MASTER THESIS

COMPUTING SCIENCE

Adversarial background augmentation improves object localisation using convolutional neural networks

Author:
Ing. HARM BERNTSEN

Supervisor Radboud University:
Prof. Dr. TOM HESKES

Supervisors Nedap:
DAAN VAN BEEK, MSc.
Dr. WOUTER KUIJPER

AUGUST 2015

Abstract

Deep convolutional neural networks have recently achieved state of the art performance on image recognition tasks. Instead of manually developing algorithms to extract information from the image pixels, artificial neural networks learn to do this by training on images combined with the expected output of the network. Because the neural network automatically learns to construct the desired output from the input, it is hard to predict how the network will perform on unseen data.

We employ a neural network to localise a specific object. We have a model to generate sufficient images of our object to localise but we do not have a model of the backgrounds the network will encounter after training. We found that the background around the object matters. A single type of background (e.g. completely white or noise) around the object made the neural network dependant on that type of background. Due to this dependence, the neural network did not localise the object correctly when it saw other backgrounds. As a solution, we generate adversarial backgrounds to train the neural network. Those adversarial backgrounds exploit weaknesses of the neural network. By augmenting the training data with the adversarial backgrounds the neural network's performance was improved.

Contents

1 | Introduction

The human visual system is good at localising objects, for computers this is still not a trivial task. Inspired by the neurons in the human brain, Artificial Neural Networks (ANNs) were invented to bring the learning power of biological life to computers. Though initial models of neurons were developed in the mid-twentieth century [MP43, Ros58] it was not until the last decade that ANNs have become effective. Better network designs, large datasets and the vast increase in computational capabilities of Graphics Processing Units (GPUs) improved the performance of ANNs enough to surpass the manually designed computer vision algorithms. Those algorithms use handcrafted rules to transform the data to another representation, a process called *feature extraction*. Handcrafted rules extract features like edges and gradients from images that can then be used for further processing to get the desired outcome. The algorithm is then precisely designed to work well for that task. When the task changes, one needs to redesign the whole algorithm. With ANNs, there is no need to precisely define how to extract relevant features from the data. This is done by a learning procedure. In the process of *supervised learning*, a significant number of inputs and desired outputs from a *training set* are passed through the network. With enough data, the network is able to learn by itself what features are relevant to produce the desired output.

The need for handcrafting designs is not completely gone with ANNs. The structure of the network and the learning procedure still need to be adapted to the problem at hand. Because the features themselves are learned, the design task moves to a fundamentally higher level of abstraction. Instead of designing algorithms to extract features, the network structure and learning procedure need to be adapted. Existing network structures and learning parameters can be used as a starting point and adapted to similar tasks. For example, Convolutional Neural Networks (ConvNets) are a special type of ANNs and have shown to work well on images, speech and text [KSH12, DYDA12, HZRS15, CW08].

A common task in computer vision is object localisation. Given an image, the task is to annotate the image with the location of an object. ConvNets have been successfully applied to the task of object localisation, e.g. [GDDM14, SEZ$^+$14, SZ15, STE13]. A ConvNet can be trained end-to-end; that is directly from the raw pixels to the desired result. For object localisation, the desired result could be designed as a binary mask [STE13]. The binary mask annotates the image with the information of which pixels belong to the object and which pixels do not belong to the object. The training set needs to be sufficiently large and representative for the context the trained network will be used in. The training set contains the images, associated with their correct binary mask, the *ground truth*.

As ANNs learn to detect the relevant features for the task, the dataset they learn from needs to be carefully constructed. In an object localisation task, it is important that the neural network learns to localise the object itself without depending on the background. If the background in the training set is always white, the network may learn to rely on the white pixels to decide which pixels are part of the object to localise. The network will then generalise poorly to other contexts because there might not be a white background around the object.

In this thesis, a novel method is developed which adjusts the background of objects in the training set to improve the performance of a ConvNet on not previously seen backgrounds. In our method, the neural network is trained on a trivial background that is easy to obtain. The training set is expanded by generating backgrounds specifically for the trained neural network, eliminating the need to manually design a model for the background.

The remainder of this chapter will briefly describe the motivation behind this thesis, introduce our problem statement and describe our proposed solution at a conceptual level. Chapter 2 discusses related work regarding object localisation in images and the generation of training data. Section 3.1 explains the basics of Deep Neural Networks (DNNs). Readers familiar with DNNs can safely skip this section. Section 3.2 explains the details of the optimisation method that will be used in Chapter 4. Chapter 4 extends the proposed method by describing precisely which choices we made. The chapter also describes our experimental set-up: how we constructed our dataset, which network structure we used and how we generate backgrounds. Chapter 5 describes the experiment around our proposed solution and discusses the results that we obtained. This chapter also concludes our work and points to ideas for further research.

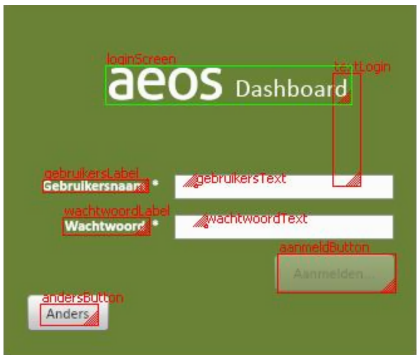


Figure 1.1: The login screen of AEOS with NTA's annotations.

1.1 Motivation

This thesis was written at Nedap Security Management. This business unit of Nedap develops AEOS: a security solution that combines access control, video management, intrusion detection and locker management. This business unit also develops an internal testing tool, called Nedap Testing Automation (NTA), to test software on the user interface level. NTA treats software as a black-box: the test tool only gets a screenshot of the user interface as input. Figure 1.1 shows the AEOS login screen with NTA's annotations. To test the login screen, NTA needs to send mouse clicks to the right locations. The locations are found by searching for known elements on the screen. This search is done by scanning the screen for predefined images of each element. The creation of those images is labour intensive. With subtle visual changes, like a change in the anti-aliasing filter, NTA will not recognise the elements on the screen any more. Nedap is interested in the application of deep learning to improve NTA's localisation of elements.

As a proof of concept, we limited ourselves to localising buttons within a user interface. We initially developed a model of a typical user interface and generated a dataset where the buttons had a very broad range of styles. A neural network was trained and we found that it was able to localise the buttons within our model. Though our model included variation in the background, the neural network failed to localise buttons correctly on unseen backgrounds. This brought us to the issue: how to get the right set of backgrounds around the buttons?

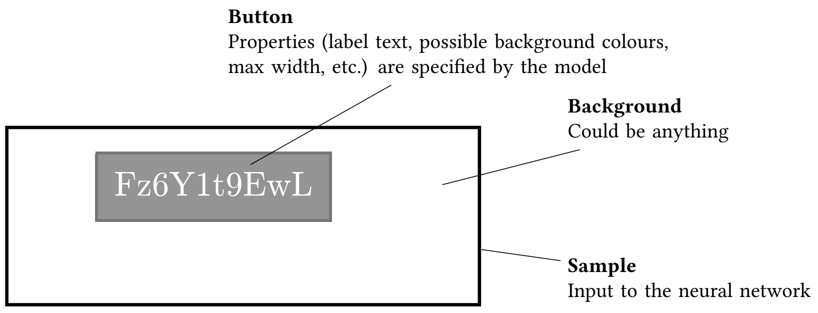

Button
Properties (label text, possible background colours, max width, etc.) are specified by the model
Background
Could be anything
Fz6Y1t9EwL
Sample
Input to the neural network

Figure 1.2: Overview of the button localisation task.

1.2 Problem statement

In this thesis we want to localise a button from a single web application. Within an application, buttons usually have the same design (e.g. always square and a border around it) but still vary in properties (e.g. like their text content, size and colour). We created a model of a button that represents one specific button style. The idea behind this model is that it captures enough variation to let the neural network localise all the buttons that comply to the design specified by our model.

Since ConvNets have shown to work well on images, we want to use their learning power to learn to localise buttons in a screenshot. We do not want to resort to designing handcrafted algorithms. With a model of the button, we can generate a sufficient number of buttons for the training set. When the trained network is applied in practice, it will have to localise the buttons of the specific web application the model was designed for. When it encounters such a button, it has already seen a similar one in the training set. The network should have learned to correctly localise that button. On the other hand, we do not have a model that describes the backgrounds the neural network will encounter. The environment around the button could be anything. Training the network on every possible background is infeasible as the number of possible backgrounds scales exponentially with the number of pixels. Somehow we need to make sure the neural network can learn to recognise the button without relying on the background around it. Figure 1.2 shows an overview of the button localisation task.

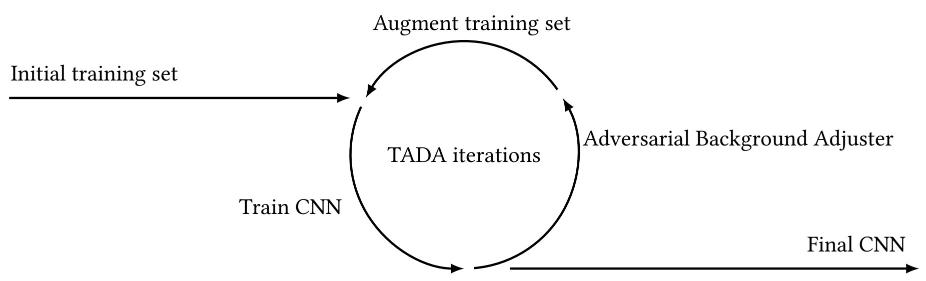


Figure 1.3: Overview of the solution approach.

1.3 Solution approach

We introduce a novel method which can still train a ConvNet to localise buttons without having to design a model of the backgrounds the neural network will encounter. First, an initial training set is constructed with buttons in front of a background that is easy to generate. A neural network is trained using this training set. The neural network should work well on the type of background that was used in the initial training set but might not generalise well to other types of backgrounds. To improve generalisation to other backgrounds, we now apply a procedure which we will refer to as the Adversarial Background Adjuster (ABA) procedure. The ABA procedure uses a subset of the samples from the initial training set and produces adjusted samples. It may only adjust the background around the buttons, the buttons themselves have to remain exactly the same. The trained neural network is used to adjust the backgrounds such that the ConvNet predicts the location as poorly as possible. The adjusted samples can cause the ConvNet to recognise parts of the background as button or vice versa misclassify parts of the button as background. The adjusted samples simply exploit weaknesses of the neural network's way of localising buttons. We augment the training set with the adjusted samples and continue training the network, giving it the ability to correct itself such that it can withstand the weaknesses that were exploited. Augmentation of the training data has been shown to improve the generalisation of neural networks. Those data transformations are often simple, e.g. taking multiple crops of one training image [KSH12, ZF14]. Our method augments the data with targeted, adversarial backgrounds.

Figure 1.3 shows an overview of the solution approach. With Train ADjust Augment (TADA) iterations we refer to an iteration of training the ConvNet, running the ABA and then the augmentation of the dataset. After sufficiently many TADA iterations, we expect that the network has learned to use the features of the button itself rather than the background because the features of the button itself are the only constant element throughout the whole training set. Although the method was tested using buttons, the design is generic. One simply needs an initial training set with sufficient input images with their ground truth of the object to localise. In this thesis, all the samples in the training set contain one button that needs to be localised. The details of our adversarial data augmentation method are described in Chapter 4.

2 | Related Work

2.1 Computer vision

Localising buttons from pixel information on the screen is not new. For example, in the Lens system [GWS], a hierarchy is built of on-screen elements to capture the structure of a user interface. They detect borders of user interface elements by searching for groups of contiguous pixels that share the same colour. Another system is Prefab [DF10]. They modelled a button by specifying how the four corners of a specific style of button look. In the input image, the corners are detected and the ones that form a square are recognised as a button. Both methods use hand-coded algorithms to detect features unique to a button. Directly working with pixels is feasible for recognising buttons from a screenshot. A button is generated through code which can produce the exact same button every time. In cases of photos it is hard to directly go from pixels to a location. First extracting relevant higher level features like the location of lines makes the task easier. Reconstructing the location of a button by using those higher level features makes the localisation task easier.

For natural image recognition, more generic algorithms have been developed. In computer vision challenges like the Pascal Visual Object Classes (VOC) [EvW+09] or ImageNet [RDS+15], teams have to compete in classification and localisation tasks regarding natural images. In the classification task, the team has to predict the category of the whole image. In such challenges, feature detectors, like Histograms of Oriented Gradients (HOG) [KHK08] or Scale Invariant Feature Transform (SIFT) [Low04], used to be manually designed to extract generic low-level features from images. Algorithms like the Deformable Parts Model (DPM) then use those low-level features for detecting the object [FGMR10].

There was a turning point when the SuperVision team showed record breaking performance in the ImageNet 2012 challenge [RDS+15]. In both the classification and the localisation task their ConvNet outperformed their competitors who were not using ConvNets. The network structure of SuperVision is referred to as AlexNet and described in [KSH12]. Unfortunately, they do not describe how they approached the localisation task. Other teams have shown how a network based on AlexNet can be used for localisation, e.g. [SEZ+14, ESTA14, STE13, SVZ14, OBLS15, WZL+14, OBLS14]. Subsequent editions of ImageNet have also been won by ConvNets.

In the previous century, ConvNets have already shown excellent performance on recognising handwritten digits [LBBH98]. Despite this success, there was little interest in neural networks by the computer-vision communities [LBH15]. The interest of research into ConvNets was revived in 2012 when Krizhevsky et al. [KSH12] outperformed their competitors in the ImageNet challenge [RDS+15]. Reasons given for the revival of ConvNets are the computational power of GPUs, the availability of large datasets and new training methods [RDS+15, TYRW14, How13]. This increase in popularity also resulted in software implementations like cuda-convnet2 [KSH12], Caffe [JSD+14], Pylearn2 [GWFL+13] (based on Theano [BBB+10]) and Torch [CKF11]. The availability of these implementations again stimulates research into deep learning.

The computations within a ConvNet involve a lot of large matrix operations. The performance on modern Central Processing Units (CPUs) is limited by the bandwidth between the CPU and memory. GPUs have a much higher memory bandwidth than regular CPUs. They also feature a large number of Single Instruction Multiple Data (SIMD) cores that can execute operations concurrently on large data vectors. GPUs are widely used by consumers for video games. This huge market drives down the prices, making them affordable for research teams as well. With the introduction of General Purpose computation on GPUs (GPGPU) frameworks like CUDA [Cor15], the high memory bandwidth and massive parallelisation capabilities of GPUs can be exploited to train neural networks. We compared the computation time of a forward and backward pass on our network (Table 4.2). We only used standard consumer hardware. A forward and backward pass on our Intel Core i7-4770 CPU was 35 times slower than our NVIDIA GeForce GTX 980 GPU. Training a ConvNet can take several days on a GPU [KSH12, NYC15, TYRW14], without GPU acceleration those experiments would take a very long time on consumer hardware.

With the ImageNet challenge, large image datasets became available [RDS+15]. ConvNets require a lot of labelled data to be trained well. ImageNet [RDS+15] provides 1.2 million training images, divided into 1000 classes. Large training sets are beneficial for ConvNets. Data augmentation methods are often applied to improve the generalisation of the ConvNet [SZ15, KSH12, CMS12, LCY13]. For example, [KSH12] multiplied the training set by 10 via mirroring and taking several crops of the training images.

New network innovations were also important to obtain the results of [KSH12]. The ReLU activation function improved the speed of learning ConvNets by several times [KSH12]. Another technique is Dropout [Hin14]. Dropout randomly disables neurons in a hidden layer during the training of the network. This stops neurons from relying too much on other neurons. When the network has been trained, all the neurons are active. That network can be seen as the average of a lot of different smaller networks. This has shown to significantly reduce overfitting.

2.1.1 Object localisation

There are various ways to annotate the location of objects in an image. One way is to do full-scene labelling. In full-scene labelling, every pixel of the input image is categorized (e.g. [BHC15, HAGM14, FCNL12]). The localisation task in the ImageNet challenge does not require such fine-grained segmentation: a bounding box around important objects is sufficient.

There are several approaches to localise objects in images via neural networks. One way is to train a neural network for classification and employ it in a sliding window fashion on the input image. A classifier by itself only identifies the class of the input image, it does not give a location. By applying that classifier at a lot of positions and scales of the input image, locations of objects can be found (e.g. [SEZ+14, OBLS14, OBLS15]). In other papers, authors first extracted possible regions of interest from an image using a handcrafted algorithm. A ConvNet trained for classification is then applied on each proposed region. This is done in the Regions with CNN features (R-CNN) algorithm [GDDM14] and also adopted by others [OLZ+14, SLJ+15, ZDGD14]. They classify each region proposal and post process those classifications to extract locations and classes of objects. A ConvNet can also be directly used to predict locations. One way is to let the network predict a 4D vector, containing the location and size of the bounding box (e.g. [SZ15, SEZ+14, SREA14, ESTA14, WZL+14]). Instead of 4D vectors, the network can also output a binary output mask of the location of the object [STE13]. Other approaches have shown that ConvNets trained for classification were able to do localisation as well via network visualisation methods [SVZ14, ZF14, ZKL+14].

2.2 Data generation and augmentation

Augmenting the training set by generating data has shown to improve the neural network's performance [SZ15, KSH12, CMS12, LCY13]. This is usually done by simply transforming the training images by cropping and mirroring them. Another way to generate data is to sample images from models of the object to recognise. In [SS14, SQLG15] the authors generate data using 3D models from real world objects. Backgrounds around the objects to recognise are sampled from photograph datasets.

Training data can also be generated by using a trained neural network. Szegedy et al. [SZS+14] discovered that a DNN can completely misclassify an image with high confidence by applying a hardly perceptible adjustment to the input image. They applied an optimisation procedure that optimises the input image's pixels by maximising the classification error while keeping the adjustments of the image imperceptible for humans. Those *adversarial samples* even worked on networks that were trained on a different dataset. Given that imperceptible changes to the image can have a big effect on the classifier's result shows that DNNs have counter-intuitive ways to recognise images. Nguyen et al. [NYC15] tested other types of adversarial images. They generated images that are completely unrecognisable to humans but yield high confidence classifications by DNNs.

Attempts have been made to make DNNs more resistant to adversarial samples. One way is to train the network on adversarial and clean samples. Doing this has shown to make the network more resistant [GSS15, NYC15] against adversarial samples. The resulting networks were not completely resistant to adversarial samples: the authors were still able to generate new adversarial samples. As [GSS15] reports, the adversarial training procedure can be seen as minimizing the worst case error when the data is perturbed by an adversary. Other attempts like [GR15] modify the network structure. Although ways were found to make DNN more resistant to adversarial samples, the resulting networks are still sensitive to new adversarial samples.

Instead of using existing datasets to use as backgrounds like in [SS14, SQLG15], we generate the backgrounds ourselves. The buttons we would like to recognise live in a digital environment instead of a natural environment. The screenshots from user interfaces simply do not look like photos from the real world. It is not trivial to manually develop a model of this environment to generate backgrounds. The absence of a model of the background is resolved by employing the power of adversarial samples to automatically produce backgrounds. The adversarial backgrounds are specifically targeted towards our trained neural network. In contrast to [GSS15, NYC15, GR15], we do not give the adversary complete control over all the pixels. In our situation the adversary may only adjust pixels that are not part of the button that needs to be localised.

3 | Theory

3.1 Deep Neural Networks

DNNs are based on a lot of simple processors, called neurons. They are inspired by a biological brain in the hope to bring their learning capability to computers. We will briefly present some theory for a basic understanding of their workings. More details can be found in resources about neural network theory, e.g. [LBH15, Hay99, Sch15].

Figure 3.1 shows a model of an artificial neuron. Each input x_i of the artificial neuron is associated with a weight w_i that is learned when the network is trained. The output of an artificial neuron is defined as:

$$y = f(b + \sum_i x_i w_i)$$

where b is a bias term that is also learned during training. The function f is a nonlinear activation function that limits the output y of the neuron. A common activation function is the ReLU which is defined as $f(z) = \max(0, z)$. In DNNs, the neurons are stacked in multiple layers where each neuron is connected to all the outputs of the neurons in the previous layer. Those layers are referred to as *fully connected layers*. Figure 3.2 shows a model of such network. In our case, the inputs to the neural network are the values of individual pixels of the input image. In the hidden layers, neurons convert the input to a different representation. They learn to recognise specific features of the input and progressively convert those features to higher level representations to construct the output of the network.

Each neuron has its own set of weights. The weights of are learnt via backpropagation with an optimisation method, in our case Stochastic Gradient Descent (SGD). Each weight is updated by calculating the partial derivative of the weights with respect to the loss function. The loss function compares the ground truth from the training data with the network's output. The optimiser changes each weight in a direction that minimises the loss.

Convolutional neural networks

A problem with fully connected layers and image data is that they are very sensitive to translation of the input. The weights are directly linked to pixels at a specific position. When a DNN has learned to recognise a certain feature at one place in the

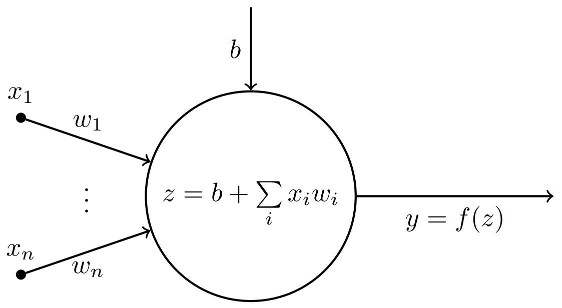


Figure 3.1: Model of an artificial neuron. An input x_i gets multiplied with its weight w_i. The results of those multiplications are summed. The bias b is added to move the decision boundary of the activation function f.

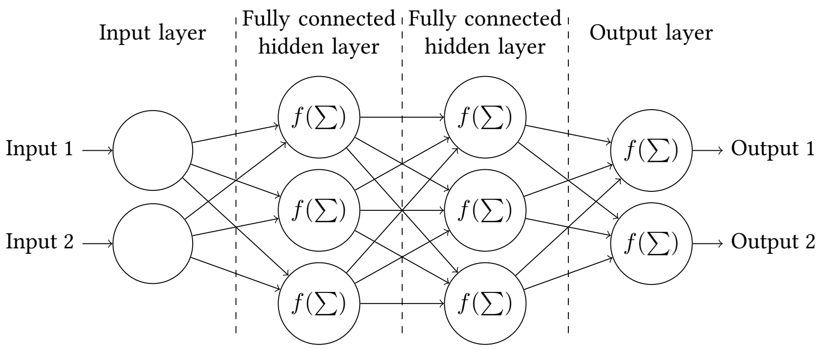


Figure 3.2: Model of a DNN. The number of neurons in the hidden layers does not have to be the same. The activation function is the same within the whole layer but can differ between layers. This figure is based on [Fau06], licensed under the Creative Commons 2.5 license.

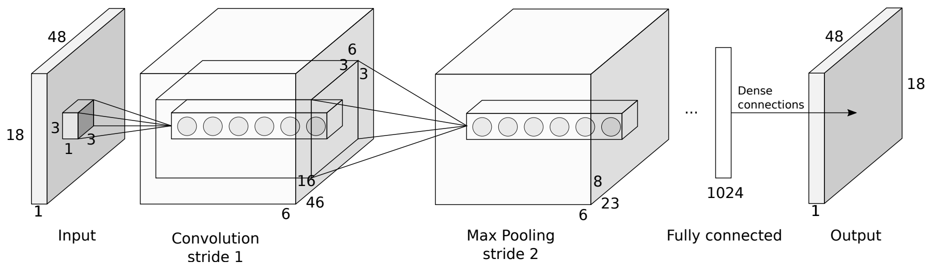


Figure 3.3: Visualisation of a ConvNet (inspired by [Kar15]). The input is an 18×48 pixel greyscale image, giving it a depth of 1. The convolutional layer has 6 filters, connected to the 3×3 receptive field. The output of the convolutional layer is calculated by sliding the bar of filters across the whole image, giving a new $16 \times 46 \times 6$ output. The max pooling layer reduces the size of the data. Here we show a pooling layer with a 3×3 window. The max-pooling layer operates on each 16×46 slice independently and only stores the maximum value of its input window.

input image, it will need to relearn to recognise that same feature at other positions in the input image. ConvNets have convolutional layers to tackle this problem. The convolutional layers are designed to provide invariance under translation, scaling and other transformations of the input.

Figure 3.3 shows a model of a ConvNet. Convolutional layers have a set of filters. Each filter has a receptive field where a single neuron is connected to the all the inputs. In this model, the receptive field of the convolutional layer sees an $3 \times 3 \times 1$ input patch. The filter is applied at an interval of *stride* inputs in the input image. The stride determines the height and the width of the output of the layer. In the first convolutional layer, filters will detect low level features like lines. The next layer sees at which position lines did occur and uses that information to construct higher level features like parts of objects. Fully connected layers at the end combine the information of all the neurons in the previous convolutional layer to produce the final output.

3.2 Simulated annealing

Our algorithm needs an optimisation procedure to optimise the backgrounds of the samples in the dataset. Although we have chosen to use simulated annealing, the optimisation procedure can be easily switched out for other optimisation techniques. Simulated annealing is a method to search for the global energy minimum of a function [KGV83]. It is based on an analogy from physics where a material is heated and slowly brought to a lower temperature. The heat excites all the atoms in the material and at that point the atoms have equal probability of being in any state. During the cooling state they settle in an optimal position, strengthening the crystal. The simulated annealing method is based on four components:

- A concise description of the state.
- A function to randomly adjust the state.

- A function to measure the poorness of the state: the loss function.
- An annealing schedule.

Simulated annealing tries to find the state for which the loss function gives the lowest output. It does so by iterative improvement. The algorithm starts with an initial state, randomly adjusts it and tests whether the loss of that new state has been improved. It can choose to do a new iteration on this state by randomly adjusting this newly obtained state again. If it would always do this, the algorithm might get stuck in a local optimum. Simulated annealing tries to prevent getting stuck in local optima via a temperature, based on the analogy from physics. The temperature starts off high and is lowered according to the annealing schedule in each iteration. When the temperature is high, it will accept almost every new state. The lower the temperature, the lower the probability that a worse state is accepted. In each step of the algorithm, the loss L of the new state is compared with the previous loss. If it is worse (so a higher loss), the new state is accepted when $\exp(\frac{-\Delta L}{T}) \geq \textsc{UniformRandomReal}(0, 1)$ holds. If the loss is lower, it is always accepted.

The annealing schedule determines the sequence of temperatures and the number of iterations that will be performed. In this thesis we used an exponential cooling schedule. In this schedule, the temperature range is specified and used to recalculate the temperature each iteration:

$$T_{\max} \cdot \exp\left(\frac{-\ln(\frac{T_{\max}}{T_{\min}}) \cdot \mathit{iteration}}{\max(\mathit{iteration})}\right)$$

4 | Architecture

This chapter describes the details of our adversarial data augmentation architecture that was introduced on a conceptual level in Section 1.3. A detailed overview of the architecture is shown in Figure 4.1. First, an initial training set is constructed with an easy to generate background. With this training set, we enter the 1st TADA iteration. The first step is to train a ConvNet on this initial training set. The ConvNet has then learnt to localise the buttons on the type of background it was trained on. This trained network can be considered as the final network or can be used to continue the TADA iteration. In the latter case the ABA tries to optimise samples such that the network's performance is as poor as possible. Those samples maximally fool the network because the background is constructed such that they exploit weaknesses in how the network has learnt to localise buttons. We expect that specifically learning with those samples helps the network to better focus on actual features of the buttons rather than the backgrounds.

In the following sections the architecture is explained in detail.

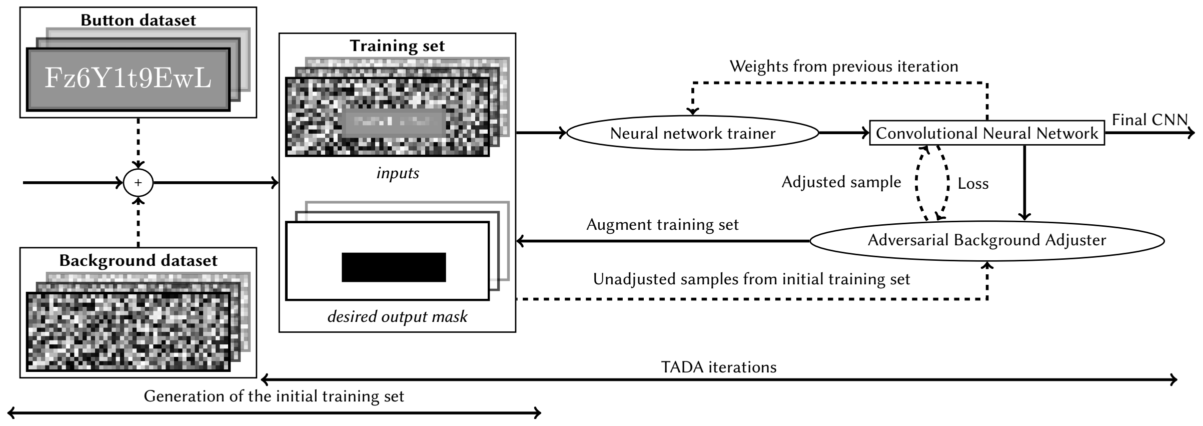

Button dataset
Fz6Y1t9EwL
+
Background dataset
Training set
inputs
desired output mask
Weights from previous iteration
Neural network trainer
Convolutional Neural Network
Final CNN
Adjusted sample
Loss
Augment training set
Adversarial Background Adjuster
Unadjusted samples from initial training set
TADA iterations
Generation of the initial training set

Figure 4.1: Schematic overview of the adversarial data augmentation architecture.

4.1 Generation of the initial training set

The generation of the initial training set involves two main components: the objects to localise and the background to put around it. Both the background and button are converted to greyscale. The backgrounds have a resolution of 48×18 pixels. The reason for using greyscale (opposed to colour) at a low resolution is to decrease the computation time, allowing more experiments to be done in a shorter time frame.

We place the button in front of the background and encode the location of the button as a binary mask. The mask indicates which pixels belong to the object to localise. The button is placed at a random location on the background with two restrictions: the whole button has to fit on that location (no cropping) and the 2 pixel radius around the centre of the background has to be covered by the button. We do this because hypothetically, the ABA could start drawing actual buttons on the background. The positioning of the button to localise is a unique property that allows the network to distinguish the button to localise from potential buttons generated by the ABA. In the case that the ABA has drawn actual buttons on the background, those buttons still miss the unique property. Without this restriction the ABA has the freedom to draw buttons that are indistinguishable from the original button to localise. We would then punish the network for recognising an actual button. The unique property simply prevents this situation from occurring because the network should only detect buttons with the unique property.

With the model of the object we can create a training set of any arbitrary size. We do not reuse buttons or backgrounds in the initial training set. In initial experiments, we found that 38000 frames of a NOS (Nederlandse Omroep Stichting, English: Dutch Broadcast Foundation) news broadcast as background was sufficient to obtain a network that performed well on another NOS news broadcast as background. Due to this result, the length of the initial training set was set to 38000 samples with a white background. We did not test with other dataset sizes.

4.1.1 Background dataset

The initial training set is constructed with a background that is completely white. After the final ConvNet has been trained, it could encounter any background. With the completely white background, the neural network can easily localise buttons by classifying non-white areas to be part of the button. We rely on the ABA to properly optimise the backgrounds, preventing the need to construct a representative model of the backgrounds the neural network will encounter.

4.1.2 Button dataset

The buttons were created by generating HyperText Markup Language (HTML) buttons [CM00, Section 4.10.6], styled via Cascading Style Sheet (CSS) [Ete11] code. This particular combination of technologies was chosen because they allowed us to easily model buttons. In the general case, the object to recognise can be generated in any way as long as a ground truth mask is available.

CSS property	Value
`background`	UniformRandomInteger$(50, 200)$ for each RGB colour channel
`border-color`	$0.8 \cdot$ `background`
(text) `color`	`#FFF`
`border-width`	2px
`min-width`	UniformRandomInteger$(3, 10) \cdot 10$px
`padding`	UniformRandomChoice$(\{5$px$, 10$px$\})$
`font-family`	UniformRandomChoice$(\{$`Andale Mono`, `Arial`, `Arial Black`, `cmmi10`, `cmr10`, `Comic Sans MS`, `DejaVu Sans`, `eufm10`, `Impact`$\})$

Table 4.1: CSS properties used to generate buttons.

The buttons are characterised by having a solid background with a slightly darker border around it, see Figure 4.1. Table 4.1 shows the CSS properties that were used with their possible values. Variation of the style of the button was done to make the localisation task less trivial. The button is filled with a random text by randomly sampling from a uniform distribution of letters (a-z, upper-case and lower-case), digits and spaces. The length of the random text was at least 2 characters and at most 10 characters. The length of the text also influences the width of the button. Buttons were scaled to 12.5% of their size, rounding to complete pixels. The downscaling is needed to let the buttons fit within the 48×18 pixel input resolution of the neural network. The average button covers 22% of this area.

4.2 TADA iterations

4.2.1 Neural network

The task of the neural network is to predict the location of the button given the input image. The neural network performs this end-to-end, so the neural network takes the raw pixels of the image and converts it into an equally sized mask. Similar to [STE13], AlexNet [KSH12] was used as a starting point for the neural network structure. Although newer network structures exist (e.g. [ZF14, SZ15, HZRS15]), AlexNet has been widely studied and has shown to work well in many situations.

We have chosen to use the open-source Caffe [JSD+14] project from the Berkeley Vision and Learning Center (BVLC). Caffe is designed as a generic neural network library. The neural network and training procedure are specified in text files, based on the available parameters and configuration options Caffe offers. Caffe can be accessed via a command-line interface and the Python and C++ programming languages to interact with it. Caffe checks the configuration and trains a neural network accordingly using optimized code. This prevents bugs that could be introduced by writing custom code. Caffe includes a model of AlexNet, called BVLC AlexNet as it differs slightly from the original AlexNet [SDK15].

Table 4.2 shows an overview of the layers in the original and the modified network we used. For more details about the network structure, see [KSH12]. The original AlexNet network was designed for assigning a single class to an image. We optimised the network for our localisation task, the most notable changes are:

- The input resolution was lowered from $3 \times 244 \times 244$ to $1 \times 48 \times 18$ as described in Section 4.1.
- The final activation function was changed from a softmax to sigmoid function.
- The number of filters in the convolutional layers and units in the fully connected layers were lowered. By visualising and experimentally lowering the number of filters in layers we observed that the computational complexity of the network could be reduced without lowering its classification performance too much.
- No splitting of convolutional layers (grouping). In the original AlexNet, some convolutional layers were split across two GPUs due to memory constraints. Some of those layers only communicated with the preceding layer on that GPU. Our hardware had enough available GPU memory so we did not need to use grouping.

The original network had a softmax as final activation function. The softmax function ensures that the output is a probability distribution over all the classes: the sum over all the outputs is 1. In our localisation task, similar to [STE13], we want to predict a full binary mask. The activation function in the last layer of the network cannot be a threshold function because the backpropagation algorithm requires differentiable functions [Hay99, Section 1.3]. A common activation function is the sigmoid function, defined by $f(x) = \frac{1}{1+e^{-x}}$. This is a continuous, differentiable function with $\lim_{x \to \infty} f(x) = 1$ and $\lim_{x \to -\infty} f(x) = 0$. After learning, a threshold $t \in (0, 1)$ can be applied to the output of the sigmoid function to convert the network's output to binary.

We lowered the size of the filters in the first layer from 11×11 to 3×3. With our lower resolution input we lower the size of the convolutional filters with it. To preserve more detail for our localisation task, the stride was changed from 4 to 1. Zeiler and Fergus [ZF14] found that lowering the filter size and the stride of the first layer helped retaining more information in the 1st and 2nd layer. Simonyan and Zisserman [SZ15] showed even better performance by using 3×3 filters with stride 1 throughout the whole network. We observed that the number of filters could be lowered without lowering its classification performance too much.

The pooling layers remove spatial details by reducing the width and height of a convolutional layer. Dosovitskiy and Brox [DB15] have reconstructed images based on feature activations of AlexNet and found that object positions are preserved in the convolutional layers. Our experiments also show that this network structure is able to localise objects with high accuracy and recall.

This network structure needs to be trained. The learning parameters are based on the parameters included in BVLC's AlexNet model [SDK15], see Table 4.3. Most parameters are constant for every TADA iteration, only the learning rate at start and the number of samples in the training set changes.

	BVLC AlexNet [SDK15]	Modified
conv1	$96 \times 11 \times 11$	$\underline{16 \times 3 \times 3}$
	st. 4, pad 0, grp. 1, ReLU, LRN	st. $\underline{1}$, pad 0, grp. 1, ReLU, LRN
	3×3 MAX, st. 2	3×3 MAX, st. 2
conv2	$256 \times 5 \times 5$	$\underline{64 \times 3 \times 3}$
	st. 1, pad 2, grp. 2, ReLU, LRN	st. 1, pad $\underline{1}$, grp. $\underline{1}$, ReLU, LRN
	3×3 MAX, st. 2	3×3 MAX, st. 2
conv3	$384 \times 3 \times 3$	$\underline{96} \times 3 \times 3$
	st.1, pad1, grp. 1, ReLU	st. 1, pad 1,grp. 1, ReLU
conv4	$384 \times 3 \times 3$	$\underline{96} \times 3 \times 3$
	st.1, pad 1, grp. 2, ReLU	st. 1, pad 1, grp. $\underline{1}$, ReLU
conv5	$256 \times 3 \times 3$	$\underline{64} \times 3 \times 3$
	st. 1, pad 1, grp. 2, ReLU	st. 1, pad 1, grp. $\underline{1}$, ReLU
	3×3, MAX, st. 2	3×3 MAX, st. 2
fc6	4096, ReLU, Dropout	$\underline{1024}$, ReLU, Dropout
fc7	4096, ReLU, Dropout	$\underline{1024}$, ReLU, Dropout
fc8	1000, Softmax	$\underline{\text{864, Sigmoid}}$

Table 4.2: **Comparison of BVLC AlexNet with the modified network used in this thesis**. Underlined text indicates where the modified network differs. Both networks start with 5 convolutional layers and end with 3 fully-connected layers. For the convolutional layers, the first row specifies #filters $\times$ width $\times$ height. The second row specifies the size of the convolutional stride, spatial padding, grouping, the activation function and whether Local Response Normalisation (LRN) [KSH12] was used. Grouping limits the connectivity of a filter by splitting the input and output into groups. If there is a third row it specifies the properties of the pooling layer.

	BVLC AlexNet [SDK15]	Modified
Weight updating moment	Mini-batch, 256 samples	Mini-batch, 128 samples
Samples in training set	$\approx$ 1.2million	$38000 + 12000 \times (\textit{TADA-iterations} - 1)$
Mini-batch iterations	450000	20000
Learning rate at start	0.01	0.0001 for the initial training, subsequent training starts at 0.00001
Learning rate lowering	Reduced by factor 10 every 100000th mini-batch	Reduced by factor 10 every 10000th mini-batch
Loss function	Cross entropy	
Optimisation method	Stochastic Gradient Descent	
Momentum	0.9	
Weight decay	0.0005	

Table 4.3: Learning parameters. We used a mini-batch size of 128 since that was used in the original version of AlexNet [KSH12]. Initial experiments indicated that we could drop the number of mini-batch iterations to 20000 without losing much of the network's performance. The learning rate was lowered to improve numerical stability, as suggested by the authors of Caffe [SD15].

4.2.2 Adversarial Background Adjuster

The ConvNet that was trained on the initial training set was good at recognising buttons in front of a white background: it was trained to do so. To improve the training set, the ABA adjusts the background such that the network's prediction is as poor as possible. The optimiser may change any pixel of the input to any value in the range $[0, 255]$. The only constraint is that that pixel is not part of the button to localise. The trained ConvNet is used to determine the effectiveness of the adjustments in the background. The adjusted samples are added to the training set and the network is trained again, starting with the weights from the previous iteration. With the additional samples, the network develops an increased focus on the features of the buttons themselves. The focus on the background is diminished as it is constantly being modified to maximally fool the network. The only constant part is the button itself.

Optimisation procedure

The ABA requires a procedure to optimise the backgrounds of the samples. We have chosen to use simulated annealing for this. The advantage of simulated annealing is its simplicity, making it easy to implement. A downside is its computational inefficiency. Due to time constraints we did not focus on implementing a more efficient algorithm. See Section 3.2 for the details of simulated annealing procedure itself.

Every iteration within simulated annealing adjusts the state. In this case the state is a single sample from the dataset. In the adjustment, we set one non-button pixel to a new random value. This gives the simulated annealing algorithm maximal freedom to adjust the input image. After the adjustment the loss is calculated by doing a forward pass through the ConvNet and comparing the result with the ground truth.

More formally, we denote by $f : \{0 \dots 255\}^{h \times w} \to [0,1]^{h \times w}$ a button localiser mapping the image $x \in \{0 \dots 255\}^{h \times w}$ to a mask that indicates the degree of confidence that the pixels belong to the button. We have a loss function $l : \{0,1\}^{h \times w} \times [0,1]^{h \times w} \to \mathbb{R}$ that maps the ground truth binary mask $t \in \{0,1\}^{h \times w}$ and the output from the localiser to a score representing how poor the localiser has predicted the location of the button. For a given sample (x, t), we then search for an adversarial background $a \in \{0 \dots 255\}^{h \times w}$ that maximizes

$$l(t, f(x \circ t + a \circ (1 - t)))$$

where $\circ$ denotes the entrywise product of the matrices.

The last layer of the neural network has a sigmoid activation function. In that context, the cross entropy loss is a logical choice for training the network [Bis95, Section 6.7]. Our implementation of simulated annealing uses the neural network as a black box: it does not look at the internals of the neural network. Due to this we use a different loss function: the L_1 loss. The L_1 loss is computed as

$$l(t, z) = \sum_{h=1}^{h} \sum_{w=1}^{w} |t_{hw} - z_{hw}|.$$

This function targets the maximal absolute difference between the network output and the ground truth binary mask. The cross-entropy loss would favour larger differences between the prediction and ground truth of individual pixels. In our case, we would expect that using the cross-entropy loss will stimulate to expand/shrink the mask the network predicts. We expect that this area is the easiest place to obtain large differences of pixels. However, we have no bias towards this area: we optimise at a global level.

Implementation

The parameters for our annealing schedule were empirically determined. We used a minimum temperature of 10^{-7} and a maximum temperature of 0.8 with 200000 steps. We implemented the algorithm in C++ to efficiently communicate with Caffe. We optimised 6000 samples in parallel instead of a single sample per step. This reduced the computation time by more than a factor of 100. First, the backgrounds of all the samples were adjusted. The new samples were then passed through the neural network in one big computation. After this computation the simulated annealing states all progressed to prepare for the next big forward pass through the network. The forward pass through the network takes most of the time. By forwarding many images at once through the network we could reduce the overhead and reduce our computation time.

4.2.3 Augmentation of the dataset

Every TADA iteration, the next 12000 samples from the initial training set were adjusted and added to the training set. So in the first iteration, the first 12000 samples were adjusted and added back to the training set, increasing the size of the training set to 50000 samples. After training the neural network on this, we tested that network again on a validation set with 2048 additional samples. The resulting network's Dice coefficient (see Section 5.1) on the additional samples was ≥ 0.92 in every TADA iteration. In the last TADA iteration, we found that the neural network scored ≥ 0.99 on all the validation sets we generated. This indicates that the additional samples add little value to the training set as the network already works well on those samples. We did not experiment with any other number of samples per TADA iteration.

No samples were deleted from the training set. Deleting samples could cause the network to unlearn a type of background that it has previously learnt. This could cause an oscillation in the learning procedure where the network alternates between learning and unlearning a certain type of background.

5 | Experiments

5.1 Method

After every retraining of the ConvNet, we measure the performance of the network. A common measure for the performance of segmentation algorithms is the Dice coefficient [Dic45]. It measures the overlap between two binary annotations and is computed as

$$D(T,P) = \frac{2|T \cap P|}{|T| + |P|},$$

with T the ground truth and P the binary output of the network. The sets T and P only contain the locations that are marked as part of the button. With $|\cdot|$ we denote the cardinality of the set and with $\cap$ the intersection of the sets. The value of the Dice coefficient ranges between 0 (no overlap) and 1 (full agreement).

The Dice coefficient requires a binary output from the network but every output pixel of the neural network has a value in the range $(0, 1)$. We convert the output to binary by defining a threshold. To find the optimal threshold, we developed a *validation set*. This validation set is only used to determine an optimal threshold. This threshold will be used when assessing the network's performance on the *test set*. The test set is used to determine the network performance when it would be used in practice. The validation set consists out of 2048 samples from each TADA iteration and 2048 samples similar to the initial training set. The samples from the validation set and test set were not used for training.

Another way to express the performance of the neural network's segmentation is via a precision-recall curve, originating from the information retrieval community [Van79] but also appropriate for computer vision. A high precision indicates that when the neural network indicates that a pixel is part of a button, it probably is correct. A high recall value indicates that most of the pixels that are part of the button are also indicated by the neural network as such. We want both scores to be high. High recall with low precision means that the network classifies anything as part of the button. A result with high precision and low recall is also easy to obtain by only classifying the pixel in the centre as part of the button (cf. Section 4.1). Precision is defined as

$$P(T,P) = \frac{|T \cap P|}{|P|}$$

and recall as

$$R(T,P) = \frac{|T \cap P|}{|T|}.$$

Both functions have a range of $[0, 1]$. The precision-recall curve is generated by calculating the precision and recall for many different threshold values, covering the whole range of possible thresholds. The Dice coefficient represents the harmonic mean of precision and recall, which is also known as the F_1 score [O'C13, Appendix 9.1]:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

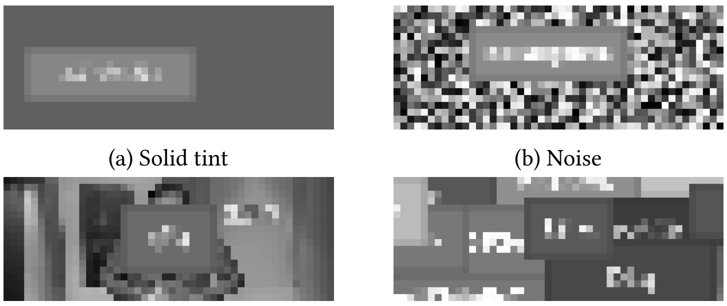
To be sure that the trained network did not overfit on the buttons, additional buttons were generated for the testing and validation. To test the final performance of the network on different backgrounds, various test sets were generated, each with 8192 samples. The types of test sets that were generated are:

- Solid tint: all the pixels in the background have the same value. As we use 8-bit greyscale images, this means that there are 256 possible backgrounds. This background has minimal complexity since all the pixels have the same value. With 8192 samples, every possible background value is generated 32 times.
- Noise: all the pixel values are sampled from a uniform distribution. The generated backgrounds have maximal complexity as the value of every pixel is randomly chosen.
- NOS. Every 4th frame of the NOS news broadcast of 2015-04-15 at 20:00 was converted to greyscale and resized to 48×18 pixels. Compared to the random and solid tint test sets, the NOS test set contains more natural shapes with a lot of structure. This gives it a mixture of various complexities across the whole background.
- Buttons: the background is completely composed out of buttons. This makes it a very hard test set. As explained in Section 4.1, the network should only recognise the button in the centre of the image. For details on the generation of this background, see Algorithm 5.1.

Samples from the 4 test sets are shown in Figure 5.1.

Algorithm 5.1 Generation of the buttons background.

```
1: $C \leftarrow$ empty canvas
2: **while** some pixels are not yet covered by a button in $C$ **do**
3:     Randomly place a button on $C$. At least one pixel of the button has to cover $C$,
       possibly overriding pixels of other buttons.
4: **end while**
```



(a) Solid tint (b) Noise

(c) NOS (d) Buttons

Figure 5.1: Random samples from each test set.

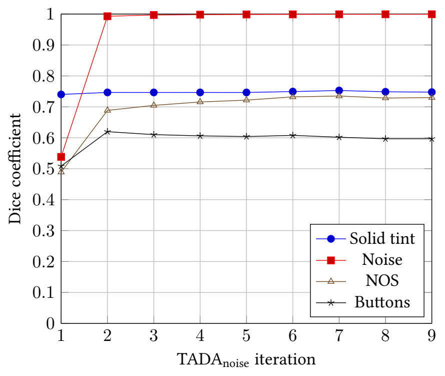
5.2 Baseline

Before testing our method we first set a baseline score. We first test the performance of our ConvNet design by training a ConvNet on 75% of the samples from each test set. We tested this neural network on the remaining 25% of each test set. The resulting Dice coefficient for each test set was ≥ 0.99, indicating that the network structure itself allows the network to learn what is required for the test sets.

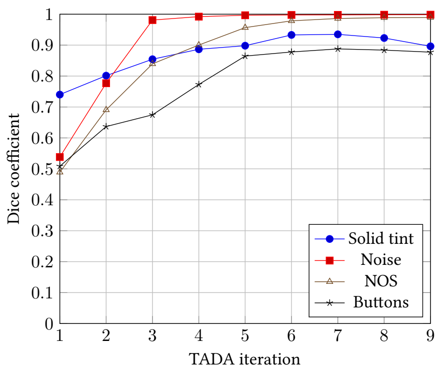
We also want to compare the effectiveness of our ABA. It could be that the sole addition of noise samples has the same effect as our ABA. To test this, we perform the same TADA iterations as described in the previous section but we replace simulated annealing in the ABA with a noise generator. We call the results with a noise generator as ABA $\text{TADA}_{\text{noise}}$. Our implementation of simulated annealing randomly sets pixels to a new value, trying to find a new background that increases the loss. If it would fail to increase the loss significantly, the background is just random noise like in the $\text{TADA}_{\text{noise}}$ test. Figure 5.2a shows the results of the $\text{TADA}_{\text{noise}}$ iterations. The augmentation of a random noise background to a dataset with only white datasets helps, but not as much as simulated annealing has done in the TADA iterations as Figure 5.2b shows. This indicates that our ABA is doing more than just adding noise to the training set.

5.3 Results

Figure 5.2b shows the Dice coefficient for each test set on each retrained version of the network. In the 1st iteration, the network performs the best on the solid backgrounds but shows an equally poor performance on the other 3 test sets. The solid tint test set performs the best, the other 3 test sets' scores are lower and very close to each other. After 4 TADA iterations, the ConvNet performs almost perfectly on the noise test set. We expect that this fast increase in performance is due to the noise patterns generated by the ABA. The NOS test set required more TADA iterations to get to a similar level as the noise test set. Compared to the NOS and noise set, the solid tint set stays behind but still performs well. The buttons test set shows itself to be the hardest one. Those two backgrounds have more areas with a single solid tint in the background. Due to



(a) **Dice coefficient per TADA$_{\text{noise}}$ iteration.** The augmentation of the noise background in iteration 2 improved the results significantly. The noise test set shows a huge performance improvement, the buttons and NOS dataset also improve a lot. The solid tint test set is almost unaffected by the addition of noise in the dataset.



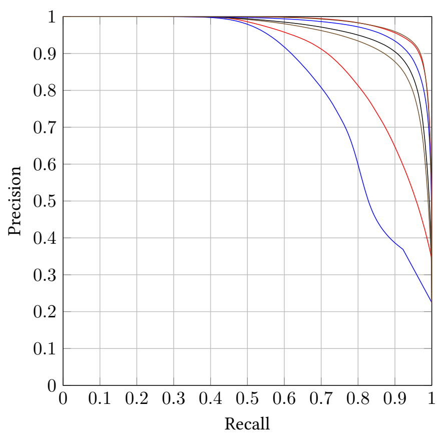
(b) **Dice coefficient per TADA iteration.** The Dice coefficient for the noise and NOS test set is monotonically increasing. The solid tint and button test sets scored best in iteration 7.

Figure 5.2: Dice coefficient per test set. In the left graph the dataset was augmented with random noise backgrounds whereas in the right graph our simulated annealing procedure was used. After the ConvNet was trained, the performance on the 4 test sets was measured. Iteration 1 shows the performance of the neural network that was only trained on white backgrounds.

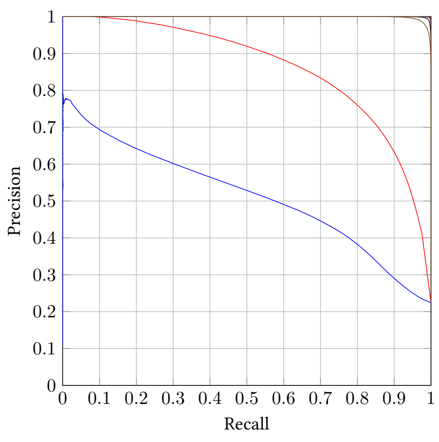
the random adjustment of pixels in the ABA, the probability of generating areas of a single solid tint is lower than generating noise patterns.

The precision-recall curves in Figure 5.3 show more detailed statistics. It shows that the curve of the first iteration has a similar shape for all test sets except the solid tint set. The network was not trained on non-white backgrounds and seems to get confused by anything else. This is what we expect, the network has learned to rely on the white background to recognise the buttons. The curves visualise the trade-off between precision and recall. For example, when we want to click on a button, we favour a high precision result from the ConvNet. For that use case it is important that we really click on the button, not 1 pixel next to it. The precision-recall curves show that with a recall of 0.7 a precision ≥ 0.95 is achieved in all the test sets.

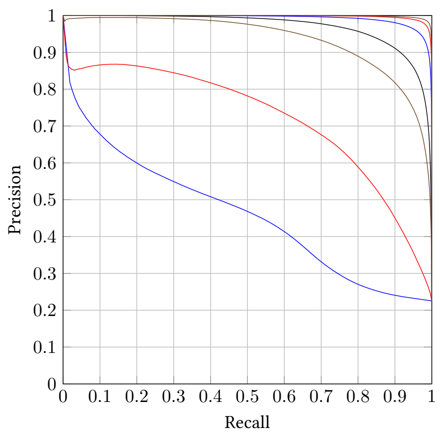
The violin plot in Figure 5.4 shows the distribution of the L_1 loss for the adjusted samples. It clearly shows that the last TADA iterations produce significantly less effective samples. Additionally, the 8th and 9th TADA iteration in Figure 5.2 show a decreasing score for the solid tint and the buttons background while the NOS and noise background scores increase slightly. The less effective samples are added to the training set but are not adding much useful information. Figure 5.5 shows two random samples from the 8th TADA iteration. They did start off as a completely white background but the adjusted result is far from the solid backgrounds test set. The noisy patterns are an artefact of the used optimisation procedure. The pixels of the background are randomly changed. With the initially high temperature of the simulated annealing procedure, changes that do not lower the loss are kept and introduce a noise pattern



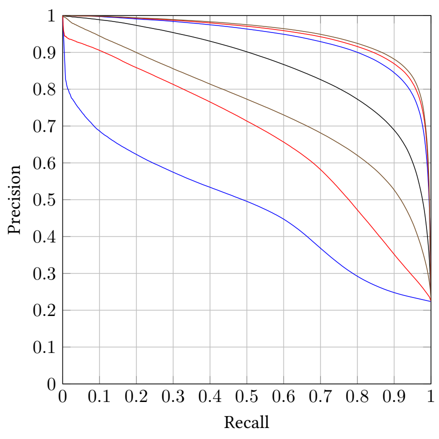
(a) Solid tint



(b) Noise



(c) NOS



(d) Buttons

Figure 5.3: **Precision-recall curves for each test set.** Every graph contains 7 curves, indicating TADA iteration $1 - 7$. The closer the curve reaches the top right corner the later the iteration. Maximum recall is achieved when the networks classifies every pixel as being part of the button. Maximum precision indicates that every pixel that was classified as being part of a button is indeed part of a button. On average, 22% of the surface is covered with a button (see Section 4.1.2). This explains the minimum precision of 22%.

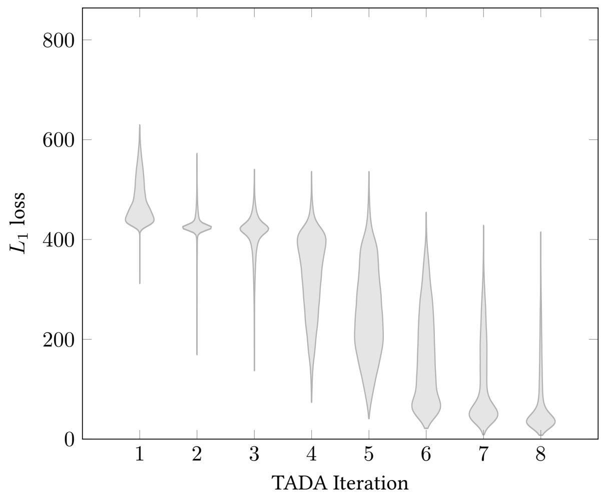


Figure 5.4: **Violin plot of the adjusted samples per TADA iteration.** This plot shows the distribution of the 12000 samples that were generated by the ABA in each TADA iteration. The width of violins indicates the probability density at different loss values. It shows that with each iteration, the average loss decreases. This indicates that it becomes harder to find a background that significantly fools the ConvNet.

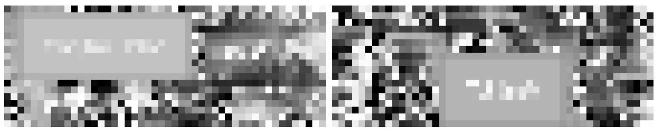


Figure 5.5: Random samples from the adjusted samples generated in the 8th TADA iteration.

in the background. In every TADA iteration, non-uniform backgrounds are added and the network's weights are tuned such that the neural network does not get fooled by the noisy patterns we generated. The backgrounds in the NOS and noise test set are more similar to the patterns generated by the ABA than the buttons and solid tint test set. Additionally, the number of samples with a white background in the training set becomes less significant due to the addition of the adjusted samples. This could cause the network to overfit slightly on the noisy type of background. We expect that those factors caused the decrease of the performance of the network after the 7th TADA iteration.

5.4 Conclusion

We found that using our adversarial background augmentation procedure to train a ConvNet improves the generalisation performance of that ConvNet. In this thesis, we trained a ConvNet on a training set with a button in front of a completely white background. This network was retrained on adversarial backgrounds. The philosophy behind our approach is that adversarial backgrounds exploit weaknesses of the network's way of localising buttons. Retraining with adversarial backgrounds gives the network the opportunity to correct those weaknesses.

We precisely defined a model that captures how the buttons we want to localise may look. Based on that model we sampled as many buttons as we needed for training. This allows the network to learn to recognise the shared properties of the buttons to localise. The background around the button is also important. A network that is trained on a background that is too simple will get confused when it sees other types of backgrounds. It becomes dependent on this background instead of the features of the button itself. Instead of creating a representative model for the backgrounds that the network could ever encounter after training, we found that we could generate adversarial backgrounds. This eliminates the need to manually design a model for the background around the button.

We initially trained a ConvNet on buttons in front of a completely white background. Via TADA iterations we improved this network by retraining it on adversarial backgrounds. We found that applying multiple TADA iterations helped to improve the network's performance significantly. The final ConvNet generalised much better to unseen backgrounds. Simply using completely white and noise backgrounds to train the ConvNet yielded a Dice coefficient of 0.74 on the NOS test set, our method boosts this score by 34% to 0.99. This is a close to perfect score.

Our method of adversarial background augmentation opens up ways to easily train new object localisers. It has allowed us to develop a button localiser without having to design a model of the background around it. Our method simply requires sufficient samples of the object to recognise and a ConvNet structure with a corresponding training procedure. Though we only tested our method on buttons, our method is not specifically designed for buttons. We expect that our method can be applied to other objects as well. The ConvNet learns from the data and our adversarial backgrounds are adapted to the trained ConvNet.

5.5 Future work

Our implementation of simulated annealing sets a random pixel of the background to a new random value. This is an inefficient way to generate adversarial backgrounds. In every TADA iteration, it takes approximately 18 hours to generate the optimised samples. Although this time can be shortened by tuning the annealing schedule and simplifying the neural network structure even more, we could try to infer the adversarial samples directly from the network itself via gradient ascent [EBCV09, SVZ14]. Since the adversarial samples are inferred from the network itself, we expect those adversarial samples are better and faster to generate. A more efficient ABA could also make using higher input resolutions more practical. The higher the resolution, the more pixels that can be altered and thus the more effort required.

Every sample in our training set always contained a button. The resulting neural network cannot cope with samples that do not contain a button at all. In Section 4.1, we explained that the ABA could start drawing actual buttons in the background while we tell the network that there is no button at that location. The probability that the simulated annealing algorithm will generate a button that conforms to our model is very low. Even if it would occur, we expect that the majority of the samples will still steer the network in the right direction. On the other hand, it could cause the network to be overly conservative in its predictions. We also did not consider the case with more than one button in the input. We had the requirement that the button was part of the centre of the input (see Section 4.1). To detect buttons in a large image, we would need to apply this network at many positions of that image. Every button of the large image has to be part of the centre of the neural network in order for it to be detected. So when sliding the input of the neural network over the large image, the stride is dependent on the minimum dimensions of the buttons. By dropping this requirement, the whole input of the neural network can be used to detect buttons, saving computation time. It also opens ways to localise multiple buttons at once.

Our method uses a binary output mask to give a pixel-wise annotation of the location of the button. This level of detail might not be needed. In our case, a 4D vector with location of the button could also be sufficient. The 4D vector can represent the coordinates of the centre, the width and height of the button (e.g. [SZ15]). Alternatively, the 4D coordinate could represent the coordinates of the bounding box (e.g. [WZL+14]). The MultiBox method [ESTA14, SREA14, HZRS15] uses a 4D vector system and supports locating a dynamic number of objects via a ConvNet.

We only tested with a square button with limited variety. We would like to know how our method would perform on more complex objects to recognise. Our object to recognise could be modelled as a 3D model. The 3D model gives the required binary mask of its location. Our method can generate the backgrounds around it. Instead of localisation, this might also help training classifiers that assign a category to an image. In future work we could test our method on existing datasets to compare the performance with the state of the art.

In every TADA iteration, we add more samples to the dataset. The more TADA iterations we do, the lower the impact of the new samples. In every iteration it also becomes harder to generate samples that significantly fool the network. Since the impact of the new samples is also reduced, the learning slows down. Environments where more TADA iterations are required might benefit from a different augmentation scheme. A different augmentation scheme that drops older samples or favours newer samples during training might speed up learning.

Bibliography

[BBB$^+$10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010. URL: http://www-etud.iro.umontreal.ca/~wardefar/publications/theano_scipy2010.pdf. 7

[BHC15] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *CoRR*, abs/1505.07293, 2015. URL: http://arxiv.org/abs/1505.07293. 7

[Bis95] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995. 20

[CKF11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. URL: http://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf. 7

[CM00] D. Connolly and L. Masinter. RFC2854: The 'text/html' Media Type, 2000. URL: https://tools.ietf.org/html/rfc2854. 15

[CMS12] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012. URL: http://arxiv.org/abs/1202.2745. 7, 8

[Cor15] Nvidia Corporation. CUDA C Programming Guide, 2015. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. 7

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008. URL: http://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf. 1

[DB15] Alexey Dosovitskiy and Thomas Brox. Inverting convolutional networks with convolutional networks. *CoRR*, 2015. URL: http://arxiv.org/abs/1506.02753. 17

[DF10] Morgan Dixon and James Fogarty. Prefab: Implementing Advanced Behaviors Using Pixel-Based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1525–1534, 2010. URL: https://homes.cs.washington.edu/~jfogarty/publications/chi2010-prefab.pdf. 6

[Dic45] Lee R. Dice. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):297–302, 1945. URL: http://www.jstor.org/stable/1932409. 22

[DYDA12] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):30–42, 2012. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=144412. 1

[EBCV09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep*, 4323, 2009. URL: http://igva2012.wikispaces.asu.edu/file/view/Erhan+2009+Visualizing+higher+layer+features+of+a+deep+network.pdf. 28

[ESTA14] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable Object Detection using Deep Neural Networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2155–2162. IEEE, 2014. URL: http://arxiv.org/abs/1312.2249. 6, 8, 29

[Ete11] Elika J. Etemad. Cascading Style Sheets (CSS) Snapshot 2010, 2011. URL: http://www.w3.org/TR/2011/NOTE-css-2010-20110512/. 15

[EvW$^+$09] Mark Everingham, Luc van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2), September 2009. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=102944. 6

[Fau06] Kjell Magne Fauske. Example: Neural network, 2006. URL: http://www.texample.net/tikz/examples/neural-network/. 11

[FCNL12] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers. In *Proc. International Conference on Machine learning (ICML'12)*, 2012. URL: http://arxiv.org/abs/1202.2160. 7

[FGMR10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part Based Models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010. URL: http://cs.brown.edu/~pff/papers/lsvm-pami.pdf. 6

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference* *on*, pages 2–9. IEEE, 2014. URL: http://arxiv.org/abs/1311.2524. 1, 8

[GR15] Shixiang Gu and Luca Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *CoRR*, 2015. URL: http://arxiv.org/abs/1412.5068. 8, 9

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *CoRR*, 2015. URL: http://arxiv.org/abs/1412.6572. 8, 9

[GWFL$^+$13] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. 2013. URL: http://arxiv.org/abs/1308.4214. 7

[GWS] Kevin Gibbs, Terry Winograd, and Neil Scott. Lens: A System for Visual Interpretation of Graphical User Interfaces. URL: https://cs.stanford.edu/~kgibbs/lens.pdf. 6

[HAGM14] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous Detection and Segmentation. In *European Conference on Computer Vision (ECCV)*, 2014. URL: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/papers/BharathECCV2014.pdf. 7

[Hay99] Simon S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999. 10, 17

[Hin14] Geoffrey Hinton. Dropout : A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. URL: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf. 7

[How13] Andrew G. Howard. Some Improvements on Deep Convolutional Neural Network Based Image Classification. *CoRR*, 2013. URL: http://arxiv.org/abs/1312.5402. 7

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, 2015. URL: http://arxiv.org/abs/1502.01852. 1, 16, 29

[JSD$^+$14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, 2014. URL: http://arxiv.org/abs/1408.5093. 7, 16

[Kar15] Andrej Karpathy. Convolutional Neural Networks, 2015. URL: https://github.com/cs231n/cs231n.github.io/blob/2e3b53f36d524b3c95c1e731c97a7e0c391aa78d/convolutional-networks.md. 12

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. URL: http://www.jstor.org/stable/1690046. 12

[KHK08] Takuya Kobayashi, Akinori Hidaka, and Takio Kurita. Selection of histograms of oriented gradients features for pedestrian detection. In *Neural Information Processing*, volume 4985 of *Lecture Notes in Computer Science*, pages 598–607. Springer Berlin Heidelberg, 2008. URL: http://link.springer.com/chapter/10.1007/978-3-540-69162-4_62. 6

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-w. 1, 5, 6, 7, 8, 16, 17, 18, 19

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf. 7

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. URL: http://go.nature.com/7cjbaa. 7, 10

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *CoRR*, 2013. URL: http://arxiv.org/abs/1312.4400. 7, 8

[Low04] David G Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International journal of computer vision*, 60(2):91–110, 2004. URL: http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf. 6

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. URL: http://link.springer.com/article/10.1007/BF02478259. 1

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015. URL: http://www.evolvingai.org/fooling. 7, 8, 9

[OBLS14] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 2014. URL: http://www.di.ens.fr/willow/research/cnn/. 6, 8

[OBLS15] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Is object localization for free? – Weakly supervised object recognition with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015. URL: http://www.di.ens.fr/willow/research/weakcnn/. 6, 8

[O'C13] Brendan O'Connor. Learning Frames from Text with an Unsupervised Latent Variable Model. *CoRR*, 2013. URL: http://arxiv.org/abs/1307.7382. 23

[OLZ$^+$14] Wanli Ouyang, Ping Luo, Xingyu Zeng, Shi Qiu, Yonglong Tian, Hongsheng Li, Shuo Yang, Zhe Wang, Yuanjun Xiong, Chen Qian, Zhenyao Zhu, Ruohui Wang, Chen-Change Loy, Xiaogang Wang, and Xiaoou

Tang. DeepID-Net: multi-stage and deformable deep convolutional neural networks for object detection. 2014. URL: http://arxiv.org/abs/1409.3505. 8

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg C., and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. URL: http://arxiv.org/abs/1409.0575. 6, 7

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. 1

[Sch15] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117, 2015. URL: http://arxiv.org/abs/1404.7828. 10

[SD15] Evan Shelhamer and Jeff Donahue. Caffe — Solver / Model optimization, 2015. URL: https://github.com/BVLC/caffe/blob/648aed72acf1c506009ddb33d8cace40b75e176e/docs/tutorial/solver.md. 19

[SDK15] Evan Shelhamer, Jeff Donahue, and Sergey Karayev. BVLC's AlexNet model, 2015. URL: https://github.com/BVLC/caffe/tree/e6c80dac40d4bb13390b56bc18294e6e91b00436/models/bvlc_alexnet. 16, 17, 18, 19

[SEZ+14] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat : Integrated Recognition, Localization and Detection using Convolutional Networks. In *International Conference on Learning Representations (ICLR 2014)*. CBLS, 2014. URL: http://openreview.net/document/cb1bf585-d8ae-4de7-aa0c-f47cdc763d8d. 1, 6, 8

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR 2015*, 2015. URL: http://arxiv.org/abs/1409.4842. 8

[SQLG15] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views. *CoRR*, 2015. URL: http://arxiv.org/abs/1505.05641. 8, 9

[SREA14] Christian Szegedy, Scott Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, High-Quality Object Detection. *CoRR*, 2014. URL: http://arxiv.org/abs/1412.1441. 8, 29

[SS14] Baochen Sun and Kate Saenko. From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. URL: http://www.bmva.org/bmvc/2014/papers/paper062/index.html. 8, 9

[STE13] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013. URL: http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf. 1, 6, 8, 16, 17

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *Workshop at International Conference on Learning Representations*, 2014. URL: http://arxiv.org/abs/1312.6034. 6, 8, 28

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015. URL: http://arxiv.org/abs/1409.1556. 1, 7, 8, 16, 17, 29

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL: http://arxiv.org/abs/1312.6199. 8

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 2014. URL: http://www.cs.tau.ac.il/~wolf/papers/deepface_11_01_2013.pdf. 7

[Van79] C. J. Van Rijsbergen. Information Retrieval. 1979. URL: http://www.dcs.gla.ac.uk/Keith/Preface.html. 22

[WZL+14] Xiaolong Wang, Liliang Zhang, Liang Lin, Zhujin Liang, and Wangmeng Zuo. Deep Joint Task Learning for Generic Object Extraction. In *Advances in Neural Information Processing Systems 27*, pages 523–531. Curran Associates, Inc., 2014. URL: http://papers.nips.cc/paper/5547-deep-joint-task-learning-for-generic-object-extraction.pdf. 6, 8, 29

[ZDGD14] Ning Zhang, Jeff Donahue, Ross Girshick, and Trevor Darrell. Part-based R-CNNs for fine-grained category detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. URL: http://arxiv.org/pdf/1407.3867.pdf. 8

[ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Computer Vision–ECCV 2014*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer International Publishing, 2014. URL: http://arxiv.org/abs/1311.2901. 5, 8, 16, 17

[ZKL+14] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Olivia, and Antonio Torralba. Object Detectors Emerge in Deep Scene CNNs. *CoRR*, 2014. URL: http://arxiv.org/abs/1412.6856. 8

Acronyms

ABA	Adversarial Background Adjuster. 5, 14, 15, 19, 24, 25, 27–29
ANN	Artificial Neural Network. 1, 2
BVLC	Berkeley Vision and Learning Center. 16–19
ConvNet	Convolutional Neural Network. 1, 2, 4–8, 12, 14, 15, 19, 22, 24, 25, 27–29
CPU	Central Processing Unit. 7
CSS	Cascading Style Sheet. 15, 16
DNN	Deep Neural Network. 2, 8, 10, 11
DPM	Deformable Parts Model. 6
GPGPU	General Purpose computation on GPUs. 7
GPU	Graphics Processing Unit. 1, 7, 17, 36
HOG	Histograms of Oriented Gradients. 6
HTML	HyperText Markup Language. 15
LRN	Local Response Normalisation. 18
NOS	Nederlandse Omroep Stichting, English: Dutch Broadcast Foundation. 15, 23, 27, 28
NTA	Nedap Testing Automation. 3
R-CNN	Regions with CNN features. 8
SGD	Stochastic Gradient Descent. 10
SIFT	Scale Invariant Feature Transform. 6
SIMD	Single Instruction Multiple Data. 7
TADA	Train ADjust Augment. 5, 14, 17, 19–22, 24–29