Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Graz University of Technology
Institute for Computer Graphics and Vision

Tadej Vodopivec

Segmentacija rok za obogateno resničnost
Hand Segmentation for Augmented Reality

magistrsko delo
master's thesis

študijski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Peter Peer
Somentor: univ. prof. dr. Vincent Lepetit

Gradec, 2016

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Tadej Vodopivec sem avtor magistrskega dela z naslovom:

Segmentacija rok za obogateno resničnost

Hand Segmentation for Augmented Reality

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Petra Peera in somentorstvom univ. prof. dr. Vincenta Lepetita,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Gradcu, 22. maja 2016 Podpis avtorja:

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, May 22 2016 Signature:

Contents

Povzetek

Naslov: Segmentacija rok za obogateno resničnost

Zaznavanje prekrivanj je pomemben del obogatene resničnosti, ker omogoča izris prepričljivih kompozicij resničnih in navideznih predmetov. Najtežji del izdelave takih kompozicij je zaznavanje kdaj resnični predmeti ležijo med uporabnikom in navideznim predmetom. Ker so roke pogosto v uporabnikovem vidnem polju, je pomembno, da se natančno določi njihovo pozicijo in kateri deli navideznih predmetov bi morali biti vidni.

Razpoznavanje rok je zelo zahtevna naloga, saj so roke lahko različnih oblik in barv, lahko izgledajo zelo različno iz različnih vidnih kotov, imajo lahko odprto ali zaprto dlan in različne položaje prstov, so lahko delno prekrite in lahko oprijemajo različne predmete.

Barva kože je ena izmed očitnejših lastnosti, a se v praksi izkaže, da ni dovolj zanesljiva, saj obstaja veliko predmetov podobne barve in tudi barva na sliki, zaradi različnih svetlobnih pogojev in ne-optimalne nastavitve beline, ni vedno zanesljiva. Kot se izkaže, se pri naši množici slik ta metoda izkaže celo slabše kot metoda večinskega razreda. Podobne metode uporabljajo kamere z detekcijo globine, so uporabne le v kontroliranem okolju, ali pa zaznajo le položaj roke brez njenih robov.

V tej nalogi je opisana metoda za segmentacijo rok na osnovi konvolucijske nevronske mreže. S to metodo smo na slikah, posnetih iz prvo-osebnega pogleda, natančno in učinkovito zaznali področje, kjer roke zasedajo del vidnega polja. Slike zajemajo različne okoliščine znotraj stavbe in izven nje, a največji poudarek je na pisarniškem okolju. Pričakujemo, da bo ta metoda

najbolj uporabna na področju obogatene resničnosti, kjer uporabnik nosi sistem za navidezno resničnost na osnovi očal in z resničnim svetom upravlja z rokami, ki jih vidi direktno skozi delno prosojna očala ali kot sliko zajeto s kamero.

Metoda uporablja konvolucijske nevronske mreže, ki so različica umetnih nevronskih mrež, prilagojena za delo s slikami. Umetne nevronske mreže so simulirana mreža nevronov po vzoru možganov in se uporablja za izračunavanje funkcij, ki so odvisne od velikega števila vhodnih podatkov. Mreže so sestavljene iz logičnih nevronov, ki so med seboj povezani in si izmenjujejo informacije. Nevroni so razporejeni v plasti, kjer je vsak nevron v neki plasti povezan z vsakim nevronom v predhodni plasti. Na prvi plasti se vnesejo vhodni podatki, na zadnji plasti se razberejo izhodni podatki, vmesne plasti pa so skrite. Informacije potujejo samo v eno smer in to od vhodne plasti proti izhodni. Ko podajamo število plasti, po navadi štejemo vse plasti razen vhodne. Tako mreža z eno plastjo predstavlja izhodno plast povezano z vhodno plastjo, brez vmesnih skritih plasti, mreža z dvema plastema pa vsebuje vhodno plast, eno skrito plast in eno izhodno plast. Izhod nevrona predstavlja vrednost, ki se izračuna kot utežena vsota njegovih vhodov in dodano konstanto. Uteži vsake povezave in konstante je mogoče prilagoditi in se s tem naučiti nove odvisnosti. Velikost mreže se po navadi opiše s številom nevronov ali številom parametrov.

Konvolucijske nevronske mreže so prilagojene za delo s slikami in izkoriščajo nekatere lastnosti slik, kot na primer dejstvo, da je slikovna točka sestavljena iz treh barv, in s tem omogočajo učinkovitejše računanje. Delujejo tako, da na vsaki plasti izračunajo zemljevide značilk, iz katerih lahko na zadnji plasti razberemo rezultat. Vsaka točka na zemljevidu značilk se izračuna na podlagi vrednosti točk v njeni okolici na prejšnji plasti, na vseh zemljevidih značilk.

Za potrebe učenja nevronske mreže smo posneli množico slik. Za vsako sliko v množici smo pripravili željen pravilen rezultat segmentacije tako, da smo ročno označili posamezne slikovne točke, ki na sliki predstavljajo del

rok. Skupno smo posneli in označili 348 slik, od katerih je bilo 191 posnetih v pisarniškem okolju in 157 pri vsakodnevnih opravilih. Slike so bile posnete na 6 različnih lokacijah pri različnih svetlobnih pogojih.

Metoda, ki smo jo razvili, je sestavljena iz dvostopenjskega klasifikatorja, kjer se na prvi stopnji izvede groba segmentacija pri nizki ločljivosti, na drugi stopnji pa se s pomočjo rezultata prve stopnje izvede končna segmentacija pri polni ločljivosti. Pri obeh stopnjah se sliki dodata tudi njeni kopiji polovične in četrtinske ločljivosti in za vsako od treh ločljivosti je zgrajena konvolucijska mreža iz treh konvolucijskih plasti. Rezultat zadnje plasti vsake izmed treh ločljivosti je nato vhodni podatek ene popolnoma povezane regresijske plasti, ki vrne končni rezultat. Razlika med stopnjama je v tem, da prva stopnja dela s sliko s četrtino ločljivosti originalne slike, druga stopnja pa deluje s sliko na polni ločljivosti. Druga stopnja ima poleg originalne slike na razpolago tudi povečan rezultat prve stopnje. Za tako razdelitev smo se odločili, ker je na ta način mogoče pri nižji ločljivosti izračunati več zemljevidov značilk in tako zaznati kompleksnejše odvisnosti.

Učenje mreže se izvaja s pomočjo učne in testne množice. S pomočjo učne množice se določijo uteži in konstante na nevronih, namen testne množice pa je preveriti, ali so te vrednosti specifične samo za uporabljene učne primere ali pa so zaključki enaki tudi pri še ne videnih primerih. Postopek se večkrat ponovi in vsakič popravi vrednosti uteži in konstant. Učenje se ponavlja dokler je mogoče najti kombinacije vrednosti uteži in konstant, ki omogočajo boljše rezultate.

Končna metoda je sposobna z visoko zanesljivostjo v realnem času določiti območje rok v pogojih, ki so podobni pogojem na slikah iz naše množice. S pomočjo večje in raznolikejše množice slik bi bilo mogoče rezultate dodatno izboljšati in klasifikator natrenirati za uporabo v še več različnih okoljih.

Ključne besede: segmentacija, obogatena resničnost, konvolucijske nevronske mreže, računalniški vid, zaznavanje rok.

Abstract

Title: Hand Segmentation for Augmented Reality

Occlusion detection is a very important part of augmented reality because it allows us to render convincing compositions of real and virtual objects. The hardest part of creating such composition is to detect when real objects lie between the user and the virtual object. Because hands are often in our field of view, it is important to accurately detect their position to determine which parts of the virtual objects should be visible. In this work we describe a method for hand segmentation based on a convolutional neural network. With this method we were able to efficiently and accurately detect the area, where the hands were directly visible in a set of first-person view images. The images ranged from outdoors to an office-like environment. We expect the method to make the biggest impact in the field of augmented reality, where the user wears glasses-based augmented reality system and interacts with the world with his hands.

Keywords: segmentation, augmented reality, convolutional neural networks, computer vision, hand detection.

Chapter 1

Introduction

Augmented reality is a set of technologies that, using illusions, give an impression that there are things in space when in reality they are not there. It is a blend between the real world and virtual reality, where the real world can be the main focus of attention or used just as context for virtual objects. Virtual reality is used to add virtual elements, which can be virtual objects or visualized information. Augmentation can be performed in real time on a video stream or on a previously captured video clip. Perspective can be first-person, where the user can move freely around the world, or third person, where the camera moves independently of the observer.

To ensure that the user perceives the virtual objects as part of the world around them, the objects have to be inserted convincingly enough. Objects have to interact with each other and with the real world in two ways: physically and visually. Physically means that the object obey the physical laws, move continuously, detect collisions and that no object is located in the same place as another real or virtual object. It is important to realize that the interaction is completely one-way. That is because the real world can affect the virtual world, but not vice-versa. Visually, on the other hand, means that the objects are properly displayed. This includes correct perspective, reflections, lighting, shadows and occlusions.

1.1 Occlusions

Occlusions occur when two or more objects in the visual field lie at the same position relative to the observer, but at a different distance. The closest objects cover more distant ones, which are then only visible partly or not visible at all. In the real world occlusions are an everyday occurrence and we do not think about them. We are so accustomed to them that we can hardly imagine a world without them. Occlusions give us very good information about where the objects are in relation to each other and how far they are from us. Without taking occlusions into account, the object is not perceived as part of the environment or we think that it hovers somewhere in space.

1.2 Hand Segmentation for Augmented Reality

A big part of augmented reality is augmentation in first person perspective in real time. The virtual objects are displayed on glasses that the user wears. While wearing the glasses, the user interacts with the real world with his hands. Because hands are a part of the person's body and usually in front of him, they are very close and often occlude at least a part of the image.

Due to the difficulty of detecting occlusions between real and virtual objects, the potential of usage of glasses for augmented reality, and frequency of the user's hands being in the field of view, we decided to develop an algorithm for automatic hand segmentation.

The goal of this project is to develop an efficient segmentation method able to segment hands over a cluttered background. In augmented reality scenarios, the user's hands are often in the field of view and should occlude the virtual objects. However, because such occlusions are difficult to estimate and render, this is usually not done.

Chapter 2

Related Work

Hand segmentation is a very challenging task as hands can be very different in shape and skin color, look very different under another viewpoint, can be closed or open, can be partially occluded, can have different positions of the fingers, can be grasping other objects or other hands etc.

Color of the skin is the most obvious cue and it would be the first thing that comes to mind [1, 2]. Unfortunately, in practice, this method offers insufficient precision for convincing integration, as, in the real world, there are a lot of objects with similar color as the skin. Some of the examples include wood, white wall under low light, many objects made of plastics. This method also relies heavily on precise color detection and correct white balance.

Some other approaches assume that the camera is static and hands are segmented based on their movement [3], use of depth information obtained by a RGB-D camera [4], or the approach assumes a very simple, sometimes even single-color background [5]. It is also possible to detect hand position without recognizing exact edges [6]. Each of these approaches has its advantages and disadvantages, but none of them provide a good result in everyday application from first-person view in a complex environment.

The method that we suggest is based on a convolutional neural network that will work on multiple scales simultaneously. As convolutional neural

networks are very computationally intensive, we increased the speed while still achieving good results by splitting the network into two parts. The first part is a low-resolution hand segmentation, which feeds the result to a smart upscaling part.

Chapter 3

Method

In this section we describe our approach and show how such a system can be built. We will start by introducing the technology used, the used dataset, describe the classifier structure and its main two parts. The base of the proposed algorithm is a convolutional neural network, which is a special case of an artificial neural network.

3.1 Artificial Neural Networks

Artificial neural network is a simulated network of neurons inspired by biological neural networks (the central nervous systems of animals, in particular the brain) [7]. They are used to approximate functions that can depend on a large number of inputs. Artificial neural networks are often presented as systems of interconnected neurons that exchange pulses between themselves. The connections have associated weights and biases that can be modified, making neural networks adaptive to inputs and capable of learning. Like other machine learning methods, neural networks have been used to solve a wide variety of tasks that are hard to solve using ordinary rule-based programming, including tasks in computer vision and speech recognition.

The basic computational unit of a biological neural network is a neuron. Current studies estimate that the average adult human brain contains

roughly 86 billion neurons [8]. As a single neuron can have thousands of synapses, the estimated number of these is in the trillions (estimated at up to 150 trillion). Figure 3.1 shows a drawing of a biological neuron (top) and a common mathematical model (bottom). Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out and connects via synapses to dendrites of other neurons.

In the computational model of a neuron, the signals that travel along the axons interact multiplicatively with the dendrites of the other neuron based on the synaptic strength (weight w_n) at that synapse. The idea is that the synaptic strengths can be modified and control the strength of influence between the neurons. In the basic model, the dendrites carry the signal to the cell body, where they all get summed. If the final sum is above a certain threshold, the neuron is fired and sends a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function, which takes a real-valued input (the signal strength after the sum) and outputs a value in range between 0 and 1. In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function). It is important to note that this is not an exact model of a neuron like in a human brain, but is only inspired by it, and much simplified. [9]

Artificial neural networks are modelled as collections of neurons that are connected in an acyclic graph as seen in Figure 3.2. This means that the outputs of neurons become inputs to other neurons, and cycles are not allowed since that would imply an infinitely repeating computation. Instead of representing them as unorganized sets of neurons, artificial neural network models are normally organized into distinct layers of neurons. Neural networks con-

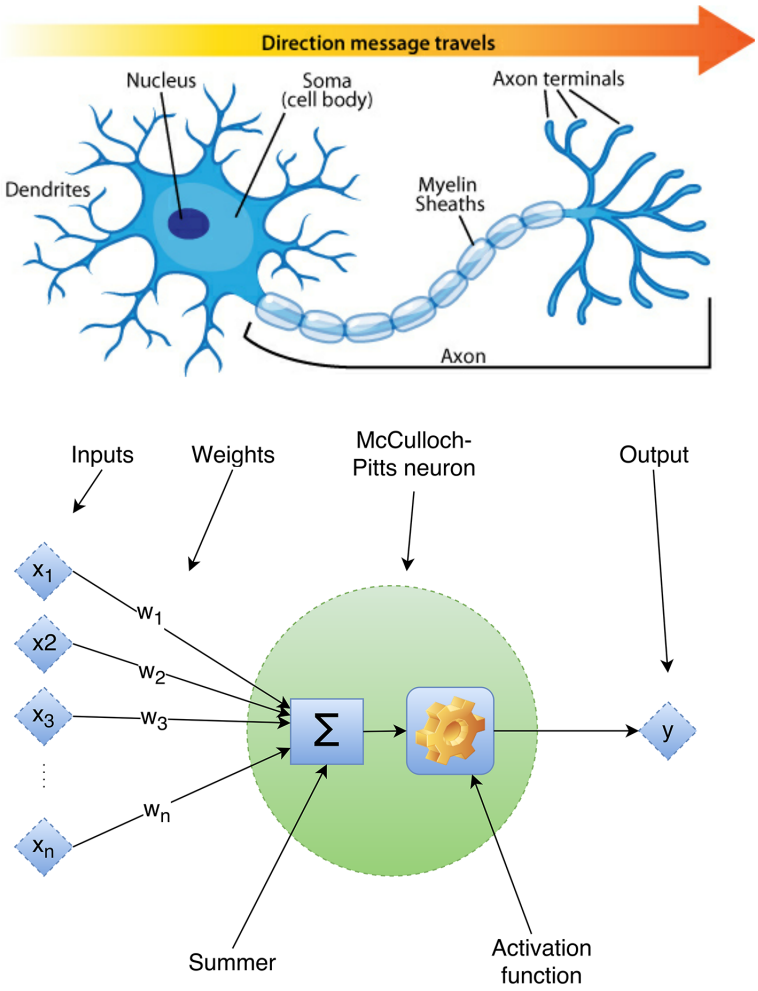

Figure 3.1: Comparison of a simplified biological [10] and a mathematical model of a neuron as first proposed by Warren McCulloch and Walter Pitts in 1943 [11]

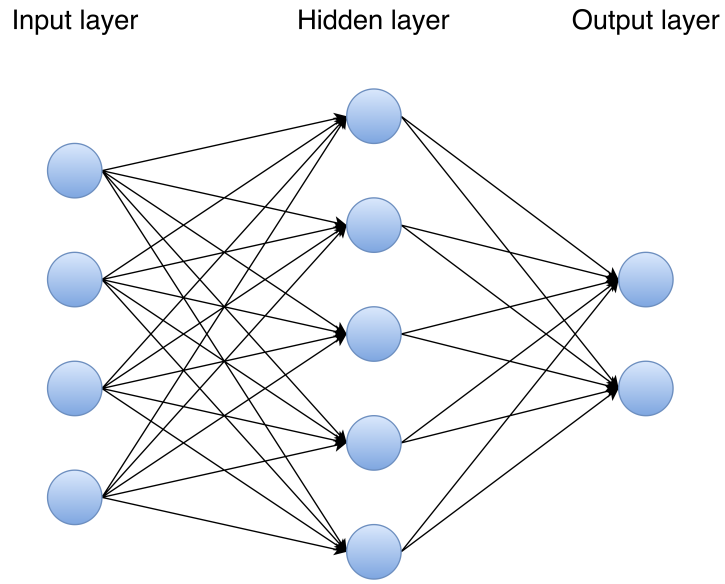

Figure 3.2: A 2-layer neural network with four input neurons, one hidden layer with five neurons and one output layer with two classes. Notice that in both cases there are connections (synapses) between neurons between layers but not within a layer

sist of multiple such layers, where the first layer is considered an input layer and the last layer is considered an output layer. All layers in between are hidden layers. When we count the number of layers, we normally omit the input layer, so a single-layer network consists of an input and output layer, but no hidden layers. For regular neural networks the most common layer type is a fully-connected layer, where outputs from each neuron in one layer is connected to an input of each neuron in the next layer, but neurons within a single layer share no connections. The size of artificial neural networks is normally described with the number of neurons or the number of parameters.

3.2 Convolutional Neural Networks

Convolutional neural networks [12] are much like artificial neural networks from the previous chapter – they are made up of neurons that have modifiable weights and biases. Each neuron receives some inputs, performs a dot product and continues with a non-linearity. The whole network takes image pixels as inputs and outputs a value on the last layer.

Thus, the difference is that convolutional neural networks assume that the inputs are images, which allows us to use certain properties of the input data to make the computation more efficient and reduce the number of parameters in the network. Artificial neural networks receive an input as a single vector and perform calculations in a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer and where neurons in a single layer function completely independently and do not share any connections. Because of this, each neuron has a lot of inputs. For a 100×100 pixel color image that would be 100×100×3 = 30.000 inputs, assuming color is represented with three values. For a 720×480 image that would be over a million inputs per neuron!

Convolutional neural networks take advantage of the fact that the input data is an image, where each pixel is a vector of multiple values (normally red, green and blue color) and has a local neighbourhood. This is why neurons in layers of a convolutional neural network are arranged in three dimensions. The input layer has the same dimensions as the input image resolution and color information. One processing step consists of a convolutional layer and pooling layer. Convolutional layer will compute the output of neurons that are connected to local neighbourhood in the input and output a volume of the same size as input image resolution and a number of feature maps that we select, and process the activations. After it is the pooling layer, which will perform a downsampling among the spatial dimensions, but leave the number of feature maps unchanged. There may be multiple such steps, where output of one pooling layer is input to the next convolutional layer, and each next convolutional layer has the same dimensions as the previous pooling layer.

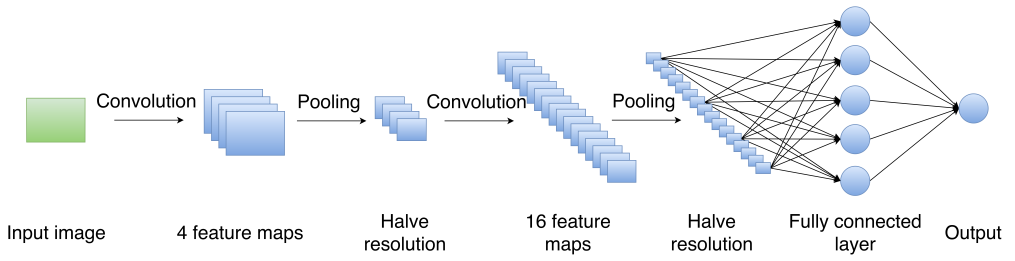

Figure 3.3: An example convolutional neural network structure

The last layer is a fully connected layer of dimensions 1×1×N, where N is the number of possible classes. This layer has the same function as in a normal artificial neural network. An example can be seen in Figure 3.3.

3.2.1 Convolutional Layer

A convolutional layer consists of a set of filters, where each filter is a set of neurons that are arranged in three dimensions [13] as seen in Figure 3.4. First two dimensions should be small. They define the area that will influence the output. The third dimension has to be the same as that of the input data – in case this is the first layer, this will normally be the number of colors in the image, otherwise this will be number of feature maps on the previous layer. During processing we convolve (slide) this filter over the input data to produce a two-dimensional feature map. Intuitively, the network will learn filters that activate when they encounter a specific feature. Output of the convolutional layer is a stack of all the feature maps. As the feature maps are 2D the output size of this layer is width × height × number of feature maps. As the filters are convolved, and for their computation neighbouring pixel values are needed, the computation can only be made for pixels that have enough neighbours, depending on filter size. Consequently, feature maps are not of the same resolution as the input image. To counter this a zero-padding is normally introduced, to pad the original image with zero values of the size of half of the filter, so that the feature maps resolution stays the same. In

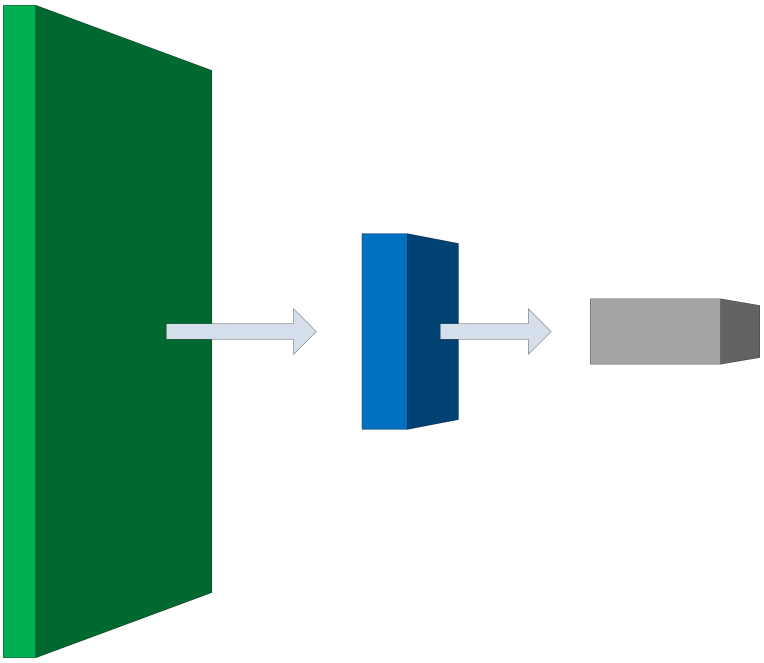

Figure 3.4: In a convolutional layer, the neurons are arranged in three dimensions (width, height, depth). Every layer of a convolutional neural network transforms a 3D input volume to a 3D output volume of neuron activations. In this example, the green input layer holds the image. Its width and height are same as the dimensions of the image, and the depth is same as the number of color channels (normally 3 for red, green and blue). Blue layer represents a subset of the green layer, where only the part that influences current filter computation is used. This is also known as receptive field. The grey block represents neurons of the current filter

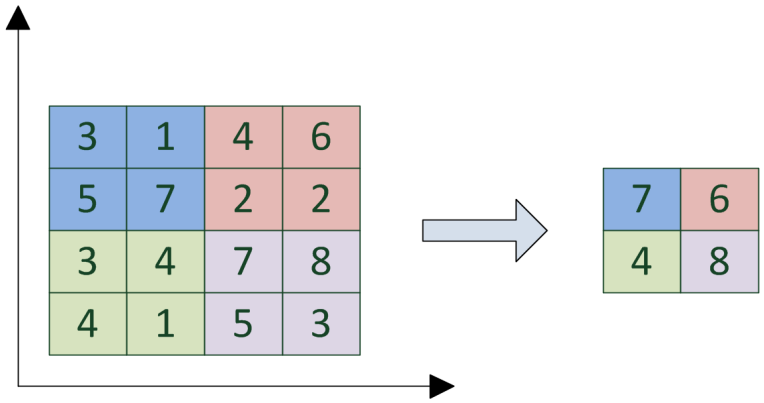

Figure 3.5: Maximal pooling with a 2×2 filter and stride=2

other words, enough pixels are added around the image border, so that the filter can be calculated for each pixel of the original image. Added pixels assume a black color.

3.2.2 Pooling Layer

The function of a pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Consequently, this also reduces the chance for overfitting. An example of such reduction can be seen on Figure 3.5. The pooling layer computes independently on every feature map of the input and resizes its dimensions, only leaving the biggest value in a specific neighbourhood. Normally, pooling is performed in such a way that it halves both width and height of the feature map, discarding 75% of information in the process. This can be described as a pooling layer with filter size 2×2 and stride of 2. Filter size determines the area in which the maximal value is determined, stride determines the step size. If the stride is smaller than any filter dimension, there can be some overlap.

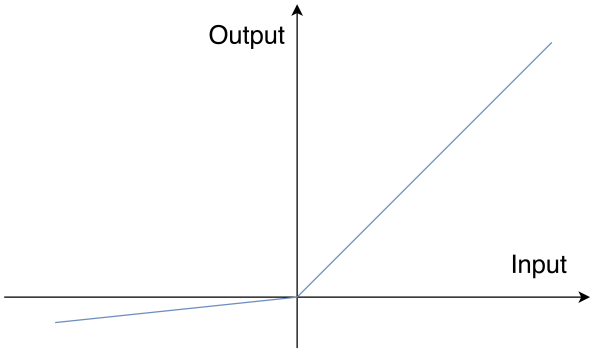

Figure 3.6: Leaky rectified linear unit that allows a small, non-zero gradient when the unit is not active. This enables it to recover from a zero value

3.3 Activation, Cost, and Parameter Update Functions

The activation function translates the input signals to output signals. Commonly used types are logistic sigmoid neurons and hyperbolic tangent neurons. While logistic sigmoid neurons are more biologically plausible than hyperbolic tangent neurons, the latter work better for training multi-layer neural networks. Rectifying neurons are an even better model of biological neurons and yield equal or better performance in spite of the hard non-linearity and non-differentiability at zero [14]. As of 2015, rectifier is the most popular activation function for deep neural networks [15].

In our method we used a leaky rectified linear unit as shown in Figure 3.6, which is a variation of rectifying neuron that allows a small, non-zero gradient when the unit is not active. This enables it to recover from a zero value [16].

For the cost function we chose a boosted cross-entropy objective function [17]. The function is inspired by the intuitive idea that people always learn more from difficult problems in which they are prone to make mistakes. Using this cost function, the network learns more from the samples, where the resulting output has lower probability. We used alpha value 2 as described in the original paper.

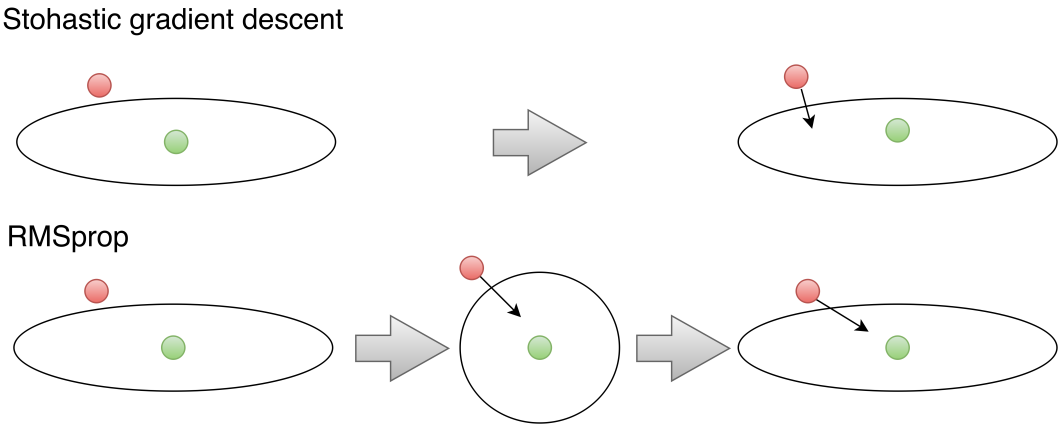

Figure 3.7: Comparison of stochastic gradient descent (top) and RMSprop (bottom). In stochastic gradient descent, the step is calculated as direction of maximal gradient, while in RMSprop, the gradient is normalized by the gradient seen in recent past, so that the direction of gradient descent is better aligned with the direction of the extreme. Green circle represents the local minimum, red circle represents current position, and the black arrow points in the direction of calculated gradient

Once the analytic gradient is computed with backpropagation, the gradients are used to perform a parameter update. In our method we used the RMSprop [18] method. It works similarly to the stochastic gradient descent – it makes small parameter adjustments using local derivative information, but the difference here is that as gradients are computed during each parameter update, a moving average of gradient magnitudes is maintained as well. At each update the weighted moving average chart is used to compute the root-mean-square (RMS) gradient value that has been seen in the recent past. The actual gradient is normalized by this RMS scaling factor before being applied to update the parameters as seen in Figure 3.7.

3.4 Training Dataset

Before we could start, we prepared a dataset of samples that would serve as ground truth of accurate hand segmentation as seen in Figures 3.8 and 3.9. These examples are used for training and for evaluation of the results. The samples consist of a pair of an image and its correct segmentation.

The basic idea is to capture images of hands on different backgrounds from first-person perspective. Based on these images, monochrome bitmaps were manually created, where white means that corresponding pixel of the original image is a part of a hand and black means that corresponding pixel of the original image is not part of a hand.

In order to achieve this, a subject performed a set of tasks while wearing a wide-angle camera that was mounted on his head near his eyes. The camera was set to take periodic images of whatever was in the field of view at that time. In total 348 images were taken. 191 of those images were taken in an office-like environment at 6 different locations under different lighting conditions. The remaining 157 images were taken in and around a residential building, while performing everyday tasks like walking around, opening doors etc.

Images were taken with an IDS MT9V032C12STC sensor with resolution of 752×480 pixels. The dataset was split between the two classifier parts as described in Sections 3.6, where equal number of images was used for each part of the classifier and further split into train and test sets as described in Section 4.

3.5 Classifier Structure

The classifier consists of a convolutional neural network and a logistic regression layer as seen in Figure 3.10. The convolutional neural network is constructed with three chains of three convolution layers, where first chain works on a full resolution image, the second chain works on an image that is half the resolution of the original image and the third chain works on an

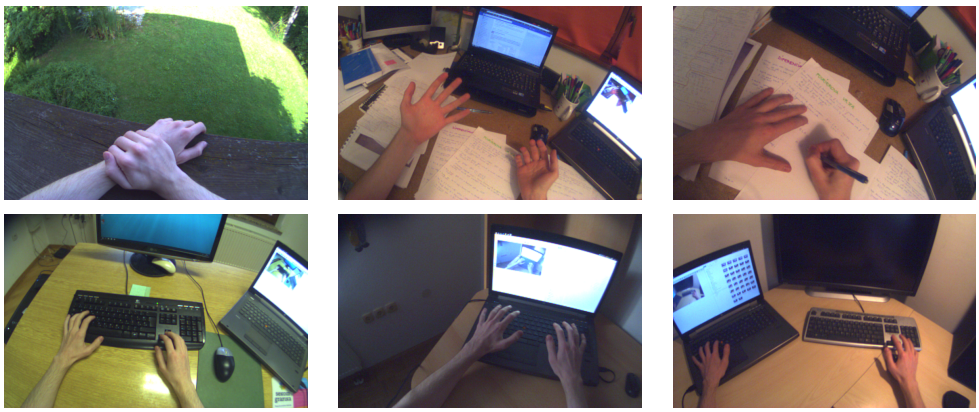

Figure 3.8: Some of the images used in training

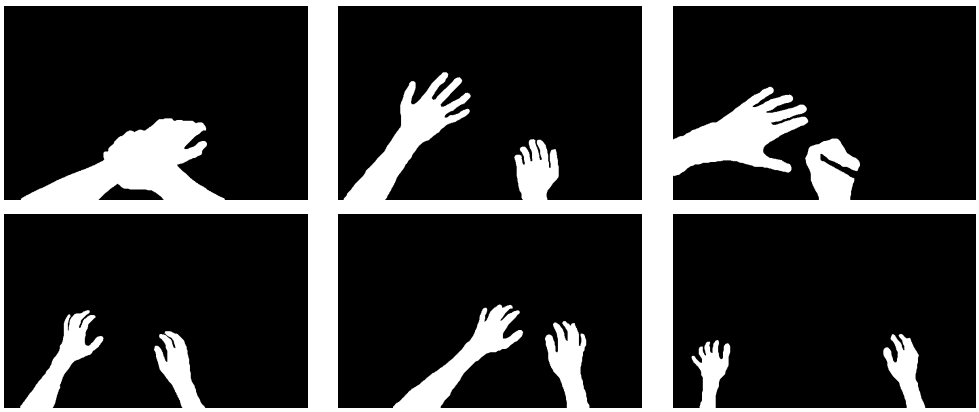

Figure 3.9: Ground truth segmentation for the images shown in Figure 3.8

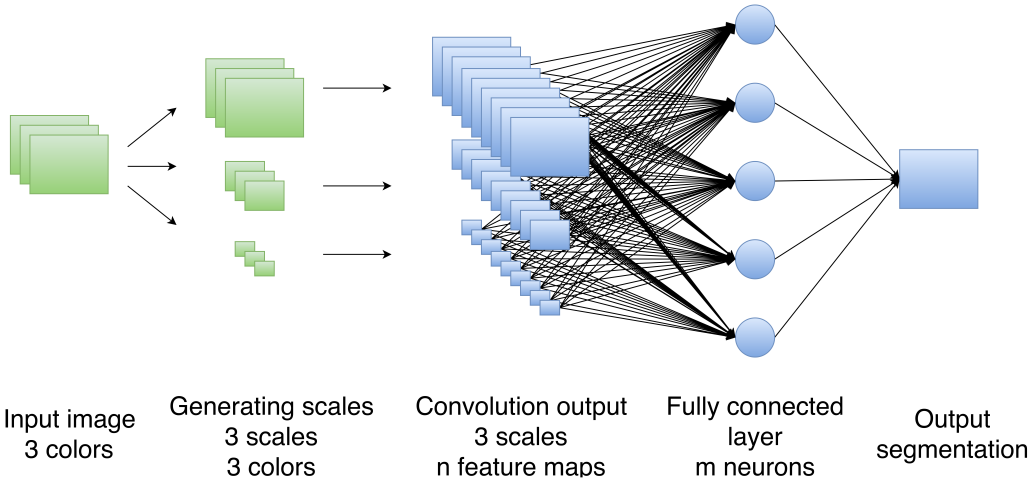

Figure 3.10: Classifier structure. Note that in order to make the structure more understandable, only one convolutional layer is shown, and the number of feature maps n and the number of neurons on the fully connected layer m is underrepresented

image that is one quarter of the original image's resolution. All three chains use same filter sizes and have the same number of feature maps. Because we want the resulting output to be of the same resolution as the input image, no pooling layers are used. Three chains that work on different resolutions are used so that the filter size in pixels can be relatively small, while still taking into account a big part of the image, which would otherwise not be possible without a pooling layer.

Output of these three chains is joined and used as an input to the logistic regression layer, which outputs probability of a pixel containing a hand for each pixel of the image.

3.6 Splitting the Classifier into Two Parts

As the classifier is very computationally intensive, we split it into two parts. The first part does the hand segmentation on an image with reduced reso-

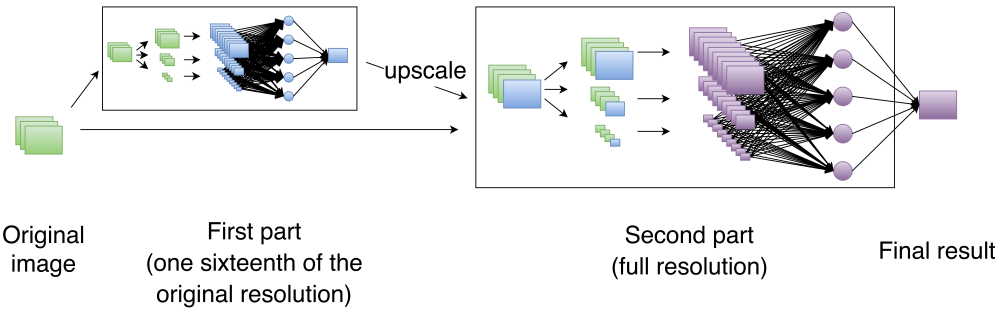

Figure 3.11: Split classifier structure. Note that, like in Figure 3.10, in order to make the structure more understandable, only one convolutional layer is shown, and the number of feature maps and the number of neurons on the fully connected layer is underrepresented in both parts of the classifier

lution. Its result along with the original image is then used as input to the second part, which performs a simplified version of the classifier to produce the final segmentation on the full resolution as seen in Figure 3.11.

The simplified version of the classifier has exactly the same structure as the classifier in the first part, the difference is that it takes an image in full resolution, with an addition of a linearly upscaled output of the first part of the classifier as the fourth channel next to the three color channels. Simplification is done by reducing the number of feature maps and using smaller filter sizes.

3.6.1 Reduced Resolution Segmentation Estimation

The first part of the classifier takes an image with resolution of one sixteenth of the original image as an input and outputs a result of the same resolution.

Original image in our case had a resolution of 752×480, which was reduced to 188×120. Three scales of the image were first generated, where each next scale is half the size of the previous. In our case the resolutions for each scale were 188×120, 94×60, and 47×30 pixels. For each scale a convolutional neural network is constructed, where each network has the same structure.

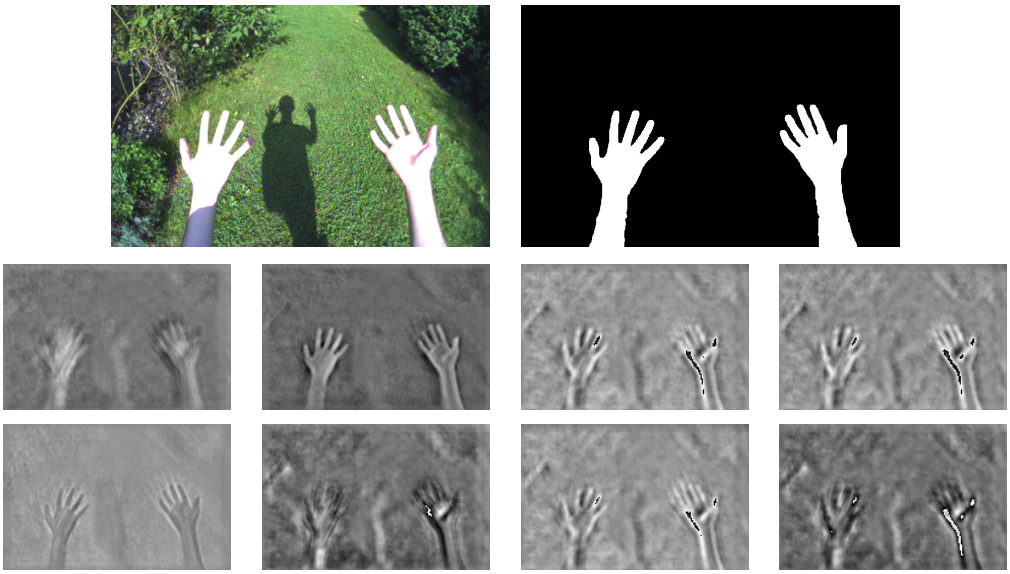

Figure 3.12: Original image, its ground truth segmentation, and some of the resulting feature maps used by the classifier

The network consists of three convolutional layers and no pooling layers, where the first two layers output 32 feature maps and the third layer outputs 16 feature maps. An example of such feature maps can be seen in Figure 3.12. Filter sizes are 3×3, 5×5, and 7×7 pixels for corresponding layers.

Outputs of these networks are then concatenated and used as the input to a logistic regression layer, which calculates the probability for each pixel that it belongs to a part of a hand. For the purpose of this classifier, pixel values of the output image are probability of the class hand instead of binary classification. This way, the output contains more information that the second part of the classifier can use. As the binary classification is just choosing the class with the highest probability, this in practice only means that the step, where class probabilities are compared, has to be skipped. An example of such image can be seen in Figure 3.13.

Figure 3.13: Sample output of the reduced resolution segmentation estimation. Shades of grey represent the probability of the class hand instead of binary classification.

3.6.2 Full Resolution Final Result

The resolution of the output of the first part of the classifier is one sixteenth of the original resolution, which is a good indication of where the hands are, but is not very precise when you upscale it to the original resolution. This is where the second part comes in. It uses the information from the first part in combination with the original image to effectively segment the image.

This part works similar to the first part, but the input image here is of full resolution, with added linearly upscaled output of the first part.

As in the first part, three scales of the image are generated. In our case the resolutions for each scale were 752×480, 376×240, and 188×120. The same network structure was constructed again, but this time the first layer outputs 8 feature maps, the second layer outputs 4 feature maps and the third layer outputs 1 feature map. Filter sizes are 3×3 pixels for all layers.

Outputs of these networks are then combined as in the first part, but the pixel values of the output image are the predicted class instead of the class probability, where white means hand and black means not hand.

Chapter 4

Training

Like most machine learning methods, convolutional neural networks require training, where the network learns what the desired output should be.

4.1 Training Reduced Resolution Segmentation Estimator

Learning process normally requires training images to be loaded and processed one by one. Because we used the GPU for processing, this meant that each image had to be copied from the computers main memory to video memory every time. To increase the speed of the learning process, we loaded all the images and ground truth segmentations into shared variables in video memory at the start of the process, so that copying from the computer's main memory was not necessary for each image.

The images were first split into a train and test set, and shared variables were generated for images and their ground truths separately. 90% of the set designated for this part of the classifier was used for training samples and the remaining 10% for the test set. This resulted in 17 images in the test and 157 images in the train set. Note that the rest of the images are used in the second part of the classifier, and thus should not be used here. Shared variables that held images included the reduced resolution images (one sixteenth of

the original size) and their scaled down versions (the three scales of before mentioned reduced resolution images). The shared variables that held ground truths included the reduced resolution ground truth for each image, where the value for each pixel was the nearest pixel of the original ground truth image using the cv2.INTER_NEAREST interpolation from OpenCV library.

When the model was created and all the images were loaded, we started the training process. This is the process where the classifier sets all the weights and biases that are used in the calculation. Training takes place in multiple epochs, where one epoch represents one iteration of learning process. Final model was trained as described in Section 4.4.

4.2 Training Full Resolution Final Classifier

When training for reduced resolution hand segmentation is finished, it can be run, and the result can be used as input to the network for the full resolution segmentation. Data preparation for the full resolution segmentation is similar to data preparation for reduced resolution segmentation estimation, only in this case we do not reduce the resolution of the image at the start and the segmentation estimation is added to the image as the fourth channel next to the three color channels. Because all channels have to be of the same resolution, segmentation estimation is linearly upscaled using the cv2.INTER_LINEAR interpolation from OpenCV library.

As before, the three image scales are generated and the result is saved in shared variables.

4.3 Data Perturbation

To avoid overfitting and to make the classifier more robust, randomness was added to the images by distorting them.

At the start of each epoch, in both the first and the second part of the classifier, data perturbation was performed. The distortions were done by

zooming by a factor between 0.9 and 1.1, rotating for up to 10 degrees, introducing shear for up to 5 degrees, and translating by up to 20 pixels in either direction (up, down, left, right). Perturbation was performed by performing translation to the center, performing the perturbation, and then translation to the original position. Translation to the center and back is neccessary to ensure that the axis of rotation and shearing are in the center of the image. It is performed using skimage.transform.SimilarityTransform function from scikit-image Python library version 0.10.0. Perturbation values were generated randomly and the images were then cropped to the original size. This was done using skimage.transform.AffineTransform function from the same library.

4.4 Optimizing Parameters

For each layer of the convolutional neural network, filter size and number of feature maps have to be defined. Filter size defines what area around a specific pixel affects the calculation. One filter produces one feature map when convolved over an image. Feature maps are the output of this layer. Number of feature maps depend on the number of filters that are used and it also defines the third dimension of the filter in the next layer.

There is no single best way to determine the optimal filter size and number of feature maps, so these have to be guessed or determined by trial and error. For this reason we trained networks with the same structure, but different parameters and compared the accuracy and running time. We started with a set of initial estimations and based on the results on training dataset, we devised the best set of parameters to test in the next batch.

The model performs its job best if it is as simple as possible, while at the same time complex enough to solve the given task. In order to achieve this, we first found the parameters that produced the best results while ignoring processing time and then simplified the model to reduce processing time while retaining as much accuracy as possible.

In order to find the best parameters for the model, we chose a set of initial parameters that we predicted should produce good results, trained the network and calculated its accuracy. We repeated the process with a set of parameters with larger filter sizes or increased the number of feature maps per layer, trained the model again and calculated the new model accuracy. We repeated this process until increase in filter size or number of feature maps did not improve the accuracy of the model any further.

After finding parameters that produced the best results, we searched for a set of parameters that would still produce usable results, but would be less computationally intensive. We achieved this by deciding on a set of test parameters, running them, analysing the results, and repeating the process. In our testing, majority of tests achieved partial convergence after 100 to 200 epochs, with the number varying between 61 and 446.

To reduce the time required to get results, we devised a procedure that enabled us to perform a greater number of tests in the same amount of time by changing the learning rate according to Equation (4.1). After the accuracy was not improved for a number of epochs as described in Equation (4.2), the new learning rate was calculated. In both cases, n is a sequential number, increased every time the calculation is performed. If the learning rate, was already at the minimal value, the training was stopped. Functions for learning rate calculation, number of epochs until next learning rate adjustment and value for lowest learning rate were defined based on experimental results to produce a model in as few epochs as possible.

$$rate = \left(0.3 + \frac{1 - \frac{1}{1+n}}{4}\right)^n \tag{4.1}$$

$$max(20, 8^{1.3 \cdot n}) \tag{4.2}$$

This process was performed first for the reduced resolution segmentation estimator and then its best results were used to train the full resolution final

classifier. For training the full resolution final classifier a different subset of training and test images was used to avoid overfitting.

Chapter 5

Results

With the described method we were able to achieve a 99.3% accuracy on our labelled dataset. Accuracy is measured as percentage of correctly classified pixels. Chosen threshold is probability of at least 50%. Half of the set was used for each of the two parts of the classifier, where 80% of each half of the images were used for training and the rest for testing. Accuracy is measured as percentage of pixels that were correctly classified as part of a hand or not hand ((true positives + true negatives) / total). Results can be seen in Figures 5.1, 5.2, 5.3 and 5.4.

The parameters where no further improvement was seen for reduced resolution segmentation estimation were 64 feature maps and filter size of 9×9 on each layer. With these parameters we were able to train a network with 98.9% accuracy, which took 0.0842 seconds to process an input image. After parameter optimization we were able to achieve the accuracy of 98.3% at 0.0160 seconds per image, where first layer had 32 feature maps and 3×3 filter size, second layer 32 feature maps and 5×5 filter size, and third layer had 16 feature maps and 7×7 filter size.

Similarly, parameters where no further improvement was seen for final segmentation were 8 feature maps and filter size of 7×7 on each layer. With these parameters we were able to train a network with 99.4% accuracy, which took 0.0923 seconds to process the input image. After parameter optimiza-

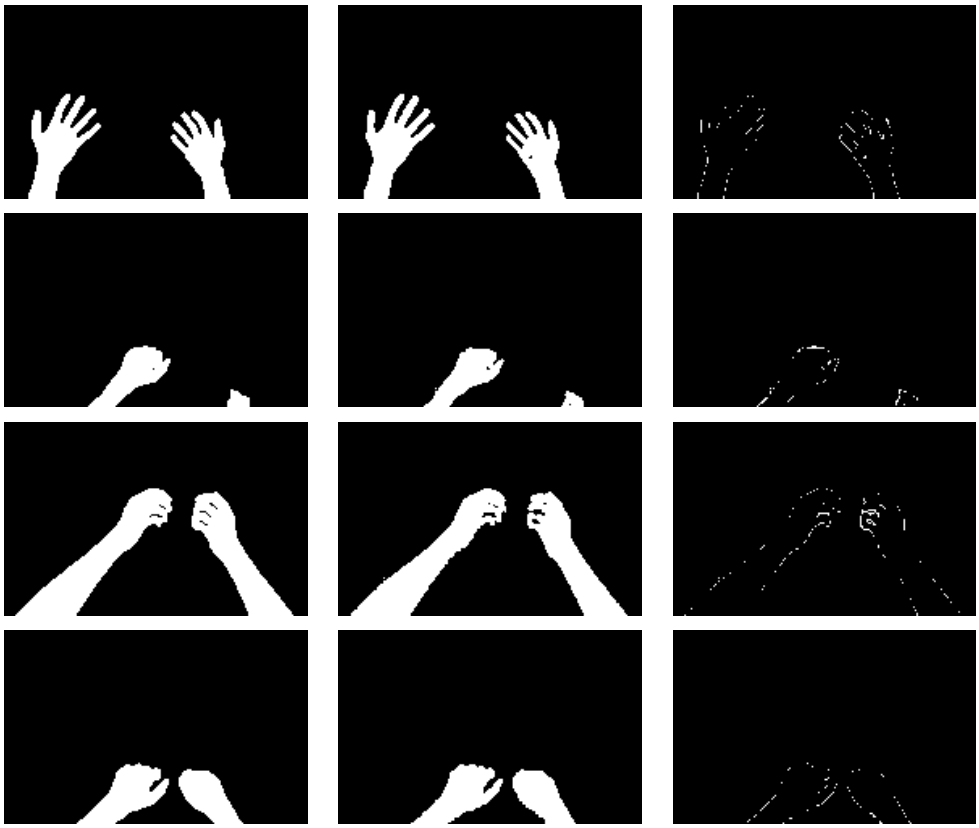

Figure 5.1: Comparison between the ground truth images and the processed segmentation. Results for three different input images are shown: in the first column is the ground truth image, in the second column is the image with the segmentation result, in the third column is the difference between the ground truth and the resulting segmentation

Figure 5.2: The final result of the hand segmentation. Five sample images are shown. In the first column the original image is shown. Second column contains an upscale of the first part of the classifier. Third column shows the final full scale segmentation. The last column contains a composite image between a grayscale of the original image and final segmentation

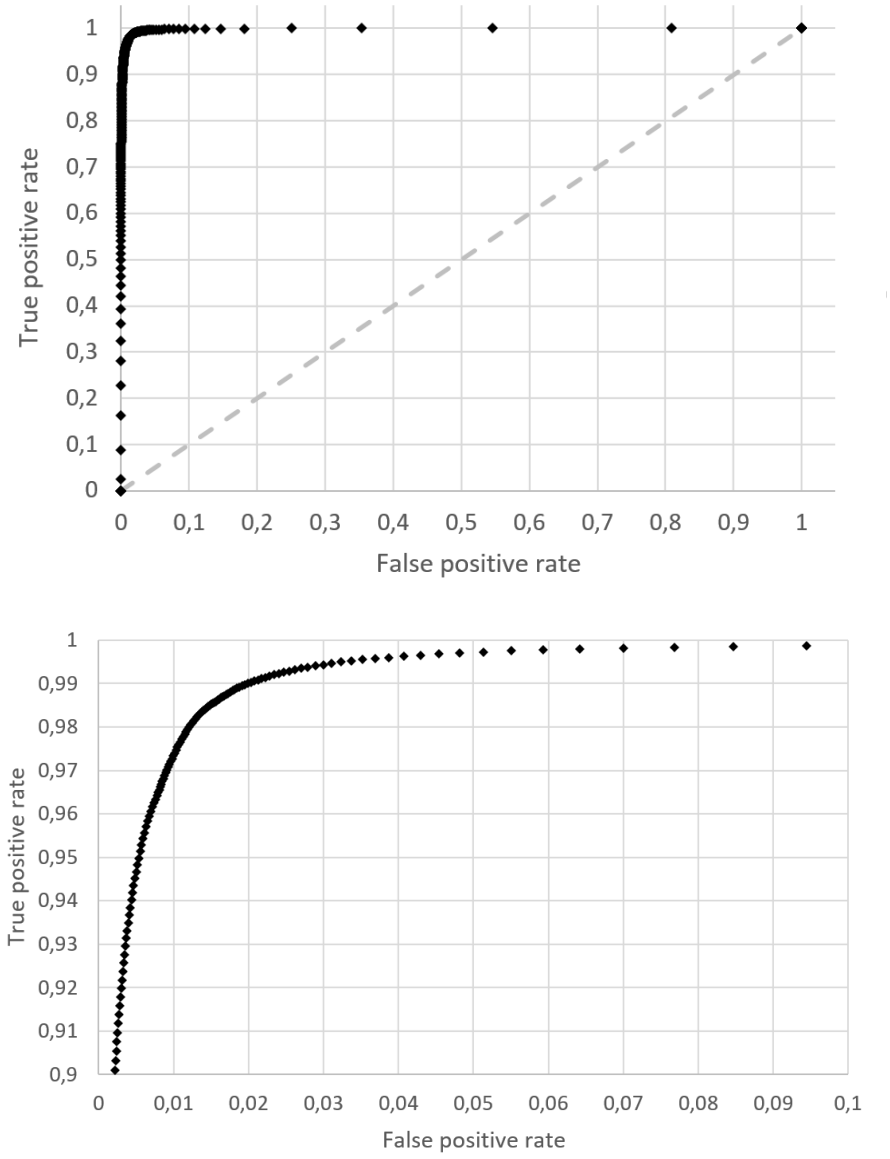

Figure 5.3: Receiver operating characteristic (ROC) curve. This curve is a good measure of the performance of a classifier, as it shows how many pixels are correctly classified when you vary the thresholds. A straight line through the middle would represent a random guess as this would mean the same probability of being right and wrong at any threshold. The value was calculated for 256 thresholds from 0% to 100% probability of class hand in equal steps of 1/256. The second image is a magnification of the curve in the area of false positive rate from 0 to 0.1 and true positive rate from 0.9 to 1.0

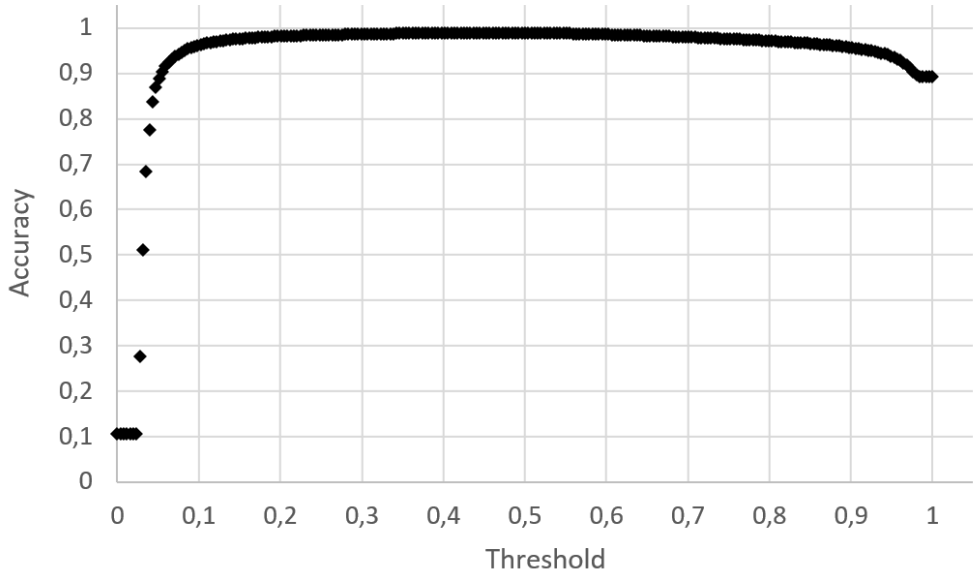

Figure 5.4: Accuracy of the classifier depending on probability threshold. Accuracy is measured as percentage of correctly classified pixels. When choosing a low threshold, regions with low probability are also classified as hand, and thus the classifier outputs more false positives. If the threshold is too high, regions with high probability, but below the threshold are classified as not hand, and thus the classifier outputs more false negatives. Majority class classifier would always classify as not hand, and thus achieve accuracy of 89.4% (as in this case when threshold is equal to 1)

tion we were able to achieve accuracy of 99.3% at 0.0392 seconds per input image, where the first layer had 8 feature maps, the second layer 4 feature maps, and the third layer 1 feature map and 3×3 filter size on all layers.

In the process we noticed that the best results were achieved when number of filters was higher on earlier layers and filter sizes were bigger at later layers. The reduced resolution segmentation estimation already provided very good results, but it still produced some false positives. Because of the lower resolution the edges were not as smooth as desired. as stated above, its accuracy was 98.3%. The full resolution final classifier was in most cases able to improve both the false positives and produce smoother edges.

In total, 98 networks for the reduced resolution segmentation estimation and 95 networks for the full resolution final classifier were trained. Used equipment was a desktop PC with Intel Core i7-3770K CPU with 16GB of system memory and nVidia GeForce 750 GPU with 2GB of video memory running Ubuntu Linux 14.04 LTS. For building the classifiers, we used Theano 0.7, that checked out from git repository on July 28, 2015.

5.1 Comparison to Similar Approaches

5.1.1 Sliding Window

For comparison, we implemented a set of three classifiers based on a sliding window method. In all three cases a sliding window was used and a classifier was trained on different example windows as if they were separate images. Classifier parameters were chosen to produce best results.

We implemented three different classifiers. As we were only interested in accuracy compared to the suggested classifier, processing time was not a priority. All three classifiers barely outperformed a majority class classifier, which had accuracy of 89.4% correctly classified pixels.

Sliding window with a fully connected logistic regression

With the fully connected logistic regression we used a sliding window of 61×61 pixels. On our dataset this classifier achieved accuracy of 90.4%.

Sliding window with a multilayer perceptron

In this example we used a multilayer perceptron (artificial neural network) with 500 neurons in the hidden layer, followed by a fully connected logistic regression layer. Sliding window size of 101×101 pixels was used and accuracy of 90.9% was achieved on our dataset.

Sliding window with a convolutional neural network

In this example we used a full convolutional neural network with two convolutional layers, followed by a hidden layer, followed by a fully connected logistic regression layer. First convolutional layer had 20 feature maps and the second layer had 50 feature maps, while both used 5×5 pixels filter size. The hidden layer used 500 neurons. Sliding window size of 28×28 pixels was used and accuracy of 91.9% was achieved on our dataset.

5.1.2 Convolution on Full Resolution Without Pooling and Upscaling

To verify that splitting the classifier into two parts performs better than a one part classifier, we constructed a new classifier that performed segmentation on full resolution images without first calculating the reduced resolution segmentation estimation.

As a base we used the second part of the suggested classifier and modified it to only use the original image. Without having results from the first part this classifier required more feature maps than the second part of the suggested classifier. For this classifier we used 16 feature maps and filter size of 5×5 pixels on each layer. Because of memory size limit on the used GPU, we were not able to train a more complex classifier, which could produce

better results. Because of the higher resolution and fewer feature maps than the first part of the suggested classifier, we expected a longer calculation time per image and lower accuracy. Processing time per image on this classifier was 0.185 seconds with accuracy of 94.0%.

5.1.3 Upscale Without the Original Image

To verify that the second part of the classifier benefited from re-introducing the original image compared to only having results of the first part, we trained a classifier like the one suggested in this work, but this time we provided the second part of the classifier with only results of the first part. In this experiment processing time was 0.0367 seconds, compared to 0.0392 seconds in the suggested classifier and the accuracy fell from 99.3% to 98.6%. As we can see, processing time is faster, but the second part of the classifier was not able to improve the accuracy much further. The second part of the classifier was able to reduce false positives from the first part, but was unable to improve accuracy on the edges between hand and not hand regions.

5.1.4 Comparison to a color-based classification

The method described in [19] Skin Segmentation Using Color Pixel Classification: Analysis and Comparison achieved accuracy of 81% on our dataset, which is worse than majority class classifier.

Chapter 6

Conclusions

Occlusions are crucial for understanding the position of objects. Their exact detection and correct rendering contribute to the feeling that an object is a part of the world around the user. In this thesis we introduced the problem of occlusions in the context of augmented reality and proposed a solution for hand segmentation in hopes of being able to better blend virtual and real world while wearing augmented reality glasses. The proposed method uses convolutional neural networks and performs well on our labelled dataset, although a larger dataset with more diverse users and lighting conditions should be tested to further support the results.

The solution that we presented still has high memory and processing power requirements for training, so training on larger datasets could be a challenge. The parameters presented in this work were obtained empirically. When searching for best parameters, we faced limitations of our hardware, as we either run out of video memory or the processing time was too long.

There is still a lot of room for improvements, especially in finding a better way to detect exact edges. One of the possible solutions is using superpixels, which we tried and which gave good results, but because of increased processing time of our implementation we decided not to use them.

Bibliography

[1] Al-Tairi, Zaher Hamid, Rahmita Wirza Rahmat, Iqbal Saripan, and Puteri Suhaiza Sulaiman, "Skin Segmentation Using YUV and RGB Color Spaces." Journal of Information Processing Systems 10, no. 2, 2014, pp. 283-299.

[2] Kawulok, Michal, Jakub Nalepa, and Jolanta Kawulok, "Skin detection and segmentation in color images." In Advances in Low-Level Color Image Processing, Springer, 2014, pp. 329-366.

[3] Baraldi, Lorenzo, Francesco Paci, Giuseppe Serra, Luca Benini, and Rita Cucchiara, "Gesture recognition in ego-centric videos using dense trajectories and hand segmentation." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 688-693.

[4] Huang, Yu-Jen, Mark Bolas, and Evan A. Suma, "Fusing depth, color, and skeleton data for enhanced real-time hand segmentation." In Proceedings of the 1st symposium on spatial user interaction, ACM, 2013, pp. 85-85.

[5] Lew, Yuan Pok, Ramli, Abd Rahman, Koaym Su Yeong, Ali, Roslizah, and Prakash, Veeraraghavan, "A hand segmentation scheme using clustering technique in homogeneous background." In Student Conference on Research and Development, IEEE 2002, pp. 305-308.

[6] Mittal, Arpit, Andrew Zisserman, and Philip HS Torr, "Hand detection using multiple proposals." In Proceedings of the British Machine Vision Conference, 2011, pp. 1-11.

[7] Wang, Sun-Chong. "Interdisciplinary computing in Java programming". Vol. 743. Springer, 2012.

[8] Azevedo, Frederico, Carvalho, Ludmila, Grinberg, Lea, Farfel, José Marcelo, Ferretti, Renata, Leite, Renata, Lent, Roberto, and Herculano-Houzel, Suzana, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain." Journal of Comparative Neurology 513(5), 2009, 532-541.

[9] Li, Fei-Fei, Karpathy, Andrej, Johnson, Justin. CS231n: Convolutional Neural Networks for Visual Recognition. Retrieved from http://cs231n.github.io/ (May 11 2016).

[10] Szymik, Brett, A Nervous Journey. ASU – Ask A Biologist. Retrieved from http://askabiologist.asu.edu/neuron-anatomy (May 11 2016).

[11] McCulloch, Warren, and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5(4), 1943, 115-133.

[12] Long, Jonathan, Evan Shelhamer, and Trevor Darrell, "Fully convolutional networks for semantic segmentation." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431-3440.

[13] Ciresan, Dan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber, "Flexible, high performance convolutional neural networks for image classification." In Proceedings of the International Joint Conference on Artificial Intelligence, no. 1, 2011, pp. 1237.

[14] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio, "Deep sparse rectifier neural networks." In Proceedings of the International Conference on Artificial Intelligence and Statistics, 2011, pp. 315-323.

[15] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton, "Deep learning." Nature 521, no. 7553, 2015, pp. 436-444.

[16] Maas, Andrew, Awni Hannun, and Andrew Ng, "Rectifier nonlinearities improve neural network acoustic models." In Proceedings of the International Conference on Machine Learning Workshop on Deep Learning for Audio, Speech, and Language Processing, 2013.

[17] Huang, Zhen, Jinyu Li, Chao Weng, and Chin-Hui Lee, "Beyond cross-entropy: towards better frame-level objective functions for deep neural network training in automatic speech recognition." In Proceedings of the InterSpeech Conference, 2014, pp. 1214-1218.

[18] Dauphin, Yann, Harm de Vries, Junyoung Chung, and Yoshua Bengio, "RMSProp and equilibrated adaptive learning rates for non-convex optimization." Preprint arXiv:1502.04390, 2015.

[19] Phung, Son Lam, Bouzerdoum, Abdesselam, and Chai, Douglas, "Skin segmentation using color pixel classification: analysis and comparison." IEEE Transactions on Pattern Analysis and Machine Intelligence 27(1), 2005, 148-154.