BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FAST ANALYSIS OF BORDERS IN IMAGE

RYCHLÁ ANALÝZA HRANIC V OBRAZE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MATEJ KOLESÁR**
AUTOR PRÁCE

SUPERVISOR **prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Kolesár Matej**
Programme: Information Technology
Title: **Fast Analysis of Borders in Image**
Category: Image Processing

Assignment:

1. Study the problematic of fast analysis of borders in image; focus on modern approaches based on machine learning.
2. Find and describe existing data sets for learning and evaluation of algorithms of fast analysis of borders in image.
3. Select suitable methods and experiment with them on suitable data. Discuss the properties of the solved methods.
4. Identify a suitable real-world problem whose solution is based on fast analysis of borders in images, collect relevant data and select a suitable method for solving it.
5. Evaluate the proposed solution in the context of the particular problem and discuss the possibilities and limitations of the selected method.
6. Assess the achieved results and propose possible extensions of the project; create a poster and a short video for presenting the project.

Recommended literature:

- https://arxiv.org/abs/1504.06375v2
- Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, Zhijiang Zhang: DeepContour: A Deep Convolutional Feature Learned by Positive-Sharing Loss for Contour Detection, CVPR 2015
- Gedas Bertasius, Jianbo Shi, Lorenzo Torresani: DeepEdge: A Multi-Scale Bifurcated Deep Network for Top-Down Contour Detection, CVPR 2015
- Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Kai Wang, Xiang Bai: Richer Convolutional Features for Edge Detection, CVPR 2017

Requirements for the first semester:

- Items 1 through 3, considerable progress on item 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Herout Adam, prof. Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: May 28, 2020
Approval date: November 4, 2019

Abstract

This thesis focuses on the problem of detecting edges in natural images while maintaining high performance per image. First, the existing approaches are analysed and from them the relevant information is extracted. This information is then used to design two architectures that use convolutional neural networks. One architecture is based on RCF and enriches the output, while the other is a combination of RCF and RCN. This combination provides better up-sampling and enriches the output even more. Evaluation was performed on the BSDS500 dataset and the best result was for achieved for the model that combined RCF and RCN with an ODS score of 0.675.

Abstrakt

Táto práca sa zameriava na problém detekcie hrán v prirodzených obrazoch pri zachovaní vysokej rýchlosti pre spracovanie obrázku. Najprv sa analyzujú existujúce prístupy a z nich sa extrahujú príslušné informácie. Táto informácia sa potom použije na navrhnutie dvoch architektúr, ktoré používajú konvolučné neurónové siete. Jedna architektúra je založená na RCF a obohacuje výstup, zatiaľ čo druhá je kombináciou RCF a RCN. Táto kombinácia poskytuje lepšie vzorkovanie a ešte viac obohacuje výstup. Vyhodnotenie sa uskutočnilo na dátovej sade BSDS500 a najlepší výsledok sa dosiahol pre model, ktorý kombinoval RCF a RCN so skóre ODS 0,675.

Keywords

neural network, machine learning, convolution neural network, edge detection

Klíčová slova

neurónové siete, strojové učenie, konvolučné neurónové siete, detekcia hraníc

Reference

KOLESÁR, Matej. *Fast Analysis of Borders in Image.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Rozšířený abstrakt

Úvod

Detekcia hraníc v obraze je jedným zo základných problémov pri spracovaní obrazu. Z tohto hľadiska je veľmi dôležité aby navrhnuté metódy boli rýchle a precízne. Väčšina nových metód používa konvolučné neurónové siete pre detekciu hraníc. V tejto práci budú analyzované staršie aj novšie prístupy pre detekciu hraníc v obraze. Výsledkom je navrhnutie 2 nových architektúr založených na konvolučných neurónových sieťach.

Analýza existujúcich algoritmických metód

Prvé riešenia, ktoré sa zaoberali týmto spôsobom, boli riešené matematickým prístupom. Tieto metódy navrhli algoritmy, ktoré boli založené na identifikovaní časti obrazov, kde sa výrazne mení hodnota intenzity pixlov. Tieto riešenia majú značnú výhodu ak sa na výsledok pozeráme z hľadiska rýchlosti, ale majú aj veľa obmedzení. Tieto obmedzenia sú spôsobené tým, že obrázky nie sú bezchybné a zmena hodnoty intenzity pre pixel nemusí znamenať, že na danej sekcii v obraze existuje hranica. Medzi najznámejšie algoritmy patria napríklad Sobel operátor, Prewitt operátor a Canny Edge detector.

Konvolučné neurónové siete

Konvolučné neurónové siete sú špeciálnym druhom neurónových sietí, ktoré sú založené na tradičných neurónovych sieťach. Boli navrhnuté pre rozpoznávanie dvojrozmerných obrazových vzorov priamo z pixelov obrázkov s minimálnym predspracovaním. Konvolučné neurónové siete tento problém riešia tým spôsobom, že extrahujú príznaky z recepčných poli neurónov. Následne je obrázok spracovávaný sieťou, ktorá optimalizuje interné váhy pre jednotlivé vrstvy s cieľom dosiahnuť čo najlepšiu presnosť. Významnou časťou konvolučných neurónových sietí sú konvolučné vrstvy. Tieto vrstvy obsahujú neuróny, ktoré sú napojené na vstupný obrázok, kde spracovávajú isté okolie na obrázku. Spracované okolie je výstup pre daný neurón a je spojené s ostatnými výstupmi neurónov, ktoré spracovávali inú časť obrázku. Konvolúcia je definovaná ako matematická operácia medzi dvoma funkciami a pri spracovaní obrazu sa používa diskrétny model tejto operácie. Konvolučné vrstvy môžu mať na vstupe výstup inej konvolučnej vrstvy a vďaka tejto vlastnosti je možné ich reťaziť a optimalizovávať výstup.

Najúspešnejšie modely konvolučných neurónových sietí, ktoré sa presadili v oblasti detekcie hrán, sú architektúry založené na VGG16 modeli medzi ktoré patrí HED a RCF. Ďalší model, ktorý dosahoval dobré výsledky, bol model RCN, ktorý je založený na resnet101 architektúre. Existuje veľa ďalších architektúr, ktoré dosiahli dobré výsledky, ale v tejto práci sa zameráme hlavne na modely HED, RCF a RCN. Tieto modely využívajú to, že pri spracovávaní obrazu sa generuje výstup pre každú konvolučnú vrstvu. Snaha týchto modelov je využiť výstupy konvolučných vrstiev a spájať ich do jedného finálneho obrazu. Tieto výstupy sú oddelené do osobitných sekcií a spájané až vo finálnej fáze.

Implementácia a návrh siete

Pre úspešne navrhnutie detektoru je potrebných veľa vecí. Základom je zaobstarať datasety, na ktorých by prebiehalo trénovanie a overovanie. Pre tento účel sa použili existujúce

datasety BSDS500, PASCAl-VOC a NYU Depth V2, ktoré boli modifikované, aby odpovedali vstupu pre architektúry. Po preskúmaní existujúcich riešení boli navrhnuté 2 nové architektúry, ktoré využívajú časti existujúcich riešení a snažia sa zlepšiť výsledky. Prvá architektúra je založená na RCF a hlavnou zmenou je pridanie vrstiev, ktoré spájajú výsledky medzi sekciami RCF modelu. Tieto výsledky sú používané pre vytvorenie finálneho obrázku, ale nie sú posielané na výstup. Originálne medzivýsledky, ktoré vznikli pred spojením, sú posielané na výstup. To je hlavne z dôvodu, že trvá dlhú dobu, kým sa dosiahnu výsledky, ktoré sú použiteľné a stratová funkcia nebola schopná správne vyhodnocovať výsledky, aby sa zlepšovali. Druhá architektúra má založený základ na RCF, ktorý bol pozmenený v počte konvolučných vrstiev a zároveň pridáva bloky z architektúry RCN do danej architektúry. Výhoda tejto modifikácie je zvýšenie parametrov, na ktorých sa učí daná sieť. Zároveň bloky prevzaté z RCN boli navrhnuté tak, aby pri zväčšovaní obrázkov, ktoré boli zmenšené pri prechode navrhnutou sieťou, bola dosiahnutá najlepšia kvalita a nedošlo k degradácii výstupu. Trénovanie navrhnutých modelov prebiehalo na Google Colab, kde je poskytnutá možnosť GPU akcelerácie, ktorá urýchli trénovanie architektúr.

Experimenty

Po navrhnutí boli na modeli prevádzané experimenty s cieľom vylepšenia kvality. Jeden zo základných parametrov, ktoré ovplyvňujú výsledok, je miera učenia, ktorá sa stanoví pre sieť. Experimenty nám ukázali, že aj s vysokou mierou učenia na začiatku je možné natrénovať detektor ak sa miera učenia upraví po začiatočných epochoch. Ďalší faktor hrala stratová funkcia. Testovanie stratovej funkcie umožnilo vybrať funkciu, ktorá produkovala najlepšiu kvalitu.

Záver

Boli navrhnuté 2 nové typy architektúr. Aj keď sa nepodarilo vylepšiť kvalitu detekcie od originálnych modelov, úspech nastal z hľadiska rýchlosti. Siete sú rýchlejšie alebo porovnateľné pre najlepšie prípady. Do budúcna je možnosť túto prácu zlepšiť, a to hlavne zameraním na vylepšenie kvality hraníc, ktoré identifikujú detektory.

Fast Analysis of Borders in Image

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by Matej Kolesár under the supervision of prof.Ing. Adama Herouta, Ph.D. I mentioned all publications and sources that this thesis was based on.

.....
Matej Kolesár
May 28, 2020

Acknowledgements

I would like to thank my supervisor prof. Ing. Adamu Heroutovi, Ph.D for all the recommendations and approaches that he gave me that made this work possible.

Contents

Chapter 1

Introduction

Edge detection represents one of the essential processes in image segmentation. Edges play an essential role as important information about the image is encapsulated in the edges. Since edge detection is used as the base for other classification tasks, it is important to produce high quality edges at a fast speed.

There are many different approaches when it comes to analyzing this issue. One category of methods consists of solving this problem using algorithmic solutions. The well-known solutions in this field are usually older approaches that are fairly fast and accurate, but only under certain conditions. These conditions make the detectors produce lower quality edges. The other category of approaches became popular in the recent years and consists of using artificial intelligence. The best results were achieved when using convolutional neural networks. These networks are not perfect but when compared to the algorithmic approaches, they produce better quality results and do not have as many limitations. Although there are several limitations present, these are usually solved with the datasets that are used to train the network and are one of the deciding factors that determines how the network performs.

The objective of this thesis is to present the existing solutions that solve above-mentioned issues, analyse advantages and disadvantages of various methods and propose new approaches that use convolutional neural network to solve this problem. The structure of this document is outlined as follows: Chapter 2 analyses the algorithmic approaches that were mainly used in the past. Chapter 3 examines the architecture of convolutional neural networks and the proposed types of networks used for edge detection. Chapter 4 covers the metrics used for evaluation of the designed models. Chapter 5 introduces some of the well known datasets used for edge detection. Chapter 6 analyses the advantages and disadvantages of existing neural networks and proposes improvements. Chapter 7 experiments with the proposed solutions and evaluates them.

Chapter 2

Algorithmic Approach to Edge Detection

Edge detection [16] comprises of various mathematical approaches that are used for identification of specific points linked to rapid changes of brightness in a digital image. The points that are associated with the rapid brightness changes are typically organized into a set of edges. Edge detection is a fundamental tool in image processing, machine vision and computer vision and it is particularly essential in the areas of feature detection and feature extraction as some of the most critical information is encapsulated in edges.

2.1 Edge Definition

Nowadays, although there are many definitions of edges, the majority of the explanations only look at the issue from a mathematical standpoint. These definitions include the change of the gradient or an abrupt change of intensity values in the image. However, this creates a problem when taking into consideration the fact that not all of these changes must mean there is an edge. For example, if there is an abrupt change of color intensity on the same object, it can be mistakenly identified as an edge. In general, all algorithmic approaches use one of these methods to detect edges, leading to possible problems. The change of gradients must not mean that an edge was detected and thus, in images where many gradients change, a high amount of false edges can be detected.

The above-explained issue represents a problem between identification of the difference between edge detection and object edge detection. Although both of these notions have many characteristics in common, there are important differences. These differences can be seen in Figure 2.1.

- **Edge detection** identifies changes in image intensity. These changes can occur in the same object and usually produce more complex edge maps.
- **Object edge detection** finds the closed shape of an object and can be used to determine the shape of an object. As this type of detection only focuses on finding closed shaped object the images are much clearer and less complex.

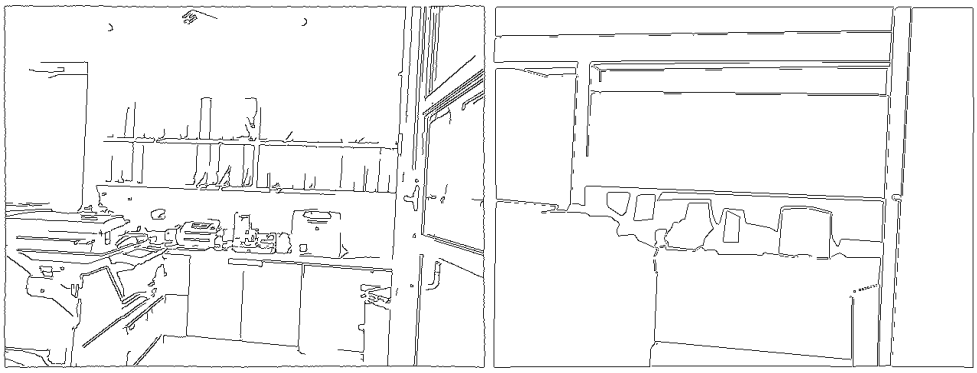

Figure 2.1: Difference between edge detection (left) and object edge detection (right). In the right image, object shapes can be identified where as to in the left image more edges are displayed. [25]

2.2 Historic Approaches to Edge Detection

Despite that there are many existing strategies for edge recognition, the majority of them can be categorized into two groups. The first group is based on approximation of the maximum of first derivatives. The most known approaches that belong to this category include Roberts operator [29], Prewitt operator [28], Sobel operator [34] and Canny edge detector [9]. The second group is focused on the detection of the zero-crossing of second derivatives. The most known approaches are the Laplacian of Gaussian [21] and the Difference of Gaussian [6].

2.3 First Order Derivative Methods

The first order methods used for edge recognition [29] are gradient focused. A gradient is a vector, whose segments measure how quickly pixel values are changing with distance in the horizontal and vertical direction. First order methods take the first derivative of the intensity value across the image and find the points where the derivative is at maximum. Generally, the first order derivative operators are very sensitive to noise and produce thicker edges.

Robert operator

Robert operator [29] represents a first order method that focuses on gradients by computing the sum between diagonally adjacent pixels. The image is convolved with two 2×2 masks. You can seen the representation of these masks in Equation (2.1).

$$D_x = \begin{bmatrix} 1 & 0 \\ -0 & -1 \end{bmatrix} \qquad D_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.1}$$

Sobel Operator

The Sobel operator [34] [29] uses intensity values in a 3×3 neighborhood around each image pixel in order to approximate the corresponding image gradient. It uses an isotropic 3×3 mask consisting of only integer values. It provides two separate masks: one for detecting

edges in a horizontal direction and another one for detecting edges in a vertical direction. The Equation (2.2) represents the matrix. The left matrix represents the vertical direction and the right matrix represents horizontal .

$$D_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & -0 & -1 \end{bmatrix} \qquad D_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.2}$$

Prewitt Operator

The Prewitt operator [28] [29] calculates edges by using difference between corresponding pixel intensities of an image. Similar to the Sobel operator mentioned in section 2.3, it consists of two derivative masks used for horizontal and vertical direction. This can be seen in the Equation (2.3) where the left matrix represents vertical direction and the right matrix represents horizontal direction.

$$D_x = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad D_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Canny Edge

The Canny edge detector [9] represents a multiple stage algorithm to detect edges in an image. There are two key parameters: an upper threshold and a lower threshold. The upper threshold is used to mark edges that are certainly edges. The lower threshold is used to find areas that are only a piece of an edge. The algorithm consists of 4 different stages that process the image:

1. **Blurring** – as most algorithmic edge detectors are prone to noise, there is a need to smooth the image. This is usually performed by blurring the image using a Gaussian blur.

2. **Finding gradients** – gradient magnitudes and directions are calculated at every single pixel in the image. The magnitude of the gradient at a pixel determines if it can be classified as an edge.

3. **Non-maximum suppression** – only local maxima are defined as edges. This step also thins the generated edges.

4. **Edge tracking by hysteresis** – the pixels found are then divided by the thresholds. If the pixel intensity is greater than the upper threshold, there is a certainty that there is an edge. In case the intensity level is below the lower threshold, it can be concluded that these are non-edges. The values, which lie in between these two thresholds are only a piece of an edge based on their connectivity. If they are connected to an upper threshold pixel, they are considered edges.

The final image after all stages are applied, can be seen in Figure 2.2.

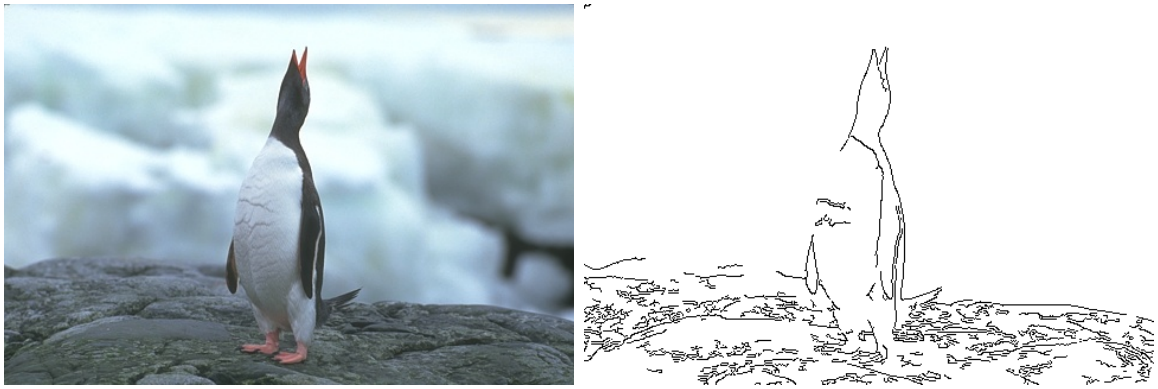

Figure 2.2: Canny edge detector applied on an image. As can be seen in the image it is dependent on changes of the pixel intensity and as such does not manage the detect the outline of the penguin [5].

2.4 Second Order Derivative Methods

Laplacian of Gaussian

As the name suggests, Laplacian of Gaussian (LoG) [21] is an image detector that consists of two main parts: the Laplacian and the Gaussian. A Laplacian operator [36] is an edge detector used to calculate the second derivatives of an image. This method monitors the rate at which the first derivatives change and afterwards calculates if the change in adjacent pixel values originated from an edge. Since the input image is represented as a set of discrete pixels, there is a need to have a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small matrices are shown in Equation 2.4. The values in these matrices can be inverted as it makes no difference in the end result.

$$D_x = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad D_y = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.4}$$

Before the Laplacian is applied, there is a need to apply the Gaussian filter to smooth the image. The Gaussian is defined as:

$$G(i,j) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{i^2+j^2}{\sigma^2})}, \tag{2.5}$$

where σ represents the standard deviation.

The final LoG detector is then defined as:

$$LoG = \frac{i^2 + j^2 - 2\sigma^2}{\sigma^4} e^{-\frac{1}{2}(\frac{i^2+j^2}{\sigma^2})}. \tag{2.6}$$

Difference of Gaussians

The Difference of Gaussians (DoG) [6] performs edge detection by performing a Gaussian blur on an image at a specified standard deviation. The resulting image is a blurred version of the original image. The detector then performs another blur, which is slightly weaker than the previous one, resulting in an image, which is less blurred than the previous image.

The final image is then calculated by replacing each pixel with the difference between the two blurred images and detecting when the values cross zero. The resulting zero crossings are focused at edges or areas of pixels that have some variation in their surrounding neighborhood.

Chapter 3

Convolutional Neural Networks and Their use in Edge Detection

There is a large number of various approaches covering the topic of edge detection. A modern approach to edge detection is to use artificial intelligence. The best results in the area were achieved using the concept of convolutional neural networks (CNN) that have the ability to learn thousands of features. The main difference between neural networks and CNN is that CNN expect the input to represent an image. This is a major change as it impacts the parameters that can be changed in the network.

CNNs [8] [27] utilize local spatial correlation by enforcing a local connectivity pattern between neurons located in adjacent layers. As such, the contributions of concealed units in layer m are from a subset of units in layer $m-1$ units that have spatially connected open fields.

There are many different approaches that can detect edges using CNN with the most successful ones being patch focused and end to end detection (HED, RCF).

3.1 Convolutional Neural Networks

Numerous differences between traditional neural networks and convolutional neural networks (CNN) can be observed. The focus is going to be on the layers that are specific to CNN and that are used in the implementation.

Convolutional Layer

Convolutional layer [27] extracts the features of an image while preserving the relationship between pixels in the image. Convolution is a mathematical operation between the given input and a filter. In most cases, the filter is represented by a $n \times n$ matrix. The definition of convolution can be seen in the Equation (3.1):

$$f(t) * g(t) = \int_{\infty}^{-\infty} f(\mathcal{T}) g(t - \mathcal{T}) \tag{3.1}$$

In the area of neural networks, this operation is performed by sliding the filter across the input image. When the filter overlaps with the pixels in the image, the above specified operation is called, resulting in a feature map.

There are many different parameters that can be specified. The most important ones include:

1. Filter – is represented by a $m \times n$ matrix. The depth of the filter is not specified as it corresponds to the depth of the image.
2. Stride – represents the number of pixels the filter is moved across the image. This majorly influences the output dimension of the image.
3. Padding – As can be see in Figure 3.1, after the operation is performed, the height and weight of the image is reduced. This is a problem as it puts a limit on the number of convolution layers if we are dealing with small images or have a large number of convolution layers. To overcome this limitation, padding is used. In order to retain the same size of the image, multiple columns and rows are added. Their value is in most cases 0, unless it is specified otherwise.

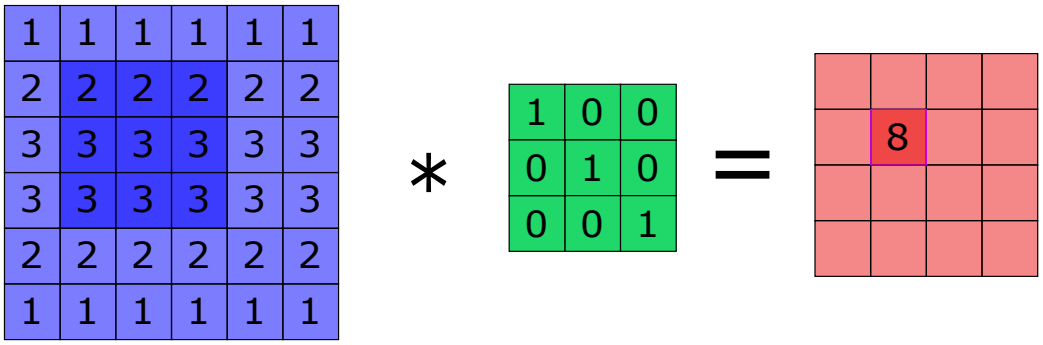

Figure 3.1: Convolution applied over image. No padding leads to reduction of the matrix size

Batch Normalization

Batch normalization [18] is a method that is used during the training of very deep neural networks. It standardizes the inputs to a layer for each mini-batch. This has a significant impact on the training of the networks.

Input Layer

The input layer [27] represents the main difference between standard neural network and CNN, as the input is expected to be an image. The image, which is represented by a tensor is given to the input layer. A tensor is defined as $number\_of\_images * image\_width * image\_height * color\_depth$. One of the distinct characteristics of input layers is that they are the first layer of the network and they have no set of weighted inputs. Their main function is to send the image to the next layer.

Pooling Layer

Pooling layers [22] are utilized to lessen the dimensions of feature maps. This is performed by decreasing the quantity of parameters that artificial intelligence needs to learn, leading to a decrease in the amount of calculation in the system.

The pooling layer condenses the features present in a neighbourhood of the feature map created by a convolution layer. In this way, further computations are performed on condensed features rather than highlighted features created by the convolution layer.

There are three main types of pooling layers:

1. Max Pooling – selects the maximum value from the selected neighborhood of the feature map covered by the filter. In this way, the output after max-pooling creates a feature map containing the most noticeable features of the previous feature map.
2. Average Pooling – as the name suggests, it creates the average value in the selected neighborhood.
3. Global Pooling – composes all values into a single value. In most cases, global pooling is combined with either average or max pooling.

Concatenation Layer

This layer takes a specified number of input tensors and concatenates them together. This is done by specifying an axis on which the operation should occur. An important prerequisite is that all input tensors must have the same dimensions and same kernel layer depth.

Activation Function

The activation function [26] determines the output of a neuron in a network depending on the given input. It depends on the weight of the neuron and is influenced by bias. Some of the widely used activation functions include for example sigmoid and ReLU.

- Sigmoid [26] – a sigmoid function is bounded by an upper limit of 1 and a lower limit of 0. Due to the existence of these limits, the optimum use of this function is often in neural networks where there is a need to predict the probability of an output. Sigmoid is defined in the Equation (3.2):

$$S(X) = \frac{e^x}{e^x + 1} \tag{3.2}$$

- ReLU [26] – a ReLU function is separated into two parts. The first part covers cases when x is lower than 0, which leads to the result of 0. The second part covers situations when x greater than 0, which results in the value always being positive. This can be seen in the Equation (3.3):

$$f(x) = \max(0, x) = \begin{cases} x_i, & x_i \geq 0 \\ 0, & x_i < 0. \end{cases} \tag{3.3}$$

Loss Functions

The loss function [1] specifies how the neural network penalizes the deviation between the predicted output and the actual output of the image during training. Common loss functions include cross entropy and pixel error.

1. Cross Entropy – measures the loss of a model whose output is a probability between 0 and 1. A very good model should have a very small value of the loss function. In an ideal case, the magnitude for a perfect model should approach values close to zero.

2. Pixel Error – this loss function measures the differences between output pixel values in an image. An advantage of this function is that it facilitates understanding of changes on a pixel level. However, there are also some drawbacks to it. The main issue is that the loss function scans the image in a pixel by pixel basis which degrades the quality of the results.

3.2 VGG16

The VGG16 [33] model is one of the most popular models in the category of convolutional neural networks. Most of the models that are focused on edge detection use this model as a baseline from which they expand further.

The input into the model is of fixed size 224×224 RGB image. The image is then passed into the convolutional layers, that use filters of size 3×3. The convolutional layers use a wide range of filter from 64 to 512. The pooling is done by 5 max-pooling layers. Max-pooling is performed over a 2×2 pixel window with stride 2.

Three fully-connected layers follow after the last pooling layers. The first two layers have 4096 channels and the third layer performs 1000 way classification and contains 1000 channels. The final layer is the soft-max layer that produces the final result. All hidden layers use the ReLU activation function. This architecture can be seen in Figure 3.2.

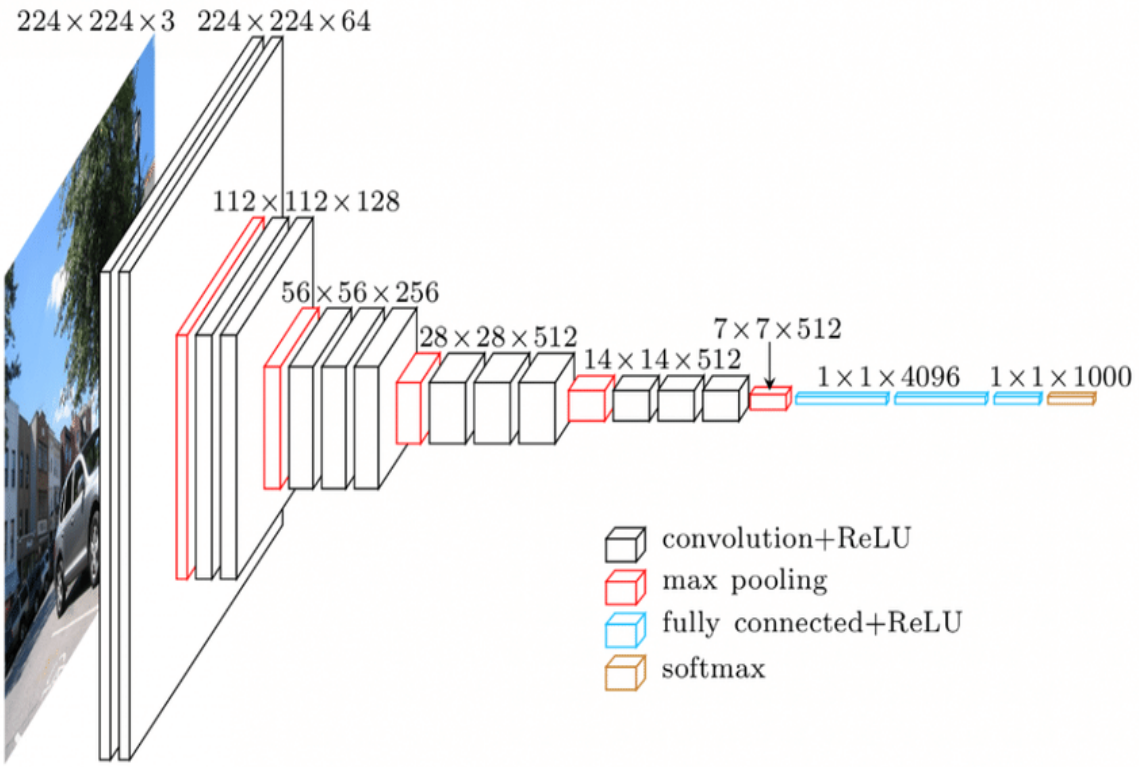

Figure 3.2: VGG 16 architecture [19].

3.3 Holistically-Nested Edge Detection

Holistically-Nested Edge Detection [35] (HED) is one of the most successful convolutional neural networks for edge detection. As many other networks, its core is based on VGG16 3.2. The main modification is that after the last pooling layers only the convolutional layers are kept and the fully connected layers are cut off. Before every pooling layer, a side branch is added where the result is saved. This produces 5 side outputs and all of them are attached to a sigmoid activation function and a cross entropy loss. All side outputs are upscaled to the original size. After all 5 outputs are produced, they are combined into one image. This architecture can be seen in Figure 3.3

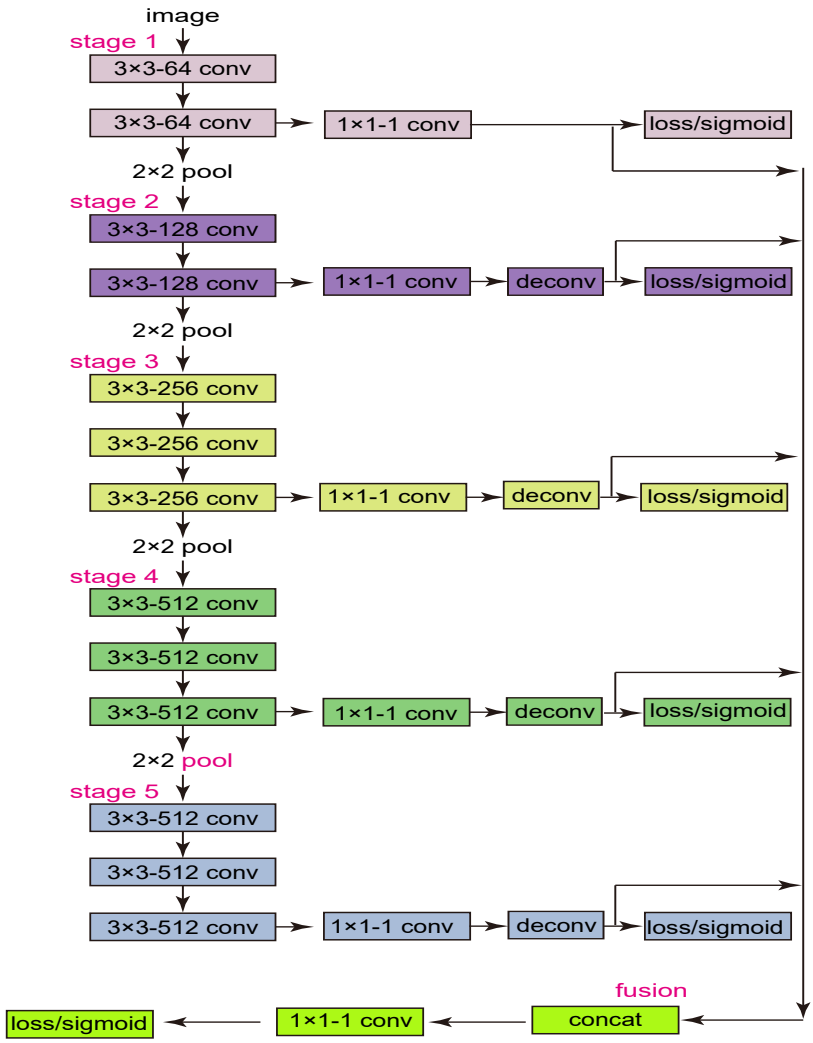

Figure 3.3: Illustration of the HED architecture [24]. After each pooling layer, a branch that captures the side output is attached. These side outputs are then combined into one image.

3.4 Richer Convolution Features

The Richer Convolution Features (RCF) [24] model is inspired by HED model. It adapts the HED architecture by enriching the side outputs. The first stage takes an input image of fixed size $n \times n$, augments it with user predefined modifications and sends it to the hidden layers. The hidden layers are mostly composed of convolution layers. The main difference from HED is that after each convolution layer, a side output is generated. Afterwards, the side output goes into two directions. The first one follows the HED architecture and it is given to the subsequent convolutional layers. The second one takes the image into a 1×1 kernel size convolution layer with 21 filters. Afterwards, all these side outputs in a given stage are combined into one image. Then the combined side outputs are upsampled and evaluated on a cross-entropy loss function. This process repeats for every stage of the architecture which produces 5 side outputs. Stages are separated by pooling layer which halves the size of the image. The subsequent convolution layer after the pooling layer doubles the filter size of the layer. Only stages 4 and 5 do not increase the filter size. After the above mentioned process is finished, the side outputs are stored and fused (concatenated) into one image. The complete architecture can be seen in Figure 3.4.

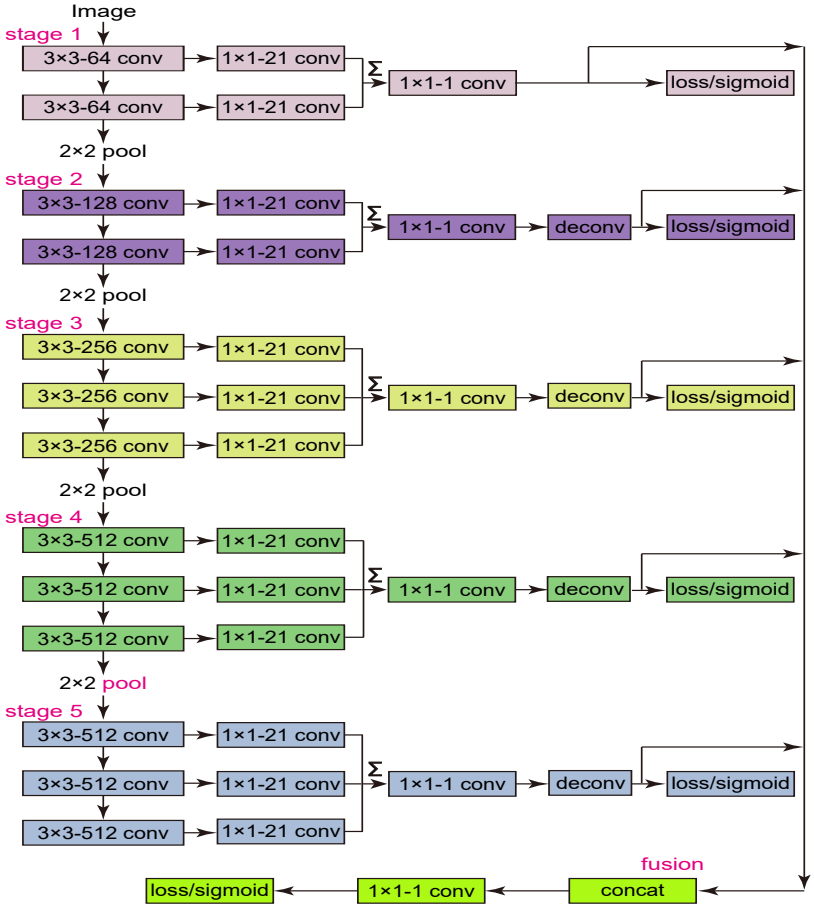

Figure 3.4: RCF architecture [24] is similar to the HED architecture except for the fact that side outputs are saved after every convolution layer. Afterwards, they are combined with other side outputs in their stage to create one image. This produces 5 side outputs that are combined into one image.

Many different architectures were tested with the best result being with a ResNet101 backend. However, the result was achieved at the cost of a significant performance decrease.

3.5 Deep Edge

The Deep Edge [7] model, compared to other CNN models, has a specific point in the data augmentation. It selects candidate points using the Canny Edge detector that is explained in section 2.3. Afterwards, a set of patches is extracted with the candidate points being in the extracted patch. The KNet [10] neural networks is used. KNet consists of 5 convolutional layers and 3 fully connected layers. Only the convolutional layers are used for the purpose of edge detection. Afterwards, there is a regression and classification branch. The comparison shows that the regression branch produces worse results than the classification branch. The model representation 3.5 shows that after the candidate edges are extracted, they are scaled into 4 different sizes and given to the network to process. The special characteristic of this model is that 3 types of pooling layers are used: max, average, and center pooling. The best result is achieved from combination of all pooling types. After the edge map is predicted, a threshold with the probability of 0.5 is applied to produce the final edge map. From the pooling layers, a surprising discovery is that max pooling, which is the most used type in different edge detectors, produced the lowest performance.

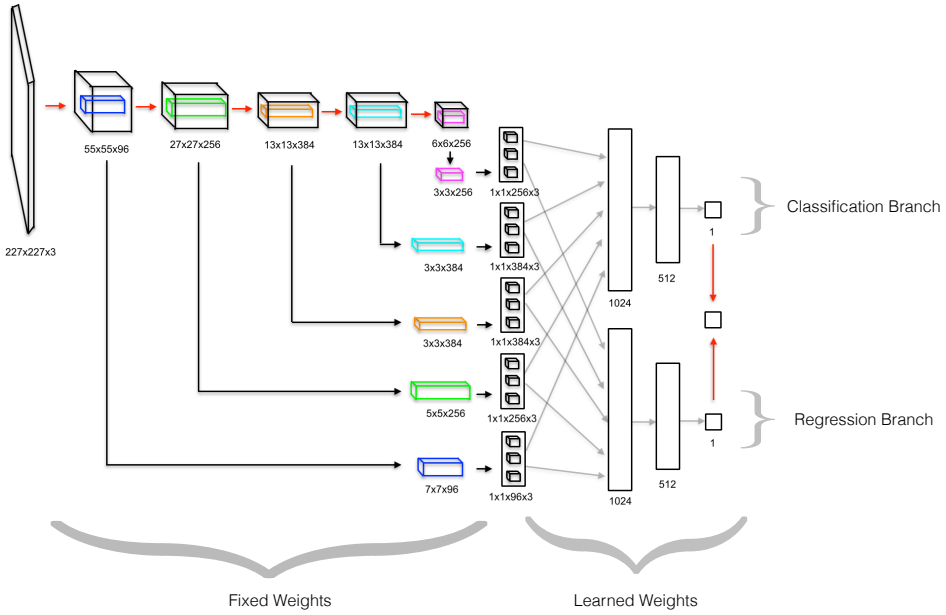

Figure 3.5: Deep edge architecture [7]. Candidate points are extracted from the canny edge model, then 4 patches at different scales are sent into the network.

3.6 RefineContourNet

All previously mentioned architectures rely heavily on down sampling the image with operations like pooling or convolution and then later upsampling the down sized images into their original size. RefineNet [23] proposes an architecture that is based on ResNet101 [15]

and is able to combine high-level segmentation with low-level features to produce high-resolution segmentation map. RefineContourNet [20] takes this architecture and modifies it. Three different block architectures are taken from the original RefineNet and those are Residual Convolution Unit (RCU), Multi-Resolution Fusion (MRF) and ChainedResidual Pooling (CRP). The RCU block is composed of two convolutional layers that enrich the network with more parameters. The MRF block takes 2 inputs, adjusts the dimensions by performing convolution and adjusts the size of the image by upsampling the smaller one and combining the 2 images. The CRP block consists of two pooling layers that help to extract more details from the image. After each pooling layer, a convolutional layer is applied. Afterwards, the results are combined with the original input. The architecture of these blocks can be seen in Figure 3.6.

The final architecture then arranges the blocks in the ResNet101 style, which from a certain point of view, is similar to RCF. There are 4 stages that produce side outputs. The last layer in the architecture applies the block operations on the image and then proceeds to give the image to the previous side output, where it is combined with the side output generated in the layer. This is repeated until the first stage is reached. The final architecture can be seen in Figure 3.6.

A major drawback of this model is that it uses ResNet101 as the backbone and even with pre-trained weights, it impacts the speed of the network due to the size of the network.

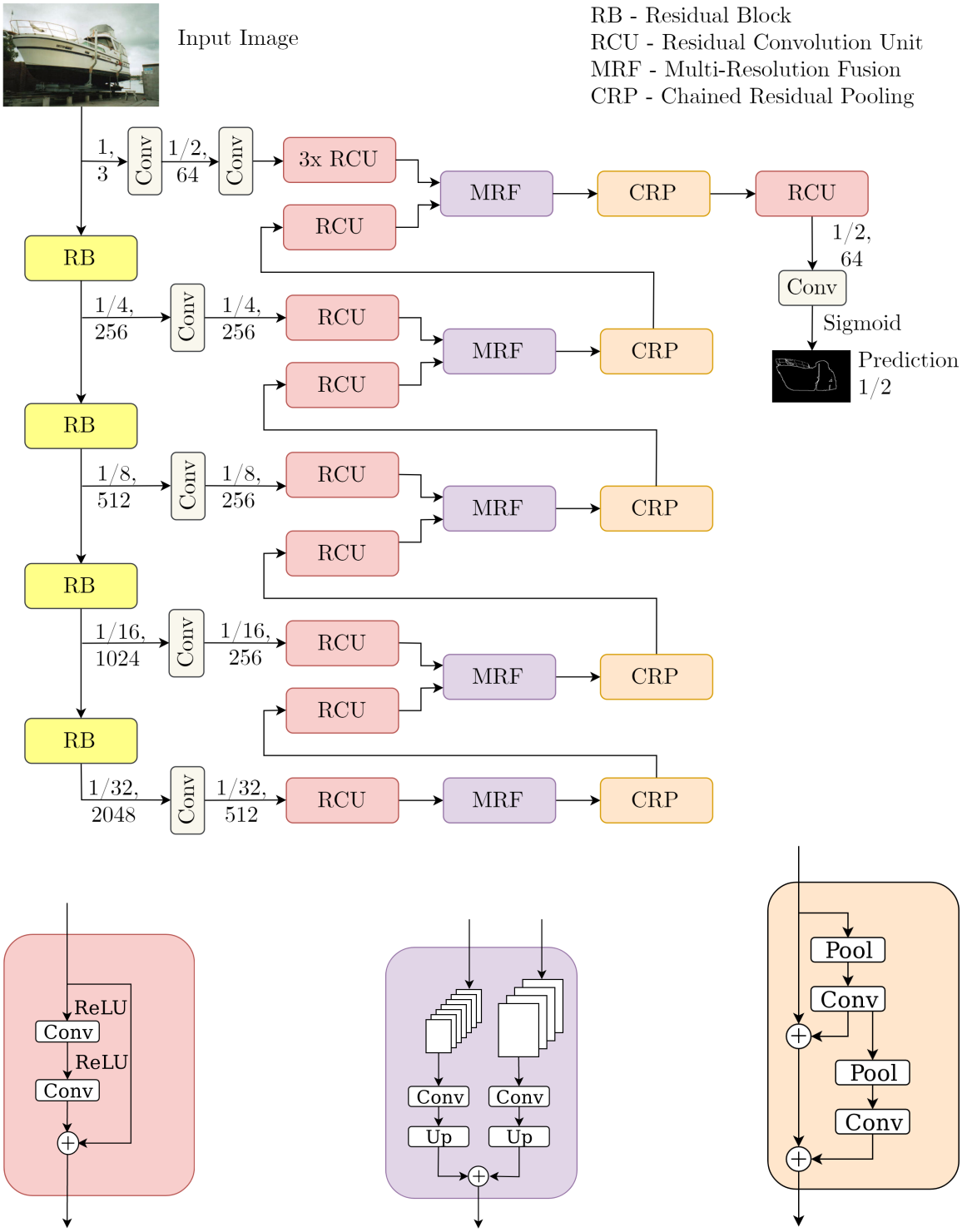

Figure 3.6: RefineContourNet architecture [20]. The whole RefineContourNet architecture with all blocks is at the top. RB represents the refine block from ResNet101. In the bottom part, the architecture of the block can be seen. Left represents the RCU block, middle MRF block and right is CRP block.

Chapter 4

Quality Evaluation Metrics

Classification is a very complex procedure and an essential component is assessing the quality of the results. Therefore, it is necessary to define several frequently occurring categories that are commonly used. The relevant terms include true positives, true negatives, false positives, and false negatives [31]. These terms help to facilitate a comparison of the various outputs of a detector.

- True positive (tp) – the model correctly predicts a pixel as an edge
- True negative (tn) – the model correctly predicts a pixel that is not an edge
- False positive (fp) – the model predicts a pixel as an edge, when it is not an edge
- False negative (fn) – the model predicts a pixel as not an edge, when it is an edge

The measure that is used for evaluation is the F-measure. It consists of 2 values that are precision and recall.

Precision [31] represents the fraction of relevant instances among the retrieved instances and is represented in the following Equation (4.1):

$$Precision = \frac{tp}{tp + fn} \tag{4.1}$$

Recall [31] represents the fraction of the total amount of relevant instances that were actually retrieved and is defined in Equation (4.2):

$$Recall = \frac{tp}{tp + fp} \tag{4.2}$$

Both the precision and recall represent an essential input in calculating the F-measure. F-measure reaches its best value at 1 and worst at 0. The calculation of the F-measure value can be seen in the Equation (4.3). R represents the recall value and P represent the value of the precision.

$$F_{measure} = \frac{2PR}{P + R} \tag{4.3}$$

Given an edge probability map, a threshold is needed to produce the binary edge map. The threshold can be set using three alternative approaches:

1. ODS [5] – a fixed threshold is set for every picture in the training set

2. OIS [5] – every picture has its own threshold that is used for evaluation

3. AP [5] – average precision (AP), is a metric used for evaluation of classification detectors. It defines the average value of the maximal values for precision for a value of recall. First, there is need for interpolation of the precision-recall curve. Afterwards, the maximum value for every recall is averaged as can be seen in Figure 4.1.

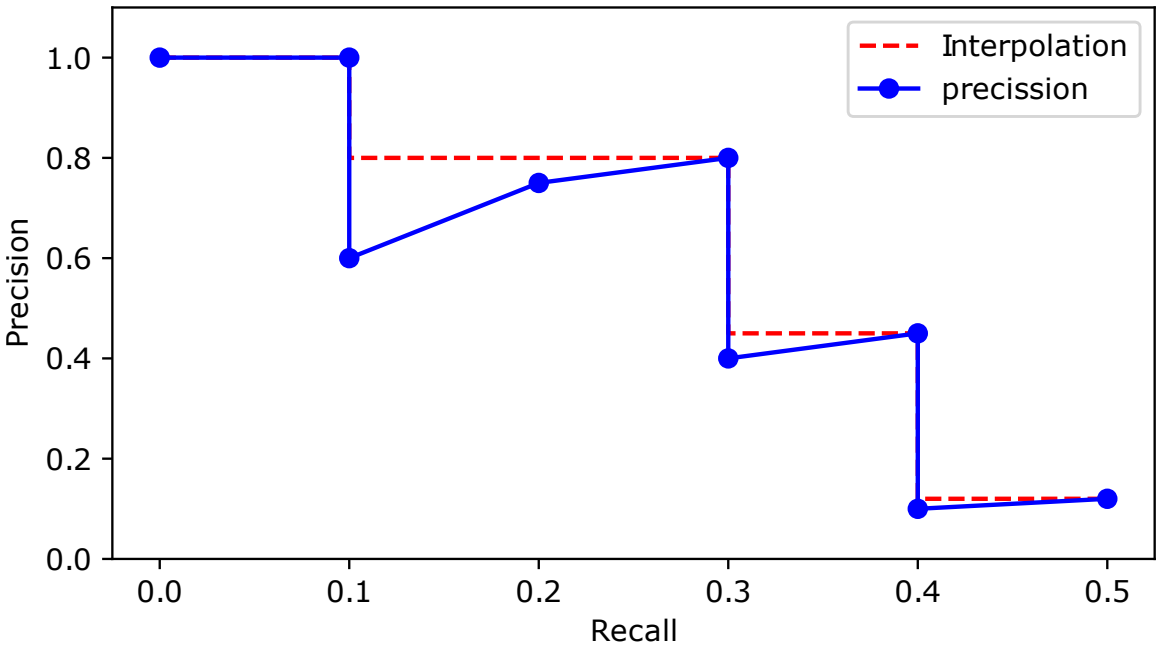

Figure 4.1: Precision-recall example with interpolated AP curve.

In order to apply the terms defined in this chapter in the context of this thesis, the toolkit from Structured Edge Detection Toolbox [12] is used. It defines a margin distance between the pixels in the actual ground truth and the predicted edge map. This ensures that some pixels are counted as edge even if location is not completely identical.

4.1 Non-Maxima Suppression

Non-maxima suppression [9] is a very important part of edge detection. After its application on an image, it can thin the edges, thus producing better results. This solves the problem that most detectors have with producing edges that are thicker than the original expected edges in the ground truth.

4.2 Cross-Validation Definition

Cross-Validation [30] is a method that is used to evaluate the model that is trained on one dataset to see how it would perform on another dataset. This helps to generalize the model quality on different types of images. It also helps to identify weak and strong points in the classification.

There are many approaches to cross validation such as Hold-out, Resubstitution, k-fold cross-validation and many more. Each of these has its own pros and cons. The focus will be

on Hold-out validation. This approach separates the data into independent subsets, which prevents overfitting but also reduces the dataset size as the images used for testing and training are independent.

Chapter 5

Existing Datasets

5.1 BSDS500

The Berkeley Segmentation Dataset and Benchmark (BSDS500) [5] is a dataset which is focused on edge detection. It is the primary dataset used for all benchmarks and evaluations. The dataset consists of 500 images of which 300 are supposed to be used for validation and training and the remaining 200 are for testing purposes. All images were manually annotated by humans and each image is segmented by five different subjects on average to create the ground truth. As can be seen in Figure 5.1, there are three different types of ground truth all hand-drawn by humans. The difference between what some people count as edges can be perfectly seen when comparing the images.

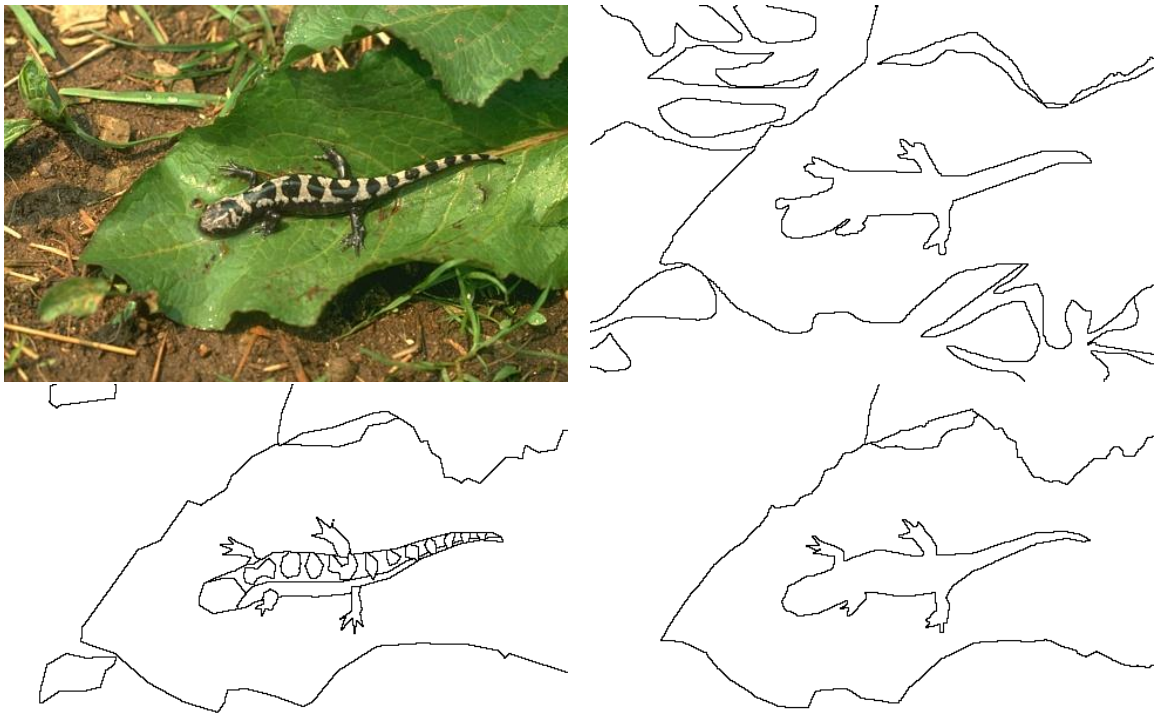

Figure 5.1: Examples from the BSDS500 dataset [5]. Upper left represents the original image, the other three images are the hand drawn ground truth.

5.2 PASCAL VOC

The Pascal Visual Objects Classes (VOC) [13] dataset is mainly used for object detection. The VOC dataset consists of annotated photographs collected from websites, the main one being flickr website. A new dataset with ground truth annotations was released between 2005-2012, with the last update containing 20 different classes with 11.530 annotated images. Examples of these annotations and original images can be seen in Figure 5.2.

Figure 5.2: PASCAL VOC [13] examples. This dataset is focused on object detection

5.3 NYU-Depth V2

The NYU-Depth V2 [25] data set is comprised of video sequences from a variety of indoor scenes. It consists of 1.449 RGB images and depth images. The ground truth that represents the objects in an image is modified and edge maps are extracted and can be seen in Figure 5.3.

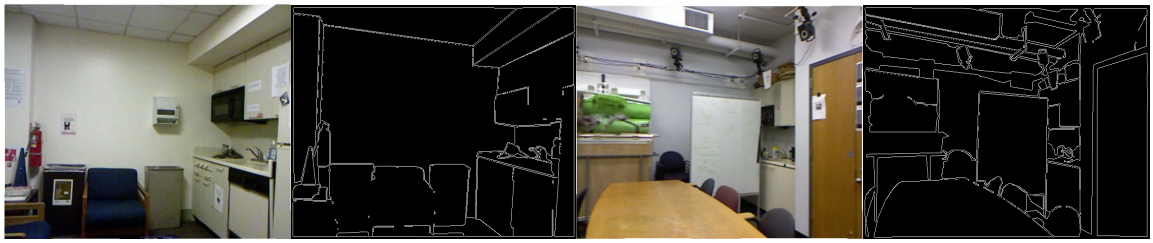

Figure 5.3: NYU-Depth V2 [25] dataset with modified depth ground truth to edge maps.

Chapter 6

Design and Implementation of the Edge Detector

This chapter is focused on analyzing existing CNN architectures, extracting the strong points of each architecture and proposing an architecture for new detection. It also mentions the implementation tools.

6.1 Implementation Tools

Keras

Keras [4] is a neural networks library written in Python. It is a part of Tensorflow. Keras is comprised of various implementations of commonly used neural network parts such as layers, activation functions, loss functions, optimizers, and many other tools that operate with images.

Compared to some of the other standard neural networks, it supports other additional layers used in convolution neural networks such as pooling, convolution and dropout.

Google Colab

The main environment used for development is Google Colab [3]. Google Colab is a free cloud service with the option of GPU acceleration. The neural net is implemented using Python version 3.7.1. Responsible for the training of the model is the framework tensorflow-gpu version 1.15, in which Keras is included. The environment provides many different types of graphic cards that are randomly assigned and can be used for training purpose. It also provides the option to use Jupyter notebooks, but that option is not used as loading a lot of external dependencies is required.

GPU Acceleration

GPU Acceleration [17] is the use of a graphics processing unit (GPU) along with a computer processing unit (CPU) in order to accelerate highly complicated calculations. As a result, while sequential calculations are performed in the CPU, highly complicated calculations are computed in parallel in the GPU.

Matlab

Matlab (Matrix Laboratory) [2] is a programming environment for algorithm development, numerical computation visualization, and data analysis. Matlab is used to evaluate the performance of the implemented neural network using Structured Edge Detection Toolbox [12]. The publicly available implementation along with the needed toolkit [12] [37] [11] is used to evaluate the detector as the same code is used in the official evaluation provided by the other models.

6.2 Analysis of Existing Edge Detectors

One of the fastest ways to detect edges is using Canny Edge. It provides an algorithmic solution based on mathematical equations. Even though its execution time is fast, it has problems with the quality of the identified images as explained in chapter 2.3. There is also a need to set the double threshold and that proves to be difficult when working with a variety of different images where multiple configurations are needed.

Compared to algorithmic approaches, neural networks produce edge maps with higher quality. Moreover, the decrease in speed observed when using these approaches can be negligible in certain architectures. The most notable results are achieved by using the RCF and RCN architectures. RCF is based on HED and it modifies the side outputs for more precise results. RCF focuses on the point that all layers produce a side output that can be used for edge detection. This proposes the idea that combining multiple images produces a higher quality of edge maps than a single edge map at the end of the neural networks even if the other generated outputs are of lower quality. The side outputs pick up cues that can be missing in the final image and thus their combination with other output increases the chance that they appear in the final image and are not lost in some of the hidden layers.

The Deep Edge model stands out from other models due to its data augmentation that applies the Canny edge algorithm to select candidate edge points. This significantly decreases the training phase as the model only needs 50 epochs. The only problem is that it is too dependent on the Canny edge algorithm, which is not perfect and suffers many limitations. Although there are some modifications possible to the original Canny edge detector, they do not increase the quality enough for this to be a consideration.

Another option for modification is proposed in the pooling layer since the Deep Edge model experimented with using average pooling instead of max pooling with good results. However, this proves to be useful only if the input image is segmented into multiple candidate edge points. This point was proven in the HED architecture where this was explored in the original paper.

Pooling method	Model	ODS
Average pooling	HED [35]	.741
	Deep Edge [7]	.730
Max pooling	HED [35]	.782
	Deep Edge [7]	.690

Table 6.1: Measurements from the original HED and Deep edge model. Average pooling only improved the quality in the Deep Edge model. All other models used max pooling to get better results.

The RCN model proposed multi-level upsampling with concatenation of images along the way. This increased the quality of the final image. It also proposed three types of blocks, which if looked at from a certain perspective, encapsulate the idea of RCF as they add multiple side outputs together. These blocks increase the number of learnable parameters in the networks. They also serve as upsampling layers.

6.3 Implementation of the Edge Detector

In order to completely implement a functioning edge detector, several essential building blocks are needed. First, the data from the datasets needs to be processed and an architecture for the networks must be designed. Afterwards, the models need to be evaluated using the same metrics as other existing models, which is necessary for comparison purposes.

The first step is to convert all the labels from the different datasets to a usable ground-truth. This format is chosen to be the ground-truth that occurs in the BSDS500 dataset. The NYU-depth dataset contains only depth ground truth in which the only highlighted parts are objects not edges. As such, although this dataset cannot be used for training like the BSDS500, it can be used with the same purpose as PASCAl-VOC for cross-validation.

To extract the edges from the NYU dataset, other methods like RCF 3.4 proposed to use a simple edge detector as the image is separated into highlighted parts and thus even algorithmic detector like Canny Edge, that suffers from lighting conditions, can produce high quality edges that can be used as ground truth. The result can be seen in Figure 6.1.

Figure 6.1: Left original image, middle object highlighted ground truth, left transformed edge ground truth.

For the parsing of the data, `cv2` is used to read each image from the datasets. Each image and ground truth is resized to the same size that corresponds to the required size that the neural network expects. One of the few problems with `cv2` is that it uses BGR color model instead of the usual RGB that is more commonly used in images. Due to this issue, a conversion is needed. A very important step is to augment the data for later use. This enhances the results as it changes the original images and therefore in a way increases the existing dataset. Each augmentation has a chance of occurring with the possibility of multiple effects being applied to one image. Augmentation is used for on the BSDS500 dataset due to the low number of training images.

Several augmentations were applied during testing. However, not all of them had a positive effect. It is important to note that although multiple effects can be applied to the image, some are mutually exclusive. The augmentations used are listed below:

- **Gaussian blur** – applying this is used to reduce image noise and detail.

- **Gamma correction** – changes the light conditions by making light areas darker and dark areas lighter.
- **Brightness change** – changes the light conditions. If increased, brightness of all areas is lighter, if decreased, all areas are darker.
- **Gaussian noise** – adds noise to the image. This is specially done after the detector had problems detecting edges in images that contained noise.
- **Rotation** – rotate the image along with the ground truth.

Design of the Edge Detector

One of the findings from section 6.2 is that although side outputs increase the quality of the image, the choice of side outputs is materially important. After comparing HED and RCF, the result that is first observed is that the first side output in HED produced a much lower quality,than the side output in RCF. The difference is in the concatenation of the images from all convolution layers instead of just one image. After a comparison with the other layers, it is observed that an increase in convolution layers before generating the image should improve the quality of the first layer. As such, the increase should also be applied to the second and third layer since the best results are observed on the fourth layer. The fifth layer produces edge with lower quality, however, it highlights part of the edges, which are usually also captured by the other layers. Therefore, the overall quality of final image should not be impacted severely as the issues linked to the fifth layer are mediated by the functionality of layers one to four.

The other important part is the concatenation of the side outputs. RCF proved that rather than ignoring the generated side outputs, the concatenation of unused convolution results improved the quality. As such, there should be a consideration to link the concatenated side outputs with other layers to produce more precise results instead of just combining them in separate layers.

A very important factor was deciding the learning rate of the model. In the first convolution layers, there is no problem even with higher learning rates (i.e. $1e-2$, $1e-1$). However, in the lower levels in architectures like HED and RCF, where the number of filters is increased from 64 to 512, a high learning rates stops the detector from picking up small details and produces side outputs with no values as can be seen in Figure 6.2.

According to these identified points, there are 2 architectures that are designed and evaluated. One is based on RCF and the other one is inspired by RCN.

RCF Based Achitecture

The first architecture is based on RCF and tries to propose modifications that would improve the quality. In the original RCF, the side outputs were combined into one final image as the final step and they were all combined at once. The proposed change is to slowly add them to the previous layer, introducing a dependency on the layers. This is performed on the basis that the results from the last stage should increase the quality of the previous stage. This dependency is introduced between all stages. All side outputs that are generated this way are still evaluated by the loss function and added to a final image. The 5 original outputs are still given to a convolution layer with a sigmoid activation that evaluates the network with the loss function. The reason that the concatenated outputs are not evaluated

Figure 6.2: Example of how a high learning rates stops the detector from identifying edges under certain conditions (i.e. large filters). Left is lower number of filter while right is higher number of filters and both were trained under the same learning rate.

is that the lower levels take a fairly long time to reach a sufficient quality and evaluating the concatenated results proved to be very difficult.

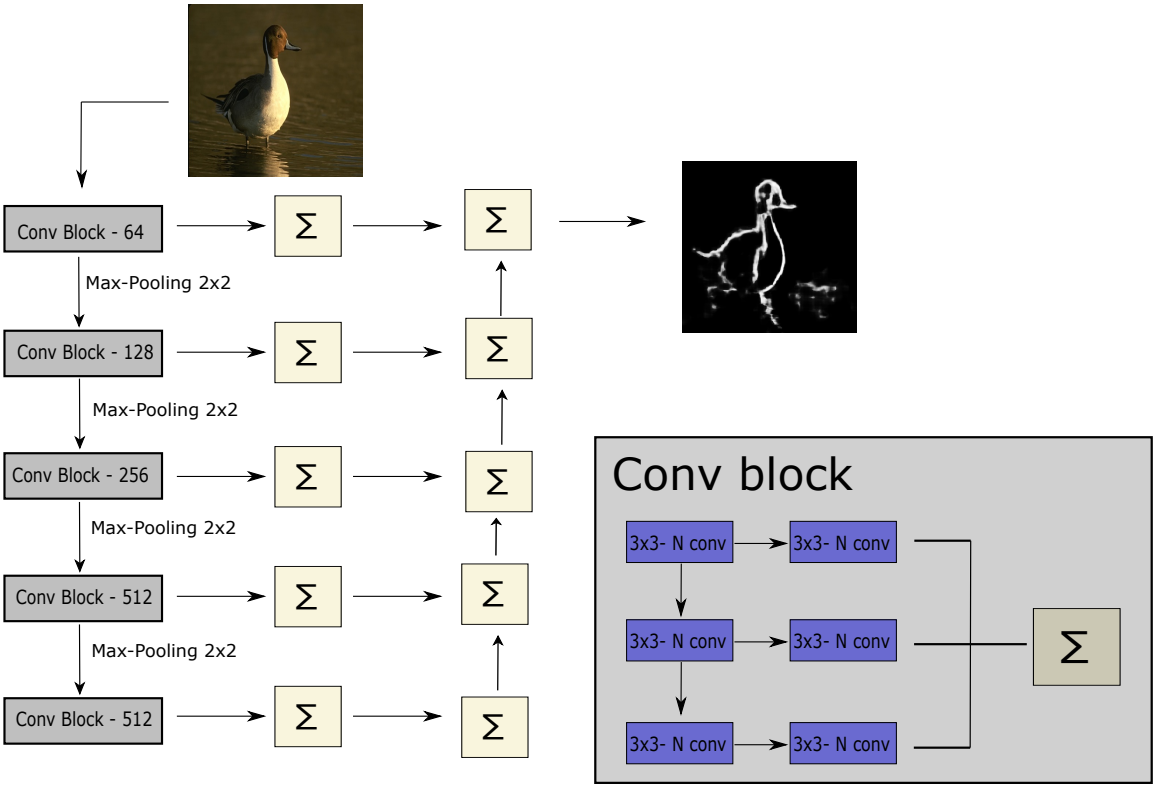

Figure 6.3: RCF based architecture.

RefineContourNet Based Architecture

The second architecture combines RCF and RCN. It takes the results found from RCF and combines them with the advantages proposed in RCN. The 5th layer from RCF is removed as the quality that it produced is too low, and in this type of architecture that relies on previous results, it would have a severe impact. The remaining 4 layers first produce the side outputs in the standard RCF form. Afterwards, for each stage, the RCU, MRF, CRP blocks are used that originate from the RefineNet architecture. An example of these block can be seen in 6.5. The last stage takes the produced side output, applies the RCU, MRF and CRP blocks on it and gives it to the previous layer. That layer repeats the process with the only difference being that the MRF block takes the input that was gotten from the next layer and combines it with the output it currently has in the MRF block. This is performed until the first layer is reached that generates the final image. The advantages of this approach are multiple. As it produces only 1 output image, the input requirements are lower than the original RCF architecture that needed multiple ground truths for validation purposes and evaluation of the side output results.

Configuration of the Detector while Training and Testing

To configure parameters for training and testing, the file **config.json** is used. There are many parameters that can be configured in this file that change the training process of the network.

One of the most important ones are:

- **Batch size** – defines the number of images that will be given to the network at a single time.
- **Epochs** – the number of epochs that the model will take to train.
- **Learning rate** – defines the value that adjusts the weights of our network.
- **Dataset path** – expects a file that contains the location of images and their corresponding ground truth. This parameter can be set for training directory and validation directory.
- **Data augmentation** – disable/enable data augmentation.

The training phase is a very cost-heavy process and is performed using Google Colab which provides the option of GPU acceleration that increases the execution speed significantly. One part that is important and is not included in the configuration file are the settings of the Keras callback functions. The callback functions used include `ReduceLROnPlateau`, `ModelCheckpoint`, `LearningRateScheduler` and `EarlyStopping`.

- **ReduceLROnPlateau** – reduces the learning rate when the loss function stagnates. In this callback function, it is defined how many epochs the function has to wait before the changes of the learning rate are applied. The learning rate changes by a predefined factor.
- **ModelCheckpoint** saves the best result of the model according to the loss function. It is defined how often the save occurs and what to save as there is an option to only save the model weights.

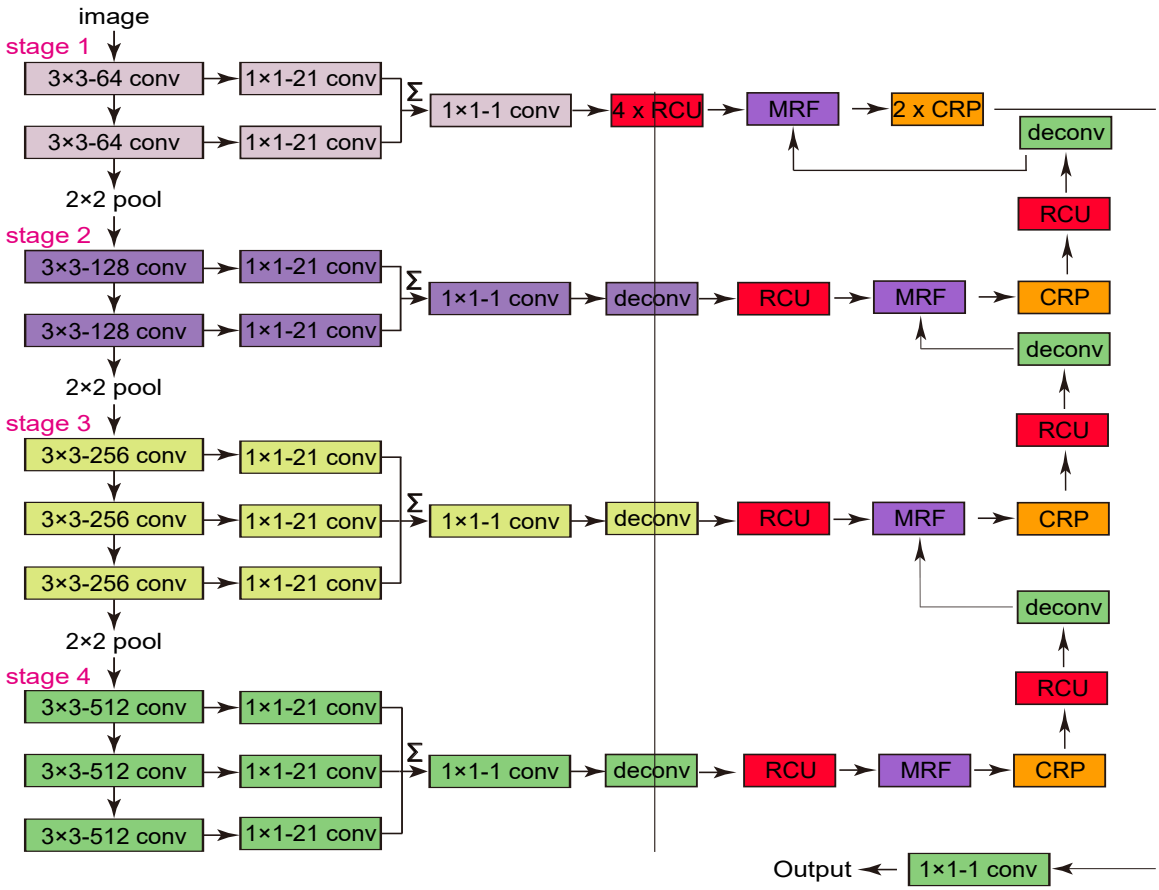

Figure 6.4: Architecture based on RCF and RCN. 4 stages generate side outputs.

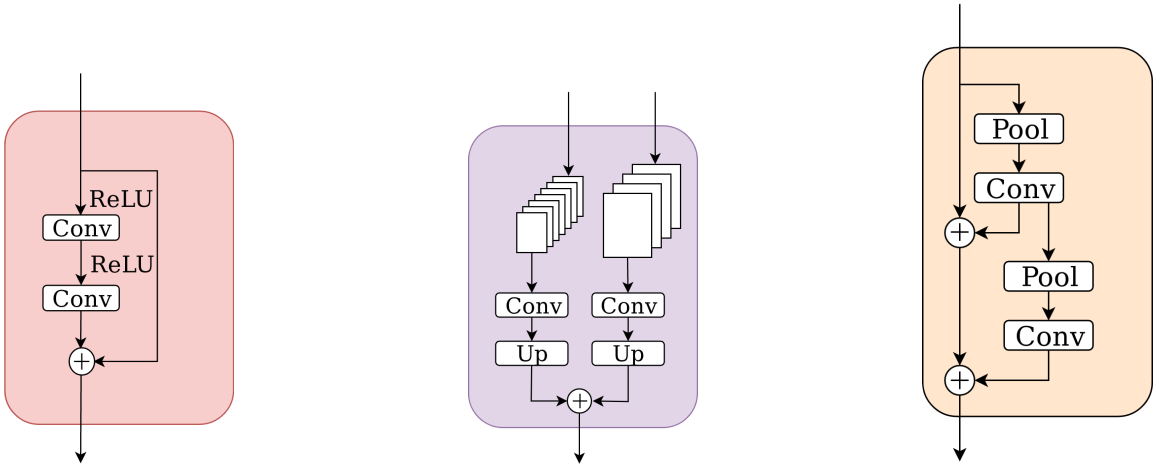

Figure 6.5: RCU, MRF and CRP blocks taken from RefineNet countour [20].

- **LearningRateScheduler** is similar to ReduceLROnPlateau as it impacts the learning rate with the difference that ReduceLROnPlateau reduces the learning rate when no improvements are made while this function changes the learning rate after a certain number of epochs is reached. The learning rate is user-defined.

- **EarlyStopping** occurs if the model did not have any improvements in a certain period. This value should always be defined much higher than ReduceLROnPlateau or LearningRateScheduler because if the learning rate is too small, it takes the model a long time to show improvements.

Chapter 7

Experiments and Evaluation

7.1 Experiments

The experiments are performed by measuring the performance of the designed edge detectors using the ODS, OIS and AP metrics. For both proposed architectures, the loss function is observed and the output results are analyzed. This helps to optimize some of the hyper-parameters (i.e. learning rate). Afterwards, the models are compared using cross-validation with the datasets mentioned in section 5. The negative results are also analyzed and solutions to improve them are proposed. The proposed models are refered to as RCF (For RCF based architecture) and RCN (For RCF and RCN based architecture).

As mentioned in section 6.3, the choice of the correct learning rate plays an essential role. In many of the studied models, the learning rate is set to $1e-6$ or even lower. As a result of this, the edge detector is fairly slow when training. The approach attempted is to slowly increase the learning rate and see if it produces desirable results even with a faster learning rate.

On the RCF architecture, the slow learning rate greatly impacted the first side output. As there are too few parameters to learn and a combination of multiple images is used, it took a fairly long time for this layer to pick up edges. Upon an increase of the learning rate, it managed to pick up more edges but at the cost of subsequent layers producing worse results. This issue could be solved by adding a scheduler. The addition proved to be helpful as the learning rate could be set high for the first 50 epochs, which helped the first layer detection. Afterwards, the learning rate was reduced to a lower level to optimize other layers.

The RCN architecture does not have problems even with higher learning rates as the block taken from RCN provided a larger number of parameters to learn in the early stages. The high learning rate helped the detector to quickly identify edges but with minimal improvements after a certain number of epochs was reached. To solve this, a lower learning rate was applied using the keras in-build callback function ReduceLROnPlateau. If the quality did not improve for 50 epochs, then the learning rate was multiplied by a factor of 0.2. This was repeated until the minimal set learning rate is reached (1e-7).

Another important aspect is the choice of the correct loss function for evaluation. There were 2 types of loss functions that were considered and experimented with: cross-entropy loss and pixel error. The best results were obtained from the cross-entropy loss and the detectors that were training using this loss function produced edges of higher quality. The main problem with the pixel error was that it produced edge maps where the object outlines

could be identified but the edges were not undisturbed. The cross entropy loss produced smooth edges in comparison that can be seen in 7.1.

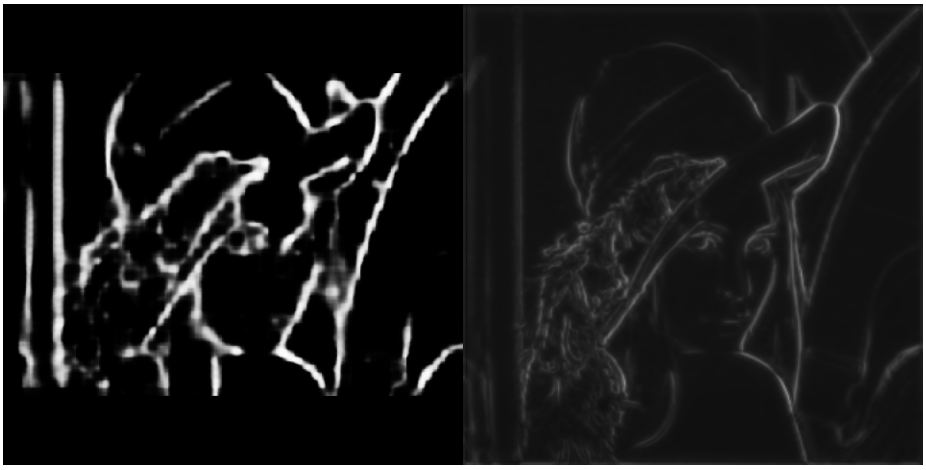

Figure 7.1: Difference between the quality of the loss functions. Left is pixel error, right is cross-entropy. Both were used and experimented on using the proposed RCN structure.

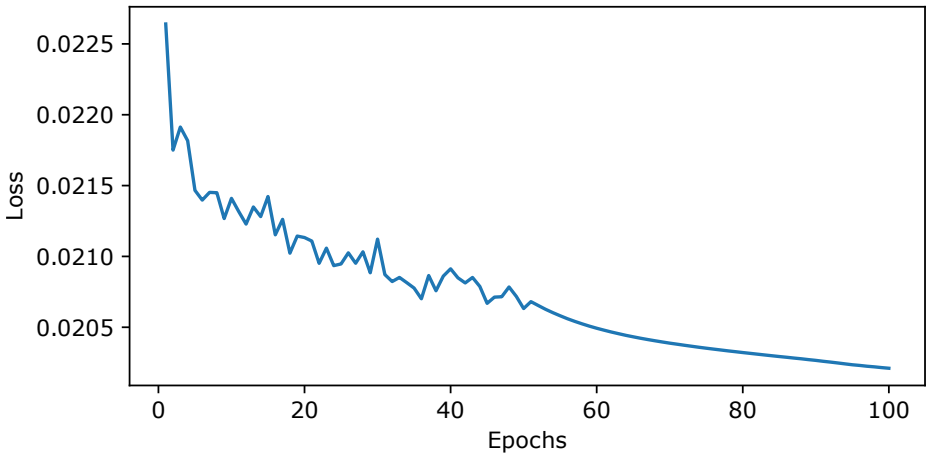

Figure 7.2: Cross-entropy loss function on the RCF model. Only the first 100 epochs are displayed because after them in improved in very small amounts.

Figure 7.2 displays how the loss function changed while training the model. Only the first 100 epochs are displayed as the loss function did not change significantly for the rest of the training. The reason the loss function does not change very drastically is the result of the set learning rate. During the first epoch the loss function started around the value of 3 but the end result 0.0225 at the end.

7.2 Quality Performance

This chapter is focused on evaluating the detectors with the best quality. When evaluating using the ODS, OIS and AP metric on the BSDS500 dataset, the recommended split was used (200 test images). All images are resized to correspond to the ground truth in

BSDS500. This produces slight problems when evaluating the results as there can be some issues when resizing the images. Despite that, this was also performed in other works and the quality drop is expected to be negligible.

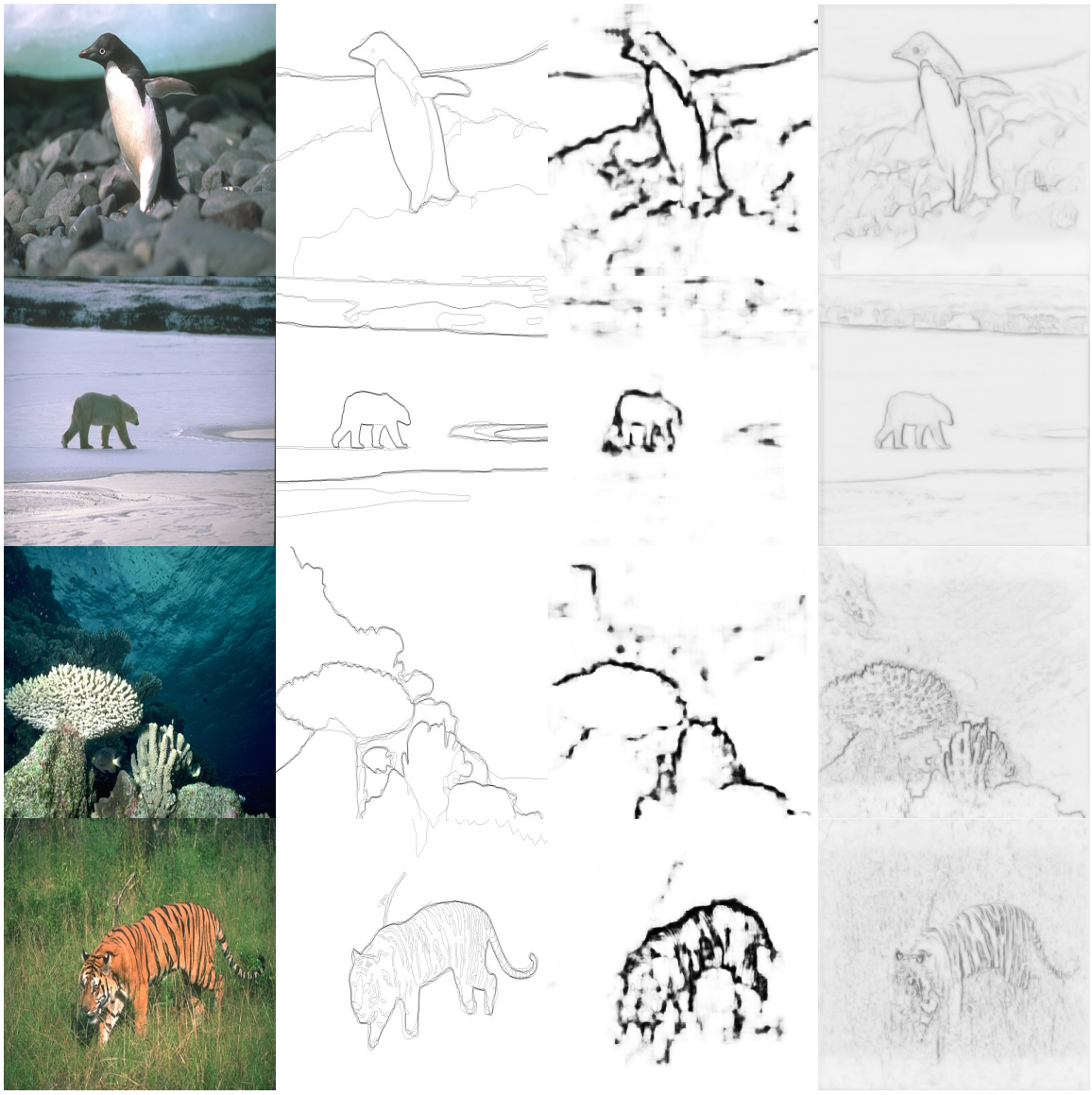

Figure 7.3: Results of the edge detectors, from left original image, ground truth, Own RCF based model, Own RCN based model.

The Figure 7.3 represents the results over the BSDS500 dataset. The RCN based model model produces more complex edges but suffers from limitations which can be seen in the last row of the image where the main outline of the ground truth is present in the image but there are many other edges although not as sharp when compared to the tiger seen in the image. The RCF based detector performs better from this standpoint as it does not detect as many edges as RCN based but at the cost of the quality of the detected edges. The best example of this behaviour is in the last row where it identifies the outline a stripes of the tiger but it is hard to identify what object it represents when looking at it with a human eye.

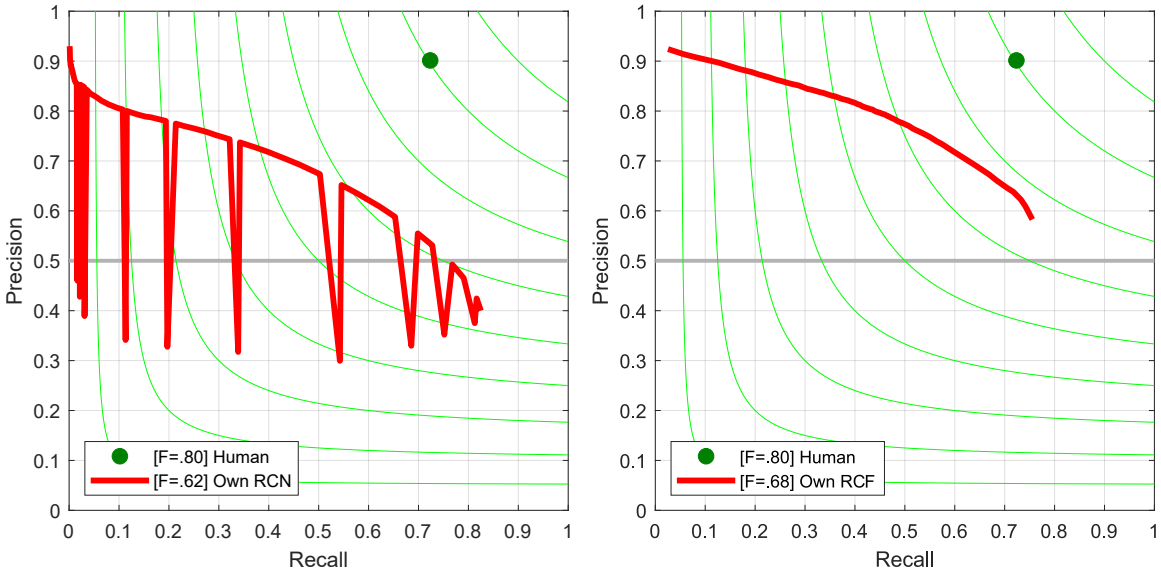

Figure 7.4: Precision recall curve for RCN (left) and RCF (right) model.

The Graph 7.4 represent the precision-recall curve that is used to evaluate the ODS, OIS and AP metrics. The results can be seen in Table 7.1 with the comparison of other methods.

Method	ODS	OIS	AP
Human	.80	.80	-
Deep Edge [7]	.753	.772	.807
HED [35]	.782	.804	.833
RCF [24]	0.806	0.823	-
Deep Contour[32]	.756	.773	.797
Canny [9]	.600	.640	.580
N4-fields [14]	0.753	0.769	-
RCN [20]	.823	.838	.853
OWN MODEL RCF	0.675	0.700	0.584
OWN MODEL RCN	0.619	0.649	0.535

Table 7.1: Performance of different models on the BSDS500 dataset

7.3 Cross-Validation

A very important factor is how the detector behaves on different types of data. For this purpose, the PASCAL VOC and NYU depth V2 datasets were used. These datasets specialize in object detection, compared to the BSDS500 dataset that is focused on edge detection and as such the predicted edge maps are expected to be more complex than the ground truths.

NYU depth V2

The NYU depth dataset is compromised of indoor images with a large amount of objects in each image. The ground truth that it expects are only the object outlines. As such the quality compared to the ground truth is relatively low. Figure 7.5 shows the results of

Figure 7.5: Results over the NYU depth V2 dataset. From left original image, ground truth, Own RCF based model, Own RCN based model.

the detectors over some examples of the dataset. The RCF based detector performs better than the RCN based detector when comparing these images. It still detects too many edges when compared to the ground truth. The RCN based model does not perform very good as it identifies too many edges and is too precise. This problem is expected as it also was a problem on the BSDS500 dataset.

PASCAL VOC

The PASCAL VOC dataset is focused on object detection. The ground truths contain 20 different classes that can be be found.

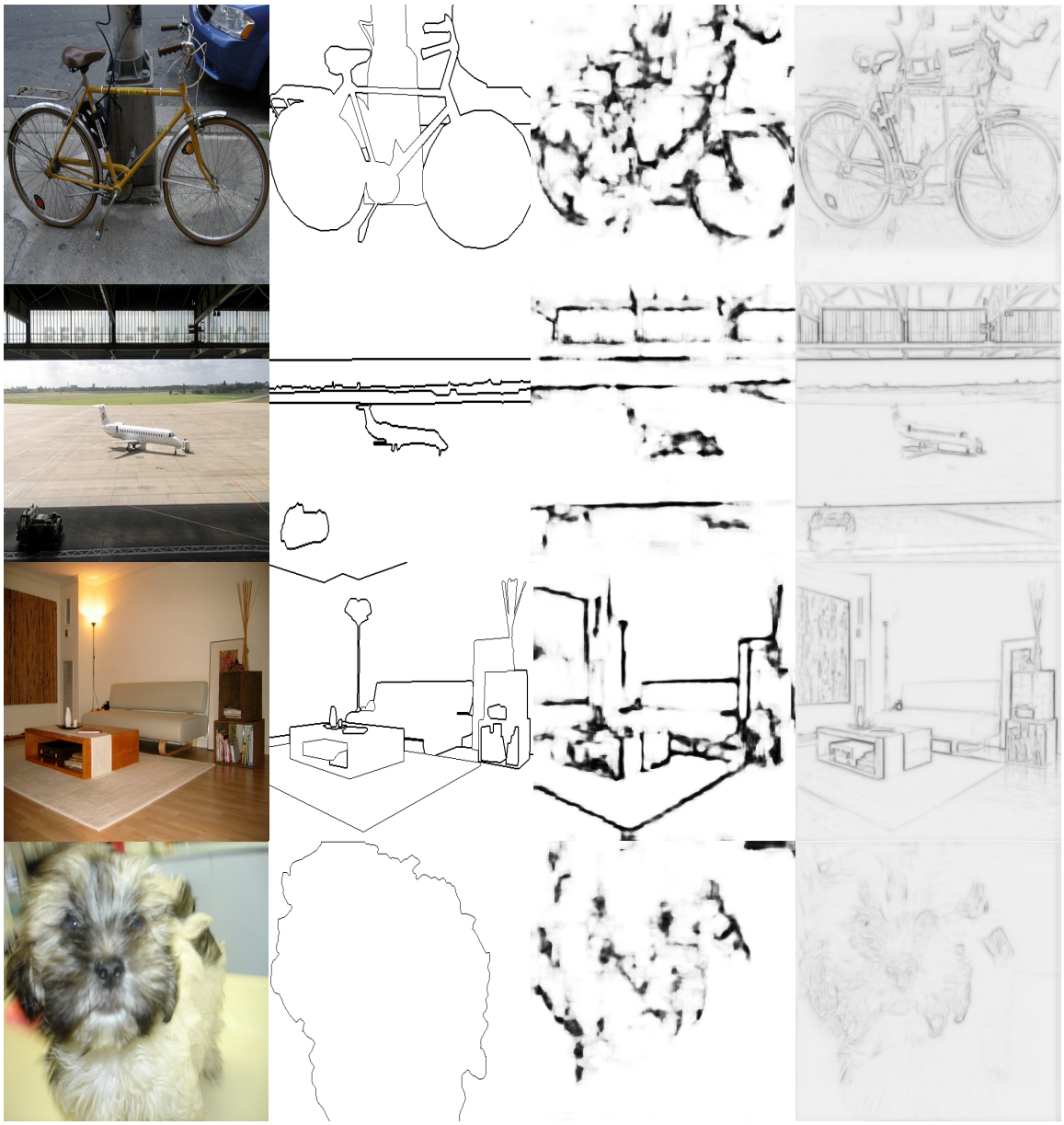

Figure 7.6: Results over the PASCAL VOC dataset. From left original image, ground truth, Own RCF based model, Own RCN based model.

The observed results are similar to the ones that were seen in the NYU depth V2 dataset. The RCF based detector seems to perform better as the edges are not as complex. The RCN based detector identifies too many edges and as such the results do not correspond to the ground truth.

Chapter 8

Conclusion

The goal of this thesis was to analyze existing edge detectors that use convolutional neural networks with the objective of designing a new detector that would either improve the quality or speed of the edge detection.

Altogether, two architectures were designed that were focused on edge detection. The first one modified the existing richer convolutional features (RCF) architecture with enriching the output. The side layers are combined in between stages to optimize the result. The second one combines the architecture of RCF and combines it with blocks that are part of RefineContourNet (RCN). This increases the number of learning parameters for the image and as such, should increase the quality. The designed models were trained using Google Colab with GPU acceleration. Existing datasets are used and augmented data is given to the models during training. Experiments were done on these models to improve the quality and afterwards the models were evaluated. The best result was produced from the architecture inspired from the combination of RCF and RCN with an ODS score of 0.675. Although the designed models did not manage to surpass the existing architectures in the quality of the detected edges, they managed to improve speed thanks to their smaller size.

Future improvements could definitely be made as the quality of the detected edges could improve with the consideration of using pre-trained weights of small architectures as the backbone and enriching them with the goal of better edge quality with the smallest drop in speed. Another option is the use of different datasets as only one dataset was used that specialized on edge detection.

Bibliography

[1] *Natural Language Processing with Deep Learning* [online]. [cit. 2020-04-22]. Available at: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture4.pdf.

[2] *MATLAB - MathWorks - MATLAB & Simulink* [online]. The MathWorks, Inc., 2019 [cit. 2020-05-22]. Available at: https://www.mathworks.com/products/matlab.html?s_tid=hp_products_matlab.

[3] *Welcome To Colaboratory - Colaboratory* [online]. 2019 [cit. 2020-05-22]. Available at: https://colab.research.google.com/.

[4] *Keras: the Python deep learning API* [online]. 2020 [cit. 2020-05-22]. Available at: https://keras.io/.

[5] Arbelaez, P., Maire, M., Fowlkes, C. and Malik, J. Contour Detection and Hierarchical Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* Washington, DC, USA: IEEE Computer Society. may 2011, vol. 33, no. 5, p. 898–916. DOI: 10.1109/TPAMI.2010.161. ISSN 0162-8828. Available at: http://dx.doi.org/10.1109/TPAMI.2010.161.

[6] Assirati, L., Silva, N. R. da, Berton, L., A. Lopes, A. de and Bruno, O. M. Performing edge detection by difference of Gaussians using q-Gaussian kernels. 2013. DOI: 10.1088/1742-6596/490/1/012020.

[7] Bertasius, G., Shi, J. and Torresani, L. *DeepEdge: A Multi-Scale Bifurcated Deep Network for Top-Down Contour Detection.* 2014.

[8] Bhanu, B. and Kumar, A. *Deep Learning for Biometrics.* Springer International Publishing, 2017. 219-225 p. Advances in Computer Vision and Pattern Recognition. ISBN 9783319616575.

[9] Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1986, PAMI-8, no. 6, p. 679–698.

[10] Cui, Q., Gao, B., Bian, J., Qiu, S. and Liu, T.-Y. *KNET: A General Framework for Learning Word Embedding using Morphological Knowledge.* 2014.

[11] Dollár, P. and Zitnick, C. L. Structured Forests for Fast Edge Detection. In: *ICCV.* 2013.

[12] Dollár, P. and Zitnick, C. L. Fast Edge Detection Using Structured Forests. *ArXiv.* 2014.

[13] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. et al. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*. january 2015, vol. 111, no. 1, p. 98–136.

[14] Ganin, Y. and Lempitsky, V. *N^4-Fields: Neural Network Nearest Neighbor Fields for Image Transforms*. 2014.

[15] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. 2015.

[16] Hlavsa, Z. et al. *Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools*. 2nd ed. CRC Press, 2010. 140-144 p. ISBN 978-1-4398-0205-2.

[17] Holloway, J., Kannan, V., Zhang, Y., Chandler, D. and Sohoni, S. GPU Acceleration of the Most Apparent Distortion Image Quality Assessment Algorithm. *Journal of Imaging*. september 2018, vol. 4, p. 111. DOI: 10.3390/jimaging4100111.

[18] Ioffe, S. and Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[19] Kamilaris, A. and Prenafeta Boldú, F. A review of the use of convolutional neural networks in agriculture. *The Journal of Agricultural Science*. june 2018, p. 1–11. DOI: 10.1017/S0021859618000436.

[20] Kelm, A. P., Rao, V. S. and Zoelzer, U. Object Contour and Edge Detection with RefineContourNet. 2019. DOI: 10.1007/978-3-030-29888-3_20.

[21] Kong, H., Cinar Akakin, H. and Sarma, S. A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications. *Cybernetics, IEEE Transactions on*. january 2013, vol. 43, p. 1719–1733. DOI: 10.1109/TSMCB.2012.2228639.

[22] Lee, C.-Y., Gallagher, P. W. and Tu, Z. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. In: Gretton, A. and Robert, C. C., ed. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Cadiz, Spain: PMLR, 09–11 May 2016, vol. 51, p. 464–472. Proceedings of Machine Learning Research. Available at: http://proceedings.mlr.press/v51/lee16a.html.

[23] Lin, G., Milan, A., Shen, C. and Reid, I. *RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation*. 2016.

[24] Liu, Y., Cheng, M.-M., Hu, X., Wang, K. and Bai, X. Richer Convolutional Features for Edge Detection. 2016.

[25] Nathan Silberman, P. K. and Fergus, R. Indoor Segmentation and Support Inference from RGBD Images. In: *ECCV*. 2012.

[26] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018.

[27] O'Shea, K. and Nash, R. *An Introduction to Convolutional Neural Networks*. 2015.

[28] Prewitt, J. M. S. Object enhancement and extraction. In:. 1970.

[29] Rashmi and Saxena, R. Algorithm and Technique on Various Edge Detection : A Survey. *Signal & Image Processing : An International Journal*. june 2013, vol. 4, p. 65–75. DOI: 10.5121/sipij.2013.4306.

[30] Refaeilzadeh, P., Tang, L. and Liu, H. Cross-Validation. *Encyclopedia of database systems*. Springer: New York. 2009, vol. 5.

[31] Saito, T. and Rehmsmeier, M. The PrecisionRecall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PloS one*. march 2015, vol. 10, p. e0118432. DOI: 10.1371/journal.pone.0118432.

[32] Shen, W., Wang, X., Wang, Y., Bai, X. and Zhang, Z. *DeepContour: A Deep Convolutional Feature Learned by Positive-sharing Loss for Contour Detection*. June 2015. DOI: 10.1109/CVPR.2015.7299024.

[33] Simonyan, K. and Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014.

[34] Sobel, I. An Isotropic 3x3 Image Gradient Operator. *Presentation at Stanford A.I. Project 1968*. february 2014.

[35] Xie, S. and Tu, Z. *Holistically-Nested Edge Detection*. 2015.

[36] Yasiran, S., Jumaat, A., Malek, A., Hashim, F., Nasrir, N. et al. Microcalcifications segmentation using three edge detection techniques. In:. November 2012, p. 207–211. DOI: 10.1109/ICEDSA.2012.6507798. ISBN 978-1-4673-2162-4.

[37] Zitnick, C. L. and Dollár, P. Edge Boxes: Locating Object Proposals from Edges. In: *ECCV*. 2014.

Appendix A

Manual

This manual displays the expected project structure and needed prerequisites. As the implementation environment was google colab the manual is going to be focused on the settings on this environment.

Project Structure

This is the recommended project structure:

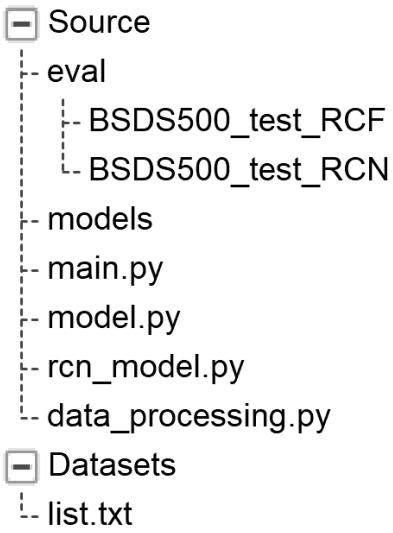

Figure A.1: Structure of the directory

Scripts description:

- `config.json` – Configuration file
- `main.py` – used for training
- `eval.py` – runs the saved weights in config file on image set
- `data_processing.py` – contains generators that supply the images for training

- `model.py` – contains the RCF based model
- `rcn_model.py` – contains the RCN based model

Folder explanation: The eval folder contains 2 subfolders. One stores the results for RCN based model. The other one stores the results for the RCF based model. Model folders contains the model weights that are saved during training.

`config.json` explanation:

- image_x and image_y – set input dimension for RGB image
- image_gt_y and image_gt_y – set input dimension for ground truth
- image_channels – defines number of image channels for both images and ground truths
- epochs – number of epochs for training
- batch_size – batch size for one epoch
- length – number of rows taken from list file. Must at most be equal to the length of images in a list file
- val_length – number of rows that will be read from validation list
- path – specifies path to the list file for the dataset
- path_val – specifies path to the validation list file for the dataset
- model – RCN or RCF, sets which model will be used
- learning_rate – set the learning rate for training
- best_res_rcn – path to stored weights for rcn model
- best_res_rcf – path to stored weights for rcf model
- use_weight – 1 or 0, sets if weights should be loaded.
- base_path – defines the base path to the dataset. Can be used in combination with list file.
- eval_image – path to test images
- eval_gt – path to corresponding test groundtruth
- enable_augmentation – 1 or 0, defines if augmentation should be used

Project Settings

The first important part is to install the correct version of keras. This is done by running the command `!pip install tensorflow-gpu==1.15` in a Jupyter notebook. For training purpose it is recommended to enable the GPU usage. To set this setting got to `Runtime` in the menu. Then select `Change runtime type` and choose `GPU` in the drop down menu for hardware acceleration. All other dependencies (Openccv..) are already included in colab and do not need installation.

To run the training phase of the program you just need to execute the script `main.py`. It automatically parses the config file. It does not take any parameters. To use the pre-trained weights and generate images for evaluation use `eval.py`

Appendix B

Poster

Figure B.1: Poster A2 format