.1: Data transfer at different time scales for the different approaches

Lifetime	No CNN	CNN uncompressed	CNN compressed
1 orbit	0.2	54.26	71.28
1 day	0.2	76.94	86.59
31 days	0.2	90.12	92.37
1 year	0.2	94.63	93.64

Table 5.2: Accuracy of the different scenarios with 10MB/orbit of bandwidth

Using the assumptions, it can be found that approximately 34787 such images are taken per orbit. Using table 4.1, sending all images without compression results in sending 5119651kB per orbit, or 4.88GB.

In order to reach 90% accuracy for the CNN a dataset with the equivalent of 367744 Q_2 images with ships is needed[3] (32809 original quality images), which is equal to 4828478 kB, or 4.6GB. Since not every image will contain ships, the total data transfer needed is 23GB.

In order to reach the same accuracy with the lower quality Q_2 images, the equivalent of 53959 Q_2 images is needed, which is equal to 708482 kB, or 0.68GB. Once again, not all images will have ships, so the total data transfer needed to get the training set is 3.4GB.

From these calculations it can easily be seen that even in a relatively short amount of time, with the wanted accuracy, it makes economical sense to use a CNN on-board of a satellite to perform object detection in space. In table 5.1 the data usage of the considered scenario's is shown. If the amount of satellites in the mission would be increased, only the amount of data needed by the approach without any CNN would increase linearly, since the CNN can be shared between the satellites.

If instead of an accuracy goal a bandwidth budget is assumed, where a more realistic down-link budget of 10MB/orbit is assumed, it can also be calculated what the expected accuracy is with the approaches using a CNN at different points in time (ignoring the cost of updating the model as it is relatively small). This is shown in table 5.2. Since the 10MB bandwidth available is only a small part of the total amount of data gathered within an orbit, this results in a low theoretically possible AP for the non-CNN approach. It can also be seen that for missions shorter than 1 year the compressed approach outperforms the non-compressed approach, after which the uncompressed approach is superior (in this specific case).

[3] Using the equation of the best fit

6

Discussion & Recommendations

While the results clearly show the approach discussed in this thesis results in a more efficient way of object detection from satellite images than an approach where no CNN is used, the research is not broad enough to conclude it is the most optimal approach. Only the Faster-RCNN architecture was researched, while other architectures may show different results. Furthermore only JPEG-XR compression was used on the images for the training set, while other compression algorithms may give different results. The method is also tested on one dataset, where the results might be different for different datasets. All these factors should be researched further.

Furthermore, due to only having 9 data points on the learning curves per quality level in the graph shown in figure 4.10, the confidence bands are relatively big, resulting in large errors in the final accuracy estimate. More points on the learning curve should be created (by training the CNN with different amounts of images) to reduce the size of the error and be able to draw better conclusions what effect image compression has on final accuracy, if any.

Another unknown is what can be considered a comparable dataset for a certain mission, that can be used to train the CNN before launch. This is obviously dependent on the specific mission, however we expect that general conclusions can be made regarding whether different imagers, ground resolutions and image processing have any effect on the usability of a dataset from a different source.

A problem that the approach discussed in this thesis may run into is the labour intensiveness of annotating images for the training set. A solution for this problem should be found, but is also researched by other parties as they have the same problem. Many different approaches exist that can reduce the labour intensiveness of this process. One example would be to use a CNN[1] to pre-annotate the images, after which a human would only have to check the annotations.

An improvement that can still be added to the approach discussed in this thesis is smart image selection, where images that are more "relevant" to training the CNN are given higher priority to be transmitted to earth. A similar approach is used in other fields where an abundance of data is available, such as by the self-driving development department of Tesla [2], where "similar" images to a case that is not solved well by the CNN can be found within the fleet (in Tesla's case) to focus the training set more on that case. An example on the dataset used in this thesis would be that it is found that dykes are often annotated to be ships by the CNN. To solve this the satellite is ordered to sent similar images to an image of a dyke, so that more dykes can be added to the training set. This can of course be implemented in many different ways. Another approach would be to send only images with ships, such that the "waste" of bandwidth by sending images without ships is reduced, however this approach might also run into a loss of generalisation, where certain types of ships might not be send anymore, since those ships are not considered ships and never added to the training set. One way to solve this would be to also send random images, where for example 90% images are send for the training set with ships found by the CNN and 10% random images.

Another factor not yet considered is the compression of the images to perform inference on. In this thesis the inference was always done on original quality images, however a CNN trained on compressed images may perform better on compressed images. Not needing to perform the compression would however result in less compute power needed on the satellite.

[1] Or a more power-hungry ensemble of CNNs

[2] `https://youtu.be/Ucp0TTmvqOE?t=7556`

For actually implementing the research done in this thesis it is recommended that one trains a CNN before launch with a dataset as comparable as possible to the images produced by the satellite. In case the characteristics of the missions imager are different from the imager that produced the dataset, it is recommended to compensate for those characteristics. During the mission it is recommended to start training with a low quality level, for example 2, until the accuracy of the CNN "plateaus", where marginal gains are made by adding more image. At that point we would recommend to increase the quality level to original, with which it is expected the highest final accuracy can be reached. The further the mission progresses the more important smart image selection, as discussed in this chapter, is likely to become, therefor it is also recommended to keep increasing the test dataset, to find potential issues the CNN has.

A different approach, that was briefly mentioned in the introduction, is to use unsupervised or semi-supervised learning combined with training in orbit, to avoid having to build a training dataset on earth. This would result in a completely different problem with different challenges, such as compute power and storage in orbit. While this approach is obviously more bandwidth-efficient, it is also harder to analyse the performance of the model and validate it is performing as expected.

While it was mentioned in the literature review that weights could be used during the fitting of the samples on the learning curve, this was not done because of the small amount of samples. Only 3 different amounts of images (1000, 10000, 30000) were used, meaning using the weights described in [9] makes little sense.

A

Computer-vision techniques

In this appendix the different types of computer vision techniques that are researched will be discussed. There are so called "DL" techniques and other techniques. The other techniques are shortly discussed, but DL is the main focus.

A.1. Deep Learning

The term DL is used for many different algorithms. In this context DL will refer to object detection DL algorithms, that take an image as input to then output the predicted objects within that image. To do this the algorithm has to be trained by supplying many different images with the objects annotated, such that the algorithm can determine what is and is not how the objects to be found look.

First the basics of DL are discussed. After this the most relevant examples of algorithms are explained.

A.1.1. Basics of DL

Since DL is a very broad term, with many different sub-disciplines, this research will focus on architectures with multiple layers of Neural Networks (NNs). This is the most relevant to image classification, specifically CNNs. A NN is a collection of nodes, that are interconnected. Each node performs a basic mathematical function (such as a multiplication). The interconnections between the nodes are weighted. These weights are tunable. By tuning these weights, the NN can be used to fit complex mathematical functions.

Most commonly, the nodes are organised in layers, where each layer connects only to the next layer. This intuitively makes sense, as each layer can add a level of abstraction. If more than 6 layers exist, we speak of a "Deep" NN. Deep NNs are responsible for a lot of the leaps in Machine Learning (ML) research, specifically in complex tasks such as Natural Language Processing (NLP) and image recognition.

For general networks, the input of this NN is a vector of all inputs. The output is then a vector with estimated probabilities for each class that the network is trained on. This way the NN can be seen as a fit on the function of input to output. This is where the term "universal fitting function", that is sometimes used to describe NNs comes from.

Of course, for the learning to work an optimisation method to tune the parameters has to be used. This is done by "back-propagation" using an optimiser. A data-point (an image, a second of spoken language, etc.) is fed to the network. The network then outputs a value from 0 to 1 for each classification. The error is then calculated. After this the error is minimised, by tuning the weights using the optimiser (most of the times with a "learning rate" hyperparameter). This way the NN "learns" from the fed data. This process is called the learning process. Different optimisers exist, which will be discussed later.

The weight that is learned can have different precisions, depending on the application. For high-precision training floating point representations are used, while for edge computation it has been shown that 8-bit or 16-bit integers can also work reasonably well (with a drop in top-1 accuracy of about 1-2% with Quantization-aware training). [1]

After this learning process is done for the complete dataset, for multiple epochs (times the complete dataset has been fed to the network) the training is complete. After this the network is tested on a sepa-

[1] https://github.com/tensorflow/tensorflow/tree/r1.15/tensorflow/contrib/quantize#quantized-accuracy-results

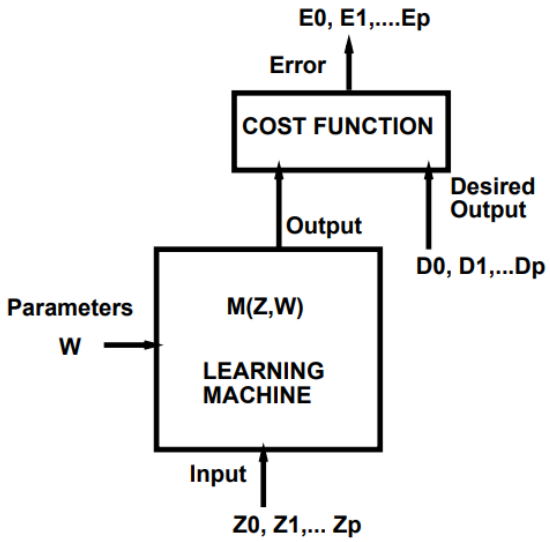


Figure A.1: Graphic overview of the gradient descent optimiser [35]

rate dataset (the test data). During this the weights are frozen, so that an accurate estimation of the accuracy can be made.

A.1.2. ML optimiser techniques/Learning methods

Since the optimiser is responsible for adjusting the weights of the NN, it is an important part of any design. There are several approaches discussed in the literature, of which most are based on Gradient Descent. An overview of what the weight adjustment looks like can be found in figure A.1

Gradient Descent

Gradient Descent is not a new method, and has had success in other fields as well. The gradient descent formula for NNs is given in [35]:

$$W(t) = W(t-1) - \eta \frac{\delta E}{\delta W} \tag{A.1}$$

Where η is the learning rate, which adjusts how far the weights are moved in each iteration. There are then several approaches to choosing the learning rate. The most simple case is tuning a scalar learning rate. Other approaches are the Newton/Quasi-Newton approaches where the learning rates are chosen by calculating the Hessian, and the learning rate is thus a diagonal matrix (meaning each parameter has its' own learning rate).

There are also several approaches as to when the weights should be adjusted. We can adjust the weights for each datum (stochastic/online learning), or we can use "batch learning", where the average gradient is calculated for a batch of data.

Batch learning is generally slower, as the redundancy in batch learning is much higher (in the extreme example of having 2 exact same data points in the set, the exact same gradient has to be calculated twice for one training step). Stochastic learning also has the advantage of being able to escape local minima, due to the "noise" on the estimated gradient. [35]

Batch learning however also has advantages: some tricks to speed up the computation are only effective on batch learning and batch learning converges closer to the (local) minimum. The convergence of stochastic learning can however also be improved, by annealing[2] the learning rate. With batch learning the second order derivative can also be computed, which can help in the estimation of the minimum of the function.

An important point to note on the optimisation is that the convergence is not as important as it might seem, since overfitting is more of a problem. The hope is that the generalisation of the network is good enough that the convergence error is no problem.

[2] https://en.wikipedia.org/wiki/Simulated_annealing

Momentum
One issue with Gradient Descent is that it has trouble with descending in one direction more strongly than other directions. A proposed method to solve this issue is the momentum approach, where the descent has momentum that is changed every iteration. This results in being able to quickly converge in one direction. [45]

Nesterov accelerated gradient
The Nesterov Accelerated Gradient (NAG) method is an improvement on momentum, where the gradient is also calculated for the next iterations' position. This can be done since the speed is known, thus the position in the next iteration is known. This results in a more accurate gradient direction and thus faster convergence.[55]

AdaGrad
AdaGrad adjusts the learning rate for each weight, where weights that are associated with more data get a slower learning rate than other weights. This results in a more robust convergence of the NN. The learning rate of weight w_i is divided by the sum of the squares of the gradients in the direction of w_i up to the current moment. This however also comes with a disadvantage, since the sum of squares can only increase, thus the learning rate can only decrease. This results in not being able to learn new knowledge after some time, since the learning rates all went to zero. [13]

AdaDelta/RMSProp
AdaDelta [60] tries to solve the diminishing learning rate, by restricting the sum of squares to a certain window, resulting in a constant learning rate with infinite time. RMSProp is an adaptive learning rate method proposed by Geoff Hinton, which results in the same algorithm as AdaDelta. [57]

Adam
Adaptive Moment Estimation (Adam) [29], in addition to the decaying gradient of Adadelta, also stores a diminishing momentum like Momentum. This results in favouring flat minima.

AdaMax is an extension of Adam, where instead of using the ℓ_2 norm to calculate the "velocity" (the gradient), it uses the ℓ_∞ norm.

Nadam
Where it was previously discussed that NAG is superior to "normal" momentum, the same can be applied to Adam. This is why Nadam [12] was proposed, which combines NAG with Adam.

AMSGrad
In limiting the sum of gradients to a window (or diminishing them) a problem in convergence was introduced for some forms of functions. [26] It is proposed to solve this issue by instead of calculating the maximum of past squared gradients, instead of the sum over the window. This appears to improve the performance for certain datasets, especially object recognition.

A.1.3. Information Feeding
Since the amount that the weights of a NN change is linearly dependent on the magnitude of the error, the training of the network happens the fastest when the most "unexpected" example is fed to it. [35] The most basic approach to this is randomly shuffling the input data. One way to improve this over random shuffling of the input data, is to rarely feed data belonging to the same class after each other. Another method of improving this is called emphasising. If a datum results in a large classification error, it is clear that the network does not handle that datum well. This datum can then be emphasised, by feeding it to the network more often than other data.

A.1.4. Convolutional Neural Networks
CNNs are a type of ML that is inspired by biological vision, specifically in cats. The vision of cats was researched in [24]. It was found that there are cells in the cortex which perform "spatial pooling" which is thought to be crucial in the vision. From this research CNNs were inspired.

CNNs are the most used ML algorithms for images. This is because of the property that they are space-invariant, which means it does not matter where a feature is in the image, nor in which orientation the feature

is. [61] This is also important for satellite imagery, as the location and orientation of the features in images are arbitrary, because of the rotation of the satellite itself. This is why in research CNNs proofed to perform better on images than Support Vector Machines (SVMs). [33]

CNNs also retain information about the location of pixels with respect to each other because of the spatial pooling. This is first described in [32], where the convolutional layers are proposed. Having this spatial information preserved is crucial for image recognition.

Convolution

Since with large input data such as images having many fully connected layers results in an infeasible amount of parameters, both due to memory usage and the susceptibility to overfitting, a smarter solution is needed. This is why the convolution operation is used. The convolutional layer can be seen as a dot product of local features with trainable weights. The number of filters (sets of weights) is a hyper-parameter that can be tuned. For example: using a 32x32 input with 3 filters will result in a 32x32x3 output of the convolutional layer. A visualisation of this process can be found on the Stanford CS231n page[3]. Usually padding is also added around the edges, for the convolution to work with the sides of the image as well. If this is not done the sides of the image become largely irrelevant to the classification.

Activation

After a convolution layer the outputs of this layer have to be normalised in some way. There are different methods to do this, some based on biological data and some based on computationally efficient performance.

The traditional way to do this is using the hyperbolic tangent function to map the output value from -1 to 1.

Another way is using a rectifier. This rectifier is defined by $max(0, x)$. This is computationally very easy and in practice also performs at least as good as the tanh function [19][31]. Another name for this is Rectified Linear Unit (ReLU). There are also other variants of the ReLU function, such as the leaky-ReLU

Pooling

Pooling is the operation which reduces the dimension of the data in the NN. This is done by applying some norm function over a window of the previous layer. It is found that in practice Max-pooling is the best performing method [52]. This operation loses information that was in the original image, but also reduces memory usage. Larger windows result in more information being lost. It is most common to use small window sizes, and use multiple pooling layers, with other layers in between them.

Fully Connected

Fully Connected (FC) layers are similar to convolution layers, however they do not operate on local subsets of the image, but on the full image. They can thus be seen as a matrix multiplication, with added bias.

Output Layer

The very last layer of the neural network is called the output layer. Hopefully all features of the input data have now been extracted, and returned into a vector. This is the input to the output layer, which from these features produces a classification. The output of this layer is a vector with a probability for each class. While the output layer gets a special name, it is just a FC layer. As will be discussed later, the output layer does get a special function with transfer learning.

Issues with CNNs

One big issue that conventional CNNs have, is that they classify which class of object is in the image, instead of finding all objects with their respective classes in the image. The output vector of the CNN is a list with probabilities of each class in the image being present. This means that to find the location of an object within an image the CNN has to be run over many regions within this image, where the region selection is still a problem to be solved. The classic (naive) approach to this is to have a sliding window moving over the image, with a step size of a certain amount of pixels. Different approaches (such as Regional Convolutional Neural Network (R-CNN), Fast R-CNN, You Only Look Once (YOLO)) to circumvent this issue have been attempted, which are described later.

[3] `http://cs231n.github.io/convolutional-networks/`

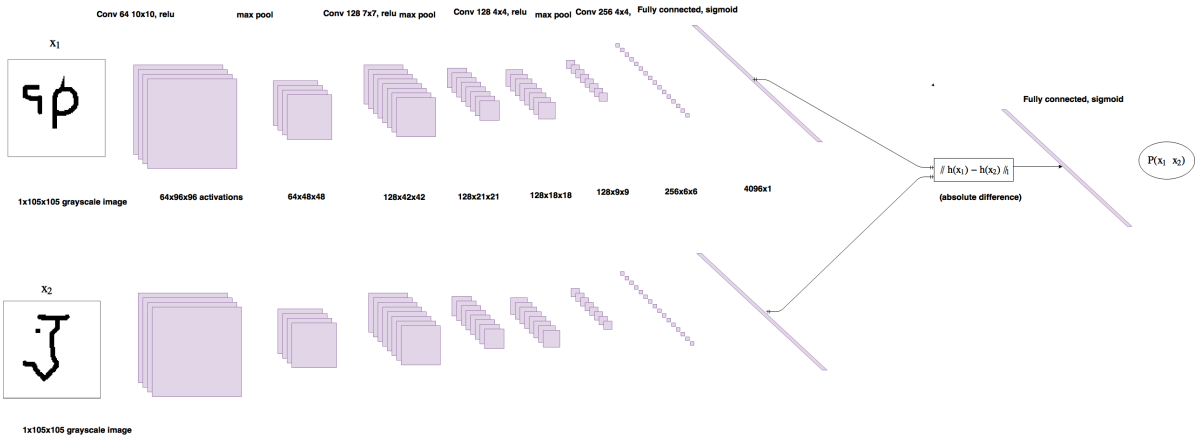


Figure A.2: Graphic overview of the OneShot method

A.1.5. OneShot classification

As described, CNNs require a lot of data, which can be expensive (or even impossible) to obtain. This is why efforts are made to reduce the data needed for classification. One such an approach is OneShot classification. [30] It uses the same approach as CNNs to extract features from an image, but instead of training a model from known images, it compares an image to 1 (or a few) known images. It does this by calculating the L2 norm between the features of the known image and the new image, for every image in the known image-set. The higher the similarity, the lower the value that is to be expected from this norm. Then the classification is the class with the highest similarity image in it. This is shown schematically in figure A.2 [30].

A.1.6. Regional Convolutional Neural Network

The R-CNN approach consists of 4 main steps: [18]

1. Select regions to be analysed
2. Warp these regions into squares
3. Compute feature vector using CNN
4. Classify using a SVM (see appendix B)

This results in only analysing a specific region once, where only the fourth step is done repeatedly for each class. This means it is computationally more efficient, especially for extraction of multiple classes from an image.

Selective Search

In order to use this approach a computationally cheap way to select candidate regions for features is needed. The way proposed in [18] is to use Selective Search, which is described in [58].

Selective Search starts by over-segmenting the image. The segmentation algorithm used is proposed in [15]. The segmentation divides the image into different regions, where within each region the image is similar. Over-segmentation is done so that no features are missed. This results in a high recall ratio, but this is not a problem since the segmentation is just the first step of the algorithm.

After the segmentation bounding boxes are drawn around all these regions. The regions are then merged hierarchically, where similar[4] regions are merged. In R-CNN the regions are (usually) merged until 2000 remain.

Classification

After the most likely regions are determined by the Selective Search, the regions have to classified. For this they are warped into squares. After this the square regions are fed into a CNN, to extract the features in the region. The features are then fed into a specific SVM [10] for each class to be analysed. The SVM returns a true/false assessment of that specific class of object being in the region.

[4] in colour, size, texture and shape

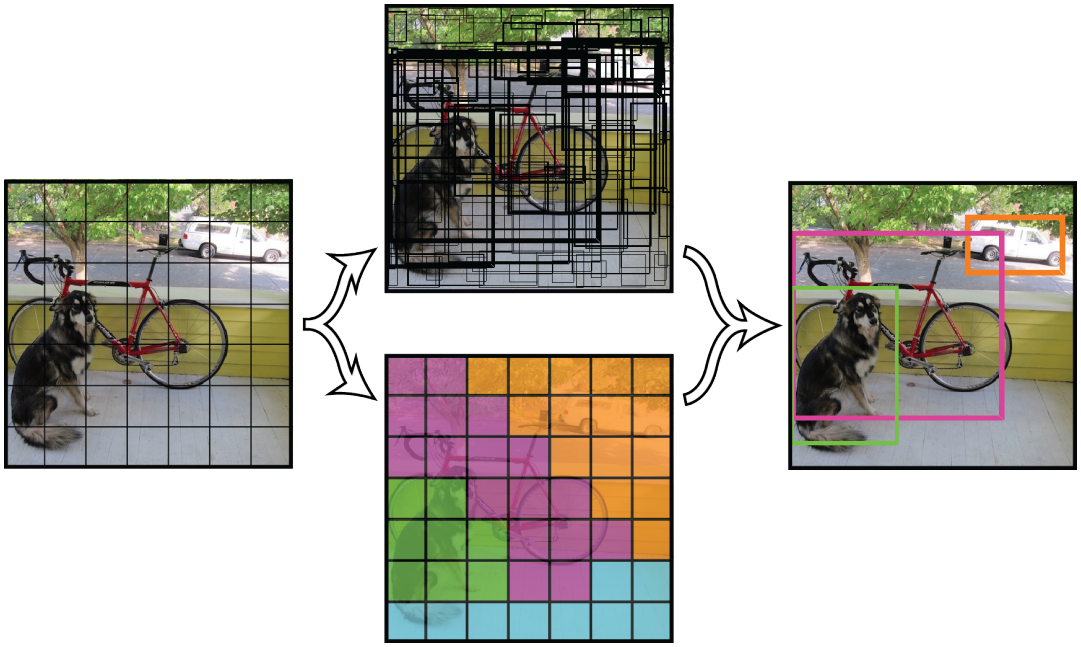


Figure A.3: Graphic overview of YOLO [48]

A.1.7. Fast Regional Convolutional Neural Network

Fast R-CNN tries to solve the performance issues R-CNN has, due to needing to analyse 2000 regions. It does this by reversing the operations. First the complete image is fed to a CNN, after which the regions of interest are found by Selective Search. This significantly improves the test performance, as well as training speed. [17]

A.1.8. Faster Regional Convolutional Neural Network

Fast R-CNNs performance is still not good enough to be computed real time, as the Selective Search algorithm impacts the performance. This is why Faster R-CNN attempts to use a different algorithm to find the Regions of Interest (RoI). It is proposed in [49]. It uses a NN to find the RoI, instead of the selective search. This NN is called the Regional Proposal Network (RPN). These RoIs are also called anchors. The RPN is trained using the IoU (see chapter 4.1) score of these anchors.

A.1.9. You Only Look Once

YOLO [48] has a different approach to R-CNN-like architectures. It divides a test-image into a SxS-grid, after which for each of the squares on the grid YOLO computes B bounding boxes (which can be larger than the square) and the class probabilities in that square. This is shown graphically in figure A.3. The final classification is then the multiplication of the bounding boxes with the class probabilities. A threshold is applied to this value, to draw the final classifications.

Because of this architecture YOLO is significantly faster than other methods, with YOLOv3 performing about 3 times as fast as RetinaNet [38] for comparable accuracy [47]. Since YOLO takes the complete image into account during its' classification it is more reliable in filtering out the background [48].

The drawback of this approach is that it is not very accurate, compared to other approaches.

A.1.10. Single Shot MultiBox Detector

The Single Shot MultiBox Detector (SSD) [39] architecture is similar to YOLO in that it tries to improve inference speed.

The first step of an SSD network is to use VGG-16 to extract a feature map. After this bounding boxes should be predicted. It does this by having a default set of bounding boxes with different dimensions. An image is divided into different resolution grids, where in each location of the grid the bounding boxes are drawn, which are then adjusted using a NN. Since there are different resolution grids, different scales of objects can be found. The size of the bounding boxes also scales with the size of the grids. After the bounding boxes are predicted (with a lot of false positives) the objects in the bounding boxes are classified, where no object is also

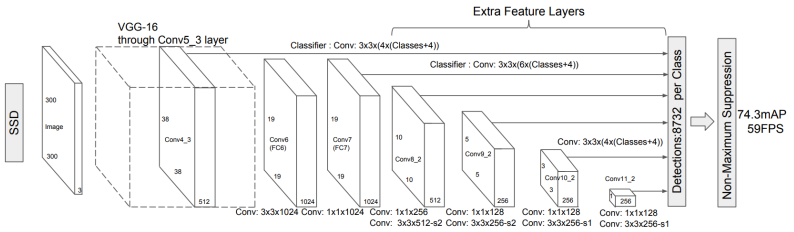


Figure A.4: Graphic overview of SSD [39]

a possible classification. The highest probability boxes are then the output of the network.

Since there are many false negatives in the bounding box prediction, training the network with all false negatives would result in bad training. This is why the ratio of negatives to positives is limited.

SSD has been shown to be outperformed by YOLOv3, both in accuracy and in inference speed. [47]

A.1.11. Generative Adversarial Network

The Generative Adversarial Network (GAN) architecture was first proposed in 2016 [20]. They use game theory (sort of) to generate data that belongs to a dataset. This is the type of network that powered DeepFakes as well as sites such as `thispersondoesnotexist.com`. The network works by having two NNs, of which one is the generator and one the discriminator. The generator tries to create images that pass the discriminator as real, while the discriminator tries to classify whether an image is real or not.

While these are not immediately useful for classification of images, attempts are being made to use this type of learning for Semi-Supervised learning, where a small amount of classified data can be used to train the neural network.

MARTA-GAN

One such attempt is the MARTA-GAN architecture. [37] The idea of this architecture is that it generates additional training data using the Generator (G), while the Discriminator (D) tries to distinguish these from the real images. This results in fake images which are close to real images, so that they can be used as additional training data. This results in needing less training data needing to be classified, thus can reduce the cost of training a model. The D is then used to classify the image, just like a classical CNN.

Deep Convolutional GAN

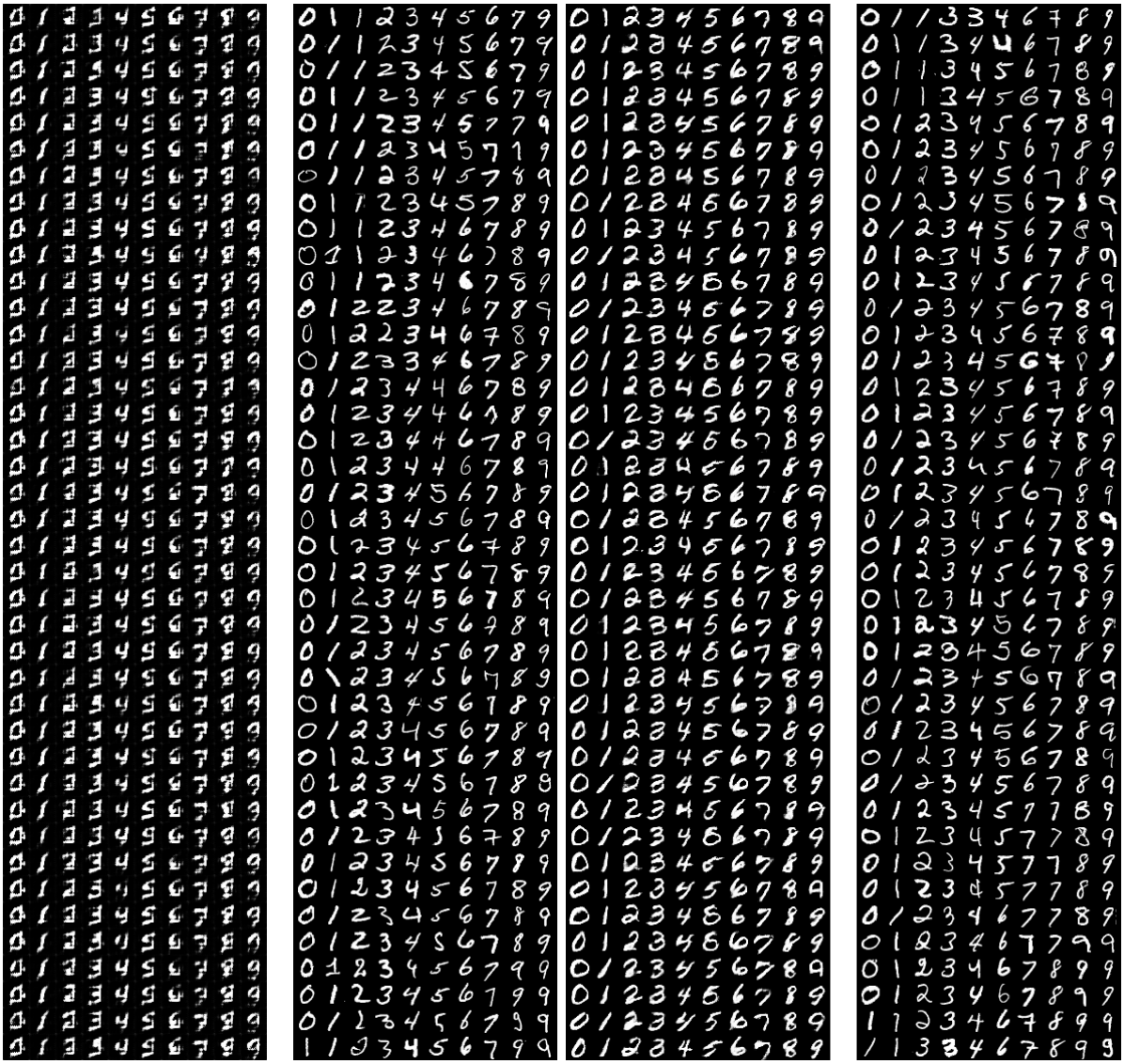
Deep Convolutional GANs works similar to MARTA-GANs. [46] This paper however, also describes a way to analyse the network and the representations it generates for objects.

Auxiliary Classifier GAN

The auxiliary classifier GAN[43] also works in a similar fashion. In this case the G tries to fool the D on a specific class. The generator "wins" if the Discriminator classifies the image as the class that the generator tried to fake. The discriminator can also classify an image as fake, instead of one of the classes. In this way the discriminator once again can be trained with very little data.

The results on the MNIST database [34] of the example AC-GAN given in [5] can be seen in figure A.5.

[5] `https://github.com/keras-team/keras/blob/master/examples/mnist_acgan.py`



(a) Results after Epoch 1

(b) Results after Epoch 9

Figure A.5: Results of the AC-GAN. Left are fakes, right are real images.

A.1.12. Specific Trained Nets

AlexNet

AlexNet is a NN that is developed by Alex Krizhevsky. It is a big part of why DL got big, when it won the ImageNet contest with a 10% lead on the second place. Its' large amount of layers is the reason it performed so well. This makes it computationally expensive, but using GPUs made this possible.[31] It is still used to classify images, but also as a component in certain architectures, such as GANs architectures.

GoogLeNet

The GoogLeNet NN proposes a new idea for a layer: the inception layer. It consists of multiple parallel convolutions, that are concatenated to form the output. The convolutions are of different sizes (1x1, 3x3, 5x5). This results in some "context" information being used, as well as details. Using this resulted in an error rate of 6.66% on ImageNet.

ResNet

The authors of ResNet argue that adding more layers to a NN should never decrease its' performance, as the layers could be identity, resulting in the exact same performance (accuracy-wise). However, in practice a decrease in accuracy with adding more layers. This has to do with the vanishing gradient problem. As the

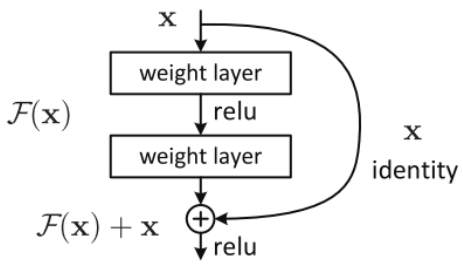


Figure A.6: Graphic overview of ResNet

gradient is back-propagated it is multiplied for each layer, resulting in a smaller and smaller value. ResNet tries to solve this problem, by using a new architecture. An identity bypass-function is added for each x layers. This then results in having at least the same performance for added layers. The architecture can be seen in figure A.6. [22]

This vastly improves the stability of training and the scalability of the network. Since adding layers will never make the performance worse (but does make the computation time worse) the depth of the network can now easily be increased to add accuracy or reduced to improve computation time.

A.1.13. Synthetic Data Creation

An important technique used in training ML algorithms is the creation of synthetic data. This is used to reduce the amount of annotated data being needed, by manipulating the existing data. A risk of this approach is over-fitting on the existing data, as it is re-used multiple times.

Well known techniques are mirroring the images, as well as rotating images (up to a certain maximum angle). Note that this should only be done if the situation could actually be encountered by the NN. Using mirroring on text does not make sense, as the NN does not have to classify mirrored text in a real use case.

Of course a GAN can also be seen as (a complex way of) creating synthetic data to train the discriminator.

Another approach to creating synthetic data is creating a 3D simulator and extracting data from the scenes generated by the simulator. [6] While this results in cheap data, it is difficult to create this simulator which produces the data.

A.2. Non-DL Techniques

Apart from the DL techniques considered, there are also non-DL techniques. While a completely ML-free solution is not considered for this research, as they do not generalise, the techniques used are still considered to be used in addition to the DL techniques.

There are two main approaches towards non-DL Computer Vision (CV). The first approach is using the form (or mostly the form) of the object to be found, the second approach is using colour (or mostly colour) as the identifier of the object.

A.2.1. Form methods

Using the form of objects seen in an image to recognise it is one of the oldest approaches to CV. Research goes back as far as 1985. [5] While this research does not propose a CV approach, it does needed research on how humans recognise objects. It also immediately indicates a problem with using only form: a zebra will look the same as a horse with only edges, so colour is still needed for some cases.

Filtering

If information about the features we are looking for is available, we can use this to filter other features away. The most simple example is that if we are looking for a red ball, we can remove all pixels with a red-channel value below x, where x is tunable. More complex filtering approaches are used for real life situations. All of these filtering approaches are very domain-specific. This results in large development cost, which is not easily ported to new products. This is why these are not further considered in this research.

[6] http://www.harrisgeospatial.com/Portals/0/pdfs/EAS/DavidGorodetzky_EAS18.pdf

Edge detection
One of the techniques that is almost always present in CV without DL is the edge detection. The idea is to find the boundaries of different features so that they can later be detected/classified. The most used way to perform edge detection is using a two-dimensional gaussian derivative on the image [6]. This is computationally efficient, because gaussians can be implemented efficiently. Edge detection is also used in some DL techniques.

A.3. Transfer learning

A commonly used technique in the training process of CNNs is called "transfer learning". In this technique a trained NN is taken, and then retrained for a new dataset. This is often done by freezing the first layers of the network and only training the last layers (mostly the very last fully connected layer) since only that part of the network is expected to be data-specific. This means less data is needed to get a well-performing CNN.

Often-times the NN is first trained on ImageNet [42, 54], assuming that the large amount of data (1.2M images) in the dataset help the pre-training step. This is found not to be true, and training on a smaller subset of the ImageNet is found to work almost as well [25].

A.4. Comparison of techniques

In this section the previously discussed methods will be compared quantitatively, focusing on different performance criteria. Since the end goal of this research is to reduce power usage, the most important criteria is the power usage (or more accurately the energy usage). Other important criteria are accuracy, inference time, memory usage (which is linearly related to the number of parameters [7]). These criteria all depend heavily on the hardware used.

B

Support Vector Machine

A SVM is a linear classifier, that maximises the margin between the classes. It generates a hyper-plane that divides the classes with the margin (distance to closest sample) for each class. These margins to 2 classes are shown in figure B.1. It can be seen that while H2 and H3 both separate the classes, H3 does this by a larger margin, thus is considered a better classifier.

It can also classify non-linear class-distributions by using a kernel. [56]

SVMs do suffer when having to classify more than one class, as a separate SVM has to be made for each 2 classes, resulting in large computational needs.

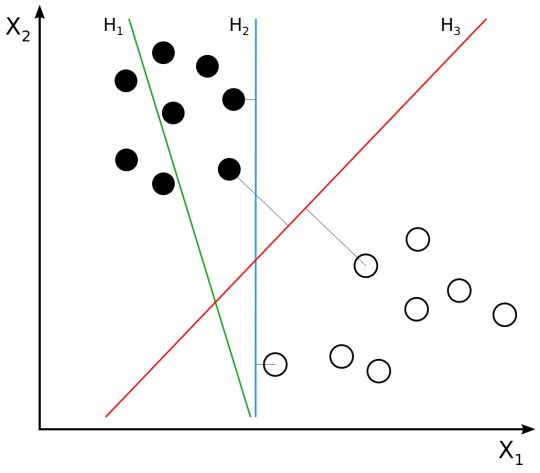


Figure B.1: The SVM algorithm (`https://en.wikipedia.org/wiki/Support-vector_machine`)

C

KNN

K-Nearest Neighbours (KNN) is a classification algorithm based on the distance between the datum to classify and the "training set". The distance calculation is a hyperparameter, where the ℓ_2 norm or the manhattan distance are usual choices. Sometimes the l_∞ norm is also used.

The other hyperparameter is K, the amount of neighbours are considered. In figure C.1 an example of a classification problem is shown. The green point is to be classified, with the solid circle being K=3 and the dotted circle being K=5. Assuming majority voting for the final decision, it can be seen that for K=3 the classification is the "red triangle" class, while for K=5 the classification is the "blue square".

Care should be taken in case an equal amount of neighbours are of the same class. For 2 classes this can be prevented by taking K odd, for more classes it can be solved by looking at the closest neighbour, or taking a random class from the top classes.

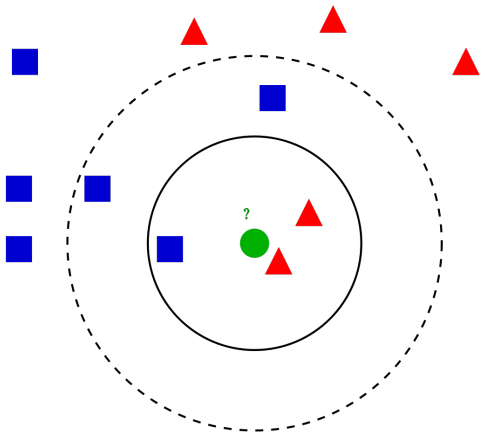


Figure C.1: The KNN algorithm, with K=3 and K=5 (`https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`)

D

Definition of compression levels

In this thesis JPEG-XR is used to compress training images. The compression function accepts a parameter called the compression level. The term compression level is used to indicate the amount of information that is lost during the compression. Compression level 100 is lossless[1] compression, whereas compression level 2 is the most lossy compression used. The file size also correlates in the same way: higher compression level results in greater file sizes.

D.1. Visualisation of the compression

In figure D.1 an example image can be seen in different compression quality levels compressed with the JPEG-XR compression. It can be seen that for a human the ship is recognizable for all compression quality levels.

In figure D.2 the average file size for the different compression quality levels can be seen. It should be noted that since the original image is not a raw image, but a JPEG-compressed image, the highest compression quality level results in a significantly larger file size than the original. Furthermore it is interesting to note that the file size reduction is not linear with the compression quality level.

So how exactly are the images compressed, and what does the compression level mean exactly?

The compression works in multiple ways. A part of the compression comes from chroma subsampling. Another part comes from encoding of the frequency-transform of blocks of the image. Before the compression is done the image is converted to the Luma Chrominance orange Chrominance green (YCoCg) colour-space. This is a so-called luma colour-space, where instead of having 3 colour channels as in RGB, the pixels are defined by their luminance and chrominances.

Chroma subsampling means that the chroma (roughly speaking the colour) of a region of pixels is sampled less frequently than the luminance (roughly speaking the brightness) of said region. This is done because a human is worse in distinguishing chroma than luminance.

The chroma subsampling works by sampling the chroma values less often than the luminance values. This is often denoted by $J : a : b$, where J is the width of a region within the image (often four). The height of the region is 2. a is the number of chrominance samples in the first row, and b is the number of changes between the pixels of the first row and the second row. b is almost always either 0 or equal to a. In the case that b is 0, the vertical chrominance resolution is halved, in the case that b is equal to a the vertical chrominance resolution is equal to the original. As an example, with $4 : 2 : 0$ chroma subsampling both the vertical and the horizontal resolution are halved.

After the chroma subsampling, the image is converted into blocks (almost always 4x4 blocks), from which for each channel all pixels are put in a specified order, after which a frequency transform is done. After this a complicated method of encoding the coefficients is done, with prediction of the coefficients as well as Huffman coding of the coefficients themselves. The final step is quantization of the coefficients, which is, apart from the optional chroma subsampling, the only lossy step in the process. Lossless JPEG-XR encoding is thus done by not doing the quantization. The amount of quantization influences the perceived quality of the image, and is one of the aspects influenced by the compression level.

[1] Depending on the colour-space transformation

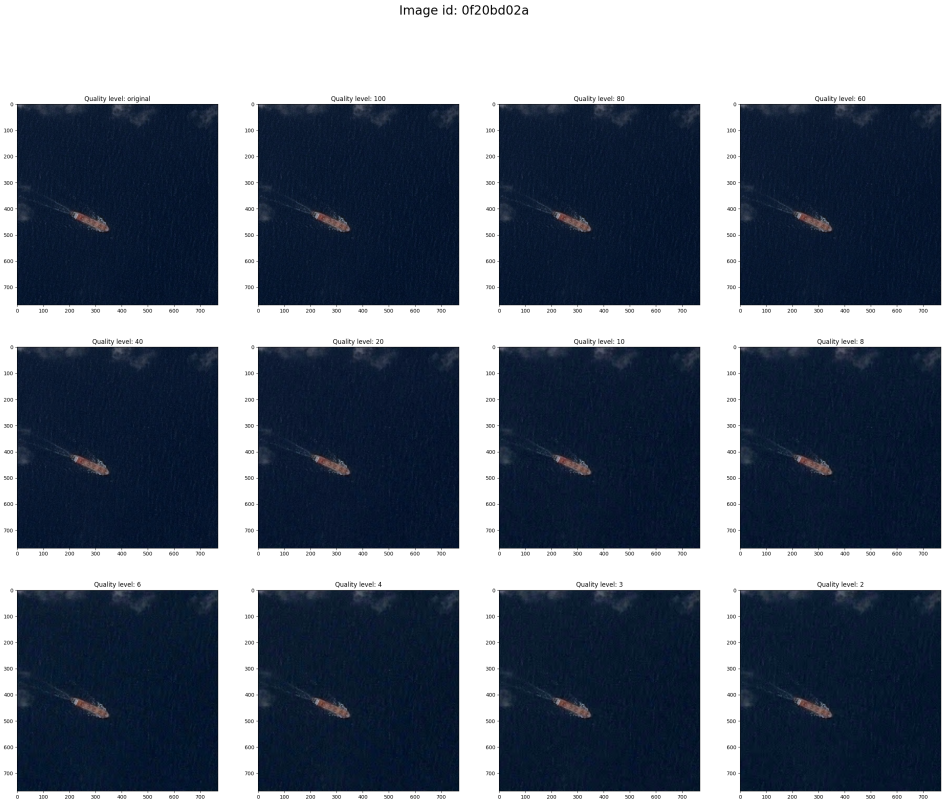


Figure D.1: The JPEG-XR compression on an example image

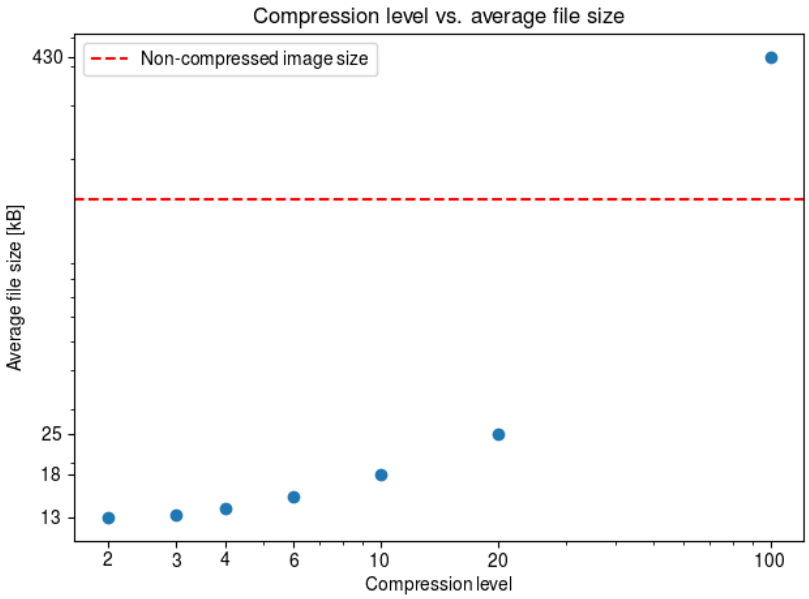


Figure D.2: The compression level versus the average file size for the dataset

The other aspect influenced is the chroma subsampling. The chroma subsampling is automatically changed from $4:4:4$ to $4:2:0$ when the compression level is smaller than 50.

Another aspect that influences the final size of the image is the bit-depth, or the amount of bits that one channel of one bit uses. This is most often either 8 or 12 bits, in the case of this thesis the images all have a bit-depth of 8 bits.

E

Types of Computer Vision tasks

For different applications, CV algorithms have to perform different tasks. There exist three main tasks that are often desired: classification, object detection and segmentation. Classification is often the easiest task, where an algorithm has to determine what an image is, i.e. is an image a cat or is it a dog? A harder, more relevant, problem is object detection: where in this image are the cats and dogs? The output of the algorithm is then a bounding box (often a rectangle) around the object. This is a harder problem because it is in essence a combination of nearly infinite classification problems, for each part of the image, at different scales. Even harder is image segmentation, where the goal is to know, for each pixel in an image, whether that pixel is part of a cat, a dog, or neither. The output is then a bit-mask of the image for each object that should be found.

For this thesis the object detection task is the most suited, as the final goal of the project will either be getting a location of an object, for which the middle of such a bounding box is accurate, or getting a snip of the image, for which the bounding box (plus some tolerance) can be used.

F

Hardware Solutions

With the rise in popularity of Artificial Intelligence (AI), different hardware solutions have also been proposed. An overview of all different solutions is given in this chapter. At the end of the chapter all solutions are compared. Sadly, it proofed to be very difficult to get comparable hard numbers, as the Domain-Specific Architectures have been released very recently, this is why some numbers are approximations. This problem has also been reported by others and attempts have been made to solve this, such as MLPerf [1], however power has not been taken into account for their results.

Do note that this chapter only considers the inferencing with the trained model, since this is the most resource-constrained step.

F.1. GPU

A GPU is a dedicated chip, used for image processing (and other highly parallel processes). They often have dedicated decoding hardware for much-used image formats, rendering hardware and other image-related dedicated hardware. They are used in combination with a Central Processing Units (CPUs) and most often can not be used separately.

Since the rise of deep learning, it was realised that GPUs are also fit for training and interfering NNs, due to the highly parallel nature of the problem. GPU manufacturers have also made an effort to optimise them for this use case, with tensor core support since CUDA 7.x[2]

F.1.1. NVidia T4

NVidia has even gone as far as marketing specific GPUs as "AI accelerators". While these are not ASICs they are specifically optimised to run NNs. One example of these is the NVidia T4. It can perform 65 TFLOPS with mixed-precision floats at 70W. [3]

F.2. FPGA

Field Programmable Gate Arrays (FPGAs) are software-programmable hardware modules, that are well fit to perform repetitive tasks power efficiently and fast. They have a large amount of logic gates, that can be connected in a way described by the hardware description language.

FPGAs are generally more power-efficient than GPUs while performing NN tasks, while they are less power-efficient than ASICs, as can be seen from figure F.1. They are however re-programmable, which does bring advantages, since different functions could be done using the FPGA.

F.3. Domain-Specific Architectures

Where as both FPGAs and GPUs were designed previous to the emergence of AI and have many different uses, there are also domain-specific architectures. These employ optimizations, such as lower-precision arithmetic

[1] mlperf.org

[2] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-and-technical-specifications

[3] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf

and better data flows, specifically for CNN applications. This results in lower power consumption per operation.

F.3.1. EdgeTPU

One example of such an integrated circuit is the EdgeTPU[4] developed by Googles Coral. It acts as an "AI accelerator", which means it does need a host to interface with. The EdgeTPU can do either 8 bit or 16 bit integer calculations, which means the models used have to be trained with quantization errors in mind. The EdgeTPU is able to perform 4 Tera Operations Per Second (TOPS) with a power usage of 1W. [1]

F.3.2. Intel NCS

The Intel NCS(2) uses the Intel® Movidius™ Myriad™ X VPU, which is a similar chip to the google EdgeTPU. It claims to perform 1 TOPS, while not mentioning any power usage statistics. There is also no mention of the type of data that it can perform these operations on, while being more expensive than the EdgeTPU.

This is why the chip will not be further considered.

F.3.3. TrueNorth

TrueNorth takes a different approach to AI. It is an ASIC, which does not try to optimize the computation of CNNs, but instead tries to simulate the human brain. It does this with binary outputs of each connection. [40] The main way this chip saves power is by not having a von Neumann architecture [5]. It does this by simulating synapses using logic gates. This results in a power-efficient architecture.

The drawback of this approach is that it can not use the researched NN architectures discussed in this chapter. While IBM has created their own architectures to perform the same tasks, they are not widely researched and in creating a custom architecture one is very dependent on IBMs software suite.

Thus, while the neuromorphic approach is promising in the long run, it is not a widely implemented approach at the time of writing.

F.4. Overview of Power vs. TOPS

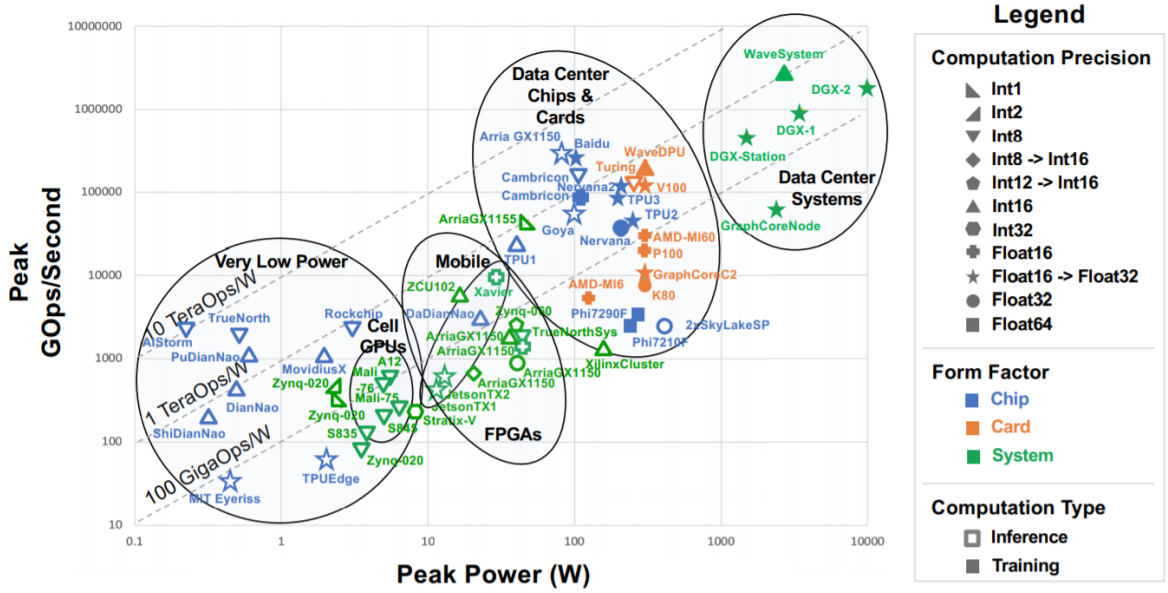


Figure F.1: An overview of AI hardware (accelerators) [51]

In figure F.1 an overview of commonly used hardware solutions for inference is shown. From this overview we can immediately see that, for efficient inference, either ASICs or Data Center chips/cards should be used. The absolute best inference operations/W is 10 TeraOps/W with the AIStorm chip. This chip is not yet on

[4] https://cloud.google.com/edge-tpu
[5] https://en.wikipedia.org/wiki/Von_Neumann_architecture

the market however, so the specifications should be taken with a grain of salt. After this the TrueNorth chip comes in.

While the TPUEdge scores low in this graph and even seems to be misplaced according to the google datasheet, this is because it is based on its' performance on MobileNetV2 for which the EdgeTPU reports 100+ fps and the MobileNetV2 paper report 585 MOPS per frame.[6] The peak performance of the EdgeTPU is 4 TOPS. [1, 2] The performance on MobileNetV2 has been updated on the site [1] to be 400 fps, which results in 234 GOPS practical for that particular architecture.

This difference in measurement illustrates a problem in the analysis of hardware solutions: theoretical OPS differ greatly from practical OPS due to factors such as pipeline usage, memory speed, re-configuring latency (if more parameters are used then supported) and input/output bandwidth.

F.5. Comparison

Different specifications are important for the hardware solution chosen are important: power efficiency, peak power usage, physical size, price, effort needed for radiation shielding, accuracy and development constraints. In table F.1 a morphological chart is given which summarises the performance of all different (high-level) options. [51]

	Power Eff.	**Price**	**Size**	**Supporting HW**	**Maximum speed**	**Ease of implementation**	**Precision**
GPU	-	-	–	–	**++**	**++**	**++**
CPU	-	-	-	**++**	+	+	**++**
FPGA	+	+	+	-	+	-	+
ASIC	**++**	**++**	**++**	+	-	-	-

Table F.1: Inference HW overview

From this we can conclude that the main trade off is between precision, speed and power efficiency. While the speed of an ASIC such as the EdgeTPU is still enough for most satellite applications, the precision might be an issue. However based on documents released by manufacturers, it is not expected that the decrease in accuracy is a problem. It is also expected that in the future ASICs will improve even further, also regarding accuracy and speed. Since GPUs are a more mature technology, the improvements expected for them is smaller. This is why it is expected that specialized hardware is the most future-proof method to implement AI.

This is why the different ASICs on the market at the moment are further compared. This is shown in table F.2. TrueNorth is not added to this comparison due to the different approach it makes to AI, resulting in it not being comparable. All *DianNao ASICs use the same chip, thus only the most powerful configuration is used.

ASIC overview	Power Usage	Peak OPS	Performance on Mobile-Net	Accuracy on Mobile-Net	Power eff. on Mobile-Net	Precision	Size (chip)	Model support
EdgeTPU	**1W**	4 TOPS	**400+ fps [36]**	70.6%	**0.1TOPS/W [36]**	**8 or 16-bit**	**15x10mm**	**Tensorflow-Lite**
MovidiusX	2W	1 TOPS[7]	50 fps [36]	**73.7%**	0.01TOPS/W [36]	**8 bit fixed point, 16 bit FP**	?? (only available as PCI or USB)	Intel OpenVino
DaDianNao	16W	**5.5 TOPS**	??	??	??	**16 bit fixed point**	3 mm^2 [8]	?? not available for purchase

Table F.2: Overview of different ASICs

[6] Information gathered via e-mail to author
[7] For NN, 4 TOPS total
[8] For 1 chip, not the complete system

G

Overview of all experiments

In table G.1 an overview of what images are used for each experiment is shown.

	Equivalent of 1000 q=2			**Equivalent of 10000 q=2**			**Equivalent of 30000 q=2**		
Data quality	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3
original	0-89	89 -178	178 -268	0-892	892 -1784	1784 -2676	0-2676	2676-5352	5352-8028
20	0-530	530 -1061	1061-1591	0-5304	5304 -10607	10607-15911	0-15911	8000-23911	16000-31911
10	0-722	722 -1444	1444-2167	0-7222	7222 -14445	14445-21667	0-21667	6000-27667	12000-33667
6	0-854	854 -1708	1708-2562	0-8538	8538 -17077	17077-25615	0-25615	4000-29615	8000-33615
4	0-930	930 -1860	1860-2790	0-9301	9301 -18603	18603-27904	0-27904	3000-30904	6000-31904
3	0-975	975 -1950	1950-2925	0-9750	9750 -19499	19499-29249	0-29249	2000-31249	2000-33249
2	0-1000	1000-2000	2000-3000	0-10000	10000-20000	20000-30000	0-30000	2000-32000	4000-34000

Table G.1: Images used for the experiments

Bibliography

[1] Accelerator module, . URL `https://coral.ai/products/accelerator-module`.

[2] Models intro, . URL `https://coral.ai/docs/edgetpu/models-intro`.

[3] Umar Albalawi, Saraju P Mohanty, and Elias Kougianos. Energy-efficient design of the secure better portable graphics compression architecture for trusted image communication in the iot. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 302–307. IEEE, 2016.

[4] Claudia Beleites, Ute Neugebauer, Thomas Bocklitz, Christoph Krafft, and Jürgen Popp. Sample size planning for classification models. *Analytica chimica acta*, 760:25–33, 2013.

[5] Irving Biederman. Human image understanding: Recent research and a theory. *Computer vision, graphics, and image processing*, 32(1):29–73, 1985.

[6] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[7] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[8] Po-Hsuan Cameron Chen, Yun Liu, and Lily Peng. How to develop machine learning models for healthcare. *Nature materials*, 18(5):410, 2019.

[9] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, and Synho Do. How much data is needed to train a medical image deep learning system to achieve necessary high accuracy? *arXiv preprint arXiv:1511.06348*, 2015.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[11] Corinna Cortes, Lawrence D Jackel, Sara A Solla, Vladimir Vapnik, and John S Denker. Learning curves: Asymptotic values and rate of convergence. In *Advances in Neural Information Processing Systems*, pages 327–334, 1994.

[12] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[14] Lilian N Faria, Leila MG Fonseca, and Max HM Costa. Performance evaluation of data compression systems applied to satellite imagery. *Journal of Electrical and Computer Engineering*, 2012, 2012.

[15] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[16] Rosa L Figueroa, Qing Zeng-Treitler, Sasikiran Kandula, and Long H Ngo. Predicting sample size required for classification performance. *BMC medical informatics and decision making*, 12(1):8, 2012.

[17] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[18] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `http://proceedings.mlr.press/v15/glorot11a.html`.

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[21] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[24] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[25] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[26] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam & beyond. 05 2018.

[27] Mark Johnson and Dat Quoc Nguyen. How much data is enough? predicting how accuracy varies with training data size, 2017.

[28] Mark Johnson, Peter Anderson, Mark Dras, and Mark Steedman. Predicting accuracy on large datasets from smaller pilot data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 450–455, 2018.

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

[30] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL `https://doi.org/10.1145/3065386`.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–104. IEEE, 2004.

[34] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2010.

[35] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[36] Leandro Ariel Libutti, Francisco D Igual, Luis Pinuel, Laura De Giusti, and Marcelo Naiouf. Benchmarking performance and power of usb accelerators for inference with mlperf.

[37] Daoyu Lin, Kun Fu, Yang Wang, Guangluan Xu, and Xian Sun. Marta gans: Unsupervised representation learning for remote sensing image classification. *IEEE Geoscience and Remote Sensing Letters*, 2017.

[38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[40] Paul Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, R. Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345:668–673, 2014.

[41] Sayan Mukherjee, Pablo Tamayo, Simon Rogers, Ryan Rifkin, Anna Engle, Colin Campbell, Todd R Golub, and Jill P Mesirov. Estimating dataset size requirements for classifying dna microarray data. *Journal of computational biology*, 10(2):119–142, 2003.

[42] Hong-Wei Ng, Viet Dung Nguyen, Vassilios Vonikakis, and Stefan Winkler. Deep learning for emotion recognition on small datasets using transfer learning. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 443–449, 2015.

[43] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[44] Yu Pu, Jose Pineda de Gyvez, Henk Corporaal, and Yajun Ha. An ultra-low-energy/frame multi-standard jpeg co-processor in 65nm cmos with sub/near-threshold power supply. In *2009 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, pages 146–147. IEEE, 2009.

[45] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1): 145–151, 1999.

[46] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2015.

[47] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[48] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[51] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. *arXiv preprint arXiv:1908.11348*, 2019.

[52] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101, 01 2010. doi: 10.1007/978-3-642-15825-4_10.

[53] Reenu Sharma and Sweta Agrawal. Image compression using dpcm. *GRD Journals-Global Research and Development Journal for Engineering*, 2(4), 2017.

[54] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[55] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/sutskever13.html`.

[56] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[57] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[58] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[59] Yuxin Wu et al. Tensorpack. `https://github.com/tensorpack/`, 2016.

[60] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[61] Wei Zhang, Kazuyoshi Itoh, Jun Tanida, and Yoshiki Ichioka. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, 29(32):4790–4797, 1990.