AALBORG UNIVERSITY
STUDENT REPORT

Vision, Graphics and Interactive Systems,
Spring Semester 2017

Semi-supervised Semantic Segmentation using Generative Adversarial Networks

Master's Thesis

Troels Høg Peter Jensen

June 8, 2017

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø
http://www.sict.aau.dk

Title:
Semi-supervised Semantic Segmentation using Generative Adversarial Networks

Theme:
Master thesis

Project Period:
P10, Spring Semester 2017

Project Group:

Participants:
Troels Høg Peter Jensen

Supervisors:
Kamal Nasrollahi

Copies: 0

Pages: 77

Date of Completion:
June 8, 2017

Abstract:

The work presented in this report, have been in relation to robot perception. The goal have been to develop a system that made it possible for a computer to get an understanding of our world. Within computer vision this could be by learning to detect and classify objects, it could also be semantic segmentation where objects in the world should be segmented and classified. The traditional way to solve there issues is by supervised learning, e.g. using Convolutional Neural Networks. Instead it was tried to develop a system, which is able to automatically learn a representation of features or object categories. The chosen method was to utilize a generative model, specifically a Generative Adversarial Method for semi-supervised learning, following the famous Richard Feynman quote: *What I cannot create, I do not understand.*. The system was therefore encouraged to generate data, that should look like masks in a semantic segmentation task. Unfortunately the system never approached something satisfactory in that regard. Furthermore the system was developed with a Region Proposal Network, that would feed the model with regions that it might find interesting.

Preface

The work presented in this report is written as the Master's Thesis for the programme "Vision, Graphics and Interactive Systems' at Aalborg University. The project have been carried out in the period February to June 2017 as a collaboration with Kube Data ApS. The author would like to thank the employees at Kube Data ApS for both providing office space, equipment, in the form of computational power and for guidance and moral support throughout the project period. Lastly, the author would like to thank Kamal Nasrollahi for supervision in regards to the project.

All code is implemented using Python 3.6.1, OpenCV 3.2 for image processing and Keras using Tensorflow as backend for the Deep Learning systems.

Aalborg University, June 8, 2017

Troels Høg Peter Jensen
thpj12@student.aau.dk

Abbreviations

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRF	Conditional Random Fields
FCN	Fully Convolutional Network
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HOD	Histogram of Oriented depths
LSTM	Long Short-Term Memorized
MLP	Multi-Layer Perceptron
MRF	Markov Random Field
NMS	Non-Maximum Suppression
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Units
ResNet	Residual Network
RNN	Recurrent Neural Network
ROI	Regions-Of-Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent

SVM	Support Vector Machine

Contents

Chapter 1

Introduction

A survey from September 2016 by International Federation of Robotics, predicts that between 2017 to 2019 industrial robots will see an average growth of at least 13 % per year, resulting in more than 1.4 million industrial robots being installed [1]. Furthermore they released a survey regarding service robots, which they project will reach almost 42 million in 2019, showing that there in the coming years is a big market and interest in robots for both industrial and service use [2].

According to the ISO 8373 standard from 2012, which specifies the vocabulary and definitions related to robotics, a robot is defined as:

"*Actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks.*"

Furthermore a more precise classification for industrial and service robots is provided, mostly this is characterized by their intended purpose. A service robot is defined as performing useful tasks for humans or equipment and include robots in areas like cleaning, medical assistance, toys and public relation [24].

Industrial robots used for e.g. automation often perform repetitive tasks in a clean environment and is therefore often not equipped with sensors to understand the world around it. Since service robot is often placed in more noisy environments, e.g. a public relation robot placed in a store, they need to understand what is located around it and how to interact to it.

Understanding the world is a complex problem though, it contains thousands of category objects, which can both look very different within the same category and visually similar within different categories. Conditions also have a strong impact on the task, e.g. lighting, distance, angle, pose or occlusion all have an affect on how an object appear.

1.1 Initial Problem Statement

The focus of the project, is related to enabling a robot to autonomously interact with its surroundings. In order to make this possible, a vision based system that can localize and categorize objects is needed.

Tousch et al. [47], Fan et al. [11] both define the world as being hierarchal and arguing that it can be used to improve image annotation and segmentation, having a prior knowledge of objects. A car is more likely to be on a road, while a sofa is more likely to be in a living room.

Furthermore the world can be divided into different building blocks, such as objects, stuff and planes. In this case objects are things like a TV, mug or car, stuff is grass, mountains and buildings, while planes is floors, walls or the sky.

The focus should be on understanding objects in the world and not planes and stuff.

Knowing the location and category of an object is not enough to understand your surroundings, being able to differentiate between objects of the same class is likewise a necessity, another focus should therefore be to differentiate between same class object.

The initial problem statement, which will be the basis for the preliminary analysis, is thereby expressed as:

How can a system be developed, that utilizes computer vision techniques to allow robots to understand objects within their environments.

Part I

Preliminary Analysis

Chapter 2

Computer Vision

When discussing computer vision, there are a set of approaches worth discussing, these are: classification, object localization, semantic and instance segmentation.

- **Classification** consist of assigning a global label to an image.
- **Localization** aims at localizing an object within an image.
- **Object detection** aims at detecting if a specific object is present in the given image.
- **Semantic segmentation** consist of creating a pixel-wise classification of an image, meaning each pixel should be assigned to a class.
- **Instance segmentation** is a one level increase in difficulty compared to semantic segmentation, its goal is to be both class and instance aware.

A visual explanation of the tasks mentioned, is seen in **figure 2.1**.

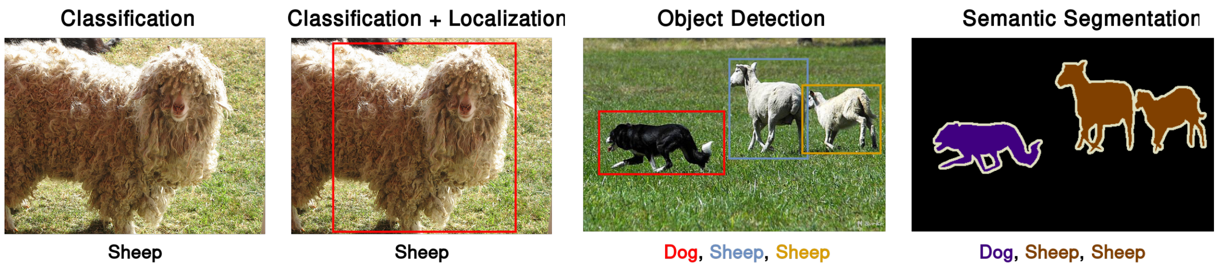

Figure 2.1: *Comparison between four tasks within computer vision, classification and localization is a single object task, while object detection and semantic segmentation is a multi object task. Instance segmentation create a pixel-wise classification like semantic, but also differs between instances within the same class.*

The focus for this project is within segmentation, for which two metrics is normally used for showing the accuracy of the system. These are class accuracy, which is the classification accuracy as described above and pixel accuracy, which is the average accuracy of the pixel-wise classification.

Chapter 3

Related Work

In the coming sections, methods used for both semantic and instance segmentation will be looked into, the reason for this is that since they are closely related, it is relevant to research both fields.

The papers described are chosen based on how recent they are, novelty of their methods and the datasets their proposed methods are tested on.

3.1 Semantic Segmentation

This section will look into different methods that have previously been used to perform semantic segmentation. First methods that creates segmentations without region proposal is described, followed by a description of the methods that do. At the end a comparison of the results the mentioned methods have obtained is conducted.

3.1.1 Proposal free approach

Couprie et al. [8] Uses RGB-D images, the images is rescaled to $240 \times 320px$ and is first preprocessed by normalization, the depth image is treated as an additional channel. The RGB-D image is given as input to a multi-scale convolutional Neural Network (CNN), meaning there are multiple copies of the same CNN which are fed different scales of the image.

The CNN consist of three stages, the first two contain a convolutional layer, a non-linear activation followed by a max pooling layer while the last stage only consist of convolutional layers. For classification two fully-connected layers are used, in the second layer the multi-scaled features are concatenated. Superpixels are used to smooth the predictions during post-processing.

Long et al. [35] introduces fully convolutional networks (FCN), which discard the classification layer and substitute the normal fully-connected layers in a CNN with convolutional layers, this allow for end-to-end training and make the network indifferent to the size of the input image. Since CNNs include subsampling they use deconvolution to upsample the feature maps to the size of the input image. The architecture consist of convolutional layers followed by pooling layers, to get a finer a finer pixel prediction, they add skip layers between layers which they fuse together. This results in three networks FCN-32, FCN-16 and FCN-8, named after the upsampling that is needed.

Zheng et al. [52] uses Conditional Random Fields (CRF) for the pixel-wise labeling, using an image a CRF models the pixel labels as random variables for a Markov Random Field (MRF). They formulate the mean-field algorithm of CRF as a stack of CNN layers, by stacking multiple mean-field iterations, it becomes equivalent to treating the inference as a Recurrent Neural Network (RNN). The idea is that the FCN stage predicts pixel labels without considering the structure of the image, while the CRF-RNN stage makes structured predictions. The architecture of the FCN stage is the FCN-8 introduced in Long et al. [35].

In Lin et al. [33] they utilize a CNN for feature extraction, specifically they resize the input image to three scales and feed them to three architectural similar CNNs and upsample them to have the same spatial size before fusing them to a final feature map. By looking at rectangular regions of the whole image and combining CRF nodes that lie within a spatial range of each other, they create a CRF graph. A FCN is then fed the feature vector from the corresponding location of each CRF node to get the unary potential. Likewise they create edge features by concatenating corresponding feature vectors of two connected nodes, these are fed to another FCN to get the pairwise potential. Combining these results in a low resolution prediction, to address this they first apply three mean field iterations, to get a coarse prediction followed by bilinear upsampling and a dense CRF for post-processing, which sharpens the object boundaries and generates the final high-resolution prediction.

Schulz and Behnke [43] uses RGB-D images to create patches that are inversely proportional to the depth. These patches are upsampled using bilinear interpolation and fed to a shallow feed forward CNN architecture. In all they use eight input maps: The raw RGB image, four maps containing a simplified Histogram of Oriented depths (HOD) and one for the height. The best results are achieved using CRF for post-processing.

In Chen et al. [7] they use a FCN version of a Residual Network (ResNet) with 101 layers, where they use atrous convolutions throughout the network. Atrous convolutions add strides within the kernel, adding zeros in between the filter values. Resulting in an increase of the receptive field, which means that the degree of downsampling of the feature maps is reduced. They construct four parallel networks using atrous convolutions with different sampling rates (stride in the kernel) and fuse them together at the end. Furthermore they utilize multi-scale image representation, using three scales of the input and bilinear interpolation for fusing the feature maps. For preprocessing the feature maps are fed to a fully-connected CRF, compared to dense CRF which tries to smooth the prediction, fully-connected CRFs tries to improve detailed local structures of the image.

Lin et al. [34] creates an architecture they call RefineNet, to utilize multi-scale image representation the system contain several separate RefineNets, that can take an arbitrary amount of inputs. Normally the smaller scale output from the previous ResNet and a current scale image. The inputs for each ResNet is fed to two consecutive simplified ResNet nodes, since the inputs have different scale they upsample the feature maps and fuse them by summation. The feature map is then given to a chained residual pooling network, which goal is to capture background context, specifically it is built as a chain of multiple blocks consisting of a pooling layer with stride 1 and a convolutional layer. The blocks takes the output from the previous as input blocks and consecutively sum the outputs. A last ResNet node is added before outputting the prediction. Lastly they feed the high-resolution feature maps to a dense softmax layers to form a dense score map, bilinear interpolation is then used to match the scale of the original image.

In Huang et al. [23] they also utilize a RNN, they built the system on top of a VGG-16 network, followed by a Gated Recurrent Unit (GRU) with explicit long range contextual dependency (RNN-ELC). The idea is that since an image is processed from left to right, top to bottom, two pixels placed vertical of each other is highly contextually related, but processed at vastly different time steps and would therefore not be seen as related using a normal RNN. Specifically they have four parallel branches, each responsible for calculating the dependencies in the four directions away from a spatial position in the feature map. These are then concatenated before being fed to two blocks containing an unpooling layer followed by convolutional layer for the final prediction and in order to upsample to the size of the original image.

Li et al. [31] also uses RNN, but the Long Short-Term Memorized (LSTM) method and processing RGB-D images. They process the RGB and depth map separately, encoding the depth map as an HHA image and feeding it into three convolutional layers, followed by a vertical context memory network layer. The RGB network utilizes the same architecture as in Chen et al. [7], before also being fed to a vertical context memory network layer, which is then fused with the depth context using a horizontal bi-directional memory network to get RGB-D context. The output is then concatenated with feature maps extracted from the RGB image, since it provides richer information than the depth images, it is then fed to a last convolutional layers with softmax activation to perform the pixel-wise scene labeling.

3.1.2 Proposal-based approach

Gupta et al. [18] proposes to use RGB-D images, they first create a contour map, taking advantage of the depth information, utilizing contour detection they generate region proposals. For each region proposal a HHA map is made and is separately fed to a modified a Region-based Convolutional Neural Network (R-CNN). The same is done for the RGB regional proposal, they are both given as input to a Support Vector Machine (SVM) that classifies the detected object. In order to perform pixel-wise labeling they utilize superpixels.

Hong et al. [22] proposes to decouple the classification and segmentation, which they argue allows them to use pretrained models for the classification and only train the segmentation network. They adopt the VGG-16 CNN for classification. Between the two networks is a bridging layers, which takes the output from the last pooling layers and constructs class specific activation maps which is the given as input to the segmentation network. They adopt a deconvolution network as was first introduced in Noh et al. [37] for this step, which is a contrast equivalent network to their classification network, utilizing unpooling and deconvolution layers resulting in a prediction map with the same scale as the input image. Softmax is used for the pixel-wise classification.

in Caesar et al. [6] they use selective search to get free-form region proposals. Multiple convolutional layers are given the input image and the bounding box region of the feature maps and free-form region is given as inputs to multiple fully connected layers working in parallel, followed by a classification and softmax layer for pixel-wise classification.

3.1.3 Summary

Table 3.1 shows the results the mentioned methods have obtained.

Table 3.1: *Comparison of semantic segmentation papers.*

		PASCAL VOC 2011 Test set	**PASCAL VOC 2012 Test set**	**Pascal Context**	**NYUDv2**			**SIFT Flow**			**SUN-RGBD (37 classes)**		
	Data type	Mean IU	Mean IU	Mean IU	Pixel acc.	Class acc.	mIU	Pixel acc.	Class acc.	mIU	Pixel acc.	Class acc.	mIU
Proposal Free													
[8] (2013)	RGB-D				64.5	63.5							
[35] (2015)	RGB (+HHA for NYUD)	62.7	62.2	37.6	65.4	46.1	34.0	85.2	51.7	39.5			
[52] (2015)	RGB	72.4	72.0	39.3									
[33] (2015)	RGB		75.3	43.3	70.0	53.6	40.6	88.1	53.4	44.9			
[43] (2015)	RGB-D				73.4	73.7							
[7] (2016) (CRF-ResNet-101)	RGB		79.7	45.7									
[34] (2016) (ResNet-152)	RGB		83.4	47.3	73.6	58.9	46.5				80.6	58.5	45.9
[23] (2016)	RGB				64.5	41.4		87.8	46.7				
[31] (2016)	RGB + HHA						49.4						48.1
With Proposal													
[18] (2014)	RGB +HHA				60.3	28.6	31.3						
[22] (2015)	RGB		66.6										
[37] (2016)	RGB		72.5										
[10] (2016)	RGB		75.2										
[5] (2016)	RGB			32.5				84.3	64.0				

As can be seen, for semantic segmentation the proposal-based methods does not achieve better results than the proposal free methods, though It can also be seen that incorporating depth data in general improves the results, which fits with the tests conducted by the respective authors.

Having looked at previous work for semantic segmentation, the next section will focus on methods used for instance segmentation.

3.2 Instance Segmentation

In Liang et al. [32] proposes to use two networks, one for semantic classification and use that for fine-tuning the instance segmentation network. For semantic segmentation they utilize the architecture introduced in Chen et al. [7] and uses a fine-tuned VGG-16 CNN for classification. A fully-connected CRF is used for pre-processing in order to smooth the segmentation prediction.

For instance segmentation they use the predicted category locations to predict the specific object locations and create a bounding box. The coordinate map from the bounding box is then used as input to the network. Furthermore they utilize multi-scale prediction, for the larger layers they subsample them before being concatenated, the idea is that they get more accurate pixel-wise prediction by incorporating the spatial coordinate maps. A 1×1 convolutional layer is then used for the final pixel-wise prediction.

Dai et al. [9] proposes to use a cascade of multitasking networks, where one network is responsible for generating bounding boxes, one generates mask instances for each bounding box and the last one categorizes these. Each of these stages uses the VGG-16 architecture and functions as a second input to the next stage, with the first input being a feature map extracted from the input image.

In Wu et al. [49] proposes to use a semantic segmentation network to calculate a score map and use bounding-box regression as a localization network. The transform maps are then applied to the score maps and search for and keep the local maximum of detected instances. A mask is then generated for each instance and using non-maximum suppression, which is responsible of thinning and making the edges of the mask sharper, a final instance segmentation is generated.

The semantic segmentation network uses the architecture of a fully convolutional resnet and are trained separately of the localization network. Furthermore, when a layer subsamples the feature map, they add a skip layer that uses atrous convolution instead.

Uhrig et al. [48] proposes to extend the FCN-8 architecture with three output channels; semantic segmentation, estimated instance depth and instance directions respectively. They use template matching on the direction prediction to extract the instance centers, adjust the templates aspect ratio depending on the semantic class prediction and further scale it based on the predicted depth class. Using non-maximum suppression within the template matchings, they get a representation of temporary instance centers, from which they determine the instances location. To get the final instance segmentation they look at the information from each channel. If there is any incomplete instance proposals they see if it is biased to a certain direction, if there is any neighboring candidates with the same semantic class and depth in that direction, they are fused into one instance. They further merge their instance predictions with the pixel-wise semantic labeling channel of the FCN.

In Pinheiro et al. [38] use a FCN version of the VGG-A architecture, they modify it by adding a top-down network that fuses the corresponding earlier feature map from the bottom-up network and upscales them using a refinement module. The network is trained to generate object masks and they argue that the architecture results in a more pixel-accurate segmentation and is better at separating objects. The network is trained in two stages, first the feed-forward network is trained to generate coarse pixel-wise segmentation and afterwards refinement modules are trained. When converged they combine the networks and train them together.

Arnab and Torr [3] first uses a semantic segmentation network combined with higher order CRF, which is a modified CRF trained for object proposal potential Arnab et al. [4]. Combining the semantic segmentation with the bounding box output from a object detector they identify instances of which they use CRF on to smooth the pixel-wise segmentation.

Zhang et al. [51] creates regions by extracting three different sizes of patches in a densely overlapping fashion. From these patches they softly score a pixel-wise instance prediction. They then utilize a densely connected Markov Random Field (MRF) in order merge the soft predictions. In order to get the soft prediction they fine-tune a FCN, based on VGG-16 and trained on ImageNet, for instance segmentation. The FCN predicts up to five instances, plus the image background, while the MRF is restricted to a maximum of nine instances per image. For the MRF colour is not used as a feature, as they argue two instances can have the same color or it can be deceiving due to shadows, saturation and specularities.

In Li et al. [30] they propose a system that is able to be trained end-to-end, they utilize Faster R-CNN in order to generate region proposals. For each region they use a sliding window approach, for which they use a single pretrained ResNet FCN to calculate two likelihoods for the pixel belonging to some object instance inside or outside of the object boundary. Jointly using the two score maps, softmax is used to predict the pixel-wise foreground probability, while a max operation produces the pixel-wise classification probability. In order to classify the whole proposed region, average pooling is used on all of the pixels likelihoods. For inference the system proposes 300 Regions-Of-Interest (ROI) which are then passed through a bounding box regression in order to refine the prediction. Using Non-Maximum Suppression (NMS) the ROIs with a large amount of overlapping are filtered out.

Ren and Zemel [40] also proposes a network that can be trained end-to-end, but using a recurrent network with an attention model. They divide the network into four components, external memory that keeps track of the state of segmented objects, a region proposal network, a segmentation network and a scoring network, that verifies if an object have been found.

The external memory is used to save information about the object boundaries from all previous steps, they argue that this helps the network reason about occluded objects and where to look next.

The region proposal network consist of Long Short-Term Memory (LSTM) network, in order to be sure to get enough information in the bounding box, the network is allowed to look at different locations for each time step. A linear layer is given the output of the LSTM's hidden state in order to predict the exact box coordinates.

In the segmentation network a pretrained FCN, as described in Long et al. [35], is used to pre-process the input, while a deconvolution network as Noh et al. [37] is used to predict the pixel-wise segmentation. Furthermore they calculate the relative angle towards the instance center, in order to force the model to encode more detailed information about the boundaries.

The scoring network is used to determine the amount of objects in the image, specifically it takes information from both the region proposal and segmentation network to produce the score. If the score gets below a predefined threshold, it is assumed that all objects have been segmented and the process terminates.

3.2.1 Summary

In **table 3.2** a comparison between the methods described in this section is shown.

Table 3.2: *Comparison of instance segmentation papers.*

		PASCAL VOC 2012 Validation set	**PASCAL Context**	**KITTI (test set)**		**City Scape (all)**	**MSCOCO (test-dev-set)**	
	Data Type	AP^r	Mean IU	MWCov	MUCov	AP	mAP	AUC
[32] (2015) (VGG-16)	RGB	58.7						
[9] (2015) (ResNet-101)	RGB	63.5					44.3	
[49] (2016)	RGB	60.9	44.5					
[48] (2016)	RGB			79.7	75.8	8.9		
[38] (2016) (MSCOCO validation set only)	RGB							20.9
[3] (2016)	RGB	58.3						
[51] (2016)	RGB			73.7 (74.1 with post-processing)	54.3 (55.2 with post-processing)			
[30] (2016) (ResNet-101)	RGB						49.5 (59.9 with extra processing)	
[40] (2016)	RGB			80.0	66.9	9.5		

One thing to notice, is that all the methods use region proposals, showing that though a CNN can be good at localizing objects within its frame, it does not know how to actually distinguish between them. Another thing to consider, is that all of the methods handle the problem, both semantic and instance segmentation, in the same manner, using supervised learning to get the system to segment the image and classify its objects within a finite number of classes.

Chapter 4

Problem Statement

As mentioned in the initial problem statement, the basis for the project is to see if a computer vision system can be developed, that allows a robot to understand their environment.

One way to accomplish this, could be generative models, which learns about our world, by generating data like it. Based on the initial problem statement and preliminary analysis, the final problem statement is expressed as:

Can a system be developed, which unsupervised can learn to perform semantic segmentation, by encouraging it to generate similar data.

Part II

Proposed Solution

Chapter 5

System Overview

In this chapter the overall system architecture will be introduced. For each subsystem a short description of its purpose will be given, which will be ground work for the detailed explanations in the following chapters. The system can be seen in **figure 5.1**.

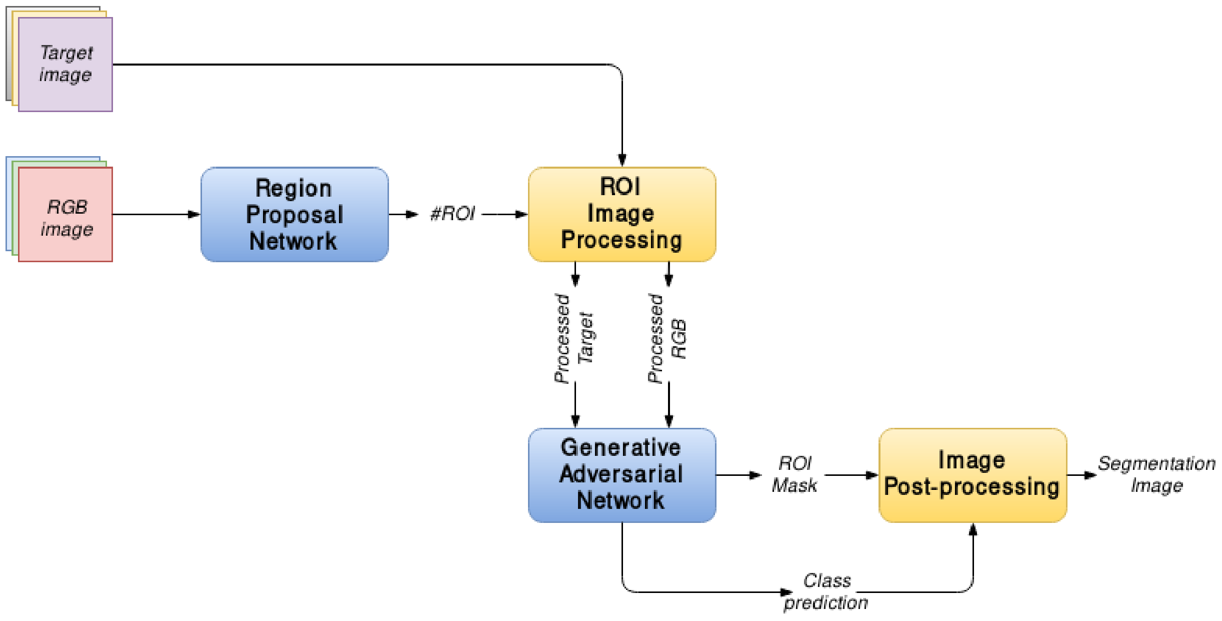

Figure 5.1: *Overview of the full system architecture.*

The first subsystem is a Region Proposal Network (RPN), based on a Convolutional Neural Network (CNN), it will be fed with an RGB image of a scene. The networks purpose is to generate a finite number of Regions Of Interest (ROI), Since the RPN is not responsible for classifying the objects within the scene, these proposals will be class agnostic and only look for objects, e.g. ignoring walls, floors, grass and the sky. Further details about the system is available in **chapter 7**.

The second block in the system, is used for preprocessing the ROIs before feeding them to the next subsystem. First of all, it makes sure that each ROI have the proper size, since it is undesirable to warp the images, the system will make crop the ROIs into smaller ROIs if too large or pad them if too small. It is also responsible for storing the location of each ROI, in the context of the original scene image. Furthermore it makes sure that the target and RGB images are correctly aligned, a detailed explanation is available in **section 7.4**.

Each ROI is given as input to the Generative Adversarial Network, which have two main purposes. First of all it is responsible for creating the object mask for the ROI and secondly it classifies the object, shown in the ROI. Further details regarding theory and architecture, is available in **chapter 8**.

The last block in the system is in regard to post-processing of the image, the main purpose is to stitch the predicted mask ROIs, such that the complete full resolution masked scene is restored. The specific details about the system is available in **chapter 9**.

For each ROI proposal output by the RPN, the three other subsystems will process the data, in order to complete the whole scene image.

As seen, two of the subsystems are based on deep learning techniques, the upcoming chapter will look further into the fundamental methods and reasoning behind such systems.

Chapter 6

Deep Learning Based Systems

As explained in **chapter 5**, the proposed system consist of two subsystem, namely a Region Proposal Network and a Generative Adversarial Network, these are based on deep learning technologies. In this chapter the reasoning behind deep learning will be explained, specifically in regards to the technique Convolutional Neural Network (CNN), for which the most important elements will get a detailed explanation.

Deep learning is a subset of techniques within machine learning, it mainly differs in how data is processed. Using traditional machine learning techniques, e.g. within computer vision, involves hand-crafting the features that will be used for classifying the data. This is needed, since these techniques traditionally have issues processing raw data, but it can also be difficult and most importantly, time consuming. Moreover, a feature extraction method that performs well on one task, seldom performs well on another comparable task, e.g. recognizing different objects in an image [29] .

In tasks, that we as humans perform intuitively, such as recognizing speech or faces, can be hard to design features for, as we ourself might have issues describing what we see or hear. Deep learning removes the necessity for hand-crafting features, by learning and improving from experience, e.g. seeing data and trying to learn concepts about what have been presented to it. This means that, depending on the complexity of the task at hand, large amounts of data is needed, the availability of readily made large datasets are also one of the reasons behind deep learning gaining popularity the later years. In the case of supervised deep learning, a rule of thumb is that training on 5.000 examples per class will result in acceptable results, while 10 million examples per class will exceed human performance [17].

Deep learning is referenced as such, since it tries to create a hierarchy of feature concepts, learning data representations in multiple layers by changing its internal parameters. In the first layers it will register edges, corners and contours, later it will register whole parts of an object, resulting in high-level features. The corresponding complexity of these networks, puts a larger demand on the computational hardware, increased availability of especially Graphical Processing Units (GPU) can therefore be seen as a key reason for making deep learning possible [17]. GPUs are designed to compute instructions in parallel by having more cores, which are clocked at a lower frequency than the cores in a Central Processing Unit (CPU) though, in general they also have a higher memory bandwidth. This also means that their efficiency is correlated with the amount of data they are being asked to work with. Previous experiments have shown that in the case of deep learning, GPUs are 10-30$\times$ faster, compared to CPUs [44].

In later years, deep learning have become a synonym with Artificial Intelligence (AI) and have proven its effectiveness in several fields of research and at various tasks, e.g. speech, Natural Language Processing (NLP), robotics and computer vision, the latter is the focus for this project [29].

6.1 Convolutional Neural Network

As mentioned above, when working with computer vision, the most commonly used technique is CNN, which is also the basis for this project. A challenge within computer vision is related to size and location of objects within an image, where traditional machine learning have issues. CNNs are inspired by research in cat's visual cortex, which showed that their receptive field consisted of a interconnected system of layers, that covered the entire visual field, CNNs have shown to overcome this problem by taking advantage of the strong spatially local correlation in images, this is done by using weight sharing in each feature map [28]. This also results in the number of free weights being reduced significantly, compared to e.g. a comparable Multi-Layer Perceptron (MLP) where all neurons are connected. Furthermore, because a CNN consist of multiple layers of non-linear mappings, conventional classifiers are linear, it is able to become sensitive to minute details and insensitive to large irrelevant details, such as position, pose, size and lighting [29].

6.1.1 Convolutional Layer

The convolutional layer is the core of a CNN, it performs the mathematical operation of convolution across the width and height of its input and a dot product along the depth, resulting in a stack of 2-dimensional feature maps. These feature maps represent 'activated' regions from the input layer, meaning a region at a spatial position where features specific to the used kernel activates. The depth of the output is a hyperparameter, which is equivalent to the amount of kernels that the network should use on the input. The network learns which features are important by updating the kernels values for each iteration through the training data.

Unlike traditional neural networks where every output neuron interacts with every input neuron, CNNs use local connectivity, this is accomplished by using a kernel that is smaller than the input, this ensures that the spatial structure of an image is taken into consideration. Meaning that each neuron in the feature map uses the same kernel in the spatial dimensions, while the neurons along the depth axis only receives input from a small local part of the image. A visual explanation of this can be seen in **figure 6.1**, the neurons in the feature maps look at a small region of the image.

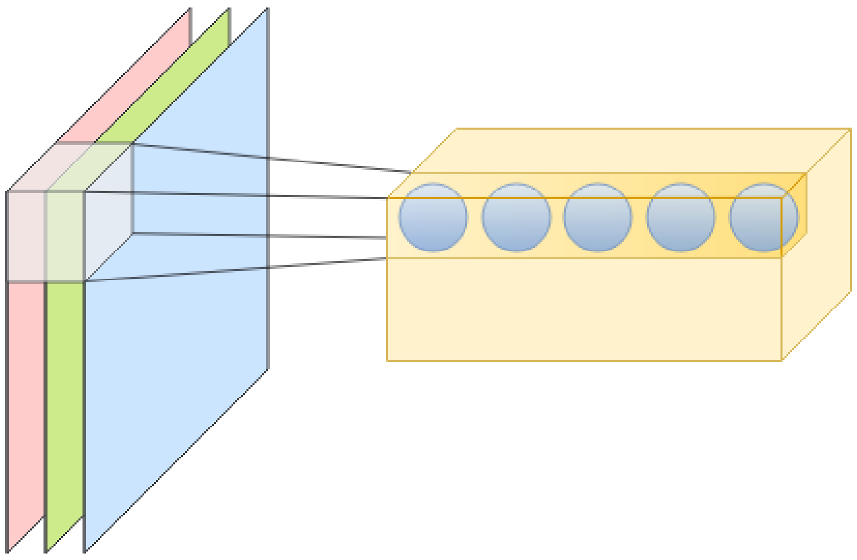

Figure 6.1: *Principle behind the convolutional layer, to the right is an RGB input image, with a kernel moving along its height and width. The corresponding output is a volume of feature maps.*

Another feature of the convolutional layer is parameter sharing, it is based on the assumption that if a feature is useful at one spatial position, it is most probably useful at another spatial position. This means that a feature map will share the same parameters along its width and height, an illustration can be seen in **figure 6.2** [17].

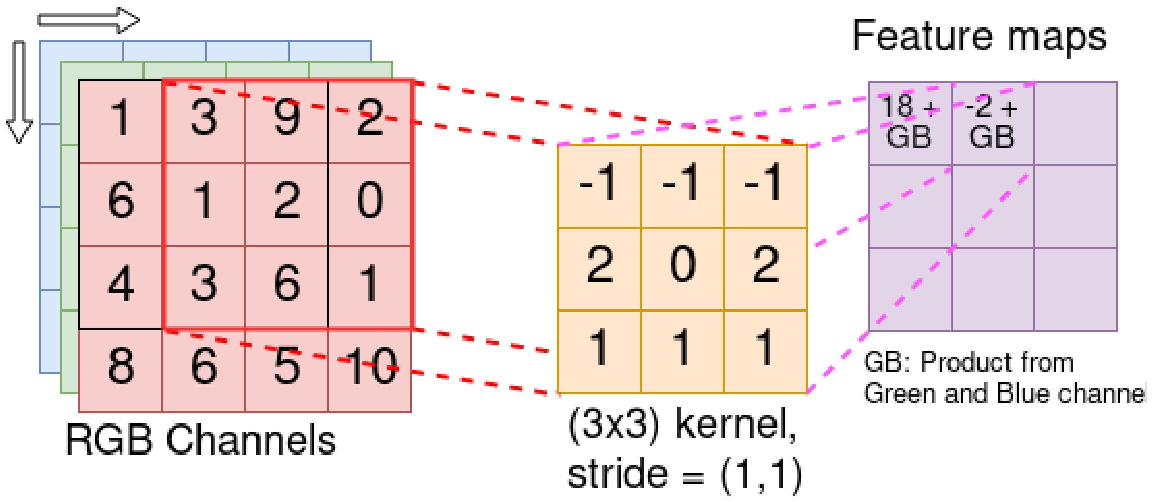

Figure 6.2: *Illustration of parameter sharing, the same kernel is moved along the width and height of the red channel. Each channel will have its own kernel and the resulting feature map is a dot product between them..*

As the figure shows, the kernel is moved along the width and height of the input image, computing the dot product between the values in the patch and the ones in the kernel. The process is repeated for each colour channel, which each have its own kernel and the final feature map value for the spatial position is the sum of their contributions. Taking normalization into account, each channel contribution can be divided by the sum of kernel values.

The bias of the kernels are normally initialized as a constant, 0, while the weights are initialized randomly. Different functions exist to initialize the weights, e.g. Gaussian, HeNormal and GlorotNormal and their uniform variations.

Gaussian initialization gets its weights from a Gaussian distribution with $\mathcal{N}(\mu, \sigma^2)$, where μ is the mean, typically 0, and σ^2 the variance.

HeNormal initialization also samples its weights from a Gaussian distribution, with mean 0, but with standard deviation $\sqrt{\frac{2}{fan_{in}}}$, where fan_{in} is the number of input neurons.

GlorotNormal again samples its weights from a Gaussian distribution, with mean 0 and a standard deviation $\sqrt{\frac{2}{fan_{in}+fan_{out}}}$, where fan_{out} is the number of output neurons.

6.1.2 Pooling Layer

The main purpose of a pooling layer is to merge similar features into one, in order to make the network robust to relative positions in an image. this is done by downsampling the spatial dimensions while keeping the depth of the feature maps. Including pooling layers in the network have the added benefit of making the system less prone to overfitting [29] [17].

One implementation of a pooling layer is max pooling, which works by running a specified kernel with a predefined stride through the feature maps. The output for each kernel operation is the highest value within the neighborhood.

Average pooling is another option, unlike max pooling its output is the average of the values in the kernel neighborhood, the output is rounded to nearest integer. An example showing the results of the two, given the same image values, can be seen in **figure 6.3**, a feature map of size 4×4 and a kernel with size (2×2) and stride (2,2) is used. For the rest of the report, it will always be assumed that stride is the same in both the x and y direction.

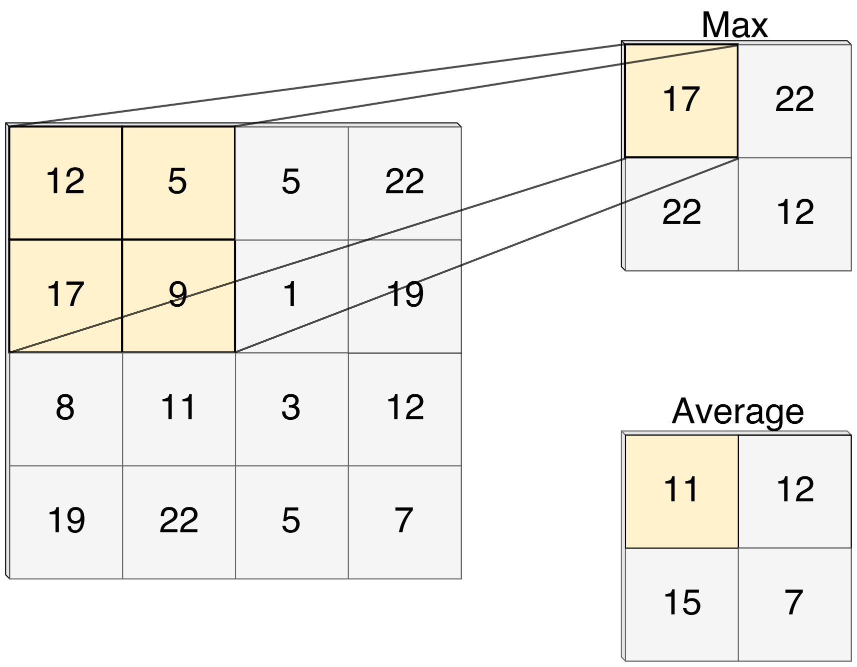

Figure 6.3: *Example showing the difference between max and average pooling, when a 2×2 kernel with a stride of 2 is*

6.2 Activation Layer

The output of e.g. a convolutional layer is a weighted sum of linear activations, each of these is then sent through a non-linear function. Some of the more common activation functions are Rectified Linear Units (ReLU), TanH and Sigmoid, a graphical comparison between the three can be seen in **figure 6.4**.

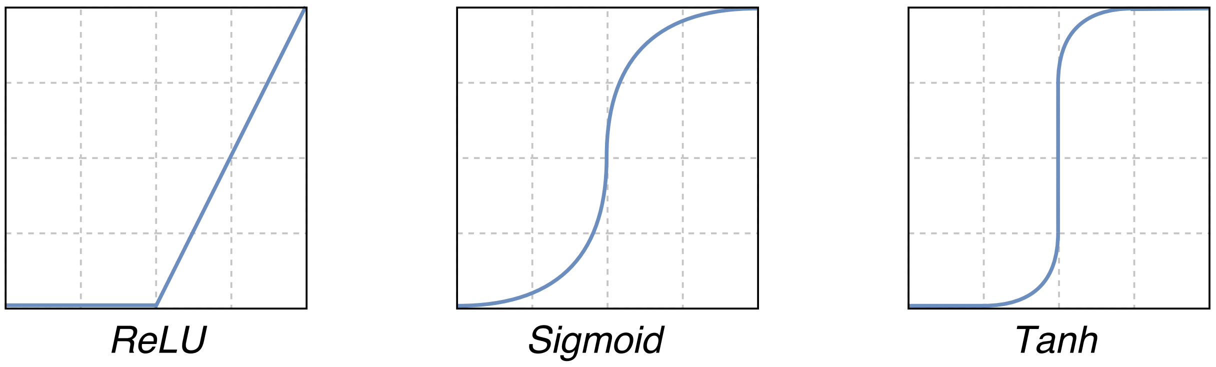

Figure 6.4: *Graphical presentation of the three activation functions; ReLU, Sigmoid and Tanh.*

ReLU is defined by the function $f(x) = max(0,x)$, meaning that it thresholds the gradient in the range $[0,\infty]$, the drawback of this is that if a neuron gets clamped to 0, then the gradient will be zero, resulting in a "dead neuron". At the same time it can be prone to exploding gradient as x increases, which also causes the neurons to not be updated.

Sigmoid is defined as $f(x) = \frac{1}{(1+e^(-x)}$, it tries to compress the activation between [0,1]. Because of this Sigmoid tends to suffer from vanishing gradient.

The tanH activation fuction is defined as $f(x) = tanh(x) = 2\sigma(2x) - 1$, it can be seen as a scaled Sigmoid in the range $[-1,1]$.

ReLU have shown to be the most popular activation function, because of its simple nature it is computationally cheap, while it have also proven to learn faster than the others when used in a CNN [29] [15].

6.2.1 Fully-Connected and Decision Layer

In order to map the high-level spatial representations of the CNN, into a lower dimensional global representation, a common technique is to use one or several concurrent fully connected layers at the end, though recent research have shown that fully-convolutional networks performs well for pixel-wise tasks [35]. A fully connected layer is a Multi-Layer Perceptron (MLP), meaning all neurons in a layer have full connection to the neurons in the previous layer. In e.g. a classification task, the fully-connected layer will have the same output dimensions as the number of available classes and holds the unnormalized log probabilities for each class. The last layer is the decision layer and consist of an activation function, for a classification problem the softmax function is typically used.

Softmax is a linear classifier, it calculates a normalized probability of the output from the decision layer. Softmax can be expressed as:

$$P(y = j|x) = \frac{\exp^{x^T w_j}}{\sum_{k=1}^{K} \exp^{x^T w_k}} \tag{6.1}$$

$P(y = j|x)$ is the predicted probability for a specific class, given data x. w_j is the weights for that class and K is the number of classes.

6.2.2 Loss Layer

The loss layers purpose is to calculate how the system should be penalized, based on the predicted distribution and true probability from the forward pass and ground truth.

For a binary classification objective, cross entropy normally follows a softmax activation and can be expressed as:

$$E = -\sum_i p_i \cdot log\ q_i \tag{6.2}$$

Which can be seen as a measure of similarity or error between the true class p_i and the predicted class q_i. Staying with the classification example, the goal for the network is to minimize the cross entropy between the true and predicted distribution. If the objective was multi classification, e.g. with three possible classes, the cross-entropy is computed as:

$$E = -\sum_{i=1}^{3}(p_i \cdot log(q_i)) + ((1 - p_i) \cdot log((1 - q_i)) \tag{6.3}$$

Having looked at how to calculate the output error of the forward pass, the next step is to discuss how to use it to improve the network.

6.2.3 Back-propagation

The way that the network learns and improves, is by updating the weights and biases throughout the network, such that the output comes closer to the ground truth, this is done by using a backward pass. Intuitively explained, back-propagation moves backwards through the network, computing the gradient for the weights in each layer based on the output loss. The principle behind back-propagation, using a MLP with three hidden layers and two outputs as an example, is shown in **figure 6.5**.

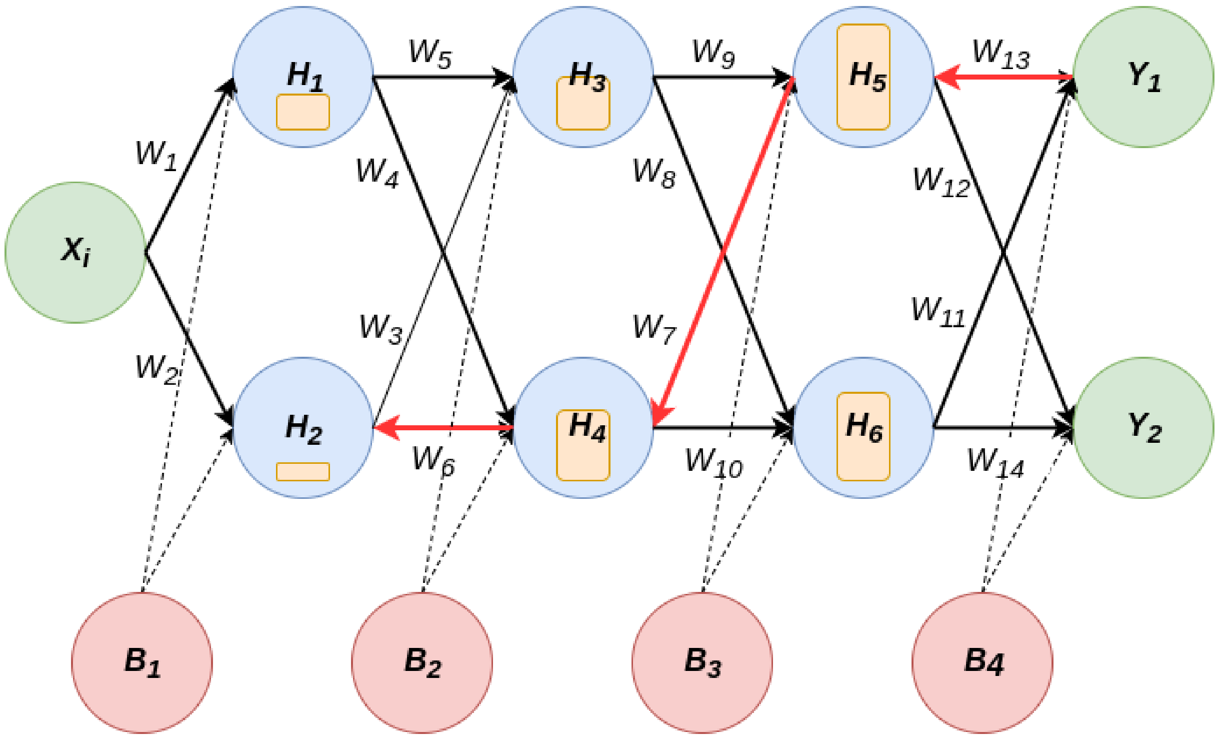

Figure 6.5: *Principle behind back-propagation in a MPL with one input X_i, three layers of hidden units H_i and two outputs Y_i.*

Following the red arrows backwards through the network, the gradient, or error, would be calculated using the chain rule. As an example, after a forward pass, the sum of errors in the output nodes is computed: $E_{total} = E(Y_1) + E(Y_2)$. It is then desired to calculate how much a change in W_{13} affect E_{total}, which is done as;

$$\frac{\partial E_{total}}{\partial W_{13}} = \frac{\partial E_{total}}{\partial Y_{1(out)}} * \frac{\partial Y_{1(out)}}{\partial Y_{1(in)}} * \frac{\partial Y_{1(out)}}{\partial W_{13}} \tag{6.4}$$

Notice that each node have an input and output which the error needs to be calculated between, for simplicity bias is not included. The process is continued backwards through the system, with each computed gradient dependent on the gradients that came before it.

Knowing the gradients in regards to the output error, the weights needs to be updated, this is done by using a optimizer function, e.g. Stochastic Gradient Descent (SGD). For a CNN, which utilizes parameter sharing, as described above, only a single set of weights needs to be updated for each feature map. SGD updates the parameters W using few training sample, as seen in **equation 6.5**:

$$W_{t+1} = W_t - \alpha \cdot \nabla_W \ell(W_t; x_i, y_i) \tag{6.5}$$

Where x_i and y_i are data points from the training set that are sequentially fed to the network. Typically this would be in the form of a minibatch, which is a subset of images within a batch. A batch is the amount of images that the network will be conditioned on throughout an epoch. The idea behind feeding the networa k images in the form of minibatches, is to update its internal parameters based on the result of several predictions, which will make the network better at generalizing and thereby converge in a stable manner. α is the learning rate which determines the step size in the gradient direction, since SGD can have issues with overshooting or oscillation, the learning rate is often set as a small value or decreasing over time. $\nabla_W \ell(W_t; x_i, y_i)$ is the gradient of the loss function, given the input data.

The result of this, is that the network can be slow to converge, various optimization algorithms have therefore bee proposed to used in combination with SGD, e.g. Nesterov momentum or Adaptive Moment Estimation (Adam). Using a standard momentum term, if the gradient is moving down a slope, the velocity will continue to accumulate, resulting in faster convergence, but since the gradient is calculated with regards to the current parameters, it can also result in overshooting if it hits an upgoing hill. Nesterov momentum prevents this by approximating the position of the next parameters and updating the gradients based on this, meaning it can start to slow down if a hill shows up. SGD with Nesterov momentum can be expressed as:

$$W_{t+1} = W_t - \gamma \cdot v_{t-1} + \alpha \cdot \nabla_W \ell(W_t - \gamma \cdot v_{t-1}) \tag{6.6}$$

Where v_{t-1} is the velocity and γ is the momentum factor, which is a value in the range $[0; 1]$.

Nesterov helps in adapting our update function based on the slope, using Adam instead, our updates can also be adapted to the individual parameters, depending on their importance. In order to do this, Adam incorporates an adaptive learning rate into the update function. Adam estimates the first and second order moment, which are expressed as:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1)\dot{g}_t \tag{6.7}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2)\dot{g}_t^2 \tag{6.8}$$

Where m_t is the biased first moment estimate and v_t is the biased second moment estimate. The β components are exponential decay rates and g_t is the gradients at timestep t. Since the moments are initialized as 0, they tend to be biased towards 0 at the initial time steps t when using small decay rates, β values close to 1. In order to counteract this, bias-corrected estimates are computed as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6.9}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{6.10}$$

The final update rule, thereby becomes:

$$W_{t+1} = W_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{6.11}$$

Overall, using Adam compared to e.g. Nesterov momentum means that less time can be spend on fine tuning the hyperparameters, while still achieving fast convergence and good results. The authors behind Adam, suggest using the default values of: $beta_1 = 0.9$, $beta_2 = 0.99$ and the smoothing term $\epsilon = 10^{-8}$. Furthermore, Adam have shown to work well with sparse gradients, which can become a problem for larger networks or throughout training [26].

6.3 Summary

The goal of this chapter have been to give an overview of deep learning methods, especially in regards to CNNs. In **section 6.1**, some of the most important aspects and methods of CNNs was explained, this included back-propagation and an explanation of how a network learns.

One aspect not mentioned is in regards to the size of a network and the effect it have on its performance. As mentioned earlier, each of these layers learn to distinguish different features and the deeper the network the more sophisticated the features potentially become. This has resulted in a trend of continuously making the networks deeper, as an example, AlexNet, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [42] in 2012 with a top-5 error rate of 15.30 %, consisted of five convolutional layers and had 60 million parameters [27]. GoogLeNet, who won the competition in 2014 with a top-5 error rate of 6.70 %, in comparison consisted of 22 layers, but had 12x fewer parameters than AlexNet [46].

Another contender in ILSVRC 2014 was the VGG network, which architecture is based on AlexNet. They experimented with designing a base network and then made the network deeper by adding convolutional layers. The smallest network had eight convolutional layers and the largest 16, all of them was followed by three fully-connected layers, the largest of these networks had 144 million parameters. In their ILSVRC submission they achieved an error rate of 7.3 %, which they later improved to 6.8 %. Most importantly, they showed that by using a standard architecture they could significantly improve the results, simply by making the network deeper [45].

Making a network deeper does not necessarily result in better results as it introduces new problems. As mentioned in **subsection 6.1.1**, it is common practice to utilize a Gaussian or uniform distribution to initialize the network parameters. which have shown to result in the network stopping to learn. Studies have shown that the gradients, calculated through back-propagation, becomes smaller and smaller when moving from the output to the input layer, resulting in the first layers learning slower then the last layers, this is known as the vanishing gradient problem. In general the problem is that, because the gradient in the early layers are a product of terms from all past layers, the gradient becomes unstable, if the parameters are initialized too large it might become a exploding gradient problem [14] [36]. An illustration of this can be seen in **figure 6.5**, where the bars within each node represent the gradient and how it vanishes when approaching the first layers.

In order to counteract this, several methods can be utilized, besides a pooling layer, this could be bbbdropout and batch-normalization layers,

Chapter 7

Region Proposal Network

As mentioned in 5, the first module in the system is the Region Proposal Network (RPN). The objective of the network is to locate objects within the given scene, which can then be fed to the next module in the system. This chapter will start by giving an overview of how the RPN functions and its inspiration, move on to the architecture of the network, before looking into the specific settings for the system during training and testing. Lastly, the ROI image processing will be described, to give an explanation on the data, that is fed into the GAN.

7.1 Overview

As mentioned above, the RPN is used to generate object proposals within a frame, in order to efficiently accomplish this, the system needs to:

1. Be fast - In order to make real-time applications a possibility.
2. Have a high recall rate, no matter the object size.
3. Have a good generalization towards unseen objects.

Various object proposal methods exists, such as Selective Search, based on superpixels and used in e.g. regional-based CNN (R-CNN) and fast R-CNN [13] [12]. Selective Search is slow when generating a large amount of region proposals, instead the RPN is based on the one introduced in faster R-CNN [41].

The RPN can be seen as running in two steps:

1. Find all possible locations of objects within the given frame and create a list of region proposal.
2. Final classification: For each region proposal, obtained in the previous step, perform binary classification to decide whether it belongs to an object or background class.

Since the following systems are dependent on the ROIs output from the RPN, as an object not detected in this stage can not be processed in the following subsystems. The system is setup to generate a number of proposals, that should both be large enough detect all the objects within the frame, while not slowing down the system significantly.

7.2 Network Architecture

As mentioned, the Region Proposal Network is based on the network proposed in [41]. In Faster R-CNN, they utilize an initial network to generate feature maps, from which they both generate object proposals and classifies the object, making the network trainable end-to-end. The original RPN uses a VGG-16 architecture to generate the initial feature maps. For this project, it have instead been changed to a ResNet (Residual Network), which architecture was first proposed in [19], the network was later revised in [20], the network have shown state of the art performance for classification tasks, e.g. the ResNet with 1001 layers achieved an accuracy of 4.92 % on the CIFAR-10 dataset and 22.71 % on the larger CIFAR-100 dataset.

As mentioned in **section 6.3**, deep networks can have trouble with vanishing or exploding gradients, ResNet tries to solve this by incorporating shortcut connections that performs identity mapping. The idea is to create a "direct" path for the gradient during back-propagation and thereby solve the issue with unstable gradients in layers further away from the output. Another added benefit from this, is that it allows deep networks to be trained from scratch. The RPN network is seen in **figure 7.1**:

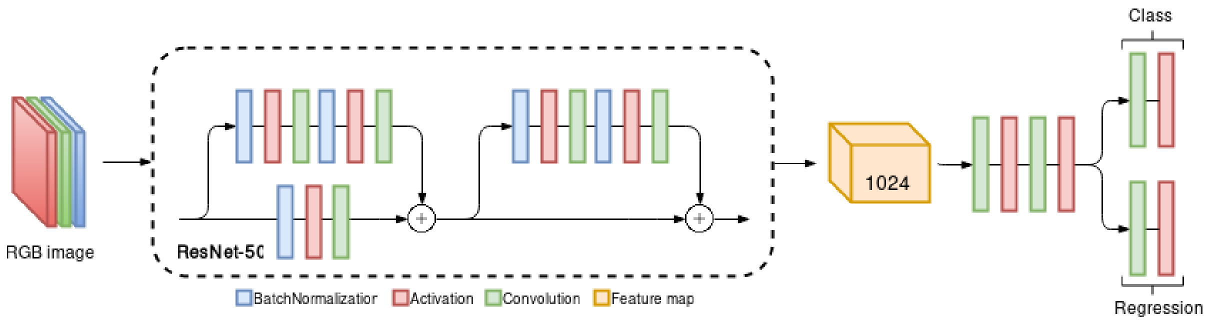

Figure 7.1: *Overview of the architecture used in the Region Proposal Network.*

Furthermore the figure shows the basic architecture of the ResNet, with a down-sampling block, followed by an identity block.

A ResNet is build up of these blocks, in the used implementation each down-sampling block is followed by two identity mappings, each of the blocks consist of three pairs of Batch Normalization, Activation and Convolution. In each of the blocks the first convolution have a kernel with size 1×1 and the following two have a kernel with size 3×3 and a stride of 1. In the down-sampling block, the first convolution have its stride set to 2 and a convolution with kernel size 1×1 and stride 2 is added to the shortcut connection. The depth of the feature maps is increased within each down-sampling block.

As with Faster R-CNN, a mini network with a kernel size of 3×3 is used as a sliding window across the ResNets output feature map, before being split into two convolutional outputs with kernel size 1×1. The classification output utilizes a softmax activation while the other one performs linear regression, in order to generate the bounding box coordinates. In order for the binary classifier to learn to distinguish between objects, the 21 classes, including background, of the PASCAL VOC 2012 dataset is used, but changed to the two classes of object or background.

For each of the sliding-windows, multiple region proposals a predicted, the number of proposed regions is defined by the amount of anchors used. As in the paper, three scales and aspect ratios for these windows is used, this includes the scales: [128, 256, 512] and three ratios: [[1,1], [1,2], [2,1]], resulting in nine anchors at each sliding window position. These anchors helps the network in being translation-invariant and makes it possible to detect objects with multiple scales and aspect ratios.

7.3 Training

During training, we follow the original procedure for the RPN, while discarding the object detection part of Faster R-CNN; Each anchor is assigned a label of being an object or not. Furthermore they are divided into positive and negative samples, where the positive samples are defined as having the highest Intersection-Over-Union (IoU) with a ground truth bounding box or an anchor has an IoU overlap higher than 0.7 with any ground truth box. If the anchors have an IoU lower than 0.3 for all the ground thruth boxes, they are labeled as negative samples, anything in between is discarded. 256 anchors is randomly sampled within the input image and the positive and negative samples are split equally, unless there is not enough positive samples, in which case more negatives are used Since the proposed regions by the RPN is used as input to the rest of the system, it is decided to only use 32 regions, these are split evenly between positive and negative samples, unless there is less than 16 positive samples, in which case more negative samples are used. Since the Generative Adversarial Network should be able to distinguish between objects and background, this is acceptable.

In order to make sure that the RPN proposes ROIs that can be used when training the GAN, it is pretrained for 100 epochs, such that it have a sense of objectness. Unlike the original paper, the network is trained using the Adam optimizer, with a learning rate of 0.00001 and the default β values of 0.9 and 0.999. For classification, the loss function is the binary cross entropy, while the regression loss is a smooth L1 distance between the proposed coordinates and ground truth.

7.4 ROI Image Processing

After the RPN have found the objects of interest, they will need to be prepared for the GAN to receive it as input. An overview of the system can be seen in **figure 7.2**.

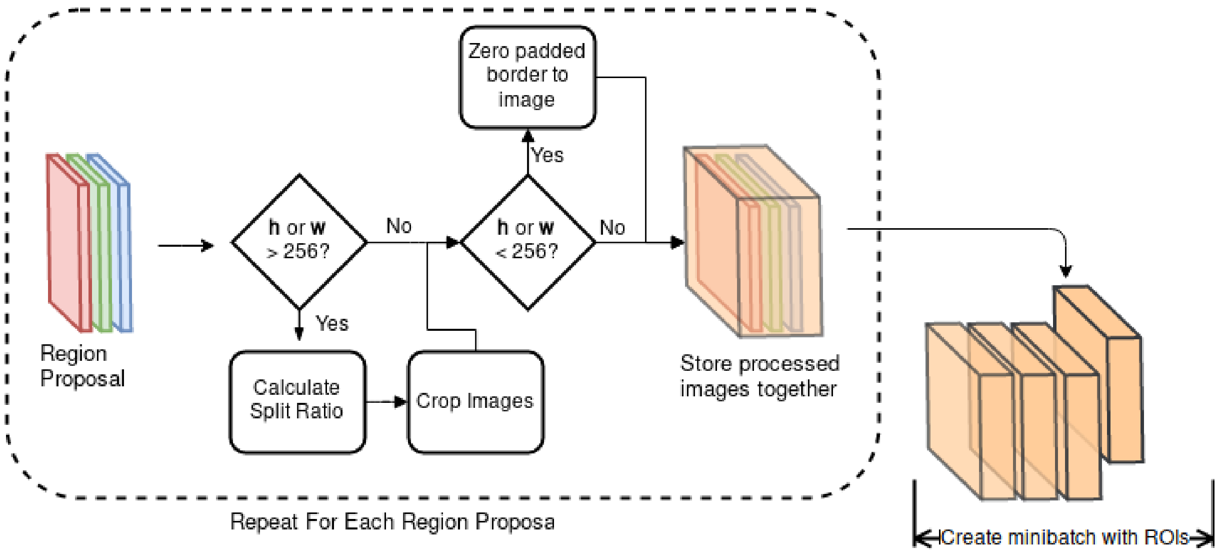
Region Proposal
h or w > 256?
No
Yes
Calculate Split Ratio
Crop Images
h or w < 256?
Yes
Zero padded border to image
No
Store processed images together
Repeat For Each Region Proposa
Create minibatch with ROIs

Figure 7.2: *Overview of the preprocessing module.*

The system receives the batch of region proposals and processes them one by one. The main objective of the system is to correct the size of each ROI, since the GAN needs to receive a specific image size. The reason for this, is that GANs are harder to train as the resolution increases, since there is extra details to learn, the result is that it outputs images of worse quality if the image resolution is too large. Until now, robustly producing images of high resolution, involves more complex solutions, like stacking multiple GANs together [50]. For this project the images will have the dimensions: $256 \times 256 \times 3$.

Since part of the objective for the GAN is to create a mask of the image, it is undesirable to warp the image in order to resize it, as is otherwise common practice for CNNs. If a ROI is too small, a black border is added to the image in the given direction, the border is always evenly split between the two affected sides. If the image is too large, the amount of needed splits it should be divided into is determined instead, since it is always rounded up to nearest integer, it is necessary to add a small border to the cropped ROIs afterwards to get the correct dimensions. Each of the new splits are then used as ROIs, but stacked together when preparing them as a minibatch to feed the GAN. As mentioned in **section 7.3**, the RPN is asked to only forward 32 region proposals, since the training of system is done on an Asus ROG Strix NVidia GeForce GTX 1070 GPU with 8 gb of RAM, a minibatch with 32 $256 \times 256 \times 3px$ images is too large. Splitting some ROIs results in an even larger minibatch, to counteract this, the system is restricted to not feed the GAN with more than 16 images at a time and otherwise it will be split into multiple minibatches.

An example of an input image, for which region proposals are first generated, before preprocessing the proposals, can be seen in **figure 7.3**.

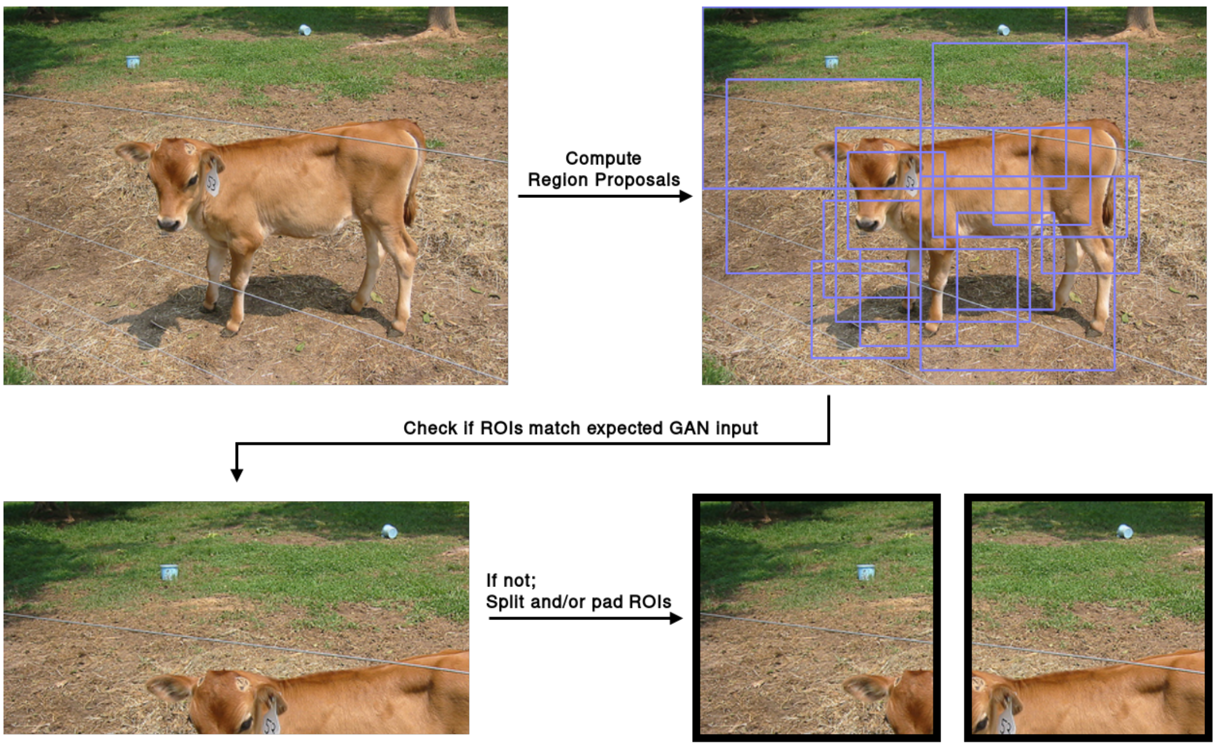

Figure 7.3: *Example of an input image, with unprocessed region proposals provided by the RPN, where one ROI is too wide and therefore cropped and padded to match the expected dimensions.*

The examples here have shown using an RGB image, but the labeled target images goes through the same process, in the same order, such that the image pairs always match each other.

7.5 Summary

This chapter have looked into the system used for generating region proposals, which is largely based on the Region Proposal Network of Faster R-CNN, but decoupled from the Fast R-CNN classification network.

Chapter 8

Generative Adversarial Network

Having explained how the data is retrieved and processed before reaching the Generative Adversarial Network, this chapter will explain how the data is utilized.

8.1 Overview

Generative Adversarial Networks is a generator framework that utilizes two networks and pits them against each other [16].

As originally explained by the authors, the two networks can be seen as taking the role of a counterfeiter and a police officer, in this case the generator is the counterfeiter, while the adversarial network is the police officer. The counterfeiters goal is to create fake currency, that are convincing enough for the police officer not being able to tell the difference. Since the counterfeiter have never seen real currency, he will start by creating something random and hand it to the police officer. Depending on how good he is, the police officer will either believe it is real currency or not, If he rejects it, the counterfeiter will be told what should happen for him to believe it was real. The counterfeiter thereby have to go back and try again, the next time the police officer may believe it is real, in that case he have an adversary that tells him it was wrong and show him how it should have looked.

This approach corresponds to a minimax two player game, in which the two networks forces each other to improve.

Keep this game going and at some point, the police officer should be unable to tell the difference. The minimax game is defined by:

$$\min_G \max_D (D,G) = E_{x\ p_x(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(Z)))] \tag{8.1}$$

If this is actually the case is still an open question, as it is not known if a Nash Equilibrium exist, where the generator wins.

Though the theory sounds simple, the networks are hard to train, as their now is two connected deep networks, for which back-propagation needs to move through.

The First GAN was developed using Multi-Layer Perceptrons, but later versions are using deep convolutional GANs (DCGAN) instead and have shown able to generate realistic data [39]. Which discovered that:

1. Batch normalization should be used in both networks.
2. Avoid pooling and use convolutional stride instead.
3. Use ReLU activations in both networks.

Furthermore, with the networks being able to generate realistic images, might mean that they have learned to understand their structure.

8.2 Generative Adversarial Network Variations

With time, variations of GAN have shown up, as an example, **figure 8.1** shows an comparison between DCGAN, mentioned above and two of the newer variations for image generation.

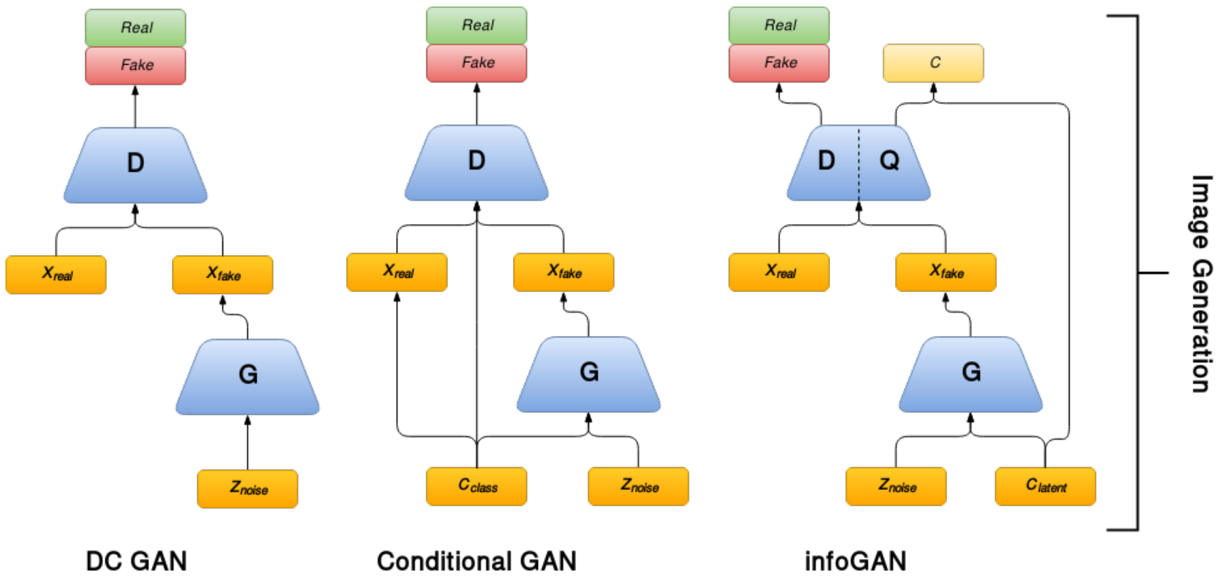

Figure 8.1: *Comparison between some of the best known GAN architectures, used for image generation.*

Taking a closer look at the DCGAN to begin with, it can be seen that the generator input is random noise from a uniform distribution in the range $[-1, 1]$, which it uses to encode the learned a data representations, Furthermore, during training, the discriminator interchangeably receives either real or fake data as input, defined by a Bernoulli distribution.

The Conditional GAN incorporates the class labels and inputs them to both of the networks, The result is that they have been "conditioned" on that variable, meaning that during testing, the output of the generator can be controlled by inputting the correct label.

InfoGAN does almost the same, but instead of the ground truth labels, they use a latent variable. They also include an auxiliary network, which shares most of its parameters with the discriminator, but instead of determining if an image is fake or real, it tries to classify the image as a latent variable.

8.3 Image-To-Image Translation

The networks mentioned above are all used to generate images, from a noise input. Another approach is to use a variation of the conditional GAN, which instead of a label, is conditioned on an image. The result is that the generator will learn to translate images between domains, e.g. colour a black and white image, this process is known as image-to-image translation [25].

In **figure 8.2** the principle behind the conditional GAN used for image-to-image translation in [25] is shown. As can be seen, both networks now receive the real input image instead.

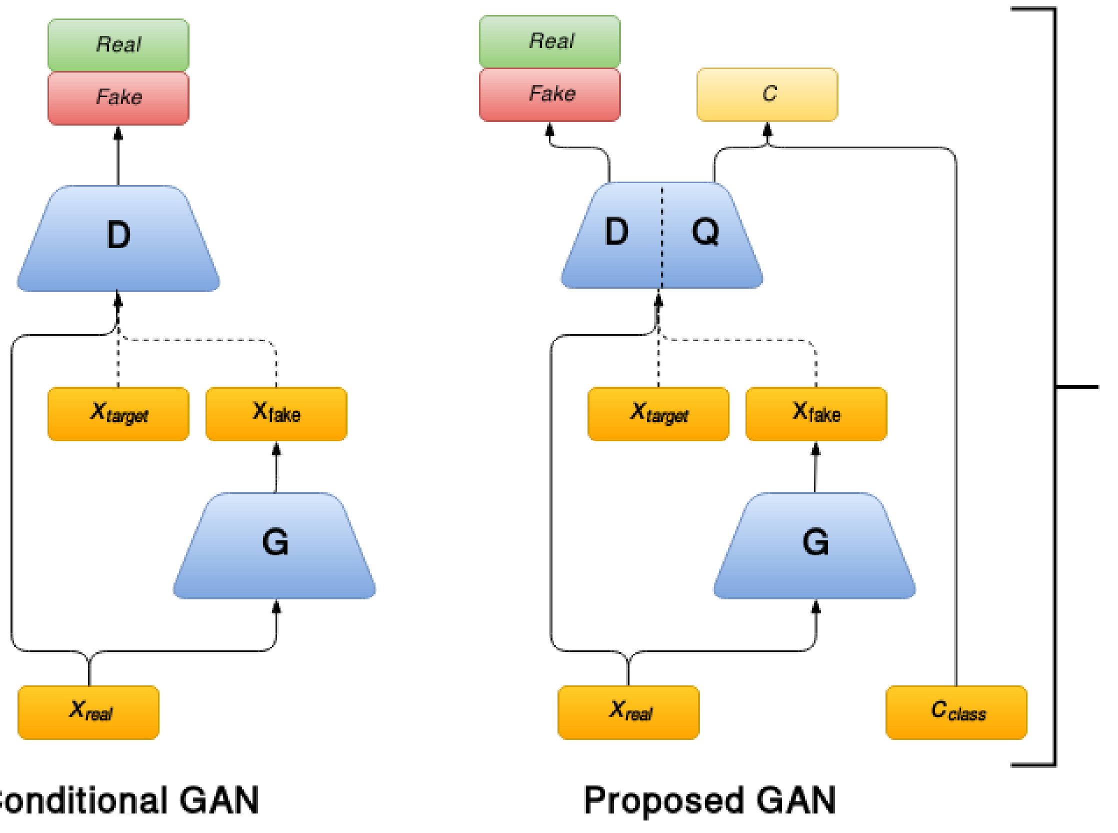

Figure 8.2: *The conditional GAN used for image-to-image translation in [25] and the proposed enhancements.*

By threating semantic segmentation as an image-to-image translation, going from an scene input image to a generated masked image, instead of a pixel-wise classification problem, a new network can be proposed. The network is inspired by the auxiliary network in the infoGAN and can be seen next to the conditional GAN. The basic idea, is that implementing an auxiliary network that shares its weights with the discriminator, an almost cost free, in terms of parameters and processing time, object classifier could be incorporated. The auxiliary network would thereby help classifying the objects within each ROI.

In mask R-CNN, they decouple the classifier and mask, their results showed that once the whole object had been classified, it was sufficient to predict a binary mask without concern for the classes, making training easier [21].

8.4 Proposed Generative Adversarial Network Architecture

Taking a closer look at the proposed GAN, for semantic segmentation the generator will be given a region proposal from the real data, from which it will generate a new image. For the discriminator, the task will no longer be to determine if the generated image is fake or not. Instead it will receive the input image, that the fake image was conditioned on and determine if the pair looks real or not.

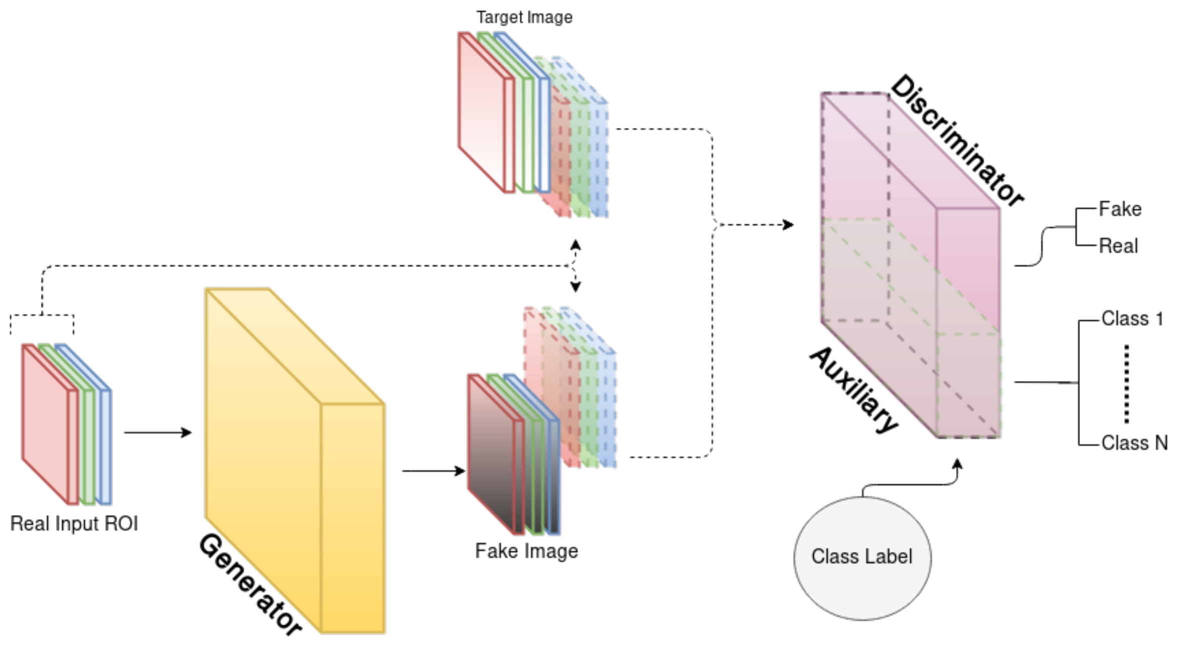

Figure 8.3: *Overview of the proposed Generative Adversarial Network.*

The task of the auxiliary network is then to predict a class given the image pair.

Training the GAN is done in two steps, first the discriminator is trained alone, receiving either a fake generated image or a real image. Afterwards the generator is trained, this is done through the whole GAN, meaning that the weights in the discriminator is frozen, the generator then generates an images, sends it to the discriminator and is then updating using back-propagation throughout both networks. The generator is therefore never shown a labeled image and will only learn how to generate the mask using back-propagation from the discriminator, making the semantic segmentation semi-supervised.

All weights are initialized using a uniform distribution and uses the Adam optimizer with a learning rate of 0.0002, $\beta_1 = 0.5$ and $\beta_2 = 0.99$.

Furthermore, in order to stabilize training, label flipping and label smoothing is incorporating. Label flipping is used for both networks and randomly flips the labels to confuse the networks. Label smoothing adds a smoothing term to the discriminator labels, such that it is between 0.9 and 1.

8.4.1 Generator Network Architecture

The generator network utilizes the same U-Net architecture as found in [25] and can be seen in **figure 8.4**.

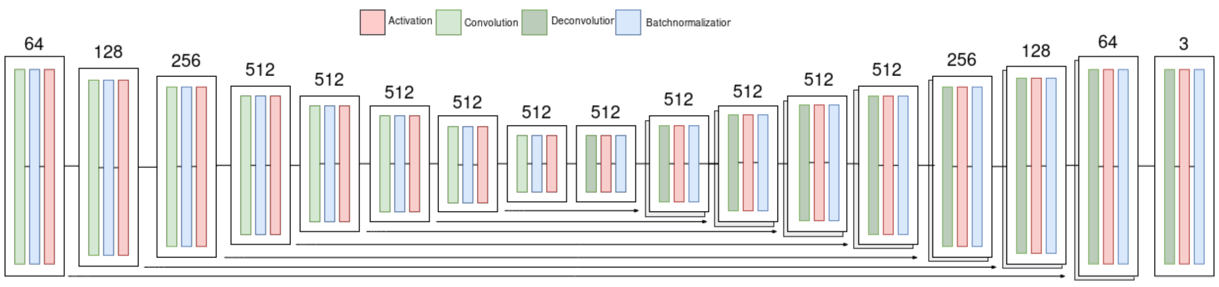

Figure 8.4: *Overview of the architecture used for the generator network.*

The architecture follows a standard autoencoder, with a parallel encoder - decoder layout. In order to retain information from the input image, the feature maps from the parallel encoder layers are concatenated in the decoder part. In the encoder part, the activation function is Leaky ReLU with *alpha* = 0.2, which help stabilizing training according to [39], in the decoder it uses a ReLU. All weights are initilized using a uniform distribution. Otherwise it uses a kernel of size 4×4 and stride (2,2) throughout both the encoder and decoder layer, the three first layers in the decoder implements Dropout with a probability of 20 %. Dropout is a regularization term, which helps prevent overfitting, by dropping random units, in a GAN it stabilizes the training.

Lastly the output is the activation function tanh, in [39] they found using the bounded activation function, helped the network learn faster and better cover the color space of the data distribution. The generator uses the l1 loss term, which might result in less sharp results, but more stable training.

8.4.2 Discriminator Network Architecture

The discriminator network have leaky ReLU in all of its layers and also uses kernels with size (4×4), but with stride (1,1) as downsampling is handled by maxpooling layers. The discriminator output first utilizes a convolution to reduce the depth to 1, its output is a sigmoid activation which returns a scalar in the range [0,1], representing the probability of real data.

The auxiliary network include an average pooling layer followed by two fully connected layers, reducing the parameters to the number of classes, in this case 21. The output is a softmax.

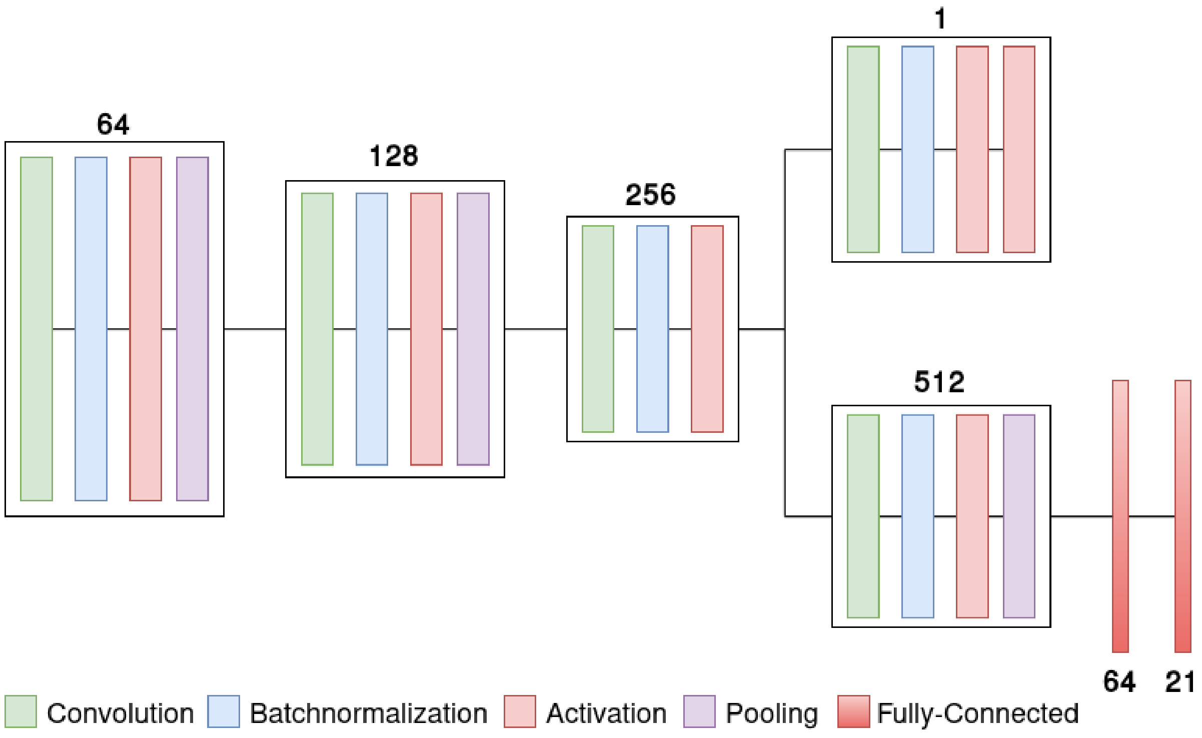

Figure 8.5: *Overview of the architecture used for the discriminator network.*

The discriminators loss function is binary crossentropy, while it is categorical crossentropy for the classifier.

8.5 Summary

Having presented the full network, it is time to look at the training results.

Chapter 9

Image Post-processing

The post-processing system is the last part of the complete system and is part of the validation of the system. Since the GAN makes predictions on cropped regions of a high resolution image, it is desirable to stitch these together to create a segmented image with same resolution as the input image.

This is the main purpose of the post-processing system, when testing, the coordinates for the region proposals are saved, together with their original dimensions. This way, the system can remove the black border if necessary and if a region have been split, it start by concatenating those together. Lastly a black canvas with the same dimensions as the input image is created, into which each ROI can be placed as they were before. As an example, **figure 9.1** shows how an input image have been cropped according to the ROIs, how the bounding boxes are located within the frame and the stitched result. The blue bounding boxes on the right, represent bounding boxes that were not split doing the pre-processing step, while the red bounding boxes were split and had to be stitched separately, before being placed on the canvas.

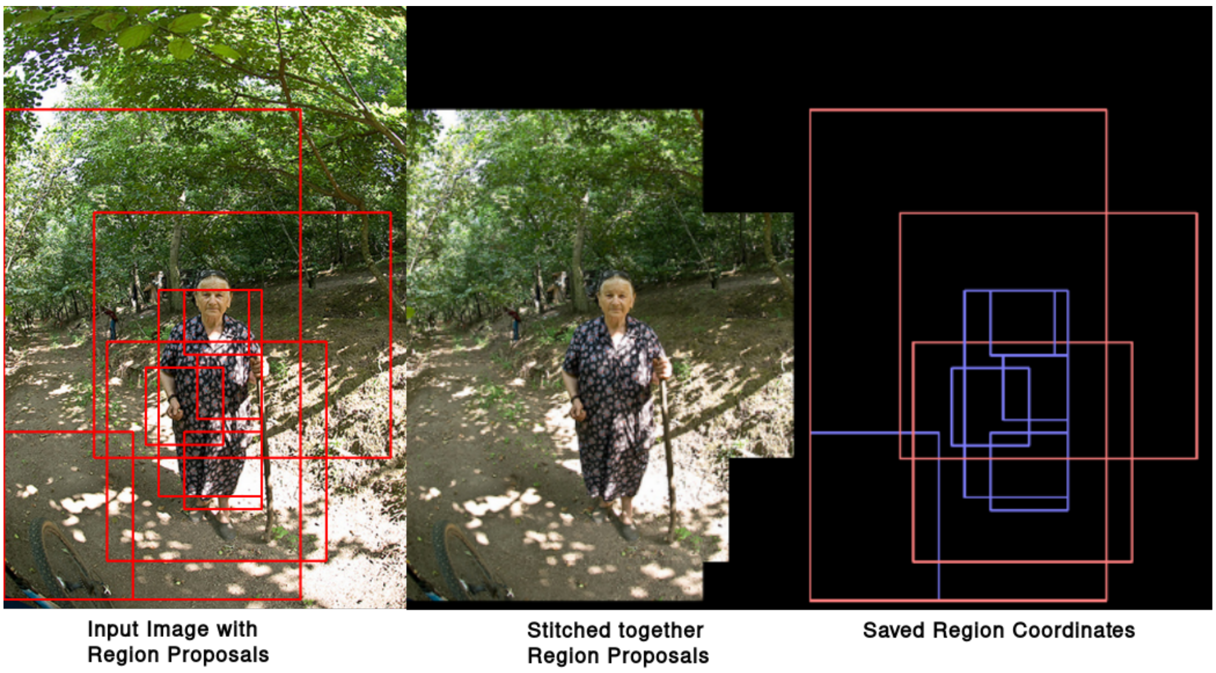

Figure 9.1: *Example of an input image, where the region proposals have been cropped, remembered and stitched back together again. The blue bounding boxes are region proposals which have not been split in order to match the required dimensions, while the red bounding boxes have been split*

As can be seen, the stitching works as intended, but the result can also end up looking less impressive. An example is **figure 9.2**, where the GAN have generated images for each region proposed for the input image, which are then stitched together.

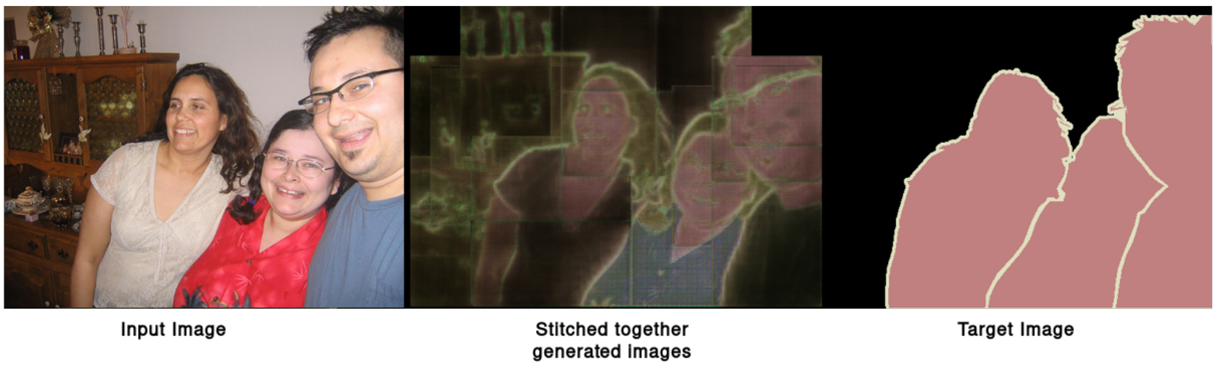

Figure 9.2: *Example where outputs from the generator, predicting on ROIs from the input image, have been stitched together.*

If the RPN have not included a relevant area, as the top of the front guys head, it will show as a black square. It can also be seen that, if the generator does not generate smooth edges on its images, the areas where the ROIs meet, are very noticeable.

All training and testing is done using the PASCAL VOC 2012 dataset, which consist of 21 classes in the object detection and segmentation task.

Part III

Evaluation

Chapter 10

Acceptance Test

Unfortunately, the GAN have not been able to produce satisfactory results, meaning an acceptance test have not been conducted.

Chapter 11

Closure

11.1 Conclusion

The goal for the project was to create a system that could learn a representation of our world, this should have been done by encouraging a Generative Adversarial Network to generate segmentation masks. Since GANs have trouble generating high-quality images larger than $256px$, a Region Proposal Network was implemented, the network was trained to distinguish between objects and background and was responsible for proposing regions to feed the generator. A preprocessing module was implemented as well, which biggest task was to make sure that the generator would receive correctly dimensioned Regions Of Interest. These systems worked as intended and fed the GAN with the proper data.

A post-processing system was also implemented, which was mostly used during testing. The system would undo the work done by the RPN and preprocessing module, stitching the generated images into a high resolution image.

Unfortunately, GANs can be hard to train, depending on the problem, they might need a lot of training iterations, before they start producing convincing results. Furthermore, if one of the networks becomes too overpowered compared to the other one, the weak one will stop learning. Normally it is desired to have the discriminator perform a little better than the generator.

The overall conclusion is that the desired results was not obtained. The generator never learned to output a good mask, though some experiments showed that it got an idea of the correct color to use for the object, meaning that it might have been a question of training time.

11.2 Discussion

With the results in mind. there is a lot to consider for future work. First of all, the classifier would need to be tested and the same with the precision of the Region Proposal Network.

For the GAN, there are several things that could still be experimented with, as training is more difficult and takes longer, one option is to try with smaller images, e.g. $64 \times 64 \times 3$. Since the network have shown to produce good results on smaller more specific datasets, an option could be to try another dataset, this could for example be the MSCOCO dataset.

Another option is to let the network keep training, as the loss for some experiments still seems to improve.

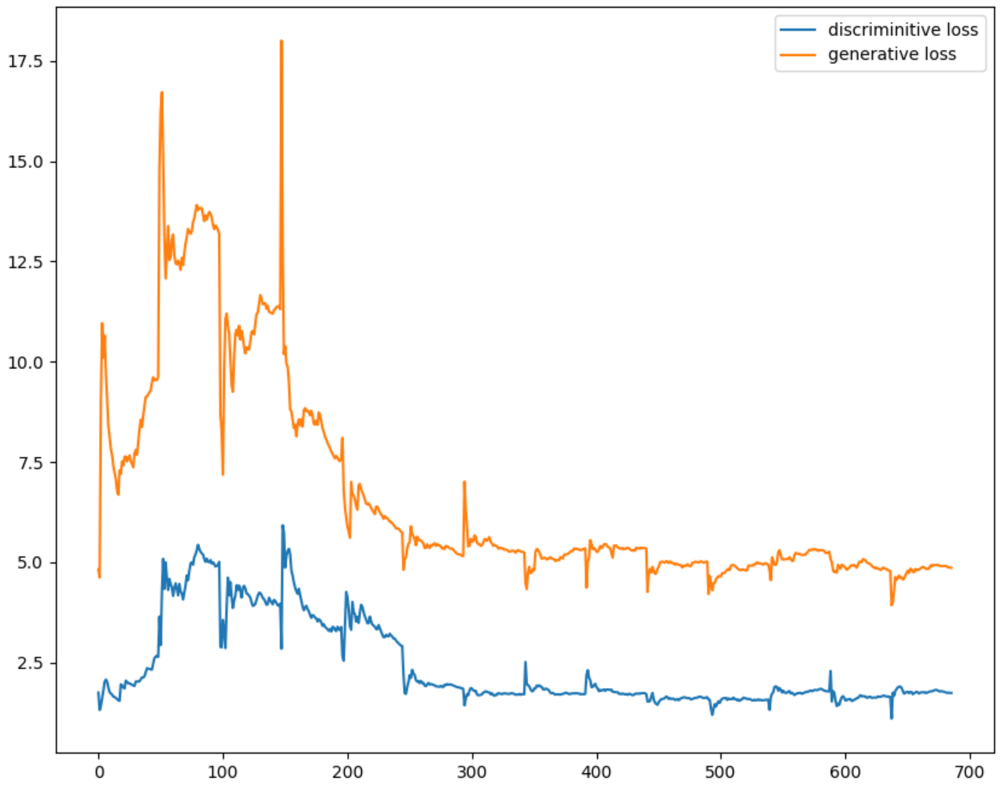

Figure 11.1: *Plot of training loss for the generator and discriminator*

Chapter 12

Bibliography

[1] World Robotics Report 2016: European Union occupies top posion in the global automation race. Technical report, International Federation of Robotics, Sept. 2016.

[2] Executive Summary World Robotics 2016 Service Robots. Technical report, International Federation of Robotics, 2016.

[3] A. Arnab and P. H. S. Torr. Bottom-up Instance Segmentation using Deep Higher-Order CRFs. *ECCV*, pages 1–12, 2016. URL `http://arxiv.org/abs/1609.02583`.

[4] A. Arnab, S. Jayasumana, S. Zheng, and P. Torr. Higher Order Conditional Random Fields in Deep Neural Networks. *Arxiv*, page 10, 2015. ISSN 0302-9743. doi: 10.1007/978-3-319-46475-6. URL `http://arxiv.org/abs/1511.08119`.

[5] H. Caesar, J. Uijlings, and V. Ferrari. Region-based semantic segmentation with end-to-end training. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 9905 LNCS, pages 381–397, 2016. ISBN 9783319464473. doi: 10.1007/978-3-319-46448-0_23.

[6] H. Caesar, J. R. R. Uijlings, and V. Ferrari. Region-based semantic segmentation with end-to-end training. *CoRR*, abs/1607.07671, 2016. URL `http://arxiv.org/abs/1607.07671`.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *arXiv Prepr.*, pages 1–14, 2016. URL `http://arxiv.org/abs/1412.7062`.

[8] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor Semantic Segmentation using depth information. *arXiv Prepr. arXiv ...*, cs.CV, 2013. URL `http://arxiv.org/abs/1301.3572v2{%}5Cnfile:///Users/poliveirap/Dropbox/Library.papers3/Articles/2013/Couprie/arXivpreprintarXiv{\protect.\kern\fontdimen3\font.\kern\fontdimen3\font.\kern\fontdimen3\font}2013Couprie.pdf{%}5Cnpapers3://publication/uuid/155C3758-59E3-47EB-9F64-C0D6650DA42B`.

[9] J. Dai, K. He, and J. Sun. Instance-aware Semantic Segmentation via Multi-task Network Cascades. *arXiv:1512.04412*, pages 3150–3158, 2015. ISSN 10636919. doi: 10.1109/CVPR.2016.343.

[10] J. Dai, K. He, and J. Sun. BoxSup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proc. IEEE Int. Conf. Comput.*

Vis., volume 11-18-December-2015, pages 1635–1643, 2016. ISBN 9781467383912. doi: 10.1109/ICCV.2015.191.

[11] J. Fan, Y. Gao, H. Luo, and R. Jain. Mining multilevel image semantics via hierarchical classification. *IEEE Trans. Multimed.*, 10(2):167–187, 2008. ISSN 15209210. doi: 10.1109/TMM.2007.911775.

[12] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL `http://arxiv.org/abs/1504.08083`.

[13] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL `http://arxiv.org/abs/1311.2524`.

[14] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proc. 13th Int. Conf. Artif. Intell. Stat.*, 9:249–256, 2010. ISSN 15324435. doi: 10.1.1.207.2059. URL `http://machinelearning.wustl.edu/mlpapers/paper{_}files/AISTATS2010{_}GlorotB10.pdf`.

[15] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *AISTATS '11 Proc. 14th Int. Conf. Artif. Intell. Stat.*, 15:315–323, 2011. ISSN 15324435. doi: 10.1.1.208.6449.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[17] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[18] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. *Learning Rich Features from RGB-D Images for Object Detection and Segmentation*, pages 345–360. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10584-0. doi: 10.1007/978-3-319-10584-0_23. URL `http://dx.doi.org/10.1007/978-3-319-10584-0_23`.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00124. URL `http://arxiv.org/pdf/1512.03385v1.pdf`.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 9908 LNCS, pages 630–645, 2016. ISBN 9783319464923. doi: 10.1007/978-3-319-46493-0_38.

[21] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL `http://arxiv.org/abs/1703.06870`.

[22] S. Hong, H. Noh, and B. Han. Decoupled Deep Neural Network for Semi-supervised Semantic Segmentation. *Nips*, pages 1–9, 2015. ISSN 10495258.

[23] Q. Huang, W. Wang, K. Zhou, S. You, and U. Neumann. Scene labeling using recurrent neural networks with explicit long range contextual dependency. *CoRR*, abs/1611.07485, 2016. URL http://arxiv.org/abs/1611.07485.

[24] ISO 8373:2012(en). Robots and robotic devices – Vocabulary. Standard, International Organization for Standardization, Geneva, CH, Mar. 2012.

[25] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL http://arxiv.org/abs/1611.07004.

[26] D. P. Kingma and J. L. Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

[27] A. Krizhevsky, G. E. Hinton, and I. Sutskever. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.*, pages 1–9, 2012. ISSN 10495258. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[28] Y. LeCun. Generalization and network design strategies, 1989. URL http://masters.donntu.edu.ua/2012/fknt/umiarov/library/lecun.pdf{%}5Cnpapers3://publication/uuid/2C3E3B3C-DE43-4109-A816-7220A5C08617.

[29] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(1):436–444, 2015. ISSN 1548-7091. doi: 10.1038/nature14539. URL http://www.nature.com/nature/journal/v521/n755/full/nature14539.html.

[30] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully Convolutional Instance-aware Semantic Segmentation. *arXiv Prepr.*, 2016. URL http://arxiv.org/abs/1611.07709.

[31] Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin. RGB-D scene labeling with long short-term memorized fusion model. *CoRR*, abs/1604.05000, 2016. URL http://arxiv.org/abs/1604.05000.

[32] X. Liang, Y. Wei, X. Shen, J. Yang, L. Lin, and S. Yan. Proposal-free Network for Instance-level Object Segmentation. *CoRR*, 1509.02636(X):1–14, 2015. URL http://arxiv.org/abs/1509.02636.

[33] G. Lin, C. Shen, I. Reid, and A. van dan Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *Arxiv 2015*, pages 1–13, 2015. ISSN 10636919. doi: 10.1109/CVPR.2016.348. URL http://arxiv.org/abs/1504.01013.

[34] G. Lin, A. Milan, C. Shen, and I. D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *CoRR*, abs/1611.06612, 2016. URL http://arxiv.org/abs/1611.06612.

[35] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 3431–3440, 2015. ISSN 10636919. doi: 10.1109/CVPR.2015.7298965.

[36] M. A. Nielsen. Neural networks and deep learning.

[37] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proc. IEEE Int. Conf. Comput. Vis.*, volume 11-18-December-2015, pages 1520–1528, 2016. ISBN 9781467383912. doi: 10.1109/ICCV.2015.178.

[38] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. *Learning to Refine Object Segments*, pages 75–91. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46448-0. doi: 10.1007/978-3-319-46448-0_5. URL `http://dx.doi.org/10.1007/978-3-319-46448-0_5`.

[39] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL `http://arxiv.org/abs/1511.06434`.

[40] M. Ren and R. S. Zemel. End-to-End Instance Segmentation and Counting with Recurrent Attention. *1605.09410v2*, pages 1–17, 2016. URL `http://arxiv.org/abs/1605.09410`.

[41] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL `http://arxiv.org/abs/1506.01497`.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[43] H. Schulz and S. Behnke. Depth and height aware semantic rgb-d perception with convolutional neural networks, 2015.

[44] S. Shi, Q. Wang, P. Xu, and X. Chu. Benchmarking State-of-the-Art Deep Learning Software Tools. *ArXiv Prepr.*, 2016.

[45] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ImageNet Chall.*, pages 1–10, 2014. ISSN 09505849. doi: 10.1016/j.infsof.2008.09.005. URL `http://arxiv.org/abs/1409.1556`.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, volume 07-12-June-2015, pages 1–9, 2015. ISBN 9781467369640. doi: 10.1109/CVPR.2015.7298594.

[47] A. M. Tousch, S. Herbin, and J. Y. Audibert. Semantic hierarchies for image annotation: A survey, 2012. ISSN 00313203.

[48] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 9796 LNCS, pages 14–25, 2016. ISBN 9783319458854. doi: 10.1007/978-3-319-45886-1_2.

[49] Z. Wu, C. Shen, and A. van den Hengel. Bridging Category-level and Instance-level Semantic Image Segmentation. *arXiv Prepr.*, 2016. URL `http://arxiv.org/abs/1605.06885`.

[50] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016. URL `http://arxiv.org/abs/1612.03242`.

[51] Z. Zhang, S. Fidler, and R. Urtasun. Instance-level segmentation for autonomous driving with deep densely connected mrfs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 669–677, June 2016. doi: 10.1109/CVPR.2016.79.

[52] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional Random Fields as Recurrent Neural Networks. *Int. Conf. Comput. Vis.*, page 16, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.179. URL `http://arxiv.org/abs/1502.03240`.