INM705: Deep Learning for Image Analysis

Víctor Abia Alonso

May 5, 2024

1 Introduction.

Watermarking problem
Watermarks serve as both a safeguard and a branding tool by embedding a semi-transparent image or text over original content to assert copyright ownership and deter unauthorized use. This protective measure is crucial for maintaining the integrity of intellectual property, particularly in the digital realm where reproduction is facile. Commercially, software companies utilize watermarks extensively in marketing campaigns, strategically placing them on images to entice clients towards purchasing a watermark-free version. Creating a watermark is relatively straightforward, involving the overlay of transparent text or logo onto an image. However, removing them is a far more intricate process that requires substantial photo editing skills. This involves meticulously retouching each pixel affected by the watermark and reconstructing the obscured portions of the image, a task that becomes exponentially more challenging when the watermark pattern is extensively repeated across the image.

This project explores different deep learning vision models to remove watermarks from images. This raises ethical concerns about using technology and the advances of artificial intelligence against the intellectual property of the creators of the images, which can have negative economic impact on photographers and artists whose business relies on licensing. The objective of this project is to explore the capabilities of this models without being detrimental for society to not pose a risk to systems that rely on watermarking. As such, the models are not optimized for generalization: a specific domain and watermark is chosen instead of training in a more diverse setting. This allows different models to be explored, while limiting their general use for unwatermarking any image.

Unwatermarking images is intrinsically a generative task as it involves generating new pixel data that is consistent with the unmarked parts of the image. As such the main models this work aim to compare are the Generative Adversarial Networks (GANs), Diffusion models and Variational Autoencoders(VAEs), which represent the state of the art of generative images. Additionally, a more basic Convolutional Autoencoder is used a baseline to compare against the other models. In order to compare them, visual qualitative analysis is used in addition with two quantitative metrics.

Dataset.
All models were trained in a supervised fashion taking the watermarked images as input and the original images as targets, which allows for easier training routines. There are little datasets with pairs of original-watermarked images probably because they promote illegal and unethical practices violating copyrights licences. The only datasets referring to watermarks involve classification tasks to detect whether an image has been watermarked or not [1]. However, creating an image dataset and adding watermarks to it is relatively straightforward.

I designed a Python script called watermarking.py which, when given an image dataset in Hugging Face, downloads it, adds watermarks, and reuploads the watermarked dataset to a new Hugging Face repository. The watermarks are words or sentences in white, for which one can specify the text, font, rotation, position, size, and opacity. Despite the potential for diversity, I decided to focus on how models behaved with just one type of watermark, which involved adding the text "transcendingvictor"

in Arial with a font size of 60 at the center, no rotation, and an opacity of 82% (210/255). To further simplify the task so that the different model capabilities could be fully appreciated on the simple task, I chose to use the Oxford Flowers Dataset [2] which includes around 8,000 images of flowers (7,169 training and 1,020 validation) from 101 categories, intended for image classification tasks; this training-test split was conserved throughout the project. As the task is not classification but generation, I used all the flowers indiscriminately and removed the original labels from them, so I reuploaded the unlabeled Original Flowers Dataset to my Hugging Face account, then I processed it and uploaded the Watermarked Flowers Dataset). The dataset is composed of first-shot pictures of very different flowers with a somewhat blurry background of grass. As such, the generative models would just have to focus on the flower itself and not on other minor details they might not have been trained for, which would be an extra layer of difficulty if I used a dataset of everyday scenes or my phone gallery.

Context for the project.
With the explosion of powerful diffusion models which revolutionized generative AI on 2022 like DALLE2 [3] or StableDiffusion [4] the task of unwatermarking an image actually was a by product of this models software like Adobe's Photoshop allowed to regenerate selected portions of the image conditioned on the surroundings. There is also this software that allows you to unwatermark some images for free and it automatically recognizes them. However, this products involve very general models with a backbone with hundreds of millions of parameters, while this project focuses on designing and training models specifically for the task of unwatermarking. There are online references [5] [6] for the task of removing watermarks that are quite vague and focus solely on GANs without exploring other models. Additionally, inspired by the research of Marimont et al. [7], which uses diffusion-based models for anomaly detection in brain scans, I decided to reinterpret their concept of anomalies as noise for my problem and consider watermarks on an image as noise as well. The inclusion of VAEs aimed to bring a global perspective to the problem of removing watermarks using different generative models.

2 Methodology

2.1 Dataset and Training

The dataset used to train and validate the models consisted of pairs of original and watermarked images of flowers. The pixel values of the images were scaled to a range between 0 and 1, which aids in the training of neural networks. The images were shuffled within the training set to encourage generalization. After each epoch, logs of the relevant training and validation losses were recorded in Weights and Biases, along with five pairs of images to visually inspect the models' performance. Additionally, checkpoints containing the model's and optimizer's parameters were saved locally after every epoch, allowing training to be resumed from any designated point if necessary.

2.2 Metrics: PSNR and SSIM

Besides the visual inspection of the data, which ultimately determines the value of the model, two metrics of image reconstruction are monitored throughout the experiments. The first metric is the Peak Signal-to-Noise Ratio (PSNR), defined as $\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}_I^2}{\text{MSE}}\right)$, where MAX_I represents the maximum possible pixel value of the image (1 in this case, as pixels are scaled, and MSE is the mean squared error between the original and the reconstructed image. This function is monotonicly decreasing with respect to the Mean Squared Error (MSE), which is already used for reconstruction loss and is included in all the model losses. As such, it is not logged separately for every epoch; instead, PSNR is exclusively used in the evaluation of the results while MSE is used for training. The second metric, the Structural Similarity Index (SSI), is considered to be more aligned with human perception as it accounts for image degradation perceived as changes in structural information. It

also incorporates perceptual phenomena such as luminance masking and contrast masking. SSI uses the average pixel values (μ_a and μ_b), the variance of each image (σ_a^2 and σ_b^2), the covariance (σ_{ab}) between them, and stabilizing constants (c_1 and c_2) to calculate the similarity between two images with the formula: $\text{SSI} = \frac{(2\mu_a\mu_b+c_1)(2\sigma_{ab}+c_2)}{(\mu_a^2+\mu_b^2+c_1)(\sigma_a^2+\sigma_b^2+c_2)}$. None of the models is directly optimized to reduce the dissimilarity between original and generated images as indicated by SSI, because doing so would be computationally intractable due to the inclusion of covariances. Instead, SSI serves as a metric that is logged for evaluation purposes.

2.3 Convolutional Autoencoder: Baseline

Autoencoders are a type of artificial neural network designed to learn dense, lower-dimensional encodings of unlabeled data. They consist of an encoder and a decoder, connected by a bottleneck layer with a smaller dimension. By minimizing the reconstruction error between the input and the output, autoencoders compel the network to learn an efficient representation. Originally, traditional autoencoders were fully connected neural networks [8]. However, with the evolution of deep learning in the 2010s, more sophisticated versions have emerged, such as Sparse Autoencoders [9], Denoising Autoencoders [10], and Variational Autoencoders [11]. For this project, a Convolutional Autoencoder (CAE) serves as the baseline model. It incorporates convolutional layers in both the encoder and decoder components, which allows it to capture spatial information crucial for analyzing images. The model is trained on pixel-wise Mean Square Error (MSE) between the input and output (reconstruction loss).

The **encoder** consists of a series of three convolutional layers, each designed to progressively refine the network's understanding of the input image. The first layer employs 32 filters with a kernel size of 3x3, using a stride of 2 and padding of 1. This configuration reduces the dimension of the input and it-s chosen to double its depth. Subsequent layers with increasing depth capture progressively abstract features, embodying the principle of **Hierarchical Feature Learning**. In this framework, textures and edges detected in early layers (which are simpler and thus require fewer filters) serve as input to deeper layers that produce more abstract representations. Additionally, the **increasing receptive field** of deeper layers enables them to encompass larger portions of the image, thereby facilitating more complex and nuanced interpretations per filter. After each convolutional layer, batch normalization is applied to stabilize the learning process by normalizing the activations. This step enhances convergence speed and improves training efficiency. The ReLU activation function is utilized for introducing non-linearities into the model with computational efficiency, enabling the capture of more complex patterns. Each subsequent layer increases the number of filters—64 and then 128—with the same stride and padding settings, further reducing spatial dimensions while expanding the receptive field of the network's deeper layers.

The output of the encoder is the **bottleneck**, which serves as a lower-dimensional representation. This compact layer captures the essence of the input data by retaining only the most significant features necessary for reconstruction. It is sized to ensure meaningful data compression while preserving critical information essential for successful reconstruction. The dimensions of the bottleneck are defined by 128 filters, each with a spatial resolution of 62 by 62. Although adding a fully connected layer in the bottleneck could further compress the representation and allow potentially useful non-linear combinations of features, this approach is not adopted in this case. Adding such a layer would disrupt the preservation of spatial information by flattening the structure, and would also lead to a significant increase in the number of parameters, calculated as $n_{\text{inputs}} \times n_{\text{outputs}}$, making the network computationally expensive. Thus, this Convolutional Autoencoder remains a fully convolutional network.

The configuration of the **decoder** mirrors that of the encoder. It utilizes a series of three transposed convolutional—or deconvolutional—layers, which incrementally reconstruct the original spatial dimensions from the compressed feature representations. The first deconvolutional layer transforms the 128-dimensional feature maps back to 64 dimensions; the second layer reduces these to 32; and the third layer restores the original 3 channels of the input image. Each layer includes batch normalization and ReLU activations to maintain non-linearity and enhance the efficiency of the reconstruction process. The final layer employs a sigmoid activation function, ensuring that the output pixel values

are normalized between 0 and 1, thus matching the format of the pre-processed input images. An additional interpolation step is introduced before producing the output to align the dimensions with the input. The progression of image dimensions (500x500 - 250x250 - 125x125 - 62x62 - 124x124 - 248x248 - 496x496) necessitates this step due to rounding errors introduced by the non-even dimension of 125, which could otherwise disrupt the dimensional integrity.

The choice of a stride of 2 in the convolutional layers, both in the encoder and decoder, balances dimensionality reduction (unachievable with a stride of 1) with the retention of sufficient information for accurate reconstruction. This setup circumvents the potential loss of detail common with traditional pooling layers, which also reduce dimensions, thus making strided convolutions a preferable choice for preserving the integrity of the image's information.

The objective in training this CAE is that by encoding watermarked images, whose decoded outputs are trained against the originals, the CAE learns to encode the essential information for reconstructing the image while omitting the watermark information.

2.4 GANs

Generative Adversarial Networks (GANs), designed by Ian Goodfellow in 2014 [12], are a class of algorithms in which two neural networks engage in a zero-sum game. These networks consist of a generator, which produces data indistinguishable from real-world data, and a discriminator, which evaluates whether data is real or produced by the generator. The dynamic competition between these networks drives continuous improvement, culminating in a highly effective generator. For this task, the generator is trained to remove watermarks from images, while the discriminator learns to classify the generator's outputs as fake. Training GANs involves a delicate balance to ensure that neither network outperforms the other significantly. Common pitfalls include one network (usually the discriminator) becoming too proficient, preventing the other from learning. To mitigate this, losses from both networks are continuously monitored, allowing adjustments to training dynamics and update frequencies as needed. Another issue is the generator potentially fooling the discriminator with subtle, undetectable changes that do not enhance image quality. To address this, outputs are routinely inspected visually.

The architecture of both networks relies on convolutional layers suited for image input. The generator produces an image, while the discriminator outputs a binary classification—the probability that the input image is original (i.e., not generated). The architecture of the generator mirrors the baseline Convolutional Autoencoder (CAE); indeed, GAN1 employs the pre-trained parameters of the CAE but introduces a new loss function focused not on reconstruction but on the likelihood of its output being classified as real by the discriminator. Subsequently, I incorporated reconstruction loss (MSE of pixel values between input and output) into the generator's loss function to counteract the discriminator's increasing accuracy (GAN2). Ultimately, I modified the training protocol to update the discriminator once every three batches and increased the weight of the reconstruction loss within the discriminator's training regimen (GAN3). More details are provided in Section 3: Results.

The architecture of the discriminator consists of five convolutional layers followed by a fully connected layer that terminates in a single neuron, ultimately deciding the authenticity of the input image. The first convolutional layer receives an input with three color channels (RGB images) and employs 32 filters. The first three convolutional layers use 3x3 filters with a stride of 2 and padding of 1, progressively reducing the spatial size of the image while doubling the number of filters. This setup captures increasingly complex features as the receptive field expands, similar to the CAE architecture. LeakyReLU, with a negative slope coefficient of 0.2, is used as the activation function, allowing the model to maintain gradient flow during training and stabilizing updates across the network. Each activation is followed by batch normalization and a dropout layer (with a rate of 25%), which help regularize the model by nullifying a portion of the features, thus preventing overfitting, and accelerating convergence. The number of filters increases sequentially - 64, 128, 256, and 512 - enabling the discriminator to capture increasingly nuanced features, crucial for making subtle distinctions between real and generated images. The final two convolutional layers do not reduce dimensionality (using a

stride of 1 and padding of 1), focusing instead on refining feature representations in preparation for the classification task. After the convolutional stages, the feature maps are flattened into a single vector and passed through a fully connected layer, which outputs a single value representing the discriminator's assessment; opting for a dense layer with a single neuron as output is preferred as it significantly reduces the parameter count.

The output of the discriminator consists of real numbers (logits), not probability distributions per se. This design is chosen because its output is always evaluated against an absolute label (0 or 1) using Binary Cross Entropy (BCE) as the loss function. For computational efficiency, the sigmoid function, which is implicitly the last layer of the model, is omitted and combined with the BCE loss. This integration helps prevent numerical overflow and underflow in intermediate steps by merging the two functions. The PyTorch function `torch.nn.BCEWithLogitsLoss` is used for training.

Binary Cross-Entropy (BCE) Loss:

$$L_{\text{BCE}} = -\left[y \cdot \log(p) + (1-y) \cdot \log(1-p)\right]$$

Discriminator Loss:

$$L_{\text{Discriminator}} = L_{\text{BCE}}(\text{real}) + L_{\text{BCE}}(\text{fake})$$

Generator Loss (GAN1):

$$L_{\text{Generator1}} = L_{\text{Fool}} = L_{\text{BCE}}(\text{fake\_preds}, 1)$$

Generator Loss (GAN2):

$$L_{\text{Generator2}} = L_{\text{Fool}} + 0.5 \times \text{MSE}(\text{fake\_imgs}, \text{original\_imgs})$$

Generator Loss (GAN3):

$$L_{\text{Generator3}} = L_{\text{Fool}} + 3 \times \text{MSE}(\text{fake\_imgs}, \text{original\_imgs})$$

Figure 1: Formulas representing the loss functions used in GAN training experiments. Note that in GAN3 the Discriminator is only trained once every 3 batches.

For training, I initially adopted a sequential learning approach, hypothesizing that the generator was already proficient due to inheriting weights from 40 epochs of training with the CAE. However, after independently training the discriminator for 10 epochs using the generator's outputs, the generator ceased to show significant improvements and training stagnated at a local minimum. This sequential approach provided the discriminator an undue advantage, subsequently rendering the task of fooling the discriminator exceedingly difficult for the generator. Consequently, no notable improvements were observed in the generator's performance, suggesting that the task of deceiving the discriminator had become overly challenging. A simultaneous training approach, in which both the generator and the discriminator are updated within each batch, is preferred as it fosters the dynamic competition that is crucial for optimizing GAN performance. This method capitalizes on the adversarial interaction, avoiding the pitfalls of each network settling into a local minimum independently.

2.5 Diffusion Models

Diffusion probabilistic models were originally [13] inspired by the way fluids mix due to the random motion of particles in non-equilibrium thermodynamics. These models leverage this concept by gradually transforming data into a random state through a noising process and then learning to reverse this transformation. Diffusion models are a type of generative model that generates data by modeling and reversing a diffusion process. The process consists of a forward phase (diffusion), which involves gradually adding noise to the data at each step using a Markov chain until the data is completely converted into random Gaussian noise, and a reverse phase (denoising), where a neural network is trained to incrementally reverse the diffusion process at each step, effectively learning to denoise the data until it retrieves a sample from the target distribution. Diffusion models have proven highly

effective and now serve as the backbone of state-of-the-art models like Google's Imagen (for images) [14], OpenAI's Sora (for video) [15], and WaveGrad (for audio) [16].

The UNet architecture is commonly employed for diffusion tasks, aiming to incrementally reduce noise at each step of the process. Consequently, the model typically involves as many passes through the UNet as there are steps in the diffusion process. The encoder in the UNet architecture consists of four 'DoubleConv' blocks. Each block applies two sets of convolutions, incorporating batch normalization and ReLU activation. The initial block accepts an input image with 3 RGB channels and expands the feature depth to 64 channels. Subsequent 'DoubleConv' blocks progressively increase the channel capacity to 128, 256, and finally 512. Each block's output is then downsampled using max pooling with a factor of 2, effectively reducing the spatial dimensions while concentrating the network's capacity on extracting salient features essential for effective denoising and segmentation. The decoding path of the UNet comprises four up-sampling 'DoubleConv' blocks, which utilize transposed convolutions to enlarge the spatial dimensions of the processed features. Unlike the simpler CAE architecture, the UNet incorporates concatenation steps that merge features from the encoder path with the up-sampled features from the decoder. This approach, often referred to as 'skip connections'—analogous to the residual connections found in a ResNet—enables the network to leverage both high-level abstract information and low-level, detailed features, which are crucial for precise denoising in generation tasks.

In order to design the training of the diffusion model, two approaches were considered. The first approach involves adding noise to images to some extent, such that they become more diffused but still allow one to infer the flower. This is crucial because if we were to train a model to construct a flower from complete noise, it would not know exactly which flower is being presented in each case. Therefore, the idea is to add noise to an image until the watermark is no longer recognizable but the flower can still be inferred. The model would then be trained on reconstructing images back to their original, denoising the watermarked images. This approach is feasible and aligns with the common use cases of diffusion models. However, it is significantly more computationally expensive than the other models in this project due to the step-by-step nature of diffusion models, which involve several passes through a U-Net. Additionally, training to reconstruct an image by making sense of the patterns is a harder task than directly aiming to remove watermarks. This process often necessitates much more extensive training to achieve reliable results.

The second approach treats the watermark as noise itself, enabling the diffusion model to learn to denoise the watermarked image. This method capitalizes on the availability of both datasets—a scenario not typical in image generation tasks [reference]. The original and watermarked images are interpolated by taking a weighted average defined by the parameter $t \in [0, 1]$, where $t = 0$ represents the clean, original image, and $t = 1$ corresponds to the fully noisy image. Intermediate values of t represent varying opacities of the watermark, maintaining the consistency of all other pixels. This approach is inspired by Marimont et al. [7].

This second approach is chosen because it efficiently meets the project's needs. The number of diffusion steps is set to 10, which is relatively low for common generative diffusion tasks but is sufficient given the high similarity between the input and target images. Additionally, this approach simplifies the diffusion problem by creating a suitable noise scheduler. The noise levels are linearly defined through the interpolation of the images, which allows for the automatic calculation of the image associated with each level of noise using a straightforward function, thus simplifying the training process. The training loss is MSE reconstruction loss.

2.6 Variational AutoEncoders.

Variational Autoencoders (VAEs) , introduced in 2013 by Kingma and Welling in their paper titled "Auto-Encoding Variational Bayes" [11], are a type of generative model that builds on the architecture of traditional autoencoders with a probabilistic twist in the bottleneck. Unlike conventional autoencoders that encode input data into a compressed latent space and reconstruct the output directly from this space, VAEs modify the encoding process by producing a distribution over the latent space. The decoder then samples from this distribution to generate the data point. In this model, the latent

space comprises a Gaussian probability distribution defined by the mean (μ) and the variance (σ^2), which are layers directly connected to the decoder. To enable training through backpropagation, which would otherwise be hindered by the non-differentiable sampling operation, an element ϵ is sampled from a normal distribution during each run of the network. The decoder input is then calculated as $z = \sigma \times \epsilon + \mu$, allowing the network to be trained on μ and σ.

The architecture used in this case closely resembles the previously discussed convolutional autoencoder, featuring three convolutional layers—with a kernel size of 4, stride of 2, and padding of 1—and three mirroring deconvolutional layers. ReLU is employed as the activation function, and no regularization layer is applied. The number of filters progresses from 3 to 64, to 128, and finally to 256 at the end of the third convolutional layer. Following this downsampling process, the architecture bifurcates into two distinct pathways using 1×1 convolutions: one for producing the mean (μ) of the latent distribution, and another for the logarithm of the variance ($\log \sigma^2$). These layers directly output the parameters of the latent space distribution from which the model will sample, which is crucial for the reparameterization step that follows. The encoder outputs two parameters: μ and $\log \sigma^2$, from which the standard deviation (σ) is derived by exponentiating half of $\log \sigma^2$. Random noise (ϵ) is then added to μ scaled by σ, thus producing a sample z from the latent distribution. This z vector is then fed to the decoder which uses three transposed convolutions with widths of 128, 64, and finally 3—the number of color channels. An interpolation layer ensures that dimensions match (the same reason as in the convolutional autoencoder where 500 is not divisible by 2^3). Additionally, a sigmoid layer ensures the output pixel values are between 0 and 1.

The reason why $\log \sigma^2$ is used instead of directly using the standard deviation, which is what is actually needed for the calculation, is that activations in the neural network can be negative, but variance and standard deviation cannot be. By parameterizing the variance in logarithmic space, we ensure that the variance is always positive and can leverage the exponential function's smooth gradient properties for efficient backpropagation. Additionally, $\log \sigma^2$ and μ were chosen to be calculated as convolutions, instead of flattened and dense layers, in the spirit of preserving the spatial features of the images and also to avoid a huge parameter count, capitalizing on the parameter sharing of convolutional layers.

The loss function of Variational AutoEncoders is called the Evidence Lower Bound (ELBO) and consists of two main components: the reconstruction loss and the KL Divergence. The reconstruction loss is measured as the Mean Squared Error (MSE) of pixels between the input and the output image. The KL Divergence between the learned distributions and the Standard Normal Distribution, which is assumed to be the prior, serves as a regularization term. This term forces the encoded latent variables to approximate the standard normal distribution, aiding in preventing overfitting by discouraging the model from encoding input data too specifically to nuances that might not generalize well outside the training set. It promotes desirable properties in the latent space, such as continuity and completeness, which are advantageous for generating new data points through interpolation and for smooth latent space traversal. Despite its beneficial properties, the usefulness of the KL term for the process of unwatermarking is not immediately apparent. In fact, the weights of the KL and MSE terms were adjusted during training after observing that the dominance of the KL term was hindering the training process. This issue led to the implementation of KL annealing, a technique to balance the influence of the KL term over time (see more in the results section).

The Kullback-Leibler Divergence (KL Divergence) quantifies how one probability distribution diverges from a second, expected probability distribution. For continuous variables, it is defined generally by the integral:

$$\mathrm{KL}(P \parallel Q) = \int p(x) \log \frac{p(x)}{q(x)} \, dx$$

Where $p(x)$ and $q(x)$ are the probability density functions of distributions P and Q, respectively.

In the context of Variational AutoEncoders (VAEs), where we deal with Gaussian distributions, the KL divergence between the approximate posterior $q(z|x)$ (represented by Gaussian parameters μ and σ^2) and the prior distribution $p(z)$ (standard normal distribution) simplifies to:

$$\mathrm{KL}(\mu, \log \sigma^2) = -0.5 \times \sum \left(1 + \log \sigma^2 - \mu^2 - \exp(\log \sigma^2)\right)$$

This formula calculates the KL divergence for each element of the latent vector and sums them up, effectively regularizing the encoder by penalizing divergences from the prior distribution.

3 Results

In this section I present and compare the results of each model in terms of the quantitative metrics and sharing some visual examples. All of the metrics provided and the images shown can be accesed by running the Jupyter Notebook inference.ipynb which is on the github repository of this project.

Model	**Average MSE**	**Average SSIM**	**PSNR (dB)**
CAE	0.00219	0.86693	26.59
Generator3	0.01115	0.63299	19.53
Diffusion	UNKNOWN	0.8999*	UNKNOWN
VAE	0.07264	0.49516	11.39

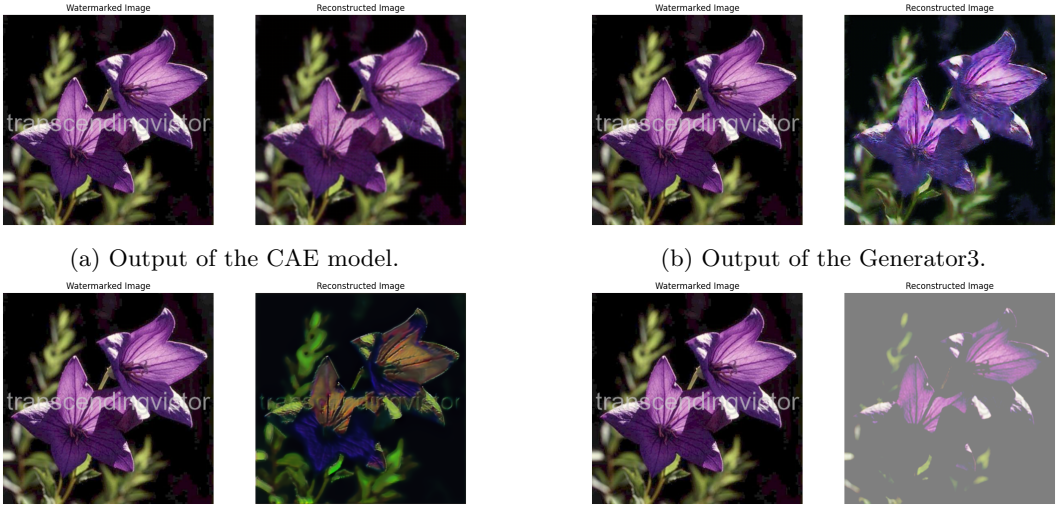
Table 1: Performance -on the validation dataset- of different models used in the project.

The performance of the models utilized in the project, as indicated by the metrics, varies significantly across different approaches. The baseline Convolutional Autoencoder (CAE) shows the best results with an Peak Signal-to-Noise Ratio (PSNR) of 26.59 dB probably because it was solely train for this task, an average Structural Similarity Index Measure (SSIM) of 0.86693. These metrics suggest a high fidelity in image reconstruction, making it a robust choice for tasks requiring precision. In contrast, the Generator3 model from the GAN setup demonstrates more moderate performance, with PSNR of 19.53 dB and SSIM of 0.63299. Although less effective than the CAE in terms of both error metrics and perceptual quality, it still offers valuable enhancements over more traditional approaches. The Variational Autoencoder (VAE), however, presents a challenge with considerably lower performance metrics: SSIM of 0.49516, and a PSNR of just 11.39 dB. These figures indicate a lesser ability to recreate accurate images, reflecting perhaps its design which trades off reconstruction accuracy for the ability to generate new images from the learned distribution, which is not necessarily useful in this case. The data for the diffussion model takes much longer to calculate precisely, and it's not very illustrative as the model clearly performs poorly; the SSIM datapoint was taken from weights and biases.

Besides the quantitative metrics that provide a sense of performance, we also evaluate the models using images from the validation dataset. Personally, I believe Image 132, which is presented, favors the CAE and does not highlight the strengths of the VAE, particularly as it struggles with dark colors. Nonetheless, it serves as a good example of how the models perform overall. It demonstrates the main results of the project: the CAE, with its simple architecture, manages to realistically remove most of the watermark, although closer inspection reveals some remnants. The GAN consistently removes the watermark in almost every case but sometimes loses detail, especially in dark backgrounds. The diffusion model does not perform well, potentially due to limited training. The VAE successfully removes the watermark but also struggles with detail in dark backgrounds. Below I comment on each model in more detail.

3.1 Autoencoder Results

The Autoencoder demonstrated promising visual results from the early stages of training. Notable improvements were observed as early as epoch 3, with gradual enhancements occurring incrementally thereafter. The model performs particularly well under conditions where the background contrasts



(a) Output of the CAE model.

(b) Output of the Generator3.

(c) Output from the Diffusion model.

(d) Performance of the VAE model.

Figure 2: Comparative outputs of validation image 132 of different models.

significantly with the watermark; however, it encounters difficulties when tasked with removing watermarks on light-colored backgrounds. This challenge arises because the pixel-wise Mean Squared Error (MSE) is minimal when white (of the watermark) is superimposed on white (of the background), thereby reducing the training incentive to adjust these areas. Additionally, the sharpness of characters within the watermark is not fully restored, likely due to the limitations in capturing high-frequency details.

In general, the model tends to lose some detail during the reconstruction process, which becomes more apparent in larger images. This loss of detail could be attributed to the inherent trade-offs of the MSE loss function, which may not perfectly align with human visual perception, particularly in terms of sharpness and texture detail. As the training progresses, the model improves in handling diverse backgrounds and varying levels of contrast between the watermark and the background, yet the subtle details and the crispness of the watermark edges continue to pose challenges. This analysis highlights the model's capabilities and limitations, providing a foundation for further refinement to enhance its performance in complex scenarios.

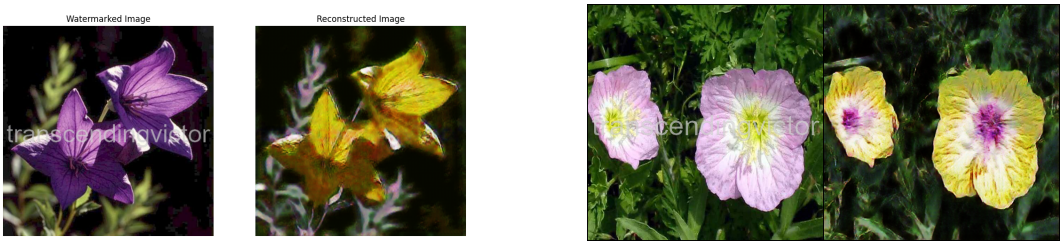
The observation of consistently lower validation losses compared to training losses in the Convolutional AutoEncoder can be attributed to the effects of batch normalization. During training, batch normalization normalizes the data based on each batch's specific statistics—namely its mean and variance. These statistics can vary significantly from batch to batch, which introduces a degree of noise into the model's learning process. This variability serves as a regularizer, helping to prevent the model from overfitting to the specific idiosyncrasies present within the training data. Conversely, during the validation phase, batch normalization employs the moving averages of these batch statistics, rather than the statistics of the individual validation batch. These moving averages are accumulated over the course of training and tend to be more representative of the dataset as a whole. This approach provides a more stable basis for the normalization process during model evaluation, which often results in more reliable performance and lower losses on validation data. This differential use of batch statistics highlights the dual role of batch normalization in improving model generalization and reducing overfitting, thereby enhancing the robustness of the model's predictions on unseen data.

3.2 GAN Results

GAN1: trained for 60 epochs solely on fooling the discriminator.
During the initial training phase, visual inspection revealed that the generator significantly lost the

ability to accurately replicate the colors of the flowers in the watermarked images, often only reconstructing the shape correctly. This degradation in performance likely results from the generator's sole focus on fooling the discriminator rather than optimizing for pixel-wise reconstruction loss. This focus provides no incentive for the generator to closely replicate the true image characteristics and just to fool the discriminator in what it sees, which doen't need to be what humans see, leading to deviations from realistic outputs. As training progressed into the later epochs, the generator became quite successful at removing watermarks, overcoming issues related to background colors that resembled the watermark. However, issues with color fidelity persisted throughout, with noticeable problems such as purple colors incorrectly turning to yellow and vice versa, indicating a significant challenge in the model's color consistency.

Considering these challenges, it becomes apparent that incorporating a term related to reconstruction loss could provide the generator with an incentive to improve image fidelity. Commonly in GAN training, generators develop patterns that primarily aim to deceive the discriminator, which may not contribute to visual accuracy, often referred to as "bad habits." This realization pressures a potential restart of the training process. However, since the unwatermarking effect was notably taking place, I opted for a midpoint solution by resuming training from epoch 40, which showed a more balanced validation loss between the generator and discriminator, instead of restarting from epoch 60, the last epoch. This approach aimed to fine-tune the model, leveraging the gains already made while addressing its deficiencies in producing realistic color representations.



(a) GAN1, epoch 41, Image 132.

(b) GAN1, epoch 60, Image 14.

Figure 3: Outputs of the GAN with failing to replicate yellow and purple colours due to the only focus of fooling the discriminator.

GAN2: trained for 10 epochs with some reconstruction loss.
After 10 epochs, the progression in model performance was not clearly defined. The SIM loss, which measures the similarity between generated and real images, generally increased slightly—indicating that the generated images were becoming more distinct from the originals, although it fluctuated throughout the epochs. Conversely, the reconstruction validation loss decreased significantly in the final epoch, suggesting an improvement in image fidelity, yet it had reverted to its initial levels just before this drop. Overall, the metrics remained stable, and no definitive conclusions could be drawn from them alone.

Visual examination of the five logged images revealed that the color quality and overall image quality were still markedly different from the expected results. In response to these observations, I opted for a more aggressive strategy by increasing the weight of the reconstruction loss in the total generator loss from 0.5 to 3. Additionally, since the discriminator's losses were still two orders of magnitude greater than those of the generator, I decided to reduce the frequency of discriminator training to one-third of the generator's training frequency. This adjustment was aimed at balancing the training dynamics and enhancing the generator's performance.

GAN3: trained for 21 epochs with the discriminator updated every three epochs.
Trained for an additional 21 epochs with a strengthened focus on reconstruction error and updating the discriminator every third epoch, the training showed signs of approaching a solution to the color accuracy problem, and the quality of the images generally improved. However, the training process was marked by instability, with performance varying significantly from epoch to epoch. Notable checkpoints were visually determined from the logged images at epoch 16, which showed promising results, and from the training data, where epoch 19 exhibited favorable loss metrics with higher discriminator loss

and lower generator loss. Despite these improvements, the discriminator's loss remained substantially lower, an order of magnitude below the generator's, still suggesting potential issues in training balance that may need further adjustments to fully optimize the model's performance.

3.3 Diffusion Results

The diffusion model proved to be exceptionally resource-intensive. The addition of one extra layer and the implementation of double convolution significantly increased its complexity, rendering it unmanageable compared to other models. Additionally, its inference times are substantially longer, more than ten times that of other models, as the process requires multiple passes through the data. Training one full epoch took about 17 hours, utilizing a batch size of only 6, whereas a batch size of 16 was optimal for the other models. Contrary to the Autoencoder, the diffusion model did not show promising results even after one full epoch of training, indicating that its setup might require significant adjustments.

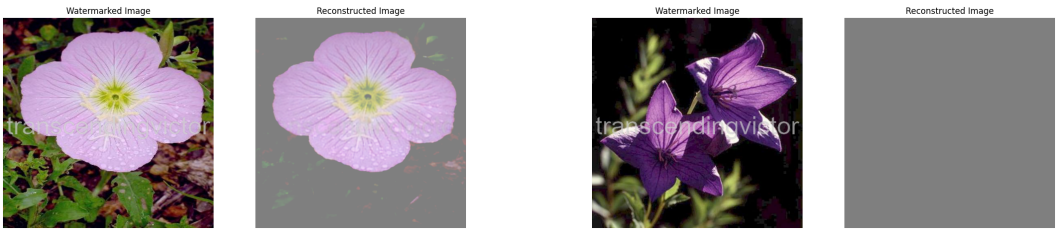
3.4 VAE Results

Initial experiments incorporating both reconstruction and KL divergence losses in the VAE did not yield promising results. The model consistently outputted grey values across all channels and pixels in every image, essentially producing no variation and failing to capture any meaningful details from the training data. Consequently, I adjusted the approach by training the model for 27 epochs solely with the reconstruction loss, effectively omitting the KL loss. This modification can be seen as deviating from the fundamental principle of VAEs, as the KL divergence is critical for regularizing the latent space to approximate a standard normal distribution.

While training solely on the reconstruction loss showed some progress, the learning process was markedly slower than that of the baseline Autoencoder, which has fewer parameters and does not involve the complexity added by sampling. Despite some advancements, the image quality remained poor and the output felt robotic, lacking the nuanced details necessary for high-quality reconstructions. Additionally, even after 8 epochs of training—by which time the baseline Autoencoder had already demonstrated substantial improvements—the VAE struggled to replicate or even hint at the background.

Given that reintroducing the KL loss was unlikely to enhance the image reconstruction directly and improvements were still necessary, I decided to extend the training for an additional 13 (27+13=40) epochs without the KL loss. This decision was aimed at optimizing the reconstruction capabilities of the model without the additional constraints imposed by the KL divergence, focusing purely on enhancing the visual quality of the generated images.

After 40 epochs of training solely with reconstruction loss, it became evident that this approach was insufficient for accurately capturing the background color, often failing to match the colors of the flowers and lacking detail in the generated images. However, there were some exceptions where light objects in certain images (such as pair 273 and pair 4) showed interesting and potentially useful results. Given these mixed outcomes, I decided to reintroduce the KL divergence loss, albeit weighted significantly lower—1000,000 times less—due to the stark disparity in the magnitudes of the corresponding losses; the KL was approximately 5×10^7 while the reconstruction loss was about 7×10^{-2}. Starting from the checkpoint at epoch 39, which exhibited the lowest reconstruction loss, this adjustment aimed to refine the generative model's performance. Unfortunately, within the first epoch of reintroducing the KL loss, it became clear that this change had undermined all prior progress, as the model reverted to outputting uniformly grey images. Despite extending the training for an additional 20 epochs, there was no improvement; the output remained consistently grey, showing no signs of recovery or change.



(a) VAE, epoch 39, Image 2.

(b) VAE with KL, epoch 31 (and any other), Image 132.

Figure 4: Outputs of the VAE varying in accuracy depending on whether the KL Loss term is used.

4 Conclusions

This project has explored the capabilities and limitations of various generative models, including Convolutional Autoencoders (CAE), Generative Adversarial Networks (GANs), Variational AutoEncoders (VAEs), and diffusion models, in the context of image watermark removal. Each model was evaluated based on its ability to reconstruct images while removing watermarks, assessed through metrics such Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR).

The CAE demonstrated the highest fidelity in image reconstruction among the models tested, showcasing its effectiveness in handling detailed textures and color consistency, albeit struggling slightly with lighter watermarks. This suggests its utility in applications where high detail retention is crucial. In contrast, the GAN approach, particularly with its later iterations focusing on balancing discriminator and generator losses, showed potential in completely removing watermarks but at the cost of sometimes losing finer image details, highlighting the challenges in training stability and model tuning. The combination of the Convolutional Autoencoder architecture (which the generator also used) has proven to showcase very good performance without the need of including any fancy training technique or architecture for the task of watermark removal. Despite its loss of detail caused by the adversarial training dynamics with the discriminator, the GAN is the only methods that systematically removed always the watermark and by controlling the training dynamics, it also managed to obtain high fidelity with the original images.

The VAE, while an innovative approach with its probabilistic representation of image data, encountered difficulties in both color fidelity and detail preservation. Experiments indicate that the model's performance was completely nullified by the presence of the KL Divergence loss. As such, the model just becomes a complicated convolutional autoencoder which, by representing data in a limited probabilistic latent spaces, doesn't manage to reconstruct the image. Still, more confident would lead to better performance and to better suiting the original images, but they trained slower than the already working CAE.

Diffusion models, despite their theoretical promise for generating high-quality images through a controlled noise introduction and reduction process, proved to be the most resource-intensive with minimal success in the initial watermark removal tasks. The extensive training time and computational resources required, coupled with their little hint of learning in the first epoch, suggesting that their use may be more suited to scenarios where computational resources are less of a limiting factor.

Overall, the project underscores the importance of model selection based on specific image processing needs and highlights the trade-offs between image fidelity, computational efficiency, and training stability.

5 Reflections

This project offered numerous learning opportunities and insights, particularly regarding the effectiveness of various image processing models and the significant role of resource allocation in machine learning projects. One of the primary challenges encountered was the implementation of the diffusion model, which due to its complexity and resource demands, prevented a thorough evaluation of its potential for treating watermarks as noise. The architecture, particularly the use of a heavy U-Net, was overly ambitious given the available computational resources. Reflecting on this, a more pragmatic approach could involve experimenting with a smaller, potentially simpler architecture that does not necessarily rely on a U-Net. This adjustment could provide more concrete conclusions about the model's capabilities and efficiency, emphasizing the importance of aligning model complexity with project resources.

Additionally, my experiments with GANs revealed a significant learning curve. Initially, I experimented with various strategies which, while educational, were not directly focused on optimizing for higher reconstruction losses combined with moderate efforts to fool the discriminator. In hindsight, a more streamlined approach, directly aiming to balance these aspects from the beginning, could have led to quicker and more effective improvements in model performance. This experience underlines the importance of strategic planning in model training and the potential benefits of applying successful strategies more decisively.

In contrast to current large-scale diffusion models, which, when tasked with removing watermarks or recreating parts of an image, may sometimes alter the image significantly while maintaining high resolution, our models were specifically optimized to reconstruct the original image through supervised training paired with their watermarked counterparts. Despite this optimization towards accurate reconstruction, our models still exhibited a significant loss of detail. This issue underscores the inherent challenge in unwatermarking, where high accuracy in the unaltered regions of the image is crucial for maintaining fidelity. The concern about reconstruction fidelity raises questions about the aesthetic appeal of the output and whether these models are effectively appealing enough to justify their use. Among our approaches, GANs are at the threshold of being sufficiently effective and practical for real-world applications. This analysis suggests that while the ability to recreate unwatermarked images without losing detail remains a challenge, GANs show potential for practical use, balancing the trade-offs between detail retention and the removal of undesired features.

Yet, the specific domain focus on flowers and the consistent use of a particular watermark throughout likely limits the practical applicability of the models in more varied real-world scenarios making this models impractial. The initial decision to narrow the scope was driven by concerns about the ethical implications and potential risks associated with unwatermarking technology, and what it represents. Nevertheless, exploring adversarial learning within this controlled framework has proven to be a fascinating and instructive approach, capitalizing on the dynamics of competition between agents. Engaging with these models has not only enhanced our understanding of generative and adversarial processes but also highlighted the complexities and challenges inherent in machine learning applications. The project was intriguing overall, providing a rich ground for further exploration and development in the field of image processing and beyond.

References

[1] Felice Pollano. Watermarked vs non-watermarked images. https://www.kaggle.com/datasets/felicepollano/watermarked-not-watermarked-images, 2023. Accessed: 2024-05-05.

[2] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, pages 722–729. ACM, 2008.

[3] OpenAI. Dall·e 2: A new ai system for creating images from text. https://openai.com/blog/dall-e-2/, 2022. Accessed: 2024-05-05.

[4] Stability AI. Stable diffusion: A latent text-to-image diffusion model. https://stability.ai/blog/stable-diffusion-public-release, 2022. Accessed: 2024-05-05.

[5] Channel or Author Name. I built an ai that destroys watermarks. YouTube, 2021. https://www.youtube.com/watch?v=0q3FyfgawSc.

[6] Lomia George. Watermark removal with deep learning. Medium, 2021. https://lomiageorge.medium.com/watermark-removal-with-deep-learning-5a93ce7de2af.

[7] Sergio Naval Marimont, Matthew Baugh, Vasilis Siomos, Christos Tzelepis, Bernhard Kainz, and Giacomo Tarroni. Disyre: Diffusion-inspired synthetic restoration for unsupervised anomaly detection, 2024.

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[9] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[10] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[13] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[14] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[15] OpenAI. Sora: Openai's text-to-video model. https://openai.com/blog/sora, 2024. Accessed: 2024-05-05.

[16] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. *arXiv preprint arXiv:2009.00713*, 2020.