FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Detection and removal of watermarks from image data
Student:	Bc. Tomáš Halama
Supervisor:	Ing. Miroslav Čepek, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Removal of visible watermarks from images is a well-known task in the image restoration domain. While some watermarks are easy to remove, others are more difficult and might require using a more sophisticated method.

In recent years, there has been a breakthrough in applying various deep learning models to computer vision tasks.

The goal of this thesis is to explore deep learning approaches to watermark removal. Adapt and apply a selected deep learning method for detecting and removing watermarks to a supplied real-world dataset.

Steps:
1) Identify and analyze types of watermarks present in the dataset.
2) Survey available deep learning methods for detecting and removing watermarks.
3) Survey transformer networks suitable for watermark removal task.
4) Design and implement a deep learning model suitable for the watermark removal.
5) Prepare a training pipeline for image watermark removal using the supplied dataset.
6) Experimentally evaluate the performance of available models and the implemented method using PSNR, SSIM or similar metrics. Discuss the limitations of the implemented method.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 19 August 2022 in Prague.

Master's thesis

DETECTION AND REMOVAL OF WATERMARKS FROM IMAGE DATA

Bc. Tomáš Halama

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Miroslav Čepek, Ph.D.
April 20, 2023

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Halama Tomáš. *Detection and removal of watermarks from image data.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

Author: *Hey ChatGPT, let's write a funny acknowledgements section for my thesis. First, let's thank my thesis supervisor, Ing. Miroslav Čepek, Ph.D.*
ChatGPT: *Of course! Big thanks to Ing. Miroslav Čepek, Ph.D., the mastermind behind my research journey, who's kept me sane and on track despite the occasional science-induced meltdown.*

Author: *Nice, now let's thank my girlfriend, Ivetka.*
ChatGPT: *To my fantastic girlfriend, Ivetka, thank you for your unwavering support, and for mastering the art of decoding my research rambles. You deserve a medal!*

Author: *Good! Now, let's acknowledge my family.*
ChatGPT: *A huge shout-out to my family, the dream team that's always had my back, even when my thesis turned our home into a fortress of books and papers.*

Author: *Great, now let's thank my friends and the Žabla tým.*
ChatGPT: *To my friends and the unstoppable Žabla tým, thank you for providing comic relief and reminding me that laughter truly is the best medicine during thesis-writing marathons.*

Author: *Lastly, let's thank the other researchers and their brilliant work.*
ChatGPT: *And finally, a toast to the inspiring researchers whose work lit the way for my own research. Your brilliance and creativity have me convinced you're part-time superheroes.*

Author: *Thanks, ChatGPT! This funny version is perfect for my acknowledgements section.*
ChatGPT: *You're welcome, Author! I'm glad I could help, don't forget to cite me [1]. Best of luck with your thesis!*

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on April 20, 2023 . . . .

Abstract

Digital image watermarking is a widely used technique for protecting intellectual property or authenticating digital media, but it can negatively impact image quality and usability. This motivates the need for removing watermarks from images, and deep learning presents a potential solution. This thesis develops a deep learning method for watermark removal, including a survey of existing techniques and the proposal of a novel architecture. The method's performance is evaluated in terms of watermark detection accuracy and image reconstruction quality.

Keywords deep learning, image editing, watermark removal, watermark, transformer, GAN

Abstrakt

Digitální obrazové vodoznaky jsou široce používanou technikou pro ochranu duševního vlastnictví nebo ověřování digitálních médií, ale mohou mít negativní vliv na kvalitu a použitelnost obrázků. To motivuje potřebu odstraňovat vodoznaky z obrázků a hluboké učení představuje potenciální řešení. V této práci je vyvinutá metoda pro odstraňování vodoznaků pomocí hlubokého učení, včetně rešerše stávajících technik a návrhu architektury. Úspěšnost metody je vyhodnocena z hlediska přesnosti detekce vodoznaků a kvality rekonstrukce původních obrázků.

Klíčová slova hluboké učení, úprava obrazu, odstranění vodoznaku, vodoznak, transformer, GAN

List of Acronyms

BCE	Binary Cross-Entropy
BN	Batch Normalization
cGAN	Conditional Generative Adversarial Network
CLWD	Colored Large-scale Watermark Dataset
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
IoU	Intersection over Union
LVW	Large-scale Visible Watermark Dataset
LSTM	Long Short Term Memory
MAT	Mask-Aware Transformer
MS COCO	Microsoft Common Objects in Context
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLP	Natural Language Processing
PSNR	Peak Signal-to-Noise Ratio
PVT	Pyramid Vision Transformer
RGB	Red Green Blue (color model)
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
SLBR	Self-calibrated Localization and Background Refinement
SN	Spectral Normalization
SSIM	Structural Similarity Index Measure
TAWR	Transformer Architecture for Watermark Removal
ViT	Vision Transformer
wBCE	Weighted Binary Cross-Entropy
WDnet	Watermark-Decomposition Network

Introduction

It is important to emphasize that removing watermarks from digital media may lead to copyright infringement. The presented research results are intended for analysis and improvement of watermarking systems and are to be used within legal boundaries only.

0.1 Motivation

Watermarking is a widely used technique for protecting intellectual property or authenticating digital media. However, the presence of watermarks can have negative impacts on image quality and usability for image processing tasks, including various tasks of data mining or feature extraction.

This motivates the need for removing watermarks from images in order to enhance their quality and make them more suited for further use. By removing watermarks from images, it is also possible to test the limits of the robustness of the watermarking methods and therefore provide insights into their effectiveness and areas for improvement.

Deep learning's success in image-based tasks such as classification, object detection, segmentation, and content generation suggests its suitability for solving the task of watermark removal. Exploring this approach for our use case is intuitive, given the potential to learn from large datasets.

0.2 Objectives

Our goal is to develop a method using deep neural networks to effectively detect and remove watermarks without compromising image quality. The explicit objectives of this thesis are to:

- survey existing watermark removal techniques and vision transformer models,
- design a deep learning architecture for removing watermarks,
- implement and train the proposed deep learning method in a suitable framework,
- and evaluate its watermark detection and image reconstruction performance.

0.3 Structure of the Thesis

This thesis is structured into six main chapters:

- Chapter 1 provides an overview of the theory and mathematical foundations behind watermarking. This chapter includes a review of relevant literature on the principles and techniques of watermarking.
- Chapter 2 contains a review of existing methods and models for detecting and removing watermarks, as well as an overview of transformer networks and their applications in vision tasks.
- Chapter 3 describes the provided dataset and its processing. We discuss the characteristics of the dataset, outline the preprocessing steps performed, and propose a watermarking process required for synthesizing the training datasets.
- Chapter 4 proposes a deep learning architecture for watermark removal. We describe the design of the proposed model in detail, including its building blocks, their purpose, the utilized loss functions and various other components required for training the model.
- Chapter 5 describes the methodology and setup of the conducted experiments. This chapter summarizes metrics used, the model variants, and their comparison scenarios, i.e. the datasets used for evaluation and their characteristics.
- Chapter 6 presents the results of our experiments using the proposed model. The chapter includes experimental results and an analysis of the model's performance in the detection and reconstruction tasks. Additionally, we provide a discussion on the model's limitations and possible improvements.

This thesis is intended for a reader who has a respectable knowledge of the machine learning field. Background information about key concepts can be found in other publications [2].

Chapter 1

Watermarks

1.1 What is a Watermark?

The general purpose of using watermarks is to embed additional information into an existing carrier (such as a piece of art or digital media) without substantially damaging it. The motivations and methods of such information attachment allow us to distinguish between different areas such as steganography, watermarking, or other forms of general information hiding [3].

(a) Watermarked document **(b)** Colored organization logo **(c)** Copyright specification

Figure 1.1 Examples of watermark usage in images.

The main distinguishing characteristic between watermarking and other methods is whether the original data is the main object of interest or whether it is merely a message carrier. If our main goal is to send a hidden message while using the carrier data for masking the message's existence, then we are dealing with steganography. In contrast, in the watermarking scenario, the carrier object is our main point of interest and we can use the information embedded by a watermark to prove the origin or affiliation of the object without significantly distorting its appearance.

In summary, attaching messages to data can be done with the goal of [3]:

- providing additional information about the data (watermarking),
- hiding a message unrelated to the data (steganography),
- or other inclusion of a message unrelated to the data, without necessarily hiding it.

1.1.1 History and Examples

The historical roots and first documented watermarks come from papermaking, where the paper was made thinner and thus more translucent in certain areas. When the paper is exposed to light, the thinner areas become lighter in color and form a watermark [4]. The term *watermark* itself was first used in the 18th century and is not actually related to water, but rather to the similarity between the appearance of watermarks and the smudges left on water-damaged paper [5]. Ever since watermarks began to be used to indicate the authenticity of paper money, counterfeits began to appear, causing a competition between watermarking and counterfeiting methods [3].

Watermarks can be created in different ways depending on the form and medium of the object. Nowadays, digital watermarks are very popular mainly because of copyright protection of media content. The concept of digital watermarking can be extended to other kinds of data such as audio [6, 7], video [8] or even neural networks [9, 10].

(a) Original image **(b)** Visible watermark **(c)** Invisible watermark

Figure 1.2 Comparison of watermark perceptibility.

1.2 Watermark Taxonomy

Watermark types can be grouped according to several criteria. Loosely following the classification given in [11], a brief comparison of the different forms of watermarks will be demonstrated. For the purposes of this work, only digital image watermarks are considered, but the described general concepts apply across different forms of media.

Although not important for our purposes, other aspects for classifying watermarking methods also exist, such as the bit capacity of the watermark, requirements for retrieving the embedded information, or any relevant party's awareness of the watermark's presence [11, 12].

1.2.1 Human Perception

One of the possible classifications is determined by human perceptibility of the watermark. In the case of image media, we distinguish between *visible watermarks* and *invisible watermarks*. A comparison between the two can be seen in Figure 1.2, where Figure 1.2c demonstrates the use of an invisible watermark. For this particular example, a steganographic tool [13] for hiding files in images was used to encode a plaintext file containing the word `watermark` into the image data. The information is imperceptible to the human eye, but can be recovered if needed.

The use of visible watermarks directly degrades the original image by obscuring it. To be effective, a visible watermark ought to cover a large enough portion or critical area of the image to prevent it from being easily removed by cropping. In contrast, invisible watermarks are not perceptible to the user and usually do not significantly distort the image signal.

1.2.2 Robustness

An important aspect to consider is the ability of the watermark to withstand various damage and modifications to the carrier. In an image, a *robust watermark* should be detectable even after applying commonly used transformations such as scaling, compression, noise addition et cetera [12]. Approaches where the watermark information is easily corrupted by minor disturbances are called *fragile*. It should be noted that the properties of perceptibility and robustness are conflicting and mutually limiting [3, 12]. Creating a more robust watermark requires a more significant change, which in turn leads to higher perceptibility of the watermark embedding in the image.

1.3 Watermark Removal

As stated before, this work focuses solely on image data, specifically on removing visible watermarks. While there are methods designed to perform robust imperceptible watermarking [12], we will omit dealing with such methods in favor of focusing on robust visible watermarks. In real life data, visible image watermarks appear in various forms and shapes. The most common are diverse transparent elements such as logos, texts or geometric shapes overlaying the image. Sometimes an opaque graphic inserted into the image can also be considered a watermark, as it still follows the definition of a watermark described in Section 1.1. This type of watermarking can be used in cases where the watermark form and shape blends in with the rest of the content or is placed outside the main subject of the image and is not too visually disruptive.

Removing watermarks can be beneficial in several scenarios. As long as we own the copyright to the content or are permitted to edit it in such a way, we can use watermark removal to improve the visual quality of the content. Additionally, watermark removal methods enable media owners to analyze the effectiveness of their own watermarking system for copyright protection. There are various approaches of attacking a watermarked image, but historically this would require significant manual human effort. Nowadays, various automated methods exist (see Section 2.1) and their effectiveness is bound to how and what information they leverage to achieve a reconstruction of the original unwatermarked image.

Manually removing a single visible watermark using common graphic editing software is usually a feasible task for an experienced user. However, bulk automatic watermark removal on a larger dataset is a significantly more difficult problem. One of the biggest challenges when trying to automatically remove a watermark is finding it, as in the most general case, we do not know where the watermark is, what it looks like and whether it is even present. As we can see, it is essential to have at least an approximate understanding of the underlying watermarking process.

Once we are able to detect the watermark's position and shape we can attempt to reconstruct the image beneath. If the watermark is opaque, we are dealing with an image inpainting scenario, as there is no information about the original data. For a transparent watermark, we can estimate the watermark and its transparency level and attempt to recover the true original data. This usually leads to a coarse result, but serves as a good baseline for further refinement [14, 15, 16].

1.3.1 Image Inpainting

Watermark removal and image inpainting share some similarities, as both involve replacing a specific region of pixels in an image with a seamless and plausible replacement. A major breakthrough in solving such tasks was made possible through the introduction of Generative Adversarial Networks (GANs) [17, 18], which have greatly increased the perceptual quality of the results.

However, one potential downside of using such techniques for watermark removal is that they are typically user-guided, which is not compatible with our lack of knowledge about the watermark's position and shape [14]. Our problem is more similar to the blind inpainting problem [19, 20], where we need to reconstruct a corrupted image without knowing which region is corrupted. Usage of such a framework would implicitly require the model to estimate the location of the corrupted region on its own. However, these methods generally do not assume any transparency of the damage, which can lead to suboptimal results when dealing with transparent watermarks.

One potential solution to this problem is to develop a deep learning model that can automatically estimate the damage mask and its transparency within an image. This model could use techniques such as image segmentation to identify the watermark and determine its transparency. By incorporating this information into the inpainting process, we could create more effective and efficient algorithms for removing watermarks (see Section 2.1.2).

1.3.2 Problem Definition

Let us introduce the watermarking process using mathematical notation. Let $X, W, M \in \mathbb{R}^{w \times h}$ be the original unwatermarked image matrix, the watermark and an alpha mask for the watermark, respectively[1]. Additionally (as the elements of M describe the opacity percentage of the watermark), $0 \leq M_{i,j} \leq 1$ holds for all valid i, j.

Using the introduced matrix notation, watermarked image $\hat{X}$ is then created as

$$\hat{X} = X \odot (1 - M) + W \odot M$$

or alternatively, following similar notation as [21], for any given particular pixel position (i, j) we can write the relationship as

$$\hat{X}_{i,j} = (1 - M_{i,j}) X_{i,j} + M_{i,j} W_{i,j}, \tag{1.1}$$

which allows us to describe the inverse relationship as

$$X_{i,j} = \frac{\hat{X}_{i,j} - M_{i,j} W_{i,j}}{1 - M_{i,j}}.$$

This relationship is important for our use case, because it describes the watermark removal process. To avoid understating the difficulty of the task, notice that in a real-world scenario, we are given only the $\hat{X}$ matrix and everything else is an unknown variable.

[1] For simplicity, we neglect that images usually have 3 channels for RGB values. These formulas can be trivially extended to account for all channels at once.

1.3.2.1 Overlapping Watermarks

Due to possibly having multiple watermarks embedded in a single image, let us mathematically describe the intuitive result of overlaying watermarks, i.e. watermarking an already watermarked image. Equation 1.1 describes the image-watermark composition, let us reuse it for two watermarks W', W'' with their respective masks M', M''. Firstly we apply the formula to a watermarked image $\hat{X}$, then we expand $\hat{X}$ using the same formula and finally simplify the expression and rearrange the terms.

$$\hat{\hat{X}}_{i,j} = (1 - M''_{i,j})\hat{X}_{i,j} + M''_{i,j}W''_{i,j}$$

$$\hat{\hat{X}}_{i,j} = (1 - M''_{i,j})((1 - M'_{i,j})X_{i,j} + M'_{i,j}W'_{i,j}) + M''_{i,j}W''_{i,j}$$

$$\hat{\hat{X}}_{i,j} = (1 - M'_{i,j})(1 - M''_{i,j})X_{i,j} + M'_{i,j}(1 - M''_{i,j})W'_{i,j} + M''_{i,j}W''_{i,j}$$

As we can see, the information from the original image X gets multiplied by the complement of both watermarks' opacities. This goes in line with the intuition that there is less information from the original image left after overlaying watermarks. This process could easily be repeated to show similar relationship for an arbitrary number of watermarks.

Chapter 2

State-of-the-Art

This chapter provides a review of existing literature on watermark removal and vision transformers. The works included are organized chronologically in order to demonstrate the evolution of these methods.

2.1 Survey on Watermark Removal

In the following text, several methods for watermark removal are presented, starting with traditional algorithmic methods and following up with deep learning approaches.

2.1.1 Algorithmic Methods

Up until recently, watermark removal methods were limited to a single image scenario, without learning from multiple examples [22, 23, 24]. These methods are mostly based on low-level features (such as textures, edges, individual pixel colors etc.) without exploiting high-level semantic information. While delivering promising results, recent breakthroughs in computer vision tasks using deep learning would suggest a data-driven approach to be more generally applicable.

The authors of [21] propose a learning method of attacking watermarks through learning the specific watermarking process. Under the assumption that all images have the same watermark, the authors show that after collecting enough samples it is possible to isolate their watermarks and use them to infer the embedding process. With this knowledge, the authors present a way to invert the process, effectively allowing them to attack any previously unseen image with the same watermark. This approach is highly dependent on nuances such as scale, image resolution, and the watermarking process being exactly the same and deterministic. While the method proves to be very effective for attacking stock images from public databases, it is unsuitable for arbitrary watermark removal.

In a recent publication [25], a new interactive method for manipulating image data is introduced. Unlike traditional methods, it doesn't need any labeled training data as it's based on building a complete representation of the image using a morphological tree. This allows users to perform various image editing tasks, including object editing, semantic segmentation and visible watermark removal. However, due to the need for user guidance and long preprocessing times, this method is not suitable for large-scale use.

2.1.2 Deep Learning Methods

Each section of the following text corresponds to a single publication on the watermark removal topic. The vast majority of the deep learning methods presented rely on common convolutional operations as the basic building block of the networks.

2.1.2.1 Large-Scale Visible Watermark Detection and Removal

One of the first methods to employ a large-scale dataset for the task is [26]. The authors introduce the *Large-Scale Visible Watermark Dataset (LVW)*, which was synthesized using tens of thousands real-world images from the *PASCAL VOC 2012* dataset [27] and 80 different gray-scale watermarks. To show generalization of the method, the authors avoid using the same watermark objects in training and testing subsets.

The proposed model is built as a straightforward two-stage architecture. The first stage detects and locates the area of the image where the watermark was placed. The second part attempts to reconstruct the original image from the cropped patch around the watermarked area as an image-to-image translation using a convolutional encoder-decoder architecture based on *U-Net* [28]. While the original experiments were limited to single colored gray-scale watermarks, the model showed very promising results.

2.1.2.2 Blind Visual Motif Removal

The approach introduced in *Blind Visual Motif Removal (BVMR)* [29] consists of using a single encoder network to obtain a latent representation of the input followed by three parallel decoders, where each serves a different purpose: one decoder outputs a full reconstruction of the image, one outputs a binary mask specifying the watermarked pixels and one outputs an estimation of the watermark.

In terms of the notation introduced in Section 1.3.2, the model tries to directly estimate all of the matrices X, M and W from its input $\hat{X}$. The final result is then composed using the estimated mask to select pixels from either the input image $\hat{X}$ or the reconstructed estimation of X. The authors synthesized their own dataset, which included watermarks with either colored text or simple clip-art images.

2.1.2.3 Photo-Realistic Visible Watermark Removal with cGAN

The authors of [30] present a GAN based solution aimed to reduce a noticeable loss of clarity during watermark removal using previous methods. The essence of the approach lies in using the generator and the discriminator networks, which are trained jointly and thus improve each other. The generator is conditioned with a watermarked image on its input and is trained to output the unwatermarked reconstruction as a convolutional image-to-image model. The discriminator is a classifier trained to differentiate between real watermarked images and generator's reconstructions. Training this model using a conventional reconstruction loss, perceptual loss and the mentioned adversarial loss leads to superior results compared to previous state-of-the-art. While the authors mention improving the results with regards to real-world diversity of watermarks, their results are shown only for the *LVW* dataset with 80 randomly scaled and positioned gray-scale watermarks.

2.1.2.4 Two-Stage Visible Watermark Removal Architecture

Motivated by observing that there has been little work put into removing watermarks of colors other than white, authors in [15] synthesized two datasets for their use case. One contained transparent white watermarks, making it similar to *LVW*, while the other contained watermarks with the color set to be the average color of the pixels the watermark is embedded into. The latter approach leads to watermarks blending in with the background more seamlessly, thus making them more difficult to detect and remove. Similarly to [30], the architecture is based on the GAN architecture, but the authors present a more sophisticated generator due to recognition of similarities to image inpainting. The generator is divided into two stages, where the first stage serves only the purpose of watermark extraction and removal while the second stage is used as a refining stage, using insights from the image inpainting task. In both settings, the proposed method outperformed previous approaches. The datasets are not publicly available.

2.1.2.5 Split then Refine

A significant improvement for the quality of watermark removal was presented in [14]. Taking inspiration from other works based on multi-task learning [31, 32, 33], the authors argue that having the model learn to do multiple tasks at once might lead to an overall improvement in performance compared to learning a single task. The name of the work is *Split then Refine*, which indicates the model's two-stage architecture. The multitask nature of the process is employed in the first part of the network, the *SplitNet*. It comprises a shared encoder and three decoders for each of the tasks. The weights of the first layers of the decoders are shared across all decoders, but their importance is re-weighted through task-specific attentions.

The three tasks being solved jointly by the *SplitNet* are: watermark detection, watermark removal and watermark recovery. The watermark detection task outputs a mask, the watermark removal task outputs a reconstruction of the occluded image, and the watermark recovery task outputs the estimation of the original watermark. This makes it architecturally similar to the model presented in [29]. While the last task is not directly needed, authors show that providing the model with additional information about the watermark's original appearance during training and having the model learn to reconstruct it improves the performance for the other two tasks. The second part of the network is called *RefineNet* and its purpose is to take the coarse result (created by replacing the part of the input by the reconstruction in areas given by the estimated watermark mask) and refine it with the goal of removing artifacts, improving textures and giving an overall seamless reconstruction for the originally watermark area.

Since the training process utilizes additional new information compared to previous work, no dataset had all the required information directly available. This led to the authors synthesizing several novel datasets: *LOGO-L*, *LOGO-H*, *LOGO-Gray* and *LOGO30K* (based on the *VAL2014* subset of the *MS COCO* dataset [34]). The first two datasets use colored watermarks and differ in difficulty, namely by the relative sizes and opacities of the embedded watermarks. *LOGO-Gray* is motivated by the observation that gray-scale watermarks are very common in the real world and it is reasonable to evaluate the model in a setting focused on this scenario. The last dataset is the largest and aims to be diverse regarding aspects such as watemark sizes, opacities and locations. All of the mentioned datasets are publicly available[1]. The results of the experiments on all of the evaluated scenarios suggest this approach to be vastly superior to previous works.

[1] https://github.com/vinthony/deep-blind-watermark-removal

2.1.2.6 Watermark-Decomposition Network

It is not uncommon in scientific discoveries for multiple people to arrive at similar solutions to the same problem through comparable approaches [35, 36]. Just one day after the *Split then Refine* model [14] was released, the authors of [16] published a model with a very similar approach (both for dataset generation and the architecture setup) called *Watermark-Decomposition Network (WDnet)*. Their model did not handle the multioutput from the first stage of the model as intricately as [14] did and nearly all of the weights are shared among the output decoders, except for the last output layer. This work's two stages used for watermark removal and refinement are called *DecompNet* and *RefineNet*. Their inputs, outputs and general role are nearly identical to the previously mentioned model. In addition to this two-stage architecture, the authors introduce a discriminator to supply an adversarial loss during the training.

Similarly as in the previous work, the authors needed to synthesize a new dataset, because the publicly available *LVW* was unsuitable for their training pipeline. The dataset was named *Colored Large-scale Watermark Dataset (CLWD)* and is presented as a successor to the *LVW* dataset, as it contains images overlayed with 200 different colored watermarks. The dataset was publicly released for further comparison[2].

Since *Split then Refine* and *WDnet* were released at the same time, there was no direct comparison available to judge which approach is better, and both of the approaches claimed to be a new state-of-the-art method. It was not until [37] was published, where a direct comparison between the two methods was made on *WDnet*'s *CLWD* dataset. Unfortunately, the authors of the comparison found an error in the evaluation code for *WDnet* and subsequently discovered its real performance to be significantly inferior to that of *Split then Refine* on its own dataset.

(a) Input	(b) Watermark	(c) Original	(d) *SLBR*	(e) *SplitRefine*	(f) *BVMR*	(g) *WDnet*	(h) cGAN

Figure 2.1 Visual comparison of watermark removal methods presented in [37].
Left to right: The watermarked image on the input, the embedded watermark, the original undamaged image, *SLBR* [37], *Split then Refine* [14], *BVMR* [29], *WDnet* [16] and the cGAN approach from [30].

[2] https://github.com/MRUIL/WDNet

2.1.2.7 Self-Calibrated Localization and Background Refinement

The most recent state-of-the-art method is called *Visible Watermark Removal via Self-calibrated Localization and Background Refinement (SLBR)* [37] and it outperforms both [14] and [16] by a significant margin. The experiments were evaluated on the *LVW* and *CLWD* datasets and demonstrate the *SLBR* model outperforming all previously mentioned approaches in all considered metrics (for visual comparison, see Figure 2.1).

The main contributions of this method lie in additional information propagation using skip-connections between the coarse and refinement stages, as well as between the watermark separation tasks in the first stage.

Several novel architectural blocks are introduced, including a mechanism for refining the estimated watermark mask and using it to guide the background reconstruction. The refinement stage of the network is implemented via architecturally repeating the refining feature-fusing block N times, which has been to shown to improve the result with growing size of N (the authors use $N = 3$).

While the method has demonstrated superiority to previous approaches based on benchmarks on publicly available datasets, and we consider it to be the current state-of-the-art method, it requires a significantly more intricate architecture and is computationally more demanding.

2.2 Survey on Transformers

Nowadays, many deep learning tasks are receiving new state-of-the-art results thanks to the transformer model architecture and its derivatives [**openai2023gpt4**, **kocian2021siamese**, 38, 39, 40]. Originally designed for solving NLP problems without using recurrent relationships, transformers proved to be able to model useful robust relationships and leverage more information than previously possible. In its core lies the self-attention mechanism, which enables modeling of pair-wise relationships between different parts of its input.

In order to be compatible with the architecture, the input is split into tokens of a fixed size which serve as the base block for finding the relationships. For NLP tasks, the tokens are some indivisible parts of a sentence such as letter n-grams, words, punctuation or special sequence control symbols (delimiters, line breaks, et cetera). For computer vision, it is also possible to employ similar concepts for image inputs. While there are some architectural adjustments needed, the approach is proving to be competitive to traditional convolutional methods (CNN) [38, 39, 41] in either performance, computational complexity or both.

2.2.1 Core Concepts

In the following sections, we will introduce the concepts behind the transformer architecture and capture its evolution to the domain of computer vision and image processing.

2.2.1.1 Attention Mechanism

The name itself is very descriptive for the purpose of the mechanism, as its motivation lies in introducing a way for machine learning methods to preserve and focus on important pieces of information and put less of an emphasis on insignificant information.

While similar approaches were presented earlier, a major work [42] used attentional mechanism for solving neural machine translation (demonstrated on French-English language pair). The method is based on an sequence-to-sequence encoder-decoder architecture, where both the encoder and the decoder are certain recurrent models (RNN). In previous works [43, 44], the vector representation is the output of the encoder after processing the last input token, which possibly leads to the vector encoding less information from the beginning of the input, as the newly incorporated tokens might degrade previously encountered information. To solve this issue, the authors present a method which builds a context vector for each of the input tokens and enables the decoder network to “attend” to different input context vectors with different weights for each output word. There are many methods for calculating the context vectors and weights, but the general idea of weighing vector representations of input tokens proved to be very useful as an enhancement to RNN models for overcoming their shortcomings.

More generally, given an input $\mathbf{x}$ split into discrete tokens x_i and some vector embedding $c(x_i) \in \mathbb{R}^n$, we can compute the importance w_i of each x_i in a given context. Usually we require that $\sum_i w_i = 1, w_i \in [0, 1]$, which is typically done using the softmax function. The final output of the computation is then obtained as the convex combination of $\sum_i w_i c(x_i) \in \mathbb{R}^n$.

The input embedding function c is some trainable mapping, such as a dictionary map or a neural network. The weights w_i are recalculated in each output iteration, as each output token attends to different locations of the input. There are many different ways of obtaining the attention weights w_i [45], but they are generally in the form of a scoring function between current output's hidden state and an embedding of the considered input token.

Building upon success of the approach, new methods solving other related tasks using attention were introduced, even related to computer vision. In [46] the authors present an image captioning model, which uses attention mechanism to describe the contents of an image using natural language. They used a CNN to extract a small-resolution feature map of the image, which was then attended to during a RNN-based generation of the output text description.

2.2.1.2 Self-Attention

Originally introduced in [47], the authors show that attention is useful not only for attending to the input when considering a particular output token, but also when relating different input tokens against each other, giving rise to the self-attention mechanism. Using this idea jointly with a LSTM network to solve tasks such as language modeling, sentiment analysis or natural language inference, the authors are able to build general contextual representations for the input sentences which perform on-par with or better than state-of-the-art methods at the time.

In practice, the self-attention module is a transformation from an input sequence of length n to n context vectors, each calculated as the aforementioned convex combination. Its output can encode both global and local relationships within the input sequence, as opposed to previously modeling relationships between some context, i.e. the output sequence, and the input sequence. This concept eventually proved very powerful and is a crucial component of transformer models.

2.2.2 Transformer

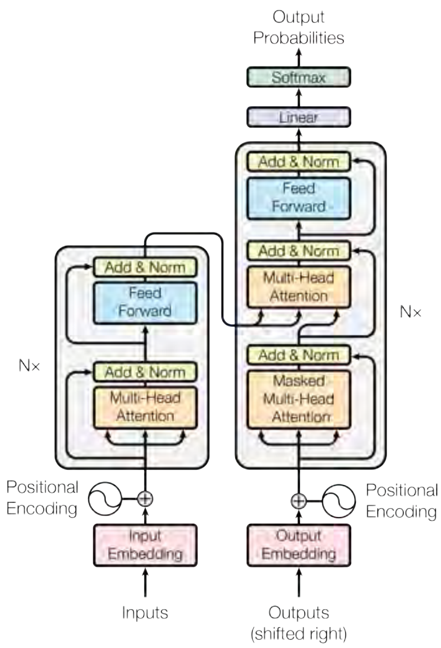


Figure 2.2 Visualization of the transformer architecture as presented in [48].

The attention mechanism proved to be a strong enhancement, but up until now, it still relied on working jointly with a different neural architecture, usually a recurrent network. RNNs inherently work in a sequential manner and are thus hard to effectively parallelize and scale.

A notional revolution in the field occurred with release of the *Attention is all you need* [48] paper, as it introduced the transformer architecture targeted towards NLP tasks, specifically sequence-to-sequence language translation. The major change against previous efforts is that it relies solely on relationships computed through attention mechanisms instead of recurrent information flow. The setup of the architecture allows significantly higher parallelization, as the input is not processed sequentially and the computation can be performed concurrently in all the positions.

2.2.2.1 Architecture

The network keeps the general encoder-decoder architecture, where the task of the encoder is to consume input tokens $x_1, x_2, ..., x_n$ and output their vector representations $z_1, z_2, ..., z_n$ with the goal of extracting useful features and relationships. The decoder's task is to generate output sequence $y_1, y_2, ..., y_m$ based on the encoder's outputs in a sequential manner, which means that other than considering all available z_i values at once, it is also auto-regressively conditioned on its own previous outputs.

As we can see in Figure 2.2, both encoder and decoder are built using common building blocks, namely:

- **Input/Output embeddings** – Learnable embeddings from the data domain (e.g. words) to a vector representation.
- **Positional encoding** – A method presented in the original paper to encode the token's position directly to its vector embedding, originally performed though adding specific sine or cosine terms to the vector. This is required since its relative location to other tokens would be lost in the self-attention calculation, which is undesirable as the semantics of the token heavily depend on its location in the input sequence. There are other ways of dealing with this problem in other use cases or domains [39, 49, 40].
- (Masked) **Multi-Head Attention** – The crucial component which models attention relationships between all of its input's locations. A single block of multi-head attention performs multiple self-attention calculations, each head with different $\mathcal{Q}, \mathcal{K}, \mathcal{V}$ transformations (described in Section 2.2.2.2). The goal of employing multiple heads is to increase the modeling capacity of each layer and thus capture different sorts of relationships. The authors claim this setup is superior to that of a single larger head with the same number of parameters. Implementation-wise the handling of all the heads is performed within a single matrix, which allows the computation to be more efficient. During the decoding phase, some of the computed elements are zeroed out in order to prevent attending to tokens yet to be generated.
- **Feed Forward** – A block consisting of a fully connected network with a nonlinear activation function. It is typically placed after the self-attention calculation, as it serves the role of compositing results of the previous computations in a non-linear manner, which is crucial for building strong representations. There is a single set of neural weights used independently in all of the input locations, which makes the calculation highly parallelizable.
- **Add & Norm** – There is heavy usage of residual connections throughout the architecture. These allow data from previous layers to be combined with outputs of current layers, with the motivation of allowing the model to transfer previously learned representations into deeper layers. The outputs are then normalized through a method outlined in [50] in order to improve stability of training and reduce convergence times.
- **Linear & Softmax** – In order for the decoder to be able to generate tokens, there is a linear classifier with a softmax activation as the last layer before each decoder's output, which tells us the probabilities for outputting any given token (i.e. a probability distribution over indices from a predefined dictionary).

As can be seen in the visualization in Figure 2.2, the encoder block is architecturally repeated N times until the final representation is built and the same applies for the decoder before outputting its result.

2.2.2.2 Query, Key and Value

Introduced along with the transformer model, a new interpretation and generalization was presented for the (self-)attention calculation. It is based on a loose analogy to database retrieval, where we have a query to use as a search criteria, keys to match it against and values to return to the user. For usage in attention computation, all three of these components are specific vectors, obtained either as a direct embedding of an input/output token, or as a result of a previous layers within the deep learning model.

For attention calculation with respect to a single location (meaning a single query), the semantics are:

- **Query** – The vector representing our current location or token of interest. The goal is to produce weights w_i that reflect the level of influence that each part of the data has relative to the query.
- **Key** – The key vectors represent the data we are attending to and are used for measuring similarity with the query. The attention weights w_i are calculated solely based on the query and the keys. We usually require that queries and keys have the same dimension.
- **Value** – The value vectors are coupled with the key vectors and they are the same dimension as the output of the attention is. The overall result is the convex combination of value vectors and attention weights.

Many different methods can be used for calculating the similarity between keys and values, but a scaled dot-product was used in the original paper, i.e. the attention weights are a result of softmax being applied to a dot-product of the query and keys, scaled by a certain factor related to their dimension.

In order to perform multiple queries at once, we construct matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ as containing the column vectors of queries, keys and values, respectively. Then we can write the general attention calculation presented in [48] as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{k}}\right)\mathbf{V},$$

where k is the dimension of the query and key space.

For the self-attention setting, assuming we have a tokenized input $x_1, x_2, ..., x_n$ embedded into m dimensional space, we can represent it as a matrix $\mathbf{X} \in \mathbb{R}^{n,m}$. We can perform three different embeddings $\mathcal{Q}, \mathcal{K} : \mathbb{R}^m \mapsto \mathbb{R}^k$, $\mathcal{V} : \mathbb{R}^m \mapsto \mathbb{R}^v$, where k is the query/key dimension size and v is the values/output dimension size. Considering strictly linear transformations, each of these mappings can be expressed as a matrix $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{n,k}, \mathbf{W}_V \in \mathbb{R}^{n,v}$. Finally, we can calculate the overall self-attention value as

$$\text{Self-Attention}(\mathbf{X}) = \text{Attention}(\mathbf{X}\mathbf{W}_Q, \mathbf{X}\mathbf{W}_K, \mathbf{X}\mathbf{W}_V).$$

The vector representations computed by stacking various attention-based layers with additional non-linear transformations in the transformer lead to a language model with strong capabilities, eventually achieving state-of-the-art results in many areas.

2.2.3 Vision Transformer

Up until adapting the transformer architecture for the image domain, CNN models dominated among newly proposed methods for computer vision problems. While the usage of attention-based components in neural networks saw several successful uses in image-based tasks, they usually were a mere enhancement to existing convolutional approaches [51, 52, 53, 54].

Motivated by the success of replacing RNNs by self-attention mechanisms for NLP problems, the authors of *An image is worth 16x16 words* [55] present a way to abolish convolutional operations through adapting the transformer architecture for image classification.

Giving a rise to *Vision Transformer (ViT)*, the authors adapt architecture setup from original transformer [48]. The main challenge to overcome was that, originally solving NLP problems, the transformer input is inherently structured as a sequence of tokens. In comparison, image data are typically represented as a 2D matrix of RGB pixels. The most straightforward way to feed an image to a transformer model is to perform per-pixel tokenization. This is practically infeasible, as each of the pixels on its own holds little semantic value and their amount in a single image leads to them being computationally intractable.

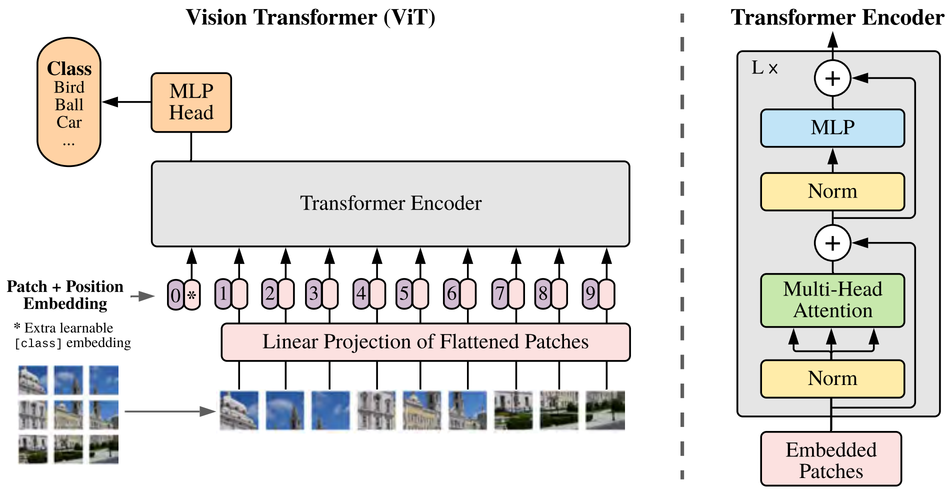


Figure 2.3 Visualization of the *ViT* architecture as presented in [55].

The approach the authors choose is to split the image into patches, each having the size of 16×16 pixels. Such a way of splitting up the data both increases the semantic information of a single patch and decreases the overall number of tokens. These 16×16 patches of 3-channel pixels are then flattened into vectors and linearly embedded in a very similar manner as within the original transformer architecture.

In direct comparison to the original transformer architecture, there are some minor alterations, but the overall information flow in the *ViT* model stays nearly the same. There is no need for a complex decoder, a simple classification head suffices, as the desired output is a single summarizing prediction rather than a dense sequence of tokens.

In order to perform the classification step, a special learnable `class` token is prepended to the linear embeddings of the patches before entering the encoder. After passing through the encoder layers, a fully-connected classifier attached to the special token's location decides the output.

Another difference is the order of operations within the encoder block. The core layout stays the same, except for the normalization step taking place before both the attention calculation and the additive residual connection. The authors of [56] suggest this order of operations to lead to better-behaved gradients, increased stability and consequently lowered training time requirements.

The authors conclude that on canonical image-classification datasets, the newly proposed method often comes short in a direct comparison to previous CNN efforts. The main culprit is identified as inductive biases of the CNN, where they are seemingly more suitable for the task due to their inherent locality, 2D structure and translation equivariance. Nevertheless, the authors are able to overcome these biases and beat the previous model architectures by pretraining *ViT* on a large image dataset and later fine-tuning it for smaller tasks of interest. This type of transfer learning is shown to perform better than state-of-the-art transfer learning methods applied to CNN classifiers. This would suggest the new vision transformer to have a high overall learning capacity, while requiring large training datasets to leverage it effectively.

2.2.4 Transformers as Computer Vision Backbones

The transformer architecture had proved to be suitable for computer vision as well, although it was yet to be shown usable for other than classification tasks. The works presented in this section have the goal of modifying the transformer architecture to be able to serve as a general backbone for image-related tasks. This follows directly from CNN architectures, where a single general architecture can be used in various scenarios and is generally called a *backbone* and serves the role of extracting features from the input image.

The main obstacles in adapting the transformer to tasks with complex dense predictions such as semantic segmentation, object detection or image inpainting are the inherently higher granularity requirements on the input and its subsequent processing. The original *ViT* model has poor scalability since the time complexity grows quadratically with image size, as well as it suffers from a loss of fine-grained detail due to smallest units being 16×16 patches.

2.2.4.1 Pyramid Vision Transformer

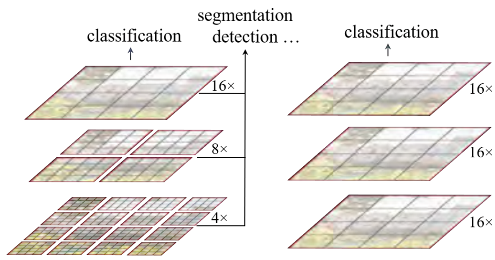
The Pyramid Vision Transformer (PVT) model [57] involves a shrinking operation that is repeatedly applied to the input image as it traverses through the network. This builds a progressive pyramid scheme of patches of decreasing resolution and increasing embedding size, which enables the method to output a high resolution prediction. The idea is similar to the way CNNs extract multiscale features, where with the increase of network depth, the channel dimension gradually grows, and the output resolution progressively shrinks.

The model's encoder replaces multi-head attention layer with a proposed spatial-reduction attention, which reduces spatial dimension of the keys and values during the calculation. This makes the model less intensive on both memory and computational resources.

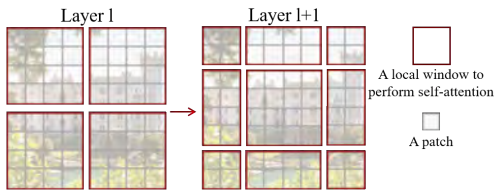
PVT is compared to the previously proposed *ViT* through extensive experiments, where the output feature maps are used jointly with standard detectors. The results show that *PVT* outperforms both *ViT* and state-of-the-art CNN backbones in many downstream tasks, including object detection, instance segmentation and semantic segmentation.

2.2.4.2 Swin Transformer

In this work [49], the transformer architecture was also successfully adapted to image tasks of object detection and semantic segmentation, while effectively reaching state-of-the-art results in both of them. The authors argue the model is versatile and efficient enough to serve as a general backbone, which was made computationally possible through its hierarchical structure. The self-attention calculation is limited to local windows of patches with exponentially increasing resolution in each of the layers. The window partitioning boundaries are shifted in each layer, which allows previously disjoint areas to exchange information in the self-attention calculation.



(a) Comparison of *Swin transformer* patches (left) and original *ViT* patches (right) and their self-attention context



(b) The proposed partitioning scheme

Figure 2.4 Partitioning of self-attention calculation windows (outlined in red) of patches (gray squares) as shown in [49].

In Figure 2.4 we can see the comparison to *ViT* patches, where the self-attention was computed globally in each layer. The hierarchical approach of isolating the self-attention mechanism to smaller patches and later merging them through the shifted windows scheme has shown strong results in both performance and (due to being linear-time with respect to image size) computational complexity, as opposed to *ViT*'s quadratic runtime.

Another modification presented in this work is introduced through abolishing the absolute position embedding, which had been inherited from the original transformer almost unchanged. This work adds a learnable parameterized bias matrix $\mathbf{B}^{n,n}$, where n^2 is the number of images patches.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{k}} + \mathbf{B}\right)\mathbf{V}$$

The authors show that this learnable relative position bias performs better than explicitly constructing absolution position embeddings.

Both *PVT* and this work share similar architecture regarding shapes of data in each of the layers, which arise as an inspiration from CNNs. The main difference of the two approaches lies in the attention calculation, where this work splits up the calculation into smaller areas and *PVT* progressively reduces the sizes of patches.

2.2.4.3 PVTv2

Authors of the original *PVT* backbone saw several possible adjustments to the framework, which led to significant improvements in computational demands and qualitative performance. Giving rise to *PVTv2* [40], there were three major modifications introduced:

- a linear-complexity attention layer,
- overlapping patch embeddings,
- and the addition of a convolutional layer to the feed-forward block.

The motivation behind these adjustments lies in poor scalability of original *PVT* to large images. Reduction in the attention calculation complexity is achieved through replacing original spatial-reduction projection by an adaptive average pooling layer, which can output a fixed-sized output for an arbitrarily sized input. The complexity of the reduction is parameterized by the size of the output shape, rather than by the reduction's magnitude, which was the case for the original *PVT* and led to poor scalability.

Tokenizing the image into disjoint patches removes local continuity among them, which is partly alleviated by positional embeddings. To further improve upon local continuity between patches, a modification to the patch embedding process is presented. The proposed method allows for overlaps in neighboring patches. Specifically, the patch size is increased to contain approximately a half of its left neighboring patch and a half of its right neighboring patch. The same applies for vertical neighbors. The number of patches stays unchanged, so the effect is equivalent to increasing the receptive field for each of the patches. The image is zero padded as necessary to allow for calculation around the borders.

Absolute position encoding has a fixed dimension for images of all sizes and is thus unsuitable for inputs of varying scales. To address this problem, the authors propose an additional convolutional layer to be added to the architecture of the feed-forward block. This is motivated by the fact that convolutional layers used jointly with zero-padding have been shown to encode positional encoding implicitly [58].

2.2.5 Task-Specific Transformers

To the best of our knowledge, nobody has yet adapted a transformer-based model to the task of watermark removal. That aside, there has been a variety of specialized architecture variations for other specific use cases. This section aims to show some successful application of vision transformers to tasks similar or related to watermark removal.

2.2.5.1 Segformer

Although the previously presented general backbones have been shown to perform well when attached to standard segmentation detectors, they posed only as an encoder. The authors of *Segformer* [39] use a hierarchical *PVTv2*-like encoder and a simple MLP decoder to solve the semantic segmentation task, which requires outputting a fine-grained per-pixel classification.

The encoder is pretrained using the *ImageNet* dataset [59] on a classifying task to later serve as a starting point for the model training on a segmentation dataset of interest. Several variants of the model are presented, differing in the number of parameters. The results on various datasets suggest robust performance, as *Segformer* outperforms previous transformer and CNN methods in comparative regimes regarding parameter count and required computational resources.

One thing to note is that image watermark detection is equivalent to performing a semantic segmentation for two classes – watermarked section and undamaged section. This might suggest that the transformer architecture is suited for at least detecting the watermark, as this is one of the tasks directly solved by some of the methods presented in Section 2.1.2.

2.2.5.2 Mask-Aware Transformer for Large Hole Image Inpainting

In [38], the authors adapt a transformer as the main backbone (based on *Swin transformer* [49]) for inpainting images, i.e. filling in a user-specified area in an image as seamlessly as possible. Exploring the approaches suggested by the authors seems encouraging for the scope of our work, as there are notable similarities between image inpainting and watermark removal (see Section 1.3.1).

The shapes of image data passing through the model are first decreasing and then increasing in resolution, being reminiscent of a CNN encoder-decoder architecture. The transformer block's attention is adjusted to account for invalid tokens, which are initially specified by the input's inpainting mask. Other than having transformer body, the model also uses convolutional layers, specifically as head and tail blocks of the whole architecture, as well as using them for the downsampling and upsampling operations on the image between transformer blocks.

Alongside the main transformer calculation, a style manipulation module is proposed to enable the outputs to be diverse and have various styles for unchanging fixed inputs. This is achieved through an additional noise input and modulating the weights of convolutional layers, thus enabling plurality of the model's output.

The model is trained with a loss consisting of three components:

- **adversarial discriminator loss**, i.e. performing a GAN-like training,
- **gradient penalty for the discriminator**,
- **perceptual loss**, i.e. minimizing the distance between hidden layer activations of a pretrained convolutional network between real and inpainted images.

This work achieves strong and diverse results on publicly available image datasets and is presented as a state-of-the-art method, dominating previous (mostly CNN) approaches in a comparable setting.

Chapter 3
Dataset

The image dataset specified in, and provided as part of this thesis' assignment contains images from car ads gathered from the internet. It is a large collection of images, coming from various sources and thus offering a significant variety in appearance and content. In this chapter, we will discuss the different types of images in the dataset, what features they contain, the steps performed for filtering the data and its usage for solving the watermark removal task.

3.1 Dataset Analysis

The car ad image dataset consists of millions of images collected from various online marketplaces. They include advertising photos taken and digitally processed by professional photographers at premium dealerships as well as amateur photos taken by low quality cameras.

3.1.1 Types of Images

The dataset includes a variety of images related to cars and after manual inspection, we decided to categorize the images into several distinct groups:

- **exterior** – Photos displaying a vehicle exterior, where the entire car or most of the car is visible.
- **closeup exterior, wheel** – Closeup view of a feature on the outside of the car such as lights, handles, registration plates, et cetera. Wheels are separated from closeup photos, as they might be more useful for other use cases, such as estimating the car's price.
- **interior** – These photos provide an inside look at a vehicle's interior features such as seating arrangements, dashboard layout and design, audio/navigation systems and others.
- **trunk, engine** – Photos showing either storage compartment or engine, as these can be both interior or exterior photos, depending on the camera's angle and position.
- **miscellaneous** – Photos not showing the car at all (dealership ads, documents, keys, ...).

With the goal of creating a homogeneous dataset for a practical usage, we eliminate some of the diversity by focusing solely on exterior images. These are often the first to be seen in the ad and are the most prone to containing an obstructing watermark.

(a) exterior **(b)** closeup exterior **(c)** wheel

(d) interior **(e)** trunk **(f)** engine

(g) miscellaneous

Figure 3.1 Samples from the recognized categories of images in the dataset.

3.1.2 Dataset Filtering

In order to isolate suitable samples for generating the training datasets, we must first classify the available images based on some criteria. This section describes the process of filtering the images via defining the required classification tasks, manually labeling a subset of the images and then training suitable classifier models to obtain labels for the entire dataset.

3.1.2.1 Classifier Network

As the classification model, a pretrained instance of a convolutional *RegNet* [60] architecture was used, in particular we chose the `regnet_y_32gf` configuration as it was suggested to be the most powerful one by its authors. We used a publicly available implementation of the model provided by `torchvision` and initialized the model using weights on pretrained on the *ImageNet* dataset [61, 59] as a starting point for fine-tuning on our data.

The following preprocessing steps were applied to the dataset before training the classifier:

1. Delete possible bit-perfect duplicates for each image.
2. Drop potential transparency layers.
3. Apply padding to a square shape to prevent content distortion on resizing.
4. Resize the image to 224×224 (as required by the *RegNet* model).

To suit our needs, the last fully-connected layer from the pretrained model is replaced to match the number of classes in a given classification scenario. While we could work with a frozen *RegNet* backbone and train only the last classifying layer, we instead fine-tune the whole model on our dataset in an effort to obtain better classifying accuracy. Our data is not distributed into classes uniformly, so we perform balancing to mitigate biasing the model towards majority classes.

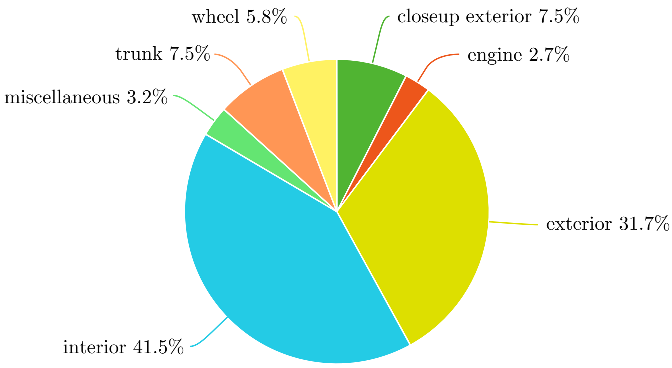


Figure 3.2 Visualization of the dataset composition.

3.1.2.2 Training the Model

We manually labeled over 10 000 image samples to the outlined categories. For fine tuning, we arrived to best results for the validation set with settings of batch size set to 16 and the Adam optimizer with a low learning rate of 10^{-5}. For the testing phase, the model's state was restored to the epoch with highest validation score during training.

The model achieved near 100% accuracy on the training set and 98.68% and 98.28% on the validation and testing sets, respectively. Upon inspecting the failure cases, almost all of them were either ambiguous images or errors made during manual labeling. We deemed these results to be sufficient to progress further and we used the trained classifier to obtain labels for the entire available dataset.

With the labeled dataset available, we filter out the desired exterior images. As the classifier can not be perfect, we leverage the output score of the model to interpret it as a confidence of the predictions. By limiting ourselves with a threshold of 90% confidence after passing through the softmax layer, we obtain a subset where the model is fairly sure of its prediction.

3.1.2.3 Watermarked Images in the Dataset

As expected, since we are dealing with real-life data, there is a lot of visual noise, including watermarks. Some examples of visual noise include but are not limited to:

- text watermarks,
- colored logo watermarks,
- opaque graphics,
- ad banners and miscellaneous contact information around the image,
- text and logos present on real objects surrounding the car,
- or censorship of registration plates.

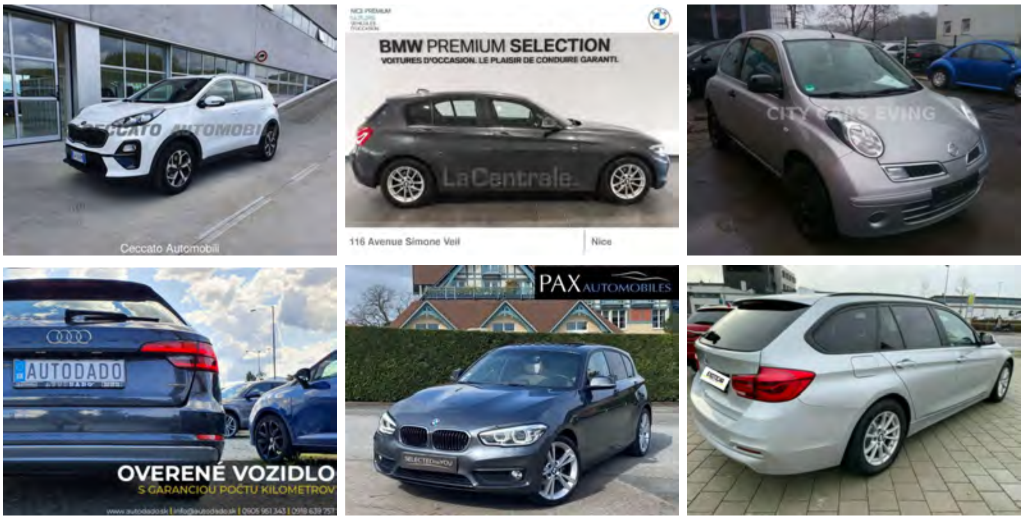


Figure 3.3 Examples of watermarks and other visual noise present in the dataset.

In order to train a model for removing watermarks, we need clean target images with no graphics inserted into them. For this purpose, we train a second classifier. Due to the wide range of visual noise present in the dataset, we manually label several thousand images into categories of:

- watermark,
- opaque graphics,
- ad banner,
- or clean image.

The process for training the classifier is the same as described in Section 3.1.2.1, except for not using a standard categorical cross-entropy loss with a softmax activation. This is due to the fact, that these categories are not mutually exclusive. Instead, we train 4 separate classification heads placed after a single common convolutional backbone as a binary classification task, one for each of the categories. For filtering, we require that confidence of a clean image is higher than 95% and confidence for other categories is less than 5%.

3.2 Watermark Synthesis Process

In order to obtain both watermarked images and ground truth images for training the model, we use a simulated watermark insertion process on the clean dataset to emulate real-world image watermarks. The types of image watermarks we synthesize are semi-transparent watermarks as these are empirically most common and most difficult to remove automatically (i.e. due to being placed in the center of the image). In order to train the model, we also create a mask that indicates the location of the watermark on the image and a mask describing its opacity.

The process begins by selecting a set of clean exterior images from the dataset and creating a simulated watermark for each image. Following the analysis of actual watermarks present in the dataset above, we propose synthesizing three types of watermarks. With the motivation for the model to generalize, we aim to capture enough characteristics of real watermarking systems.

3.2.1 Text Watermarks

Text watermarks are added to an image using a variety of different fonts, sizes, and colors. The pool of fonts used were gathered as a subset of the fonts available in a default installation of *Windows 10* [62], totaling nearly 100 different samples.

We generated the text as a string consisting of a random number of random characters from a predefined set of letters, numbers and punctuation marks. The font, size, and color of the text are also randomly selected, with a possibility of a colored outline as well. The watermark is then superimposed on the image at a random position, with a random transparency within a range from 20% to 80%.

Figure 3.4 Example of a generated text watermark.
In order: Watermarked image, watermark, transparency mask, binary mask.

3.2.2 Colored Logo Watermarks

This type of watermarking works by randomly selecting a logo image from a set of predefined watermark images, and then resizing and rotating the watermark to a random size and orientation. The transparency level of the watermark is also randomly selected, same as for text watermarks. The watermark is then overlaid on the image at a random position.

In order to prevent the model from removing only a certain type of watermarks, we gathered a set of image logos with transparent backgrounds. These logos are compiled from two sources:

- a publicly available Kaggle dataset [63] (after filtering out images with no alpha channel),
- and logos used in the *CLWD* dataset [16].

The resulting pool of available watermarks consists of nearly 5000 semi-transparent logos and is thus much more diverse in comparison to *CLWD*, which contained only 200 watermarks.

Figure 3.5 Example of a generated logo watermark.
In order: Watermarked image, watermark, transparency mask, binary mask.

3.2.3 Ad Banner Watermarks

Another type of watermark we consider is an ad banner. It consists of a banner overlaid on an image, usually containing a variety of different designs and colors. These fundamentally differ from logo watermarks, as they usually span the entire width of the image and are thus very intrusive, but positioned near an edge of the image.

To simulate this kind of intrusion, we used an advertisement dataset [64] and filtered only images, which had the correct aspect ratio (i.e. they were significantly wider than taller). This lead to a pool of over 10000 images to use for watermarking. These images are then randomly cropped to have a banner shape and are inserted either to the top or the bottom edge of the image with a random transparency level.

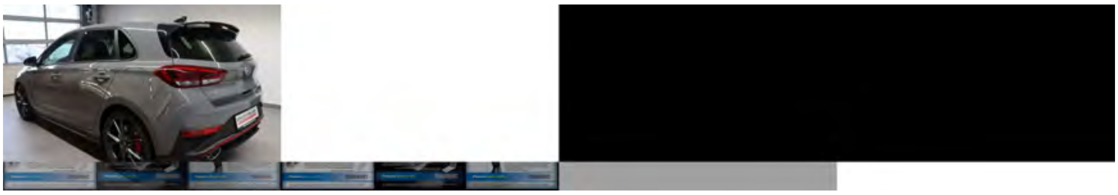


Figure 3.6 Example of a generated ad banner watermark.
In order: Watermarked image, watermark, transparency mask, binary mask.

3.2.4 Dataset Generation

We synthesize two different datasets, each with over 340 000 samples. The first dataset is simpler, as it imposes only text watermarks on the image. The second scenario uses combinations of all of the watermark types, with a possibility of overlaying multiple watermarks over one another. If this happens, the opacity mask is calculated in accordance with formulas presented in Section 1.3.2.1.

After obtaining the dataset, we can use it as input for our model's training pipeline and quantitatively verify its effectiveness at removing synthetic watermarks and empirically judge its performance on real-world data. More information about the training dataset format can be found in Section 4.2.1 and Table 5.2.

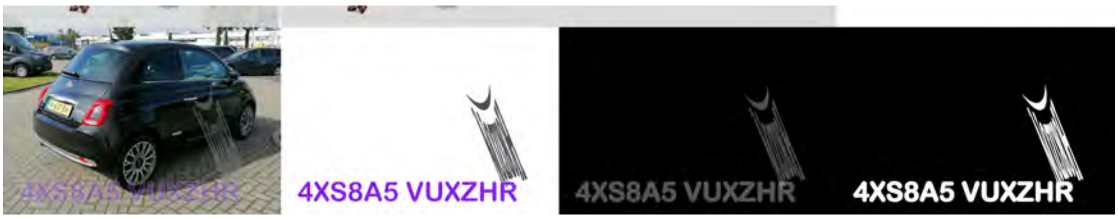


Figure 3.7 Example of combining all watermark types.
In order: Watermarked image, watermark, transparency mask, binary mask.

Chapter 4
Transformer Architecture for Watermark Removal

In this chapter, we propose an architecture for watermark detection and reconstruction. Our model utilizes a transformer-based encoder-decoder architecture and a refinement network with a GAN-like discriminator. The key components of the model and their functions are described, including a comprehensive summary of the training loop, including all the loss functions and related hyperparametrs.

4.1 Architecture

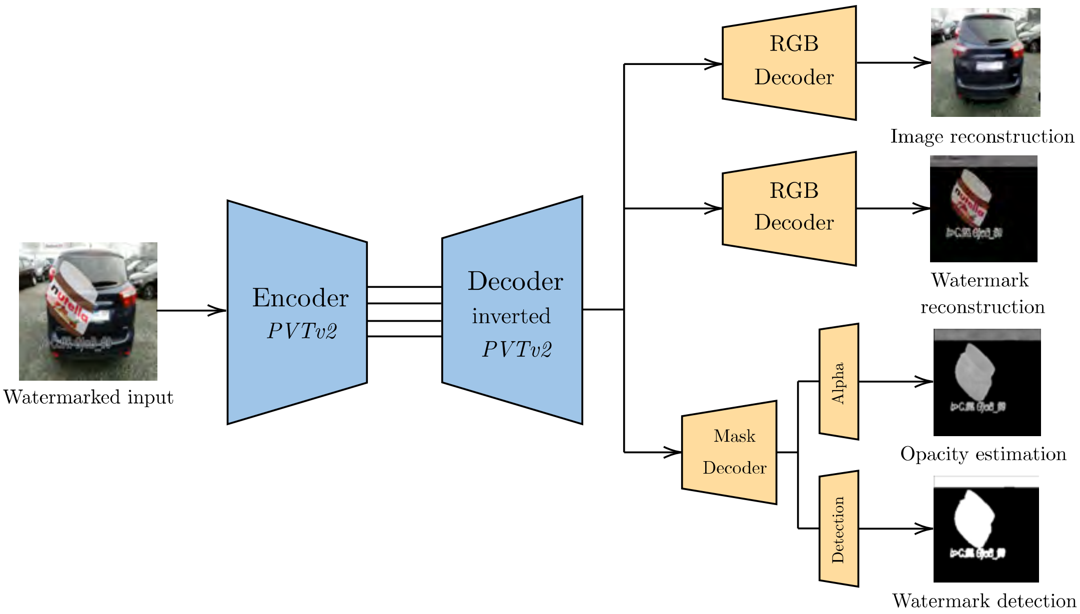
The following section will outline the key components of the model and their functions. The text is divided into three parts, each focusing on one of the major stages of the model.

4.1.1 Watermark Remover

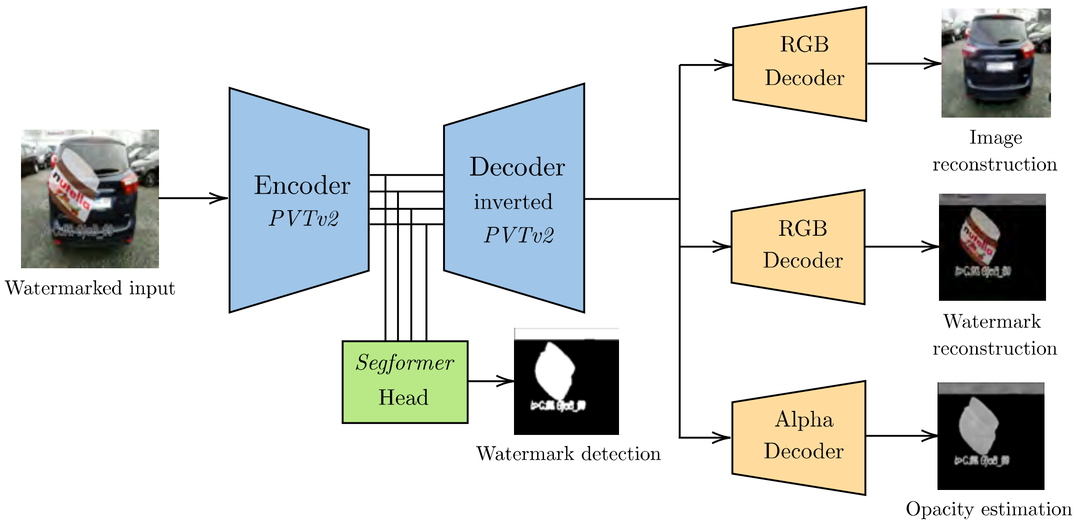
The first stage of the model is responsible mainly for detecting the watermark, and providing a rough estimate for further refinement. Namely, there are five tasks the model performs in this stage in order to provide input for subsequent stages:

- **Feature extraction** – Initial preprocessing of the image.
- **Image reconstruction** – Aims to recover the original image.
- **Watermark reconstruction** – Learns the appearance of the watermark.
- **Watermark detection** – Classifies each pixel in the image as watermarked or undamaged.
- **Opacity estimation** – Estimates the opacity used to embed the watermark.

Following the notation presented in Section 1.3.2, the module learns to provide an estimate for each of the X, M and W matrices.



(a) Variant with watermark detection performed via a convolutional decoder.



(b) Variant with *Segformer*-based segmentation detection.

Figure 4.1 Visualization of the two proposed architectures for the first stage of the model, comprising transformer modules (in blue), convolutional decoders (in orange) and MLP-based modules (in green).

4.1.1.1 Feature Extraction

The first stage of the model utilizes a transformer encoder-decoder architecture to construct features usable for watermark detection and rough estimation of components used for creating the watermarked image. Within our experiments, we utilize different variants of the *PVTv2* encoder, initialized using pretrained weights published by its authors [40]. We build the decoder by reusing the building blocks of the encoder, but with an additional upsampling step in the patch embedding to increase the spatial resolution in each of the blocks. This is similar to how the authors of [38] created a transformer decoder, although they used a different base model [49].

The encoder generates hierarchical feature maps at different resolutions for a given input image. For example, for a 256×256 input image, the encoder produces four outputs with a decreasing spatial resolution (i.e. 64×64, 32×32, 16×16, 8×8) and an increasing number of channels. The later outputs of the encoder contain higher-level representations, and we use the decoder to combine them with the lower-level representations using skip connections between the encoder and decoder. The decoder then uses these representations to generate its final output with a resolution of 64×64. This setup allows us to use a shared symmetrical setup of the encoder and decoder. The decoder's representation is spatially smaller than the original image, we use subsequent decoders for tasks such as watermark detection and image recovery to reach original resolution.

This module is highlighted in blue in Figure 4.1. While the data flow is consistent across all conducted experiments, various sizes of the model were evaluated to assess the impact on overall performance. Specifically, the size of the embedding dimensions used for projecting the patches in the self-attention calculation was varied across experiments (see Section 5.2.1).

4.1.1.2 Image and Watermark Reconstruction

Both image and watermark reconstruction modules share the same CNN-based image decoder architecture. The decoder applies a set of convolutional and upsampling operations to progressively increase the spatial resolution of the image. To introduce non-linearity, the model uses the *LeakyReLU* activation function in the convolutional layers, except for the last layer, which uses a *ReLU* activation. The module includes batch normalization after every convolutional operation.

The estimation of the watermark is not necessary during inference as the primary focus is on reconstructing the original image. Nevertheless, previous works suggest that training the model to recover the watermark as well improves overall performance [14]. These two modules are referred to as RGB Decoders in Figure 4.1 and maintain a consistent architecture across all performed experiments.

4.1.1.3 Watermark Detection and Opacity Mask Estimation

The task of estimating the watermark mask involves two subtasks: binary segmentation of the watermark and estimation of its alpha channel. For the opacity estimation subtask, we propose using a convolutional decoder with the same architecture as the image and watermark decoders, with the difference of it outputting a single channel rather than an RGB image.

For the binary segmentation subtask, we propose two approaches. The first approach is to treat it as an image decoding task using convolutions. For this scenario, first couple of layers and their weights are shared with the opacity estimation model, since the tasks are nearly identical. Only the final output layers are task-specific in this scenario.

The second approach is to directly leverage the hierarchical output from the *PVTv2* model and use a decoding head compatible with multilevel features, such as the output module from the *Segformer* model [39]. This approach has advantages of being lightweight, natively compatible with the *PVTv2* transformer, and being capable of achieving state-of-the-art results for semantic segmentation of objects in an image, which is a closely related task.

The two approaches and their differences are visualized in Figure 4.1. Each of them is shown in a separate figure, as the data flow is influenced by this architectural change.

4.1.2 Watermark Refiner

The previous stage of the model is capable of learning to differentiate between the watermark and the background image, as well as provide us with an estimation of the components necessary to synthesize the original input. Nevertheless, the reconstruction of the image is often blurred, color degraded or otherwise unsatisfactory to be directly used as a replacement for the damaged pixels. We will refer to the image reconstruction from the *Watermark Remover* stage as a "coarse" result.

The refining stage is designed to improve the fidelity and visual quality of the reconstructed image, allowing for more accurate and detailed results. This is achieved through a convolutional model based on the well-known *U-Net* architecture [28]. The module is built as a fully-convolutional autoencoder-like architecture with skip connections implemented via concatenating the features between the encoder and decoder.

The first step is to replace the watermarked pixels in the original image with the coarse reconstruction, which can be done using the binary mask provided by the watermark detection decoder.

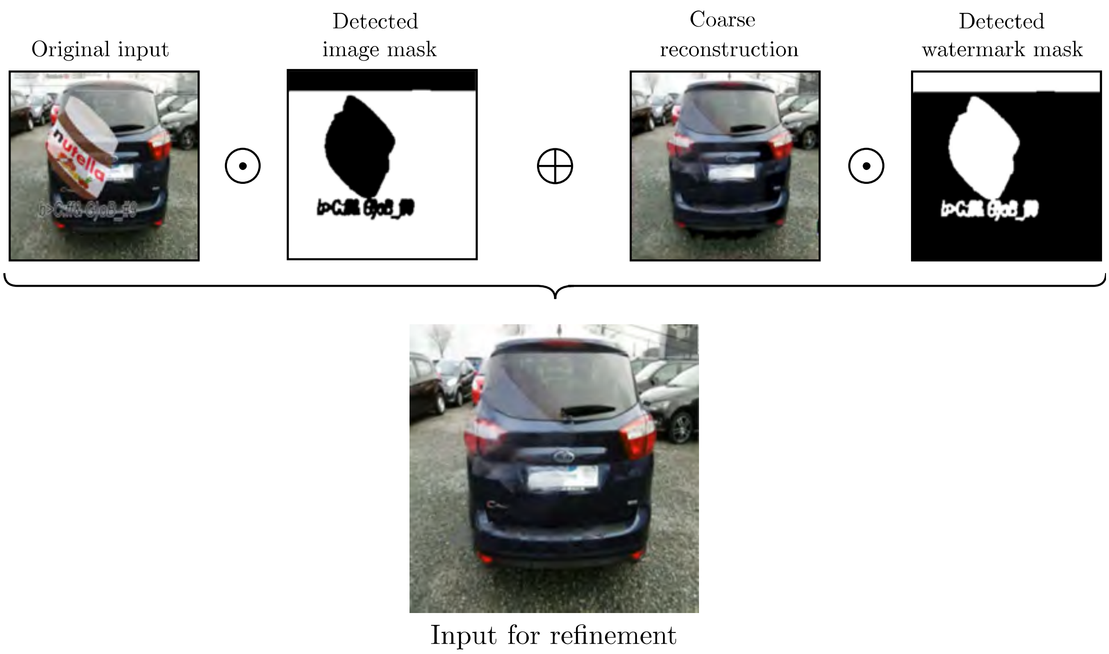


Figure 4.2 Composition of the coarse result and undamaged areas of the original image.

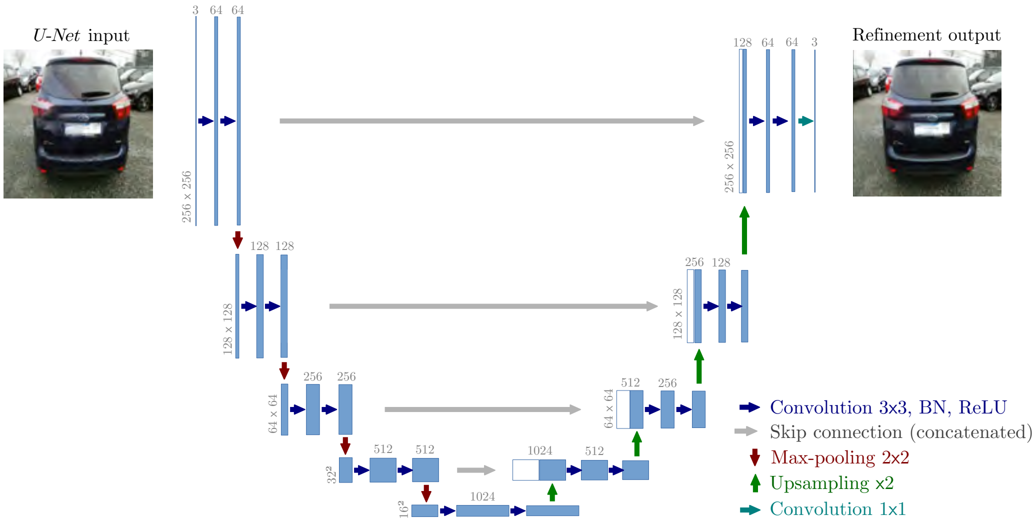


Figure 4.3 Architectural layout of the refinement network. Visualization created as a modification of a figure from the original *U-Net* paper [28].

The use of skip connections in *U-Net* is a key feature, which improves the propagation of high-resolution information. This enables the model to effectively leverage the undamaged parts of the image to create a more accurate and sharper replacement for the originally watermarked area. During the training, the refiner aims to reconstruct the entire image, but a significantly higher weight is placed on the objective of reconstructing the area given by the detected mask.

4.1.3 Discriminator

Finally, the authenticity of the refined image is judged by a convolutional discriminator. Leveraging an adversarial training mode, originally introduced in [17] (GAN), we train this module to classify between real unwatermarked images and reconstructions outputted from the refining stage. As the discriminator learns to recognize the difference between genuine and reconstructed image, the rest of the model is trained to confuse the discriminator. This effectively aims to increase the likeliness of the results to the original images, to help with the reduction of artifacts and improving the overall perceived authenticity of the reconstructed image. This approach is motivated by the prevalent use of GAN-based modules for other tasks related to image manipulation via deep learning [65, 38, 66].

Since the discriminator network is used only during training and can be completely discarded for inference, we decided to also include the ground truth watermark mask in its input. This decision was made to help the discriminator focus on the reconstructed parts of the image, as it is likely that most of the input is the same as the original unwatermarked image.

During the experiments, we compare two different variants of the discriminator network. For the first variant, we use a lightweight convolutional network, consisting of four strided convolutional blocks, each with a progressively increasing number of filters and decreasing spatial resolution. The blocks are equipped with a batch normalization layer, dropout and a *LeakyReLU* activation function. The final layer is a linear layer with a single output and a sigmoid activation function and its output is interpreted as a probability of the input being a real image.

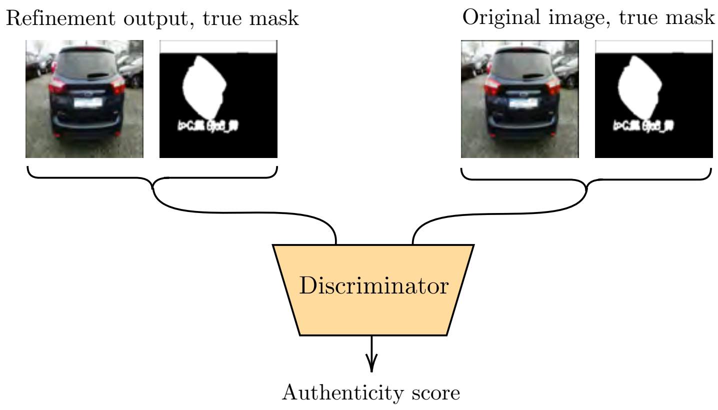


Figure 4.4 Discriminator's inputs and outputs visualized.

The other variant of the discriminator's architecture serves an identical purpose, but has a higher number of parameters and thus an increased learning capacity. It consists of 8 convolutional blocks and the general idea of the model is very similar to the first variant, except for being considerably larger. The exact architecture for the discriminator was borrowed from *SRGAN* [66], but we additionally perform spectral normalization (SN) on each convolutional layer [67].

4.2 Training Pipeline

This section outlines a training pipeline for the proposed architecture. The pipeline includes steps for the dataset setup, loss functions definition, hyperparameter selection, training loop and post-processing for inference. By following this pipeline, we were able to train the model to achieve the performance reported in Chapter 6. The specific details of each step will be discussed in the following subsections.

4.2.1 Dataset Setup

In order to train the model for watermark removal, it is necessary to obtain a dataset for watermark removal in a particular format. Specifically, the dataset should include watermarked input images, their corresponding clear ground-truth images, the watermarks themselves, their opacity masks, and their binary segmentation masks. All of these components should be resized to a common resolution (256×256 in our scenario).

The composition of the dataset and the watermarks should be diverse enough and should reflect the target real-world scenario. This is vital in ensuring that the network generalizes well and can be effectively utilized to attack real watermarked images. For example, if the dataset only includes watermarks from a single source, the model may not be able to effectively remove watermarks from other sources.

In Section 3.2.4 we discussed the process of synthesizing our training datasets composed of car images with diverse watermarks. Any other type of image can also be used as long as it follows the same format for the data. The resolution of images we chose was decided mainly for computational reasons and for simpler comparison with previous methods [14, 37]. Depending on the specific requirements of the application, one can increase the size of the images as necessary (with possible minor architectural changes to the model).

4.2.2 Loss Functions

The goal of training is to converge to model parameters which minimize the overall loss function and maintain the performance on the validation set. The overall objective loss function for the training process is given as a combination of different loss functions, each with a weight that determines its relative importance. Each of these loss functions constitutes a different aspect of the model's performance, all of which will be described in this section.

Please note that all the formulas presented in this section assume a single input and output for clarity, however, in reality, they relate to optimizing the expected value of the function over the data and are actually computed in batches of samples during training. The loss functions' input arguments are left out to keep the presentation brief, but it is assumed that the reader has a general understanding of their concepts and purpose.

4.2.2.1 Binary Cross-entropy

For watermark detection we experiment with two variants of architecture (see Figure 4.1), each with a different loss function. For the *Segformer* variant, we use a common binary cross-entropy loss, which is usually defined as follows:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})],$$

where N is the number of pixels in the image, y_i is the ground truth value of the ith pixel (either 0 or 1), and $\hat{y}_i$ is the predicted value of the ith pixel.

For our specific case of watermark removal, we recognize that the majority of pixels are usually undamaged, and we want the detection algorithm to find all of the damaged pixels, even at a possible cost of classifying some undamaged pixels as watermarked. To aid the training towards this goal, we use weighted BCE, where we increase the weight of the positive samples by a factor of λ_{pos}:

$$L_{wBCE} = -\frac{1}{N}\sum_{i=1}^{N}[\lambda_{pos} y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})].$$

4.2.2.2 Mean Error Losses

The L_1 or L_2 loss functions are commonly used for image restorations tasks. In our case, we also use them for watermark detection in the case of using a convolutional decoder, i.e. we treat the binary mask estimation problem as general regression rather than binary classification. These loss functions are defined as follows:

$$L_1 = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}|,$$

$$L_2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2,$$

where N is the number of pixels in the image, $\hat{y}_i$ is the value of the ith pixel in the output, and y_i is the value of the ith pixel in the target.

The L_2 loss corresponds to MSE and measures the difference between the watermarked and watermark-free images in terms of squares of errors of the pixel values, while the L_1 loss measures the absolute difference between the two images.

Intuitively, it may be expected that the L_2 loss would perform better as it places more weight on larger errors. Nevertheless, previous research [68] has demonstrated it often produces worse results compared to using the absolute L_1 for image restoration tasks. This is likely due to being more prone to getting stuck in local optima. Similar behavior was empirically observed during our initial experiments.

To conclude with a summary, in our reported experiments we use the L_1 loss for the following tasks: coarse image reconstruction, watermark reconstruction, opacity estimation and refined image reconstruction. We use either L_1 or $wBCE$ for watermark detection, depending on the architecture.

4.2.2.3 Adversarial Loss

As described in Section 4.1.3, our architecture uses an adversarial discriminator, which is trained jointly with the network. This results in two "competing" loss functions, each optimizing a different part of the model. The standard GAN loss is built as a modification of the BCE loss, which stems from the fact that the discriminator performs a two-class classification. The formulas presented below assume the discriminator to have a sigmoid activation function at its output.

Let X be the original unwatermarked image, $\hat{X}$ the watermarked image, X_{refine} the output of the final refining stage of the proposed model for watermark removal and $\mathcal{D}$ the discriminator. The discriminator is taught to optimize the following expression, equivalent to L_{BCE} objective for classifying between real images and the model's outputs:

$$L_{disc} = -\log D(X) - \log\left(1 - \mathcal{D}(X_{refine})\right).$$

The rest of the model then optimizes the opposite objective given by

$$L_{adv} = \log\left(1 - \mathcal{D}(X_{refine})\right).$$

Note that the first term is omitted, as the watermark removing model has no direct influence over the terms present in it.

4.2.2.4 Overall Objective

With the loss functions presented in the previous sections, we can build the overall loss function $\mathcal{L}$ the model ultimately optimizes. The objective for the discriminator is given by L_{disc} above and will not be discussed further, as its objective is trained independently and separate from the other loss functions being discussed.

Let us denote the terms needed to compute the loss. From the dataset, we need the following:

- the original unwatermarked image X,
- the watermarked image $\hat{X}$,
- the watermark W,
- the true watermark binary mask M and opacity mask M'.

After feeding the watermarked image $\hat{X}$ through the first stage of the model, we obtain the following predictions (see Section 4.1.1):

- the coarse reconstruction X_{coarse},
- the estimation of the watermark W_{coarse},
- the estimation of the watermark's binary mask M_{det} and its opacity M'_{coarse},
- thresholded M_{det} into a bool array M_{bool}

and then finally by refining the coarse image we obtain X_{refine} (see Section 4.1.2).

The composition of the final loss function is given by adding the respective loss functions, and weighing them through λ-denoted hyperparameters. The overall objective function for the *Watermark Remover* and *Watermark Refiner* parts of the model are given by:

$$
\begin{aligned}
\mathcal{L} =& \lambda_{L1}\Big(L_1\left(X, X_{coarse}\right) + L_1\left(X, X_{refine}\right) + L_1\left(W, W_{coarse}\right) + L_1\left(M', M'_{coarse}\right)\Big) + \\
&+ \lambda_{L1m}\Big(L_1(X \odot M_{bool}, X_{coarse} \odot M_{bool}) + L_1(X \odot M_{bool}, X_{refine} \odot M_{bool})\Big) + \\
&+ \lambda_{det} L_{det} + \lambda_{adv} L_{adv}.
\end{aligned}
$$

The L_{adv} term is defined above in Section 4.2.2.3 and relates to the adversarial component of the training process. The L_{det} term can be either $L_1(M, M_{det})$ for convolutional detection or $L_{wBCE}(M, M_{det}, \lambda_{pos})$ for *Segformer* detection (see Section 4.1.1.3).

The terms weighed by λ_{L1m} correspond to an increased reconstruction effort placed on the image areas detected as watermarked. An important aspect of these terms is that the thresholded M_{bool} matrix is detached from the computational graph and thus does not propagate gradients for optimization. This is done to prevent biasing the watermark detection area towards lower reconstruction error and thus hindering the performance of the detection itself.

4.2.3 Composition of the Final Output

While the final directly optimized output of the model is the X_{refine} image mentioned above, we perform an additional post-processing step to improve the image quality. Similar to how the coarse estimation is composited with the original image via the estimated watermark mask (see Figure 4.2), we perform the same operation to combine the original watermarked image and the refined output. This ensures there is no degradation of quality to the parts of the image which were not detected as watermarked.

Using this operation, we can also perform naïve high-resolution inference for watermark removal. Although the model is limited to a resolution of 256×256 for the scope of this work, we can resize the refined image to match the original image and compose them together. While this leads to image quality degradation and blurring in the originally watermarked areas, it might serve as a suitable compromise, depending on the use case. For visualized results on real-world data obtained via this process, see Appendix B.

4.2.4 Hyperparameter Selection and Training

Choosing the right optimizer is a crucial decision, as it determines how the model's parameters are adjusted for minimizing the loss function. We picked the Adam optimizer [69], which has attained over 136 000 citations at the time of writing, making it one of the most widely-used optimizers for similar applications in deep learning. Its ability to adapt the learning rate for each parameter makes it a popular choice. In comparison to other widely used optimizers, Adam often achieves faster convergence and better performance [70]. It was used for optimizing both the proposed model and its adversarial discriminator.

The values of the hyperparameters used during training the model were selected through a combination of domain knowledge, prior experience, and trial-and-error method. The selection was primarily based on empirical evaluation with a focus on balancing the scale of the loss functions' values to match their relative importance.

While a more thorough search for the optimal hyperparameter values through experimentation or other methods would be preferred, it is not feasible due to constraints in the scope of this project. Training a single model from scratch takes more than a week, making a large-scale investigation infeasible.

The following table presents a summary of hyperparameters and their values, as they were used within the scope of the conducted experiments:

Table 4.1 Hyperparameter values and their meanings

Meaning	Value	Notation
Shared loss weight for L_1 reconstructions	1	λ_{L1}
Additional weight for L_1 loss in watermarked areas	10	λ_{L1m}
Weight for the adversarial loss	$5 \cdot 10^{-4}$	λ_{adv}
Detection loss weight (*Segformer*)	10	λ_{det}
Detection loss weight (convolutions)	1	λ_{det}
Weight for the positive class in BCE calculation	5	λ_{pos}
Max number of epochs for training	100 epochs	–
Batch size	8	–
Image size	256×256	–
Optimizer	Adam	–
Initial learning rate for the model	10^{-3}	–
Initial learning rate for the discriminator	10^{-3}	–
Learning rate schedule	65 epochs	–
Learning rate decay	10^{-1}	–

The learning rate is set to decrease by multiplying it with the learning rate decay value. This process happens with a frequency given by the scheduling parameter. In our setup, it leads to the last third of training having the learning rate decreased by a factor of 10, which aims to reflect the expected convergence towards the end of the training.

4.3 Implementation

This section presents a brief overview of various tools employed in the model's implementation. It covers the technologies utilized, such as hardware, programming languages, libraries, as well as external sources, including pre-existing tools that were utilized in the process.

4.3.1 Technologies

The model is built using the standard Python data science stack, including libraries and tools such as `NumPy` [71], `PIL` [72] or `OpenCV` [73] for image data manipulation and preparation, and `scikit-learn` [74] for tasks such as dataset preprocessing and evaluation. `PyTorch` and `torchvision` [61] were used as the base framework for deep learning, providing us the essentials for defining and training all of the proposed models.

4.3.2 External Sources

For building the model, parts of publicly available architectures and their implementations were used. Reusing proven and well-performing components speeds up the development while reducing the risk of introducing new bugs.

The project's code is structured based on the public implementation[1] of *SLBR* [37]. The codebase was refactored and adapted for our purposes and no components from their original architecture were utilized. It served as a useful starting point. The authors of *SLBR* appear to have adapted the codebase from another publication [14], but its previous origins are unknown to us.

Our architecture draws upon several existing works, summarized in the list below.

- The used variants of the *PVTv2* [40] encoder were obtained via the official implementation[2] and accompanying pretrained weights. We customized the code to build the *PVTv2* decoder by reusing and modifying the encoder submodules.
- The *Segformer* detection uses the `SegformerHead` class from the `mmseg` library [75].
- The larger discriminator variant is adapted from [66] and was implemented by modifying a public implementation[3].
- The refining network is a *U-Net*-based architecture [28], using a public implementation[4].
- The official *MAT* [38] implementation[5] provided us with a wrapper to simplify working with 2D convolutional layers.

In conclusion, the use of various external components was a crucial aspect of the implementation process. The public repositories and libraries disclosed above were essential for achieving the desired results.

[1] https://github.com/bcmi/SLBR-Visible-Watermark-Removal
[2] https://github.com/whai362/PVT
[3] https://github.com/ozanciga/gans-with-pytorch#srgan
[4] https://github.com/milesial/Pytorch-UNet
[5] https://github.com/fenglinglwb/MAT

4.3.3 Hardware

For training such relatively large models on large-scale datasets with hundreds of thousands of samples, a powerful hardware system is required. The majority of training and evaluation described within this thesis was performed using the *NVIDIA DGX Station A100* hardware system. This system is equipped with a high-performance CPU, 512 GB of system memory, a high-speed RAID storage and four *NVIDIA A100* GPUs which provide a total of 160 GB of GPU memory. Combination of these components achieves substantial performance for deep learning applications.

Additionally, some of the models were trained on a different slightly less powerful machine with *NVIDIA V100* GPU. The training took slightly longer, but helped us to carry out the experiments in a timely manner.

The training setup was identical on both of the mentioned machines. The university's provision of access to such powerful hardware was an invaluable learning experience and played a key role in the successful conduct of this research.

Chapter 5

Methodology

This chapter describes the methodology and setup of the conducted experiments. We summarize the types of metrics used, the trained model variants and their comparison scenarios.

5.1 Metrics

The proposed model is evaluated using a combination of metrics to assess its performance in both the watermark detection and removal tasks. First, let us introduce the metrics for measuring the quality of overall image reconstruction.

5.1.1 Image Reconstruction

For evaluating the similarity between the watermarked and watermark-free images, these metrics will be used: root mean squared error (RMSE), peak signal-to-noise ratio (PSNR), and structural similarity index (SSIM):

- **Root Mean Squared Error** is a commonly used metric to measure the error in regression tasks with continuous outputs. In the context of our model, a low RMSE between the model's output and the ground truth image indicates higher quality of the reconstruction. It is calculated as the root of the mean squared error per pixel between the watermarked and watermark-free images:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2,$$

$$RMSE = \sqrt{MSE},$$

  where N is the number of pixels in the image, x_i is the value of the ith pixel in the watermarked image, and y_i is the value of the ith pixel in the watermark-free image.

- **Peak signal-to-noise ratio** measures the similarity between the watermarked and watermark-free images in terms of their pixel values. It is defined as follows:

$$PSNR = 10\log_{10}\frac{MAX_I^2}{MSE},$$

  where MAX_I is the maximum possible pixel value (e.g. 255 for 8-bit images). A higher $PSNR$ value indicates a lower reconstruction error. The metric is commonly used to evaluate the performance of image compression, restoration, denoising or similar methods.

- **Structural similarity index** measures the similarity between the watermarked and watermark-free images in terms of their luminance, contrast, and structure. The function is defined as follows:

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

  where x and y are the watermarked and watermark-free images, respectively, μ_x and μ_y are the means of x and y, σ_x and σ_y are the standard deviations of x and y, σ_{xy} is the covariance of x and y, and C_1 and C_2 are constants for stabilizing in the case of a low denominator.

5.1.2 Watermark Detection

For watermark detection, the model's ability to accurately detect the presence and location of watermarks in images will be evaluated using the following metrics:

- **Mean intersection-over-union (IoU)** measures the overlap between the predicted watermark mask and the ground truth mask. It is defined as the ratio of true positive predicted pixels to the total number of pixels:

$$IoU = \frac{TP}{TP + FP + FN},$$

  where TP, TN, FP and FN are the numbers of true positive, true negative, false positive and false negative pixels in the predicted masks, respectively. This metric ranges from 0 to 1, with a higher value indicating a larger overlap and thus a higher performance.

- **Accuracy** measures the overall accuracy of the predicted watermark mask. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

  where N is the number of pixels in the image, y_i is the ground truth value of the ith pixel, and $\hat{y}_i$ is the predicted value of the ith pixel. A higher $Accuracy$ value indicates a more accurate prediction.

- **F1 score** accounts for class imbalance in a classification problem. It is the harmonic mean of two related metrics, with higher scores indicating better performance.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall},$$

  where $Precision$ is the fraction of true positives among the predicted watermark pixels, and $Recall$ is the fraction of true positives among all actually watermarked pixels:

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}.$$

- **Recall** is a particularly interesting metric for our use case, as it measures the model's ability to identify all watermarked pixels, regardless of any over-extension into non-watermarked areas. In our scenario, a higher recall, even at the cost of slightly lower values of other metrics, might be preferable for following watermark removal attempts.

5.2 Experimental Setup

As we proceed, several variants of the proposed model will be evaluated using the metrics presented above to assess and compare their performance in the watermark removal and detection tasks.

Experimenting with various model architectures on datasets of varying sizes and complexities can help us gain a deeper understanding of the strengths and limitations of each model variant and the factors influencing their performance. By identifying which models perform best on specific tasks, we can gain insights into the underlying causes of their successes and failures.

5.2.1 Model Variants

The proposed architecture allows for several degrees of freedom in their specific design and implementation. For the scope of this work, as described in Section 4.1, we experiment with altering the following aspects of the architecture:

- the size of the *PVTv2* encoder (`b0` or `b1` variant with 3.7M or 14M parameters, respectively),
- watermark detection module (*Segformer* or convolutional decoder, see Figure 4.1),
- the size of the discriminator (simple or *SRGAN*, see Section 4.1.3).

We decided to use one concrete variant of the model as the base scenario and derive three other variants by altering one of the aspects of the model. The base scenario has the `b1` encoder, watermark detection through a convolutional decoder and a simple discriminator. Each of the three derived models is created by changing exactly one of these aspects to its alternative. The models are summarized in the table below.

Table 5.1 Summary of the model variants used in experiments.

Model Name	Encoder	Watermark Detection	Discriminator
TAWR Base	*PVTv2* `b1`	Convolutional	Simple
TAWR Lite	*PVTv2* `b0`	Convolutional	Simple
TAWR Segformer	*PVTv2* `b1`	*Segformer*	Simple
TAWR Adversary	*PVTv2* `b1`	Convolutional	*SRGAN* + SN

While other architectural aspects that could be explored exist, the combination of the presented model variants provides a promising starting point for our exploratory experiments. However, it is worth highlighting that the training process of these models is resource and time intensive, which limits the number of combinations that can be examined within the restricted scope of this work. Nevertheless, the outcomes obtained from the models described above are expected to provide valuable information in terms of insights into the effectiveness of the proposed architecture and its respective components.

5.2.2 Benchmarking Datasets

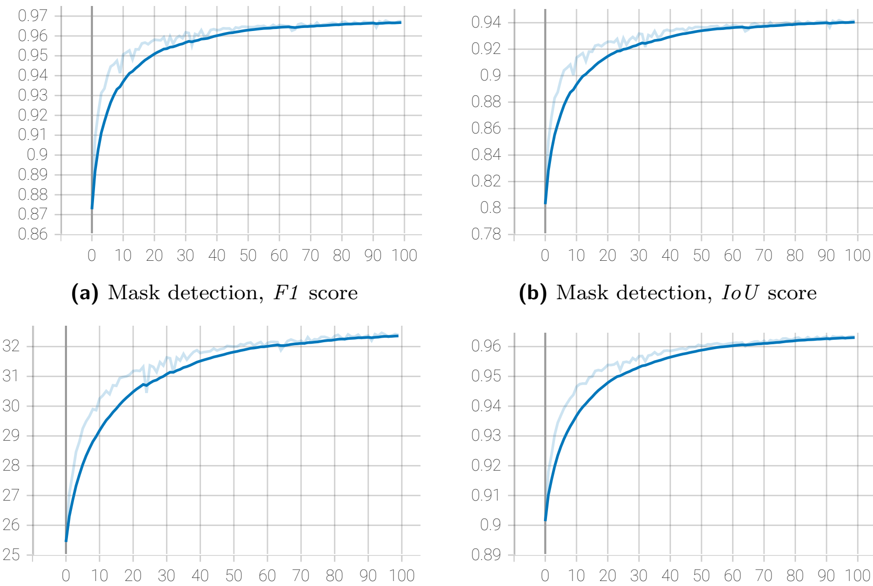
The performance measures are directly evaluated on three datasets. One of them is the publicly available *CLWD* dataset [16], commonly used for benchmarking watermark removal methods. The other two are based on the car dataset provided for the purposes of this thesis (see Chapter 3). The proposed model variants are directly compared against current state-of-the-art model *SLBR* [37] on all the datasets and against some other methods presented in Section 2.1.2 on the publicly available *CLWD* dataset. Regarding the image size and information available for each sample, the datasets share a common nearly identical format. The main properties of the datasets are summarized in the following table.

Table 5.2 Summary of the datasets used for evaluation.

Dataset name	Description	# training samples	# val. samples
CLWD	diverse photos + logos	60 000	10 000
Cars Text	cars + texts	313 032	34 764
Cars Various	cars + texts, logos, ads	312 976	34 776

5.2.3 Training the Models

Each of the proposed model variants is trained on the three described datasets and additionally the *SLBR* model is trained on the car datasets. The *PVTv2* encoder module is initialized with pretrained weights (see Section 4.3.2). Training takes ˜11 days on a single *NVIDIA A100* GPU for the car datasets and a little over 2 days for the *CLWD* dataset. To remain consistent, we save the weights with the highest *PSNR* score after 100 epochs, following the *SLBR* training logic.



(a) Mask detection, *F1* score

(b) Mask detection, *IoU* score

(c) Image reconstruction, *PSNR* score

(d) Image reconstruction, *SSIM* score

Figure 5.1 Validation metrics progress over 100 training epochs, *TAWR Base* on the *Cars Various* dataset.

5.2.4 Evaluation on Real-World Images

Unfortunately, it is difficult to quantitatively assess the performance of the models on the supplied real-world data in a straightforward manner, as there are no ground truth labels. Nevertheless, we have included visualized comparison of multiple models in Figure 6.2.

Some additional visualizations obtained via the *TAWR Segformer* model variant are provided in Appendix B. This variant was chosen for visualizations based due its high *Recall* score (see Chapter 6) on synthetic data, which we observed to correlate with improved detection capabilities in real-world data.

It is worth noting that the sample images we used were not chosen based on their performance. Rather, they were the first images in our demo set that had watermarks that were deemed interesting. We excluded images that lacked watermarks or had unremarkable ones.

Chapter 6
Results and Analysis

In this final chapter, we present and analyze the results of the conducted experiments. In the discussion section, we highlight the limitations of the model and suggest areas for potential improvement through future work.

6.1 Results

The quantitative results are presented first section, followed by output visualizations to provide an insight into the human perception of the watermark removal results.

6.1.1 Performance Metrics

In this section, we present the results and comparison for each of the evaluated methods. The process for obtaining the results and their explanation is thoroughly explained in Chapter 5.

	Watermark detection				**Reconstruction**		
	F1	*Recall*	*Accuracy*	*IoU*	*PSNR*	*SSIM*	*RMSE*
SLBR [37]	**0.9451**	0.9453	**0.9977**	**0.9014**	**38.74**	**0.9821**	**3.03**
TAWR Base	0.8989	0.8958	0.9958	0.8266	37.76	0.9802	3.415
TAWR Lite	0.5049	0.4176	0.9829	0.3849	32.06	0.964	7.319
TAWR Segformer	0.7990	**0.9753**	0.9889	0.6768	37.93	0.9812	3.355
TAWR Adversary	0.9025	0.9028	0.9959	0.8317	37.74	0.9802	3.423

Table 6.1 Evaluation metrics on the validation *Cars Text* dataset.

	Watermark detection				**Reconstruction**		
	F1	*Recall*	*Accuracy*	*IoU*	*PSNR*	*SSIM*	*RMSE*
SLBR [37]	**0.9852**	0.9847	**0.9962**	**0.9734**	**35.46**	**0.9801**	**4.954**
TAWR Base	0.9677	0.9635	0.9914	0.9419	32.46	0.9632	6.75
TAWR Lite	0.4557	0.3538	0.8696	0.3372	22.39	0.918	22.61
TAWR Segformer	0.9467	**0.9899**	0.9841	0.9044	32.15	0.9616	7.012
TAWR Adversary	0.9574	0.9576	0.9887	0.9291	31.19	0.9543	7.809

Table 6.2 Evaluation metrics on the validation *Cars Various* dataset.

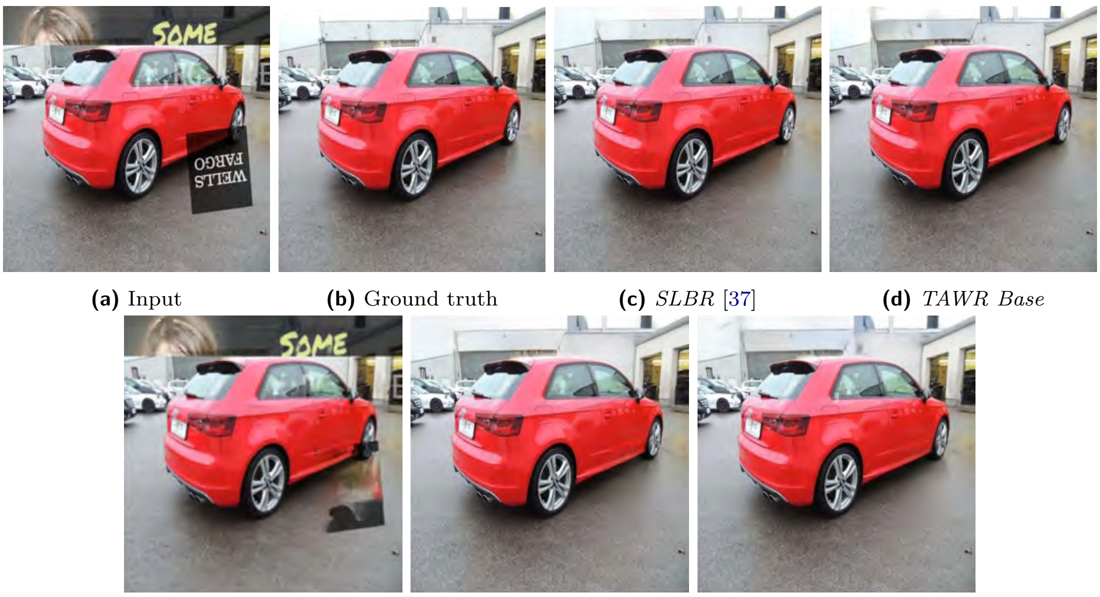
The comparison in Table 6.3 also includes results of some other models presented in Section 2.1.2. These values were taken directly from the latest surveyed work [37], as it provides a thorough comparison for the reconstruction metrics.

	Watermark detection				Reconstruction		
	F1	*Recall*	*Accuracy*	*IoU*	*PSNR*	*SSIM*	*RMSE*
SLBR [37]	**0.834**	0.829	**0.9883**	**0.7589**	**38.19**	**0.9805**	**3.807**
Split then Refine [14]	–	–	–	–	37.41	0.9787	4.23
BVMR [29]	–	–	–	–	35.89	0.9734	5.02
WDNet [16]	–	–	–	–	35.53	0.9738	5.11
TAWR Base	0.7715	0.7641	0.9842	0.6817	36.6	0.9756	4.56
TAWR Lite	0.6961	0.6899	0.9781	0.597	34.95	0.9695	5.596
TAWR Segformer	0.7363	**0.8724**	0.9748	0.6214	36.65	0.9761	4.616
TAWR Adversary	0.763	0.7546	0.9834	0.6719	36.2	0.9743	4.786

Table 6.3 Evaluation metrics on the validation *CLWD* dataset.

Please note that our measurements of *SLBR* [37] performance on the *CLWD* dataset [16] show slight differences from the values reported in the original paper. We used the original authors' code to obtain these values, but their published pretrained weights likely differ from the ones used for the original evaluation. Nevertheless, there are only slight differences without an effect on the relative ordering of the methods.

6.1.2 Visualized Outputs

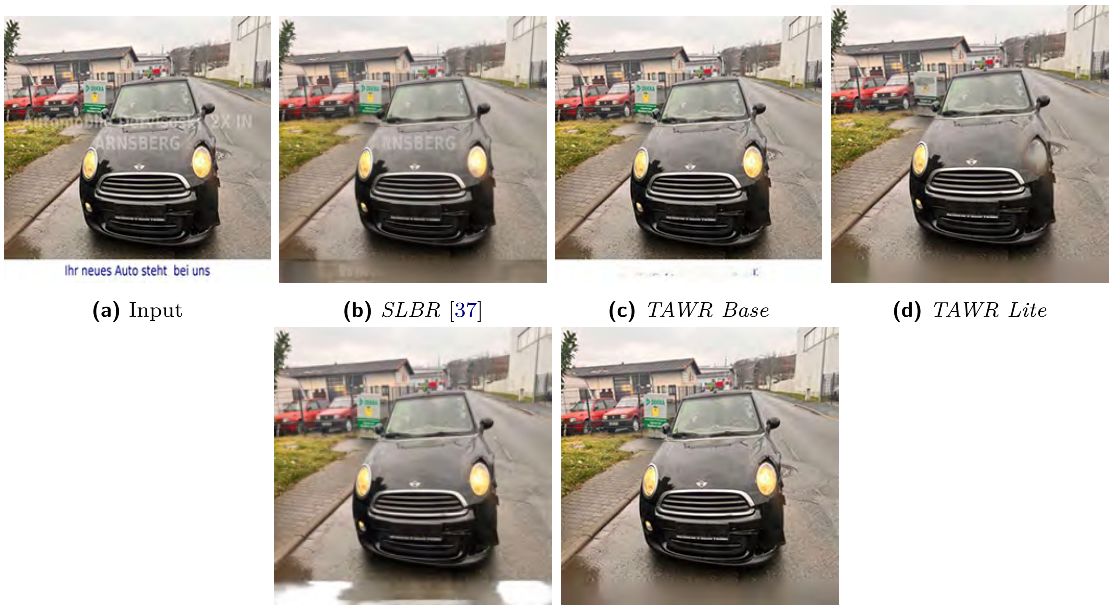


(a) Input **(b)** Ground truth **(c)** *SLBR* [37] **(d)** *TAWR Base*

(e) *TAWR Lite* **(f)** *TAWR Segformer* **(g)** *TAWR Adversary*

Figure 6.1 Removal demonstration on a sample from the *Cars Various* dataset.

Figures in this section provide a comparison of the performance of watermark removal techniques on both synthetic and real-world images. The two cases show different scenarios, with the first Figure 6.1 demonstrating the effectiveness of various algorithms on a synthetic image created



(a) Input **(b)** *SLBR* [37] **(c)** *TAWR Base* **(d)** *TAWR Lite*

(e) *TAWR Segformer* **(f)** *TAWR Adversary*

Figure 6.2 Removal demonstration on a sample from the real-world dataset.

by the same process as the training data, while the second Figure 6.2 provides a comparison on a sample from the real-world watermarked images in the original dataset.

Please note that the samples were not selected based on the results, but rather hand-picked for their perceived complexity and only evaluated afterwards. Nevertheless, it is important to stress that the limited number of samples used in this section makes it difficult to draw definitive conclusions about expected performance based solely on the visualizations presented, given the diversity of possible watermark damage. For a more detailed display of the outputs from various models, some additional visualizations are provided in Appendix A and Appendix B.

6.2 Discussion

Our proposed model achieved satisfactory results for the considered metrics when tested on synthesized watermarks and was effective in removing various watermarks from diverse images. However, we acknowledge that our model's performance falls short of the current state-of-the-art method, *SLBR* [37], in most cases. The only exception is the *Recall* detection metric, where our *TAWR Segformer* model consistently outperforms *SLBR*. While this is a promising result for certain scenarios and improves reconstruction metrics compared to other *TAWR* variants, it is not sufficient to match the performance of the state-of-the-art method reconstruction-error wise.

6.2.1 Transformer Size

We observed the *TAWR Lite* perform notably worse compard to other variants, which is a consequence of using a smaller transformer in the model's first stage. This suggests that further increasing the capacity of the transformer for initial feature extraction could potentially yield better results. It is to be noted that we only experimented with the two smallest architectures (*PVTv2* `b0` or `b1`) presented in the original work [40], therefore exploring even larger architectures could be beneficial, while leading to significantly higher computational requirements.

6.2.2 Real-World Image Performance Disparity

The output visualizations provided further support for our quantitative results, demonstrating that our proposed model could successfully remove various types of watermarks while preserving the image's overall visual quality.

We can observe the surprisingly poor performance of the *SLBR* model on the real-world data, although the quantitative results on the synthesized data would suggest otherwise. This is not an isolated example and we observed similar behavior while empirically evaluating the visual results, where our method seems significantly better at generalizing from synthetic to real data. From what we observed, the *SLBR* method was able to generalize to real-world data for the first few epochs and then its performance started to degrade. The behavior is very reminiscent of overfitting on the training data, except we observed no decrease on the synthetic validation set performance.

If this observation really is a specific kind of overfitting, it might be worth adjusting the synthesizing process. Namely, instead of having a static training dataset, we could utilize dynamic synthesizing, i.e. generate the watermarked samples randomly at training time. This could possibly help to alleviate this issue, as the model would never see the same exact sample twice.

As mentioned before, unfortunately, we have no way of measuring the reconstruction error on real-world dataset, as the ground truth label is not provided. We deem this as phenomenon worthy of further research.

6.2.3 Thresholded Mask

Some artifacts were observed in the reconstructed images in both the synthesized and real-world scenario, which could be addressed by improving the image composition process. Specifically, our current approach utilizes a thresholded method (see Section 4.2.2.4) for image composition, where the model internally generates an intermediate image using a binary mask between the first and second stages of the architecture (see Figure 4.2).

We selected the thresholded binary mask composition based on our specific requirements for the output image. For instance, we needed to identify which precise pixels come from lower resolution representation for upscaling purposes. Instead of using a binary mask, we could have used a continuous mask that would produce the intermediate image as a weighted sum of the coarse and input image. We believe that utilizing this approach could improve the model's capabilities in ambiguous areas around the watermark, which are prone to producing artifacts.

6.2.4 Adversarial Training Impact

Although a larger discriminator is more expensive to train, its cost for inference remains unchanged since the discriminator is discarded after training. However, we cannot conclude that using a significantly larger discriminator in the *TAWR Adversarial* variant resulted in a significant improvement to the model's output. While the metrics may not show an improvement compared to other variants, our visualization results (see Appendix A) indicate that the larger discriminator had a positive impact on the model's ability to reconstruct the original watermark from the image, which we interpret as a better understanding of the watermarking process.

Additionally, we suppose further experimentation with tuning the adversarial component of the entire model might lead to better results, especially given that its validation performance continued to improve even at the 100 epoch training limit, implying that the model may require additional training.

6.3 Limitations and Possible Improvements

Apart from not reaching state-of-the-art results, the proposed model has some limitations and shortcomings, including the low resolution of images at training time and a high cost of training and inference. Training the model on higher resolution images at various scales is likely to lead to improved quality and generalization, but unfortunately also exceedingly increases the time required for the training.

Due to computational and time constraints, the model has also not been fine-tuned to achieve the best possible results. Given the high performance demands, thoroughly searching the hyper-parameter space is infeasible.

Additionally, a larger transformer or an architecture modification may be crucial for improved feature extraction in the first stage, as mentioned in the previous section. Unfortunately, this is likely to come at an increased cost.

One possible inexpensive improvement is the continuous version of the composition mask mentioned earlier. Furthermore, taking note of the *SLBR* [37] mask refinement process for inspiration, it may be beneficial to incorporate an additional mechanism to enhance the accuracy of the detection mask. This could be accomplished by leveraging the adversarial discriminator already present in the architecture. Although the discriminator currently only judges the reconstructed image in relation to the true mask, it may be sensible to let it also assess the authenticity of the estimated mask with respect to the watermarked image.

Furthermore, if we could obtain original ground truth images for the real-world data, we could likely train a model that would perform much better in the real scenario than one trained on synthetic data.

6.3.1 Future Work

Considering the limitations and potential improvements discussed above, we are currently working on incorporating some of the proposed changes to the model. We aim to share our findings at a conference to contribute to the research in this field.

Conclusion

In this thesis, we aimed to develop a method using deep neural networks to effectively detect and remove watermarks without compromising image quality. To accomplish the set out objectives, we performed the following steps:

- We conducted a comprehensive survey of existing watermark removal techniques and vision transformer models. This helped us to understand the state-of-the-art approaches in this field and allowed us to make decisions for designing the architecture of the proposed method.
- We analyzed the supplied image data and designed a process for creating datasets in a format suitable for our training. The steps to achieve this included manual labeling of data, training several classifiers and synthesizing various watermark types.
- We proposed a deep learning architecture for removing watermarks from images. Our method uses the insights from previously published works on the topic, but incorporates novel architectural modifications with the aim of improving the achieved results.
- We trained several experimental variants of the proposed deep learning method, as well as the current state-of-the-art method [37] for direct comparison. Each model variant was trained on two newly synthesized datasets and one publicly available dataset.
- We evaluated the performance of our method in terms of watermark detection accuracy and image reconstruction quality and provided an analysis, interpretation and visualization of the achieved results.

In conclusion, we have achieved our goals of designing a novel deep learning method for removing watermarks from images, which we have quantitatively evaluated in the scenario where we have access to the precise original watermarking process. Although our method did not reach decisive state-of-the-art results, there is considerable scope for improvement.

Our implementation may be useful for future research, as well as for practical applications in areas such as digital forensics, copyright protection, and image editing. We encourage the open-source use of our work and have made the code for the *TAWR Segformer* variant available on GitHub[1] for others to freely access and utilize.

[1] `https://github.com/halamto2/TAWR`

Appendix A

Watermark Removal Visualization Samples

This appendix chapter aims to visually compare the performance of the evaluated watermark removal methods on two randomly selected samples from each of the three datasets introduced in Chapter 5.

The samples come from the validation set and are inferred at a resolution of 256×256. Each visualization contains the model's input, the ground truth unwatermarked image, the original image reconstruction, the true watermark location mask, the predicted watermark location mask and the estimated watermark image where applicable (as the *SLBR* [37] model does not estimate the appearence of the original watermark).

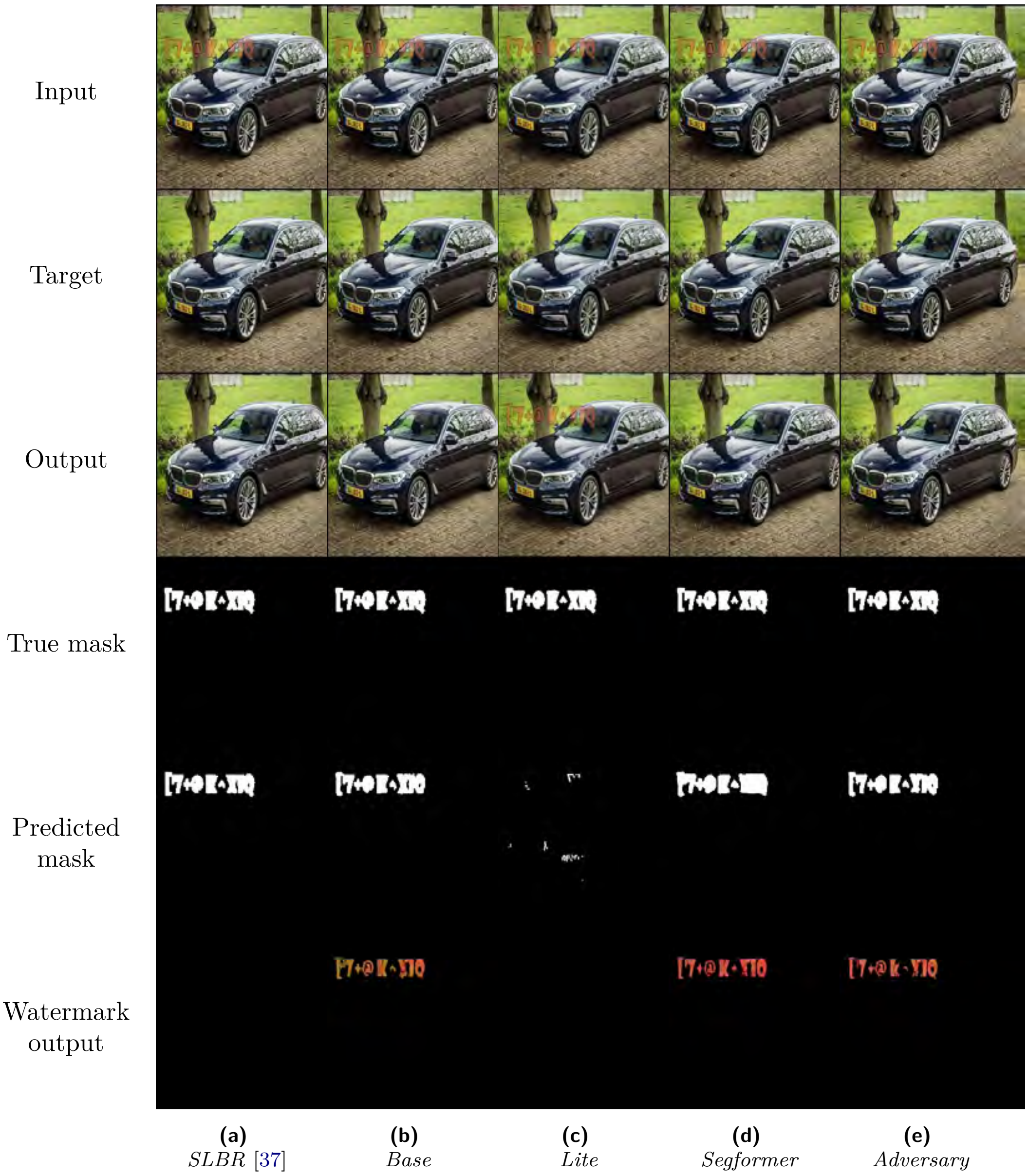


Figure A.1 Visualization of methods on a random sample from the *Cars Text* dataset.

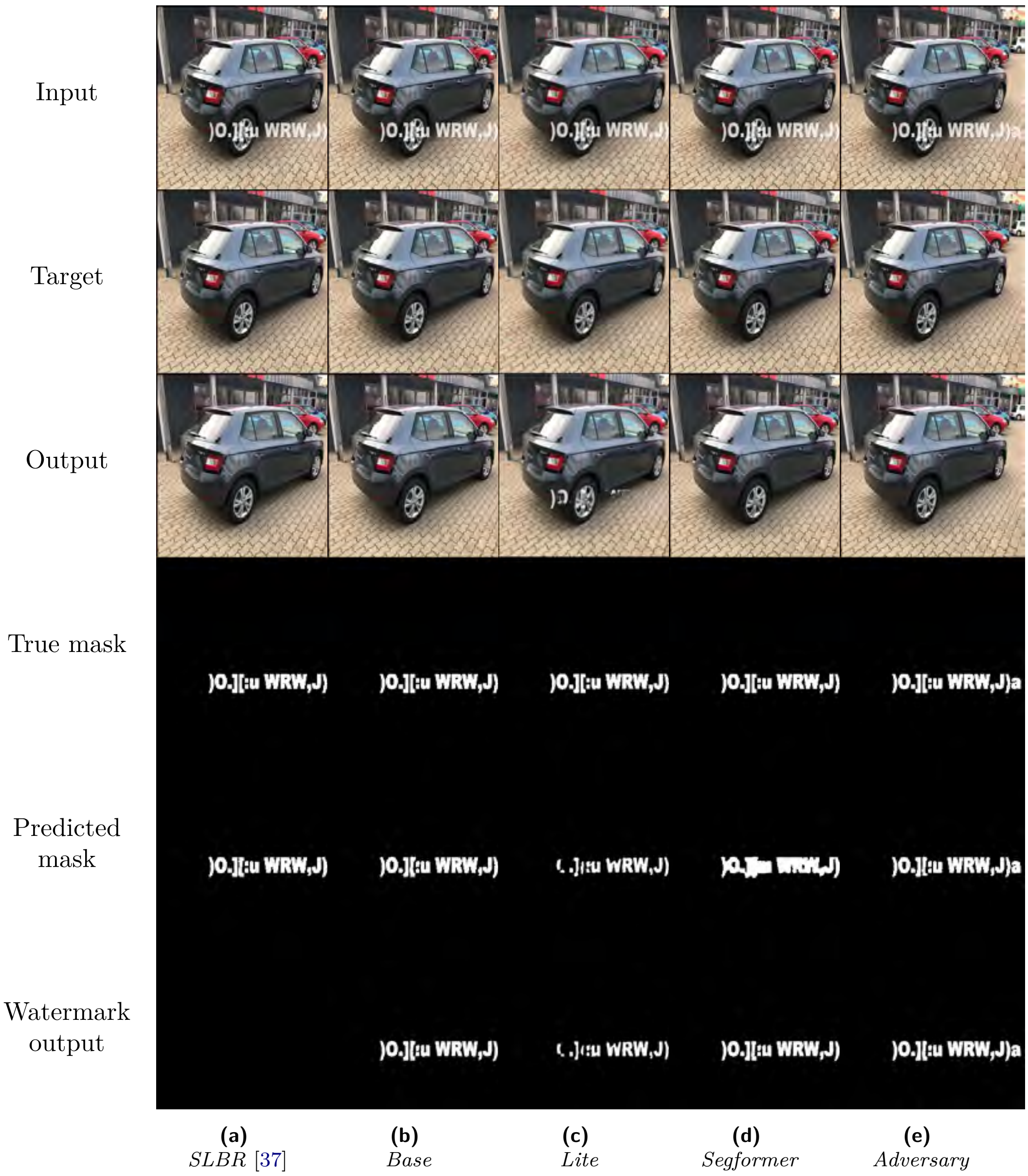
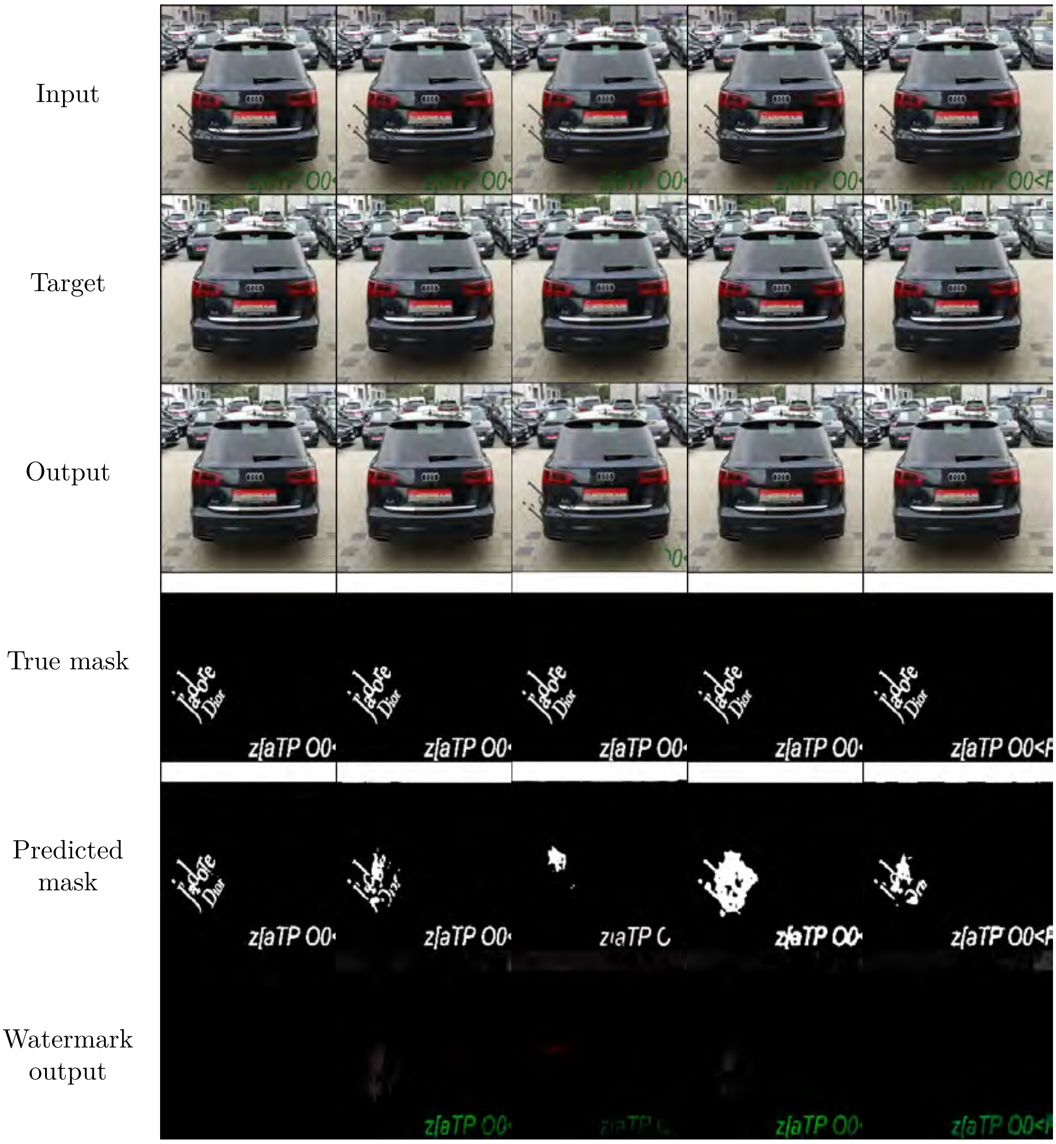
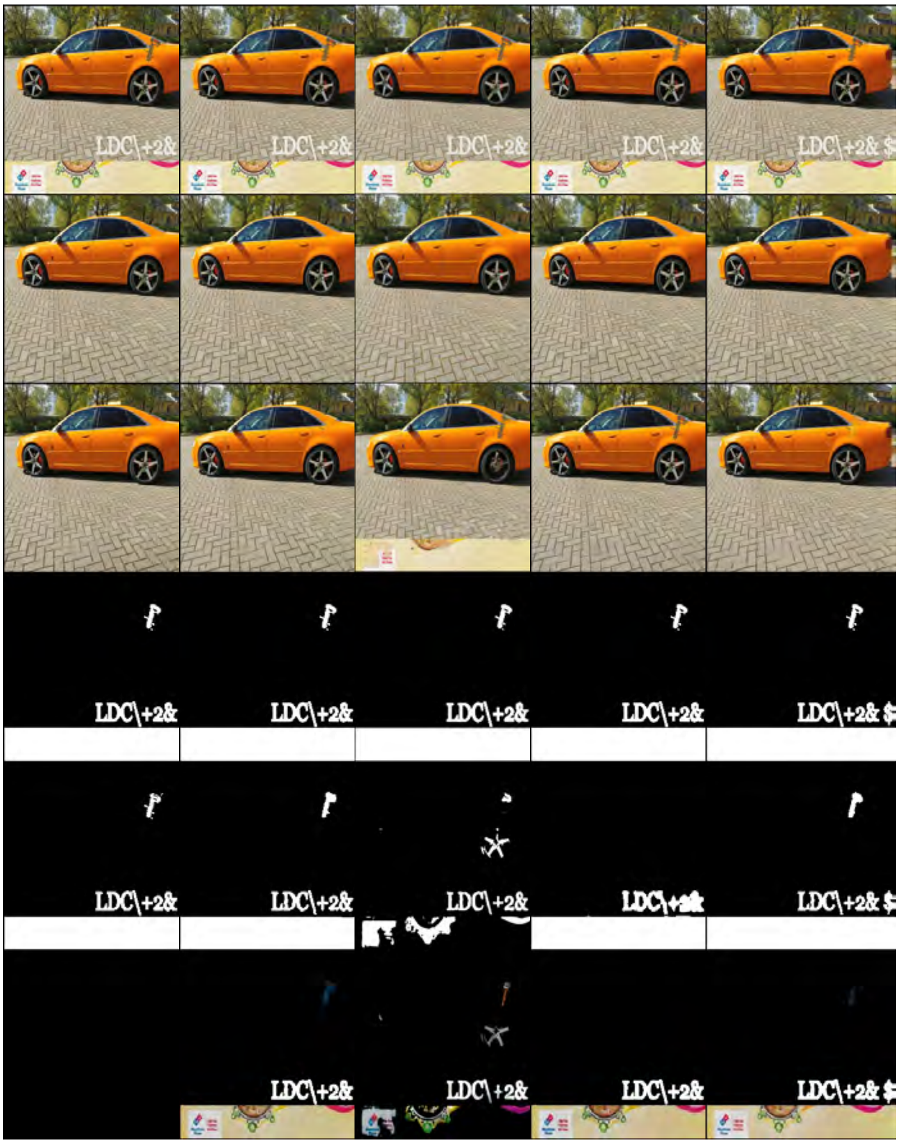


Figure A.2 Visualization of methods on a random sample from the *Cars Text* dataset.



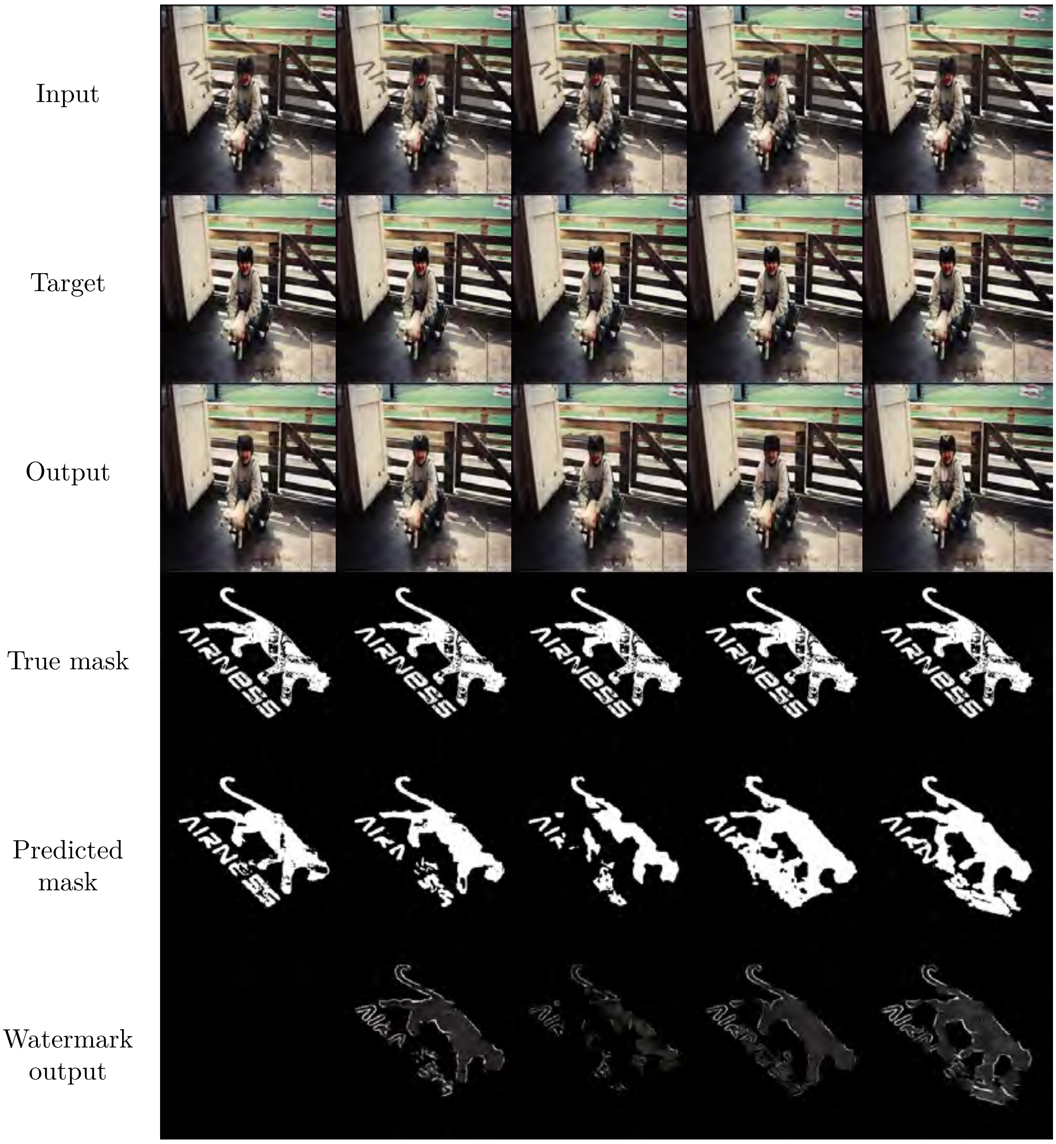
(a) *SLBR* [37] **(b)** *Base* **(c)** *Lite* **(d)** *Segformer* **(e)** *Adversary*

Figure A.3 Visualization of methods on a random sample from the *Cars Various* dataset.



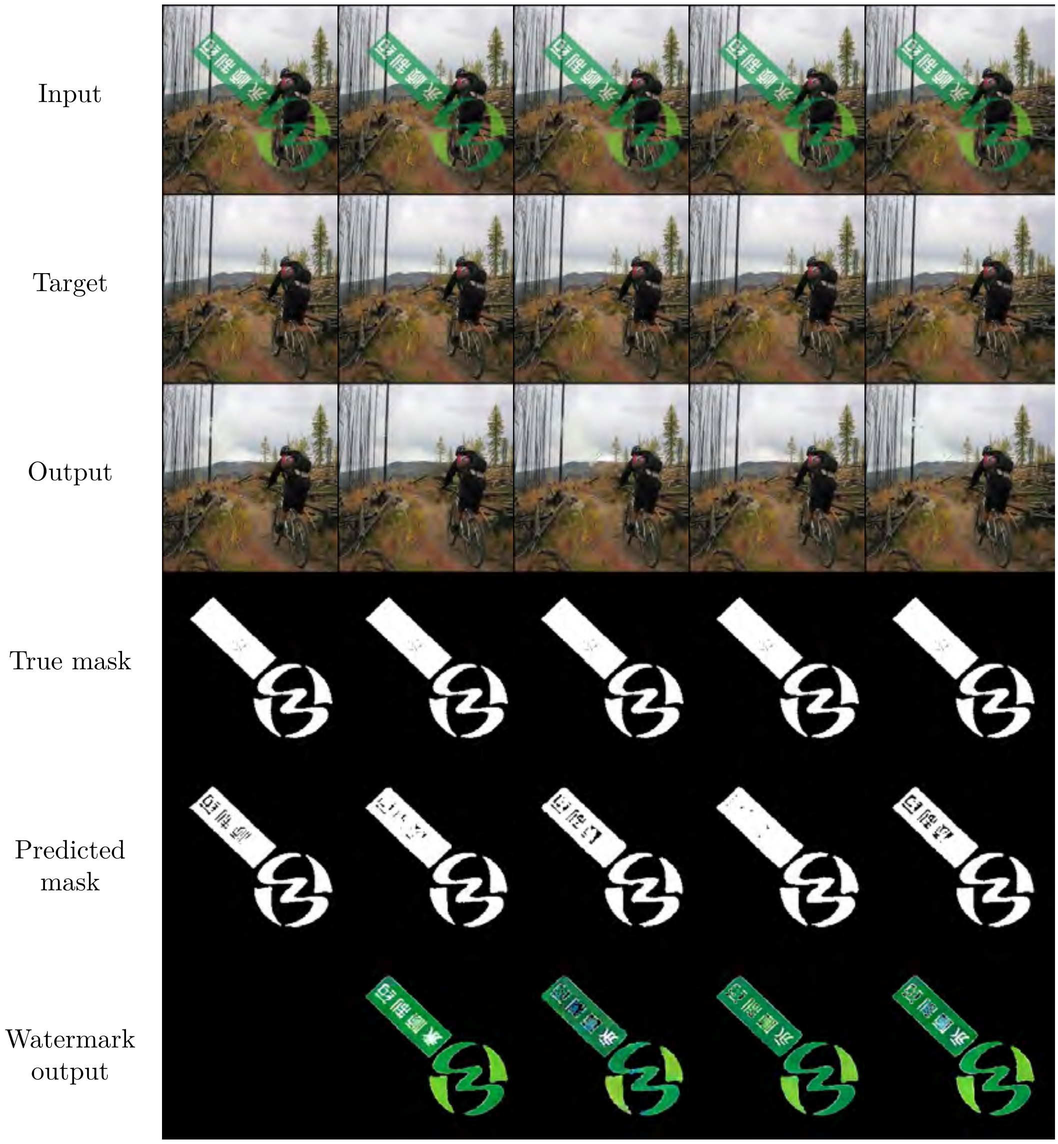
(a) *SLBR* [37] **(b)** *Base* **(c)** *Lite* **(d)** *Segformer* **(e)** *Adversary*

Figure A.4 Visualization of methods on a random sample from the *Cars Various* dataset.



(a) *SLBR* [37] **(b)** *Base* **(c)** *Lite* **(d)** *Segformer* **(e)** *Adversary*

Figure A.5 Visualization of methods on a random sample from the *CLWD* dataset.



(a) *SLBR* [37] **(b)** *Base* **(c)** *Lite* **(d)** *Segformer* **(e)** *Adversary*

Figure A.6 Visualization of methods on a random sample from the *CLWD* dataset.

Appendix B
Visualization for Naïve High-Resolution Inference

This appendix chapter provides visualized examples of naïve high-resolution inference for watermark removal introduced in Section 4.2.3. The motivation behind this upscaling operation is to mitigate the impact of the low internal resolution of the watermark removal model and partially alleviate its influence. The model used to produce the results for these samples is the *TAWR Segformer* variant trained on *Cars Various* dataset. The inference is performed on randomly selected real-world samples, rather than the synthesized validation set.

(a) Watermarked image **(b)** Upscaled composition after removal

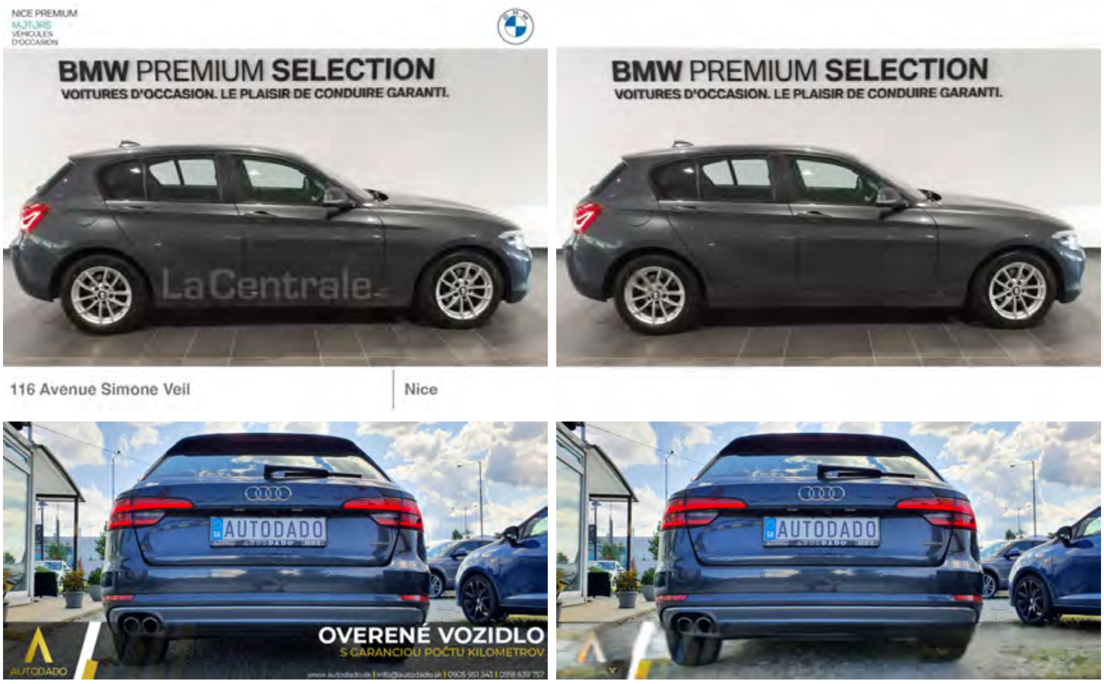


Figure B.1 Visualization for naïvely upscaled results on real images.

(a) Watermarked image **(b)** Upscaled composition after removal

Bibliography

1. OPENAI. *ChatGPT: A Large Language Model by OpenAI*. 2021. https://www.openai.com.
2. MURPHY, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012. ISBN 0262018020.
3. COX, Ingemar; MILLER, Matthew; BLOOM, Jeffrey; FRIDRICH, Jessica; KALKER, Ton. *Digital Watermarking and Steganography, 2nd Ed. (The Morgan Kaufmann Series in Multimedia Information and Systems)*. Hardcover. Morgan Kaufmann, 2007. ISBN 978-0123725851.
4. BIERMANN, Christopher J. *Handbook of Pulping and Papermaking*. Hardcover. Academic Press, 1996. ISBN 978-0120973620.
5. HARPER, Douglas. Etymology of watermark. In: *Online Etymology Dictionary* [online]. 2014 [visited on 2022-09-29]. Available from: https://www.etymonline.com/word/watermark.
6. TAI, Yuan-Yen; MANSOUR, Mohamed F. *Audio Watermarking over the Air With Modulated Self-Correlation*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1903.08238.
7. JAIN, Ritu; TRIVEDI, Munesh Chandra; TIWARI, Shailesh. Digital Audio Watermarking: A Survey. In: BHATIA, Sanjiv K.; MISHRA, Krishn K.; TIWARI, Shailesh; SINGH, Vivek Kumar (eds.). *Advances in Computer and Computational Sciences*. Singapore: Springer Singapore, 2018, pp. 433–443. ISBN 978-981-10-3773-3.
8. JAYAMALAR, T; RADHA, V. Survey on digital video watermarking techniques and attacks on watermarks. *International Journal of Engineering Science and Technology*. 2010, vol. 2, no. 12, pp. 6963–6967.
9. CHEN, Xinyun; WANG, Wenxiao; BENDER, Chris; DING, Yiming; JIA, Ruoxi; LI, Bo; SONG, Dawn. Refit: a unified watermark removal framework for deep learning systems with limited data. In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 2021, pp. 321–335.
10. LI, Yue; WANG, Hongxia; BARNI, Mauro. A survey of deep neural network watermarking techniques. *Neurocomputing*. 2021, vol. 461, pp. 171–193.
11. CHAWLA, Gaurav; SAINI, Ravi; YADAV, Rajkumar, et al. Classification of watermarking based upon various parameters. *International Journal of Computer Applications & Information Technology*. 2012, vol. 1, no. II.
12. TAO, Hai; CHONGMIN, Li; ZAIN, Jasni Mohamad; ABDALLA, Ahmed N. Robust image watermarking theories and techniques: A review. *Journal of applied research and technology*. 2014, vol. 12, no. 1, pp. 122–138.

13. NATHANAËL COTTIN. *Hide & Reveal*. 2013. Version 1.7.0. Available also from: http://hidereveal.ncottin.net/.
14. CUN, Xiaodong; PUN, Chi-Man. Split then refine: stacked attention-guided ResUNets for blind single image visible watermark removal. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, vol. 35, pp. 1184–1192. No. 2.
15. JIANG, Pei; HE, Shiwen; YU, Hufei; ZHANG, Yaoxue. Two-stage visible watermark removal architecture based on deep learning. *IET Image Processing*. 2020, vol. 14, no. 15, pp. 3819–3828.
16. LIU, Yang; ZHU, Zhen; BAI, Xiang. Wdnet: Watermark-decomposition network for visible watermark removal. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 3685–3693.
17. GOODFELLOW, Ian; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. Generative adversarial networks. *Communications of the ACM*. 2020, vol. 63, no. 11, pp. 139–144.
18. HALAMA, Tomáš. *Image Inpainting Using Generative Adversarial Networks*. 2020. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology.
19. WANG, Yi; CHEN, Ying-Cong; TAO, Xin; JIA, Jiaya. Vcnet: A robust approach to blind image inpainting. In: *European Conference on Computer Vision*. Springer, 2020, pp. 752–768.
20. LIU, Yang; PAN, Jinshan; SU, Zhixun. Deep blind image inpainting. In: *International Conference on Intelligent Science and Big Data Engineering*. Springer, 2019, pp. 128–141.
21. DEKEL, Tali; RUBINSTEIN, Michael; LIU, Ce; FREEMAN, William T. On the effectiveness of visible watermarks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2146–2154.
22. SANTOYO-GARCIA, Hector; FRAGOSO-NAVARRO, Eduardo; REYES-REYES, Rogelio; SANCHEZ-PEREZ, Gabriel; NAKANO-MIYATAKE, Mariko; PEREZ-MEANA, Hector. An automatic visible watermark detection method using total variation. In: *2017 5th International Workshop on Biometrics and Forensics (IWBF)*. IEEE, 2017, pp. 1–5.
23. PEI, Soo-Chang; ZENG, Yi-Chong. A novel image recovery algorithm for visible watermarked images. *IEEE Transactions on information forensics and security*. 2006, vol. 1, no. 4, pp. 543–550.
24. HUANG, Chun-Hsiang; WU, Ja-Ling. Attacking visible watermarking schemes. *IEEE transactions on multimedia*. 2004, vol. 6, no. 1, pp. 16–30.
25. WANG, Jieying; SILVA, Dennis J; KOSINKA, Jiří; TELEA, Alexandru; HASHIMOTO, Ronaldo F; ROERDINK, Jos BTM. Interactive image manipulation using morphological trees and spline-based skeletons. *Computers & Graphics*. 2022.
26. CHENG, Danni; LI, Xiang; LI, Wei-Hong; LU, Chan; LI, Fake; ZHAO, Hua; ZHENG, Wei-Shi. Large-scale visible watermark detection and removal with deep convolutional networks. In: *Chinese conference on pattern recognition and computer vision (prcv)*. Springer, 2018, pp. 27–40.
27. EVERINGHAM, Mark; ESLAMI, SM; VAN GOOL, Luc; WILLIAMS, Christopher KI; WINN, John; ZISSERMAN, Andrew. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*. 2015, vol. 111, no. 1, pp. 98–136.
28. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

29. HERTZ, Amir; FOGEL, Sharon; HANOCKA, Rana; GIRYES, Raja; COHEN-OR, Daniel. Blind visual motif removal from a single image. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 6858–6867.
30. LI, Xiang; LU, Chan; CHENG, Danni; LI, Wei-Hong; CAO, Mei; LIU, Bo; MA, Jiechao; ZHENG, Wei-Shi. Towards photo-realistic visible watermark removal with conditional generative adversarial networks. In: *International Conference on Image and Graphics*. Springer, 2019, pp. 345–356.
31. XIAO, Jin; GU, Shuhang; ZHANG, Lei. Multi-domain learning for accurate and few-shot color constancy. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3258–3267.
32. WANG, Xudong; CAI, Zhaowei; GAO, Dashan; VASCONCELOS, Nuno. Towards universal object detection by domain attention. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 7289–7298.
33. LIU, Shikun; JOHNS, Edward; DAVISON, Andrew J. End-to-end multi-task learning with attention. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 1871–1880.
34. LIN, Tsung-Yi; MAIRE, Michael; BELONGIE, Serge; HAYS, James; PERONA, Pietro; RAMANAN, Deva; DOLLÁR, Piotr; ZITNICK, C Lawrence. Microsoft coco: Common objects in context. In: *European conference on computer vision*. Springer, 2014, pp. 740–755.
35. WHITTY, Robin W. Some Comments on Multiple Discovery in Mathematics. *Journal of Humanistic Mathematics* [online]. 2017, vol. 7, no. 1, pp. 172–188 [visited on 2022-10-10]. Available from DOI: 10.5642/jhummath.201701.14.
36. WIKIPEDIA. *List of multiple discoveries — Wikipedia, The Free Encyclopedia* [http://en.wikipedia.org/w/index.php?title=List%20of%20multiple%20discoveries&oldid=1101649825]. 2022. [Online; accessed 10-October-2022].
37. LIANG, Jing; NIU, Li; GUO, Fengjun; LONG, Teng; ZHANG, Liqing. Visible Watermark Removal via Self-calibrated Localization and Background Refinement. In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 4426–4434.
38. LI, Wenbo; LIN, Zhe; ZHOU, Kun; QI, Lu; WANG, Yi; JIA, Jiaya. MAT: Mask-Aware Transformer for Large Hole Image Inpainting. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10758–10768.
39. XIE, Enze; WANG, Wenhai; YU, Zhiding; ANANDKUMAR, Anima; ALVAREZ, Jose M; LUO, Ping. SegFormer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*. 2021, vol. 34, pp. 12077–12090.
40. WANG, Wenhai; XIE, Enze; LI, Xiang; FAN, Deng-Ping; SONG, Kaitao; LIANG, Ding; LU, Tong; LUO, Ping; SHAO, Ling. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*. 2022, vol. 8, no. 3, pp. 415–424.
41. CUENAT, Stéphane; COUTURIER, Raphaël. Convolutional Neural Network (CNN) vs Vision Transformer (ViT) for Digital Holography. In: *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*. IEEE, 2022, pp. 235–240.
42. BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. 2014.
43. CHO, Kyunghyun; VAN MERRIËNBOER, Bart; GULCEHRE, Caglar; BAHDANAU, -Dzmitry; BOUGARES, Fethi; SCHWENK, Holger; BENGIO, Yoshua. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014.

44. SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems.* 2014, vol. 27.

45. LUONG, Minh-Thang; PHAM, Hieu; MANNING, Christopher D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025.* 2015.

46. XU, Kelvin; BA, Jimmy; KIROS, Ryan; CHO, Kyunghyun; COURVILLE, Aaron; SALAKHUDINOV, Ruslan; ZEMEL, Rich; BENGIO, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. In: *International conference on machine learning.* PMLR, 2015, pp. 2048–2057.

47. CHENG, Jianpeng; DONG, Li; LAPATA, Mirella. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733.* 2016.

48. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N; KAISER, Łukasz; POLOSUKHIN, Illia. Attention is all you need. *Advances in neural information processing systems.* 2017, vol. 30.

49. LIU, Ze; LIN, Yutong; CAO, Yue; HU, Han; WEI, Yixuan; ZHANG, Zheng; LIN, Stephen; GUO, Baining. Swin transformer: Hierarchical vision transformer using shifted windows. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2021, pp. 10012–10022.

50. BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450.* 2016.

51. YU, Jiahui; LIN, Zhe; YANG, Jimei; SHEN, Xiaohui; LU, Xin; HUANG, Thomas S. Generative image inpainting with contextual attention. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pp. 5505–5514.

52. ZHANG, Han; GOODFELLOW, Ian; METAXAS, Dimitris; ODENA, Augustus. Self-attention generative adversarial networks. In: *International conference on machine learning.* PMLR, 2019, pp. 7354–7363.

53. WANG, Fei; JIANG, Mengqing; QIAN, Chen; YANG, Shuo; LI, Cheng; ZHANG, Honggang; WANG, Xiaogang; TANG, Xiaoou. Residual attention network for image classification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 3156–3164.

54. XIAO, Tianjun; XU, Yichong; YANG, Kuiyuan; ZHANG, Jiaxing; PENG, Yuxin; ZHANG, Zheng. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 842–850.

55. DOSOVITSKIY, Alexey; BEYER, Lucas; KOLESNIKOV, Alexander; WEISSENBORN, Dirk; ZHAI, Xiaohua; UNTERTHINER, Thomas; DEHGHANI, Mostafa; MINDERER, Matthias; HEIGOLD, Georg; GELLY, Sylvain, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.* 2020.

56. XIONG, Ruibin; YANG, Yunchang; HE, Di; ZHENG, Kai; ZHENG, Shuxin; XING, Chen; ZHANG, Huishuai; LAN, Yanyan; WANG, Liwei; LIU, Tieyan. On layer normalization in the transformer architecture. In: *International Conference on Machine Learning.* PMLR, 2020, pp. 10524–10533.

57. WANG, Wenhai; XIE, Enze; LI, Xiang; FAN, Deng-Ping; SONG, Kaitao; LIANG, Ding; LU, Tong; LUO, Ping; SHAO, Ling. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2021, pp. 568–578.

58. ISLAM, Md Amirul; KOWAL, Matthew; JIA, Sen; DERPANIS, Konstantinos G; BRUCE, Neil DB. Position, padding and predictions: A deeper look at position information in cnns. *arXiv preprint arXiv:2101.12322.* 2021.

59. DENG, Jia; DONG, Wei; SOCHER, Richard; LI, Li-Jia; LI, Kai; FEI-FEI, Li. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

60. RADOSAVOVIC, Ilija; KOSARAJU, Raj Prateek; GIRSHICK, Ross; HE, Kaiming; DOLLÁR, Piotr. Designing network design spaces. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10428–10436.

61. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

62. CORPORATION, Microsoft. *Windows 10* [https://www.microsoft.com/en-us/windows/]. 2015–.

63. PAWAR, Rutuparn. *Logo Images Dataset* [online]. 2021. [visited on 2022-09-21]. Available from: https://www.kaggle.com/datasets/inputblackboxoutput/logoimagesdataset.

64. HUSSAIN, Zaeem; ZHANG, Mingda; ZHANG, Xiaozhong; YE, Keren; THOMAS, Christopher; AGHA, Zuha; ONG, Nathan; KOVASHKA, Adriana. Automatic understanding of image and video advertisements. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1705–1715.

65. WU, Xian; XU, Kun; HALL, Peter. A survey of image synthesis and editing with generative adversarial networks. *Tsinghua Science and Technology*. 2017, vol. 22, no. 6, pp. 660–674.

66. LEDIG, Christian; THEIS, Lucas; HUSZÁR, Ferenc; CABALLERO, Jose; CUNNINGHAM, Andrew; ACOSTA, Alejandro; AITKEN, Andrew; TEJANI, Alykhan; TOTZ, Johannes; WANG, Zehan, et al. Photo-realistic single image super-resolution using a generative adversarial network. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

67. MIYATO, Takeru; KATAOKA, Toshiki; KOYAMA, Masanori; YOSHIDA, Yuichi. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*. 2018.

68. ZHAO, Hang; GALLO, Orazio; FROSIO, Iuri; KAUTZ, Jan. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*. 2016, vol. 3, no. 1, pp. 47–57.

69. KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

70. CHOI, Dami; SHALLUE, Christopher J; NADO, Zachary; LEE, Jaehoon; MADDISON, Chris J; DAHL, George E. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*. 2019.

71. WALT, Stéfan van der; COLBERT, S. Chris; VAROQUAUX, Gaël. *NumPy* [http://numpy.org]. 2022.

72. CLARK, Alex; CONTRIBUTORS. *Python Imaging Library (PIL)* [https://pillow.readthedocs.io/en/stable/]. 2022.

73. TEAM, OpenCV. *OpenCV* [https://opencv.org/]. 2000–.

74. PEDREGOSA, Fabian; VAROQUAUX, Gaël; GRAMFORT, Alexandre; MICHEL, Vincent; THIRION, Bertrand; GRISEL, Olivier; BLONDEL, Mathieu; PRETTENHOFER, Peter; WEISS, Ron; DUBOURG, Vincent; VANDERPLAS, Jake; PASSOS, Alexandre; COURNAPEAU, David; BRUCHER, Matthieu; PERROT, Matthieu; DUCHESNAY, Édouard. *Scikit-learn: Machine Learning in Python* [http://scikit-learn.org]. 2022.

75. CONTRIBUTORS, MMSegmentation. *OpenMMLab Semantic Segmentation Toolbox and Benchmark*. 2020. Available also from: https://github.com/open-mmlab/mmsegmentation.

Enclosed Media Contents

All code attachments for the thesis can also be found at https://github.com/halamto2/TAWR, along with a manual for setting up the Python environment. The state of the repository as provided is equivalent to the *TAWR Segformer* variant (see Section 5.2.1).

```
TAWR
├── README.md ........................... Project description and environment setup guide.
├── conda_packages_env.txt .................. List of conda packages used in the project.
├── env_setup_cmds.txt ............... Commands for setting up the Python environment.
├── train.sh .......................... Script for starting the training loop for the model.
├── validate.sh ......................... Script for starting the model validation process.
├── dataset ............................... Scripts for generating and loading the dataset.
│   ├── CLWDDataset.py
│   ├── CarDataset.py
│   └── generate_dataset_demo.ipynb
├── modules ................................ Building blocks for the deep learning model.
│   ├── Discriminator.py
│   ├── TAWR.py
│   ├── WatermarkRefiner.py
│   └── WatermarkRemover.py
├── pretrained_weights ...................... Initial weights for the transformer module.
│   └── pvt_v2_b1.pth
├── scripts ............................... Scripts for training and evaluating the model.
│   ├── evaluation.py
│   ├── options.py
│   ├── train.py
│   └── validate.py
└── trainers ......................................... Wrappers for training the model.
    ├── BasicTrainer.py
    └── TAWRTrainer.py
```