Removing Watermarks From Images With a Good Attitude

Jop Zitman 2670863
Sander Vržina 2678335
Tiberiu Iancu 2659445
Irakliy Marsagishvili 2637597
Sebastian Iozu 2663383

https://github.com/EndPositive/glowing-potato
Group 81

Abstract

Watermark removal is the task of reconstructing images that have been previously overlaid with obstructive elements (such as logos or text), to prevent copyright infringement. In this paper we propose two models to tackle the task of watermark removal: UNetWR, a Convolutional Neural Network-based architecture, and SwinWR, a model that uses the Swin transformer as the main building block. We find that UNetWR outperforms SwinWR, reaching a MS-SSIM score of 0.982 after just a few hours of training, while the transformer model does not manage to converge.

1 Introduction

Watermarks are patterns deliberately overlaid on images by stock image providers (e.g. Shutterstock, Adobe Stock etc.) or individuals, as a way to prevent media theft. Usually, watermarks are patterns that contain a few elements such as text, grid lines and icons, which vary widely in combinations, skewness and opacity. Reconstructing the original from a watermarked image is not a trivial task, as pixel information under the watermark is concealed or heavily altered.

Low-level image processing refers to image manipulation tasks, such as denoising, upscaling, deraining etc. The task of watermark removal is a low-level image processing problem, which lies in-between denoising (removing noise from images) and infilling (filling in missing parts of the image).

In the past years, such image processing tasks have been tackled mainly using **C**onvolutional **N**eural **N**etworks (**CNN**). Recently, the Transformer [15] has seen increased use in image processing tasks. Although originally aimed at tackling sequence transduction problems, such as **N**atural **L**anguage **P**rocessing (**NLP**), its success has inspired recent research to apply it as a building block in other machine learning areas. Notably, Liang et al. [9] proposes SwinIR, an image restoration model having as main building block the Swin Transformer [10]. It manages to achieve state-of-the-art performance across all low-level computer vision tasks, while maintaining a relatively low number of parameters.

Another CNN-based architecture, U-Net [11], has been primarily used in the past for the task of image segmentation. Notably, this network has a large number of feature channels in the upsampling part, which may prove useful in watermark removal.

In this paper, we tackle the task of watermark removal by building on top of U-Net [11] (**UNetWR**) and a pre-trained SwinIR [9] (**SwinWR**), and comparing their performance.

2 Background

The Transformer [15] is an artificial neural network architecture that fully relies on self-attention mechanism to process sequential data. Since its introduction, it has seen a massive increase in popularity among the research community, thanks to its wide success in NLP tasks. However, by itself, the Transformer was not fit to solve problems concerning input and output modalities other than text, such as images. The Swin Transformer [10], a Transformer-based image processing model, aims to improve on general-purpose image processing by constructing hierarchical feature maps containing windows with patches of fixed size and applying self-attention mechanism only locally within one window. This ensures that there is always a linear complexity to the size of the image. The local windows move and resize between self-attention layers, providing connections between local windows, which significantly enhances modelling power.

Transfer learning is a machine learning technique that aims to decrease training time and increase performance by starting the training process from a model pre-trained on large amounts of data, and retraining it on a smaller data set for a different (and usually more specialized) task. This has been common practice in solving image processing tasks with CNN-based networks, such as state-of-the-art classification models ResNet [7], VGGNet [12], AlexNet [8], InceptionV3 [13] etc.

As models in recent years have grown significantly in number of parameters, the computational power required to train such models has also increased. Notably, transformer-based natural language processing models, such as GPT-3 [4], have reached hundreds of billions of parameters. Thus, it has become customary to employ transfer learning from transformer models pre-trained on large amounts of data.

3 Method

3.1 Model Architecture

SwinWR. For the Swin-based model, we start from the pre-trained denoising SwinIR with an embedding size of 180 and window size of 8. Figure 1 shows a high level overview of the architecture. section 6 shows the original SwinIR architecture. We leave both shallow and feature extraction components unchanged, and freeze the weights for training. The last convolutional block is removed and replaced with a series of convolutional layers. Initially, 1x1 convolutions are performed to reduce the number of channels from 180 to 64 (similar technique to InceptionV3 [13]), which is then followed by a block of 3x3 convolutions from 64 to 3 channels. The model has a total of 131 thousand trainable parameters.

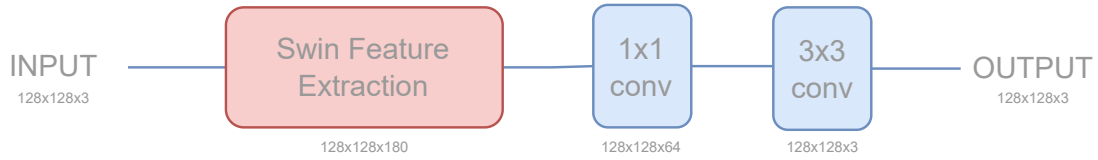


Figure 1: SwinWR architecture. The output shape of the operations is displayed next to each cell. We apply ReLU over the output of the convolutions.

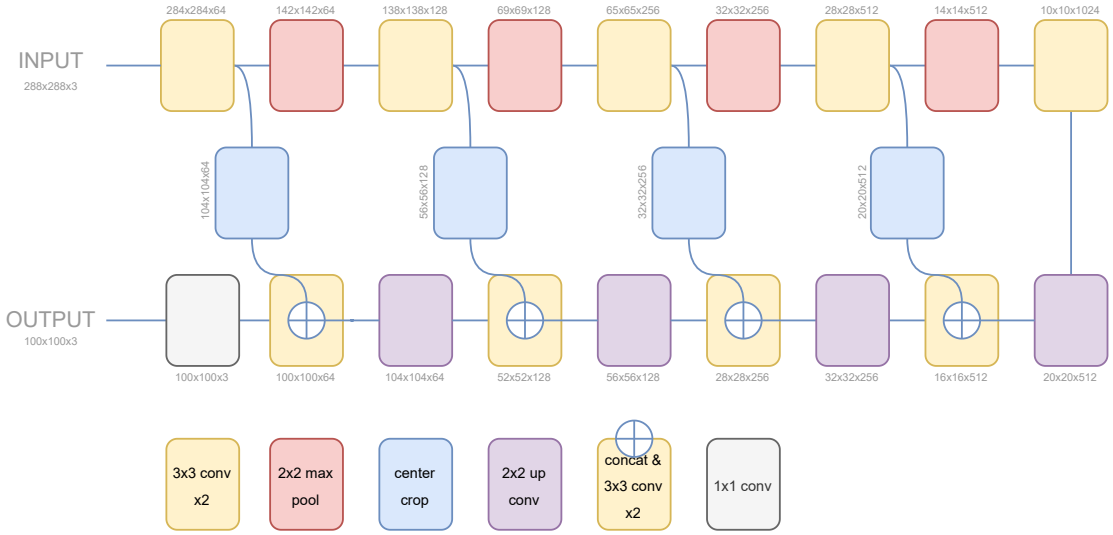


Figure 2: UNetWR architecture. The output shape of the operations is displayed next to each cell. The concatenation operation is performed along the channel axis and the double convolution block that follows decreases the number of channels by half. Each convolution is followed by ReLU and batch normalization, except the last one.

UNetWR. For the CNN-based model, we start from the U-Net architecture proposed by Ronneberger et al. and make a few modifications. A high level overview of the model is shown in Figure 2. The input size is set larger than the output, similarly to U-Net. In theory, this should allow for better infill performance, as a result of the model having more spatial context. Moreover, this lends us the possibility of larger batch sizes during training. We settle on an input size of 288x288 and adjust the output size accordingly to 100x100. We align image dimensions to the input size using mirror padding. The model has a total of 33 million (trainable) parameters.

3.2 Training Regime

The two models are trained on data consisting of images from Open Images V6 [6] overlaid with randomly generated watermarks (see section 6). Given the watermarked input image I_W, the models output the reconstruction of the clean image I_R:

$$I_R = H_{SwinWR}(I_W) + I_W \qquad I_R = H_{UNetWR}(I_W) \tag{1}$$

where $H_X(\cdot)$ is the function of the respective model X. Both models are trained to minimize the pixel-wise L_1 loss between the predicted reconstruction I_R and the clean image ground truth I_C:

$$\mathcal{L} = ||I_C - I_R|| \tag{2}$$

Since UNetWR's output size is smaller than the input, we perform a center crop on the target clean image I_C, same size as the output.

Figure 3:
Abstract watermark

Figure 4:
Shutterstock-like watermark

Figure 5:
123RF-like watermark

Initially, the images are overlaid with abstract watermarks consisting of randomly drawn lines, text and shapes of varying opacity, rotation, size and color (as seen in Figure 3). The models are trained until convergence, then we reprocess the data set, this time with specific watermarks (e.g. company logos). Intuitively, the models will converge faster on this new set, if previously trained on the abstract watermark set. This allows us to target a variety of watermarks with a fraction of the training time.

Figures 3, 4, and 5 show examples of the abstract watermark and two specialized watermarks, respectively. To recreate the specialized watermarks, every element is inserted with different sizes, rotations, colors and opacities; this enforces generalization.

4 Experiments

We perform grid search using the validation set to find the best hyperparameters for the CNN-based model. We settle for the Adam optimizer [5] with learning rate 0.001, weight decay of 0.0001 (L2 regularization) and dropout probability of 0.3. The batch batch size to 12, as to maximize the usage of the available 8GB VRAM of the RTX 3070Ti. We train both models on one GPU for 100 epochs. The hyperparameter search space is shown below in Equation 3.

$$\begin{aligned} dropP &= [0.2, 0.3, 0.4] \\ weightD &= [0.001, 0.0005, 0.0001] \\ LR &= [0.001, 0.0005, 0.0001] \end{aligned} \quad \Longrightarrow \quad \begin{aligned} dropP &= 0.3 \\ weightD &= 0.0001 \\ LR &= 0.001 \end{aligned} \tag{3}$$

After 12 hours of training, SwinWR is stuck in local minima and only learns to reproduce the input. An analysis of why SwinWR does not converge is presented in section 5. For the rest of this Section we focus on UNetWR.

Figure 6 displays an example of watermark removal using UNetWR. The model manages to remove most of the watermark elements. However, there is still noticeable artifacting left upon closer inspection. More examples of reconstructed images can be found in section 6.

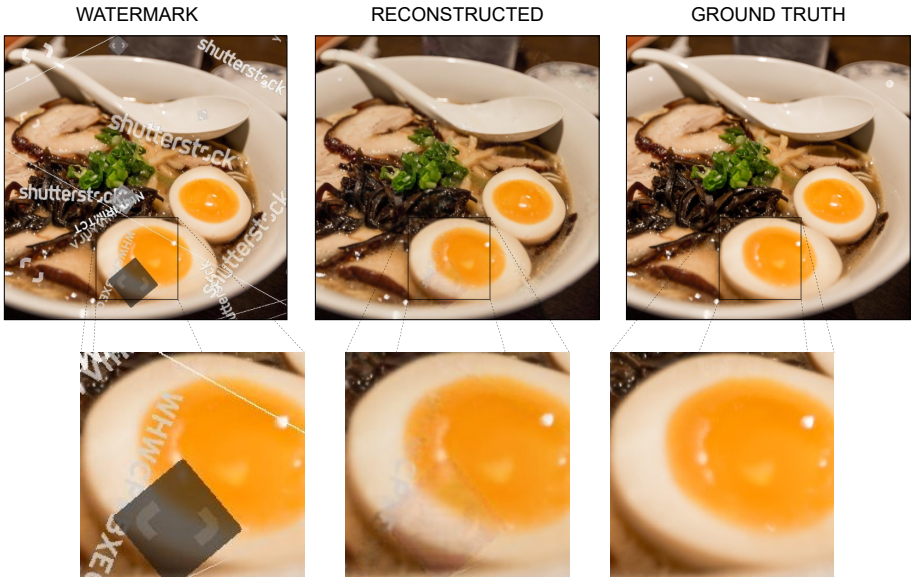


Figure 6: UNetWR specialized on Shutterstock-like watermarks.

We evaluate the performance of the model using MS-SSIM [16] between the reconstructed image and the ground truth. As a baseline, we calculate the MS-SSIM between the watermarked images and the ground truth. The results are summarised in Table 1.

	Abstract	Shutterstock-like	123RF-like
UNetWR	0.949	0.982	0.968
Baseline	0.811	0.845	0.864

Table 1: MS-SSIM score for UNetWR and baseline across the three watermark types.

In Figure 7 we display the usage of UNetWR that is specialized to remove the watermark from a real Shutterstock stock image. The reconstructed image shows a good attempt at removing the watermarks, however it failed to properly remove the author name in the middle. Furthermore, there are noticeable artifacts in the sky, which may be caused by a white transparent watermark on a similarly coloured background. A higher quality image is available in section 6. The validation loss over time is plotted over time in section 6.

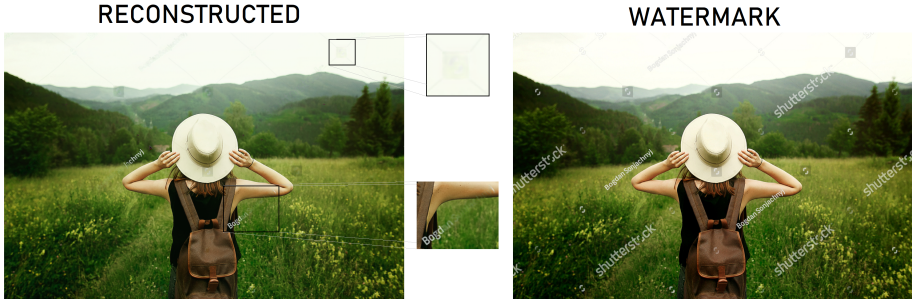


Figure 7: UNetWR reconstruction on a real Shutterstock watermark.

5 Discussion and Future Work

During training, SwinWR did not show any promising results, but rather it only learned to reproduce the input. In past reasearch, the Swin Transformer has proven to be very versatile in low-level image processing tasks. We attribute the poor performance in our task to three main pitfalls.

First, the Swin feature extraction component might get confused by the erratic watermarks we have presented during training, which it has not seen during the pre-training on the denoising task. As such, the extracted features might not provide useful information to the last convolutional layers. Second, the convolutional layers introduced by us might not be capable of learning such a difficult task with only 131 thousand parameters. Although SwinIR manages state-of-the-art performance with only a few thousand parameters in the last convolutional layer, in this research we did not re-train the feature extraction to fit the task. Third, due to hardware and time limitations, we could only train the model for 48 hours, which might not be enough for the model to show results.

SwinWR is asked to output a patch of size 128x128, same as the input. This gives it less context than UNetWR. To combat this, we have implemented MultiSwinWR, a model that takes an arbitrary number of input images (of size 128x128), passes them through the feature extraction, and concatenates the extracted features of each image before the last convolutional blocks. The advantage of this approach is that we can pass multiple patches of the same image through the network to give the output layers more context. For example, we could add a downscaled version of the image as extra context, or some surrounding patches of images. Due to time limitation, we have not been able to pursue this approach, so it remains a hypothesis for future research. The basis for this approach can already be found in the code in section 6.

Certain watermarks are designed to be fully opaque, and as such, removing them corresponds to the task of infilling. Since SwinIR [9] has shown great results in the past, we would like to see future research attempt this task using a more appropriate version of the Swin Transformer.

U-Net[11] was originally designed to be used in image segmentation tasks. A topic for future research could be watermark segmentation using U-Net, followed by watermark removal using a modified version of SwinWR, given the segmentation map.

For the hyperparameter grid search we used a training time of 1.5 hours per hyperparameter setting. We acknowledge that 1.5 hours might be too little to accurately determine which setup is best. Due to the amount of different parameter settings, it is infeasible to search for such a large amount of combinations on our setup (e.g. we did not perform grid search on the batch size, as the VRAM from the GPU was the limiting factor). For the same reasons we believe that SwinWR has most likely not fully converged.

6 Conclusion

In this work we propose two models to tackle the task of watermark removal. We show that the CNN-based UNetWR manages to converge and discover that the Swin-based model fails to do so due to some pitfalls in the convolutional layers implemented by us, which may not have enough parameters to generalize or do not receive useful information from the extracted features. UNetWR obtains an average MS-SSIM of 0.982 after only a few hours of training, and shows usable results after just one epoch.

References

[1] Datasets, transforms and models specific to computer vision, 2022.

[2] Tensors and dynamic neural networks in python with strong gpu acceleration, 2022.

[3] S. L. AB. Pillow (pil fork) documentation, 2022.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[5] J. B. Diederik P. Kingma. Adam: A method for stochastic optimization, 2014.

[6] Google LLC. Open images dataset v6, 2019. Open Images is a dataset of 9M images.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[9] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte. Swinir: Image restoration using swin transformer, 2021.

[10] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.

[14] Tali Dekel, Michael Rubinstein, Ce Liu, William T. Freeman. On the effectiveness of visible watermarks, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[16] Z. Wang, E. Simoncelli, and A. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003.* IEEE.

Appendix

This section discusses additional implementation details and optimizations, and provides additional examples from the models. We publicly release the source code for this paper on GitHub.

Data processing

We use Pillow [3], an image processing library, to overlay watermarks on top of images. Pillow performs operations on the CPU, while our models train on the GPU using PyTorch [2]. To avoid a major processing bottleneck due to GPU idling, we apply the watermarks before training; we reapply the watermarks every 10 training epochs to avoid overfitting. To further improve training speed, a custom image pre-fetching function was added to the data loader used during training, which ensures that the next batch of images in in the data set is already loaded in the GPU memory. As a result the active loading time of images was reduced from an average of 0.4 seconds to 0.008 seconds, which sped up the training by $\sim$ 2.5x.

We've implemented a custom `CropMirrorTransform` that uses use Torch vision transforms [1] to pad, crop and normalize during training. Random cropping ensures that every training epoch we get a different part of the image. When "over"-cropping causes black at the borders, the image is mirrored onto the border.

Watermarking

We create three types of watermarkers from scratch. The abstract watermarker generates a grid, lines, shapes and text using Pillow. For the specific watermarks (Shutterstock and 123RF-like), we attempted to extract and restore the elements as accurately as possible using a completely green stock photo, where the watermark was added on automatically by the websites. But due to the transparency of the elements, it is challenging to pinpoint the exact colors. Furthermore, some stock photo providers add randomized variables in their watermarks (e.g. Shutterstock slightly warps the text and shapes to make masking out using alpha matting impractical [14]).

More examples

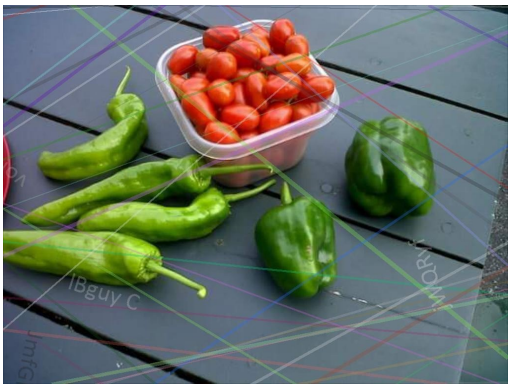


Figure 8:
Chaotic watermark

Figure 9:
Reconstructed

Figure 10:
Shutterstock watermark

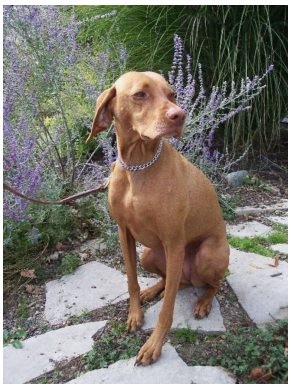
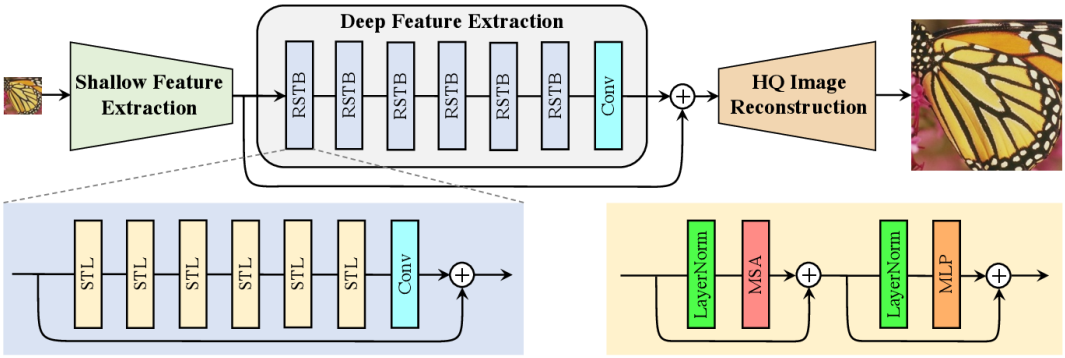


Figure 11:
Reconstructed

Figure 12:
123RF watermark

Figure 13:
Reconstructed

SwinIR architecture



(a) Residual Swin Transformer Block (RSTB)

(b) Swin Transformer Layer (STL)

Shutterstock Watermark Removal

Shutterstock validation loss against epochs