UPPSALA
UNIVERSITET

UPTECH STS 22027

Examensarbete 30 hp

Juni 2022

Importance sampling in deep learning

A broad investigation on importance sampling performance

Mathias Johansson

Emma Lindberg

Civilingenjörsprogrammet i System i Teknik och Samhälle

UPPSALA
UNIVERSITET

Importance sampling in deep learning

Mathias Johansson
Emma Lindberg

Abstract

Available computing resources play a large part in enabling the training of modern deep neural networks to complete complex computer vision tasks. Improving the efficiency with which this computational power is utilized is highly important for enterprises to improve their networks rapidly.

The first few training iterations over the data set often result in substantial gradients from seeing the samples and quick improvements in the network. At later stages, most of the training time is spent on samples that produce tiny gradient updates and are already properly handled. To make neural network training more efficient, researchers have used methods that give more attention to the samples that still produce relatively large gradient updates for the network. The methods used are called ''Importance Sampling''. When used, it reduces the variance in sampling and concentrates the training on the more informative examples.

This thesis contributes to the studies on importance sampling by investigating its effectiveness in different contexts. In comparison to other studies, we more extensively examine image classification by exploring different network architectures over a wide range of parameter counts. Similar to earlier studies, we apply several ways of doing importance sampling across several datasets. While most previous research on importance sampling strategies applies it to image classification, our research aims at generalizing the results by applying it to object detection problems on top of image classification.

Our research on image classification tasks conclusively suggests that importance sampling can speed up the training of deep neural networks. When performance in convergence is the vital metric, our importance sampling methods show mixed results. For the object detection tasks, preliminary experiments have been conducted. However, the findings lack enough data to demonstrate the effectiveness of importance sampling in object detection conclusively.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Willem Verbeke (Zenseact), Erik Werner (Zenseact) Ämnesgranskare: Per Mattsson

Examinator: Elísabet Andrésdóttir

Populärvetenskaplig Sammanfattning

Utvecklingen av mjukvara och hårdvara för att möjliggöra självkörande bilar är idag ett mycket aktuellt högteknologiskt område där flera av fordonsindustrins giganter är med och försöker skapa den bästa lösningen. Självkörande bilar kan delas upp i sex nivåer av autonomi som sträcker sig från ingen automation till total automation där en förare inte behövs. Ingen tillverkare kan idag erbjuda en konsekvent pålitlig lösning där föraren elimineras totalt i bilkörning. Zenseact [1] är ett av företagen som tagit sig an utmaningen i att producera mjukvara med mål att uppnå total autonomi för bilar. Företaget har sitt ursprung i Volvo Cars som än idag har delägarskap och täta samarbeten med företaget. Volvos historiska fokus på säkerhet kopplat till bilkörande är något som lever kvar inom Zenseact som, framför allt annat, ser självkörande bilar som en möjlighet att drastiskt minska antalet olyckor på bilvägar. Denna uppsats är skriven i samarbete med Zenseact med förhoppningar att kunna bidra till den fortsatta utvecklingen av företagets produkt. För att uppnå totalt oberoende av en förare krävs såväl avancerad mjukvara som hårdvara. I denna uppsats lägger vi tankar om hårdvara åt sidan och fokuserar istället på mjukvaruutvecklingen för självkörande bilar.

För att uppnå självkörning krävs det att bilen, likt oss människor, kan se, identifiera och sedan agera i sin omgivning. Lösningen för att kunna se sker genom den tidigare nämnda hårdvaran. De senare delarna kräver förståelse från bilen. Denna förståelse försöker man läsa genom artificiell intelligens (AI) på olika sätt. Mer specifikt tillämpas olika deep learning modeller, på svenska djupinlärningsmodeller, på det som bilen ser (även kallat input). Deep learning är ett område inom AI där man skapar stora nätverk som ämnar att imitera hur en hjärna fungerar och processerar information. Nätverken benämns ofta som neurala nätverk. De är uppbyggda av lager med noder (neuroner) som i sitt första lager tar emot exempelvis en bild. Denna bild analyseras lager efter lager i nätverken och varje neuron i varje lager har en särskild uppgift när det kommer till analysen av denna bild. När bilden har gått igenom samtliga lager kommer slutligen en gissning från nätverket, denna gissning avser bildens innehåll. Till en början är dessa nätverk, likt ett nyfött barn, usla på att avgöra vad de ser. Men efter omgångar av träning blir dessa nätverk, likt när barnet växer upp, mer och mer trygga i att identifiera vad de ser. En stor, och viktig skillnad mellan fallet för en människa och ett neuralt nätverk är att det inte tar flera år för nätverket att skapa sin kompetens. Med hjälp av prestandan hos dagens processorer kan denna träning ske på en betydligt kortare tid.

För att beskriva träningen av nätverket använder vi oss återigen av exemplet där vi använder en bild som input. Träningen av nätverken sker genom att återupprepade gånger skicka in bilder som man har det ”rätta svaret” för. Med det rätta svaret menar vi att vi har markerat vad som finns i bilden. Genom att låta nätverket processa bilder och sedan gissa på innehållet kan vi, för de bilder där vi har det rätta svaret, berätta för nätverket hur nära eller långt ifrån sanningen gissningen var. När vi berättat detta för nätverket har den möjlighet att göra förändringar i hur den väljer att behandla bilder för att i fortsättningen bättre kunna identifiera innehållet. Efter många tusentals iterationer av denna process är det troligt att nätverket till en hög utsträckning kan skapa pålitliga gissningar. Hur bra ett nätverk blir på att göra sina gissningar beror dels på hur svåra bilderna är att identifiera men även på hur det neurala nätverket är uppbyggt. Uppbyggnaden av nätverken är en otroligt komplex process vilket vi inte kommer redogöra för i denna sammanfattning.

Med detta ur vägen kan vi nu börja diskutera vilket problem denna uppsats tänkt undersöka. Träningen av stora neurala nätverk är extremt processortunga och idag dyra såväl sett till tid som monetärt [2]. En nedskalning av träningstiden med enbart ett fåtal procent är något som skulle vara

väldigt betydelsefullt då det sparar väldigt mycket pengar men kanske framförallt då det möjliggör en snabbare feedback-loop. Att behöva spendera vad som kan vara veckor för att få tillbaks resultat är idag en flaskhals när det kommer till utvecklingen av mjukvaran för självkörande bilar.

I denna uppsats undersöker vi ifall en uppsnabbning av träningen går att uppnå med hjälp av en metod som heter "Importance sampling", en så kallad samplingmetod. Som det går att utröna från namnet försöker denna samplingmetod fokusera på de "viktiga" bilderna. För att förstå vad nätverket ser som viktiga bilder behöver vi först förstå hur en typisk träningsprocess av ett neuralt nätverk ser ut. Efter en kortare tids träning är nätverket oftast kapabel att korrekt identifiera majoriteten av bilderna som den tränas på. Den tidskrävande delen av träningprocessen handlar därför om att lära sig resterande, det vill säga en minoritet av bilderna. Det är dessa bilder som anses vara svårare och därmed "viktiga". Metoden "Importance sampling", som ska snabba upp träningen, fungerar därför genom att först avgöra vilka bilder vårt nätverk upplever som svåra. Därefter, vet nätverket vilka bilder som i större utsträckning ska användas för träning. På så sätt, genom en större exponering av de svåra bilderna, är förhoppningen att nätverket snabbare än tidigare ska komma till en tillfredsställande prestanda.

Forskningsfältet som denna typ av lösningar ryms inom kallas för "computer vision", eller på svenska datorseende. Vi har i denna uppsats utfört experiment inom två av de mest populära uppgifterna inom computer vision. De två uppgifterna är "object detection" och "image classification". De båda är till för bildanalys men vad de utför för analys skiljer sig. Image classification berör enbart klassificering av bilder. När sådana uppgifter löses innefattar bilderna, som är input till nätverket, maximalt ett objekt och objektet är det solklara fokuset i bilden. Uppgiften är här för nätverket att avgöra vad för typ av objekt som finns i bilden. När vi istället utför object detection med hjälp av ett neuralt nätverk är bilderna som skickas in som input till nätverket representerade i en större kontext och det kan vara så att det finns fler, eller inga objekt i en bild. Nätverkets uppgift är här att dels, likt image classification, avgöra vilka typer av objekt som finns i bilden. Utöver det ska nätverket även avgöra vart objekten befinner sig. Med det sagt är object detection en mycket mer komplex uppgift att lösa inom Compute vision än image classification.

Den absoluta majoriteten av uppsatsens resultat kommer från experiment på image classification. I dessa experiment har vi dels kontrollerat hur en baslösning av importance sampling förhåller sig i prestanda jämfört med den traditionella metoden där alla bilder har samma sannolikhet att bli valda vid träning av neurala nätverket. Utöver baslösningen har vi experimenterat med ett antal olika parametrar. Vi har undersökt hur ett neuralt nätverks komplexitet påverkar importance samplings prestanda. Vi har även undersökt olika tung viktning av de svårare bilderna och hur detta påverkar prestandan av importance sampling. Slutligen har vi även utrett hur olika metoder för importance sampling presterar i jämförelse med varandra. Resultaten från dessa experiment tyder på att importance sampling har stor potential när det kommer till uppsnabbning av träningen för ett neuralt nätverk, speciellt i början av träningen. När vi ser till slutresultatet, när nätverket har konvergerat i prestanda, är det dock inte lika självklart att importance sampling är att föredra framför en metod där vi väljer alla bilder med samma sannolikhet.

När det kommer till experimenten för object detection har vi gjort ett försök i att överföra metoderna och resultaten från image classification. De experiment som vi presenterar för object detection visar inte samma lovande resultat. Detta beror på att det neurala nätverk vi använt för object detection visar dåliga resultat oavsett om vi använder importance sampling eller ej. Därmed kan studien inte ge något definitivt resultat huruvida importance sampling snabbar upp träningen för object detection eller inte. Vi ser dock ett stort behov av att fortsätta utreda metoden för object detection; dels motiverat genom de lovande resultaten från image classification men även då de resultat vi presenterar i denna uppsats inte fullt ut kan ses som representativa.

Acknowledgements

We present this thesis as a collaboration with Zenseact. The company has, from the start, provided us with resources and a foundation to enable the project completion. Specifically, we would like to thank our mentors at Zenseact, Willem Verbeke and Erik Werner. They have been significant contributors to this thesis. By active daily dialogue, always providing helpful insights or further piquing our interest, and constantly helping us move forward, they have made this thesis possible. We would also like to extend gratitude toward our subject supervisor, Per Mattsson at Uppsala University, who has always been ready to help if needed. Additionally, we would like to thank the city of Gothenburg for occasionally hosting us during this semester; it has been a pleasure.

Acronyms

ANN	Artificial Neural Network
AUC	Area Under the Curve
AP	Average precision
CNN	Convolutional neural network
CV	Computer vision
DNN	Deep Neural Network
FN	False negatives
FP	False Positives
FPR	False positive rate
GTSRB	German Traffic Sign Recognition Benchmark
IS	Importance sampling
MLP	Multilayer perceptron
NIS	No importance sampling / Uniform sampling
OD	Object detection
ROC	Receiver operating characteristics
SGD	Stochastic gradient descent
SSD	Single shot detection
TN	True Negatives
TP	True Positives
TPR	True positive rate
YOLO	You Only Look Once

Contents

Chapter 1

Introduction

Efficient and robust detection of objects under a broad range of possible circumstances is a central challenge in the development of autonomous vehicles [3]. The object detectors used by market leaders are commonly deep neural networks trained on a large number of frames captured by on-car cameras [3]. The resolution of the input images to the object detection (OD) network is necessarily large to ensure sufficient performance [4]. Combining a very complex neural network, high-resolution frames, and a large data volume results in a need for immense computational power during training. High monetary costs are a direct consequence of this, which is an actively discussed issue within the field [2, 3, 5, 6].

During the first few iterations, the training of a neural network tends to converge relatively quickly. Most frames produce substantial contributions to how we want to optimize the network; these contributions are known as gradients. At later stages of training, the network training slows down significantly. To a large extent, this is due to the network already being able to produce correct predictions for the bulk of the frames in the training set [5], resulting in tiny gradient updates. A fraction of frames corresponding to objects and circumstances more challenging to handle for the network will keep producing relatively large gradients [5]. Such frames significantly contribute to developing the network training.

In this thesis, we will systematically investigate the possibility of improving the training time of neural networks on CV tasks through importance sampling strategies. Oversampling the frames that have a high impact on the learning of the network might be a powerful way of decreasing the training time of neural networks on CV tasks while achieving a similar network performance in the end [5, 7]. Doing so would have a host of benefits as issues could be addressed more quickly in the new version of the network. As a result, more experiments could be carried out to improve the network performance, and as the data volume keeps growing larger and larger, the training time would stay manageable.

The structure of this thesis is as follows: in Chapter 2 the necessary knowledge for understanding the results of this thesis is presented. A short overview of CV is followed by an extensive section explaining deep learning. We go through what neural network architectures we use and how the training and evaluation happen. In Chapter 3 there is a summary of previous studies regarding importance sampling. This chapter concludes with an explanation of how this study aims to develop the area further. In Chapter 4 we present the different datasets that are being used in our experiments. This is also where we present the tools used in the thesis. The chapter concludes with an explanation of our process when working on this thesis. Next is Chapter 5, where all the different experiments are explained in detail, and their results are presented and also discussed. Lastly, Chapter 6 is where we present the contributions and conclusions made from all the experiments combined. Even if the thesis aims to improve OD training times, this thesis will implement its importance sampling strategies on both image classification (IC) and OD tasks. Most chapters are divided in a logical way where we separate discussions on IC and OD into different sections.

1.1 Purpose

This thesis investigates if and how we can improve the training times of neural networks on CV tasks by using importance sampling. We aim to provide a broad understanding of importance sampling strategies, like how and when they can be helpful. We do this by conducting experiments on different datasets, networks, problems, and hyperparameters.

1.2 Collaboration

This thesis is done in collaboration with the software company Zenseact which develops high-end software intended to ultimately create safe self-driving software that can be used in the real world. The idea behind this thesis originated from the supervisors at Zenseact. They saw a need for a speed-up of the training of their neural networks for OD. During the duration of this project, we, the authors, have been able to work closely together with the supervisors through daily communication and weekly check-up meetings. The results were presented to Zenseact as a means to share the intel we gathered through this process.

Chapter 2

Background

Computer Vision (CV) is the scientific inquiry in how computers can gain understanding from videos and images [8]. OD and IC are two of the most common tasks in CV and the development of complex image and video analysis [9]. IC is a fundamental task in CV. Given an input image X, the aim is to predict the class Y it belongs to among some predefined classes. Both classic CV and DL (DL) [10] can be used to tackle such tasks. However, in some tasks, such as detecting other road users in a self-driving vehicle, the classification of images is not enough. An integral part of OD is not only to classify an object but also precisely estimate the location of said object. This type of method is called OD. Two subtasks need to be solved in OD: 1) the localization and 2) the classification of objects [11]. Recently, DL has been the method of choice for real-world OD and can generally be divided into two categories, 1) Two-staged detectors and 2) Single-stage detectors [12]. Two-staged detectors are generally more accurate but too slow to be used in real-time applications as they would achieve far worse detection performance than single-stage detectors. [12]. As autonomous vehicles are a real-time application, the priority of quick inference is essential; therefore, single-stage detectors are used in this thesis [12]. They treat the task as a single regression problem where objects are located and classified in the same stage [12]. This thesis will evaluate importance sampling for both IC and OD to see if the conclusions we make for each case can be generalized between the problems.

Understanding how neural networks work and train is needed so that we, later in this chapter, grasp why and how an importance sampling algorithm can speed up the training of neural networks. DL methods have outperformed previous state-of-the-art machine learning (ML) techniques for many tasks, with CV being one of the most prominent fields [13]. There is a variety of DL frameworks in use today. For this thesis, two different convolutional neural networks (CNNs) will be used for the IC, while we use YOLOv2 for OD, which is a single-staged OD architecture [14]. CNNs are a class of neural networks commonly used in CV; this section will explain their structure and how they work. Afterward, we describe how training and evaluation are conducted in general and specifically for this thesis.

2.1 Artificial Neural Networks

CNNs are a class of artificial neural networks (ANNs). ANNs are structured in a way that imitates biological brains. This structuring has been proven helpful when dealing with CV tasks. The largest component of the ANNs structure is its node layers. There is one input layer, one output layer, and one or more hidden layers in a network. A simple representation of such a network can be seen in Figure 2.1. Note that the network in the figure is a deep neural network, which is just a ANN with multiple hidden layers. We send information from the input layer iteratively through each layer in the network until the output layer is reached. At which point the network, based on the information processing of each layer, makes a prediction. Information processing depends heavily on each layer's task and will be discussed in Section 2.1.1. With this very broad description of a ANN, we now focus on what happens inside and between node layers.

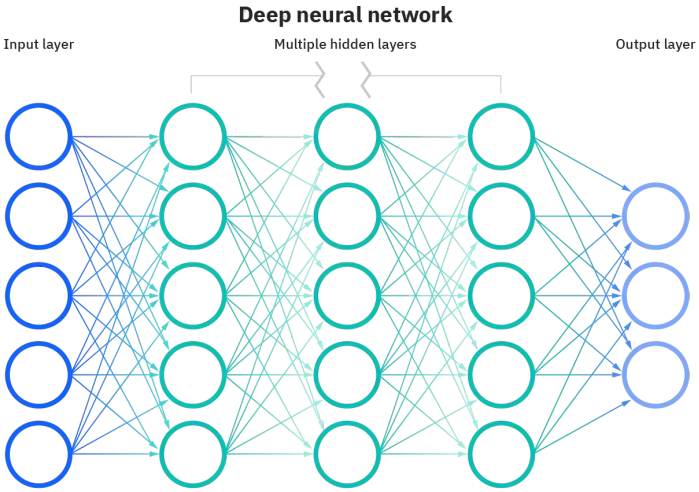

Figure 2.1: The basic element of a neural network: node computation. [15]

At their core, node layers consist of nodes that are called neurons, much like a biological brain. Together with their weights, the neurons are what process the tasks of the neural network. The computations connected to each neuron consist of

1. a set of inputs O_n,
2. a set of weights w_n which determine the importance of each input, and
3. a bias b which determines the importance of that neuron.

How each neuron processes information from 1 to 3 can be viewed as the linear regression model

$$\sum_{i=1}^{n} w_i \times O_i + b \tag{2.1}$$

The inputs O_n are the outputs of neurons from the previous layer. The weights w_n of our neuron are added to determine the importance of each input. These are tuned during the training stage [16]. Weighting is a crucial part of how an ANN works. The weights and bias are updated based on how far from the truth our predictions from the output layer are. How this is done specifically is discussed in Sections 2.2.1.1 and 2.2.1.2.

The resulting sum in 2.1 is then usually passed through an activation function f. These are used to add non-linearity and ensure that the network does not degenerate into a single-layer network that would have been linear [17]. For a visual representation of a neuron's integral computations, see Figure 2.2. As earlier noted, how a neuron processes, its input depends heavily on what type of node layer it is a part of. From Figure 2.2, we can observe connections with input going from O_0 to O_n; these connections are what send information between neurons. The resulting O_j from the neuron in Figure 2.2 will send information forward to neurons in the next layer, where it will be represented as one of the O_is that is being received as input.

With what is usually thousands of these neurons divided into different node layers, a ANN can process and send information through its network and hopefully, with training, provide correct predictions for its tasks.

The idea we have presented in this section is for so-called multilayer perceptrons (MLPs), a class of neural networks that use fully connected layers. They are characterized by having every neuron in one layer send input to all neurons in the next one [19], making them fully connected. These are often used to make predictions when the input is tabular data. Due to MLPs becoming heavily parameterized

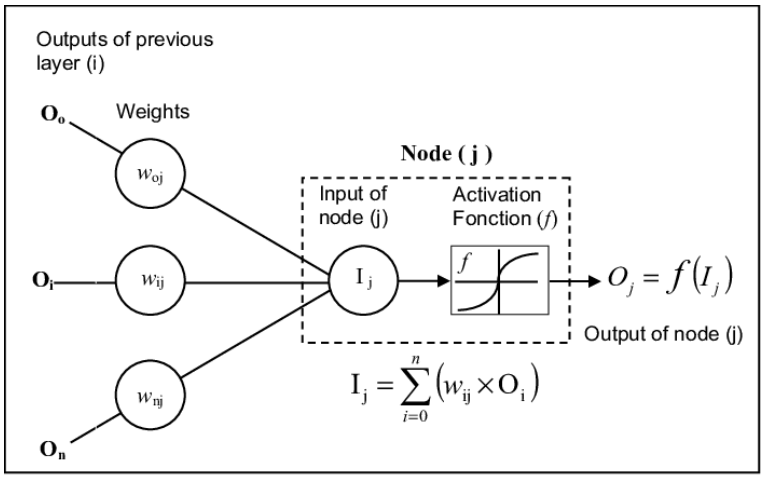

Figure 2.2: The basic element of a neural network: neuron computation. [18]

from high dimensional input, these networks are inferior to CNNs for CV tasks. Additionally, CNNs also have the upper hand in CV because of the being spatially invariant, which MLPs are not.

2.1.1 Convolutional Neural Network

CNNs are currently the go-to method for CV. Its role is to reduce the images into a form that is easier to process without losing critical features for getting a good prediction [20]. While not all their layers do, some reduce the dimensionality of their input. As a result, they do not see the same problems as MLPs when handling input of high dimensions. Additionally, a significant advantage of CNNs is that they are spatially invariant. Compared to an MLP, which is acutely aware of which exact pixel some information came from, a CNN is not. This precisely gives CNN their spatially invariant powers while an MLP learns to only care about certain features if they appear in a particular pixel or set of pixels. A noticeable difference to MLPs is also that CNNs can, instead of being exclusively fully connected, be sparsely or partially connected [21]. In a CNN, there exist at least three types of layers. Those are:

- **Convolutional layers**, which are the pillars of any CNN. As the name suggests, the convolutional layer performs a "convolution". It initially handles input like previously described, multiplying it with weights. The convolutional layers differ by using what is called "filters". They are matrices smaller than the input that can be used to detect specific parts of the input, such as the edges of objects. They work by taking the dot-product of a filter-sized part of the image and the filter. Later, summarizing the values of the dot-product through addition [22]. The final value of this process alludes to how well the region resembles the filter. By doing this repeatedly, traversing the input, filters process the image. To visually understand how these works see Figure 2.3. How it does this will depend on the two hyperparameters' stride and padding. The stride defines the step size of the filter as it passes over the input. The padding defines how far outside the input the filter can go. At each step, the summarized value from the dot product is added to what is called a "feature map"; which is the output from the convolutional layers filters [23]. The feature map is then passed through an activation function before becoming the layer's output. These filters are what cause the earlier described spatial invariance of CNNs. By being iteratively used over segments spanning the entire input, the filters can detect the features we look for anywhere in an image [22].

- **Pooling layers** are layers which, as the name gives away, pool together the input it receives and reduces the dimensionality [21]. Given an input of feature maps from a convolutional layer, the pooling layer typically divides the input into smaller regions. Each of these regions gets assigned one value each, representing that entire region going forward [23]. There are different methods

of obtaining this representative number. Two of the more common ones are max- and average pooling. Max pooling will take the highest value in each region as a representation, while average pooling will take the region's average value. Pooling is used to reduce the dimensionality, thus decreasing the computational power required to process the data [20].

- **Fully connected layers** have, as previously mentioned, every neuron in one layer sends input to all neurons in the next one [19]. These layers are frequently used in the later parts of CNNs to build more robust capabilities from the features given to them by previous layers [24].

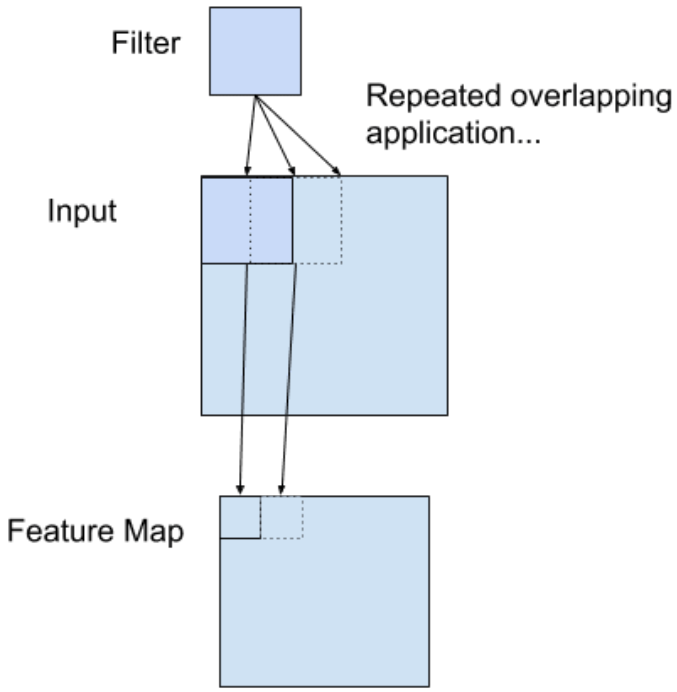

Figure 2.3: Example of a Filter Applied to a Two-Dimensional Input to Create a Feature Map. [22]

An example of how all these layers look together in a CNN can be seen in Figure 2.4. In addition to these layers, we use activation functions between layers. For this thesis, the activation functions ReLU [25] and the softmax function [26] have been used. These layers can be seen in Figure 2.4. Another challenge when working with CNNs is that it is hard to train models as the parameters in each layer change during training, and thus the input and output to each layer change. This challenge is referred to as internal covariate shift and is commonly addressed by batch normalization, which normalizes the inputs [27]. For this study, we apply batch normalization as it allows for a higher learning rate and less careful initialization [27]. The network's final output is received after a pass through a combination of all these types of layers.

The training and optimization of CNNs in practice will be further explained in the following sections. Nevertheless, it is now time to address why we use CNNs for this thesis. One of the main advantages of CNN is that the location is invariant, which means that the filter can find the pattern anywhere in an image, no matter where the pattern is located [21]. This makes the network more suited for an image-focused task as it allows to encode image-specific features into the architecture [21].

2.2 Training neural networks

This section describes the necessary knowledge to understand the training and optimization of our neural networks. Before explaining how this thesis specifically optimizes our neural networks, we give

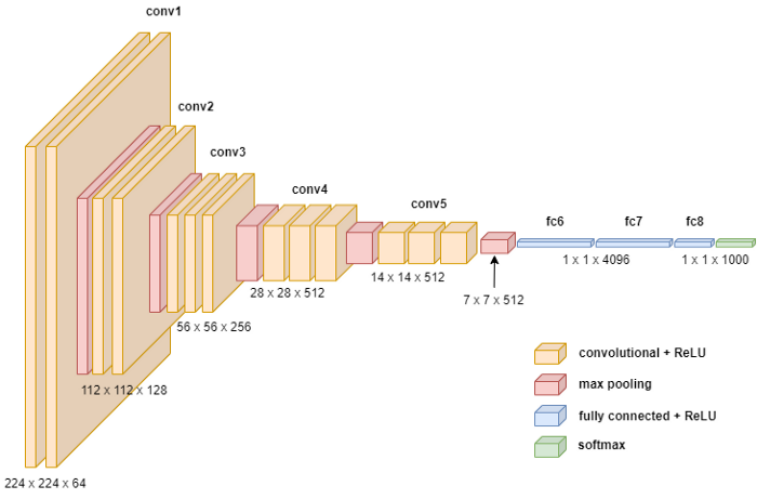

Figure 2.4: A simple CNN architecture showing its most common components. The network in the example give one prediction for 1000 different classes. [28]

a general introduction to the optimization algorithm gradient descent. Lastly, we present the loss functions for our different tasks. A basic familiarity with these subjects is crucial to understanding the aim of our importance sampling strategies.

2.2.1 Gradient Descent

Several existing methods exist for training and optimizing neural networks, and gradient descent is one of the most popular and common algorithms [29]. The gradient is calculated from training on labeled data where the algorithm minimizes the objective function $J(\theta)$ parameterized by the model parameters θ. The parameters are updated in the opposite direction of the gradient of the objective function $J(\theta)$ with respect to the parameters. To determine the size of the steps the algorithm utilizes a learning rate n, which is used to scale the gradient update. Through this process, the goal is to reach a local minimum, preferably not just any local minimum but a very low one. The process of finding one is a task tackled differently depending on the method used; this is discussed in Section 2.2.1.3. This process can be visualized in Figure 2.5. At each training step, we move in the direction of the slope of the hypersurface created by the objective function. We do this until the minimum is reached [29]. We will explain each part of the training process in the following subsections. Before we go into details about gradient descent it is worth mentioning that our description applies to the simple MLP case. The MLP case is the easiest example from which you can illustrate the mechanisms.

2.2.1.1 Forward propagation and cost function

A forward pass is the first step in optimizing the model's objective function, referred to as the cost function. Furthermore, the parameters θ we aim to optimize will be referred by weights w, and biases b as this more specifically representing what we actually optimize in a neural network. The forward pass refers to the calculations of the output of a neural network. This series of calculations start from the input data and then iteratively calculates each neuron in the next layer. Going from the input to the output of each layer until the output of the last layer is obtained. The last layer's output corresponds to the actual prediction. After the forward pass, the predicted output is compared to the target and measured with a cost function which we aim to optimize in training [31].

2.2.1.2 Back propagation - computing gardients

After a forward pass, the model performs backpropagation to calculate the gradients; this is the main step in the gradient descent algorithm. The gradients tell us in what direction the parameters need to

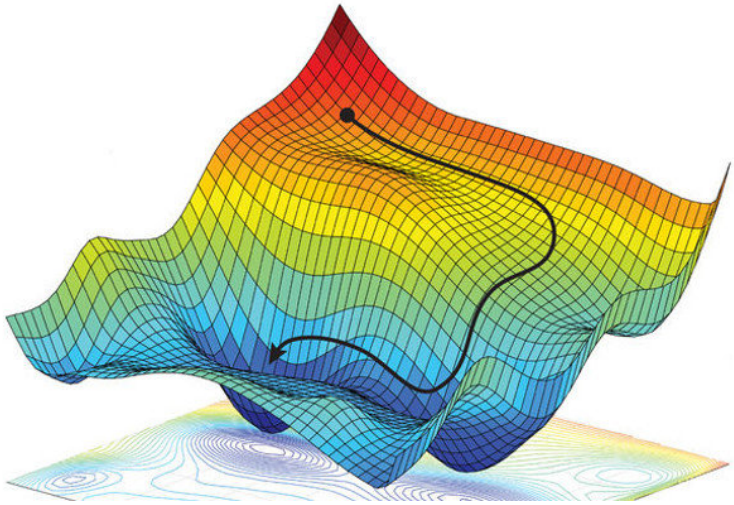

Figure 2.5: A visualization of how gradient descent finds the local optimum on a hypersurface. The gradients descents path marked by the black line. [30]

change to minimize the cost function. There are different approaches for computing these, which differ in how many samples are used in the gradient computation [29]. The most common approaches are discussed in the subsection 2.2.1.3. Even though these methods accomplish the task differently, they all share common characteristics in minimizing the cost function. The core idea of backpropagation is to compute the relationship between the parameters of the neural network and the cost function. This is done by calculating the partial derivatives of the cost function

$$\frac{\partial C}{\partial w_i{}^l_j} \tag{2.2}$$

of for the weight w, which adjust the strength of connections between neurons,

$$\frac{\partial C}{\partial b^l_j} \tag{2.3}$$

of for the bias b, which make adjustments within neurons, in the neural network. These expressions tell how changing the weights and biases change the behavior of cost function and thus the neural network [31].

To understand the computation of the mentioned partial derivatives some details and expressions about the network need to be clarified. Notation $w_j{}^l_k$ will be used to express the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. Similarly, for the activation and biases in the network: a^l_j expresses the activation of the j^{th} neuron in the l^{th} layer. While b^l_j expresses the bias of j^{th} neuron in the l^{th} layer. w^l express all weights connecting to layer l. Similarly, the activation and bias vectors are defined as b^l and a^l whose components take values a^l_j and b^l_j for each neuron in the layer l. Another notation is σ to denote vectorizing a function (elementwise application of a function). This is because the activations in one layer can be expressed and related to activations in earlier layers $a^l = \sigma(w^l a^{(l-1)} + b^l)$. When this is used, the quantity inside σ denotes as z^l and is called the weights input to neurons in layer l. These expressions will be used in a detailed description of backpropagation, where the goal is to find the partial derivates of (2.2) and (2.3) [31].

To find the partial derivatives the first need is to find the loss δ^l_j, for every j^{th} neuron in every layer l, which is defined as follow [31]:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \tag{2.4}$$

Four fundamental equations are used to compute the loss and the two partial derivatives. The first expresses the loss in the output layer L [31]

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2.5}$$

The first term of (2.5) denotes how fast the cost function changes for the j^{th} output activation. In other words, δ_j^l will be small if C does not depend on the output from the j^{th} neuron. The last term expresses how fast the activation function σ changes at z_j^L. Another fundamental equation is the loss δ_j^l expressed in terms of the next layer δ_j^{l+1} [31]

$$\delta_j^l = ((w^{l+1})^T \delta_j^{l+1}) \odot \sigma'(z^l) \tag{2.6}$$

The loss in layer $l+1$ multiplied by the weight matrix for layer $l+1$ forms the error at the output of layer l. When then taking the Hadamard product [32] $\odot\sigma'(z^l)$ of that, the error moves through the activation function in layer l resulting in the error δ_j^l in the weighted input to layer l. With (2.5) and (2.6), it is possible to compute the error δ^l for any layer in the network. This is done by using (2.5) to compute the error in the last layer δ^L and then using (2.6) to compute the error δ^{L-1} and then (2.6) again back to the first layer in the network [31]. The last two fundamental equations are for the rate of change of the cost for any bias

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.7}$$

and,

$$\frac{\partial C}{\partial w_j{}_k^l} = a_k^l \delta_j^l \tag{2.8}$$

any weight (2.8) [31]. The cost rate of change for the bias is exactly equal to the error δ_j^l while the cost rate of change for the weight is the error δ_j^l multiplied by a^{l-1} [31]. These computations effectively measure the change of parameters concerning the cost function and tell the model in what direction the parameters should be updated to minimize the cost function and thus improve the neural network [31].

In summary, the backpropagation's equations from (2.5) - (2.7) provide a way of computing the gradients. z^l and a^l are computed for each layer during the feed-forward process. After that, the loss in the final layer L is computed with (2.5). Then followed by the computation of the loss backward for each layer with (2.6). As a final step the gradients are computed with (2.8) and (2.7) [31]. The backward movement through the network is the reason for the name backpropagation, starting from the final layer and iterating backward through the network to find all losses. This explanation of the backpropagation algorithm should provide enough understanding of the subject for this thesis. As mentioned earlier, different approaches include varying data usage in backpropagation. These approaches will be described in the three following subsections.

As mentioned, the description applies to the simple MLP case. It is a bit different in practice as it uses automatic differentiation. Automatic differentiation is an approach for calculating values of the partial derivatives in a given point, for instance, the gradient. Without going into details, it decomposes any complex mathematical expression into a sequence of elementary ones for which the derivative is known. Then the chain rule is applied to get the mathematical expression; gradients in our case [33]. Note that it is the numerical value of each partial derivative.

2.2.1.3 Gradient Descent Variants

There exist some variants when computing the gradients during backpropagation. The three most common approaches for computing these differ in the number of samples used in the gradient computation and will be described in the following paragraphs. Instead of using the cost function where we

optimize the biases and weights, we will use the more general term, objective function J(θ) parameterized by the model parameters θ, again.

Batch gradient descent
The first variant, batch gradient descent, uses the entire dataset when computing the cost function and the gradients using the four equations described in (2.5) - (2.7). If any trainable parameter is expressed as θ, the update of θ can be described as

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t) \tag{2.9}$$

The parameters are updated in the opposite direction of the gradients, where the update size is determined by the learning rate η. When the gradient, $\nabla J(\theta)$ for each parameter θ is computed by using the whole dataset, this approach is slow and can cause memory problems for big datasets [29]. The main advantage of batch gradient descent is that it guarantees to converge to the global minimum for convex error surfaces and to the local minimum for non-convex problem [29].

Stochastic gradient descent
In contrast to batch gradient descent, Stochastic gradient descent updates the parameters for each training sample [29] and the parameters optimization is described as

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t; x^i; y^i) \tag{2.10}$$

Stochastic gradient descent performs an update after each sample; due to frequent updates with high variance, the fluctuation is noticeable [29]. This fluctuation enables us to jump to a new and potentially better local minimum. At the same time, Stochastic gradient descent can complicate convergence to the exact minimum as there is a risk of overshooting if the learning rate η is too large. By decreasing the learning rate η to a reasonable rate, Stochastic gradient descent shows the same convergence behavior as batch gradient descent [29] apart from the fact that there is still noise from the statistical uncertainty.

Mini-batch gradient descent
The third variant, Mini-batch gradient descent, mixes the two above. In this case, the parameters are updated for every mini-batch of size n [29] which result in

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t; x^{i:i+n}; y^{i:i+n}) \tag{2.11}$$

The size of the batch varies depending on the experiment [29]. Mini-batch gradient descent is a common choice when training neural networks as it provides a more stable convergence thanks to the variance reduction in parameter updates [29]. In addition, it also does not have any of the drawbacks of batch gradient descent. This variant of gradient descent is the baseline in this thesis. In selecting batches in this approach, it is typical to reshuffle the dataset after each epoch and then iterate through the dataset by sampling one batch at a time [34].

2.2.1.4 Optimization

Backpropagation is used to calculate the gradients efficiently, and as described above, this can either be done by using one sample, all samples or averaging over n samples in a batch. Optimizers then use the gradients computed during backpropagation to update the neural network parameters to minimize the loss function. The Equations (2.9), (2.10) and (2.11) describes how the update of a neural network's parameters works. Each parameter θ in each layer is updated in the opposite direction of the gradient $-\nabla J(\theta)$. The learning rate η determines how big the parameter update should be. Picking a tiny learning rate can cause slow convergence, while a too big learning rate can hinder convergence since this can cause overshooting of the optimal point [29]. Another challenge with the learning rate is that

the same one is not necessarily the optimal one for all parameter updates. For instance, it can be preferred to perform a larger update on parameters connected to rarely occurring features [29]. Due to these challenges, the adaptive learning rate is often used during training [29]. For this thesis, we use the Adam optimizer [35]. Adam stands for Adaptive Moment Estimation and is one adaptive learning rate method. It uses momentum by having a moving average that includes the gradient of our current step with gradients of previous steps in calculating the direction. Adam is chosen in this study as its performance has been widely tested and is also computationally efficient [35].

2.2.2 Loss functions

In this section, we are talking about the cost functions used in our study. The loss function and cost function are synonymous, the loss function is used when talking about a single training example while the cost function, on the other hand, is the average loss over the entire training dataset. For simplicity, we will in this section refer to loss functions and only look at one training example when we describe what we applied to our experiments. For the loss function in IC see Section 2.2.2.1, and for OD see Sections 2.2.2.2 and 2.2.3.

2.2.2.1 Loss function for image classification

For IC, the loss function is much more trivial than for OD. In the IC case, the probabilities for each class will be compared to the target label with a loss function. In this study, the IC will be a multi-class classification task; therefore, categorical cross-entropy [36] will be used as the loss function. The following equation describes it:

$$\text{Loss} = -\sum_{i=0}^{C} y_i \log \hat{y}_i \tag{2.12}$$

where $\hat{y}_i$ is the ith scalar value in the model output, y_i is the target label, and C is the number of scalar values in the model output; in other words, the number of classes. The cross-entropy loss is designed so that the neural network maximizes the likelihood of its parameters with respect to the training data set [37].

2.2.2.2 You Only Look Once

OD is generally a more complex CV task to solve than IC. The following section explains how it works with our chosen YOLO-model for OD. DL-based OD solutions can mainly be divided into two sub-categories 1) Two-staged and 2) Single-stage detectors [12]. The first category is often referred to as Region Proposal Based Framework and follows a more traditional pipeline by first locating objects by generating region proposals and then classifying them [11]. The second category, single-stage detectors, treats the task as a single regression problem where objects are located and classified in the same stage. Therefore it is also common to refer to them as regression-based methods [12]. As single-staged detectors can reduce time expense, this method is preferred for real-time applications, such as autonomous vehicles, since quick inference is important [12]. The single-staged detectors includes MultiBox [38], AttentionNet [39], G-CNN [40], YOLO [41], SSD [42], and more. For this thesis, we use a version of YOLO called YOLOv2.

YOLO is a single CNN method that directly predicts bounding boxes, a rectangle surrounding an object, and class probabilities [41]. The first version of YOLO was developed in 2015; then, it was a new approach for object detction [41]. The idea behind YOLO is that the network divides the image into regions, and for each region, it predicts bounding boxes and probabilities, visualized in Figure 2.6. More specifically, YOLO divides an image in an $S \times S$ grid where each grid cell is responsible for detecting all objects whose center is inside that cell [41]. Every grid cell predicts X bounding boxes, and each box holds a confidence score and class probabilities C [41]. The confidence score is between 0 and 1, and defined as

$$p(Object) * IOU^{truthpred} \tag{2.13}$$

where $p(Object)$ reflects how likely the box contains an object and $IOU^{truthpred}$ reflects how accurate it thinks the box's predicted coordinates are [41]. The IOU refers to intersection over union and is a value between 0 and 1, representing the overlap between a predicted box and the ground truth. If an object exists in the grid cell, this score should be close to 1 and otherwise close to 0 [41]. The C, class probabilities, is a conditional class probability on the grid cell containing an object

$$p(Class_i|Object) \tag{2.14}$$

[41]. With this explained, the predictions are an $S \times S \times (X * 5 + C)$ tensor, where the number 5 represents each bounding box confidence score as well as the four coordinates: x, y, w, h where (x, y) is the center location and (w, h) is the width and height of the bounding box [41]. In our study, we use $S = 13$, $X = 4$, and $C = 80$. The reasons behind the different numbers are described in Section 2.2.3.

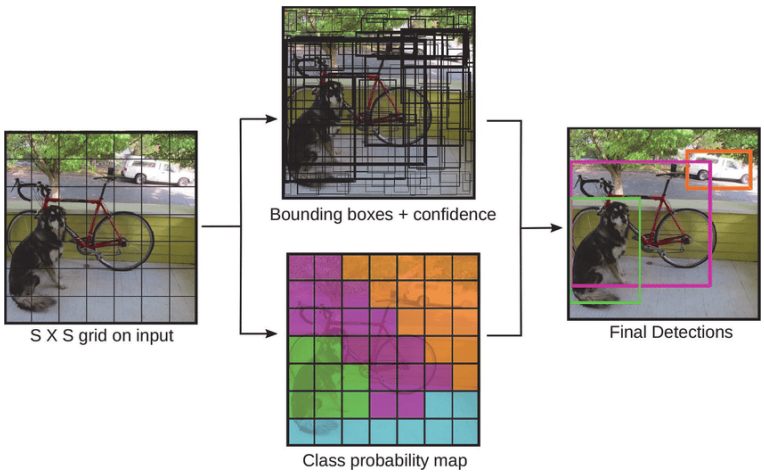

Figure 2.6: The picture illustrates the main idea of YOLO. The method divides the image into regions, an $S \times S$ grid, for each region it predicts X bounding boxes. Each predicted bounding box has a confidence score as well as C class probabilities. [41]

During training, YOLO optimizes the following loss function [41]

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{S^2} 1_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

(2.15)

The first two terms penalize the bounding boxes' coordination predictions, the two following terms penalize the confidence score predictions for the bounding boxes, and the last one penalizes the classification prediction for the bounding boxes. Each loss term does not consider all predicted bounding boxes when calculating the total loss. To understand this and the loss function, we must understand

some fundamental parts behind YOLO. The first thing to acknowledge is that the grid cell where an object's center falls is responsible for detecting that object; this will be referred to as a responsible cell. Furthermore, there are three indicators to understand in the loss function (2.2.2.2). The first one 1_{ij}^{obj} is 1 if an object's center falls inside the grid cell i and if the bounding box j gives the best IOU among all X bounding boxes in that cell, that predicted box would be referred to as the responsible box. The second one 1_{ij}^{noobj} is 1 for all bounding boxes j in a grid cell i where no objects center falls into, to be referred to as the non-responsible cell. The last indicator 1_{i}^{obj} denotes 1 if it is a responsible cell and thus any object's center appears in cell i [41].

If we then go term by term in the loss function(2.2.2.2), the first two terms are only calculated for bounding boxes where 1_{ij}^{obj} is 1. The 1 refers to all responsible boxes, where each should be close to the corresponding ground truth bounding box coordinates (x, y, w, h) to minimize the coordination loss. By taking the square root of the width and height in term two, we want deviations on larger boxes to have a more negligible effect than deviations on smaller boxes. We achieve this by using the square root as it downscales high values to a greater extent than lower ones. The loss terms are multiplied by λ_{coord} to emphasize the localization error [41].

Continuing to the third and fourth terms. These calculate the loss of the predicted confidence score. For the responsible boxes, it should predict as close to 1 and is optimized by term three. However, the loss of the confidence score is calculated not only for the responsible boxes but also for grid cells with no object assigned. For non-responsible cells, the confidence score should be as close to 0 as possible for all bounding boxes. This is optimized by term four in the loss function(2.2.2.2). As most cells do not contain any object, we have to weight this loss-term down to not train the model to detect background more frequently than detecting objects; this is done by λ_{noobj} [41].

The last term refers to the optimization of the classification. It is only calculated for the responsible cells, grid cells responsible for predicting an object. This term minimizes the difference between the conditional class predictions and the label class. Note that in the loss function(2.2.2.2), from original YOLO, this classification error only calculates once per responsible cell as we have the indicator 1_{i}^{obj}. Important to note is that this is not the case for YOLOv2, which is the version used in this study. The difference is that in YOLOv2, this is moved to the bounding box level, and thus it no longer assumes that only one label is assigned to a grid cell. That means the indicator in the loss-class term also changes to 1_{ij}^{obj} and thus penalizes the error for responsible boxes [14].

Another thing that changed in the loss function from YOLO to YOLOv2 is that two more contestants λ_{obj} and λ_{class} are added to weight terms three and five [14].

To summarize, the loss function for YOLOv2 penalizes coordination and classification errors only for predictions that are responsible boxes in the responsible cells, bounding boxes that give the best IOU among all bounding boxes in a grid cell where the ground truth's center falls into. In comparison, the confidence loss penalizes the loss function for grid cells that have and do not have objects assigned, even if the boxes' from responsible cells and non-responsible cell loss are weighted differently [14].

2.2.3 YOLOv2

In this study, YOLOv2, which is an improvement of the first release, will be used. This version focuses on improving objects' localization and bettering the relatively low recall seen in the first release [14]. One thing that was added and worth explaining in detail is anchor boxes. The original version of YOLO predicts bounding boxes using fully connected layers after the convolutional feature extractor [14]. In Faster R-CNN, a two-stage object detector, hand-picked priors are used [43]. In YOLOv2, we also use priors to predict bounding boxes by removing the fully connected layer. Anchor boxes are a set of boxes defined to match the desired ground truths in the training set best [14]. The shape of an anchor box is width and height to match ground truth bounding boxes. Two hyperparameters have to be predefined to decide the anchor boxes: 1. the number of anchor boxes, and 2. their shapes. The YOLOv2 paper suggests using the K-means clustering algorithm to find these [14]. K-means clustering consists mainly of two steps. First, we set the number of clusters and initialize those centers.

The second step is to find the closest cluster for each ground truth in the training dataset and calculate the IOU mean of all objects in each cluster. The mean is then used to recalculate the centers of the clusters. The center of the clusters corresponds to the anchor boxes' width and height. This second step is repeated until two iterations give the same cluster centers, as this means we have found the optimum of the cluster centers, and the algorithm has converged. We run this algorithm for different K clusters to find the optimal number of anchor boxes. The scale of the anchor boxes is between 0 and 1, where the boxes are rescaled to localize objects for each cell.

It is also worth mentioning that YOLOv2 is improved by shrinking the resolution from 448x448 pixels to 416x416. This is because big objects often seem to have the center exactly in the middle of the picture, and thus an odd number of grids are wanted, so we have a single-center cell [14]. Additionally, to improve accuracy and regularize the model we use batch normalization for all convolutional layers in YOLOv2 [14].

Aside from what has been improved, it is also essential to mention what flaws YOLOv2 has. The main limitation of YOLOv2 is that it struggles to detect small objects, mainly due to only predicting objects at one scale [44]. Later versions of YOLO used at least three different scales to be able to predict smaller objects [44].

2.3 Importance sampling

Importance sampling is a technique used as an approximation method in statistics. In its application importance sampling creates a new distribution used to sample from an original distribution. The method is often used because the original distribution is hard to sample from or because importance sampling can act as a variance reduction method, [5]. The sampling method is historically most commonly used in Monte Carlo computing [45], [46], [47], [48] but has also seen implementations in DL [6], [34] [7]. Its DL applications are used to obtain harder samples for the network to handle and hopefully improve the learning. We start by demonstrating the methodology of importance sampling in the context of Monte Carlo computations. Subsequently, we expand the method to the context of DL.

A function $f(x)$ and a density function $p(x)$ has for its continous case the expected value

$$\mathrm{E}_p[f(x)] = \int p(x) f(x)\, dx \tag{2.16}$$

Our goal in importance sampling is to approximate the expected value of $f(x)$. If (2.16) has a high dimensionality it can often be computationally intractable to calculate. To solve this it is possible to approximate the expectation value (2.16) by computing an average over the sampled values from $p(x)$ and computing the function f for the sampled values of x. Resulting in

$$\mathrm{E}_p[f(x)] \approx \underbrace{\frac{1}{N}\sum_{n=1}^{N} f(x_i)}_{m_1}, \qquad x_i \sim p(x) \tag{2.17}$$

By applying the central limit theorem [49] to m_1, which is the result of (2.17), it approaches the distibution

$$m_1 \to \mathcal{N}(\mu, \sigma^2) \begin{cases} \mu = \mathrm{E}_p[f(x)] \\ \sigma^2 = \frac{1}{N}\mathrm{Var}_p[f(x)] \end{cases} \tag{2.18}$$

This equation's variance is limited by the variance of $f(x)$ over the distribution p, so the square root limits the σ. Importance sampling specifically introduces a new distribution when approximating the values, which is fundamentally arbitrary but chosen to $q(x)$. This distribution is used as a substitute

distribution to $p(x)$ and will be used in sampling. We can introduce this new distribution $q(x)$ into (2.17) by rewriting it as

$$\mathrm{E}_p[f(x)] = \int p(x)f(x)\frac{q(x)}{q(x)}\,dx \tag{2.19}$$

This will not change the values of $\mathrm{E}_p[f(x)]$ as long as $q(x) > 0$ whereever $p(x)f(x) \neq 0$. (2.19) can be written as an expectation value over q instead of p:

$$\mathrm{E}_q\left[\frac{p(x)}{q(x)}f(x)\right] = \int q(x)\left[\frac{p(x)}{q(x)}f(x)\right]dx \tag{2.20}$$

As we previously did to obatin (2.17) and (2.18) we are now able to get an approximation of (2.20) by computing an average over the sampled values from $q(x)$ and computing the function $(f * p)/q$ for the sampled values of x. Resulting in

$$\mathrm{E}_q\left[\frac{p(x)}{q(x)}f(x)\right] \approx \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[\frac{p(x_i)}{q(x_i)}f(x_i)\right]}_{m_2}, \qquad x_i \sim q(x) \tag{2.21}$$

In the same way, as we did for m_1, we now get the distribution in which m_2 approaches

$$m_2 \to \mathcal{N}(\mu,\, \sigma^2)\begin{cases}\mu = \mathrm{E}_p[f(x)] \\ \sigma^2 = \frac{1}{N}\mathrm{Var}_q\left[\frac{p(x)}{q(x)}f(x)\right]\end{cases} \tag{2.22}$$

In so, we can now sample $q(x)$.

With (2.21) and 2.22 for the importance sampling we can now aim at reducing the variance of the samples by making it so that the variance in (2.22) become lower than that in (2.18). To accomplish this $q(x)$ should be high where $|p(x)f(x)|$ is high [50] [37]. In making the variance of the sampling alogrithm decrease it is possible to obtain a better approximation of the expectation value faster than previously.

Before we explain how this is done in DL, it is appropriate to demonstrate why importance sampling can help us in CV tasks. In DL, we want to achieve an understanding of some unknown distributions. In IC, for any class A, the model predicts the probability for class A given x, where x is any sample. In order to understand what the model's output describes, we can use Bayes' theorem, then the model's output is $p(A|x)$, called the posterior in Bayes theorem. If we invert the posterior, using the Bayes theorem again, we can understand that the model needs to know a ratio of two pdfs and some trivial prior of class A $p(A)$. In the region where both pdfs are small, the tails will sample rarely, and that is where it will learn more slowly.

Quite early on in the network training, it becomes good at distinguishing between classes where one of the classes' distribution is large relative to the other. During training, the real challenge is to approximate the distribution's tails (i.e., challenging samples). To visualize the tails, we use a relatively easy DL task. We have a binary classification problem where the unknown distribution of the two classes can look like in Figure 2.7. When the model is trying to approximate these distributions, the real challenge is where X is close to 0 in Figure 2.7. In this area, the distributions are similar in size, and both classes rarely fall in this region, so it will naturally take far more samples for the network to become good at distinguishing between classes. Oversampling that area compared to the two classes' combined distribution density will result in the network learning how to handle these samples better. In so, we will be making fewer misclassifications. Two important notes are regarding our reasoning above. Firstly, Figure 2.7 is a visualization of the classes distribution density; in reality, we are sampling points in some unknown high-dimensional hyperspace. However, there will still be an unknown region between the two classes where the model the real challenge, and that is what the figure 2.7 is trying to visualize. Secondly, we have to distinguish between our tasks and DL in general.

In our case, we can assume that the classes are perfectly separable, which is not the case for DL tasks. However, this is the reason it makes sense to oversample the challenging area.

Furthermore, by oversampling the tails, we will be oversampling some samples, and thus we might also adjust the prior class distribution $p(A)$. Therefore, the gradient might be biased because we shift the class priors. We also use "area under the curve" (AUC) as an evaluation metric, as it is invariant under the prior class distributions and thus robust against the bias. More details of AUC will follow in Section 2.4.2.

However, in making the sampling more concentrated on the tails of the distribution where the more challenging samples are located, we reduce the sampling variance as in (2.22). Compared to the Monte Carlo case, where we wanted the sampling to have a high probability where $|p(x)f(x)|$ was high, we, in this case, need some metric that can determine how hard a sample is for the network and then shift the sampling appropriately. When this is accomplished, we select samples more efficiently, seeing where we need to get better at classifying. In so, we possibly make the training of the network more efficient.

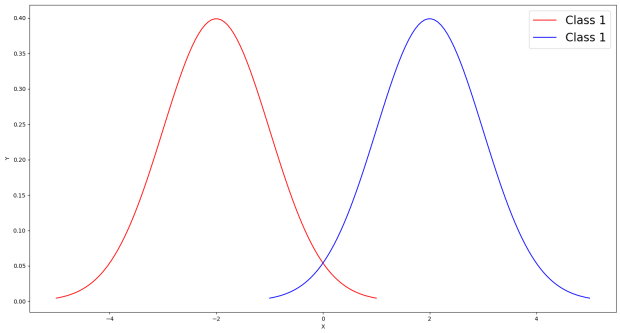

Figure 2.7: Example of two normal distributions with some overlap.

To understand how importance sampling translates to DL, we will start by understanding what we are trying to improve by introducing an importance sampling scheme. At its core, importance sampling for DL is used to obtain more challenging samples at a higher rate than uniform sampling. Within the application of DL, previous studies show that in doing so, we can improve the gradient estimate for models such as classifiers where much of the cost function is made up of a minority of misclassified samples [37]. Earlier studies have concluded that sampling these more complex samples with a higher frequency can reduce the gradient's variance, ultimately resulting in an increased convergence speed [51]. We, therefore, focus on these more complex samples knowing that the network does well enough on the easier ones.

To understand which samples are complex or easy for the network to handle, it is possible to use the gradient of each sample. However, due to the high computational cost caused by continuously calculating the gradients, it is computationally more effective to apply importance sampling to an approximation of the gradient. One such is the loss function shown theoretically by Katharopoulos et al. [7]. Their paper uses the loss function and constructs a framework that reduces the variance and achieves better performance in speed and accuracy than uniform sampling. We apply the same strategy for this thesis, using the loss function directly and sample proportional to it. A summary of the theoretical background to why importance sampling can speed up the training of a network by reducing its variance is as follow.

In the training of our CNN, the goal is to minimize

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(\psi(x_i; \theta), y_i) \tag{2.23}$$

,having x_i and y_i as the i-th output of the trainingset, $\psi(x_i;\theta)$ as the DL model parameterized by vector θ, and N being the number of samples in the trainingset.

Using the equation earlier presented for updating the parameters of Stochastic gradient descent 2.2.1.3, one iteration for updating the parameters in the vector θ is given by

$$\theta_{t+1} = \theta_t - \eta\alpha_i \nabla_{\theta_t} L(\psi(x_i;\theta), y_i) \tag{2.24}$$

with α as a weight of the i-th sample from a distribution p and a learning rate defined as η. Comparing θs of one iteration to the next makes it possible to define the algorithm's convergence speed (noted as S). This is done by using each version's distance from the optimal solution θ^*, resulting in

$$S = -\mathrm{E}_p[||\theta_{t+1} - \theta^*||_2^2 - ||\theta_t - \theta^*||_2^2] \tag{2.25}$$

Much like equation (2.17) for the Monte Carlo algorithm it is possible in this case to define

$$\mathrm{E}_p[\alpha_i \nabla_{\theta_t} L(\psi(x_i;\theta), y_i)] = \nabla_{\theta t} \frac{1}{N} \sum_{i=1}^{N} L(\psi(x_i;\theta), y_i) \tag{2.26}$$

, using this Wang et al. [52] has proven that S is

$$\begin{aligned} S = {} & 2\eta(\theta_t - \theta^*)\mathrm{E}_p[\alpha_i \nabla_{\theta t} L(\psi(x_i;\theta), y_i)] \\ & - \eta^2 \mathrm{E}_p[\alpha_i \nabla_{\theta t} L(\psi(x_i;\theta), y_i)]^T \mathrm{E}_p[\alpha_i \nabla_{\theta t} L(\psi(x_i;\theta), y_i)] \\ & - \eta^2 Tr(\mathrm{Var}_p[\alpha_i \nabla_{\theta_t} L(\psi(x_i;\theta), y_i)]) \end{aligned} \tag{2.27}$$

Looking at the term $\eta^2 Tr(\mathrm{Var}_p[\alpha_i \nabla_{\theta_t} L(\psi(x_i;\theta), y_i)])$ in (2.27), we can see that minimizing this term would lead to a speedup of S. The gradient expressed as the term $\alpha_i \nabla_{\theta_t} L(\psi(x_i;\theta), y_i)$ is used by Katharopoulos et al. [7] to show that the loss can compute a tighter upper bound to the gradient norm than uniform sampling. Katharopoulos et al. [7], proves that the variance reduction achieved by sampling according to the loss achieves similar results to that of the gradient norm. Thus the loss can be used as an approximation of the gradient [7]. In addition to Katharopoulos et al.'s proof, the use of loss as an approximation of the gradient norm can be further motivated by Section 2.2.1.2 where the relationship between loss and gradient is described in detail. In this thesis, the samples are therefore chosen for training at each iteration based on how large the loss ($L(\psi(x_i;\theta), y_i)$) is for each sample. The probability (Loss_i) of each sample being drawn is set by calculating its probability,

$$\mathrm{p}_i = \frac{L(\psi(x_i;\theta), y_i)}{\frac{1}{N}\sum_{j=1}^{N} L(\psi(x_j;\theta), y_j)} \tag{2.28}$$

We have chosen to base the sampling on these probabilities instead of a uniform distribution that selects batches randomly. Performing this probabilistic sampling will make it so that samples with a high loss will be oversampled compared to the majority. In so, the variance which we looked to minimize in (2.27) will decrease and in so decrease S.

2.4 Evaluation metrics

It is essential to have evaluation metrics that allow easy analysis to understand how well a model performs. In this thesis, the confusion matrix has been the foundation of the evaluation. Using the confusion matrix over multiple "thresholds" (explained below), we can create more visual representations such as AUC and "average precision" (AP) for specific tasks. We explain these metrics here.

2.4.1 Confusion matrix

A confusion matrix is one of the more common metrics used to analyze performance in ML [53]. It is an easy-to-understand way of visualizing how good a model is that we utilize in both the IC and the OD part of this thesis. The matrix is split into four boxes, each containing a number representing how many times that event has occurred in the ML model prediction [54]. A visual representation of a confusion matrix can be seen in Table 2.1. The first measurement is true positives (TP), representing the number of times the model can correctly predict an object. If there is an image where no object is present, and the model correctly concludes not classifying any object in that image, then this is the case of a true negative (TN). Note that TN is not used in OD. The latter two metrics are for false classifications. The first is false positives (FP). It is when the model falsely classifies an image class or falsely detects an object when there is no object present. The last, false negative (FN), is when the model does not predict the image as a member of any class or detect an object, but it is in reality. [54].

		Prediction outcome		
		p	**n**	**total**
Actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

Table 2.1: Confusion matrix

Combining different parts of the confusion matrix result in new measurments. One is the true positive rate (TPR) or recall

$$\text{TPR} = \frac{TP}{TP + FN} \tag{2.29}$$

, which is the ratio between all the TP's and the total number of ground truth positives. Another important metric for this thesis is the false positive rate (FPR)

$$\text{FPR} = \frac{FP}{FP + TN} \tag{2.30}$$

, which is the ratio between FP and the total number of ground truth negatives. Lastly, this thesis uses precision,

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.31}$$

, which represents the ratio between TP and the total number of predicted positives, the rate at which the model's predictions are correct.

2.4.2 Area under the curve with Receiver Operating Characteristics (ROC)

An integral part of this thesis evaluation is the area under the ROC curve, or the AUC for short. It is one of the more popular methods for analyzing classification models. Before understanding the

AUC measurement, the ROC must first be understood. The AUC is used widely with different curves that calculate area. In our IC case, the curve that AUC refers to in its name is the ROC. The ROC curve shows a model's performance at all classification thresholds. A classification threshold governs what we do or do not view as belonging to a specific class. At "0", the threshold tells us that all cases belong to a class, and at "1", it tells us that no samples belong to a class. The ROC curve is formed by plotting a model's TPR (y-axis) against its FPR (x-axis), making it a probability curve telling us how the TPR will behave over an increasing FPR. To visualize this process see the Figures in 2.10. With a sliding classification threshold going from the right side of Figure 2.8 towards the left, the ROC curve in 2.9 will be created. These figures represent the simplistic task of classification with two classes. When there are more than two classes, a ROC for each class is created where the class is compared to all other classes for its ROC [54].

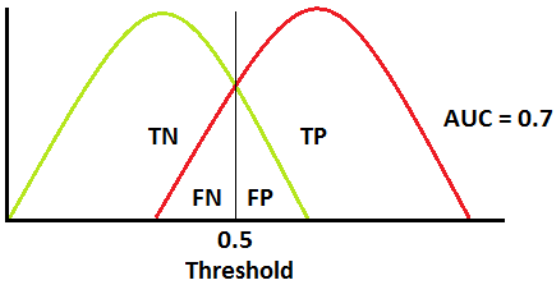

Figure 2.8: Example of a confusion matrix distibutions.

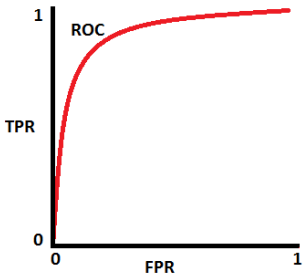

Figure 2.9: Example of a ROC-curve.

Figure 2.10: An example of a ROC-curve and the two distribution that creates it.[55]

To represent the model's overall performance with the AUC for a task with more than two classes, we use the same methodology as for the ROC. Moving from a ROC to an AUC is done by calculating the area under the ROC curve, which is visualized in Figure 2.11. The AUC is calculated for each class in comparison to all others. After that, an average of all AUCs is calculated as the final one [54]. In this study, all classes are weighted equally. This metric works well to indicate how ably the model is at distinguishing classes. The average AUC provides an insensitive evaluation metric and can withstand potentially unbalanced data created in oversampling rigid frames, as in importance sampling. The AUC is a good way to compare importance, and uniform sampling as the average AUC score compensates for unusual sampling frames.

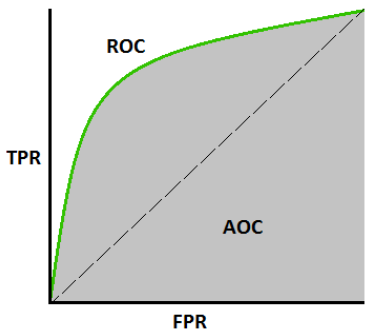

Figure 2.11: Example of how a ROC curve (green line) relate to the AUC (gray area). [55]

2.4.3 Average precision

For OD, the mainly used metric for this thesis is AP. This metric makes use of some parts of the confusion matrix, namely the recall2.29 and the precision2.31 [56]. The AP creates a metric that

combines these two measurements with the intersection over union (IOU)[56]. The IOU is a value between 0 and 1, representing the overlap between a prediction and the ground truth. See Figure 2.12 for a visual representation[56]. It is used to assert if a prediction is a FP or a TP given some threshold that is commonly set at $IOU = 0.5$. The prediction is defined to TP if $IOU > 0.5$ and FP if $IOU < 0.5$[56]. When there are no predictions for the ground truth, it becomes a FN. It is also categorized as a FN if $IOU > 0.5$, but the class is wrong [56]. It is also common to calculate TP and FP for a range of different IOU thresholds[57]. In this study, we only use one threshold; this is the most traditional way of doing it.

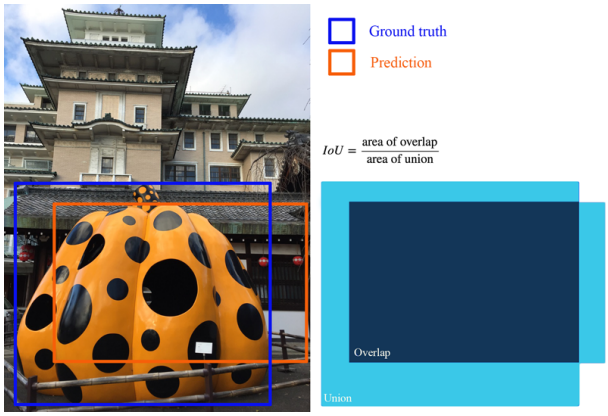

Figure 2.12: How the IOU for a prediction is calculated by being combined with its ground truth. [58]

With the IOU being able to identify which predictions are correct or not, it is possible to calculate the recall and precision according to (2.31) and (2.29). Continue to AP, defined as precision averaged across all recall values. To calculate the AP-curve, we need to rank all predictions, which is done by their confidence score[57]. When all predictions are ranked, we can iterate over them and calculate the precision to the corresponding recall value[56]. To create the precision-recall curve, we plot these against each other. One example of this is in Figure2.13 where the y-axis is the precision, and the recall is the x-axis[56]. Calculating the area under that PR-curve results in AP score[56]. Note that mAP often can refer to the same thing as AP, but it can also refer to taking the mean AP over all classes and overall IOU thresholds[57]. In this thesis, the metric is referred to as AP as we neither use the mean over classes nor IOU thresholds.

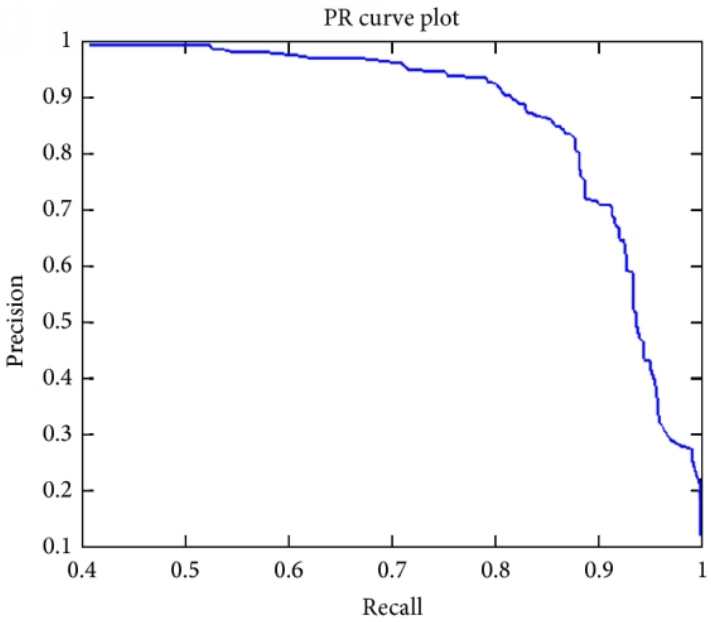

Figure 2.13: Example of how an arbitrary AP curve could look.

[56]

Chapter 3

Related Work

Several previous studies have used importance sampling to accelerate the training of neural networks. One application of importance sampling within DL has been in designing a scheme that provides the neural network with increasing difficulty of samples, reflecting human learning [59]. More closely related to this thesis, several studies use the history of losses, gradients, or an approximation of the gradients for all seen samples as weighting to prioritize training examples during training [5–7, 34, 60, 61]. In this chapter, existing importance sampling methods will be revisited.

Both Schaul et al.[60] and Loshchilov et al.[34] use the losses as the weighting of the samples. Loshchilov et al. [34] propose a simple strategy where all samples are ranked by their last known loss value. The samples were then chosen with a probability that decays exponentially as a function of their loss rank. In the study by Loshchilov et al. [34] they use the MNIST dataset to perform experiments on image classification. The results show that their proposed sampling strategy speeds up by a factor of 5. The paper also states that additional speedup would be possible by linking learning rates and batch size [34]. A limitation of this study is that it does not generalize to other datasets or other DL techniques such as OD. The study made by Schaul et al.[60] also uses the losses to weight samples. Their result obtains faster learning with this approach and proves that the algorithm's results are robust and scalable. They test two variants, one which prioritizes proportional to the loss and another that does rank-based prioritization. Both show to speed up compared to a uniform baseline [60]. The experiments of the study are in Deep Q-Networks, which is a neural network that utilizes deep Q learning [62] [60]. The results are demonstrated on Atari 2600 games, and the network can play games on a human expert level. They show that prioritized experience replay outperforms uniform experience replay on 41/49 games [60]. Uniform experience replay stores all experiences in a replay memory, and during training of the network, draw random samples from memory. To summarize, both studies [34, 60] show promising results for importance sampling methods using loss as weighting. One common challenge for both studies [34, 60] is that the history of losses may poorly reflect the current situation as the model is constantly changing, and thus the loss for each sample does. Because of this, the main limitation of the studies by Schaul et al. [60] and Loshchilov et al. [34] are that many hyperparameters have to control the effect of samples with outdated weights. Schaul et al.[60] use different approaches to smooth out the losses that work as the weighting, and Loschilov et al. [34] control when losses are computed and how sampling distribution is computed based on the ranking by hyperparameters. In contrast, prioritized experience replay prioritizes experiences that the reinforcement learning agent learns most efficiently from [60].

Katharopoulos et al. have done two studies [5, 7] about importance sampling within DL. The first one, which was released in 2017[7], shows that the loss can be used as an importance metric during training in DL. The study results show that importance sampling effectively reduces the training time compared to uniform sampling. Furthermore, the study[7] theoretically shows that faster convergence is achieved by reducing the variance of the gradient, something that in this case is done through importance sampling based on each sample loss. The paper show both theoretically and empirically that the training of a DL network can be accelerated by using loss as an importance metric. Their

study is based on image classification and language modeling tasks using deep convolutional and recurrent neural networks. As the loss requires a complete forward pass, they propose a creation of a parallel model to be able to approximate the loss and thus the importance of the samples to have a low computational overhead. In comparison to their first study, Katharopoulos et al., in their second study, released in 2018 [5], present importance scoring based on an upper bound to the gradient norm. They use an upper bound for the gradient norm and not the actual gradient norm due to it being computationally prohibitive. The upper bound can still be computed with only one forward pass. In addition to that, the approach presented in the paper uses the upper bound to predict how useful importance sampling will be. In so, it decides when to switch the importance sampling on and off during training. The study [5] empirically proves that the importance sampling method achieves lower training and test loss after equal length on three tasks: image classification, sequence classification, and fine-tuning [5].

Another paper using gradient norm as an importance metric is Alain et al. [6]; this paper uses an approach where gradient norms are calculated without storing the gradients themselves. This study only applies to MLPs and not CNNs where parameter sharing is present. The results do show that the importance sampling method led to significant improvements in training [6]. However, their approach is computational heavy as more than a single forward pass of the network is needed. A sixth study [61] claims an algorithm that is not sensitive to hyperparameters that neither requires a calculation of all gradient norms, which is consuming to calculate. The study proposes a robust approximate importance sampling that achieves at least a 20% speed up. Their proposed importance sampling scheme approximates the gradient with an uncertainty set and then minimizes the worst-case value of all possible gradient norms in that uncertainty set. The exact calculation and algorithm can be found in their report [61]. However, compared to others presented work, this algorithm is not sensitive to hyperparameters as robust approximate importance sampling trains the uncertainty set in an adaptive manner[61].

3.1 Additions to the field

This study aims to add value to the understanding of importance sampling by doing the same experiment on different datasets, providing more robust evidence for our importance sampling results. A majority of previous studies have been carried out focusing on one dataset in determining the convergence speed of their importance sampling [5–7, 34, 60]. Studies on importance sampling have shown that the method outperforms uniform sampling in many tasks. However, no earlier study has determined how importance sampling varies in performance depending on several different methods used in sampling. Neither has any study determined how importance samplings performance depends on the neural networks' complexity. This study has explored different batch generators for importance sampling and tried all experiments on two different neural network models to achieve context for when importance sampling can be fruitful. While many of the previous studies on importance sampling have examined the possibilities of the method when implemented in IC, [5, 7] research on its implementation on an OD problem is something that remains largely unexplored. Adding an assessment on importance sampling for OD can hopefully further contribute to understanding the method.

Furthermore, most of the earlier mentioned studies [5, 7, 34, 61] show that IS outperforms uniform sampling in convergence speed, but commonly they do not evaluate until full convergence is reached. Two studies do evaluate until convergence is reached, [6, 60], but their experiments are not performed on CNNs. The study by Alain et al. is performed on MLPs, and the study by Schaul et al. is performed on Deep Q-Networks. With that said, we would like to make sure what happens later on during training when comparing IS to NIS also for CNNs, as CNNs are one of the most commonly used DL networks today [63].

Finally, the most common evaluation of network performance in the above studies seems to be evaluation or training loss [5–7, 34, 61]. As we know, importance sampling might get a biased estimate due to oversampling of informative samples. Thus changing the prior, we propose an evaluation metric

robust to this bias. This is important to be able to check the network performance in an unbiased way.

Chapter 4

Method

This thesis aims to systematically investigate the possibility of improving an OD network's training times through importance sampling strategies. In the following paragraph, we describe the methodology of investigating whether importance sampling is a powerful way of decreasing the training time of neural networks.

The first step was to study scientific papers to identify promising methods that could improve the training time by oversampling the frames that significantly impact the network's learning. Part of the result from this study is in Section 3. The next step was to create an environment where we used uniform sampling on an IC and an OD problem. We created this using Python and TensorFlow4.3. After ensuring the network worked well with uniform sampling, we implemented the importance sampling methods. The central part of the overall process was developing different importance sampling methods we could apply to the network's training. Later, these methods were evaluated on both the training and validation dataset to see how well they managed to speed up the training. These two steps were done agilely; designing and building new setups was followed by testing and reviewing them. It involved changing hyperparameters to understand if the weighting of samples affects the training time and how they do so. In addition to the theoretical analysis explaining how importance sampling can reduce training time by variance reduction, our developed sampling methods were empirically proven by conducting experiments. The result can be seen in Section 5.

This section will now explain the methods we have applied during the thesis. We split it to explain IC and OD separately, each going through the different datasets and networks they use. The section ends with an explanation of which framework we used to solve all CV tasks.

4.1 Image Classification

This section describes the datasets used in the training and evaluation of the IC tasks. It also provides information about which network architectures we used in the image classification experiments.

4.1.1 Data

The experiments on IC in this thesis are done using the German Traffic Sign Recognition Benchmark (GTSRB) dataset and the Mapillary traffic sign (Mapillary) dataset. The GTSRB dataset is a single-image set containing 43 classes at a total of 51,840 images[64]. The data was collected from a video recording of a 10-hour drive in Germany on different road types. Each sign in the dataset consists of a sequence of images referred to as a track. After data collection, the data was compiled by discarding tracks with less than 30 images and discarding classes with less than nine tracks[64]. The reason for keeping several images for each track is that since the traffic sign can vary a lot over the complete track, for instance, at a high distance, the traffic sign may result in low resolution, and closer ones result in motion blur[64].

The Mapillary dataset has, in comparison to the GTSRB dataset, a variety of weathers, seasons, time of day, and cameras [65]. The full dataset has a global geographic reach consisting of 100,000 images collected from 6 continents with over 300 traffic sign classes [65]. The actual dataset consists of bounding box annotations for detecting and classifying traffic signs worldwide. This study used the cropped version of all traffic signs bounding boxes, making it an IC problem instead of an OD problem.

The main results of this thesis stem from experiments on the Mapillary dataset, while the results from the GTSRB dataset function essentially as a way of validating the results from Mapillary and generalizing the results. The GTSRB dataset was split into the same two subsets, the training and validation set, having a 3:1 split ratio for all experiments. For Mapillary, the training and validation set was split randomly with a ratio of 7:3 for each experiment.

4.1.2 Network Architecture

This section describes the architecture behind the two models used for the IC problems. We utilize architectures which will be referred to as CrapNet (Section 4.1.2.2) and ResNet (Section 4.1.2.1). Experiments carried out on both networks use similar hyperparameters in training. Both utilize the same batch size of 256 and the Adam optimizer. The length we train each network is a difference between the two. The more complex ResNet architecture converges significantly faster than CrapNet in our experiments because of its greater capacity to solve complex tasks. The baseline used for the models is what we present in the following sections. Depending on the experiment, the networks may differ in how they look. All convolutional layers in the two networks are followed by batch normalization to stabilize training, speed up convergence, and regularize the model [14], and lastly, a ReLU activation function. For clarity these are not visualized in the architecture display in Tables 4.1 and 4.2. When downsampling the image resolution, both ResNet and CrapNet use convolutional layers with stride 2. In these cases, the number of filters doubles. For ResNet, we use five stride-two layers, which decreases the resolution to a 32:rd (2^5) of the original input. For CrapNet, two stride-two layers decrease the resolution to a fourth (2^2) of the original input. At the end of each network, we use global average pooling to get a $1 \times \mathit{channels}$ vector, on which we then apply a fully-connected layer for classification. Lastly, both networks apply a softmax activation function.

4.1.2.1 ResNet

ResNet is a residual network that, when introduced, was mainly focused on improving learning in general. Another significant contribution from ResNet is that it also combats the problem of vanishing gradients [66] and enables training on a high number of layers while maintaining good performance. The architecture solves the problem by adding so-called "shortcut connections" that send information from one layer to another to skip multiple layers. Specifically for the thesis, we use a version of ResNet, which is called ResNet-18. The architecture was created in 2015 by Kaiming et al. [67]. The architecture of the network can be seen in Table 4.1. The network consists of mostly convolutional layers, one fully connected layer, one max-pooling layer, a global average pooling layer, and a softmax function. The network starts with a convolutional layer with a kernel size of 7×7 filters, followed by a max-pooling layer. Most of the network consists of 16 convolutional layers, each applying 3×3 filters. The architecture ends with a global average pooling before applying a fully connected layer and a softmax function for a final output of classes $\times$ 1. Missing in the table are the shortcut connections that occur every other layer for ResNet-18 starting from the max-pooling layer [67].

Type	Filters	Size/Stride	Output image resolution
Convolutional 1	64	7 ×7/2	112 ×112
Maxpool		3 ×3/2	56 ×56
Convolutional 2	64	3 ×3/1	56 ×56
Convolutional 3	64	3 ×3/1	56 ×56
Convolutional 4	64	3 ×3/1	56 ×56
Convolutional 5	64	3 ×3/1	56 ×56
Convolutional 6	128	3 ×3/2	28 ×28
Convolutional 7	128	3 ×3/1	28 ×28
Convolutional 8	128	3 ×3/1	28 ×28
Convolutional 9	128	3 ×3/1	28 ×28
Convolutional 10	256	3 ×3/2	14 ×14
Convolutional 11	256	3 ×3/1	14 ×14
Convolutional 12	256	3 ×3/1	14 ×14
Convolutional 13	256	3 ×3/1	14 ×14
Convolutional 14	512	3 ×3/2	7 ×7
Convolutional 15	512	3 ×3/1	7 ×7
Convolutional 16	512	3 ×3/1	7 ×7
Convolutional 17	512	3 ×3/1	7 ×7
Avg. pooling		Global	classes ×1
Fully connected			classes ×1
Softmax			

Table 4.1: The architecture for ResNet-18

4.1.2.2 CrapNet

CrapNet is a simple CNN benchmark explicitly created for this thesis, for which architecture can be seen in Table 4.2. It is a simpler network than ResNet but still responds well to image classification tasks. The networks used in real-time applications are much more complex than any network we use. They also have much more data than parameters in their network. The networks are therefore limited compared to the size of the datasets they use in training. Even if those networks are far more complex than CrapNet, it might be a good comparison as the smallness of the network compared to the size of the dataset resembles each other situation. CrapNet, with its longer learning curve, also serves the purpose of exploring how different architectures react to IS. The architecture consists of 7 convolutional layers, one global average pooling layer, a fully connected layer, and a softmax function. It initiates with seven convolutional layers where the first layer applies 7×7 filters and the following six use 3×3 filters. The first convolutional layer differs from all others by not using ReLU or batch normalization. CrapNet ends in a global average pooling, a fully connected layer, and a layer with softmax.

Type	Filters	Size/Stride	Output image resolution
Convolutional 1	8	7 ×7/1	28 ×28
Convolutional 2	8	3 ×3/1	28 ×28
Convolutional 3	8	3 ×3/1	28 ×28
Convolutional 4	16	3 ×3/2	14 ×14
Convolutional 5	16	3 ×3/1	14 ×14
Convolutional 6	32	3 ×3/2	7 ×7
Convolutional 7	32	3 ×3/1	7 ×7
Avg. pooling		Global	classes ×1
Fullly connected			classes ×1
Softmax			

Table 4.2: The architecture for CrapNet.

4.2 Object Detection

This section describes the datasets used in the training and evaluation of the OD tasks. It also provides information about which network architecture we used in the OD experiments and details how we use said network.

4.2.1 Data

For OD tasks, a variety of popular datasets are in use today. Common ones are COCO [68] and Pascal VOC [69] for 2D OD, and KITTI [70] and NuScenes [71] for 3D OD. While we use multiple datasets in the IC experiments for the thesis's OD part, we mainly wanted to explore if the methods would translate. Due to this goal and time constraints, we implemented one dataset chosen to be COCO. The dataset consists of 328,000 images containing 2.5 million labeled objects that belong to 91 classes of everyday objects. The thought behind the dataset's content is to show objects in their natural habitat to make it as realistic as possible and make it practically applicable [68]. The classes used in the dataset are "entry-level" categories, i.e., a German Sheppard dog is a member of the class

"dog" and not specifically its breed. In contrast, COCO, which we use in IC, has images not explicitly created for the dataset. Instead, it is collected from online images using search motors such as Google and Bing. In collecting data, they wanted a majority of so-called "non-iconic" images, which are images with more than one object that also contain contextual information. To avoid making the dataset contain too many similar images, we added a constraint of a maximum of five images from one photographer during a short time window. For our experiments, we have used parts of the entire dataset. More precisely, there are 82,081 images in the training set and 40,137 in the validation set, roughly a ratio of 2:1 for training and validation.

4.2.2 YOLOv2 Anchor Boxes

As explained in the background about YOLOv2 2.2.3, we need to find anchor boxes for the COCO dataset. We achieved this with the K-means clustering algorithm. The first step in the algorithm is to look at the menu IoU. The results of the clusters with the different number of anchor boxes can be seen in Figure 4.1 where it is easy to see that more clusters result in a better mean IoU of all objects in the dataset. We expected this result as the optimal solution would have been to have K = N objects, then the mean IoU would have been 1. Nevertheless, as it is very computationally heavy to have even nine anchor boxes, it is common to use an elbow curve [72] to decide on a point where diminishing returns are no longer worth the additional cost. To find the elbow on the curve, we look at where the slope of the mean IoU stops being "substantially" large. Given our curve, which can be seen in Figure 4.2, at least four or more anchor boxes would be a reasonable choice as the rise of the curve flattened out after $N = 4$. In Redmon et al.'s paper [14] they use five anchor boxes which made us try the experiments on both four and five anchor boxes. An important note is that when using anchor boxes, there is a risk of having too specialized predictors. Thus some objects may not achieve IoU of 50% with predefined anchor boxes. However, as this study aims not to create the best object detector but instead to create a network that can compare sampling methods, this is ok. When adding the anchor boxes, the class prediction mechanism also changes slightly from the original YOLO. Now the class is predicted for every anchor box instead of predicting class probabilities per grid cell as in the original YOLO [14]. This improvement increases the recall [14].

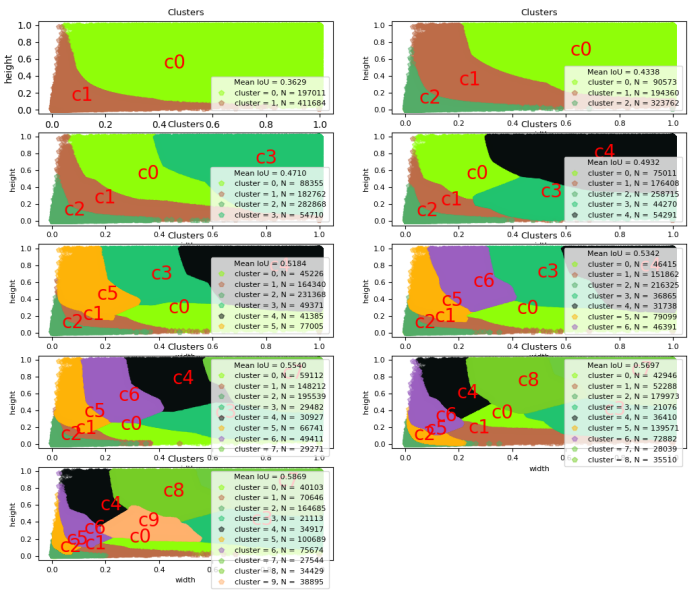

Figure 4.1: Our IoU mean for different number of Anchor Boxes. Ranging from two to ten Anchor Boxes.

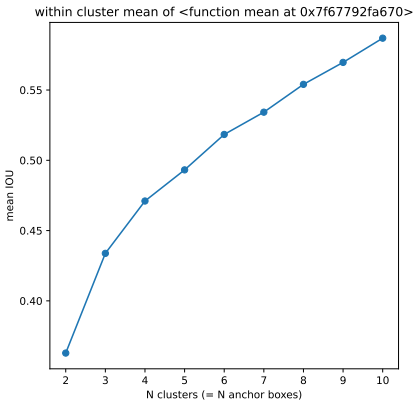

Figure 4.2: Our elbow curve for different number of Anchor Boxes. The idea is that the true number of clusters is captuered when the increase in the mean IoU slope is "substantially" large. In this case, four or five anchor boxes may be a good size.

4.2.3 Object Localization

As we have already evaluated importance sampling methods for IC, the most interesting part to analyze in OD is the object localization. Therefore experiments with YOLOv2 without the classification have been performed. We accomplish this by removing the classification part of the network. This is done by changing filter size from anchorsboxes×85 to anchorboxes×5. The change alters the dimensionality of the final convolutional layer from the output space. Thus, the final output tensor was changed to $S \times S \times (X \times 5)$ instead of $S \times S \times (X \times 5 + C)$. As a result, the rest of the code needed to change accordingly. The main change was that the loss function in (2.2.2.2) remove the classification error as it is not taken into account when optimizing the neural network

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

(4.1)

4.2.4 Network Arcitecture

This section explains the networks which YOLOv2 uses as its backbone. Much of what we applied to the networks used in IC also applies to the OD network. All experiments use the Adam optimizer. A major difference compared to IC is that a limiting factor is GPU memory when we operate on high-resolution images, which we need for the localization of objects. The batch size varies between experiments and is between 16 and 128. It is important to note that what we present in the following sections is the baseline used for the models. Depending on the experiment, the networks may differ in how they look. Unlike IC, the network used in OD was pre-trained. The convolutional layers and the batch normalization layers were initialized with weights. The model was pre-trained on COCO, and the weights were taken from YOLO's official website[73].

There are some cases where we want to see how performance depends on a model's complexity. For those experiments, we increase the trainable parameter count of the model. This does not add any new layer to the model, as it is done by widening the models and increasing the channel count of all convolutions.

4.2.4.1 DarkNet

DarkNet is the network architecture we use for OD. It was introduced in 2016 by Redmon et al. [14]. The core architecture consists of 23 convolutional layers, five max-pooling stride-two layers, and a layer used to reshape the output. The structure can be seen in Table 4.3. For the convolutional layers, the model switches between using 3×3 and 1×1 filters [14]. A notable difference between convolutional layers that use 3×3 and those that use 1×1 filters is that the ones that adopt 1×1 use half the number of filters in their layer. Most convolutional layers are followed by batch normalization and a ReLU function. The final output from the model is of shape $13 \times 13 \times 4 \times 5$ if we use four anchor boxes. 13×13 represents all the cells in the grid, 4 represents the bounding boxes predicted for each cell, and 5 represents the bounding box confidence score plus its x, y, w, and h coordinates.

For DarkNet, one shortcut connection is added between the 13th and the 21st convolutional layer. DarkNet, like the other networks, utilizes stride-two, which, when applied, cuts the size of each axis in the feature map/output in half. We double the number of filters used from there on out for these layers.

Type	Filters	Size/Stride	Output image resolution
Convolutional 1	32	3 ×3/1	416 ×416
Maxpool 1		2 ×2/2	208 ×208
Convolutional 2	64	3 ×3/1	208 ×208
Maxpool 2		2 ×2/2	104 ×104
Convolutional 3	128	3 ×3/1	104 ×104
Convolutional 4	64	1 ×1/1	104 ×104
Convolutional 5	128	3 ×3/1	104 ×104
Maxpool 3		2 ×2/2	52 ×52
Convolutional 6	256	3 ×3/1	52 ×52
Convolutional 7	128	1 ×1/1	52 ×52
Convolutional 8	256	3 ×3/1	52 ×52
Maxpool 4		2 ×2/2	26 ×26
Convolutional 9	512	3 ×3/1	26 ×26
Convolutional 10	256	1 ×1/1	26 ×26
Convolutional 11	512	3 ×3/1	26 ×26
Convolutional 12	256	1 ×1/1	26 ×26
Convolutional 13	512	3 ×3/1	26 ×26
Maxpool 5		2 ×2/2	13 ×13
Convolutional 14	1024	3 ×3/1	13 ×13
Convolutional 15	512	1 ×1/1	13 ×13
Convolutional 16	1024	3 ×3/1	13 ×13
Convolutional 17	512	1 ×1/1	13 ×13
Convolutional 18	1024	3 ×3/1	13 ×13
Convolutional 19	1024	3 ×3/1	13 ×13
Convolutional 20	64	1 ×1/1	26 ×26
Convolutional 21	1024	3 ×3/1	13 ×13
Convolutional 22	1024	3 ×3/1	13 ×13
Convolutional 23	20	3 ×3/1	13 ×13
Reshape			13 ×13 × 4 × 5

Table 4.3: The architecture for DarkNet.

4.3 Framework

Throughout the experiments in this thesis, the end-to-end platform for ML, Tensorflow[74], is the DL framework we use. Tensorflow implements automatic differentiation as the underlying principle of gradient computation. To differentiate automatically, Tensorflow needs to know the order of operations during the forward pass. By remembering this, Tensorflow traverses the graph of operations and computes the gradients during the backward pass[75]. More closely, we used the tf.GradientTape API, provided by Tensorflow, for automatic differentiation[75]. This API records the order of operations inside the context of a tf.GradientTape onto a *tape* and then use this to compute gradient using reverse mode. Thus the loss for the current batch is calculated as a forward pass inside the GradientTape as shown in Code-snippet 1. Both are passed as sources to the gradient method to get the final gradient of the loss for the trainable parameters in the model[75].

Algorithm 1 Code sample of gradients calculation using Tensorflow tf.GradientTape API

```
with GradientTape as tape:
    y = model(x)                                          ▷ Forward pass
    loss = loss_function(y)                               ▷ Forward pass
gradients = tf.tape.gradient(loss, trainable_variables)   ▷ Backpropagation
```

Chapter 5

Experiments

In this section, we present all experiments used to analyze the behavior of the proposed importance sampling schemes. The section is divided between 5.1, 5.2.1 and 5.2.2. Common for all is that we compare every experiment using importance sampling to an experiment with uniform sampling. For readability, this chapter will refer to these as IS (importance sampling) and NIS (uniform sampling). Instead of the average AUC score, we use the 1-AUC score to compare sampling methods in IC. We report it on a logarithmic scale where all classes contribute equally. Note that, opposite to the AUC score, this metric achieves higher performance as the values decrease. In OD evaluation, we use the metrics recall, precision, and AP.

The baseline of how we use the IS scheme in all our experiments is as presented in (2.28). To perform IS, we need to get the losses for all samples in the dataset. Therefore we need to complete one iteration over the entirety of the dataset. Before we initialize the training, the strategy is to set every sample loss in the loss table to a much higher value than they will ever receive from training. Every sample is likely to be selected for training once before any of them is selected a second time. In turn, IS has no effect during the first epoch of the training.

For robustness, each experiment presented has been running 20-100 times, and the average is what we report. One important thing to point out is that all the experiments deviate slightly from the theoretical analysis presented in Section 2 by sampling Mini-batch gradient descent instead of Stochastic gradient descent. In addition, the Adam optimizer is used instead of the constant learning rate presented in (2.24). Other than that, there is some variation in how the IS is sampled depending on the experiment. These variations of the sampling will be discussed for each experiment separately.

5.1 Image Classification

This section represents the majority of all experiments carried out for this thesis. In Section 5.1.1 we explain experiments examining how a network's complexity affects IS performance. In Section 5.1.2, we review how the weighting for losses implementation affects IS in different training stages. Lastly, in Section 5.1.3 a comparison between different IS methods is presented. In order to check IS performance at a later stage in training, we run the experiments long enough to reach full convergence. We partly motivate this by lacking previous studies analyzing IS performance in convergence. To summarize what the experiments of this chapter aim to investigate, we use the following four points.

1. If IS can gain convergence speed compared to NIS.
2. How IS is related to model complexity in terms of the number of parameters.
3. How different models will differ in performance by making the losses exponentially larger or smaller.
4. How different IS methods perform against each other.

5.1.1 Model Complexity

While previous studies, see Section 3, show that IS outperforms NIS in terms of convergence speed. This study aims to analyze if there is any correlation between IS and a model's complexity to gain a deeper understanding of the effectiveness of IS.

We decided to customize the models by changing the number of trainable parameters to set up the experiments, thus getting a range of different model complexities. The two model architectures, ResNet and CrapNet, were used where ResNet is more complex than CrapNet. We use each architecture, and its range of model complexities for all experiments we perform to map out how model complexity affects IS performance.

Initially we present CrapNet displaying the average 1-AUC results for three different models respectively having 2.1×10^5 (Figure 5.1), 3.4×10^5 (Figure 5.2), and 5.2×10^5 (Figure 5.3) number of trainable parameters. The first one, Figure 5.1, which is less parameterized, clearly shows that IS performs better faster than NIS. One can also see that the final performance does not differ significantly between NIS and IS. Similar results are found for the other, more heavily parameterized CrapNet models see Figures 5.2 and 5.3.

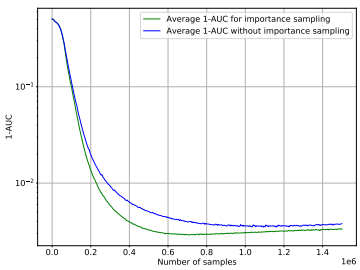

Figure 5.1: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on CrapNet with 2.1×10^5 trainable parameters using the Mapillary dataset.

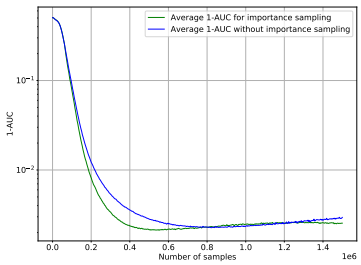

Figure 5.2: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on CrapNet with 3.4×10^5 trainable parameters using the Mapillary dataset.

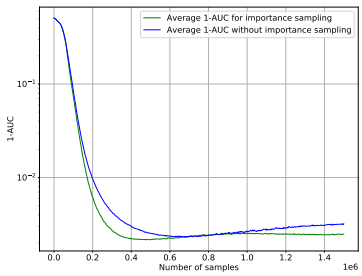

Figure 5.3: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on CrapNet with 5.2×10^5 trainable parameters using the Mapillary dataset.

Our similar experiments for ResNet can be seen in Figures 5.4, 5.5, and 5.6. In the first one, Figure 5.4, IS still outperforms NIS. In the other two Figures, 5.5 and 5.6, the models are heavily parameterized. They most likely overparameterize compared to the dataset, and the effect of IS seems not to be as competitive anymore. In some cases, it even seems to reach worse performance in the later stage of training5.5.

To get an overview of the performance of IS in correlation to the model complexity, we ran the experiment for a range of different numbers of parameters for each type of network. For CrapNet, it ranged between 25k and 520k trainable parameters for a combined ten different models. Moreover, for ResNet, it ranged between 120k and 9300k parameters for ten different models. The experiment results are shown in two types of graphs, one showing the convergence speed against the number of trainable parameters and the other showing max performance against the number of trainable parameters. The convergence speed, on the y-axis, in Figures 5.7 and 5.9, expresses how soon we reach max performance in terms of number of samples. The maximum performance is where we reach the average maximum performance. The average performance over 2000 batches was compared against each other to find this average maximum performance. As we are interested in where the convergence starts, the earlier one was selected if two average values had the same performance. As mentioned, the models have been trained for a long enough time to ensure convergence, and thus average maximum value will reflect where the model starts to converge.

Starting with CrapNet, Figure 5.7 represents the convergence speed versus the number of pa-

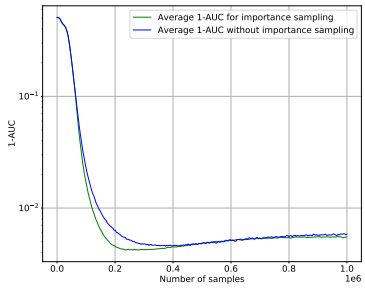

Figure 5.4: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on ResNet with 2.9×10^6 trainable parameters using the Mapillary dataset.

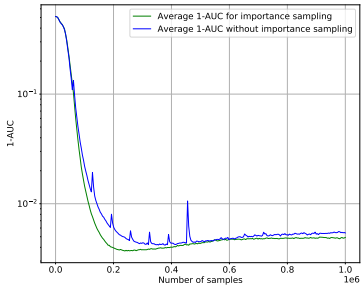

Figure 5.5: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on ResNet with 5.6×10^6 trainable parameters using the Mapillary dataset.

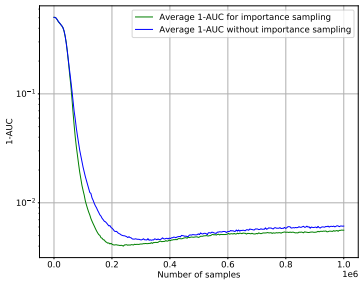

Figure 5.6: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on ResNet with 9.3×10^6 trainable parameters using the Mapillary dataset.

rameters, and Figure 5.8 the max performance versus the number of parameters. The figures are complements of each other, as no conclusion can be made from only analyzing the convergence speed without checking the performance at a particular convergence stage. In both figures, the values of one standard deviation are shown by the area around the two lines. These are used to understand how the 100 runs are distributed, thus giving us insight into how stable the methods are.

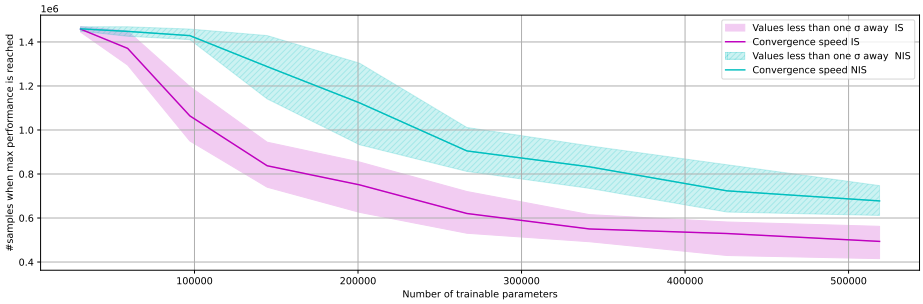

Figure 5.7: A comparison of convergence speed depending on the number of trainable parameters in the network. The plot compares NIS and IS and displays each sampling method's average performance with one standard deviation. The y-axis shows how many samples have been processed before the network's 1-AUC value converges; convergence is given by which 2000 batches have the highest average performance. The experiment is on CrapNet using the Mapillary dataset.

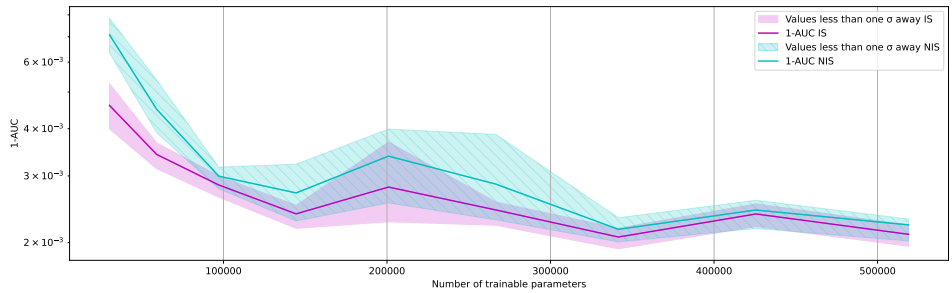

Figure 5.8: How the performance of a model varies based on its number of trainable parameters. The performance is given by the model 1-AUC score at its highest average performance over 2000 samples. The runs are on CrapNet using the Mapillary dataset.

Looking at the comparison between the sampling methods for CrapNet in Figure 5.7, one can see that IS reaches convergence faster than NIS. In addition, Figure 5.8 shows that IS performs better or is similar to NIS. The experiments on ResNet shows similar results, see Figure 5.9 and 5.10.

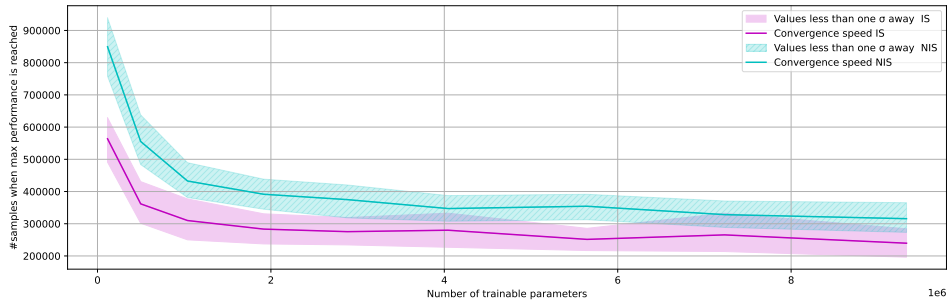

Figure 5.9: A comparison of convergence speed depending on the number of trainable parameters in the network. The plot compares NIS and IS and displays each sampling method's average performance with one standard deviation. The y-axis shows how many samples have been processed before the network's 1-AUC value converges; convergence is given by which 2000 batches have the highest average performance. The experiment is on ResNet using the Mapillary dataset.

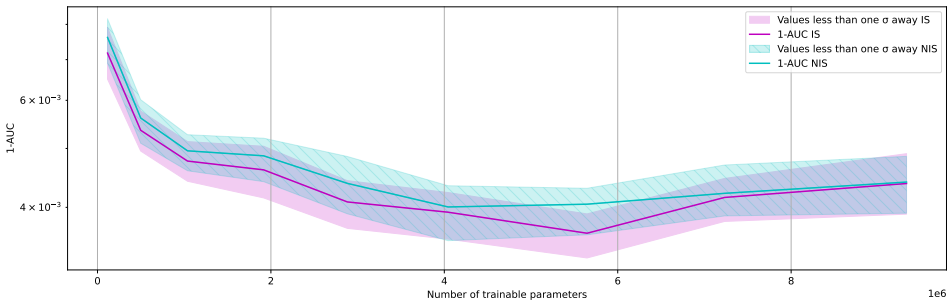

Figure 5.10: How the performance of a model varies based on its number of trainable parameters. The performance is given by the model 1-AUC score at its highest average performance over 2000 samples. The runs are on ResNet using the Mapillary dataset.

Therefore, one can argue that IS needs fewer samples to reach max performance and get the same or better asymptotic performance. We cannot argue for any clear correlation between IS and the complexity of either ResNet or Crapnet from the results of the Mapillary dataset. One can argue that IS would be more beneficial for less complex model architectures, represented by CrapNet (see Figure 5.7), as the difference between IS and NIS convergence speed is slightly larger than for ResNet (see Figure 5.9). In addition, less complex models naturally take a longer time to train, thus increasing the need to speed up the training.

Earlier, we discussed the costs of training a neural network. Increasing training times extensively for a minor increase in performance, which we might be able to achieve in convergence, is not always possible or profitable. One can argue that IS would be more useful when full convergence is not needed, which is often the case. By that logic, IS could be advantageous independent of how it performs in convergence. An arbitrary comparison of NIS and IS can be seen in Figure 5.11. This comparison shows that IS outperforms NIS significantly during the first half of the training. When the model reaches a stage of convergence, NIS's 1-AUC score becomes almost identical to IS's. We should add that the comparison is made regarding the number of samples, not the actual time. As we know, IS needs a complete forward pass to compute the weighting of samples. Therefore it would be an engineering challenge to not incur any runtime overhead from sampling the images tagged by IS. That said, IS should be viewed as the preferred sampling method when full convergence is not needed. When max performance is needed, the conclusion of whether IS or NIS is preferred is not as clear.

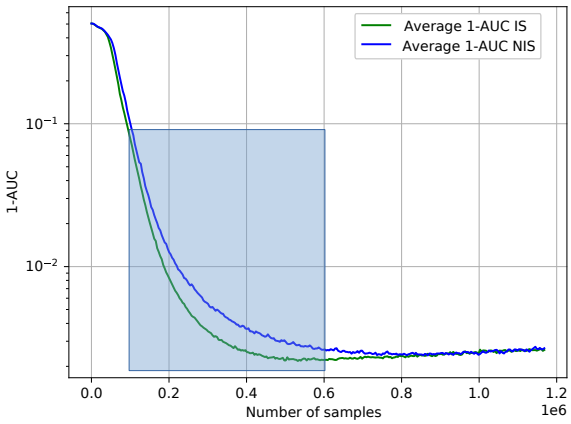

Figure 5.11: Comparison of 1-AUC score for NIS and IS over number of samples on CrapNet. The window clearly showing that IS outperforms NIS.

When checking the results for GTSRB dataset, IS continue to reach max performance earlier than NIS see Figures 5.12 and 5.14. When looking at average max performance in Figure 5.13 and 5.15, NIS clearly reach better final performance than IS. These results would further motivate that IS is not preferred when max performance is needed. Before deciding whether IS should be preferred when max performance is not needed, we must discuss the differences in the datasets we use. As mentioned in Section Data 4.1.1 GTSRB is a more narrow dataset than Mapillary. Therefore, the Mapillary dataset results should be considered as the immediate results, but it is interesting to see how IS performance varies depending on multiple parameters.

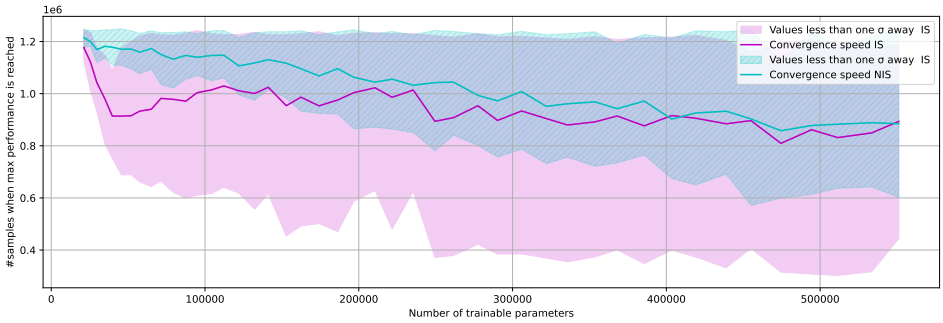

Figure 5.12: A comparison of convergence speed depending on the number of trainable parameters in the network. The plot compares NIS and IS and displays each sampling method's average performance with one standard deviation. The y-axis shows how many samples have been processed before the network's 1-AUC value converges; convergence is given by which 2000 batches have the highest average performance. The experiment is on CrapNet using the GTSRB dataset.

Figure 5.13: How the performance of a model varies based on its number of trainable parameters. The performance is given by the model 1-AUC score at its highest average performance over 2000 samples. The runs are on CrapNet using the GTSRB dataset.

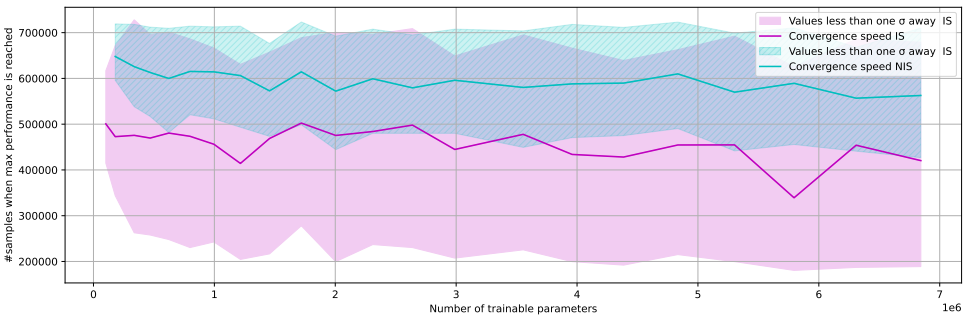

Figure 5.14: A comparison of convergence speed depending on the number of trainable parameters in the network. The plot compares NIS and IS and displays each sampling method's average performance with one standard deviation. The y-axis shows how many samples have been processed before the network's 1-AUC value converges; convergence is given by which 2000 batches have the highest average performance. The experiment is on ResNet using the GTSRB dataset.

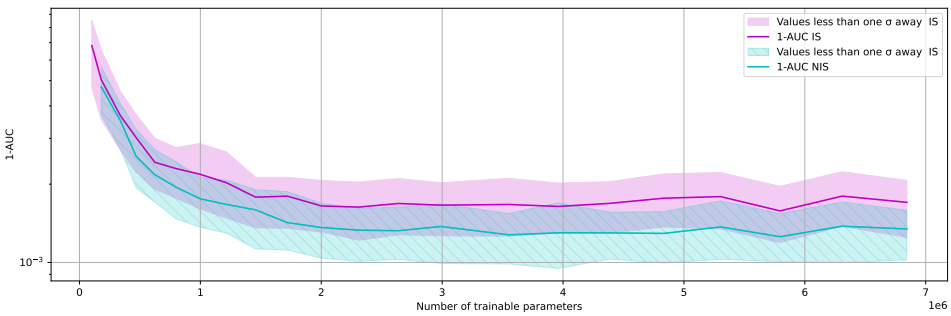

Figure 5.15: How the performance of a model varies based on its number of trainable parameters. The performance is given by the model 1-AUC score at its highest average performance over 2000 samples. The runs are on ResNet using the GTSRB dataset.

When checking the results of the 1-AUC score for the GTSRB dataset using ResNet, one representative result is found in Figure 5.16, the model learns almost everything in the first two epochs and then only improves its 1-AUC score marginally. It suggests that our model is overparameterized, and thus all challenging samples can be learned by heart. We believe this could be why NIS outperforms IS regarding the final performance.

Looking at CrapNet for the GTSRB dataset, one can see that the least parameterized models in Figure 5.17 are still very advantageous in IS when looking at convergence speed. When the number of parameters increases, as in Figure 5.18, for the CrapNet model, the results differ from what we observed on the Mapillary dataset. IS reaches better or equal performance for the figure's first 10-15 epochs. After that, IS becomes much worse in performance and oscillates upwards. This explains why Figure 5.13 is showing better maximum performance for NIS. One potential explanation could be that the model gets biased toward the more challenging samples and "forget" the easier ones. However, drawing any definitive conclusions regarding this is hard without deeply analyzing specific samples chosen in IS.

With the results from the different datasets, network architecture, and network complexity in mind, we can say that IS correlates with model complexity in some cases. Regarding the above results, what can be said is that our IS method mainly performs worse or is equal to NIS when max performance is needed. Whether IS is preferred when max performance is not needed depends on how complex the model is compared to the dataset, but possibly more things. If the model is not overparameterized, it is more likely that IS would be preferred, as motivated by the Mapillary results.

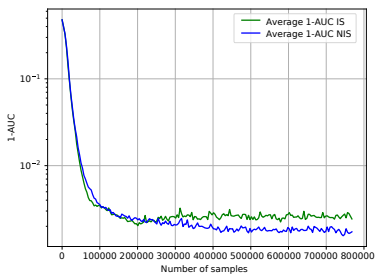

Figure 5.16: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on ResNet with 3.9×10^6 trainable parameters using the GTSRB dataset.

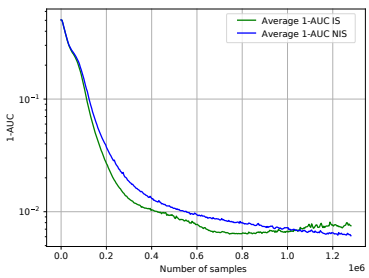

Figure 5.17: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on CrapNet with 2.1×10^4 trainable parameters using the GTSRB dataset.

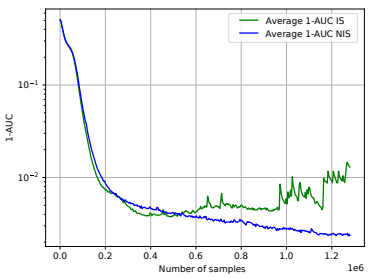

Figure 5.18: Comparison of 1-AUC score for NIS and IS over number of samples. The runs are on CrapNet with 2.1×10^5 trainable parameters using the GTSRB dataset.

5.1.2 Weighting of Loss Function

In assessing the effectiveness of IS, a comparison of how weighting each sample loss would affect the convergence speed is yet to be conducted. In the past, there have been studies examining how weighting losses are connected to the performance of a network, for example, using dynamically-weighted loss functions for prognostics in the automotive and aerospace industry [76]. Said study does, however, not involve IS. Given (2.28) which is used for calculating the probability of each sample getting chosen, a growing curiosity in how weighting losses by exponentially scaling came to be. We make it so that the loss distribution differences become larger or smaller in terms of percentage through exponential scaling. This result in an increased or shrunk difference in the probability distribution of all samples. To set up this experiment we needed to update (2.28) by adding an exponential variable ω,

$$\text{Loss}_i = \frac{(L(\psi(x_i;\theta), y_i)^\omega}{\frac{1}{N}\sum_{j=1}^{N}(L(\psi(x_j;\theta), y_j)^\omega)} \tag{5.1}$$

We use the resulting equation to create the new probability distributions, which we then sampled. To determine whether a large or small exponent was most suited for increasing the convergence speed, several different values of ω were used. Varying from experiment to experiment, the ω's used were: $[0.1, 0.5, 1.0, 1.5, 2.0]$, all of which were compared to NIS.

We introduce our results from running this experiment on the Mapillary dataset. In Figure 5.19 the experiment can be seen on the ResNet network, where the network has a total of 2.7×10^6 trainable parameters. As expected, there is no or only a tiny difference between each loss exponent in the beginning. After the first epoch, when IS takes effect, the higher exponents (1.5, 2.0) 1-AUC go down faster than the lower ones (0.1, 0.5), and all IS exponents perform better than the line corresponding to NIS. After this initial performance boost, which occurs during the second epoch, the different exponents vary in performance over the following several epochs. We can see a clear performance difference between the higher and the lower exponents. Here the 1-AUC for the highest exponent (2.0) starts becoming larger as it finally converges with the highest 1-AUC among the used exponents. The same behavior is observed, but not to the same extent, for other high loss exponents such as 1.0 and 1.5, where the 1-AUC score converges at roughly the same value. The lower exponents 0.5 and 0.1 converge later in their runs at lower values than those of the higher exponents.

After the initial peak in performance from higher loss exponents, loss exponent 0.1 becomes the exponent with the lowest 1-AUC value and remains so for the rest of the experiment. We also observe that some high exponents obtain a worse 1-AUC score than NIS midway through the experiment. In the end, NIS converges at a 1-AUC value similar to the high loss exponents. When looking at Figure 5.19 a pattern becomes apparent. Partly that the higher the loss exponent, the earlier the peak, but also that they in their peak beat out every other runs 1-AUC. We also see that the lower the exponent, the better the performance in convergence.

In Figure 5.20 the corresponding experiment but on CrapNet can be seen where the network has 2.1×10^5 trainable parameters. Other than the behavior of the different IS exponent coming far later in these runs, caused by the differences between CrapNet and ResNet, changing the model does not change much in behavior. The most significant difference compared to Figure 5.19 is that NIS cannot outperform any exponent. Comparing the CrapNet result to the ResNet ones, we see the same performance increase for the higher exponents in the beginning. That switches during the third epoch as lower exponents start getting better performance. A difference between CrapNet and ResNet is that we cannot see as clearly that smaller exponents are the best in convergence as they are far closer together in performance.

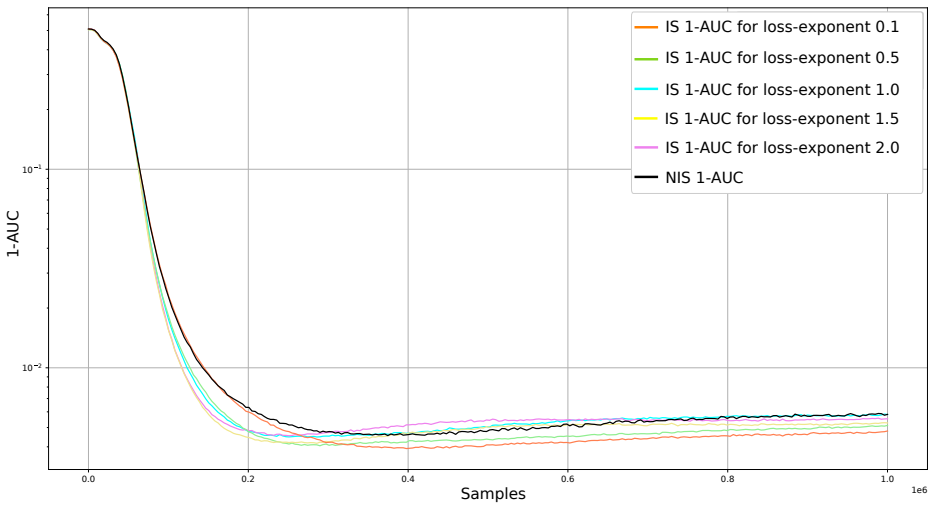

Figure 5.19: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for ResNet with 2.7×10^6 trainable parameters on the Mapillary dataset.

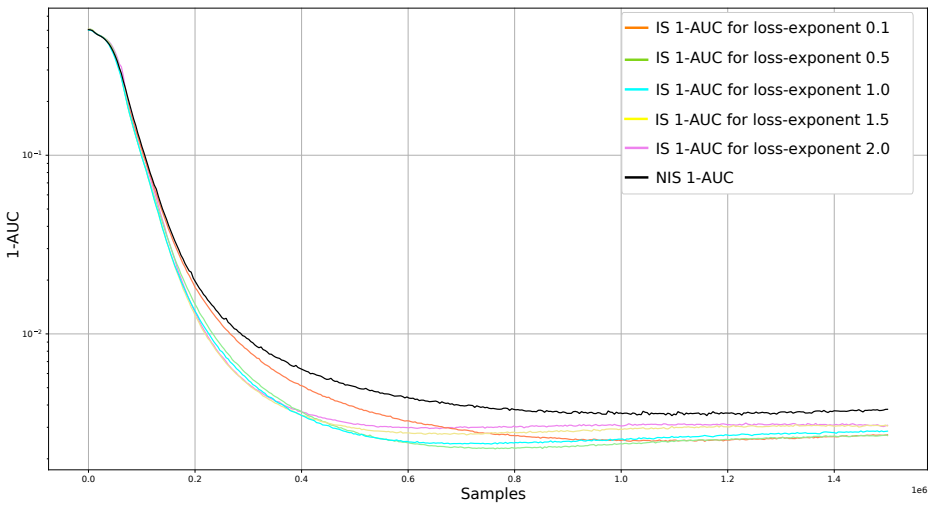

Figure 5.20: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for CrapNet with 2.1×10^5 trainable parameters on the Mapillary dataset.

When performing the experiments on the GTSRB dataset, as seen in Figure 5.21 for the ResNet network, we observed the same behavior as ResNet did on the Mapillary dataset. After that, the result becomes much harder to distinguish between as they are relatively close in performance. It is possible to see the previous behavior where higher exponents perform better early and peak early. In convergence, we see a clear difference between high and low exponents as the low ones perform far better. Compared to the experiment on the Mapillary dataset, NIS is quite close to being the best performer throughout the run. Some IS runs beat it, but in convergence, it is apparent that only the lowest exponent, 0.1, can compete with its 1-AUC score.

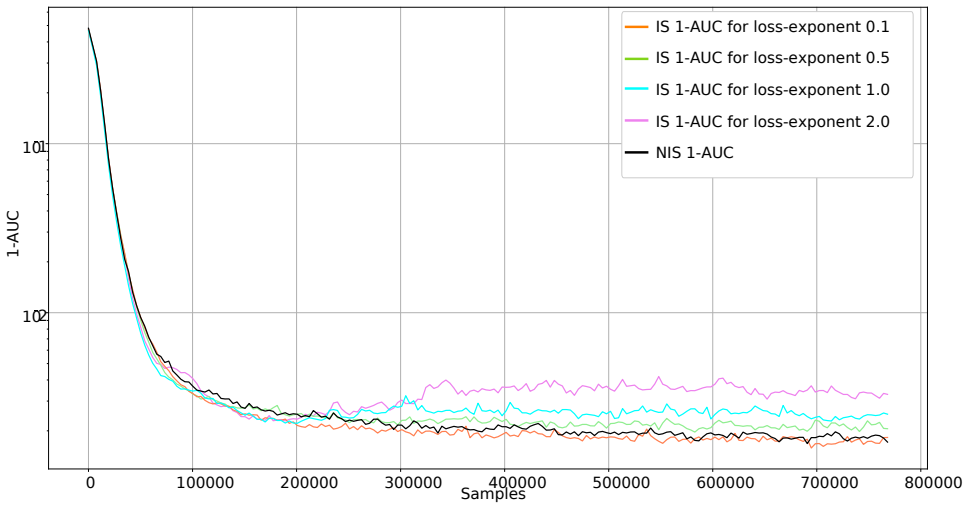

Figure 5.21: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, and 2.0 and NIS. The runs are for ResNet with 2.7×10^6 trainable parameters on the Mapillary dataset.

In Figure 5.22 we see the same experiment for the GTSRB dataset on CrapNet. The result tells much of the same story. The observed behavior in Figure 5.21 is far more exaggerated in this case. There are two significant differences between this experiment and all previous ones. First, it seems as if the higher exponents not only peak once but twice and also get the same oscillations that we saw in Figure 5.18. Secondly, NIS beat all IS sampling methods in convergence.

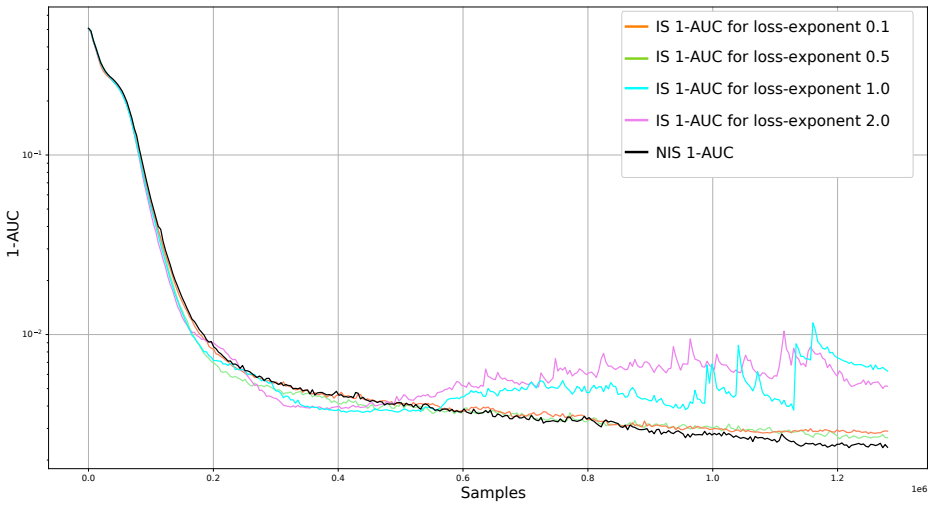

Figure 5.22: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, and 2.0 and NIS. The runs are for CrapNet with 2.1×10^5 trainable parameters on the GTSRB dataset.

In combination with the experiments described in Section 5.1.1 the experiments on loss exponents were expanded. To build upon our earlier analysis of the correlation to the model complexity, we made sure to do the experiments on both ResNet and CrapNet with three different number of parameters of each model. For CrapNet that is 2.1×10^5, 3.4×10^5, and 5.2×10^5 trainable parameters and for ResNet that is 2.7×10^6, 6.3×10^6, and 10.6×10^6 trainable parameters. The result of all experiments can be seen in Appendix A7 from 7.1 until 7.4 for the experiments on the Mapillary dataset, and from 7.5 until 7.8 for the experiments on the GTSRB data. An overview of which loss exponents we used on each experiment can be seen in Table 7.1.

The patterns we have already noticed in the previously described figures are present when looking at the plots in said table. First is a segment where higher exponents achieve the lowest 1-AUC scores. From there on out, the IS exponents, starting with the highest and then moving on to lower ones, peak out and slowly start getting worse. In convergence, the highest exponents get the highest 1-AUC scores. In contrast, the lower ones get an incrementally better score until they converge with the lowest exponent achieving the lowest 1-AUC. In the beginning, the runs with NIS attain the worst 1-AUC score. In the later phase, the performance varies depending on the dataset. For the GTSRB dataset, the performance of NIS is better than all but the very lowest loss exponent (0.1). For Mapillary, it cannot compete with the lowest exponent but can achieve similar performance to high IS depending on the model complexity.

We are unsure what causes the high exponents to get the worst 1-AUC score in convergence. It could be explained by oversampling challenging samples more frequently, leading to a bias toward those samples. A potentially better solution than the ones we have displayed in this section would be to use a high exponent in the beginning, and once it no longer outperforms lower ones, make a switch in how we sample. This switch could either be using NIS or having a decaying exponent. We have toyed with a decaying exponent for this thesis but did not manage to execute the idea at a level where it speeds up the training.

We determine that the performance of the different weighting of samples presented in (5.1) gave no or minimal effect when comparing against different trainable parameters for each network type. The only significant difference we see is the one between CrapNet and ResNet.

5.1.3 Importance Sampling Methods

We have used the same method of selecting samples for all previous experiments when performing IS. To clarify, that is the IS scheme where we select the samples by calculating each sample's probability based on their loss and then sample based on that probability. This method is referred to as IS-probabilistic in this section. While we have seen promising results in previous sections where the IS-probabilistic scheme outperforms NIS in some aspects of the training, we wanted to evaluate how our sampling process would compare to other IS variants. To accomplish this, we introduce two new methods for IS. The first one, called IS-randomsubset, selects a random subset of all samples and then selects a batch consisting of the samples with the highest losses from the subset. The size of the subset that we used was 1/10 of our training set. The second variant, called IS-highsubset, selects a subset consisting of the 1000 highest losses. With this subset, we then select a batch based on the probability given by each sample loss.

IS-randomsubset immediately showed great first indications in our tests. However, IS-highsubsets performance was way worse than any other IS method or NIS, and it became apparent quickly that it would be useless to us. After we ensured that the method performed the task it was set out to do and realizing it feathered no result, we moved on from it. One possible explanation for the lack of performance of the IS-highsubset is that we only select the same high loss samples for the subset repeatedly. Although, we have not been able to confirm that this was the case. An observation made from toying with IS-highsubset is that not seeing easier samples is devastating to network learning. Consequently, the undesirable performance excluded IS-highsubset from being run 100 times as we otherwise have done for all our experiments. Therefore IS-highsubset will not be represented in the comparisons we make in this section.

The comparison presented and discussed in this section is between NIS, IS-probabilistic, and IS-randomsubset. The methods have been benchmarked against the same networks presented in Section 5.1.2. For CrapNet that means comparisons with models using 200 906, 341 498, and 518 954 trainable parameters. For ResNet that means comparisons with models using 2 707 420, 6 308 486, and 10 650 054 trainable parameters. We conducted these on the Mapillary and the GTSRB datasets like in previous experiments.

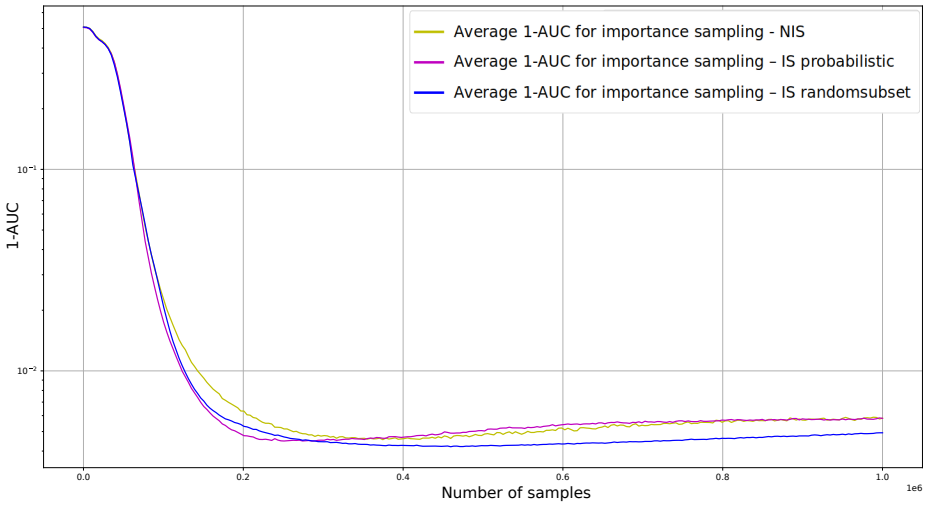

Figure 5.23: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 2.7×10^6 trainable parameters on the Mapillary dataset.

In Figures 5.23 and 5.24 the experiment is presented when applied to Mapillary and GTSRB respectively. The experiment is done on the ResNet network with the fewest trainable parameters among the used networks. As seen in previous sections, IS-probabilistic outperforms NIS at the beginning of training. This performance can be seen for IS-randomsubset as well.

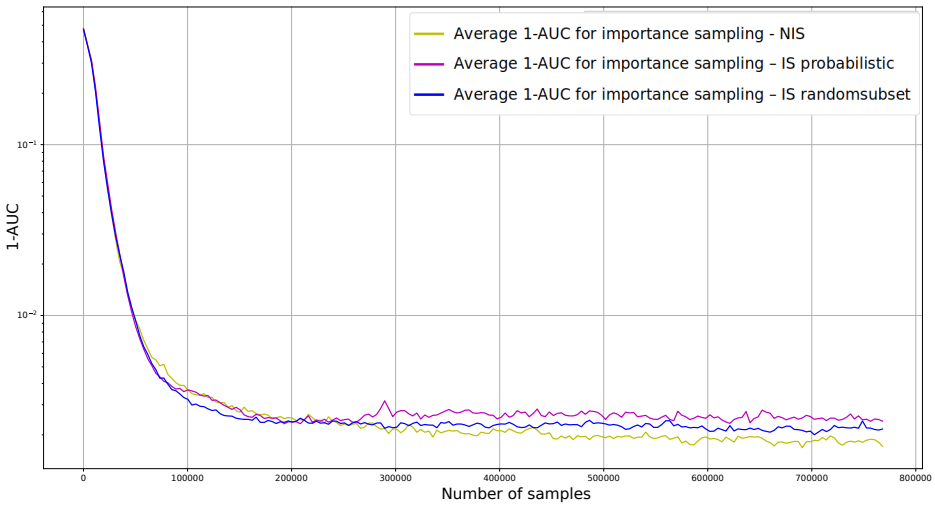

Figure 5.24: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 2.7×10^6 trainable parameters on the GTSRB dataset.

Interestingly, when we reach convergence, IS-randomsubset outperforms both NIS and IS-probabilistic for the Mapillary dataset. IS-probabilistic, as previously seen, performs almost identically to NIS in convergence on Mapillary. For GTSRB in convergence, IS-randomsubset can outperform IS-probabilistic once again. Similar to IS-probabilistic, it cannot achieve an as good 1-AUC value as NIS.

If we only compare the two IS methods with the two datasets, we observe that both methods start by achieving a similar speed-up of the training. These continuous improvements are maintained for IS-randomsubsets 1-AUC score while IS-probabilistic 1-AUC slowly achieves tinier improvements. This pattern of IS-randomsubset outperforming IS-probabilistic in later stages of training is something we observe in the majority of our experiments on ResNet. Since IS-probabilistic mainly lacked performance compared to NIS in its convergence, this indicates that IS-randomsubset could be advantageous. To view the experiments for all the different trainable parameters of ResNet see Appendix B7 Figures 7.9 and 7.10 for the Mapillary experiments and 7.13 and 7.14 for the GTSRB experiments.

When it came to applying this experiment to CrapNet, we see the result for the model with 200 906 parameters in Figures 5.25 and 5.26. The same behaviors that we were able to detect in ResNet runs are observable here. Both IS methods outperform NIS initially; IS-randomsubset can maintain its steady improvement for the 1-AUC score while IS-probabilistic fades in performance. In convergence, NIS outperforms both IS methods for the GTSRB dataset while IS-randomsubset outperforms both for the Mapillary dataset. For all results of the CrapNet experiments see Appendix B7 Figures 7.11 and 7.12 for the Mapillary experiments and 7.15 and 7.16 for the GTSRB experiments. Taking all experiments into account on CrapNet, we see that the findings on CrapNet, in large, further reinforce what we saw for ResNet.

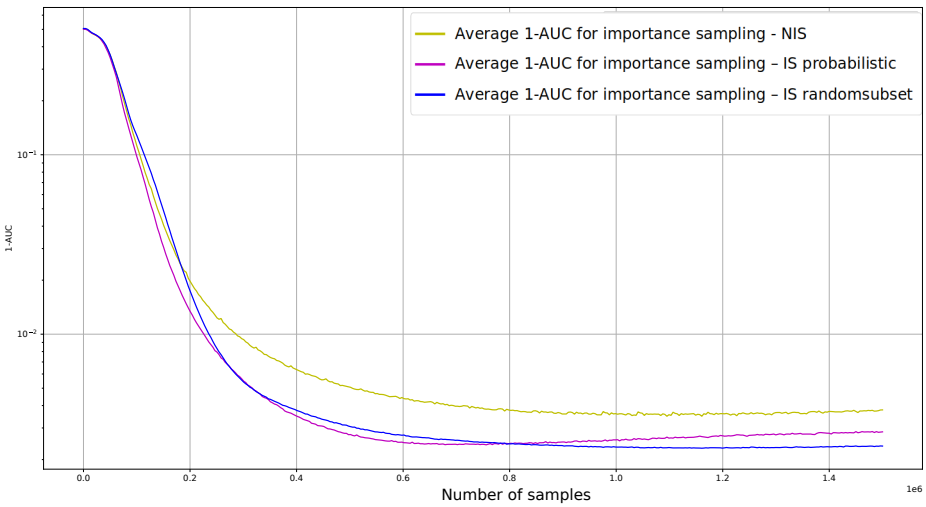

Figure 5.25: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 2.1×10^5 trainable parameters on the Mapillary dataset.

To generalize our findings from this section, we see that IS methods perform better than NIS in the initial training phases. In convergence, NIS lacks performance for the GTSRB dataset but is still outperforming it for the Mapillary dataset. Other than some small segments of the runs for specific network comparisons between the two IS methods, this section shows that IS-randomsubset outperforms IS-probabilistic in the long rung. This is an important note from this section, as we in the earlier Sections 5.1.1 and 5.1.2 identify a challenge with IS in it not reaching an as good convergence performance as NIS. With that said, the two IS methods, IS-probabilistic and IS-randomsubset, still give us results that indicate that they are the preferred choice compared to NIS when max performance

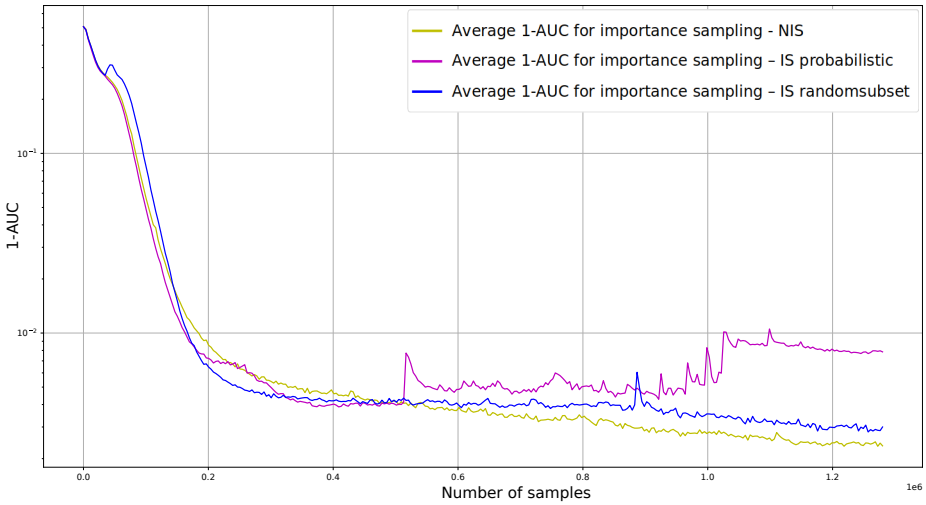

Figure 5.26: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 2.1×10^5 trainable parameters on the GTSRB dataset.

is not needed. What is new from this section is that IS-randomsubset also seems to be preferred when max performance is needed, motivated by the Mapillary results. We see a need to study IS-randomsubset more deeply to make the findings robust. A good start would be to implement it for an OD task.

5.2 Object detection

This section presents all experiments that have been made on the OD task. It will include a description of what experiments were done and how they were conducted. In addition, it will also include the challenges we have encountered.

5.2.1 Object detection with classification

We are starting with the experiments on OD that include classification. Figure 5.27 shows 1-AP_{50} score after each epoch except during the first, where we wanted to evaluate it twice as we expected larger changes in the beginning. AP_{50} is the AP score using an IOU threshold of 0.5, which means the predicted bounding box must have overlapped more than 50% with the ground truth bounding box to be classified as a true positive. We can mainly state from this figure that the 1-AP_{50} score in general, both for IS and NIS, does not reach good performance. As we compared YOLOv2 with other methods, see Table 5.1, people can achieve higher performance with other methods. Note that the performance for YOLOv2 on COCO is 44 % in this table, but reaching this requires many augmentations, tuning an LR schedule, and probably more. Our results reached after 70 epochs were merely 28 % AP_{50}. It shows that the network learns a lot during the first ten epochs and keeps improving for 20 more. After that, the training seemed to converge. YOLOv2 was chosen as a method before knowing it was inadequate for COCO.

As mentioned in Chapter 4, we compare OD using a pre-trained DarkNet architecture. Pre-trained means that the network weights have already been adapted to identify relevant features in the images. What is pre-trained, to be precise, is DarkNet-19 which is the backbone of YOLOv2. It is a CNN that extracts feature maps from the image. In YOLO, the backbone is then followed by the neck and head, which is the part of the YOLO architecture that predicts the bounding boxes and classification of the objects. However, worth mentioning is that we initially ran some experiments with DarkNet-19 backbone that was not pre-trained, but as the performance only reached around 11 % AP_{50}, we chose to continue with a pre-trained backbone. All results presented in this section and Section 5.2.2 use a pre-trained backbone. Furthermore, we have used a batch size 32.

Coming back to the bad performance of our implementation. Due to that, we wanted to exclude that the model was learning the wrong thing. Therefore we ran training on two minimal training sets, one with one image and one with ten images. The results showed that the loss reached almost 0 on the training sets and that we could detect almost all objects in the images. From this test, we could exclude that the model tried to learn the wrong thing. Therefore, extensive experiments of this kind excluded the fact that anything was wrong with our code.

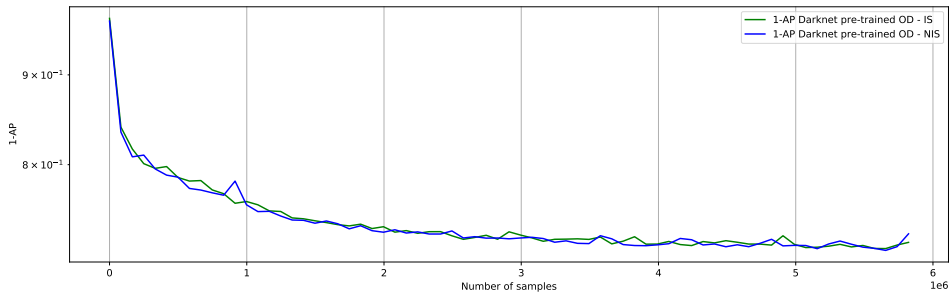

Figure 5.27: 1-AP_{50} score comparision for IS and NIS in OD for COCO. The DarkNet backbone is pre-trained and evaluated every epoch.

Practically implementing OD is highly complex, but we wanted to do it from scratch to have complete control. YOLOv2 was not the best solution for COCO; if we had had more time, another method should also have been implemented. However, the small data tests described above have been

Method	Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2

Table 5.1: OD results brought from [77]. AP_{50} score comparision between different OD methods for COCO dataset. From the comparison one can see that YOLOv2 is not a state-of-the-art model. [77]

used to verify that our implementation is entirely correct. To narrow this experiment further, we decided to remove the classification part. We could do this because IS was already evaluated on the classification part earlier in the thesis. We state the results of this change in the next section.

5.2.2 Object detection without classification

Moving on to OD without classification, we will present what challenges we saw, what we tried that did not work, and what we find interesting for future studies. In order to remove the classification, we mainly changed the loss function by removing the term that creates an error when the class prediction is wrong. This could be done as the classification is done independently of objects' localization.

5.2.2.1 Challenges identified with Object Detection

The first thing to describe will be the challenges identified with OD. It is essential to understand which challenges we have identified with our implementation of OD before going through any conclusions we could draw from the results. We first verified that we could still perfectly solve the task on the small training sets of one and ten images when removing classification. For this task, our model managed to predict correct localizations on the training set, and thus we could conclude that the new loss function optimizes the right thing. One example from these results can be seen in Figure 5.28.

In this case, the model learns the images by heart when using a small training set. However, when using the full COCO dataset for training with the same network, we still only achieved an AP_{50} score of 28 %. With proof that we cannot achieve good results on large datasets, we checked the predicted detections on a set of about 100 images. The aim was to check if we could identify any pattern of the model's difficulties in learning. Our conclusion from the experiment with the trained model on both the validation and training sets was that larger objects could be detected more often than smaller ones, which is a known flaw for YOLOv2 [14] as YOLOv2 only has one output scale. To alleviate the scale issue, we tried to use the Deep Layer Aggregation [78] network as a backbone, which operates at eight times the resolution compared to DarkNet. We did not get immediate results, but this should be investigated further. Nevertheless, the important takeaway is that YOLOv2 can not solve COCO well. As this thesis aimed to compare IS and NIS and not create a state-of-the-art model, we decided to check if there was any difference between IS and NIS, even though the model was not reaching good

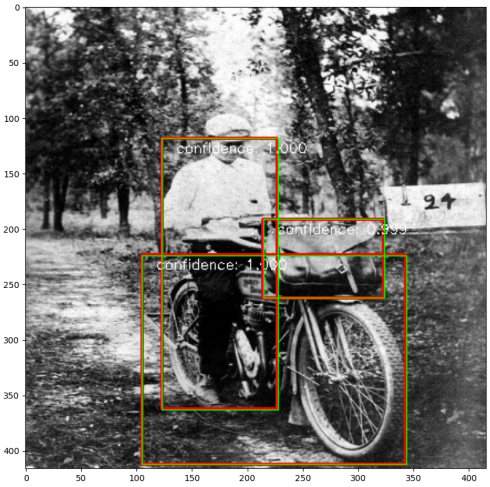

Figure 5.28: Detection results on one image that was included in a training set of ten images. Red boxes are predictions, green boxes are the ground truth boxes. White text shows confidence score for each predicted box.

performance.

Figure 5.29 clearly shows no difference between IS and NIS. We see a potential explanation for the lack of difference between IS and NIS by checking the loss distribution. The loss distribution after one training epoch is visualized in Figure 5.30. Most of the samples were in a small range. Rechecking it later in training, see Figure 5.31, the range was still not as expected. By checking Figures 5.30 and 5.31, one can understand that the probabilities for the different samples will not differ that much, and thus IS will not have any significant impact on the choice of samples. Going back to what we said before, we believe the main reason behind our lack of results comes from the model not being good enough. The model is not in a stage where it produces correct predictions for the bulk of the samples in the training set. Due to this, most of our losses are not close to zero. Ultimately the impact on IS is that the distribution has no clear tail, and thus we can not find more informative samples. Compared to the IC case, we had a clear tail in the distribution, where all informative samples were located.

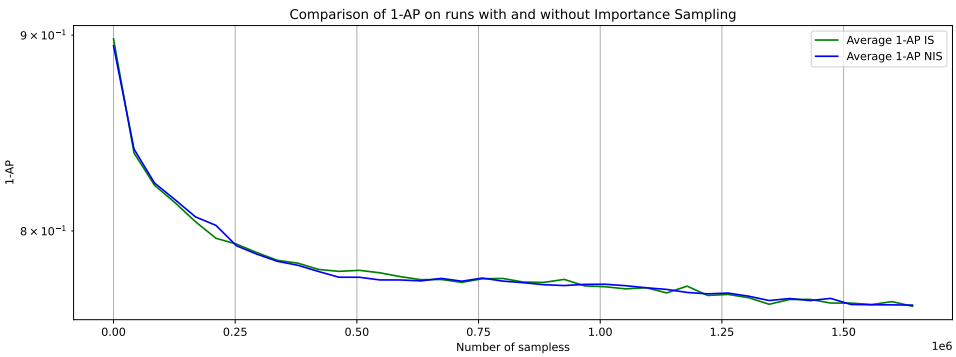

Figure 5.29: 1-AP score comparision for IS and NIS in OD w/o classification for COCO. The DarkNet backbone is pre-trained and evaluated every epoch.

However, even if the model could not produce correct predictions for the bulk of our samples, we wanted to investigate if IS could make a difference in speed up even before this stage. When drawing samples with a probability proportional to the loss distribution, given in Figure 5.30, the samples will

more or less have similar probabilities. Furthermore, even if there is a difference in how informative the samples are, it will not be shown with the current IS method. To clarify, we wanted to investigate whether the images with a loss of 950-1050 are significantly more informative than those around 500 or if all images are approximately equally informative. We did experiments by scaling the losses and using an IS method where all samples were ranked according to their loss instead. The adjusted IS methods we tried are described in the following Section 5.2.2.2.

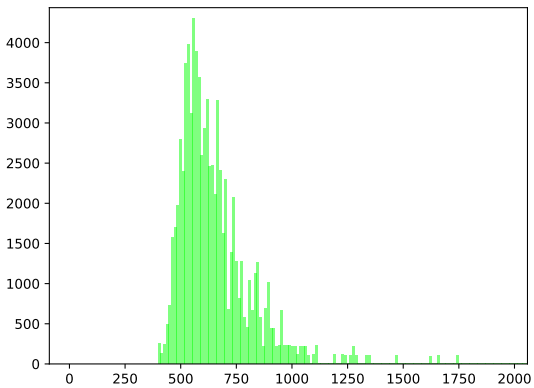

Figure 5.30: Loss distribution for IS after 1 epoch

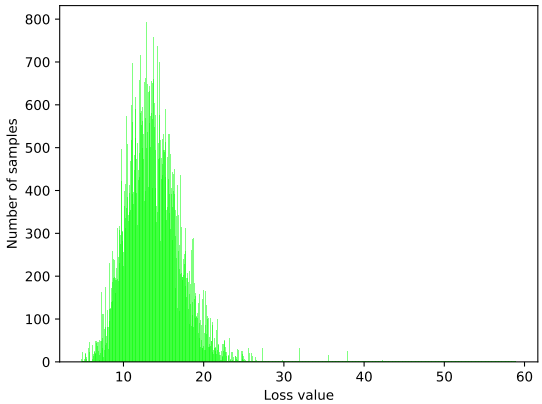

Figure 5.31: Loss distribution for IS after 40 epochs

5.2.2.2 Things that we tried that did not work

We introduced a loss exponent in the IS sampling scheme to create a higher probability of samples with higher losses getting chosen. The idea is to use (5.1) and weight the probability to be sampled by exponentially scaling their losses by an ω. Doing so increases the probability that samples with higher losses will get selected for the next batch. For details of how we did this, see Section 5.1.2.

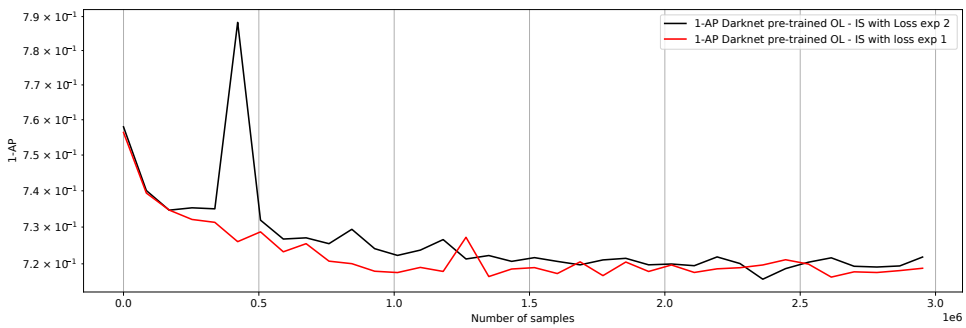

Figure 5.32: 1-AP$_{50}$ score comparision for IS methods with different loss exponents in OD w/o classification for COCO. The DarkNet backbone is pre-trained and evaluated every epoch during 40 epochs.

When checking the loss distribution after 40 epochs, the resulting distribution is wider but not enough to make a massive difference in the probabilities. See the difference between loss distribution when using loss exponent "2" 5.33 and loss exponent "1" 5.34. Again the main explanation for why IS is not creating any major difference compared to NIS is probably because the model cannot solve the OD task well. If the model cannot handle most samples correctly, a minority of informative samples that generate larger gradients than others will not exist. Thus, IS will not work theoretically.

As the loss distributions in OD do not enable us to oversample higher losses to the same extent as

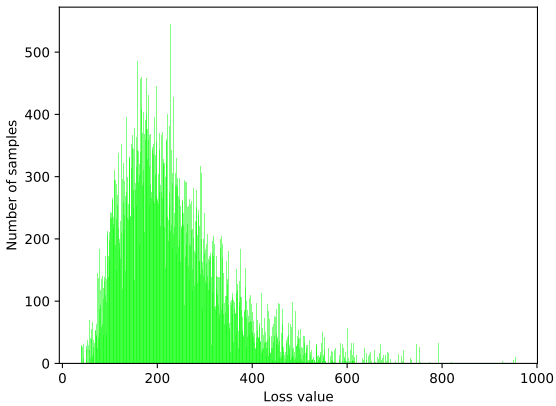

Figure 5.33: Loss distribution for IS after 40 epoch with loss exponent 2

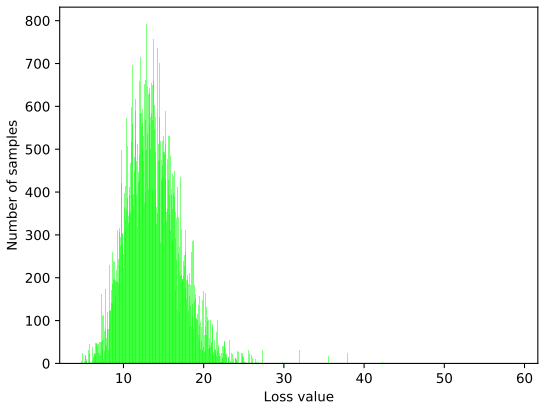

Figure 5.34: Loss distribution for IS after 40 epochs with loss exponent 1

we want, we decided to try another IS method. To do so, we used the IS method presented in Section 5.1.3 called IS-randomsubset. It selects a random subset of all samples and then selects a batch consisting of the samples with the highest losses in the subset. To a greater extent, this approach will choose samples with the highest losses independently of how the loss distribution looks; therefore, this approach is more rank-based. The results after 40 epochs can be seen in Figure 5.35. We can identify a difference between NIS and IS. However, it is hard to draw any conclusions as we need to evaluate the experiment for more runs. Nevertheless, from this result, IS seems not to outperform NIS.

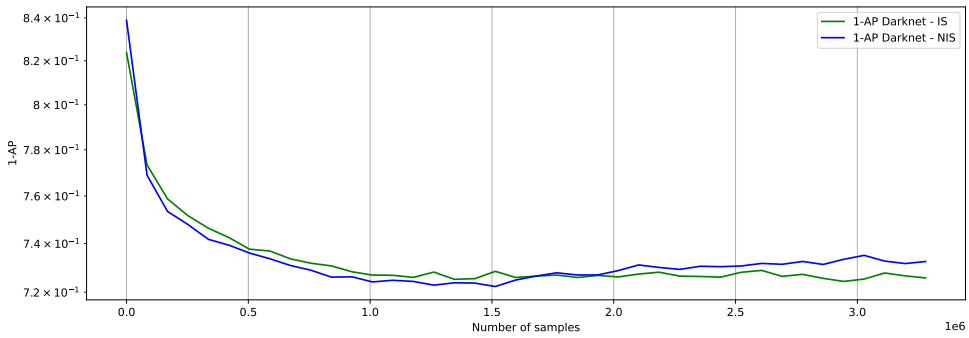

Figure 5.35: 1-AP_{50} score comparison for IS-randomsubset and NIS in OD w/o classification for COCO. IS-randomsubset selects a random subset of all samples and then selects a batch consisting of the samples with the highest losses from that subset. The DarkNet backbone is pre-trained and evaluated every epoch during 40 epochs.

Returning to the fact that the model does not solve the OD task well. Much time was spent discovering why the model was not performing better than around 28%AP_{50}. As described in Section 4, we chose to use four anchor boxes and wondered if this selection negatively affected our performance. We tried using nine anchor boxes, but the AP_{50} score did not increase, see Figure 5.36. Still, the performance was not good enough for us to provide a good comparison between IS and NIS. As the model could not solve the OD task, not even able to solve for the most straightforward cases, the result is hugely different from the IC tasks, where we could correctly handle most samples after the first or second epoch. The IC results are what initially motivated the interest for IS in OD. When it comes to OD, the samples seem to produce substantial contributions in optimizing the network for much longer. Thus, we believe IS should be interesting to try at later stages of the training. However, we have not been able to analyze this further in our thesis.

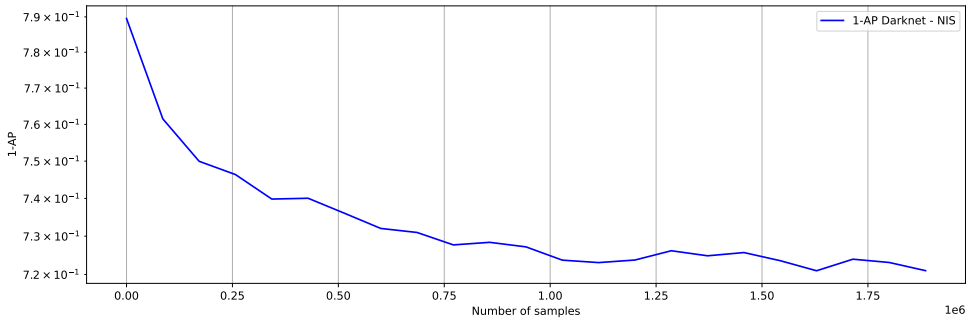

Figure 5.36: 1-AP_{50} score for NIS in OD w/o classification for COCO using 9 anchor boxes. The DarkNet backbone is pre-trained and evaluated every epoch during 35 epochs.

To summarize, we have not been able to train our model to a stage where it produces correct predictions for the bulk of the samples in the training set. Mainly because YOLOv2 was not adequate for COCO. Effectively this caused us to be unable to analyze IS's impact comprehensively. Motivated by the promising results for IS in IC, we still believe that further research on IS in OD is interesting and needed as it could garner convergence speed-ups.

Chapter 6

Conclusions

In this paper, we evaluated importance sampling for accelerating DL training. Most of the experiments were for IC problems, with a minority of the results coming from experiments on OD problems. The results from these experiments are rather conclusive and robust for IC problems but are yet to show the same confidence for OD problems. From our conclusions, we mainly propose an importance sampling method based on the history of losses of samples in the dataset but also see great promise in other potential sampling methods. It is essential to consider that all of our results are measured by the number of samples rather than the time it would take for the training to be conducted.

6.1 Image Classification

We can state that, after the initial pass over all samples, importance sampling does see a better performance than that of uniform sampling, see the result in Sections 5.1.1, 5.1.2, and 5.1.3. The excellent performance lasts a few epochs; after that, the evidence becomes inconclusive. We see performance better, worse, or at par with the performance of uniform sampling depending on the dataset and network architecture. Poor performance could be explained by the high exposure to challenging samples, resulting in the model becoming biased toward these and consequently "forgetting" the easy samples. Therefore the conclusion is that importance sampling could be helpful to speed up the training in the beginning. However, if the aim is to reach maximum performance and complete convergence, importance sampling is not always the right choice.

Furthermore, less complex models are better suited for importance sampling methods. Naturally, lower complexity translates to a need for more training. Therefore improvement of the convergence speed for less complex models is more compelling. Additionally, the results for less complex models show that importance sampling, compared to uniform sampling, does not show worse final performance to the same extent as the more complex models.

Our findings suggest that experimenting with exponentially scaling the losses for importance sampling indicates a faster performance improvement for high exponents. This only last in the first few epochs when the importance sampling is applied. After that, the lower exponents work better into convergence. There is an observable pattern of higher exponent having an accelerated learning curve in the beginning and reaching convergence earlier. The exact correlation is observable for the convergence performance as well; the higher the exponent, the worse the performance. All exponents can maintain better performance throughout the runs than uniform sampling for a more complex dataset. For a simpler dataset, uniform sampling can beat out most importance sampling runs in convergence.

When we introduced new importance sampling methods, we saw that the performance of the methods varies in large. We concluded that not sampling based on probabilities from sample losses could be more beneficial than doing so. Specifically, we saw that selecting a subset of the highest losses and then selecting a random batch from that subset achieved the best performance. This is most probably because the approach makes sure to select the most challenging samples at the beginning of the training process. Later in the training process, this approach still ensures more

accessible samples, which would counteract the fact that we create a model that becomes biased to challenging samples/classes in the long run.

6.2 Object Detection

Coming from promising results for IC tasks, the same methodology was applied to the OD tasks to generalize our results. As far as this study goes, the result does not generalize as we cannot recreate the same superiority in OD tasks. In solving the task, we have applied different exponents for loss, sampling methods, and the number of anchor boxes, to name a few of the attempts made. From what we have seen, we believe that the main issue was our choice of network architecture, as it cannot achieve performance for the standard case of using uniform sampling. Furthermore, the distribution of losses in OD does not as clearly distinguish challenging samples from the majority of the dataset. This has been a problem when we try translating the results from IC to OD has been observed

However, from the results on IC, importance sampling shows that it can be used to effectivize the computational power used during training. In the future, when even more data will be available when training neural networks for autonomous vehicles, this algorithm will be even more important to investigate further.

Chapter 7

Future Work

We have identified several aspects of importance sampling that we believe would be interesting to further investigate through our work. Some have origins in promising results in our research, and some from us not yet being able to fully take the methods to the depth they need to be for us to make conclusions.

We have seen in Section 5.1.2 that exponentially increasing or decreasing the losses associated with each sample significantly changes the performance and that high and low exponents are beneficial in different segments of network training. Therefore we believe that taking this experiment to the next step by using an exponent that varies throughout the training could be favorable. In the progress of doing this thesis, we have tried making use of a decaying exponent and also completely switching from importance to uniform sampling in the middle of training. Our attempts did not give us any conclusive evidence for or against them as an idea. However, we believe that a key to making this as effective as possible needs to revolve around figuring out the timing of the decay and making it generalizable for any model. To further discuss this topic, although we see how training neural networks could benefit from a solution like this, we also see the difficulty in generalizing it. It is unclear what a solution would look like that is not optimized for specific networks and datasets and could be used by anyone implementing importance sampling. Switching importance sampling on and off is something that our results additionally support. This task has already been solved by Katharopoulos et al. [5] as they can switch importance sampling on when it will result in an actual speed-up of the training.

Section 5.1.3 suggests that our baseline importance sampling is inferior to the newly introduced strategy of using a random subset and then selecting a batch with the highest losses from that subset. Experimenting with the size of the subset to find the optimal solution is something small that we believe could be further tested as we have shown that it generalizes for different networks and their complexity.

Continuing the discussion and research around which importance sampling method works best is a topic we believe to be compelling for the future. There was a noticeable difference between just our two methods of importance sampling. These are only a few of the possible ways of doing importance sampling. Extending experiments to even more importance sampling methods to figure out which method provides the better speed up of the training is something we find intriguing.

Perhaps the most exciting suggestion for future work relating to our thesis is to continue researching how importance sampling translates to OD tasks. The area is largely unexplored in the scientific community, and the need to speed up the network training for these tasks is vast.

Furthermore, in line with our belief that more studies on OD and importance sampling are necessary, we see a need to investigate further how IS's performance varies based on the model's complexity. Our findings are conclusive for IC but looking at if it translates to other CV areas such as OD is an interesting topic that would contribute to mapping importance sampling as a study subject. Also, looking at other network architectures is something that we do believe is important in generalizing our findings.

Bibliography

[1] Zenseact. "Zenseact: About Us". URL: https://www.zenseact.com/about-us/ (visited on 05/18/2022).

[2] B. Dickson. "Why reducing the costs of training neural networks remains a challenge". 2020. URL: https://bdtechtalks.com/2020/10/12/deep-learning-neural-network-pruning/ (visited on 05/15/2022).

[3] A. Balasubramaniam and S. Pasricha. "Object Detection in Autonomous Vehicles: Status and Open Challenges". 2022. DOI: 10.48550/ARXIV.2201.07706.

[4] F. E. Sahin. "Long-Range, High-Resolution Camera Optical Design for Assisted and Autonomous Driving". *Photonics* 6 (2019). DOI: 10.3390/photonics6020073.

[5] A. Katharopoulos and F. Fleuret. "Not All Samples Are Created Equal: Deep Learning with Importance Sampling". *CoRR* abs/1803.00942 (2018). URL: http://arxiv.org/abs/1803.00942.

[6] G. Alain et al. "Variance Reduction in SGD by Distributed Importance Sampling". 2015. DOI: 10.48550/ARXIV.1511.06481.

[7] A. Katharopoulos and F. Fleuret. "Biased Importance Sampling for Deep Neural Network Training". *CoRR* abs/1706.00043 (2017). URL: http://arxiv.org/abs/1706.00043.

[8] D. H. Ballard and C. M. Brown. "Computer Vision". New Jersey, USA: Prentice Hall, 1982. URL: http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/bandb.htm.

[9] D. Wang, J.-G. Wang, and K. Xu. "Deep Learning for Object Detection, Classification and Tracking in Industry Applications". *Sensors* 21 (2021), p. 7349. DOI: 10.3390/s21217349.

[10] O. Lorente, I. Riera, and A. Rana. "Image Classification with Classic and Deep Learning Techniques". *CoRR* abs/2105.04895 (2021). URL: https://arxiv.org/abs/2105.04895.

[11] Z.-Q. Zhao et al. "Object Detection With Deep Learning: A Review". *IEEE Transactions on Neural Networks and Learning Systems* PP (2019), pp. 1–21. DOI: 10.1109/TNNLS.2018.2876865.

[12] A. Balasubramaniam and S. Pasricha. "Object Detection in Autonomous Vehicles: Status and Open Challenges" (2022), p. 6.

[13] A. Voulodimos et al. "Deep Learning for Computer Vision: A Brief Review". *Computational Intelligence and Neuroscience* 2018 (2018), pp. 1–13. DOI: 10.1155/2018/7068349.

[14] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". 2016. DOI: 10.48550/ARXIV.1612.08242.

[15] IBM. "Neural Networks". URL: https://www.ibm.com/cloud/learn/neural-networks. (accessed: 03.06.2022).

[16] A. Dongare, R. Kharde, A. D. Kachare, et al. "Introduction to artificial neural network". *International Journal of Engineering and Innovative Technology (IJEIT)* 2 (2012), pp. 189–194.

[17] W. Hao et al. "The Role of Activation Function in CNN". 2020, pp. 429–432.

[18] H. Ghedira and M. Bernier. "The effect of some internal neural network parameters on SAR texture classification performance". 6 (2004), 3845–3848 vol.6. DOI: 10.1109/IGARSS.2004.1369962.

[19] M. Gardner and S. Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". *Atmospheric Environment* 32 (1998), pp. 2627–2636. DOI: https://doi.org/10.1016/S1352-2310(97)00447-0.

[20] S. Saha. "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way". URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (accessed: 03.06.2022).

[21] K. O'Shea and R. Nash. "An Introduction to Convolutional Neural Networks". *CoRR* abs/1511.08458 (2015). URL: http://arxiv.org/abs/1511.08458.

[22] J. Brownlee. "How Do Convolutional Layers Work in Deep Learning Neural Networks?" URL: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/. (accessed: 03.06.2022).

[23] Arc. "Convolutional Neural Network - An Introduction to Convolutional Neural Networks". URL: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05. (accessed: 06.05.2022).

[24] J. Wu. "Introduction to convolutional neural networks". *National Key Lab for Novel Software Technology. Nanjing University. China* 5 (2017), p. 495.

[25] C. Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". 2018. DOI: 10.48550/ARXIV.1811.03378.

[26] T. Wood. "What is the Softmax Function?" URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. (accessed: 06.05.2022).

[27] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *CoRR* abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167.

[28] K. Leung. "How to Easily Draw Neural Network Architecture Diagrams". URL: https://towardsdatascience.com/how-to-easily-draw-neural-network-architecture-diagrams-a6b6138ed875. (accessed: 06.05.2022).

[29] S. Ruder. "An overview of gradient descent optimization algorithms". *CoRR* abs/1609.04747 (2016). URL: http://arxiv.org/abs/1609.04747.

[30] A. Amini et al. "Spatial Uncertainty Sampling for End-to-End Control" (2018).

[31] M. Nielsen. "Neural Networks and Deep Learning". Determination Press, 2015. URL: https://books.google.se/books?id=STDBswEACAAJ.

[32] Wikipedia. "Hadamard product (matrices)". URL: https://en.wikipedia.org/wiki/Hadamard_product_(matrices). (accessed: 12.05.2022).

[33] A. Géron. "Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems". Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[34] I. Loshchilov and F. Hutter. "Online Batch Selection for Faster Training of Neural Networks". *CoRR* abs/1511.06343 (2015). URL: http://arxiv.org/abs/1511.06343.

[35] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". 2014. DOI: 10.48550/ARXIV.1412.6980.

[36] R. Gómez. "Understanding Categorical Cross-Entropy Loss". URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/ (visited on 05/24/2022).

[37] C. A. Goodfellow I. Bengio Y. "Deep Learning". MIT, 2016.

[38] D. Erhan et al. "Scalable Object Detection using Deep Neural Networks". *CoRR* abs/1312.2249 (2013). URL: http://arxiv.org/abs/1312.2249.

[39] D. Yoo et al. "AttentionNet: Aggregating Weak Directions for Accurate Object Detection". *CoRR* abs/1506.07704 (2015). URL: http://arxiv.org/abs/1506.07704.

[40] M. Najibi, M. Rastegari, and L. S. Davis. "G-CNN: an Iterative Grid Based Object Detector". *CoRR* abs/1512.07729 (2015). URL: http://arxiv.org/abs/1512.07729.

[41] J. Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". *CoRR* abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640.

[42] W. Liu et al. "SSD: Single Shot MultiBox Detector". *CoRR* abs/1512.02325 (2015). URL: http://arxiv.org/abs/1512.02325.

[43] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". *CoRR* abs/1506.01497 (2015). URL: http://arxiv.org/abs/1506.01497.

[44] J. Redmon and A. Farhadi. "YOLOv3: An Incremental Improvement". *CoRR* abs/1804.02767 (2018). URL: http://arxiv.org/abs/1804.02767.

[45] J. W. Pharr M. and H. G. "Physically Based Rendering: From Theory To Implementation". 2018.

[46] D. Siegmund. "Importance Sampling in the Monte Carlo Study of Sequential Tests". *The Annals of Statistics* 4 (1976), pp. 673–684. URL: http://www.jstor.org/stable/2958179 (visited on 04/25/2022).

[47] S. T. Tokdar and R. E. Kass. "Importance sampling: a review". *WIREs Computational Statistics* 2 (2010), pp. 54–60. DOI: https://doi.org/10.1002/wics.56.

[48] M.-S. Oh and J. O. Berger. "Adaptive importance sampling in monte carlo integration". *Journal of Statistical Computation and Simulation* 41 (1992), pp. 143–168. DOI: 10.1080/00949659208810398.

[49] W. W. LaMorte. "Central Limit Theorem". 2016. URL: https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_probability/BS704_Probability12.html (visited on 04/27/2022).

[50] A. E.C. "Monte Carlo Methods and Importance Sampling". 1999. URL: https://ib.berkeley.edu/labs/slatkin/eriq/classes/guest_lect/mc_lecture_notes.pdf.

[51] G. Hinton, S. Osindero, and Y.-W. Teh. "A Fast Learning Algorithm for Deep Belief Nets". *Neural computation* 18 (2006), pp. 1527–54. DOI: 10.1162/neco.2006.18.7.1527.

[52] L. Wang et al. "Accelerating Deep Neural Network Training with Inconsistent Stochastic Gradient Descent". 2016. DOI: 10.48550/ARXIV.1603.05544.

[53] Y. Liu. "The Confusing Metrics of AP and mAP for Object Detection / Instance Segmentation". URL: https://yanfengliux.medium.com/the-confusing-metrics-of-ap-and-map-for-object-detection-3113ba0386ef. (accessed: 28.05.2022).

[54] T. Fawcett. "An introduction to ROC analysis". *Pattern Recognition Letters* 27 (2006). ROC Analysis in Pattern Recognition, pp. 861–874. DOI: https://doi.org/10.1016/j.patrec.2005.10.010.

[55] S. Narkhede. "Understanding AUC - ROC Curve". URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. (accessed: 05.05.2022).

[56] J. Hui. "mAP (mean Average Precision) for Object Detection". URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. (accessed: 05.05.2022).

[57] R. J. Tan. "Breaking Down Mean Average Precision (mAP)". URL: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52. (accessed: 05.05.2022).

[58] S. Narkhede. "Understanding Confusion Matrix". URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. (accessed: 05.05.2022).

[59] Y. Bengio et al. "Curriculum learning". *Journal of the American Podiatry Association* 60 (2009), p. 6. DOI: 10.1145/1553374.1553380.

[60] T. Schaul et al. "Prioritized Experience Replay". *CoRR* (2015). URL: https://arxiv.org/pdf/1511.05952.

[61] T. B. Johnson and C. Guestrin. "Training Deep Models Faster with Robust, Approximate Importance Sampling". 31 (2018). Ed. by S. Bengio et al. URL: https://proceedings.neurips.cc/paper/2018/file/967990de5b3eac7b87d49a13c6834978-Paper.pdf.

[62] Wikipedia. "Q-learning". 2022. URL: https://en.wikipedia.org/wiki/Q-learning (visited on 05/10/2022).

[63] G. Yao, T. Lei, and J. Zhong. "A review of Convolutional-Neural-Network-based action recognition". *Pattern Recognition Letters* 118 (2019). Cooperative and Social Robots: Understanding Human Activities and Intentions, pp. 14–22. DOI: https://doi.org/10.1016/j.patrec.2018.05.018.

[64] J. Stallkamp et al. "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition". *Neural Networks* (2012), pp. -. DOI: 10.1016/j.neunet.2012.02.016.

[65] "Mapillary Traffic Sign Dataset". https://www.mapillary.com/dataset/trafficsign. Accessed: 2022-04-25.

[66] Wikipedia. "Residual neural network". 2022. URL: https://en.wikipedia.org/wiki/Residual_neural_network (visited on 05/11/2022).

[67] K. He et al. "Deep Residual Learning for Image Recognition". 2015. DOI: 10.48550/ARXIV.1512.03385.

[68] T.-Y. Lin et al. "Microsoft COCO: Common Objects in Context". 2014. DOI: 10.48550/ARXIV.1405.0312.

[69] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". *International Journal of Computer Vision* 111 (2015), pp. 98–136.

[70] A. Geiger et al. "KITTI". URL: http://www.cvlibs.net/datasets/kitti/ (visited on 05/16/2022).

[71] "NuScenes". URL: https://www.nuscenes.org/ (visited on 05/16/2022).

[72] Wikipedia. "Elbow method (clustering)". URL: https://en.wikipedia.org/wiki/Elbow_method_(clustering) (visited on 05/20/2022).

[73] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". URL: https://pjreddie.com/darknet/yolov2/. (Accessed: 2022-05-17).

[74] "TensorFlow basics". https://www.tensorflow.org/guide/basics. Accessed: 2022-04-25.

[75] "Introduction to gradients and automatic differentiation". https://www.tensorflow.org/guide/autodiff. Accessed: 2022-04-25.

[76] D. Rengasamy et al. "Deep Learning With Dynamically Weighted Loss Function for Sensor-based Prognostics and Health Management". *Sensors* (2020). DOI: 10.3390/s20030723.

[77] T.-Y. Lin et al. "Focal Loss for Dense Object Detection". *CoRR* abs/1708.02002 (2017). URL: http://arxiv.org/abs/1708.02002.

[78] F. Yu, D. Wang, and T. Darrell. "Deep Layer Aggregation". *CoRR* abs/1707.06484 (2017). URL: http://arxiv.org/abs/1707.06484.

Appendix A

Image	Network model	Dataset	No. parameters	Loss exponents
5.19	Mapillary	Resnet	2.1×10^5	[0.1, 0.5, 1.0, 1.5, 2.0]
7.1	Mapillary	Resnet	3.4×10^5	[0.1, 0.5, 1.0, 1.5, 2.0]
7.2	Mapillary	Resnet	5.2×10^5	[0.1, 0.5, 1.0, 1.5, 2.0]
5.20	Mapillary	Crapnet	2.7×10^6	[0.1, 0.5, 1.0, 1.5, 2.0]
7.3	Mapillary	Crapnet	6.3×10^6	[0.1, 0.5, 1.0, 1.5, 2.0]
7.4	Mapillary	Crapnet	10.6×10^6	[0.1, 0.5, 1.0, 1.5, 2.0]
5.21	GTSRB	Resnet	2.1×10^5	[0.1, 0.5, 1.0, 2.0]
7.5	GTSRB	Resnet	3.4×10^5	[0.1, 0.5, 1.0]
7.6	GTSRB	Resnet	5.2×10^5	[0.1, 0.5, 1.0]
5.22	GTSRB	Crapnet	2.7×10^6	[0.1, 0.5, 1.0, 2.0]
7.7	GTSRB	Crapnet	6.3×10^6	[0.1, 0.5, 1.0]
7.8	GTSRB	Crapnet	10.6×10^6	[0.1, 0.5, 1.0]

Table 7.1: The characteristics for every run from Figures 5.19 untill 7.8.

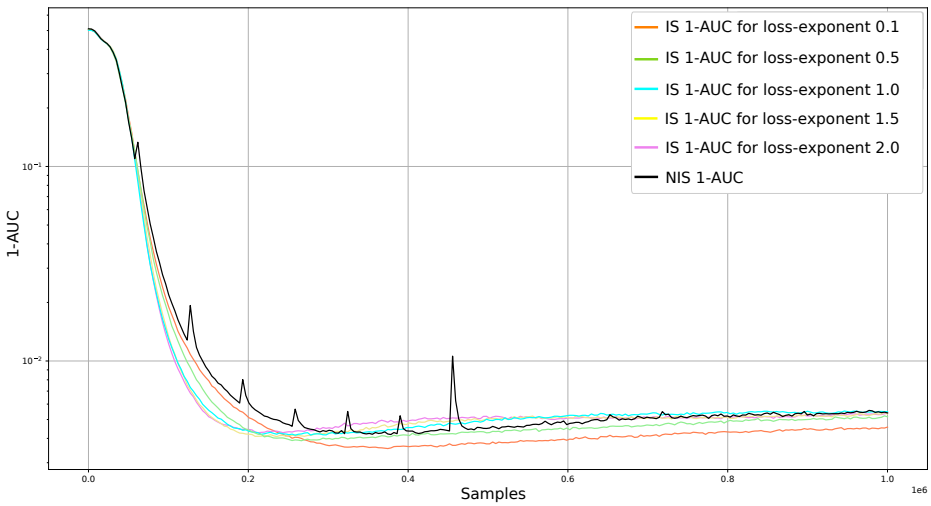

Figure 7.1: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for ResNet with 6.3×10^6 trainable parameters on the Mapillary dataset.

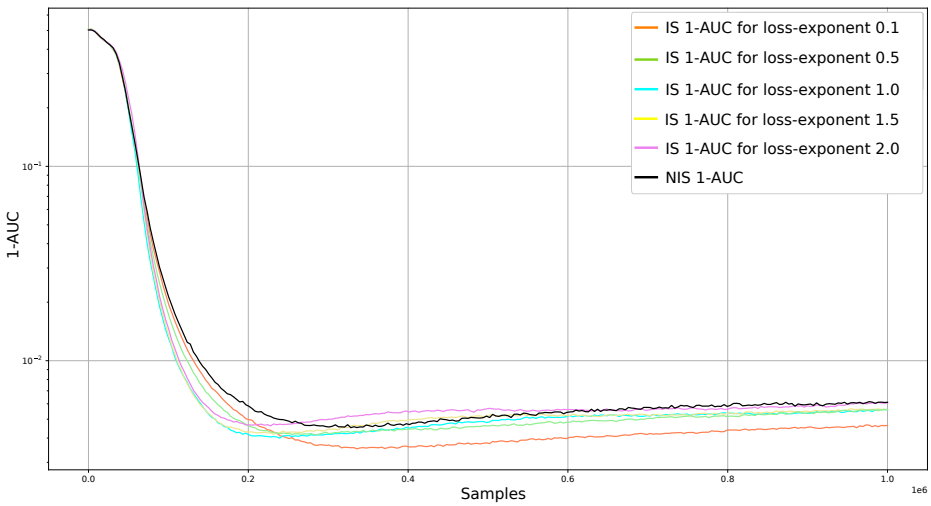

Figure 7.2: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for ResNet with 10.6×10^6 trainable parameters on the Mapillary dataset.

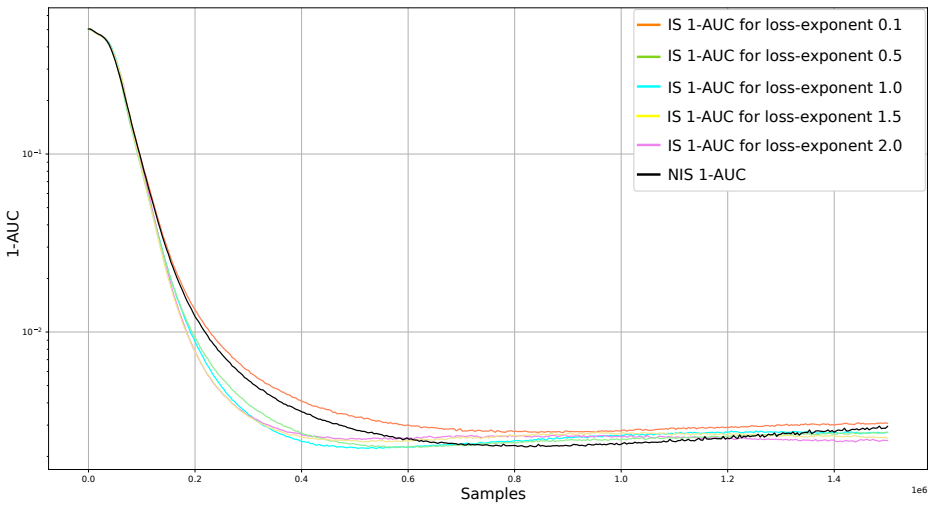

Figure 7.3: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for CrapNet with 3.4×10^5 trainable parameters on the Mapillary dataset.

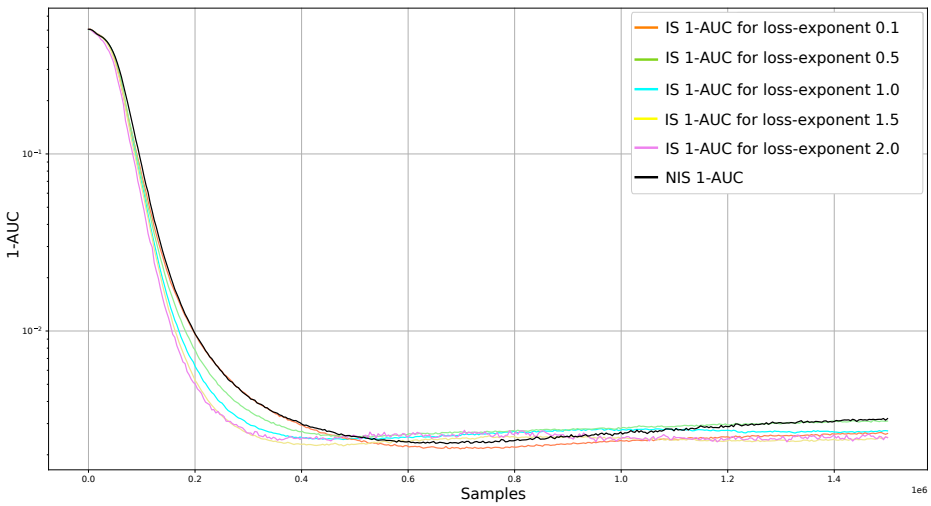

Figure 7.4: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for CrapNet with 5.2×10^5 trainable parameters on the Mapillary dataset.

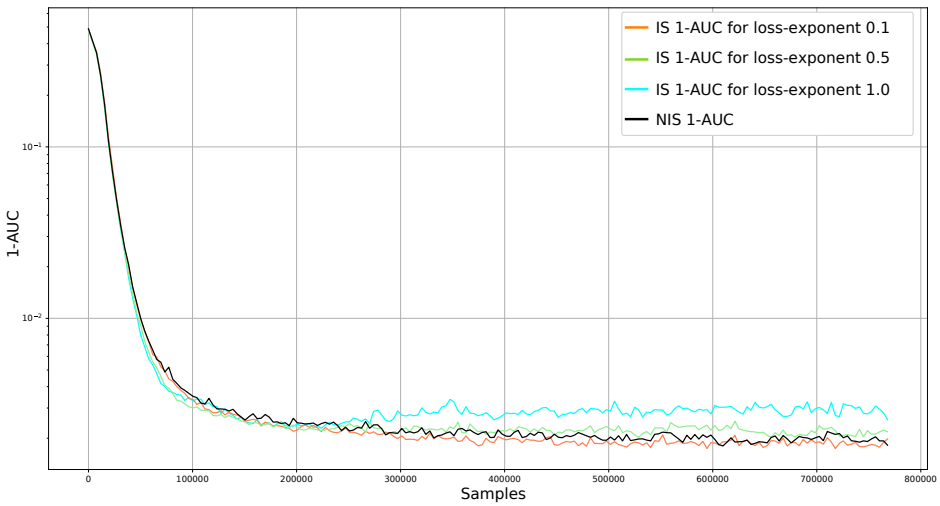

Figure 7.5: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for ResNet with 6.3×10^6 trainable parameters on the GTSRB dataset.

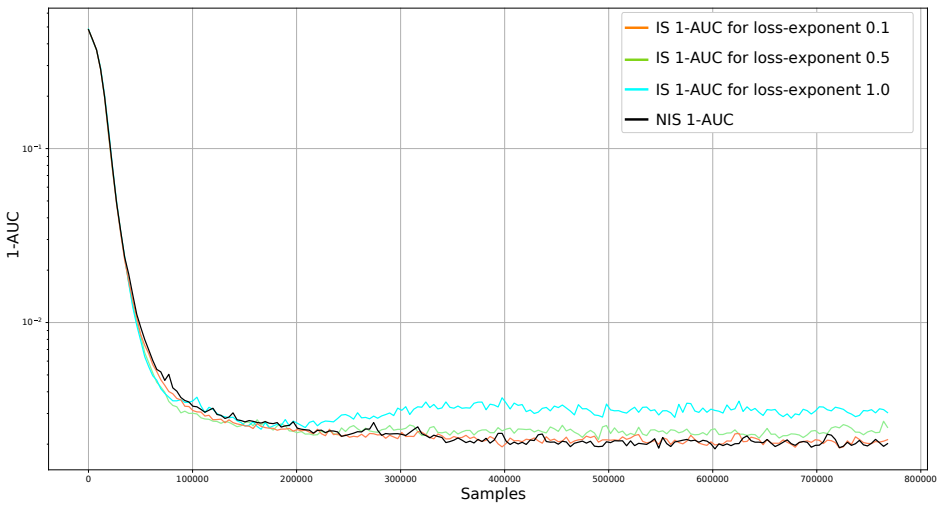

Figure 7.6: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for ResNet with 10.6×10^6 trainable parameters on the GTSRB dataset.

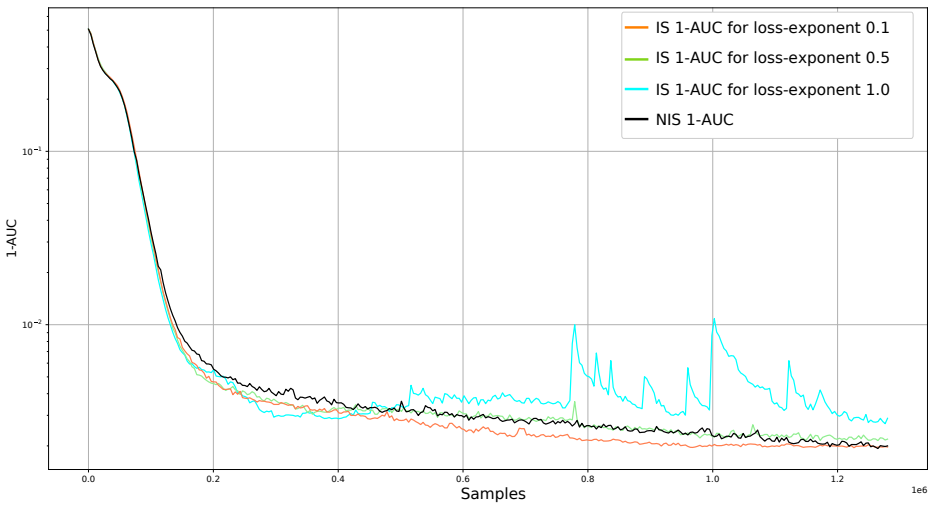

Figure 7.7: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for CrapNet with 3.4×10^5 trainable parameters on the GTSRB dataset.

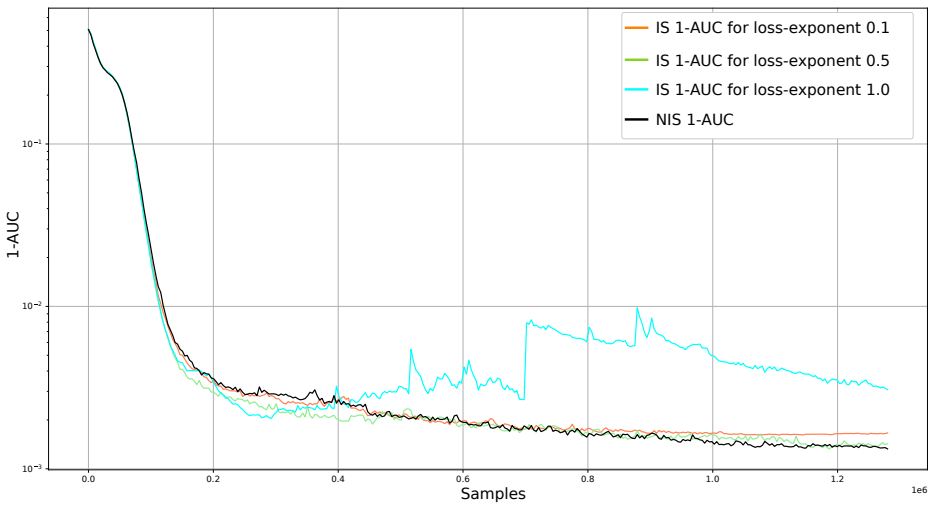

Figure 7.8: Comparison of 1-AUC over the number of samples between IS with loss exponents 0.1, 0.5, 1.0, 1.5, and 2.0 and NIS. The runs are for CrapNet with 5.2×10^5 trainable parameters on the GTSRB dataset.

Appendix B

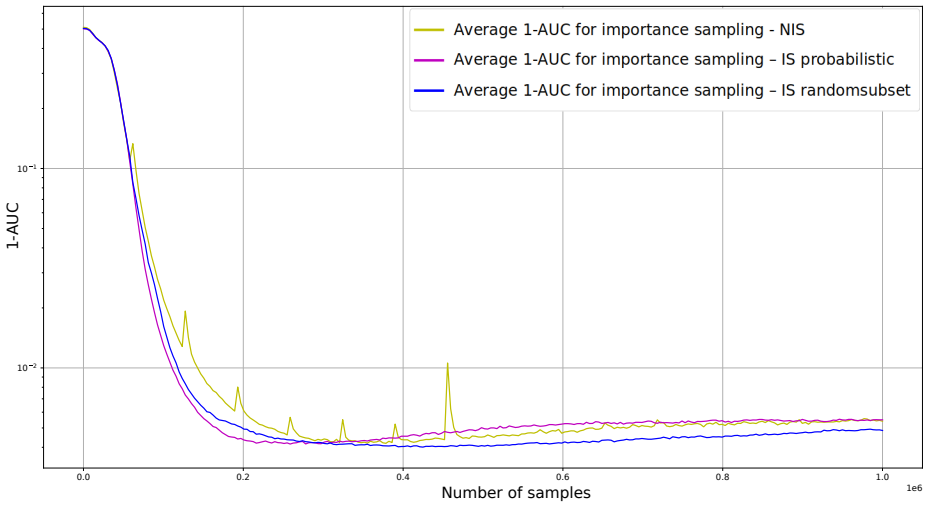

Figure 7.9: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 6.3×10^6 trainable parameters on the Mapillary dataset.

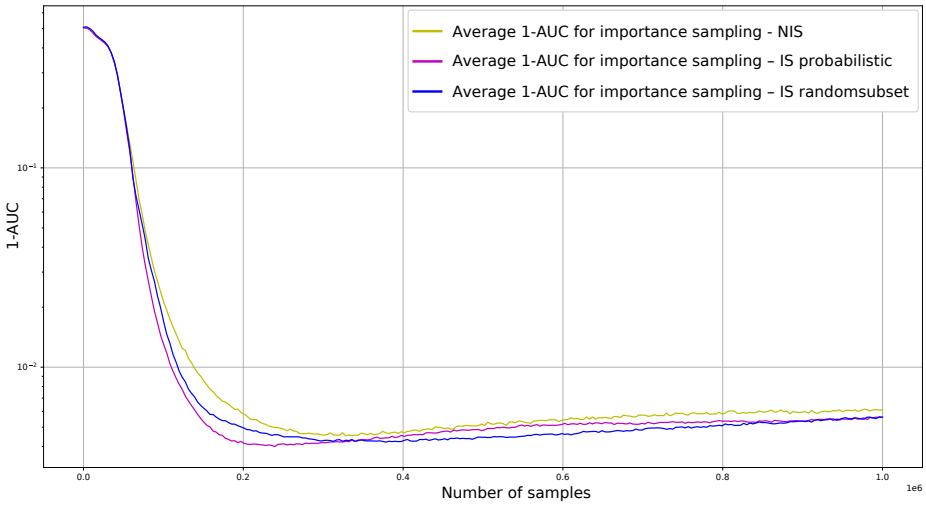

Figure 7.10: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 10.6×10^6 trainable parameters on the Mapillary dataset.

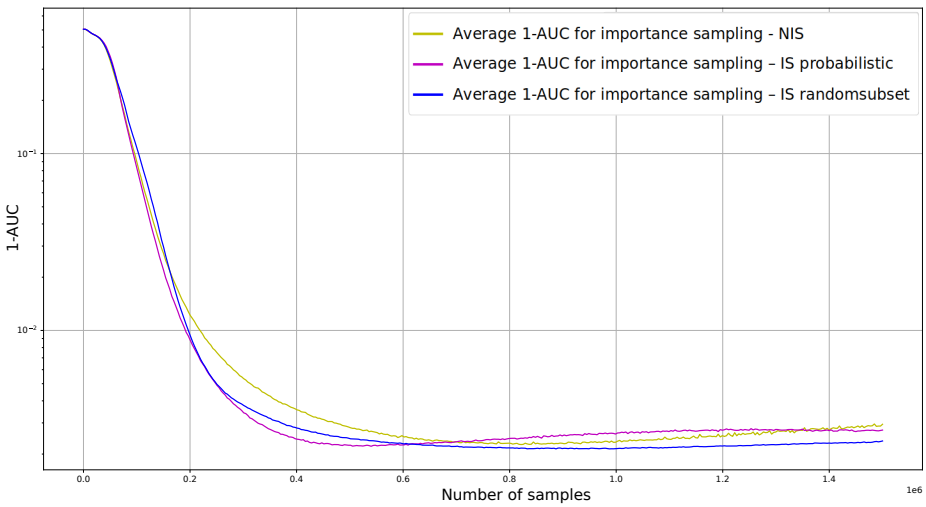

Figure 7.11: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 3.4×10^5 trainable parameters on the Mapillary dataset.

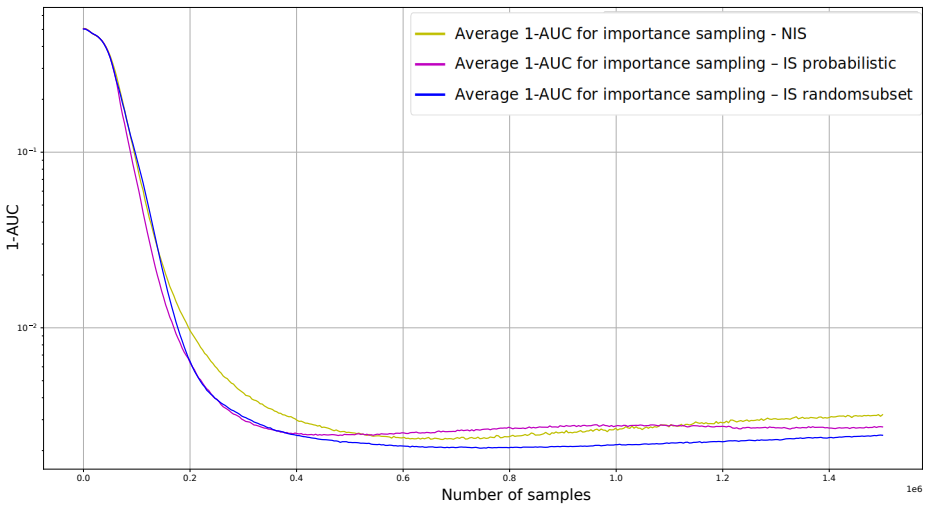

Figure 7.12: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 5.2×10^5 trainable parameters on the Mapillary dataset.

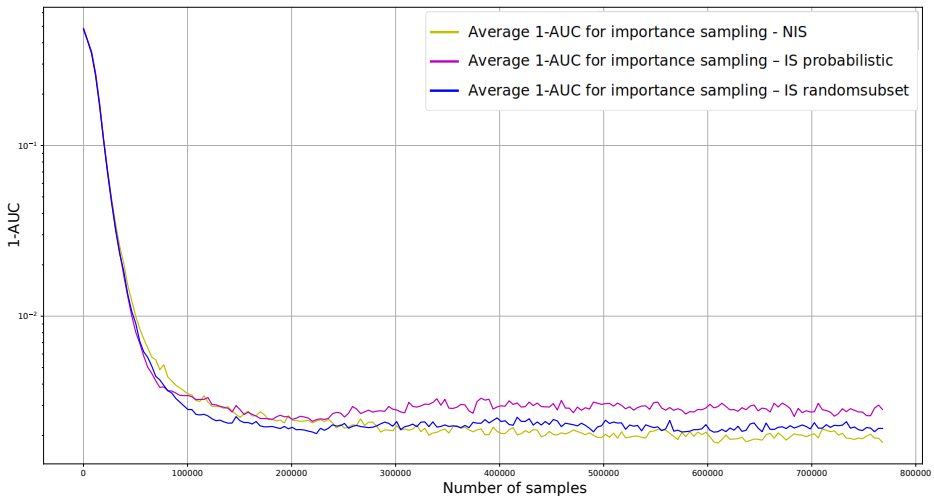

Figure 7.13: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 6.3×10^6 trainable parameters on the GTSRB dataset.

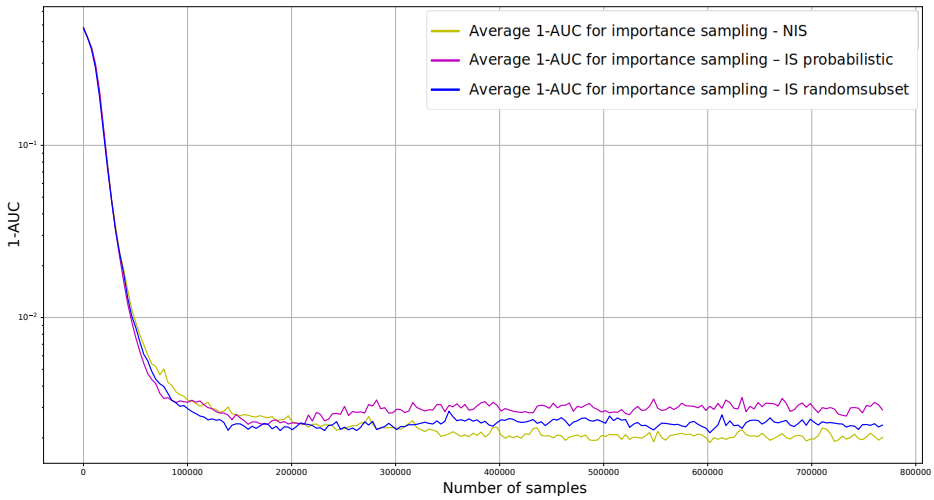

Figure 7.14: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for ResNet with 10.6×10^6 trainable parameters on the GTSRB dataset.

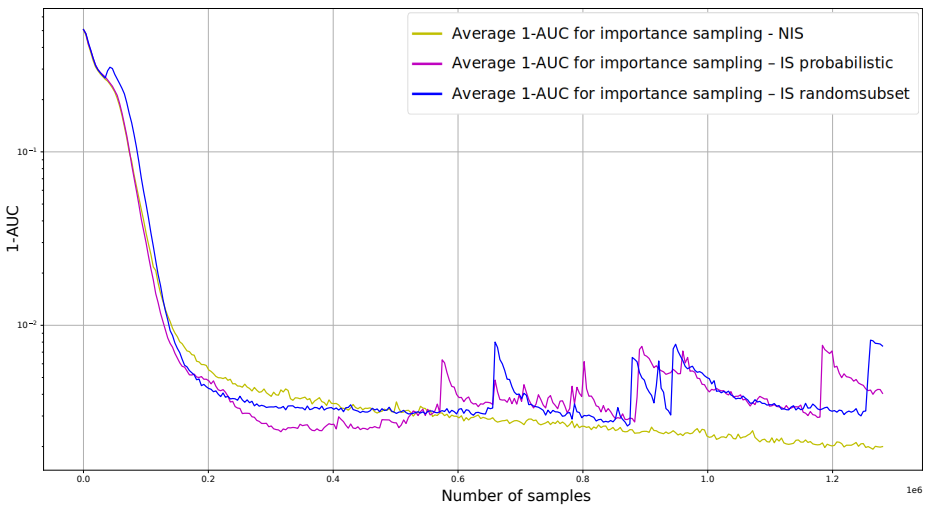

Figure 7.15: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 3.4×10^5 trainable parameters on the GTSRB dataset.

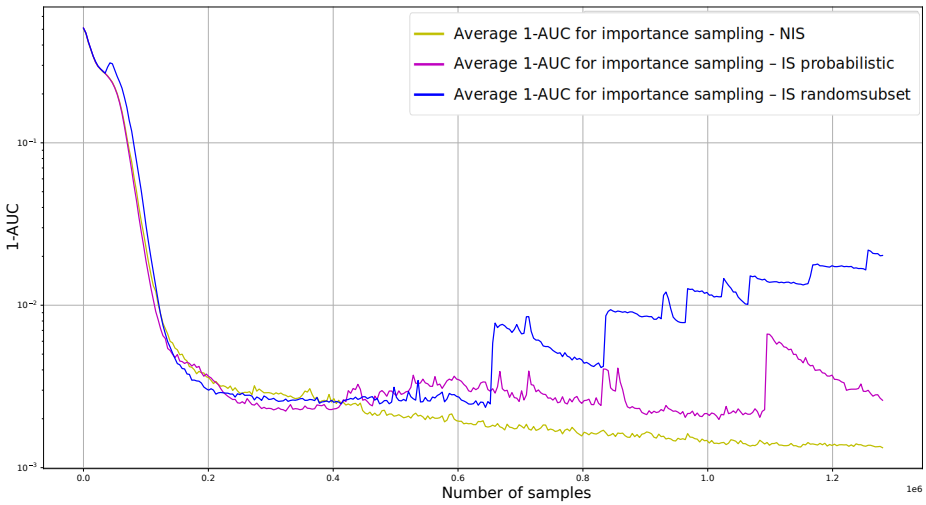

Figure 7.16: Comparison of 1-AUC over the number of samples between IS-probabilistic, IS-randomsubset and NIS. The runs are for CrapNet with 5.2×10^5 trainable parameters on the GTSRB dataset.