University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

A deep learning approach to artefact correction for photographic film

Daniela Ivanova

March 27, 2020

Abstract

This project demonstrates the viability of using a neural network framework to identify reliably and repair film damage artefacts from analogue media scans using deep learning, thereby removing the need for hand-annotation.

The project investigated adapting a state-of-the-art perceptual loss-based method used for colourisation and superresolution to the task of analogue artefact removal. The perceptual loss based on feature activations from a VGG16 loss network was modified by introducing a new loss derived the Structual Similarity (SSIM) perceptual metric; it was utilised to compare image predictions versus targets, as well as feature activations extracted from several hidden layers of the loss network.

A U-Net restoration network with a pre-trained ResNet34 encoder was developed and trained with a baseline per-pixel loss, as well as with several perceptual losses, including one with the novel SSIM modification. Upon evaluation, the modified perceptual loss was shown to improve restoration quality, and minimise the introduction of new damage by the network.

Finally, experiments in the project suggest that transfer learning and feature extraction using models pre-trained on natural image data such as ImageNet can be leveraged to quickly train a restoration network to repair artefacts that are uncommon for natural images.

Acknowledgements

I would like to thank my supervisor, Dr Paul Siebert, for his invaluable knowledge, support and patience. I would also like to express my gratitude to the team behind Fast.ai, for collecting, curating and creating some of the most incredible techniques in the deep learning field, and sharing them in an accessible way.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Daniela Ivanova Date: 27 March 2020

Contents

1 Introduction

This chapter describes the general issues faced with current methods of analogue film restoration, and outlines why deep learning can provide a solution to these shortcomings.

1.1 Motivation

With the advent of digital photography, many thought that film was destined to become obsolete. However, as of 2020, film (that includes motion picture stock, as well as 35mm and 120 photographic film emulsions) is still being produced by companies such as Kodak and Fuji - in fact, Kodak's film revenue grew by 21% in the third quarter of 2019, according to the company's Q3 2019 earnings press release [1].

There is a strong case that can be made in favour of shooting on film and digitising after versus shooting digital even in 2020. This is because film grain follows a pseudo-random pattern, whereas digital images comprise of pixels arranged in a perfect grid. There is evidence to suggest pseudo-random sampling is better suited for reliable image reconstruction (Fannjiang 2011; Herrmann 2010). In practice, this means that digitising, or sampling, analogue images to a higher resolution such as 8K or 16K would yield better results than upsampling digital content originally recorded in 4K to 8K.

Even if film demand and usage were to cease completely, which is evidently not the case, there would still be an immense amount of information in the form of still photography and motion pictures already recorded via the medium since it was first used in the 19th century. Digitisation, whether in order to make content more accessible or to preserve it, is a mandatory part of an analogue record's life cycle. While analogue photography itself might not be a thing of the past yet, the process of manual film restoration certainly is - however, deep learning has the potential to change that.

Film is a physical, perishable medium, which is prone to degradation over time. Emulsion can further be subject to deterioration due to improper handling or storage - for instance, when artefacts such as dust or scratches are introduced, or when colour fades. When analogue film is digitised, such artefacts introduced by the medium are translated into the digital domain as well.

Film restoration can be framed in several different ways depending on the types of missing information that is to be restored. In the case of dust and scratches, film restoration involves *artefact removal*, i.e. in-painting, which in its own turn is dependent on accurate *artefact detection*. If artefact removal is to be performed with acceptable results, human input and perception is required to detect the image regions where defects are present and in-paint them.

The process can be incredibly labour-intensive, time-consuming and expensive. Fully automating the digital restoration of film would lower both costs and the number of human hours currently spent on the tasks of artefact removal and colourisation. This could allow for more pieces of valuable analogue image media to be digitally restored and made accessible to wider audiences.

[1] See full report: *Kodak Reports Third-Quarter Revenue of $315 Million and Growth in Key Product Areas.*

2 Background

The following chapter will discuss existing methods for analogue artefact removal, describe core deep learning concepts for vision and review relevant state-of-the-art approaches based on these concepts for relevant image restoration tasks.

2.1 Existing methods

In the context of image processing, *restoration* is a rather general term. It can involve in-painting, restoring colour information, or missing fine detail. artefact removal (physical artefacts on film, artefacts generated through JPEG compression, or other) is one type of image restoration.

2.1.1 artefact detection and removal

The task of artefact removal can be broken down into two sub-tasks: identifying the damaged regions, and restoring the missing information. The problem with identifying whether dust and scratches are present in an image in an automated way is that they are also random - their size, shape and intensity varies.

While automated tools such as Digital ICE (Image Correction and Enhancement)[1] do exist, they can be limiting in several ways. Digital ICE and its similar technologies use long wave infrared light to localise dust particles when scanning a film slide. As the infrared light will be able to pass through the slide but not through the dust, a dust mask can be produced, which is then used for in-painting the affected regions through methods such as bicubic interpolation. However, this approach is not applicable to black and white film, as silver halide reflects infrared light, or to non-transparent mediums, such as prints. The infrared light can be also absorbed by the cyan layer of some color emulsions, such as Kodachrome, decreasing the method's efficiency. It is also possible for Digital ICE to falsely detect grain instead of dust, therefore it requires careful manual hand-tuning. Lastly, the availability of hardware which supports infrared scanning, as well as access to the original slides so that such scanning can be performed, pose an additional limitation.

Once film damage is detected via Digital ICE or other such techniques, the missing data is restored via various in-painting methods - these can be both manual and automated. Automated approaches most often rely on traditional image processing based on statistics, or more recently, turn to machine learning to in-paint the missing image data in the affected regions.

2.1.2 Deep learning for image restoration

After conducting preliminary research on analogue film restoration via machine learning (and more specifically, *deep learning*), it became clear that there is emphasis on interpreting "restoration" as colourisation or denoising, rather than artefact removal. In fact, some of the most exciting state-of-the-art deep learning approaches have captured the attention of both researchers and the general public through demonstrating compelling results in the task of colourisation of old black-and-white photographs (Zhang et al. 2016; 2017). The successful application of such

[1] See the article *Digital ICE: Defect Detection and Correction Using Infrared-enabled Scanners* for detailed explanation.

techniques in similar contexts within image processing, such as in-painting (Mao et al. 2016; Ulyanov et al. 2018), denoising (Mao et al. 2016; Ulyanov et al. 2018; Meng Tang et al. 2018), superresolution (Mao et al. 2016; Ulyanov et al. 2018; Ledig et al. 2017), presents deep learning as a potential solution to the otherwise laborious task of dust and scratch removal too. Some commercial solutions have already employed a combination of deep learning and traditional signal processing to tackle the specific task of artefact removal [2], however, peer-reviewed research on this specific case of denoising is scarce.

An important constraint for any restoration is that non-corrupted image features need to be preserved. In the case of artefact removal, this requires the accurate identification of corrupted regions, i.e. areas of the image where dust and scratches are present. In other words, it is a *blind* reconstruction task. Still, the task is similar enough to colourisation and superresolution to justify the application of some of the advanced deep learning techniques used for those tasks such as transfer learning and progressive resizing, perceptual loss functions or adversarial training.

2.2 Deep learning

Deep learning is a subset of the machine learning field which takes inspiration from biology to create layered models with *neurons* at each layer, where all layers besides the input and the output ones are referred to as *hidden* layers, i.e. *deep* neural networks (DNN). The term neuron in this context refers to the fundamental building block of a network, representing a mathematical function that receives one or more values as inputs, which are multiplied by values called *weights* and summed together with another type of learnable parameter called *bias*.

A non-linear function, also known as an activation function, is applied to produce the neuron's output. Neurons with similar purpose are grouped into layers; a neuron which belongs to a given layer will pass its output as input to the neurons it is connected to from the next hidden layer, or simply output the result value if it belongs to the last layer.

The learning process consists of gradually updating the weights and biases for a network such that a high-dimensional non-convex function of the network's output, known as the loss function, is minimised. The gradient of the loss function with respect to the weights of the network is computed using backpropagation (Rumelhart et al. 1986; LeCun et al. 1989); it is then utilised within an optimisation method, traditionally Stochastic Gradient Descent, or more recently, *adaptive* gradient descent approaches, such as Adam (Kingma and Ba 2014).

2.2.1 Convolutional Neural Networks

Examples of deep neural networks include Recurrent Neural Networks (RNN), Deep Belief Networks (DBN) and Convolutional Neural Networks (CNN). CNNs traditionally comprise of sequences of convolutional operations, pooling operations and activation functions, along with simple fully connected layers.

The rationale for the usage of CNNs for vision tasks can be traced back to a paper by Hubel and Wiesel (1959), in which the authors demonstrated that certain neurons found in cats' primary visual cortex responded to stimuli in specific areas of the visual sensory space. In a following study on macaque and spider monkeys, Hubel and Wiesel (1968) formalised a system of three types of visual cells, characterised by the type of stimulus that excites them:

- simple cells: a response is elicited through light patterns forming straight edges with particular orientation being present in the respective receptive fields;
- complex cells: these cells have larger receptive fields and respond to patterns with a certain orientation, their motion and its direction, while being invariant to location;

[2] See *Neural Network Learns to Remove Scratches from Still and Moving Images.*

- hypercomplex cells: along with the stimuli that excite complex cells, these cells also respond to the length of the light pattern.

The authors state that some cells are responsive to colour, and additionally suggest composing different types of visual cells into a hierarchical model to be used in pattern recognition tasks. The intuition for this is that image features adhere to a hierarchical structure: cats have ears, which have fur texture, which in turn is made up of edges.

These concepts are reflected in the Convolutional Neural Networks described by Fukushima (1980), where two types of layers are defined - convolutional and downsampling. The input of each neuron in a convolutional layer has a receptive field mapping to only a subset of neurons in the previous layer; if the given layer is the input layer, the receptive field corresponds to the image matrix representing the pixels in an image. The kernel of each convolution corresponds to a filter which is applied to produce a convolved feature map, or in other words, extract specific features. Each matrix element in a given filter form the set of the weights that need to be learned. As a defining feature of CNNs, weights are be shared between neurons to reduce the number of learnable parameters. The downsampling layers are applied to the convolved feature maps to reduce their spatial size, thereby further reducing the input for the next convolutional layer. Traditionally, max pooling is used for this purpose. In the resulting layered system, receptive fields of earlier layers will consist of smaller, localised regions of the input image and the neurons in those layers will respond to low-level features such as edges, while later layers will have more global receptive fields and will in turn act as filters for more complex features (Zeiler and Fergus 2014). This removes the need for structured training data, as features for both low and high order are extracted via the model's layered architecture. The resulting network is remarkably similar to the observations made by Hubel and Wiesel (1959; 1968). By closely mimicking the visual systems of living organisms, CNNs have become the primary architectural choice for deep neural networks regarding visual tasks, and are therefore the most relevant to the aims of this project.

2.2.2 Non-linear activation functions

Introducing non-linearity via activation functions applied to the convolved features allows for more complex tasks to be solved by the model compared to only using linear operations. In fact, Cybenko (1989) and Funahashi (1989) provided evidence that any continuous function can be approximated by a single *hidden* layer of sigmoidal units, where the size of the layer tends to infinity. These findings are at the core of what is known as "The Universal Approximation Theorem". Hornik (1991) demonstrated that the property of universal approximation is invariant to the choice of specific activation function, and related to the layered architecture of neural networks with hidden layers.

One of the most (if not *the* most) commonly used activation functions nowadays is the Rectified Linear Unit (ReLU) (Ramachandran et al. 2017). The rectifier function is an activation function which transforms negative values to zero. It is defined by the equation:

$$f(x) = \max(0, x), \tag{2.1}$$

where x is the input to a neuron in a ReLU layer. A paper by Glorot et al. (2011) demonstrated ReLU increased network performance and efficiency compared to earlier activation functions such as Tanh or Softplus. One of the identified advantages is that this function produces sparse outputs, since for negative inputs the ReLU neuron will not activate at all - similar to how real-life neurons are not fully activated all the time for any given input, but are rather specialised. Intuitively, sparse networks are also faster than dense ones, as the evidence from Glorot et al. (2011) shows. However, in some instances, certain ReLU neurons can suffer from what is known as the "dying ReLU" problem - this happens when a neuron outputs only 0, or in other words, never gets activated. That, in turn, causes the gradient to go towards 0, therefore the weights never get updated during backpropagation, and the neuron becomes stuck in the non-activated

state. One popular way to mitigate this issue is to use modified versions of ReLU, such as Leaky ReLU, which has a non-zero gradient over its entire domain (Maas et al. 2013).

2.2.3 Normalisation

Normalisation in a general context is the process of subtracting the mean and dividing by the standard deviation. Within deep learning, this can refer to several concepts.

Feature normalisation In machine learning, it is common practice to normalise input data by centering it around the origin along every dimension by subtracting the mean. As an additional step, dividing by the standard deviation to scale to unit variance can also be performed. In the case of computer vision tasks, input data is most commonly formatted as single-channel (grayscale) or 3-channel (RGB) image arrays, where pixel values in in a channel range between 0 and 255. The process of mean subtraction is therefore performed per-channel in the case of multiple channels. Centering and scaling the input image data allows the network to learn quicker and be more stable since gradients act uniformly for each channel.

Batch normalisation Even when the input to the network is normalised via pre-processing, the distribution of the input to an arbitrary hidden layer will change with respect to the distribution of the parameters of the preceding layer - this will cause small adjustments in earlier layers to be amplified abnormally in later layers, making the network less stable and more difficult to train. This issue, known as internal covariate shift, is tackled by a method called batch normalisation, introduced by Ioffe and Szegedy (2015). Batch normalisation centers and normalises the activations of a preceding layer by subtracting the batch mean and dividing by the batch standard deviation, before passing them as input to the following layer. A more recent study by Santurkar et al. (2018) argues that the success of batch normalisation has less to do with minimising the internal covariate shift, and more to do with the smoothing effect it has on the optimisation landscape. Nevertheless, the approach has been widely adopted newer architectures. Ioffe and Szegedy (2015) identify some additional benefits to batch normalisation, such as minimising the chances of vanishing and exploding gradients - for instance, since ReLU has no upper bound on the activations it will output (see 2.1), the values are prone to exploding as multiplications accumulate between ReLU layers. As another rather helpful side effect, batch normalization introduces noise (since it is applied per batch) and can therefore be used instead of empirical regularisation methods such as dropout (Srivastava et al. 2014) to prevent overfitting.

2.2.4 VGG

VGG is a classic CNN architecture introduced by (Simonyan and Zisserman 2014). In 2014, VGG presented some interesting improvements over its contemporaries such as AlexNet (Krizhevsky et al. 2012): the convolutional layers in VGG use a smaller receptive field of 3 by 3, compared to AlexNet which has 11 by 11 size for its convolution kernels. The convolutional layers' stride is fixed to 1 by 1 so that spatial resolution is preserved, compared to AlexNet which uses 4 by 4 strides. Dowsampling is performed via maximum pooling operations every few convolutions.

2.2.5 ResNet

There is a trend of depth-increase for recent CNNs, as demonstrated state-of-the-art deep learning architectures, for instance AlexNet with 5 convolutional layers (Krizhevsky et al. 2012), VGG with 19 convolutional layers (Simonyan and Zisserman 2014), or GoogleNet with 22 convolutional layers (Szegedy et al. 2015). Clearly, additional layers give the network more expressive power which allow it to learn more complex functions or distributions, but it also introduces more parameters to be trained.

A paper by He et al. (2016*a*) empirically demonstrates that the training error of a 56-layer "plain" convolutional network with batch normalisation layers is higher than similar network but with

just 20 layers. This is odd, since the deeper network could be derived from a shallower one through the addition of layers; therefore, the deeper network should perform at least as well as the shallower one. One interpretation of this phenomenon is that due to the curse of dimensionality, deeper networks are not able to learn simple functions such as the identity function, which is what the theoretical additional layers in the example would have to learn in order for the modified deeper network to perform just as well as before the new layers were added.

To combat this issue, He et al. (2016*a*) propose an addition to the CNN building block toolkit called the *residual block*, illustrated in Figure 2.1. As we can see, residual learning involves learning the difference (residual) from x to $F(x) + x$, instead of the mapping $F(x)$ of x to y - see Figure 2.1. In addition to this, downsampling is achieved via 1 by 1 convolutions instead of maximum pooling as in VGG. A following study reports improved performance using this approach for a CNN with 1001 layers (He et al. 2016*b*).

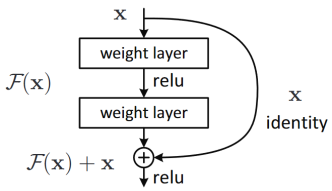


Figure 2.1: *"Residual learning: a building block." Adapted from He et al. (2016a). The proposed structure stacks convolution operations and ReLU's, similar to a traditional CNN. However, notice than besides the sequential pass through the layers, there is a "skip connection" mapping the identity transform.*

ResNets also have their biological analogue - for instance, neurons from cortical layer VI of the brain receive input from layer I, and skip intermediary layers (Thomson 2010).

A similar approach is to concatenate the input for a block of convolutional layers with its output, instead of adding them, resulting in each layer receiving the concatenated outputs of all preceding layers. Concatenating feature maps is done to encourage feature reuse (Huang et al. 2017). This results in a densely connected network, called DenseNet.

2.2.6 U-Net

Restoration is an image-to-image task, meaning that the output should also be an image, traditionally of the same pixel size, rather than a feature vector a classification label. Image segmentation is also an image-to-image task that aims to produce a segmentation mask which partitions an input image into clusters describing objects (or instances of objects) present in it. Therefore, in addition to using a CNN to transform an image into a feature vector representation, the information from the vector needs to be transformed back into an image - a segmentation mask, or in the case of this project, a restored image.

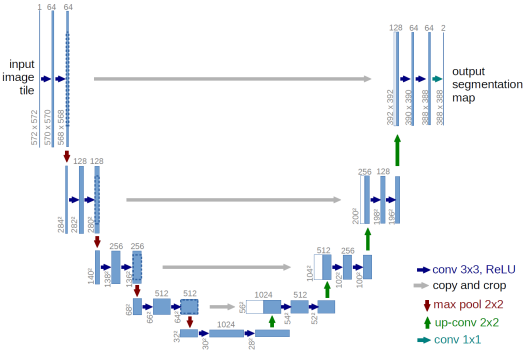


Figure 2.2: *"U-net architecture (example for 32x32 pixels in the lowest resolution)." Adapted from Ronneberger et al. (2015). The downsampling part of the network (on the left) resembles a traditional CNN. For each upsampling stage, feature maps from the corresponding downsampling stage are concatenated to the input - similar to DenseNet.*

U-Net is an architecture which was developed for the specific task of biomedical image segmentation (Ronneberger et al. 2015). This is an area where accuracy and detail of the segmented regions is crucial. The U-Net architecture is illustrated in Figure 2.2 - the overall shape resembles the letter "U", hence its name.

The U-Net architecture is constructed by concatenating an encoding (downsampling) part, consisting of convolution operations that produce a feature vector representation of the input,

and a decoding part, which produces the output image by "upsampling" feature representation produced by the encoder via transposed convolution operations.

Sub-pixel convolution In order to upsample a given input, the network needs the opposite operation to convolution - i.e. *transposed convolution*. Transposed convolution adds zero padding (or white pixels) around the input pixels. An alternative to padding with zeroes would be to interpolate the additional pixels, however, a smarter approach for filling in the gaps in the upscaled image was developed by Shi, Caballero, Huszár, Totz, Aitken, Bishop, Rueckert and Wang (2016) as part of the Efficient Sub-Pixel Convolutional Neural Network (ESPCN) architecture.

Instead of initialising the padding between and around the input values as zeroes or interpolating them, the authors introduce sub-pixel convolution, which combines convolution and pixel shuffle. The idea is expanded on and formalised by Shi, Caballero, Theis, Huszar, Aitken, Ledig and Wang (2016), where the authors prove that convolution combined with pixel shuffle is equivalent to transposed convolution. At the same time, since the convolution is done on a lower spatial dimension using more channels, it also has higher representational power at the same speed; the large number of feature channels in the decoder allows for important context information to reach higher resolution layers.

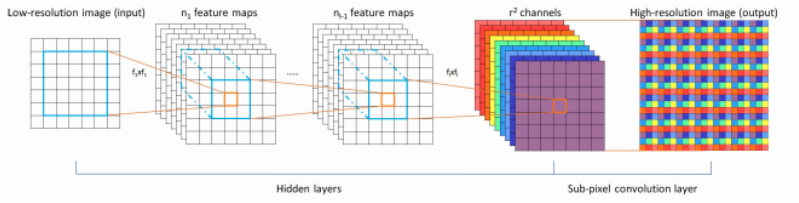


Figure 2.3: *"The proposed efficient sub-pixel convolutional neural network (ESPCN), with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from LR space and builds the SR image in a single step." The combination of convolution and pixel shuffle is done at a lower spatial dimension using more channels, compared to transposed convolution. Adapted from Shi, Caballero, Huszár, Totz, Aitken, Bishop, Rueckert and Wang (2016).*

Upsampled results can suffer from checkerboard artefacts due to random initialisation of the convolution weights in sub-pixel convolution - Aitken et al. (2017) build upon the sub-pixel convolution approach by initializing the convolution weights in a way which allows the convolution plus pixel shuffle sequential operation to produce the same results as nearest neighbor interpolation, and be trained from that point. Adding an average pooling operation after each sub-pixel convolution offers additional help against checkerboard artefacts (Sugawara et al. 2018).

Skip connections To further improve the fine detail of the upsampled image, long skip connections can be added to the U-shaped encoder-decoder setup. Those are illustrated as gray lines in Figure 2.2. Rather than skipping short distances, i.e. a couple of layers like in a ResNet, the outputs from each downsampling stage "skip" a section of the middle of the network. They are concatenated to the inputs for the corresponding upsampling stage, rather than added, similar to a DenseNet. This helps to recover spatial information lost during downsampling (Ronneberger et al. 2015). Additionally, Drozdzal et al. (2016) build a strong case in favour of combining short and long skip connections in the same model.

Self-attention Generative adversarial networks (GANs) have become the primary choice for image generation tasks where high degree of realism and detail is important as they can successfully model low-level features such as texture patterns - a more detailed review of the fascinating background of GANs can be found in Appendix E. However, they face difficulty in translating structural and geometric features, i.e. features which reflect global dependencies and patterns. This results in artefacts and blurriness found in examples generated via standard GANs. This can

be explained by the fact that convolutional networks are proficient at capturing local dependencies, but increasing the size of the convolution kernel or adding more layers with progressively larger receceptive fields to remedy that is not viable in terms of computational resources. Self-Attention Generative Adversarial Networks, or SAGANs, proposed by Zhang et al. (2018), address this by extending the generator and discriminator networks with a self-attention layer, inspired by natural language processing (Cheng et al. 2016; Vaswani et al. 2017), via which long range dependencies are taken into account. While self-attention was developed in the context of GANs, it can be independently utilised in conventionally trained generators too.

2.2.7 Optimisation

Optimisation algorithms are responsible for minimising the objective (loss) function of a given network through updating the network's internal parameters (the weights and biases). The choice of optimiser is a crucial step in the modern neural network training pipeline, and the attention in research has long shifted from the standard stochastic gradient descent algorithm. Adaptive Moment Estimation (Adam) has been at the forefront for several years, after the algorithm was first published by Kingma and Ba (2014) and experimentally evaluated to outperform other optimisers. While SGD maintains a constant learning rate for all learnable parameters, Adam computes different learning rates for different parameters. It uses the squared gradients (Hessian) to scale the learning rate, and additionally utilises momentum by using moving average of the gradient of the loss function (rather than the gradient itself, as used by SGD). In a comprehensive review of modern optimisation algorithms, Ruder (2016) states that Adam "might be the best overall choice" of optimiser. Loshchilov and Hutter (2017) point out that when Adam is used, L2 regularisation and weight decay are no longer equivalent as is the case with SGD, and propose an empirically tested method of "weight decay decoupling", which is implemented in practice in libraries such as PyTorch and Tensforflow.

2.2.8 Cyclical Learning Rates

Finding the optimal learning rate is another important aspect of neural network training. Too small of a learning rate will cause the loss to converge slowly, whereas too high of a learning rate will cause it to diverge. Adaptive learning rate optimisers such as Adam (Kingma and Ba 2014) remedy that by annealing the learning rate over time. However, as Adam relies the squared gradient to update the learning rate, it can be expensive to compute (Ruder 2016).

Another approach of gradually changing the learning rate was developed by Smith (2017). Instead of simply decreasing the learning rate, the proposed approach oscillates the learning rate values between a minimum and a maximum over a cycle. A cycle consists of increasing the learning rate over a chosen number of iterations, then decreasing it over the same number number of iterations. While this can hinder performance for a while, the optimum learning rate will eventually be found within the given bounds, resulting in long-term improvement (Smith 2017). In a follow-up study, Smith and Topin (2019) describe a phenomenon called "super-convergence" that allows networks to be trained much quicker, which is attributed to a method named One Cycle Policy. One Cycle Policy consists of a single cyclical learning rate phase (with two stages, increasing then decreasing the learning rate) with a very large higher learning rate limit. After the cycle is complete, the learning rate is decayed further.

2.2.9 Transfer learning

Transfer learning has captured research attention as it allows for the reuse of knowledge obtained from models trained using readily available data along with quick and easy to calculate loss functions. This is done by transferring the learned weights and fine-tuning them for tasks where training data is of limited availability, or where the loss function is more complicated. Pre-trained

weights on data sets such as ImageNet (Deng et al. 2009), MNIST (LeCun n.d.), COCO (Lin et al. 2014), for popular architectures such as ResNet (He et al. 2016*a*), DenseNet (Huang et al. 2017) have been made widely available online as part of various Model Zoo collections.

Kornblith et al. (2018) demonstrate that there is a strong correlation between the accuracy values of models achieved in ImageNet classification tasks and the accuracy of models which use pre-trained ImageNet weights for a different, more specific task - 0.960 and 0.990, respectively.

Studer et al. (2019) examine the viability of transfer learning using ImageNet for historical documents character recognition, style classification, manuscript dating, semantic segmentation, and content-based retrieval. Historical documents, similar to scanned analogue photographs or motion picture film, have different image properties from the examples in ImageNet. The study evaluated different architectures and found that ResNet152 pre-trained on ImageNet benefits from a 9.5% increase in accuracy for style classification, and a 17.3% increase in accuracy for manuscript dating, compared to conventional training. These experiments give evidence for the benefit of using transfer learning for cross-domain tasks, even if the target domain data has different statistics from the data in the source domain.

2.2.10 Progressive resizing

Progressive resizing is another state-of-the-art method which was developed to speed up network training. It consists of gradually increasing the input size to guide the network into learning finer detail over time; correspondingly, as input size increases, more layers are appended to the network incrementally. It was first presented in the context of residual networks by Lim et al. (2017) and in the context of GANs by Karras et al. (2017), where high-quality images of human faces are generated in the impressive spatial resolution of 1024 by 1024 pixels.

2.2.11 Mixed Precision Training

Traditionally, weights, activations and gradients in deep learning are represented using the single-precision floating-point format, which occupies 32-bits of memory. As the number of network parameters increases, the demand for memory resources required to train the network also grows. This problem is addressed by Micikevicius et al. (2017) through a technique called mixed precision training. Mixed precision training includes training in half-precision, i.e. using 16-bit floating point numbers instead. However, to avoid loss of information and imprecise weight updates, weights are updated in single-precision; furthermore, to combat the possibility of gradient underflow in 16-bit, the gradients of the loss function are scaled so as to avoid falling out of the range that is representable in half-precision. This technique could theoretically see the memory usage demands of training larger networks reduced in half, and the authors of the paper demonstrate empirically reduced memory consumption for a variety of deep networks, including CNNs.

2.2.12 Loss functions

An objective function in machine learning in general and in deep learning in particular is a function of the network's output and the expected outcome. It measures how good the model's prediction is with respects to the expected outcome. During optimisation, the network is trained with the goal to find parameters (weights and biases) for it such that the objective function is minimised or maximised, depending on its formulation. When the goal is to minimise the objective function, it is often referred to as a *loss* function.

The aim of image restoration is to predict a version of damaged image in the undamaged image domain. Therefore, the loss function for models which try to solve this task can be expressed as a

distance measure between the restored image (network output) and an image where no damage is present.

Per-pixel loss functions The simplest way to compare two images is to calculate their pixel-wise distance, and then average it over the number of pixels. In the context of neural network training, the loss function based on this principle is called Mean absolute error (MAE). MAE calculates the absolute difference between the network output and the true observation for each item in the training set, and averages that over the number of training examples. A similar error measure is the Mean squared error (MSE), which measures the average of the squares of the differences between the model prediction and the corresponding true observation for each item in the training set.

As both MSE and MAE are average per-pixel losses, they tend to produce blurry outputs, as this is the easiest solution that would minimise the average pixel difference - however, while achieving a low error rate, such results would not be perceived to be of high quality by humans due to the lack of detail. For this reason, state-of-the-art loss functions used in problems such as style transfer or superresolution that achieved higher *perceptual quality* are reviewed in this chapter.

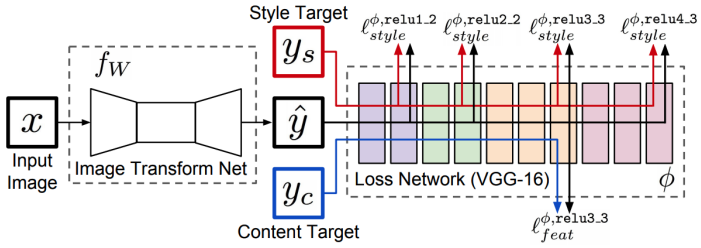


Figure 2.4: *"(...) We use a loss network pretrained for image classification to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process." In the paper the authors used VGG16 as their loss network, however, other architectures could be adapted using the concept. Adapted from Johnson et al. (2016).*

Perceptual loss Perceptual loss is a concept developed by Johnson et al. (2016), based on earlier research in style transfer (Gatys et al. 2015), in which two types of perceptual losses that measure high-level perceptual and semantic differences between images, rather than pixel distance, are defined: *feature reconstruction loss* and *style reconstruction loss*. Both perceptual losses depend on the activations of a pre-trained classification network, as seen in Figure 2.4. The key idea is that a pre-trained classification network has already learned to encode feature representations of higher level semantic, style and content features in images. The authors use the activations from the ReLU layers of the loss network to compare the outputs from the network that is being trained to the true observations. This method was evaluated for the tasks of style transfer and superresolution, for both of which improved perceptual quality of the output as well as reduced training times were reported. The idea of VGG-based perceptual losses for superresolution was used in conjunction with another network-based perceptual loss, i.e. adversarial loss, by Ledig et al. (2017) who also reported improved structural similarity scores compared to per-pixel losses. Despite DeOldify's results not being scientifically evaluated yet, Antic (2020) provides qualitative evidence in support of the usage of adversarial training in addition to perceptual losses.

Erhan et al. (2009) provide a qualitative approach to interpreting the inner layers of CNNs similar to style transfer. A random noise image is forwarded through the network in evaluation mode, and the average activation of the filter from a specific layer that we are interested in. The gradients w.r.t. the input image's pixel values are then computed and used to update the input image to a pattern which maximises the mean activation value of the selected filter.

2.2.13 Evaluation metrics and approaches

Evaluating a network involves calculating metrics which describe its performance. One of the easiest way to do that is to look at the network's training history – at the end of each training epoch, the loss value achieved on the training data and the validation data, if available, is recorded. After training is completed, loss values are derived for the unseen test set to describe how well the network generalises for data it has not been trained on.

Evaluation metrics, similarly to loss functions, vary based on the task, and need to be chosen carefully so that they reflect the core of what the model aims to learn. For image tasks in particular, it makes sense to use visualisations of the inner (feature) space of the network, as well as to visually inspect results and measure their perceptual quality, in addition to calculating prediction error.

Structural similarity Structural similarity index (SSIM) is a well-regarded quality measure developed to evaluate the perceived index of quality in digital television and movies, developed by Wang et al. (2004). It assumes that pixels are related to each other in some meaningful way, especially if they are spatially close – in this way, it improves upon Mean Squared Error (MSE) or its closely related Peak Signal to Noise Ratio (PSNR), which are two popular metrics for image information loss that preceded it. SSIM is calculated over image windows of standard size 11 by 11 pixels. It compares three different aspects – luminance l, contrast c, and structure s – of two images, x and y, as follows:

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \qquad c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \qquad s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{2.2}$$

where μ_x, μ_y are the means of x and y, $\sigma_x\sigma_y$ are their respective variances, σ_{xy} is their covariance; c_1, c_2 and c_3 are constant values derived from the dynamic range of pixel values and used to stabilize division by a weak denominator. SSIM is calculated as a weighted combination of the three terms:

$$SSIM(x,y) = \left[l(x,y)^\alpha \cdot c(x,y)^\beta \cdot s(x,y)^\gamma \right]. \tag{2.3}$$

Setting the weights α, β, γ to 1 results in:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.4}$$

SSIM is widely regarded as a reliable indicator of image quality degradation (Søgaard et al. 2016; Gore and Gupta 2015; Dosselmann and Yang 2011). It compares a distortion-free version of an image to a version where degradation is present. Values of 1 indicate that no distortion is present and the two images are identical; as the degree of degradation increases, the SSIM score decreases.

SSIM as a loss function Given the success of perceptual losses for image restoration tasks, and the reliability of SSIM as a measure of image degradation, it makes sense to translate SSIM to a loss function. Lu (2019) propose a Weighted SSIM loss (LWSSIM), calculating the score between different filter sizes, then produce a weighted sum over the score for each size, and use the loss to train autoencoders (a simpler encoder-decoder architecture, similar to U-Net). This recalls multi-scale SSIM (MS-SSIM) (Dosselmann and Yang 2011), a more complex SSIM formulation.

3 Analysis & Approach

In order to analyse the problem and design an approach to solve it, we must identify the scope of said problem. This will inform the final problem statement of the project.

3.1 Scope and limitations

Due to the limitation imposed by the project's two-semester time frame, it is crucial that appropriate scope is defined. The task domain will be limited to still images as video restoration would introduce several additional requirements, which would extend this project beyond its time scope. The project scope is also directly related to the type of image restoration task we will investigate, and how we define *restoration*. Finally, the availability of resources, both hardware and data, should also be considered.

3.1.1 Problem statement

As detailed in Chapter 2, analogue film restoration can be understood as either removing damage such as dust and scratches through in-painting, or colourising for images where little or no colour information is present. Deep learning has revolutionised multiple tasks which fall under the class of image restoration, such as colourisation of black and white images and superresolution (restoring or generating missing image detail), however, far less attention has been given to artefact removal. The evident similarities of the three tasks, detailed in Section 2.1.2 provide motivation for us to employ state-of-the-at deep learning approaches from the colourisation and superresolution domains to the task of artefact removal.

At the same time, there is a fundamental difference between the information that a deep neural network would have to learn to generate for each of the three tasks. In the general case, if the degradation present in the corrupted image is linear and spatially invariant, modelling that degradation, and by extension removing it, are relatively easy. However, film artefact damage varies across a number of factors, such as shape, size and colour. Analogue artefact removal is additionally a blind image restoration problem, since the damaged regions are not explicitly separated from the rest of the image content, which needs to be preserved. In comparison, colourisation tasks commonly call for colour information to be restored across the whole input image; similarly, superresolution linearly expands the spatial resolution of the input and generates detail across the whole input image. Hence, the borrowed approaches require evaluation and modification in order to successfully be able to solve the artefact removal task.

Therefore, **the primary goal of this work is to produce a convolutional neural network approach that can reliably detect and remove dust and scratches for analogue image scans without introducing additional damage.** The network will accept the damaged image as input, and produce a restored version of it. We will employ state-of-the-art approaches such as residual neural networks, U-Net with skip connectons, transfer learning, perceptual loss functions, and as an extended goal, adversarial training to achieve these objectives.

3.1.2 Computational resource limitations

Deep learning (as well as artificial intelligence and machine learning in general) is a field which has seen great progress in the last decade; although many of the ideas involved in training deep neural networks and CNNs in particular have been explored in the literature much earlier, it was only in recent years that computational resources that could meet the demands of the network training process became available. Still, novel architectures that achieve state-of-the-art results can take several days to train on multiple instances of the latest GPUs.

Since this project relies on freely available GPUs such as the ones provided by Google Colab, the focus will instead be on utilising existing approaches that minimise training time and lower resource demands. Besides additional justification for the usage of transfer learning, the limitations on hardware resources as well as on available relevant data create the last major critical requirement of the project, which is to train the networks as quickly and efficiently as possible, with a relatively small number of training data samples. In order to observe no trade-off in the achieved results, proven techniques such as progressive resizing (Karras et al. 2017), optimisation via Adam (Kingma and Ba 2014), one cycle training policy (Smith 2017; Smith and Topin 2019), and mixed precision training (Micikevicius et al. 2017) will be incorporated into the training process. Employing these techniques should allow for more time to be spent on designing and evaluating potential architectural and loss function solutions.

Adopting the NoGAN approach will be considered only as an extended objective to this project. One justification for this decision also stems from the computational resource limitations - training a second network (the discriminator) presents an additional computational overhead, as well as an increased memory storage demand.

3.1.3 Data availability limitations

Crafting a data set to train, validate and evaluate the developed networks is the first critical requirement. An important consideration when creating the data set is the availability of data. It is only restored versions of film and photographs that are usually made widely available to the public. Similarly, producing scratch masks that indicate scratches and dust present in analogue images would be an extremely time-consuming manual process. Due to the time limitations of this project, this option has been ruled out.

One alternative involves a data set consisting of two classes of matching pairs - *dustified* and *clean* - being created for the purposes of this project. However, in order to obtain matching images between these classes, the corrupted examples will be synthetically generated by programatically applying film overlays over the clean image. There is a limitation to how truly random the overlays can be, and therefore, to how representative the synthetic *dustified* class would be of real-life analogue damage. Still, learning from paired data would potentially be a much easier task for a neural network, and is worth considering even as a bootstrapping method.

3.1.4 Objectives

Based on the limitations identified in the previous section, as well as the background research of state-of-the-art deep learning methods for image restoration, the following objectives are derived:

- Curate a data set of paired clean and damaged images for training and evaluation.
- Implement state-of-the-art approaches used for colourisation and superresolution, such as transfer learning and perceptual loss and apply them to the task of artefact removal.
- Evaluate the results and identify the relevance of the above approaches to artefact detection and removal.

- Adapt and improve these approaches to develop a novel method for the task of artefact restoration; evaluate the proposed novel method.
- As an extended goal, evaluate the NoGAN approach for the task of artefact removal.

3.2 Proposed Approach

Based on the background review of state-of-the-art approaches for similar tasks, as well as the analysis of the potential caveats of the artefact restoration task and the identified objectives, this section will provide the high-level approach taken over the course of this project.

3.2.1 Data collection

In the digital age, high quality analogue scans of images are made available in the public domain through heritage organisations around the world. One such collection which has been party digitised and released on the platform Flickr by The US National Archives is *Documerica*[1]. *Documerica* is a project funded by the US Environmental Protection Agency, in which multiple photographers were hired to capture American life as well as increasingly worrying environmental problems in the 70s. Due to the time period and the structure of the project, the photos vary in artistic style and subject matter, but at the same time are all taken on film, and therefore possess traditional film qualities such as warmer tones, grain, etc. Furthermore, since this is a publicly-funded project, the images have been digitised and released under no copyright restriction. As the collection made available on Flickr consists of thousands of images, and owing to the fact that Flickr provides an easy to use API to download them, *Documerica* was chosen as the source for image data used to construct the analogue image data set. Overall, 6232 *Documerica* images were collected using the Flickr API.

Pre-processing As discussed in Section 3.1.3, the data set to be created should be structured as a collection matching pairs of images. Each pair includes a clean version of a given image, along with a version where dust and scratches are present. To produce such pairs, artefacts are added to each clean image programatically. While the random nature of artefacts such as dust and scratches is impossible to perfectly replicate in a deterministic way, there are pre-processing steps that can be taken to ensure that the noise applied to images to create the damaged examples mimics true analogue damage.

The designed pre-processing approach aims to produce a "dustified" version for each raw image, at each three target sizes to be used during training.

Resizing and Cropping Another consideration to take into account at this stage is spatial size of the images that will be processed by the network. The highest resolution that can be processed via the 16GB GPU provided by Google Colab without affecting batch size is 256 by 256 pixels width and height. Furthermore, as the network training process will utilise progressive resizing of the input during training inspired by Karras et al. (2017), two smaller sizes, 64 by 64 and 128 by 128 pixels, are also required. Therefore, the first pre-processing step is to resize and center-crop the raw images to these target sizes. Resizing is done using inter area interpolation, so that there is minimal image quality loss.

Dustifying Adding dust and scratch overlays to digital photos is a popular way to give images a "film look". To produce believable damaged versions of the collected images, a limited set of dust and scratch overlays of this type are to be applied to each of the 6232 raw images obtained from the *Documerica* collection. One such free set of overlays has been provided online by SpoonGraphics[2]. In order to ensure that there is at least some pseudo-random variety in the set of noise applied from the limited number of overlays, the following simple approach was devised: randomly select

[1] See the digitised collection at *Flickr*.

[2] Credit: 30 Free Film Dust Textures to Add Dirty Effects to Your Work by *SpoonGraphics*.

a dust overlay, crop a random square patch from it, warp and invert it, then apply to the image. For each image, this is to be performed twice, so that the dust pattern on each image is a different random combination of two dust overlays. Finally, this step needs to be performed for each corresponding target size, i.e. 64 by 64, 128 by 128 and 256 by 256 pixels, in order to produce different patterns for the three different sizes of the same clean image, so as to prevent overfitting.

Data augmentation After collecting and processing the data from Flickr, we obtain 6232 images, each of which has a clean and a "dustified" version at several different resolutions as detailed above. Despite the diversity of the subject matter in the *Documerica* pictures, this is a fairly small data set size, especially when the goal is for the network to be able to restore any image. Therefore, we also employ a technique named data augmentation. Data augmentation is the process of applying random transforms to the data samples before they are forwarded through the network. This is one of the most important regularisation techniques when training models for an image based task. In order to teach the model to generalise well and not just memorise the training data, we change it slightly through image transforms such as light variation, cropping, warping etc. One option is to perform data augmentation as an additional pre-processing step, similar to how different resolutions of the images are created for progressive resizing. Alternatively, these transforms can be applied at random each time a batch of images is loaded from memory during training, therefore the network will process a slightly different version of each image at every training iteration.

3.2.2 Restoration network design

The restoration network aims to translate a corrupted image to a restored version of it, therefore, an encoder-decoder type of architecture is an appropriate choice for the design of such a network. A state-of-the-art architecture of this family is U-Net with skip connections. U-Net was initially developed for medical image segmentation tasks which require highly precise image output (Ronneberger et al. 2015); in the context of this project, precision is a crucial requirement for the task of artefact restoration as well.

Encoder The encoder part of the network is responsible for downsampling the input to a feature vector representation, which the decoder can use to generate the restored output. Clearly, this representation must allow for artefacts to be separated from the rest of the image, so that only the affected areas of the image are restored by the encoder. At the same time, the rest of the image features need to be encoded reliably so that useful image information such as detail is not lost during the upsampling stage. Residual connections in the encoder, such as the ones used in the ResNet architecture would ensure that both high- and low-dimensional features can be encoded, and thus preserved during decoding.

It is common for U-Nets nowadays to be enhanced by using a pre-trained network as the encoder part of the network; a popular choice for the encoder is the VGG16 architecture (Iglovikov and Shvets 2018). More recently, DeOldify incorporated a WideResNet as the encoder for the task of colourisation with great success (Antic 2020).

For this project, a ResNet is to be incorporated as the encoder part of the network due to the positive effect residual blocks have on training times and stability, as well as the improved representational power owed to skip connections (He et al. 2016*a*;*b*; Lim et al. 2017).

ResNet34 was chosen over deeper residual CNN architectures, as it has 21 million parameters, compared to 23 million and 42 million for ResNet50 and ResNet101 respectively (He et al. 2016*a*). This is taken into account due to the fact that we have to also build a corresponding decoder part of the network, so the overall number of parameters is expected to roughly double these numbers. Due to the computational resource limitations identified earlier, ResNet34 presents an acceptable trade-off between the network's expressive power and its size.

Decoder Two additional convolutional layers are added as the bottleneck of the "U" shape in order to transition to the decoder part of the overall network. In a U-Net setup, once the encoder has downsampled the input, the decoder has the task of translating the resulting feature vector into a restored version of the image. In other words, it needs to in-paint the areas where the decoder has detected the presence of dust and scratches. In order to do that, the decoder needs to learn to leverage the features encoded by the encoder to believably generate image data of high perceptual quality - that is, the in-painted regions should be indistinguishably blended with the rest of the image, and unnatural artefacts should not be introduced. This can be achieved through using novel upsampling approaches such as sub-pixel convolution (Shi, Caballero, Huszár, Totz, Aitken, Bishop, Rueckert and Wang 2016; Shi, Caballero, Theis, Huszar, Aitken, Ledig and Wang 2016). To preserve long range dependencies in the input image and minimise checkerboard artefacts that can be introduced during the upsampling process, self-attention (Zhang et al. 2018) is added to the second upsampling block of the decoder, counting from the bottleneck. The input image itself is also concatenated with a dense cross-connection to the input of the last upsampling block in the decoder. Finally, sigmoid range activation is applied to produce a 3-channel image with the same spatial size as the input.

3.2.3 Loss function engineering

To train the resulting restoration network, a loss function is needed. Candidate loss functions reviewed during the research stage of the project and detailed in Chapter 2 range from simple per-pixel losses, such as Mean Squared Error, to more sophisticated approaches which measure the overall perceptual difference between the network's output and the target.

Simple loss function (baseline) The baseline training process trains the restoration network using per-pixel Mean Squared Error as the loss function. Pixel-based error measures do not take into account inter-pixel dependencies and therefore images which receive low error scores could actually be of perceptually bad quality - for example images where salt and pepper noise is applied. Furthermore as they push the network towards averaging pixel values, per-pixel loss functions tend to produce blurry results.

Perceptual loss functions Instead of explicitly comparing two images in the pixel domain, another network can be used as a comparison tool. Both GANs and perceptual losses use an additional loss network (called the discriminator in the case of GAN) to express the difference between real and generated samples, which adds an additional level of abstraction to the system. While in GAN the additional network is trained simultaneously to the generator, the perceptual loss aproach uses a frozen, pre-trained network's activations to quantify the difference between output and target. Adversarial training was identified as an extended goal of this project in the previous chapter. The main advantage of this approach as applied by Antic (2020) for the task of colourisation is that it causes the generator to be more "adventurous" in its predictions in order to fool the discriminator, instead of minimising a static loss function - intuitively, a static loss would cause a colourisation model to predict average, or "safe" colours, such as brown, too often. However, the artefact removal task is different as it less unconstrained compared to colourisation: an apple could be red, yellow or green, whereas the set of possible ways to inpaint a scratch mark so that it blends with the rest of the image is more tightly constrained. Therefore, the benefit of adversarial training is less relevant to the task of analogue artefact removal; however, GAN is a highly-cited approach and are widely adopted in practice, and some additional investigation of it is detailed in Appendix E.

Feature loss network Perceptual losses were developed as a more comprehensive loss measure in style transfer and later super resolution (Gatys et al. 2015; Johnson et al. 2016; Ledig et al. 2017), where style and content are important concepts. To compare the generator's output to the target image, the two images are passed through a pre-trained network, and the activations before every downsampling step are taken. Johnson et al. (2016), as well as Antic (2020) used a pre-trained

VGG16 network, illustrated in Figure 2.4 – in this case, the activations are taken before each max pooling layer. These perceptual losses are adapted for the purpose this project's task, which is to remove artefacts. Comparing the generator output with the target via a pre-trained network's feature space allows for a more comprehensive expression of the difference. Johnson et al. (2016) define two perceptual losses, comparing style and content. To calculate these losses, the generated image as well as the target image are passed through the pre-trained loss network.

The content feature loss is the element-wise difference between the feature maps extracted from the selected layers, adapted for the restoration task as follows:

$$\mathscr{L}_{content}(I^R, I^{GT}; \phi, l) = \frac{1}{C_l H_l W_l} D(\phi_l(I^{GT}) - \phi_l(I^R)), \tag{3.1}$$

where I^R is the reconstructed image, I^{GT} is the target clean image (ground truth), ϕ is the loss network (VGG16), $\phi_l(I^{GT})$ and $\phi_l(I^R)$ are the resulting feature maps for the target and the reconstructed images at layer l, and C_l, H_l and W_l are the number of channels, width and height for the feature maps. $D(x)$ is an arbitrary distance function – the authors used Euclidean distance in the original paper, whereas Antic (2020) use Manhattan distance.

The style loss is calculated in a similar way, however, the feature maps need to be transformed to a spatially-invariant form first – the idea of style loss is to measure the difference between the *distributions* of the feature map activations, as well as the correlation between features within each feature map that is produced (Johnson et al. 2016). To find the correlation between features within a feature mapping, the Gram matrix of the feature map is calculated; the Gram matrix is the dot product between each pair of flattened feature vectors in the feature map – therefore it measures which features tend to activate together. The style loss used in this project, adapted from Johnson et al. (2016), is defined as:

$$\mathscr{L}_{style}(I^R, I^{GT}; \phi, l) = D(G(\phi_l(I^{GT})) - G(\phi_l(I^R))), \tag{3.2}$$

where in addition to the previous definitions, G is the Gram matrix operation, i.e. the result of multiplying the flattened feature matrix with the transpose of itself. Again, Antic (2020) used Manhattan distance instead of the Frobenius form used by Johnson et al. (2016).

To construct the final combined loss function, a sum of the style and content loss functions across the selected layers from the VGG16 loss network, along with a distance between the prediction and the target is calculated:

$$\mathscr{L}_{overall}(I^R, I^{GT}; \phi) = \alpha \sum_{n=1}^{n} \mathscr{L}_{style}(I^R, I^{GT}; \phi, l_n) + \beta \sum_{n=1}^{n} \mathscr{L}_{content}(I^R, I^{GT}; \phi, l_n) + \gamma D(I^{GT} - I^R), \tag{3.3}$$

where n is the number of layers selected from the VGG16 loss network, and α, β, γ are optional scaling weights (since terms are likely to have different magnitudes).

The style and content loss sums can also be weighted – the weights used in the DeOldify project (Antic 2020) are 0, 0, 20, 70, 10 for each respective ReLU layer in the VGG16 network – i.e. the activations extracted from the first two ReLUs are ignored.

Self-similarity loss The perceptual quality metric known as Self-similarity index (SSIM) was discussed in Chapter 2 in the context of it being an evaluation metric. Instead of comparing feature maps and images via MSE or MAE, we calculate the self-similarity loss between them; as we seek to maximise the SSIM index, we have defined the self-similarity loss as:

$$D_{SSIM}(x, y) = 1 - SSIM(x, y), \tag{3.4}$$

where $D_{SSIM}(x, y)$ is defined as in equation 2.4.

Relevance of loss network layers Antic (2020) takes advantage of activations that are extracted from deeper layers of the VGG16 network which precede maximum pooling operations: layers 22, 32 and 42. Those are ReLU layers, as seen in Figure 2.4. However, for the task of scratch and dust removal, we will first perform a preliminary study to find out which of the ReLU layers preceding a MaxPool layer in VGG16 would be the most "responsive" to dust and scratches, and weigh them accordingly in our feature loss implementation.

To determine how relevant the learned features in these ReLU layers are, we will investigate if any of these layers are particularly sensitive to images where analogue artefacts are present. When forwarding the images through the pre-trained VGG16, we will record the mean activations for the filters in each ReLU layer. We will then visualise the filters that get excited the most by the dustified image versus the clean one to infer whether a difference between the two images has been detected by the layer using the approach by Erhan et al. (2009).

Final proposed loss function Our final proposed loss function combines the perceptual loss from Johnson et al. (2016); Antic (2020) with novel loss based on the SSIM perceptual metric, defined in 3.4. Based on our investigation into layer relevance, we assign the following layer weights: 2, 4, 5, 6, 6; we use those for both the style and content loss sum terms in 3.3. Additionally, since the feature maps produced by each layer do hold spatially relevant information, we measure the distance between feature representations in the content loss term using our SSIM loss, rather than MAE. While feature maps produced by hidden layers will have more than 3 channels (as standard RGB colour images do), SSIM is not bound to 1- or 3- channel images, and can be implemented using a sliding window over multiple channels. We keep MAE as the distance measure for the style loss term, as the gram matrix operation flattens the feature maps, therefore they are spatially invariant. We measure both the MAE and SSIM loss between the predicted and the target image. Therefore, our final loss formulation is:

$$\begin{aligned}\mathscr{L}_{overall}(I^R, I^{GT}; \phi, w) = \alpha \sum_{n=1}^{n} \mathscr{L}_{style}(I^R, I^{GT}; \phi, l_n, w, D_{MAE}) \\ + \beta \sum_{n=1}^{n} \mathscr{L}_{content}(I^R, I^{GT}; \phi, l_n, w, D_{SSIM}) \\ + \gamma D_{SSIM}(I^{GT} - I^R) + \delta D_{MAE}(I^{GT} - I^R),\end{aligned}$$

where in addition to the definitions of 3.3, w is a set of layer weights, D_{SSIM} is the SSIM loss, D_{MAE} is absolute per-pixel distance, and δ is an additional term-scaling weight.

3.3 Training process

The network is trained progressively over the three different input sizes, i.e. 64 by 64, 128 by 128 and 256 by 256 pixels, for 10 epochs. As the input size grows, the batch size shrinks to match it. During this training, the ResNet34 backbone is frozen - therefore the feature space learned from ImageNet is preserved and used to encode the damaged input. The decoder part of the U-Net is trained to transform the generated feature representations into the target restored image. An additional training phase at the highest resolution is performed for another 20 epochs, by unfreezing the backbone and fine-tuning the encoder. During this training, the loss landscape is optimised via the Adam optimiser, using standard momentum and weight decay values (Kingma and Ba 2014). One cycle training policy (Smith and Topin 2019) is used to drastically reduce training times.

4 Implementation

This chapter provides technical detail about the process of collecting and pre-processing the data set used to train and evaluate the networks, the process of building the architectures described in the previous chapter, as well as the training process used to train the implemented networks.

4.1 Data collection and pre-processing

Collecting, pre-processing and curating the appropriate data set for the task of image restoration was the first part of this project to be implemented. It consisted of several stages, namely: scraping the images, cropping them, resizing them and finally, applying dust masks to produce the "damaged" class of samples.

4.1.1 Scraping the images

To collect the *Documerica* images, the Flickr API[1] was used. We developed a Python script called `flickr_scrape_set.py` which expects a list of photo set ID's. For each of these, a JSON response containing the metadata for the images in the photo set is fetched. This metadata is used to retrieve the highest available resolution of each image in the set. Overall, this resulted in 6232 photos fetched from several *Documerica* photo sets which are part of the overall photo collection.

4.1.2 Pre-processing the images

Initial processing of the images was done through another script, named `process_documerica.py`. This is an utility script to process the raw JPGs from Flickr. Black framing was present around the scanned images, so they were first cropped within a margin of 50 pixels on each side. Then, all images are downsized to 516 pixels on the shorter side (width or height) using OpenCV's INTER_AREA for interpolation in order to preserve as much detail as possible. Following that, center-crop patches are produced using a second utility script called `crop_patches.py`. For each image, a center-cropped patch is produced at sizes 516 by 516[2], 256 by 256, 128 by 128 and 64 by 64 pixels. The latter three of those sets are what was ultimately used as the three resolutions during progressive resizing; an example image at each resolution is shown in Figure 4.1. Downsizing is again done via OpenCV's INTER_AREA interpolation method.

4.1.3 Dustifying the images

At this stage we have three different "clean" versions of each of the 6232 images at three different resolutions. A final Python script called `dustify.py` was written to produce a corresponding "dustified" version for each image, at each resolution. The `dustify.py` script expects several arguments: the path for the directory containing the clean images at a given resolution, the path for the directory where the dust masks are contained, the output path where the dusty images

[1] Flickr API Documentation

[2] The largest resolution was not used due to running out of GPU memory when trying to train with such large images. However, it is still made available as part of the complete data set.

(a) 64x64 pixels *(b) 128x128 pixels* *(c) 256x256 pixels*

Figure 4.1: *A clean image from the data set at three different resolutions.*

will be written to, target resolution and random seed value used for the process of "generating" a pseudo-random dust mask for each image. The seed ensures that we can generate *different* dust masks for the same image at each different resolution, as demonstrated in Figure 4.2.

(a) 64x64 pixels *(b) 128x128 pixels* *(c) 256x256 pixels*

Figure 4.2: *Three dustified versions of the same image from the data set at three different resolutions. Notice that the damage "pattern" is different as to help the network generalise better.*

Dust masks are created by selecting two random patches from a limited set of twelve dust overlays. Those patches are then blended with the target clean image via Numpy (Oliphant 2006–) and OpenCV (Bradski 2000) to produce dustified images with a varying degree and patterns of damage present in the each image.

4.1.4 Final data set

The final data set contains a "clean" and a "dustified" folder for each resolution. Each folder contains 6232 images. The compressed data set has been hosted on archive.org. In this form the data set can be downloaded to Google Colab, Google Cloud VM or locally as a `.tar.gz` archive in preparation for training the networks.

4.1.5 Data augmentation

PyTorch provides a DataLoader class for loading data into batches and applying normalisation and transforms to it. Fast.ai wraps that class into a DataBunch which allows for the easier mapping of data for image-to-image tasks. We split the data set using a 8:1:1 training-test-validation ratio, resulting in 4895 pairs in the training set, 623 in the validation set, and 624 in the test

set. The random split is performed using the Pandas library; the split is recorded in three files, `train.csv`,`validation.csv`,`test.csv` so that our experiments can be reproduced.

Training and validation data are subject to data augmentation, whereas testing data is not. The training and validation data is loaded as matched image pairs of 3-channel images from the clean and dustified classes at each respective resolution. The data is normalized and scaled using the ImageNet data set's overall mean and standard deviation values (per-channel), which is a standard operation for natural image data sets. We apply the following random transformations to the data:

- random horizontal flip, probability 50%;
- symmetric warp with magnitude between (-0.25, 0.25), probability 75%;
- random zoom up to x1.5, probability 75%;
- brightness variation between (0.25, 0.75), probability 75%;
- contrast scaling between (0.5, 2.0), probability 75%.

Figure 4.3 presents an example of this random set of transforms applied eight times to the same arbitrarily chosen dustified image.

Figure 4.3: *Transforms are applied when a batch is loaded so on every iteration, a slightly different set of transforms is applied to each image.*

The validation data is used to evaluate the network at the end of each epoch; the testing data is not seen by the network until training has been completed. Augmentation is not applied to the testing data.

4.2 Networks implementation

To implement the designed generator network, as well as the described loss functions, PyTorch (Paszke et al. 2019) and its extension library Fast.ai (Howard et al. 2018) were used. PyTorch was selected as the deep learning framework to use in this project since it provides an easy-to-learn, well documented modular system of building blocks required to create neural networks. Common operations, such as convolutions, ReLU activations, maximum or average pooling operations, are provided out of the box as objects which extend the base class *torch.nn.Module* of PyTorch. Each module has a *forward* method which calculates the output of the operation for a given input.

Fast.ai provides an additional level of abstraction by using these simple modules to define more complex functions such as basic residual blocks, bottleneck residual blocks or U-Net blocks.

Skip connections are implemented via PyTorch *hooks* which allow for an intermediate layer's output to be extracted and used elsewhere in the network. Furthermore, pre-trained weights for commonly used architectures such as ResNet as well as the architectures themselves implemented as sequences of Pytorch modules are provided by the *torch.torchvision* package. Each model is wrapped in a Learner wrapper from Fast.ai which allows us to use their implementation of one cycle policy for training.

4.2.1 Generator implementation

The restoration network implemented for this project is a U-Net with a ResNet34 backbone.

The ResNet34 architecture contains a single 7 by 7 Conv2d layer and a single MaxPool2d operation, followed by sequences of basic residual and bottleneck residual blocks which progressively downsample the input to 64, 128, 256 and finally 512-channel feature map outputs. Examples of each type of block can be found in Appendix A.1 and A.2, respectively. These blocks form the backbone. The full model provided by PyTorch comes with an average pooling layer, fully connected layer, and a softmax operation - those are responsible for classification, so we can slice and remove them from the model in order to use the backbone as a feature extractor for the encoder of the restoration network.

Dynamic U-Net Fast.ai provides a Dynamic U-Net constructor which expects the encoder architecture and optionally pre-trained weights for it as arguments, creates the bottleneck of the U-Net by appending two additional convolutions, and then finally constructs the decoder part using sub-pixel convolutions for upsampling so that it corresponds to the encoder. It also allows for different image input size to be set from the standard 224 by 224 pixels.

For our restoration network, we use the ResNet34 pre-trained on ImageNet provided by PyTorch. We pass it to the Dynamic U-Net constructor which appends two convolutions giving us 1024 channel feature map outputs, followed by four U-Net blocks (corresponding to each downsampling phase of the residual blocks), and a final standalone sub-pixel upsampling block (PixelShuffle_ICNR). For an example of a U-Net block module see Appendix A.3. The excerpt illustrates the structure of a U-Net block, which contains a sub-pixel upsampling block. The second U-Net block contains a self-attention module, for example structure see Appendix A.4.

Skip connections are implemented via PyTorch hooks, so that residual output from the encoder can be retrieved and appended to the input of the decoder U-Net blocks. Due to the network structure, note that the number of parameters is the same for all three input sizes: 41,405,588, of which 20,137,940 are trainable and 21,267,648 are frozen initially. Upon unfreezing the encoder during the fine-tuning stage, *all* 41,405,588 parameters become trainable.

4.3 Perceptual loss implementation

The perceptual loss used in the project is inspired by Johnson et al. (2016) and the implementation of Antic (2020). It expects several arguments:

- base_loss - comparison function to find distance between images and/or feature maps,
- ssim_loss - a second distance loss parameter which is optional,
- m_feat - model to extract the feature maps from,
- layer_ids - a list of layers which to hook from the model,
- layer_wgts - weights to apply to each feature map loss term from each corresponding layer that has been hooked.

The loss is then attached to the Fast.ai Learner wrapper to our PyTorch model. We use the pre-trained VGG16 model as a feature extractor and we hook layers 5, 12, 22, 32, 42 from it, which correspond to ReLU layers before each MaxPool2d layer.

The base loss is usually MSE or MAE. We contribute an additional loss type, SSIM loss. All loss implementations extend the *torch.nn.Module* base class. SSIM loss implementation is derived from the SSIM metric implementation in PyTorch by Su (2017). We used window size of 5 by 5.

4.4 Training

To train, we only need to call the fit_one_cycle method that each Fast.ai Learner object has with the number of epochs, the learning rates, and optionally the percentage of iterations over which the learning rate should increase within a cycle. When new layers are initially appended to a pre-trained backbone, their weights are randomly initialised. As we first train only those layers (the backbones are frozen), the learning rate is increased for 80% of the iterations; on the contrary, when the backbone of a network is unfrozen and we are fine-tuning it, learning rate increases only 50% of the time. To optimise the loss landscape, we use Adam optimiser with the PyTorch default betas. Weight decay is set to $1e-3$. During training, apart from the training and validation loss values, we record metrics such as MSE and SSIM, averaged over the validation data. Using Weights & Biases (Biewald 2020), we log predictions on an arbitrary batch from the valdation data at each epoch. To prevent overfitting, we save the model with the lowest validation error at each epoch, and reload that at the end of training using a Fast.ai Callback. We train in mixed precision by calling to_fp16() method on the Learner wrapper.

4.5 Visualising feature activations

We use PyTorch hooks to extract the activations from the network, then calculate the mean activation for each feature map obtained via the hook for that layer to select the features (filters) that show the strongest activation to a dustified and a clean version of an image from the data set. This allows us to select the most strongly-activated filters. We then implement the approach described by Erhan et al. (2009) to optimise a random image tensor so that it maximally activates these filters in order to obtain a pattern which we use to visualise the filter. The optimisation implementation utilises Fast.ai along with PyTorch to define a class called FilterVisualizer [3]. The FilterVisualizer employs a similar process to progressive resizing during optimisation in order to generate high-quality visualisations of patters associated with filters of relatively small, static size. The random image is initially optimised at a low spatial resolution, then upscaled by a factor, and optimised again at the new resolution. The class takes the following arguments: size - the lowest resolution starting size in which to optimise the randomly generated image; upscaling_steps and upscaling_factor - number of upscaling steps for and upscaling factor value for progressive resizing. We used starting size of 56 for the randomly generated noise image, and optimised it for 40 iterations, with 12 upscaling steps and an upscaling factor of 1.200[4]. Optimisation on the randomly initialised image's pixels is performed via Adam (Kingma and Ba 2014); since Adam minimises the loss, and we want to maximize the average activation filter, we invert the sign of the loss by multiplying by -1. We use the learning rate of $1e-1$ for this optimisation.

[3] The implementation used in this project is based on the article *How to visualize convolutional features in 40 lines of code* by Fabio M. Graetz.

[4] In some cases, the optimisation of the random noise image for the last ReLU layer does not converge to generating a pattern for some filters within the 40 iterations - it is possible to optimise for longer to remedy that, however, for the purpose of this study it is enough to demonstrate whether the filters activated between the two images within a pair overlap or not.

5 Experiments & Evaluation

This chapter details the experiments performed during this project. In order to evaluate how well the existing solutions fit the task of artefact removal, we performed preliminary baseline experiments using the existing feature loss (i.e. perceptual loss) approach from Antic (2020). We then identified issues with it through quantitative and qualitative evaluation and gradually amended the approach in order to address these issues.

5.1 Restoration network trained with MSE loss function

The first experiment we performed is training the U-Net restoration network using a simple per-pixel loss function - mean squared error (MSE). The images were loaded as 3-channel RGB tensors, scaled and normalized.

5.1.1 Training

In this section we track and analyse the model's training using the training and validation data.

Stage 1: Progressive resizing, frozen encoder The first stage of training involves training only the decoder of the U-Net, while keeping the pre-trained ResNet34 encoder frozen; we progressively increase the input/output image size every 10 epochs. The initial hypothesis is that the feature space learned by the encoder from training on the ImageNet data set would be sufficient to represent the damaged images as feature vectors that the decoder can then use to at least identify the damaged regions. Through progressive resizing and one cycle training policy, we allow the decoder to learn as quickly as possible - the total training time over all 30 epochs was 29 minutes and 13 seconds. The training history across all three sub-stages of progressive resizing is displayed in Figure 5.1.

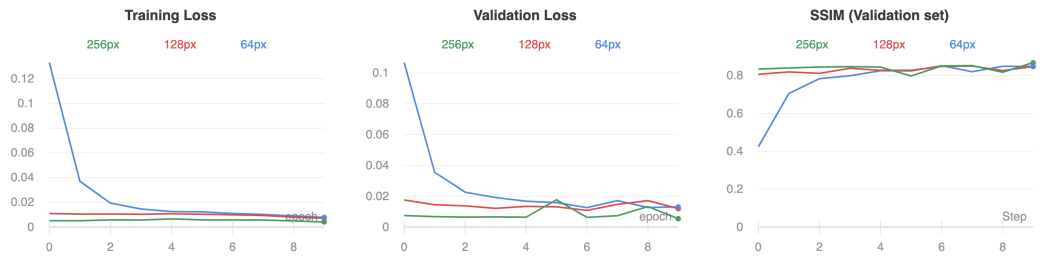


Figure 5.1: *Training history of Stage 1. We can see that progressive resizing helped the restoration network learn quickly in lower resolutions (64x64, 128x128), and improve further in the highest resolution (256x256).*

The final training loss is 0.004, validation loss is 0.005, and the final average SSIM score across the validation set achieved by the model is 0.860. Additionally, we logged predictions on a randomly selected batch of the validation set at the end of each epoch for visual inspection. An example is shown in Figure 5.2. We can clearly infer that the generator has successfully learned to detect

the damaged regions in the image - however, the repair that has been performed is not excellent, with a high-frequency noise pattern visible in the sky section of the prediction in 5.2a. Still, it is notable that the weights obtained via transfer learning in the encoder part of the network allowed it to be able to separate "natural" image features, i.e. similar to ImageNet, from unnatural artefacts, such as dust and scratches.

(a) After the first 10 epochs, training at 64x64 resolution.

(b) After the second 10 epochs, training at 128x128 resolution.

(c) After the last 10 epochs, training at 256x256 resolution.

Figure 5.2: *Visualisations of the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target. We have visualised the same image from the validation set at the end of each 10-epoch sub-stage, i.e. at each different resolution, after 10 epochs of training.*

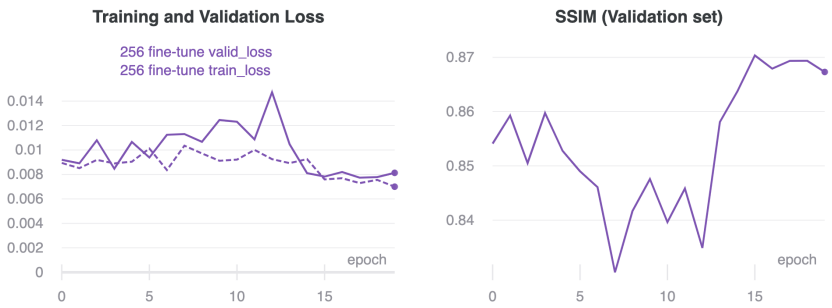


Figure 5.3: *Training history of Stage 2. We can see the loss oscillates and ultimately decreases slightly.*

Stage 2: Unfreezing the generator's encoder At this stage we unfreeze the ResNet34 encoder and fine-tune the whole network for 20 epochs - we hypothesise that fine-tuning the weights of the encoder to this specific task will allow the network to achieve improved quality of restoration,

as the feature space will be adapted to better fit the specific artefact removal task. This stage took 39 minutes and 5 seconds and the corresponding training history is shown in Figure 5.3.

The achieved training and validation loss values are 0.007 and 0.008 respectively, while the overall SSIM score of the validation set remained at 0.860. This shows no improvement over the scores achieved before unfreezing the backbone; it is possible that this is due to only training for 20 epochs. To investigate further, we qualitatively inspect the results on the validation data. There is marginally small improvement between 5.2c and 5.2. The repaired regions are still visible, especially in the sky section of the image.

Figure 5.4: *Visualisations of the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target. We have visualised the same image from the validation set as in 5.2.*

5.1.2 Inference and evaluation

For the purposes of this investigation, we calculate the SSIM scores for each clean-dustified pair, in the training set and plot the distribution of the result. The distribution of the obtained values is displayed in Figure 5.5.

The damage present in the test set is varied, as evidenced by the wide range of SSIM scores obtained by the set of clean-dustified pairs: lowest score is 0.200 SSIM (i.e. a large degree of damage is present), while the highest score is 0.970 SSIM (i.e. very little damage is present). As all clean-dustfied pairs produced SSIM scores of less than 1, we can infer that all damaged images in each pair have some degree of damage.

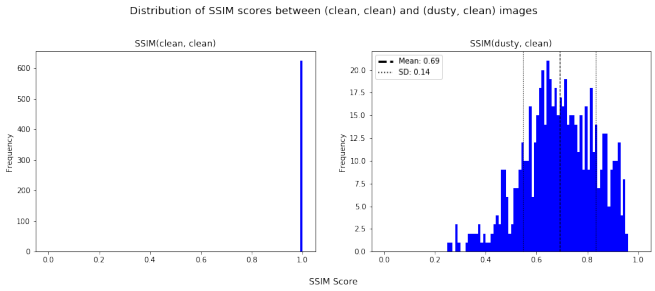
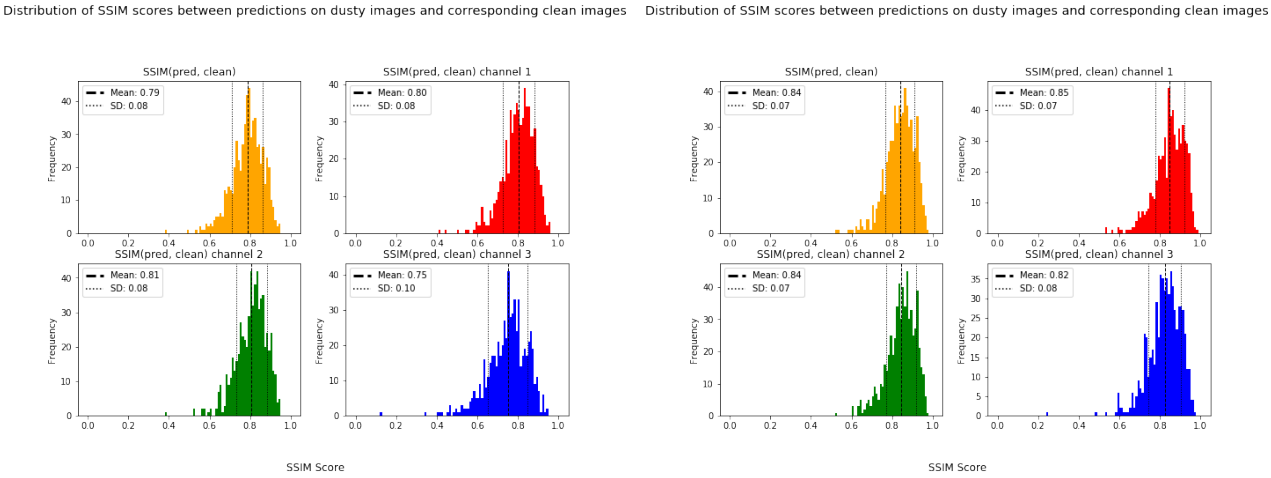


Figure 5.5: *Distribution of SSIM scores for items in the test set. For comparison, we have displayed the SSIM of clean-dustified pairs against the SSIM of each clean image and itself.*

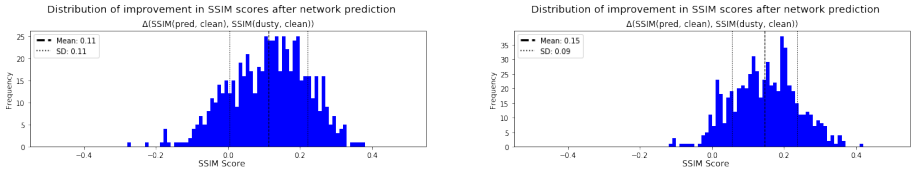
Restoring artefact damage After training the model at the highest resolution (256 by 256 pixels), we perform inference on the training set and evaluate by calculating the SSIM scores between the predicted restored images and the target clean images; we calculate overall image SSIM as wel as per-channel SSIM. Comparing the histograms for the network before unfreezing the weights (Figure 5.6a) and after unfreezing the weights (Figure 5.6b) with the SSIM scores when no restoration has been performed (Figure 5.5), we can see that in both cases the distribution range has been squeezed: the standard deviation of the SSIM score for the damaged images is 0.140, and has shrunk to 0.800 and 0.700 for the images restored by the generators with frozen and unfrozen encoders respectively. Furthermore, the mean of the distribution has been shifted right from 0.690 to 0.790 and 0.840 for each respective case.



***(a)** before unfreezing the encoder* ***(b)** after unfreezing the encoder.*

***Figure 5.6:** Distribution of SSIM scores for predictions on the test set before and after unfreezing the encoder. First histogram (yellow) in each group shows overall image SSIM scores, while the other three (red, green, blue) correspond to the SSIM scores for each channel.*

Comparing the performance between the two stages of training the network, we observe that the blue channel distribution has a larger spread than the other two channels, with some samples scoring just over 0.100. Unfreezing the weights has helped mitigate this, and has significantly improved the standard deviation and mean, as well as further shrinking the spread of both the overall and per-channel SSIM scores. However, the blue channel still has an outlier with an SSIM score of 0.200.



***(a)** before unfreezing the encoder* ***(b)** after unfreezing the encoder.*

***Figure 5.7:** Distribution of improvement (δSSIM) for predictions on the test set before and after unfreezing the encoder. These histograms show us by how much images increased (or decreased) their SSIM scores after being restored by the generator before unfreezing the encoder and after unfreezing it and fine-tuning.*

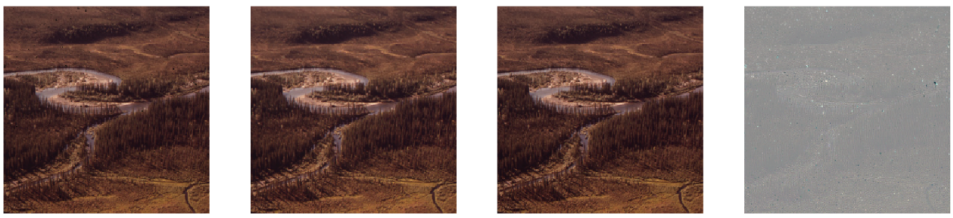
Additionally, we seek to quantify the degree of *improvement* seen across the test set when predictions are made. We calculate the difference between the SSIM score of each clean-dustified pair and the SSIM score of each corresponding clean-restored pair to produce the *δSSIM* measure. The distribution of this measure for the two stages of training are shown in Figures 5.7a and 5.7b, respectively. Again, the spread has been decreased and the mean has been shifted to the left in the fine-tuned model. In both cases we notice that some examples have seen a negative change in their SSIM scores after being restored by the network - i.e., the network has introduced additional damage in the process of restoration.

Table 5.1: *Scores and deltas for the top 3 most damaged test images by the network after each training stage. Note that the three images most damaged by either network (with frozen or fine-tuned encoder weights) are different.*

(a) *before unfreezing the encoder*			***(b)*** *after unfreezing and fine-tuning*		
SSIM (dusty, clean)	SSIM (pred, clean)	*δSSIM*	SSIM (dusty, clean)	SSIM (pred, clean)	*δSSIM*
0.883	0.605	−0.278	0.946	0.831	−0.115
0.833	0.609	−0.223	0.932	0.883	−0.109
0.888	0.705	−0.183	0.950	0.888	−0.106

Table 5.1 supports the proposition that the fine-tuned network performs better - the deltas of the most damaged images have smaller magnitude.

(a) *Prediction with the second lowest delta (−0.223)* ***before*** *unfreezing the encoder.*



(b) *Prediction with the lowest delta (−0.115)* ***after*** *unfreezing the encoder and fine-tuning.*

Figure 5.8: *Hand-picked samples from the top 3 lowest-delta-scoring items, refer to 5.1. Visualisations include the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target.*

After visual inspection of these samples (Figure 5.8), we noticed a visible colour shift in the predictions made before unfreezing the encoder, which was not present in the sample predictions made after unfreezing it and fine-tuning. This confirms that fine-tuning the ResNet34 encoder helped with the network introducing damage to the colour information in the images. We conclude that fine-tuning the encoder is beneficial.

(a) *δSSIM* =0.413

(b) $\delta SSIM$=0.363

Figure 5.9: *Hand-picked samples from highest-delta-scoring items after restoration from the fine-tuned network. Visualisations include the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target.*

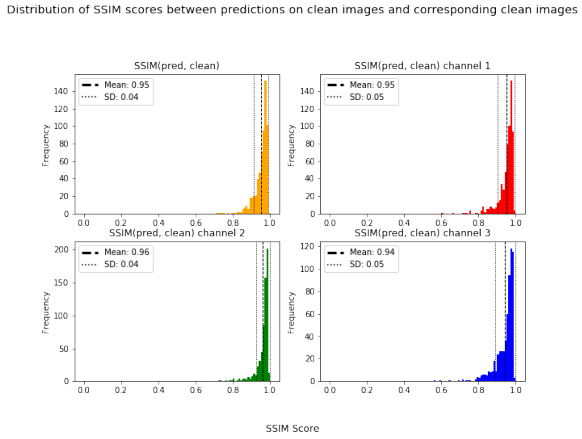


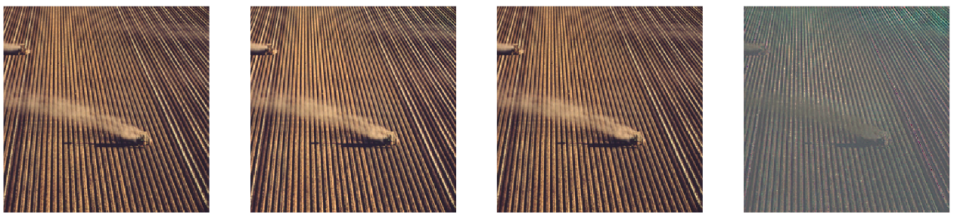
Figure 5.10: *Distribution of SSIM scores for predictions on clean images after unfreezing and fine-tuning the encoder. First histogram (yellow) shows overall image SSIM scores, the other three (red, green, blue) correspond to the SSIM scores for each channel.*

Finally, we visually inspect the predictions with largest $\delta SSIM$ scores for the fine-tuned network, i.e. the ones that saw the biggest improvement after restoration - showed in Figure 5.9. We observe that the inputs have a high degree of artefact damage, and that the damaged sections are correctly identified by the network, but the in-painting has created a distinguishable pattern, most visible in the sky sections. This can be remedied by substituting the per-pixel loss function with a loss which evaluates the overall perceptual quality of the restored output.

"Restoration" damage In the above experiment we observe that the most damaged samples by the second network were images who had very little damage to begin with - see Table 5.1 and Figure 5.8b.

We designed an additional experiment inspired by this observation: pass *clean* images through the restoration network, and measure how much damage is inflicted. As seen in Figure 5.5, images with no damage have SSIM scores of 1.000 when compared to themselves, therefore the histogram represents a single bin, with mean of 1.000 and no variance. We can see in Figure 5.10 that the network has "spread" the distribution of SSIM scores for the clean test images after "restoring" them.

(a) Least damaged clean image: SSIM(pred,clean)=0.990



(b) *Most damaged clean image: SSIM(pred,clean)*=0.719

Figure 5.11: *Samples from predictions made on clean images. Visualisations include the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target.*

Additionally, we visualise the most and least damaged examples in Figure 5.11. The lowest-scoring sample displays a small colour shift in the upper corner.

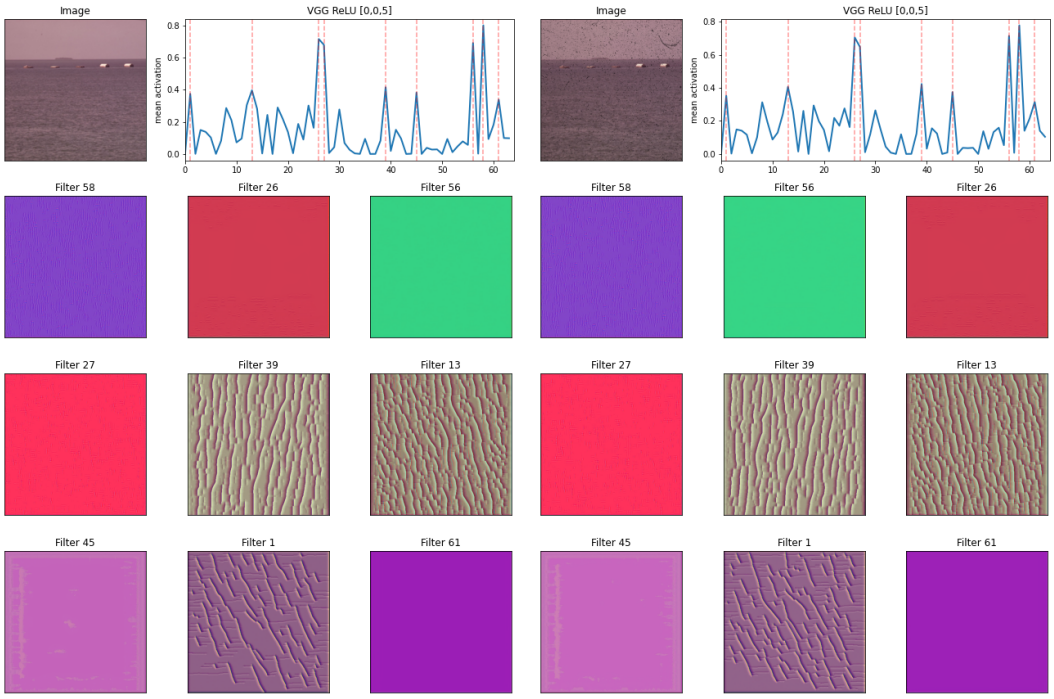
5.2 Restoration network trained with Perceptual loss

To improve upon the pattern that is created in place of artefacts by the network trained using MSE, we employ the perceptual loss approach for colourisation from Antic (2020), which in turn is borrowed from style transfer and superresolution feature losses by Johnson et al. (2016).

5.2.1 Investigation of the loss network

As mentioned in Chapter 3, the activations extracted from the first two ReLUs of the VGG16 loss network are ignored in the DeOldify project implementation for perceptual loss (Antic 2020). This could be explained with the fact that low-level features learned by earlier layers are not relevant to the task of colourisation, since to colourise something in a correct way, a higher-level, semantic knowledge is required, and that is learned by the deeper layers of the VGG16 network. On the other hand, we hypothesise that the activations for shallower layers could be relevant to the task of artefact removal and should not be discarded.

In order to inform our approach better, we will visually inspect the filters in each ReLU layer that get activated by the clean versus the dustified version of the same image. If we can demonstrate that certain layers are sensitive to the presence of analogue damage, we will assign larger weights to those layers when calculating the perceptual loss function for the restoration network. We performed this investigation on the highest-delta-scoring items after restoration from the fine-tuned MSE network (shown in Figure 5.9). We also tested a third arbitrary image with a smaller amount of damage present. For the selected images, we pass both the clean and the dustified image from the same pair through the pre-trained loss network - VGG16. For each photo, we hook the activations at the interesting ReLU layers of the VGG16 network, as used in the feature loss formulations - i.e. layers 5, 12, 22, 32, 42. Each of those will have 64, 128, 256, 512 and 512 feature maps (filters), as per the VGG16 architecture definition. The mean activations for each filter are recorded, and the 9 most activated filters are recorded. The technique implemented in 4.5 is used to create an image pattern which corresponds to each of the selected filters. We then compare the mean activations at the same layer for the clean and dustified version of each image. This investigation resulted in ten visualisations for each image pair that was tested - one image per layer for each image in the pair, for the five ReLU layers. All visualisations can be found in Appendix B. For the purpose of this investigation, we will show one example visualisation where the top 9 actvations for the clean image versus the dustified one were the same, and two examples where they were different. We also observe the difference of the overall distribution of mean activations for each image within a pair, shown as a line plot in each respective figure, when assigning a layer's relevance.



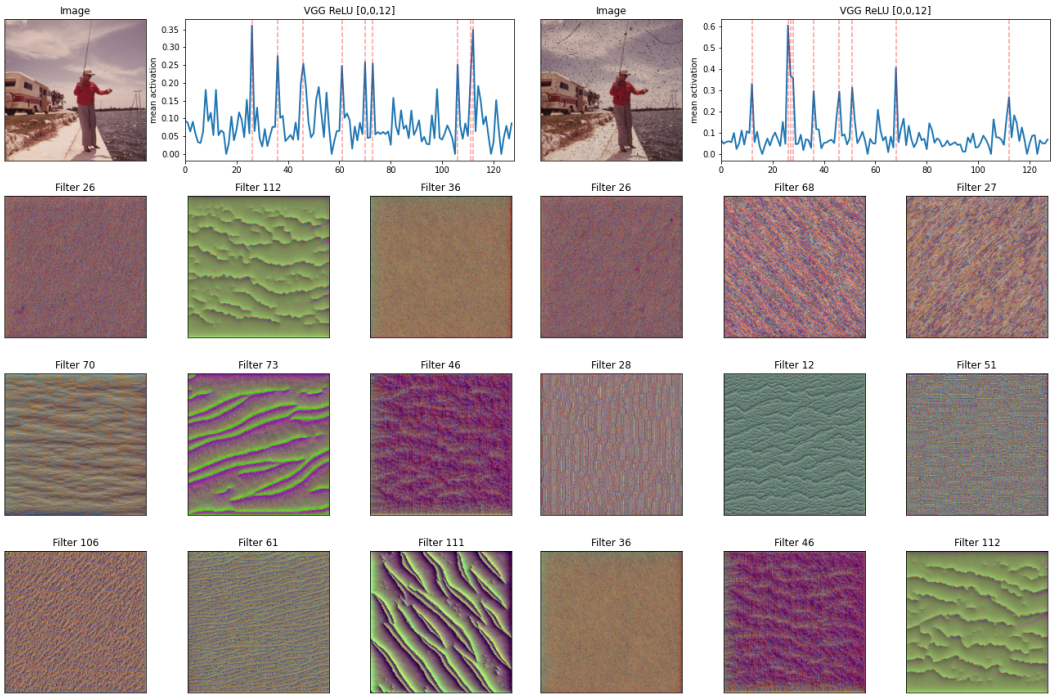
***(a)** clean image* ***(b)** dustified image*

Figure 5.12: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][5] for the clean (a) and dustified (b) version of the image from 5.9a (Image 1). Also included is a plot of the mean activations all 64 filters in the layer. The orange dotted lines correspond to the 9 most activated filters.*

In 5.12, the visualisation demonstrates that virtually the same filters were activated in the first interesting ReLU layer by the clean and by the dustified image from the same pair, corresponding to the repaired image from 5.9a. As these filters are from a relatively shallow layer in the VGG16 network, most of them get activated by colour and simple patterns, rather than more complex patterns. In the visualisations for this layer for our other test image pair, Figure 5.9b, which can be found in Appendix B, we found that 7 of the top 9 filters were the same, and the overall mean activation distribution was very similar. For the third image tested, we also found that all top 9 filters were shared between the clean and dustified images. We therefore conclude that this layer is not strongly sensitive to dust and scratches being present in an image.

On the other hand, the visualisations for next ReLU layer (12-th layer of the VGG16 network) showed that different layers get activated by the clean and dustified image in a pair. For instance, the visualisation for the image corresponding to the restoration from Figure 5.9b is displayed in Figure 5.13. We observe the mean activation spikes in the plots for the two images are different; 5 of the top 9 filters are also different. Similar behaviour was observed for the other strongly damaged image from 5.9 at this layer too. In the third image we chose with a smaller degree of damage, we observed one different filter in the top 9 between the clean and damaged versions. We conclude that this layer is more sensitive to dust and scratches than the previous one.

We found that stronger sensitivity was exhibited by deeper ReLU layers, such as the 32-nd layer of the VGG16 network. One example is shown in Figure 5.14 and the rest can be found in Appendix B. In Figure 5.14 we can see that while there is an overlap of 6 between the two top 9 most activated sets, the overall distribution of the mean activations is different for the clean and the dustified image.



***(a)** clean image* ***(b)** dustified image*

Figure 5.13: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][12] for the clean (a) and dustified (b) version of the image from 5.9b (Image 2). Also included is a plot of the mean activations all 128 filters in the layer. The orange dotted lines correspond to the 9 most activated filters.*

These visualisations recall the visual cell hierarchy of Hubel and Wiesel (1959; 1968) - the first ReLU layer has a very small receptive field (5 by 5 pixels[1]), which is smaller than the spatial size of most scratches, whereas the second ReLU's receptive field is slightly larger (16 by 16 pixels), and at that stage the network begins to detect the dust and scratches.

Table 5.1 summarises the number of different filters within the top 9 sets for each clean and dustified image pair, for the three images examined. The first two images have a high degree of analogue artefact damage present, while the third has a smaller degree of damage. All visualisations are available in Appendix B.

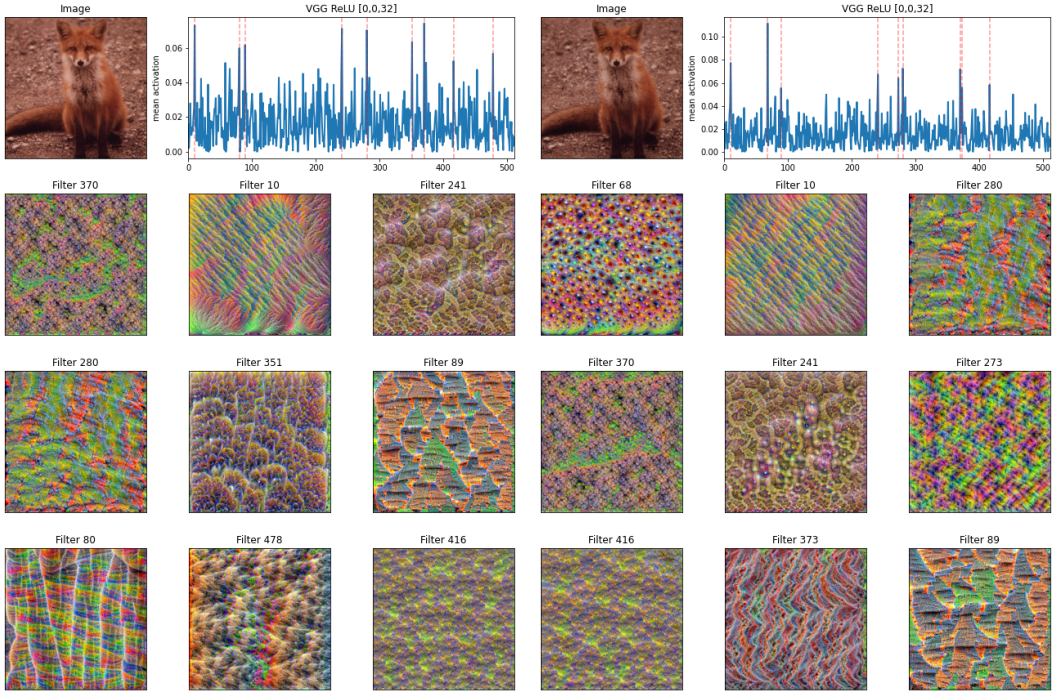
Table 5.2: *Number of different filters within the top 9 filter sets for the clean and dustified version of each image. If all filters in both top 9 sets are the same (irrespective of ranking), the result would be 0 different filters, and on the contrary, if they are all different, the result would be 18.*

	Image 1	Image 2	Image 3
ReLU 5	0	4	0
ReLU 12	10	10	6
ReLU 22	6	8	2
ReLU 32	14	16	6
ReLU 42	14	12	12

While we recognise this investigation is limited as we only examined a small selection of images, it is evident that the presence of dust and scratches is related to the activations of the three deeper ReLU layers used in the perceptual loss formulation of Antic (2020). However, the first two layers (ReLU 5 and ReLU 12), discarded in that formulation, also respond to the presence of analogue artefacts (especially the second one), according to this

[1]Equations used to calculate this can be found in this article: *A guide to receptive field arithmetic for Convolutional Neural Networks*.

investigation. For that reason, they are also accounted for in our perceptual loss formulation.



(a) *clean image* ***(b)*** *dustified image*

Figure 5.14: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][32] for the clean (a) and dustified (b) version of an image with lesser degree of damage (Image 3). Also included is a plot of the mean activations all 512 filters in the layer. The orange dotted lines correspond to the 9 most activated filters.*

5.2.2 Modified Perceptual loss

Based on the observations in the previous section, we trained the restoration network with three different perceptual loss functions, all based on the VGG16 architecture:

- Perceptual Loss **Model 1**: We assigned the layer weights as used in the DeOldify project, i.e. 0, 0, 20, 70, 10 (Antic 2020) corresponding to layers 5, 12, 22, 32, 42 from VGG16 (ReLUs). We also used the same distance function, MAE - refer to 3.3 for both style and content loss terms, as well as for the image distance term. See section C.1.
- Perceptual Loss **Model 2** : Based on our experiment investigating the sensitivity of each of the five ReLUs, we assign the following weights to them: 2, 4, 5, 6, 6. In other words, we give some weight to the first two layers, which were previously ignored, and progressively increase the weights for deeper layers. Again, we use MAE as the distance measure for both style loss and content loss. See section C.2.
- Perceptual Loss **Model 3**: We train with the weights of Model 2, and also add our own distance metric - SSIM loss (see 3.4) with a window size of 5 by 5 pixels. We use the SSIM loss as a distance measure between the prediction and target images, as well as the distance function in the content loss term; for the style loss term, we use MAE, and we also measure the MAE distance between the target and the prediction. See section C.3.

We use the same training policy as with the model trained using MSE, more detail in section 4.4.

Table 5.3: *Average SSIM scores on the validation set achieved by each of the three perceptual loss models before and after unfreezing the encoder and fine-tuning.*

	Before fine-tuning	After fine-tuning
Model 1	0.834	0.841
Model 2	0.866	0.886
Model 3	0.892	0.898

Besides training and validation loss while training, SSIM and MSE were recorded as independent metrics while training the models in order to be able to compare them (since the loss functions' magnitudes are different). In Figures C.1, C.2, C.3 from Appendix C we have used the SSIM metric to compare the training of the three models. Results are summarised in Table 5.3. All models exhibited similar behaviour during training, achieving SSIM scores above 0.800 on the validation set without unfreezing the encoder. Additionally, all three models saw some degree of improvement in the achieved SSIM scores after fine-tuning. Naturally, Model 3, which has the SSIM loss between prediction and target as an explicit term in its loss function, achieved the best SSIM scores.

5.2.3 Evaluation and inference

We evaluate the trained models using the same approach we used to evaluate the models trained using per-pixel loss. We assess the ability to restore artefacts, and measure what degree of new damage is introduced to clean images, for each model, through calculating the SSIM score between the model's prediction and the target clean image. We compare the obtained distributions to the SSIM score distributions of the clean-dustified pairs in the test set (see Figure 5.5), as well as the baseline model trained and fine-tuned using per-pixel loss (see Figures 5.6b, 5.10. Histogram plots for the scores of the three perceptual losses are available in Appendix D. Results are summarised in Table 5.4 and compared to the model trained using MSE pixel loss as a baseline. A mean of 1.000 and standard deviation of 0.000 is ideal for both predictions on clean and dustified images - it would mean that the model has perfectly repaired the dustified images, and has perfectly preserved the clean images.

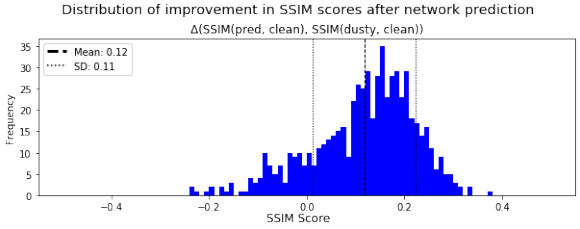
Table 5.4: *Summary of mean and standard deviation values for the SSIM scores distribution obtained by each model on the test set of 624 image pairs.*

	SSIM of Predictions on dustified images		SSIM of Predictions on clean images	
	Mean	SD	Mean	SD
MSE Model	0.840	0.070	0.950	0.040
Model 1	0.800	0.080	0.770	0.080
Model 2	0.880	0.050	0.940	0.040
Model 3	0.880	0.050	0.980	0.020

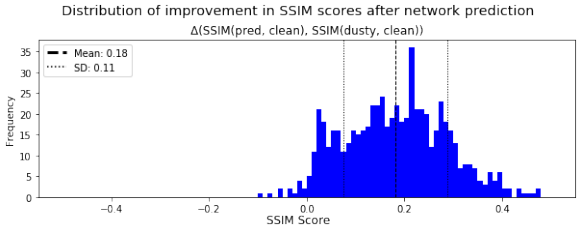
Model 1 The SSIM scores distribution for predictions made by Model 1 on the 624 dustified images from the test set are shown in Figure D.1a, including per-channel breakdown. This model obtained worse scores than the model trained using simple pixel MSE as loss function - see Table 5.4. It obtained an especially low mean value for the SSIM scores of predictions made on clean images (Figure D.1b) - suggesting that the network has a destructive effect on non-artefact image data - possibly fine detail, as only activations from deeper VGG16 layers are included in the loss function.

Model 2 The SSIM scores distribution for predictions made by Model 2 on the dustified images from the test set are shown in Figure D.2a. Our proposed inclusion of the first two layers in the perceptual loss, as well as amended weights for all layers based on our experiments in Section 5.2.1, improved model performance. This model has a higher mean value than both the baseline and Model 1 for SSIM scores on predictions over dustified images, as well as smaller spread of the overall distribution - see Table 5.4. The mean SSIM scores for predictions on clean images (Figure D.2b) is comparable to that of the baseline and much better than those obtained by Model 1, which demonstrates that the inclusion of the first two ReLU layers in the loss function aided Model 2 in learning to preserve non-artefact image information much better than Model 1.

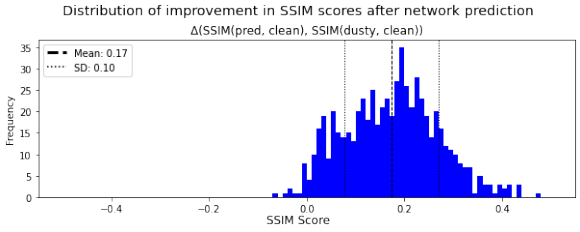
Model 3 The SSIM scores distribution for predictions made by Model 3 on the dustified images from the test set are shown in Figure D.3a. This model includes the addition of SSIM loss term between prediction and target, as well as as a distance measure for comparing feature activations in the content loss term of the feature loss. The mean and standard deviation for the SSIM scores are the same as Model 2 (see Table 5.4) - however, the histogram in Figure D.3a is more symmetric than Figure D.2a, i.e. we have fewer samples with scores at the tail of the distribution. Furthermore, this final model obtained the highest mean SSIM score on predictions over clean test images, 0.980, along with smallest standard deviation, 0.020, demonstrating that it was by far the least intrusive out of all networks we trained. Histograms in D.3b show the distribution of SSIM scores on clean images is much more narrow than those of previous models.



(a) Model 1



(b) Model 2



(c) Model 3

Figure 5.15: *Distribution of improvement (δSSIM) for predictions on the test set for the three perceptual loss models. These histograms show us by how much images increased (or decreased) their SSIM scores after being restored by each network.*

Visual inspection These results suggest that the final model has learned to recognise analogue artefacts in damaged images and introduce little new damage. Next, we examine if besides recognising, the networks can seamlessly restore the missing information in areas affected by analogue artefacts. We use our *δSSIM* measure to quantify the degree of "improvement" the network introduced in predictions on dustified images, shown in Figure 5.15. Model 3 achieved the smallest spread of *δSSIM*. Both Model 2 annd 3 and produced a smaller number of predictions which obtained lower SSIM after they were restored by the network (i.e. they have *δSSIM* below 0). Based on the *δSSIM* scores, we visually examine some of the dustified images that each network improved the most, versus the ones it improved the least.

In Figure 5.16a Model 1 successfully detected the artefacts in an image with a high degree of damage, but failed to in-paint them seamlessly. Figure 5.16b shows a sample with a negative *δSSIM*; the damaged input has a lower degree of damage, and along with removing it, the network has degraded detail as well as colour information.

Model 2 performs better; Figure 5.17a shows the prediction with the highest *δSSIM* across all predictions from all models, with small loss of fine detail. Figure 5.17b shows that it has further damaged an image with a low degree of damage minimally, with some loss of fine detail.

Model 3 also successfully detected and in-painted regions damaged by artefacts, as seen in Figure 5.18a; some fine detail was lost, as seen in the absolute pixel-wise difference visualisation. Notice that in the example with the lowest *δSSIM*, artefact-like specks are present in the ground truth, and the model has removed them. The difference shows that the model has exclusively focused on entities that look like analogue damage.

(a) Prediction with one of the highest deltas - 0.336.

(b) Prediction with one of the lowest deltas - −0.233.

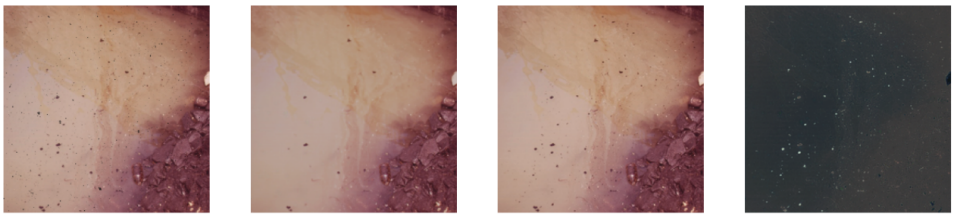
Figure 5.16: *Hand-picked sample predictions from Model 1 which saw the biggest and the smallest improvement to their SSIM score after restoration. Visualisations include the input, prediction, target, as well as absolute per-pixel difference between the prediction and the target.*

(a) Prediction with one of the highest deltas - 0.589.

(b) Prediction with one of the lowest deltas - −0.093.

Figure 5.17: *Hand-picked sample predictions from Model 2 which saw the biggest and the smallest improvement to their SSIM score after restoration. Visualisations same as 5.16.*

(a) Prediction with one of the highest deltas - 0.478.



(b) Prediction with one of the lowest deltas - −0.047.

Figure 5.18: *Hand-picked sample predictions from Model 3 which saw the biggest and the smallest improvement to their SSIM score after restoration. Visualisations same as 5.16.*

As a final qualitative comparison between the three networks, we will visualise how each of the three restored the same image. Figures 5.19 and 5.20 are best viewed enlarged.

(a) Input *(b) Model 1 prediction* *(c) Model 2 prediction* *(d) Model 3 prediction* *(e) Target*

Figure 5.19: *Input (a) has a large amount of small dust specks. Model 1 (b) has introduced a high-frequency pattern in place of the artefacts. Model 2 (c) and Model 3 (d) have successfully removed the artefacts, without introducing new ones.*

(a) Input *(b) Model 1 prediction* *(c) Model 2 prediction* *(d) Model 3 prediction* *(e) Target*

Figure 5.20: *Input (a) has a large scratch artefact. Model 1 (b) blurred the image, introduced a colour shift and lost a lot of fine detail, without removing the large scratch. Model 2 (c) has rpeserved detail but has not removed the scratch. Model 3 (d) has successfully removed most of the large scratch, and preserved detail.*

5.3 Summary

To summarise, we found that the inclusion of the earlier ReLU layers from the VGG16 network helped preserve detail, minimise colour shift and improve the quality of in-painting. Additionally, the introduction of SSIM loss as part of the the perceptual loss to compare both predictions and target, and the feature activations from each ReLU layer in the content loss, allowed the the network to generalise over different artefact sizes and shapes, and made it minimally invasive to areas where no artefacts are present.

6 Conclusion

This dissertation presents a case study for the usage of convolutional neural networks to repair analogue artefact damage from old film photographs. Ultimately, the work on this project resulted in the development a novel loss for restoration which combines perceptual loss based on the activations from feature extractor network with a difference measure derived from the Structural Similarity Index (SSIM) metric.

The background research covered state-of-the-art techniques used for similar image-to-image restoration tasks, such as superresolution and colourisation. The conducted analysis resulted in identifying several interesting points of investigation, namely:

- How useful are pre-learned natural image feature representations to the task of artefact restoration?
- How can the relevance of the hidden layers from a feature extractor network be reliably measured when engineering feature-represenation-based loss functions?
- Are there more effective ways of comparing the extracted feature representations than L1 or L2 distance?

In an attempt to answer these questions, we collected, processed and curated a data set of clean and synthetically "damaged" images, using photographs from the *Documerica* collection. We trained and evaluated a restoration network based on the U-Net architecture, with a pre-trained ResNet34 encoder.

Our first set of experiments showed that the encoder pre-trained on natural image data (ImageNet), along with progressive resizing, one cycle training policy and mixed precision training allowed the baseline restoration network to learn to identify analogue artefact damage within 30 minutes of training. Fine-tuning was shown to improve restoration quality and reduce the amount of new damage that the network would introduce. However, mean squared error (MSE), i.e. a per-pixel loss, proved insufficient as an objective function for the task of artefact restoration, according to our quantitative and qualitative evaluation. Unsurprisingly, the network trained only with MSE loss produced predictions which lacked in detail. Colour shifts, especially in the blue channel of the RGB predictions, were observed – it is possible that this is due to analogue image scans generally skewing towards warmer tones.

The second set of experiments was concerned with assessing the sensitivity of certain layers of the VGG16 network, inspired by the feature representation hierarchy found in CNNs, which is reminiscent of biological visual systems (Hubel and Wiesel 1959; 1968; Thomson 2010). Through forwarding both dustified images and their corresponding clean images through a pre-trained VGG16 and extracting the mean activations at each relevant layer, we observed that shallower layers did exhibit sensitivity to the presence of artefacts in the image, albeit lower than that of deeper layers. While these experiments were limited both in terms of the damage being synthetic, and because we only examined three image pairs due to time constraints, they provided an argument in favour of modifying the perceptual loss function to also include activations from shallower layers. We based our hypothesis on the fact that earlier layers have smaller receptive fields, some of which only respond to colour or small, localised textures and patterns; therefore, we include them in the perceptual loss for both style and content in an attempt to improve fine detail in the predictions and reduce possible colour shifts.

The observations from these tests served as the basis for an amended set of layer weights for the perceptual loss. We compared our proposed weights with the weights used by Antic (2020) for colourisation by measuring the *perceptual* closeness of predictions on both clean and dustified images to the clean ground truths. The results include:

- improved mean SSIM score for predictions on dustified test images from 0.800 to 0.880, and reduced SSIM score standard deviation from 0.080 to 0.050 – meaning that more restored predictions, on average, were perceptually close to the clean targets;
- improved mean SSIM score for predictions on clean test images from 0.770 to 0.940, and reduced SSIM score standard deviation from 0.080 to 0.040 – demonstrating that the inclusion of the earlier layers as well as the modified layer weights dramatically reduced new damage introduced by the network, and further suggesting that the network had learned to separate analogue artefacts from natural image features which need to be preserved;
- improved quality of in-painting based on qualitative inspection.

Seeking to improve the quality of restoration and reduce the damage that the network itself might introduce, the final stages of this project involved incorporating the SSIM perceptual quality metric as part of the VGG16-based perceptual loss. We derived a simple loss from the SSIM metric which was used to compare the restored predictions with the ground truth. Additionally, we also substituted the absolute distance measure between the feature map activations in the content loss term for perceptual loss with SSIM loss. We evaluated a model trained with this modified perceptual loss against the previous models and observed an improved mean SSIM score for predictions on clean test images from 0.940 to 0.980, and reduced SSIM score standard deviation from 0.040 to 0.020. Furthermore, we saw no quantitative or qualitative reduction in restoration quality in predictions on dustified images compared to the previous models.

6.1 Future work

The main purpose of conducting this project was to evaluate the viability of deep learning as a step towards fully automating the film restoration process. We developed an analytical approach to neural network training through interpretation of and direct intervention into how a network learns to deconstruct and reconstruct image data. The results of the performed experiments strongly suggest that state-of-the art deep learning approaches can be feasibly applied to photographic film restoration. However, due to the limited scope of this project, we recognise that there is further work involved in bridging the gap between our experimental set up, and real life analogue film restoration.

6.1.1 Wild data set

Collecting a wild data set, i.e. a data set of images with real-life analogue artefact damage, would allow for the approach to be evaluated more rigorously. This could be done in collaboration with professional and amateur film photographers, as well as with the help of heritage preservation organisations. Such a data set would also allow us to take into account film grain preservation when evaluating our approach – since the images we used were scraped from Flickr, we have no control over variables present in the digitisation process, for example DPI when scanning the slides (or negatives). We also have little information about any digital post-processing performed on the digitised images. A diverse wild data set would improve the validity of the performed evaluation, and potentially help identify ways in which the framework can be improved.

6.1.2 Colourisation

Our proposed perceptual loss layer weights for artefact removal were compared to the approach used by DeOldify (Antic 2020), which was developed for the task of colourisation. Colourisation

is highly dependent on semantic knowledge, so the usage of deeper VGG16 layers for the loss formulation are intuitive and justified. However, our experiments suggest that including the earlier layers into the perceptual loss had a positive effect in that it reduced colour shifts in RGB predictions. Furthermore, since we compared our approach to DeOldify in the context of artefact removal, it would be only fair to compare the two approaches in terms of colourisation as well.

We conducted some initial experiments using our modified perceptual loss for colourisation, however, evaluating them was beyond the scope of this project. For these experiments, we used CIE-Lab colour space, as it separates the intensity and colour information in images, while RGB does not.

6.1.3 NoGAN

NoGAN was developed for DeOldify as perceptual loss training was causing the models to make overly "safe" predictions by colouring objects in (numerically) average colors, such as brown. GAN training pushed the colourisation network towards being able to produce more realistic colourisation results. Colourisation is a more unconstrained compared to artefact removal, as there are often several "correct" colours for an object in an image, and we found that we achieved satisfactory results without incorporating adversarial training into our pipeline. Nevertheless, as a future suggestion, adversarial training can be implemented in conjunction to or as an alternative to our modified perceptual loss.

6.1.4 Further investigation into the encoder

In our initial experiments we observed that a pre-trained ResNet34 encoder provided a good starting point for the network to learn to perform artefact repair, but the network still required fine-tuning and a tailor-made perceptual loss to achieve satisfactory results. Another interesting question which arises from these experiments is whether an encoder trained on natural image data could be leveraged in alternative ways for the task of artefact repair. A pre-trained encoder already knows how to represent natural image features, where artefacts such as dust and scratches are, at the very least, uncommon. If we wish to repair an analogue image so that such artefacts are removed, there is a possibility that the feature space of the encoder is sufficient to perform such a task. One interesting approach which might be related to this is Deep Image Prior by Ulyanov et al. (2018). In the context of image generation, priors restrict the output image so that it is realistic and with minimised noise. The authors substitute the process of learning a prior with optimising the model parameters using gradient descent over a randomly initialised latent vector to generate the output image[1]. A comparison is demonstrated between the change in loss when optimising "realistic" images and noisy images to show that for convolutional networks it converges much more slowly for noisy input - arriving at the paper's main idea: halting training before the model has fitted the noise of the input to produce a restored and realistic output.

6.1.5 Stability loss

A final suggestion for future work inspired by the null hypothesis experiments in this project is defining a stability loss. Given that a restoration network should learn to approximate the same clean image whether a clean or a damaged version of it are forwarded through the model, this could be formalised as an additional loss term which will penalise the network if it predicts different outputs for the same image if damage is present, compared to if no damage being present. To do so, during a forward pass in the training phase, we could pass the target through the network, and calculate some distance between the prediction on the damaged input and the prediction on the clean output.

[1] Recall that this is exactly what we did in order to visualise filters from the VGG16 loss network.

A Network modules summary

This section outlines the basic building blocks of the networks that have been used in this project. Implemented using PyTorch and Fast.ai.

A.1 Basic residual block

```
BasicBlock(
          (conv1): Conv2d(64, 64, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
        )
```

A.2 Bottleneck residual block

```
BasicBlock(
          (conv1): Conv2d(64, 128, kernel_size=(3, 3),
          stride=(2, 2), padding=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
          (downsample): Sequential(
            (0): Conv2d(64, 128, kernel_size=(1, 1),
            stride=(2, 2),  bias=False)
            (1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
            affine=True, track_running_stats=True)
          )
        )
```

A.3 U-Net block

```
UnetBlock(
```

```
      (shuf): PixelShuffle_ICNR(
        (conv): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1),
          stride=(1, 1))
        )
        (shuf): PixelShuffle(upscale_factor=2)
        (pad): ReplicationPad2d((1, 0, 1, 0))
        (blur): AvgPool2d(kernel_size=2, stride=1,
        padding=0)
        (relu): ReLU(inplace=True)
      )
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
      (conv1): Sequential(
        (0): Conv2d(192, 96, kernel_size=(3, 3), stride=(1, 1),
        padding=(1, 1))
        (1): ReLU(inplace=True)
      )
      (conv2): Sequential(
        (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1),
        padding=(1, 1))
        (1): ReLU(inplace=True)
      )
      (relu): ReLU()
    )
```

A.4 Self-attention block

```
Sequential(
        (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1),
        padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): SelfAttention(
          (query): Conv1d(384, 48, kernel_size=(1,), stride=(1,),
          bias=False)
          (key): Conv1d(384, 48, kernel_size=(1,), stride=(1,),
          bias=False)
          (value): Conv1d(384, 384, kernel_size=(1,), stride=(1,),
          bias=False)
        )
      )
```

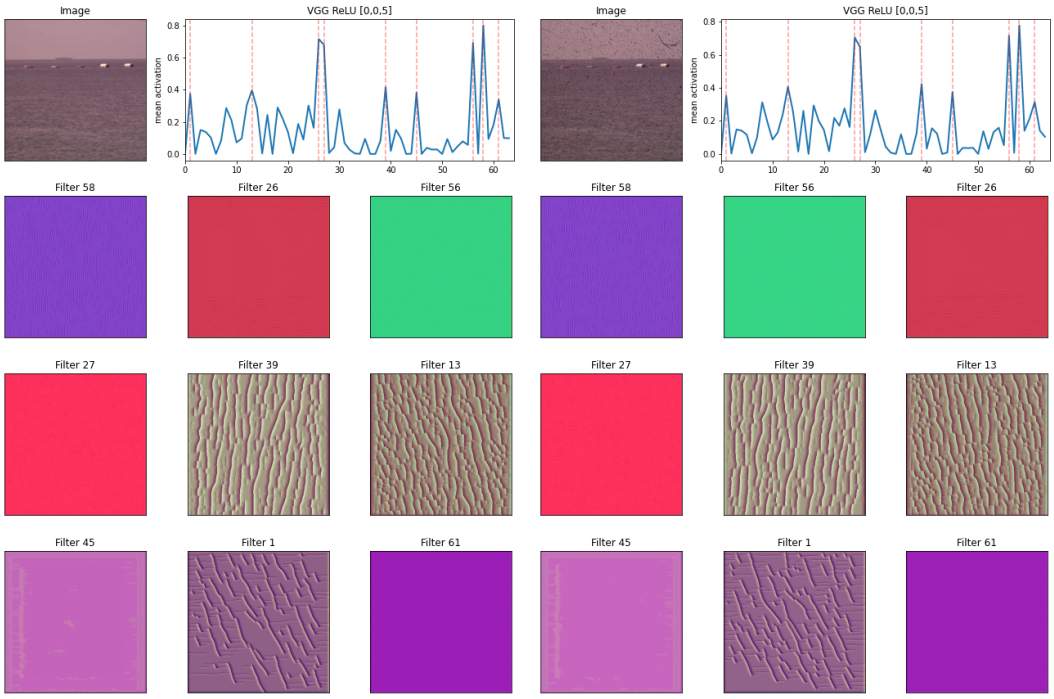
A.5 Custom classifier head

```
  (3): Dropout2d(p=0.15, inplace=False)
  (4): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(4, 4), stride=(1, 1),
    padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): SelfAttention(
      (query): Conv1d(1024, 128, kernel_size=(1,),
```

```
    stride=(1,), bias=False)
    (key): Conv1d(1024, 128, kernel_size=(1,),
    stride=(1,), bias=False)
    (value): Conv1d(1024, 1024, kernel_size=(1,),
    stride=(1,), bias=False)
  )
)
(5): Dropout2d(p=0.15, inplace=False)
(6): Sequential(
  (0): Conv2d(1024, 2048, kernel_size=(4, 4), stride=(1, 1),
  padding=(1, 1))
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
)
(7): Dropout2d(p=0.15, inplace=False)
(8): Sequential(
  (0): Conv2d(2048, 4096, kernel_size=(4, 4), stride=(1, 1),
  padding=(1, 1))
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
)
(9): Sequential(
  (0): Conv2d(4096, 1, kernel_size=(4, 4), stride=(1, 1),
  bias=False)
)
(10): Flatten()
```

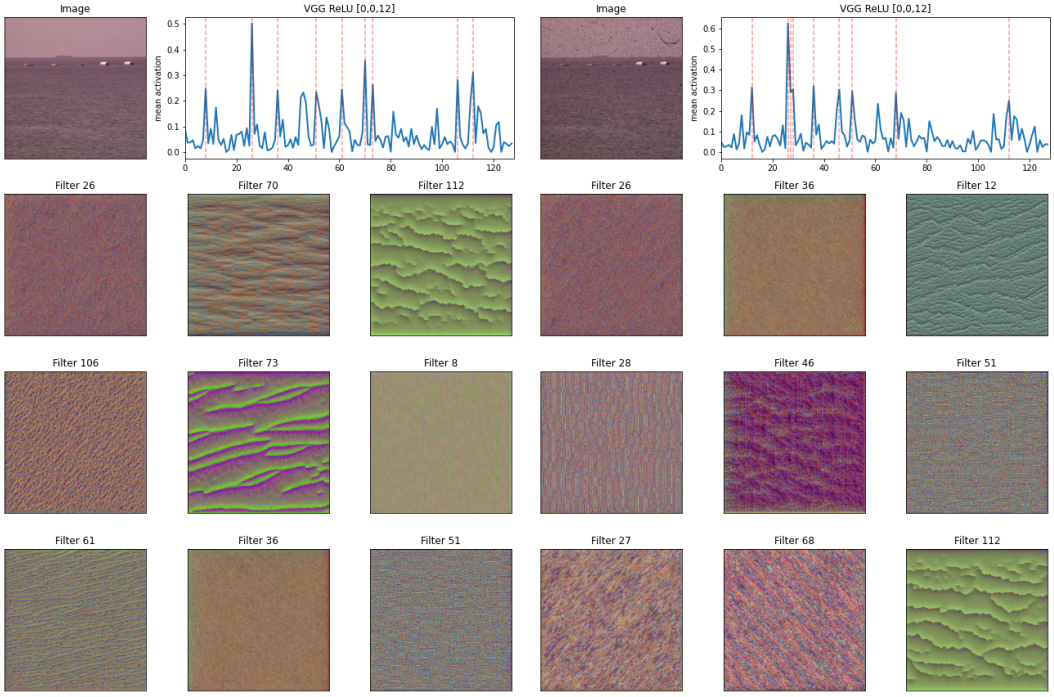
B Visualising VGG16's hidden layers

As part of crafting the feature loss functions for artefact removal, we investigated which of the ReLU hidden layers from the pre-trained VGG16 network are sensitive to analogue artefact image. To do so, we visualise the most relevant filters from each ReLU layer of the VGG16 pre-trained network for three clean-dustified pairs. The following figures are best viewed enlarged. We have chosen two images from the test set with a high degree of (synthetic) damage, those were two of the images for which the model trained using a per-pixel loss had improved the most in terms of SSIM score. Upon visual inspection, we noticed however that it had successfully located the artefacts, but the in-painting that was performed was not seamless. We also included another image with a smaller degree of damage present. We did not include visualisations comparing the activations for two clean images, since naturally, they are exactly the same.



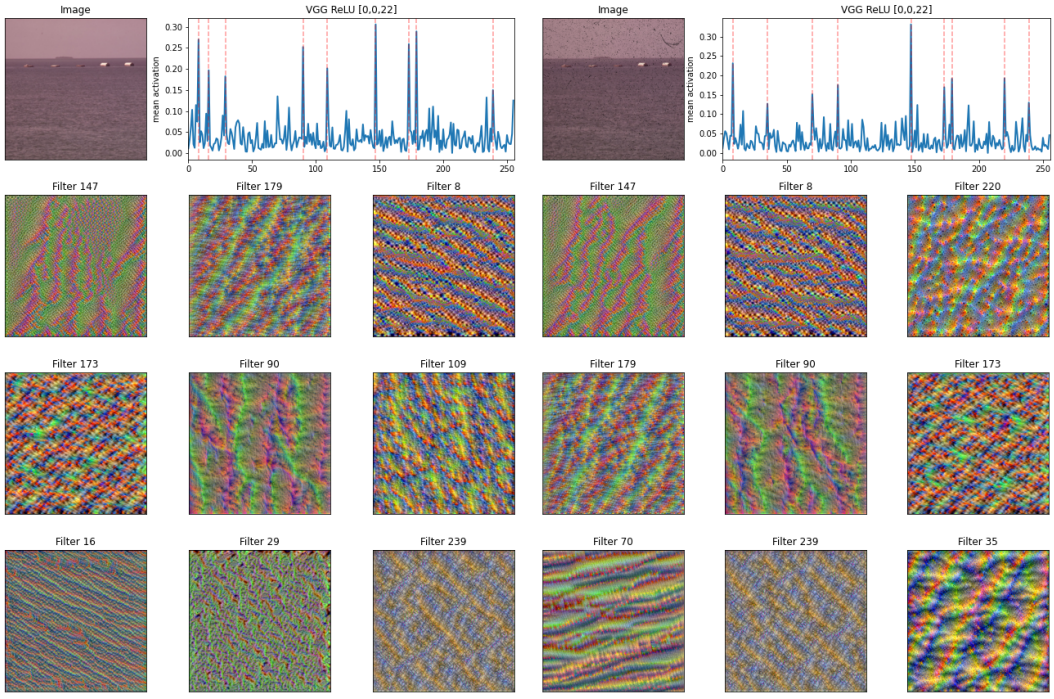
(a) *clean image* ***(b)*** *dustified image*

Figure B.1: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][5].*



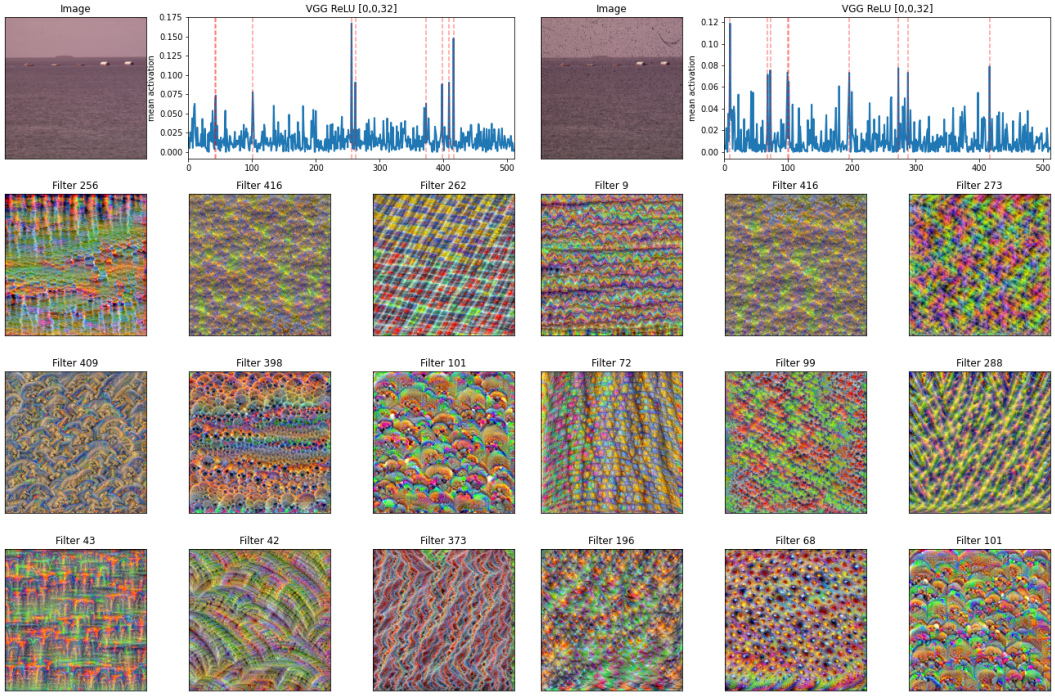
(c) *clean image* ***(d)*** *dustified image*

Figure B.1: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][12].*



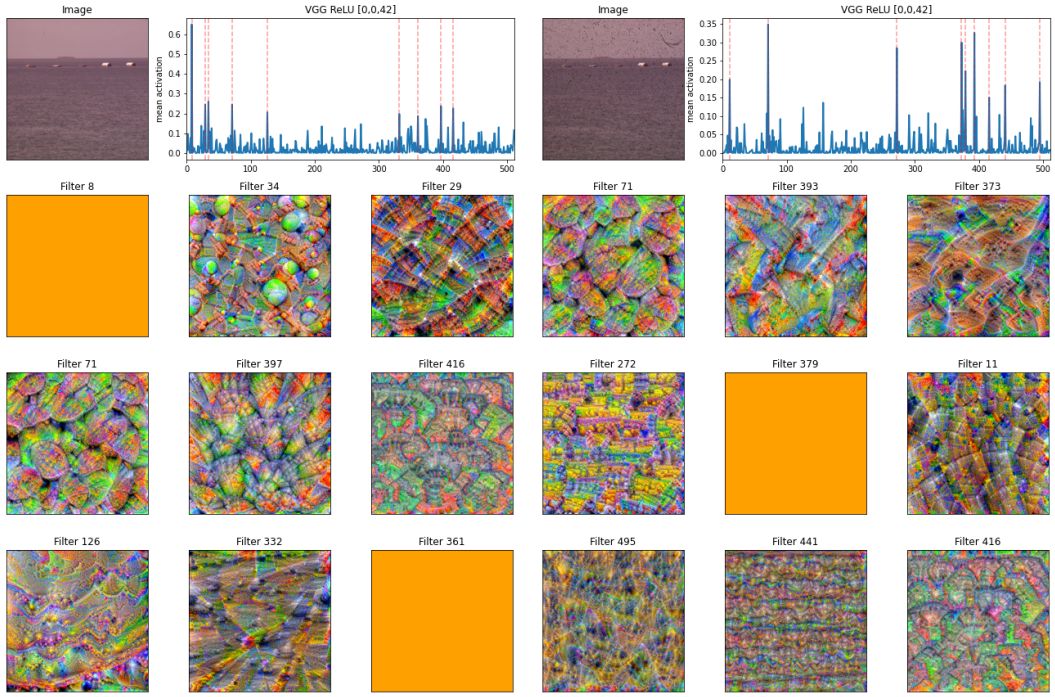
(e) *clean image* ***(f)*** *dustified image*

Figure B.1: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][22].*



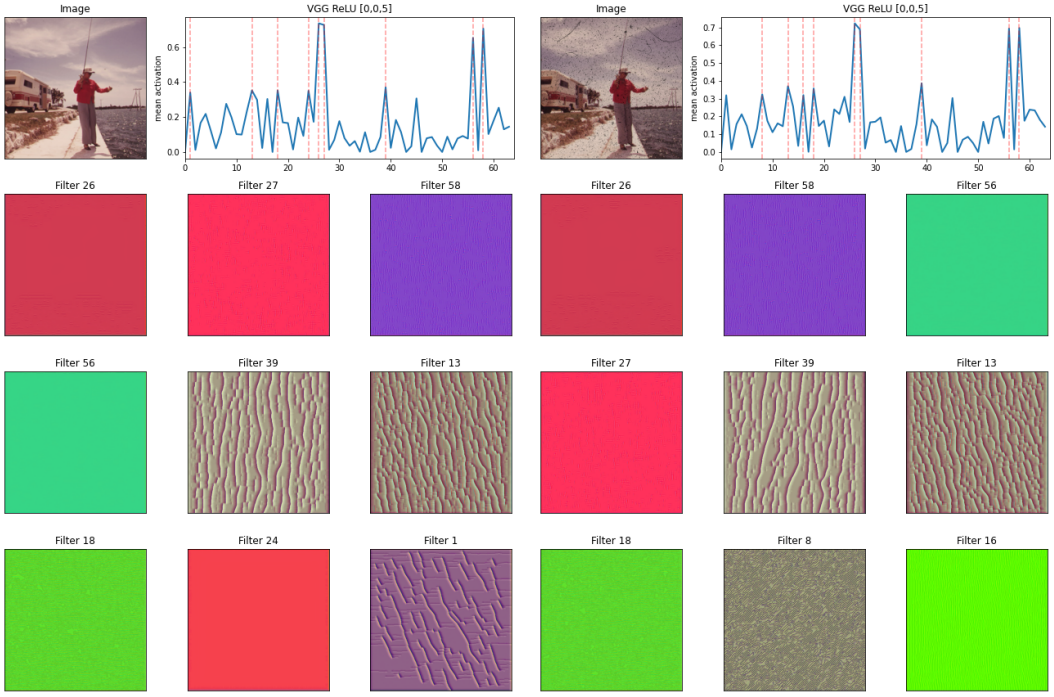
(g) clean image *(h) dustified image*

Figure B.1: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][32].*



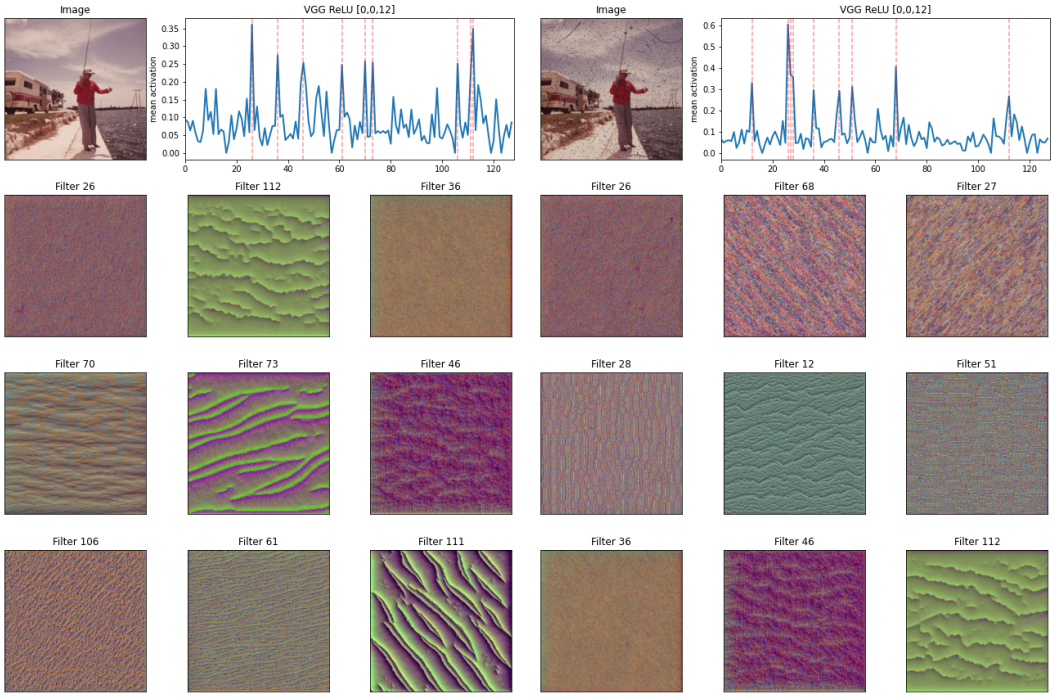
(i) clean image *(j) dustified image*

Figure B.1: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][42].*



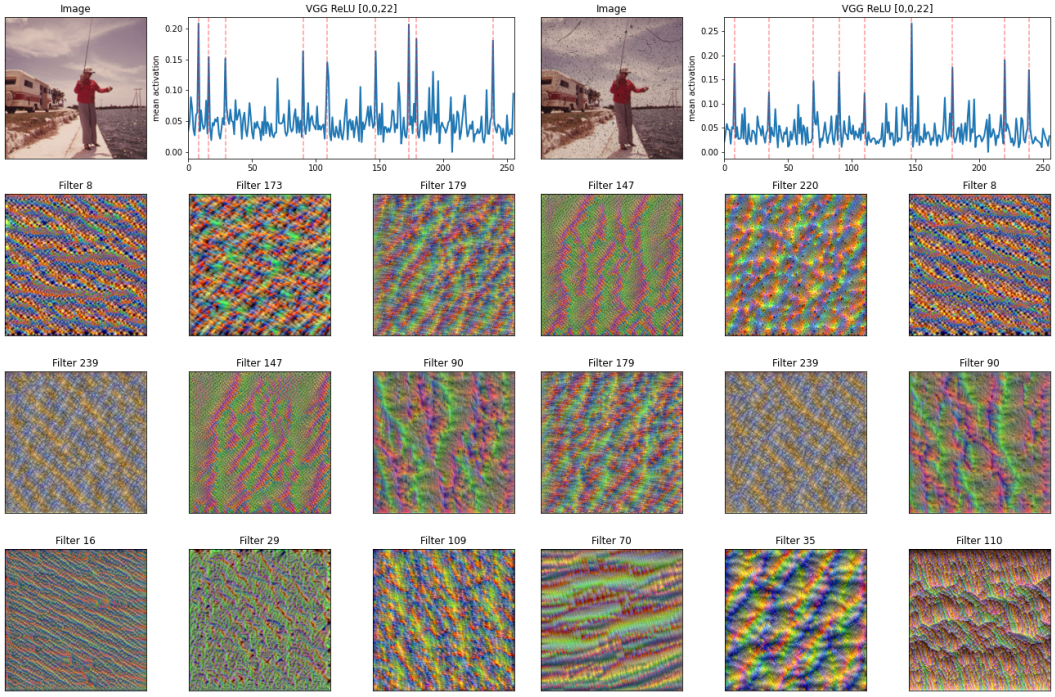
(a) *clean image* ***(b)*** *dustified image*

Figure B.2: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][5].*



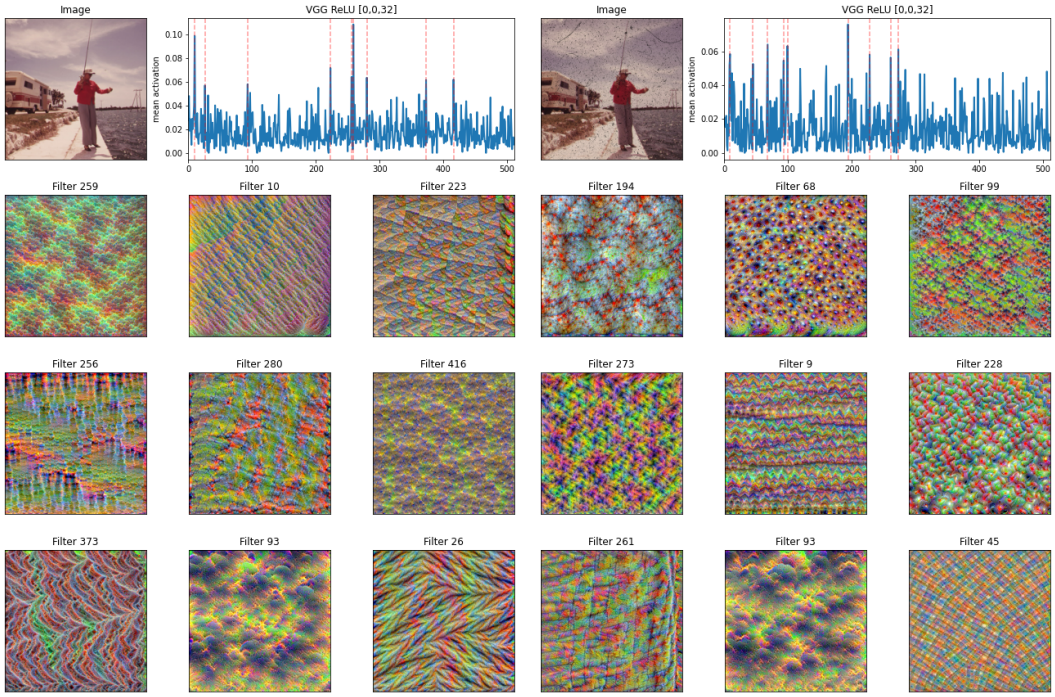
(c) *clean image* ***(d)*** *dustified image*

Figure B.2: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][12].*



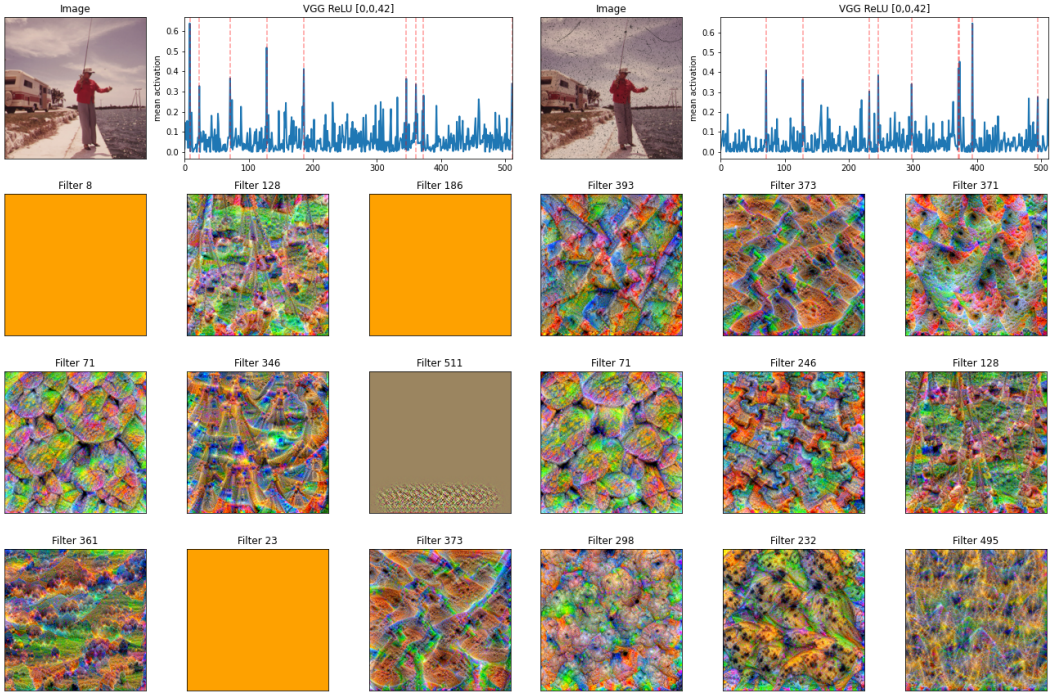
(e) clean image *(f) dustified image*

Figure B.2: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][22].*



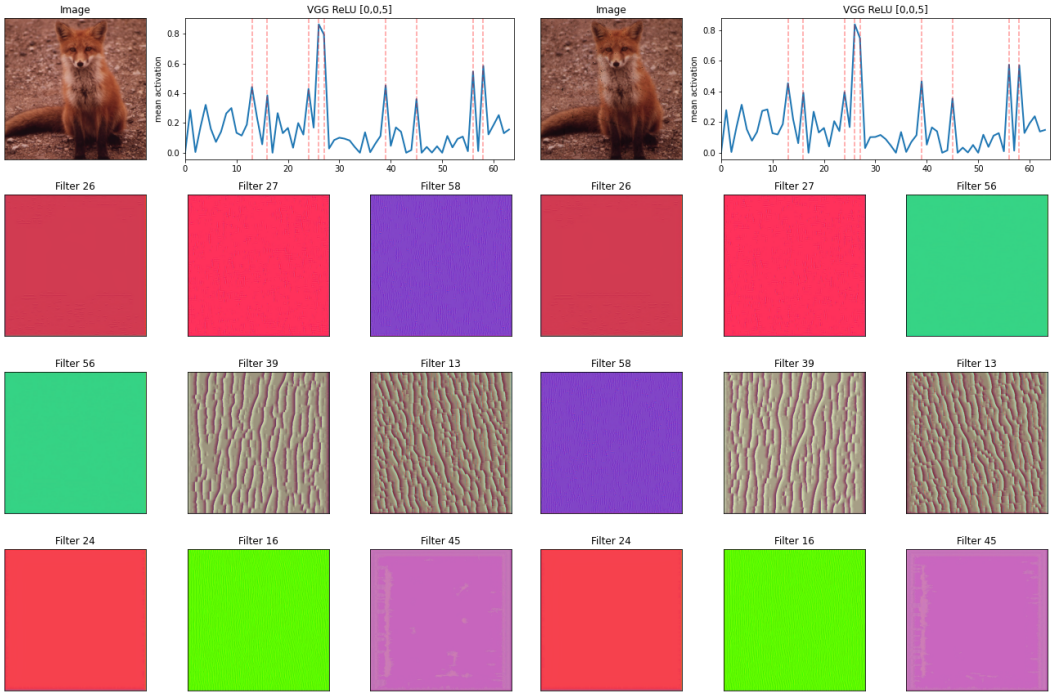
(g) clean image *(h) dustified image*

Figure B.2: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][32].*



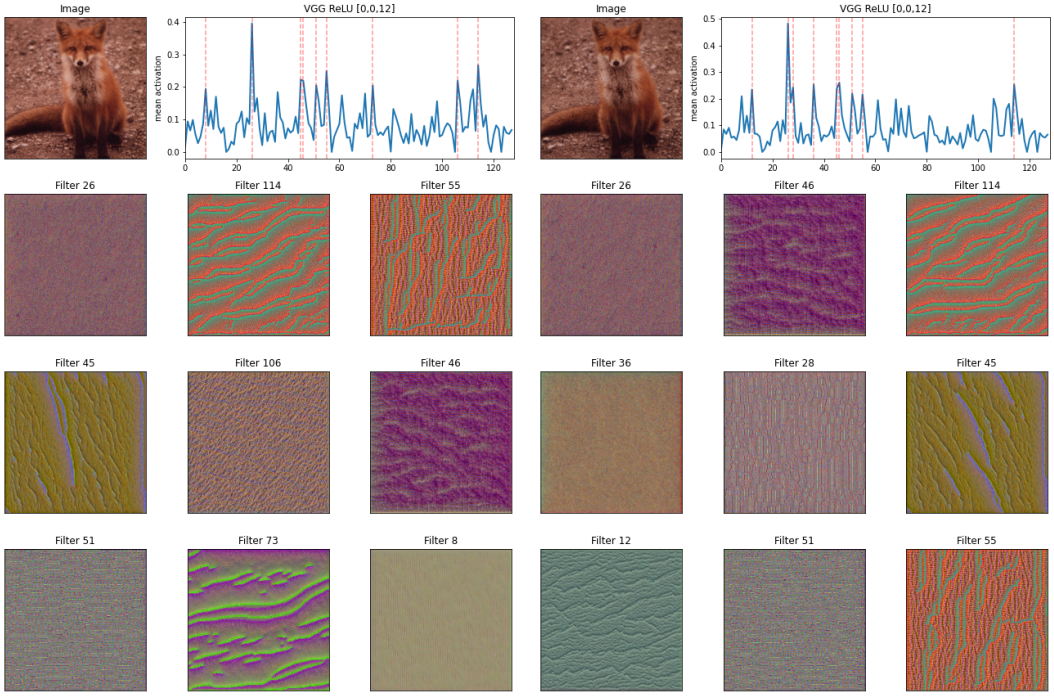
(i) *clean image* ***(j)*** *dustified image*

Figure B.2: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][42].*



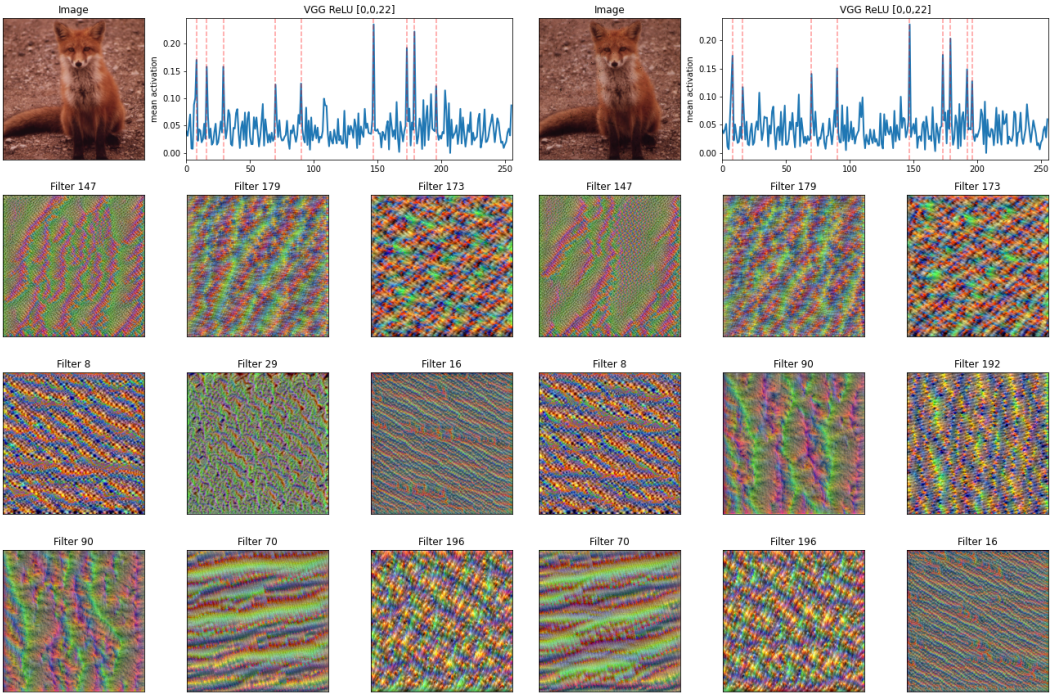
(a) *clean image* ***(b)*** *dustified image*

Figure B.3: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][5].*



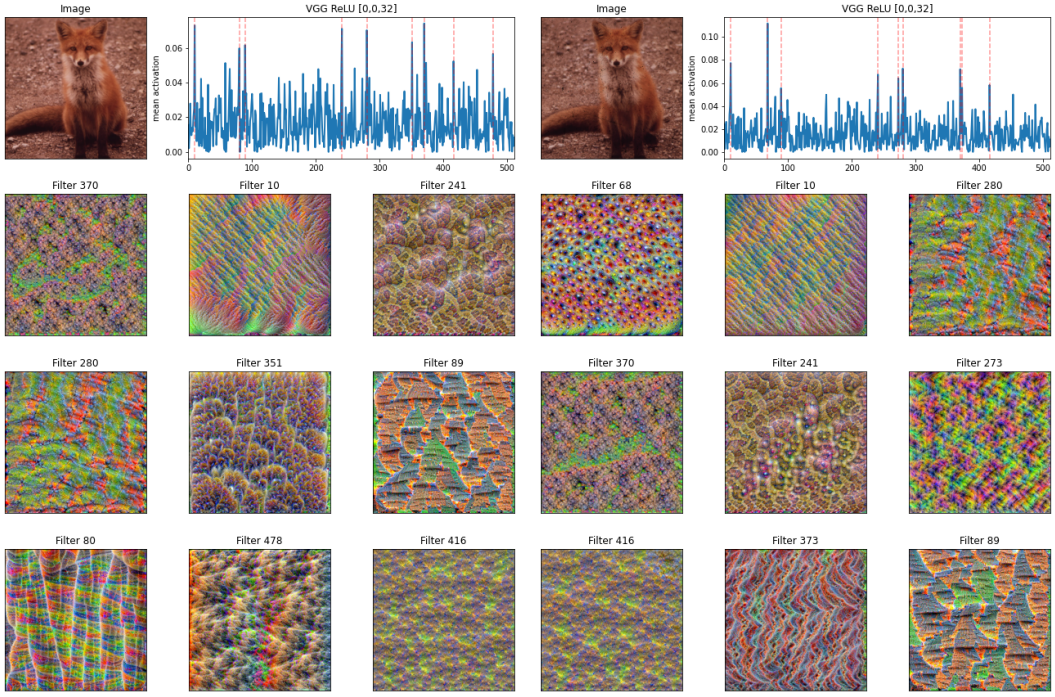
(c) *clean image* ***(d)*** *dustified image*

Figure B.3: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][12].*



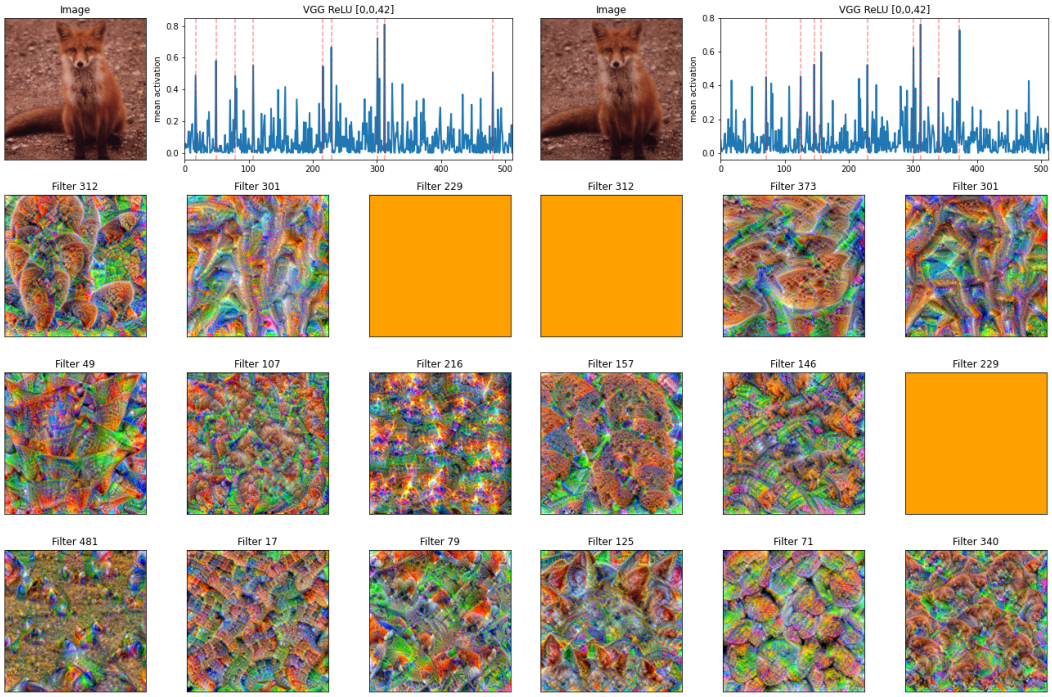
(e) *clean image* ***(f)*** *dustified image*

Figure B.3: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][22].*



(g) clean image *(h) dustified image*

Figure B.3: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][32].*



(i) clean image *(j) dustified image*

Figure B.3: *Top 9 most activated filters from the ReLU layer at VGG16[0][0][42].*

C Training history of Perceptual loss networks

C.1 Training history of Model 1

In Figure C.1 we see the training history of the first model we trained. Unfreezing the encoder improved SSIM scores achieved on the validation set by a small margin - the highest score obtained at the end of the third phase of progressive resizing, before unfreezing the encoder, is 0.8337, compared to 0.841 after unfreezing it and fine-tuning.

Figure C.1: *Training history of Model 1. Each progressive resizing stage takes 10 epochs, while the fine-tuning stage takes 20 epochs.*

C.2 Training history of Model 2

This is the model where we adopt our proposed layer weights. Training history can be seen in Figure C.2. Before unfreezing the encoder, the highest achieved SSIM score over the validation set is 0.8655; fine-tuning improved this score to 0.886.

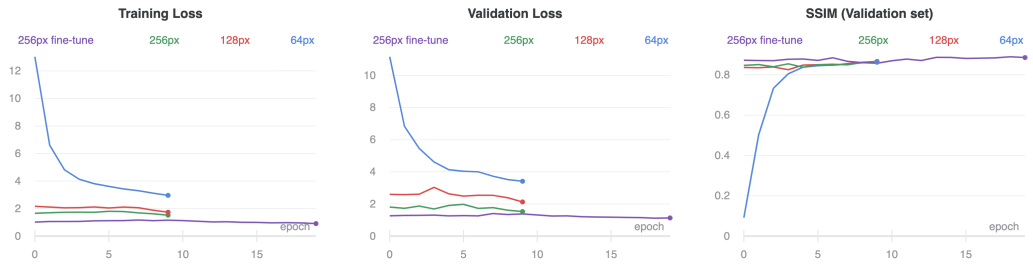


Figure C.2: *Training history of Model 2. Each progressive resizing stage takes 10 epochs, while the fine-tuning stage takes 20 epochs.*

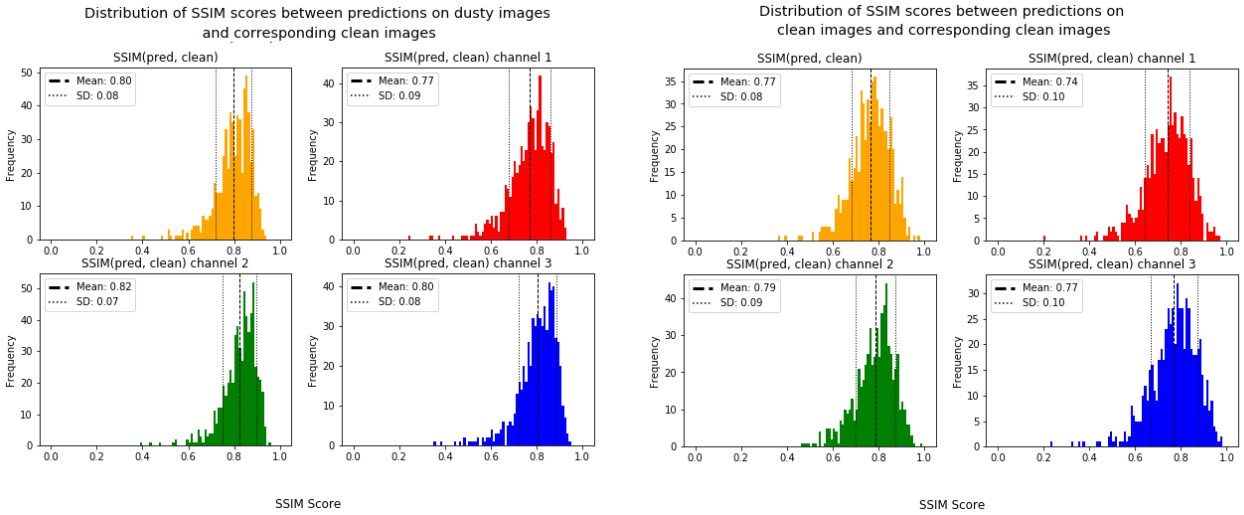
C.3 Training history of Model 3 (ours)

This final model adopts our proposed weights, as well as a SSIM loss term between prediction and target, and SSIM loss (instead of MAE) as a comparison measure between the feature activations from the loss network for the prediction and the target. Training history is visualised in Figure C.3. Highest SSIM score obtained on the validation set before unfreezing the encoder is 0.8923, which is marginally improved to 0.898 after fine-tuning.

Figure C.3: *Training history of Model 3. Each progressive resizing stage takes 10 epochs, while the fine-tuning stage takes 20 epochs.*

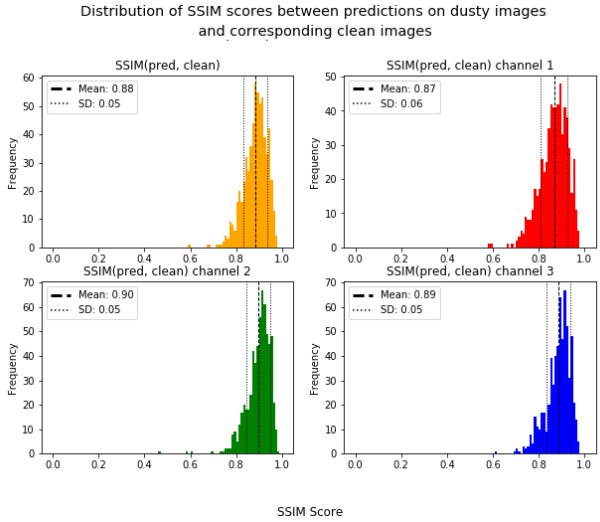
D SSIM scores of Perceptual Loss models

Below are detailed visualisations of the SSIM scores obtained for each of the three perceptual loss models on the test set (624 images).

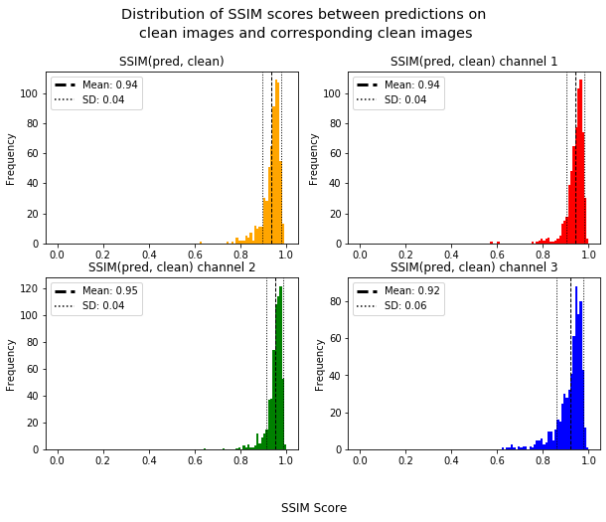


(a) *predictions on dustified images* ***(b)*** *predictions on clean images*

Figure D.1: *Distribution of SSIM scores achieved by Model 1 for predictions on the test set for dustified (D.1a) and clean (D.1b) images. First histogram (yellow) in each group shows overall image SSIM scores, while the other three (red, green, blue) correspond to the SSIM scores for each channel.*

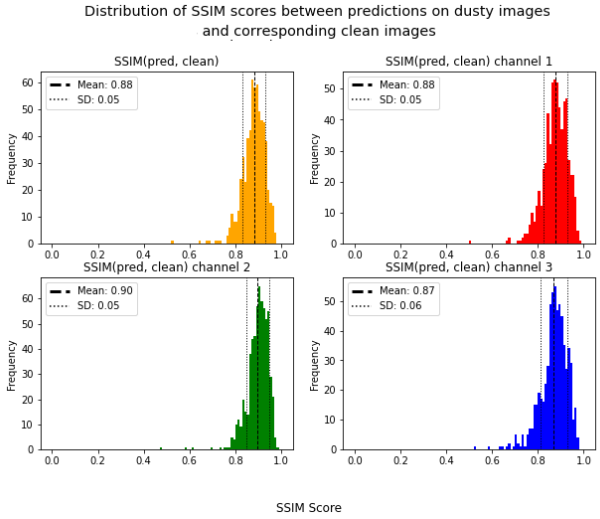


(a) *predictions on dustified images*

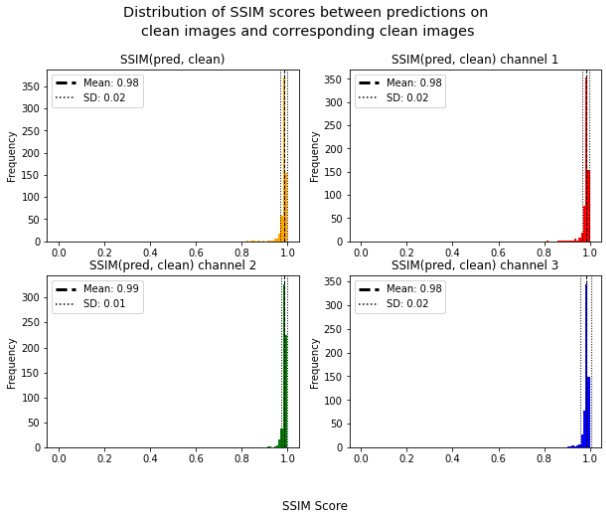


(b) *predictions on clean images*

Figure D.2: *Distribution of SSIM scores achieved by Model 2 for predictions on the test set for dustified (D.2a) and clean (D.2b) images. First histogram (yellow) in each group shows overall image SSIM scores, while the other three (red, green, blue) correspond to the SSIM scores for each channel.*



(a) *predictions on dustified images*



(b) *predictions on clean images*

Figure D.3: *Distribution of SSIM scores achieved by Model 3 for predictions on the test set for dustified (D.3a) and clean (D.3b) images. First histogram (yellow) in each group shows overall image SSIM scores, while the other three (red, green, blue) correspond to the SSIM scores for each channel.*

E NoGAN training

This chapter details work on this project related to the the promising NoGAN approach developed by Antic (2020) for the task of colourisation. As this project progressed we implemented and adapted the approach to the task of scratch removal, however, further evaluation was beyond the scope of the project.

E.1 Background

E.1.1 Generative adversarial networks

Generative adversarial networks (GANs) were introduced in what is now considered as a groud-breaking paper by Goodfellow et al. (2014). The main idea is simple, yet brilliant: the paper describes and successfully implements a framework where two networks, a generator and a discriminator, are trained in competition. The generator is is a network which generates "fake" samples from a given distribution. The discriminator is a CNN used as binary classifier which tries to classify whether an observation is from the training data, or if it was generated. In adversarial fashion, the generator's job is capture the distribution of the training data well enough to be able to create plausible fake samples. Training continues until the discriminator is fooled about half of the time.

GANs laid the groundwork for much of the current research done in the field of deep learning. They have produced impressive results for image-to-image tasks such as superresolution (Ledig et al. 2017). Frameworks which extend the original GAN approach for image-to-image translation, such as CycleGAN (for unpaired examples) (Zhu et al. 2017) and Pix2Pix (for paired examples) (Isola et al. 2017), have also become the front-runners for challenges such as deraining (Meng Tang et al. 2018), de-fogging (Engin et al. 2018), monochrome image colourisation (Isola et al. 2017).

As the number of networks to train simultaneously within the framework increases, so does the demand for computational resources. Additionally, while incredibly powerful, GANs can be notoriously unstable and suffer from an issue called "mode collapse" (Goodfellow 2016).

NoGAN method To improve GAN stability, Jason Antic developed a novel approach used in his DeOldify project for colourising black and white photos, called "NoGAN" (Antic 2020). The approach is based on work developed as part of the Fast.ai massive open online course (MOOC) [1]. The main idea behind the approach is to pre-train the generator and the discriminator, before additionally training them under a GAN setup. Naturally, this helps the networks to converge faster. While the results of the project are not formally evaluated and published in the literature yet, the author reported the networks were substantially easier and quicker to train, and the qualitative results demonstrated are incredibly impressive. The most important takeway from this project, however, is the fact that adversarial training can be performed without the need to train two large networks simultaneously from the ground up.

[1] GAN Superresolution example sing a U-Net generator: *Practical Deep Learning for Coders, v3* Lesson 7: Resnets from scratch; U-net; Generative (adversarial) networks

The loss used to train GANs can be compared to the perceptual losses mentioned in Chapter 2 - the discriminator's aim is to evaluate whether the generator's outputs look, i.e. can be *perceived* as "real" or "fake". In the original GAN paper by Goodfellow et al. (2014), the generator and the discriminator network are trained simultaneously using the same loss function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))], \tag{E.1}$$

where $D(x)$ is the discriminator's estimate of the probability that the real data observation x is real, $E_{x \sim p_{data}(x)}$ is the expected value over all real training items, $G(z)$ is the generator's output based on input z, $D(G(z))$ is the discriminator's estimate of the probability that a generated example is real, $E_{z \sim p_z(z)}$ is the expected value over all input items for the generator. The generator tries to minimise this loss, while the discriminator tries to maximise it. Notice that the generator's output can only affect the second term, so effectively the generator needs to minimise $\log(1 - D(G(z)))$ - therefore, it needs to create such outputs that would cause the discriminator to perceive them as real.

Related to this, in the NoGAN approach developed by Antic (2020), the generator and the discriminator are pre-trained via standard loss functions. Since the discriminator is a binary classifier, it is pre-trained using binary cross-entropy loss (BCE).

E.2 Approach

As described in the GAN overview in the previous section, GAN training involves an additional network, the discriminator, which is trained as a binary classifier that can differentiate between generated and real images. By training it along with the generator, i.e. the restoration network in the context of this project, the discriminator pushes the restoration network to produce outputs with increasing perceptual quality, i.e. such that could fool the discriminator. The architecture of the discriminator used for adversarial training in this project is again based on ResNet34. The last fully-connected layers (the ones responsible for classification) are substituted with a custom "head" for the network. The head implementation is adapted from the discriminator used by Antic (2020), who does not use a pre-trained backbone for the discriminator.

E.3 Implementation

The discriminator implementation is based on a ResNet34 backbone provided by PyTorch, where the last layers have been again removed, similar to the main network implementation. Instead of the removed layers, we attach a custom classifier head with self-attention inspired by Antic (2020). The structure of the custom head is shown in Appendix A.5.

We implement adversarial training using the discriminator network detailed above. We use the NoGAN approach developed by Howard et al. (2018) and Antic (2020). After pre-training the restoration network (generator) as described in Chapters 3, 4, we generate fake images using the training data and save them in the `.../documerica/dataset/` directory. We create the discriminator network through using PyTorch's resnet implementation and appending our own classification layers at the end. We then train the discriminator network for 30 epochs frozen, at learning rate $2e-5$, unfreeze the backbone and train for additional 10 epochs. Adam is again used for optimisation.

NoGAN training is then performed using a Fast.ai Switcher object which takes the discriminator and the generator as arguments. It freezes the generator and trains the discriminator for one step by getting one batch of clean images and generating one batch of restored (fake) images, which the discriminator evaluates as real or fake. The calculated loss is backpropagated to the

discriminator weights, after which it is frozen. The generator is unfrozen and a new batch of restored images is generated. The frozen discriminator evaluates them, but this time the loss is calculated w.r.t. the generator, rewarding it if it manages to fool the discriminator. Our switcher trains the discriminator until its loss is below a 0.65 threshold, then switches back to the generator for one iteration.

Bibliography

Aitken, A., Ledig, C., Theis, L., Caballero, J., Wang, Z. and Shi, W. (2017), 'Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize', *arXiv preprint arXiv:1707.02937* .

Antic, J. (2020), 'Deoldify: A deep learning based project for colorizing and restoring old images (and video!)'.
URL: *https://github.com/jantic/DeOldify*

Biewald, L. (2020), 'Experiment tracking with weights and biases'. Software available from wandb.com.
URL: *https://www.wandb.com/*

Bradski, G. (2000), 'The OpenCV Library', *Dr. Dobb's Journal of Software Tools* .

Cheng, J., Dong, L. and Lapata, M. (2016), 'Long short-term memory-networks for machine reading', *arXiv preprint arXiv:1601.06733* .

Cybenko, G. (1989), 'Approximation by superpositions of a sigmoidal function', *Mathematics of control, signals and systems* **2**(4), 303–314.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. (2009), ImageNet: A Large-Scale Hierarchical Image Database, *in* 'CVPR09'.

Dosselmann, R. and Yang, X. D. (2011), 'A comprehensive assessment of the structural similarity index', *Signal, Image and Video Processing* **5**(1), 81–91.

Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S. and Pal, C. (2016), The importance of skip connections in biomedical image segmentation, *in* 'Deep Learning and Data Labeling for Medical Applications', Springer, pp. 179–187.

Engin, D., Genç, A. and Kemal Ekenel, H. (2018), Cycle-dehaze: Enhanced cyclegan for single image dehazing, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 825–833.

Erhan, D., Bengio, Y., Courville, A. and Vincent, P. (2009), 'Visualizing higher-layer features of a deep network', *Technical Report, Univeristé de Montréal* .

Fannjiang, C. (2011), Better images, fewer samples: Optimizing array configuration for compressed sensing in radio interferometry.

Fukushima, K. (1980), 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biological cybernetics* **36**(4), 193–202.

Funahashi, K.-I. (1989), 'On the approximate realization of continuous mappings by neural networks', *Neural networks* **2**(3), 183–192.

Gatys, L. A., Ecker, A. S. and Bethge, M. (2015), 'A neural algorithm of artistic style', *arXiv preprint arXiv:1508.06576* .

Glorot, X., Bordes, A. and Bengio, Y. (2011), Deep sparse rectifier neural networks, *in* 'Proceedings of the fourteenth international conference on artificial intelligence and statistics', pp. 315–323.

Goodfellow, I. (2016), 'Nips 2016 tutorial: Generative adversarial networks', *arXiv preprint arXiv:1701.00160* .

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014), Generative adversarial nets, *in* 'Advances in neural information processing systems', pp. 2672–2680.

Gore, A. and Gupta, S. (2015), 'Full reference image quality metrics for jpeg compressed images', *AEU-International Journal of Electronics and Communications* **69**(2), 604–608.

He, K., Zhang, X., Ren, S. and Sun, J. (2016*a*), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

He, K., Zhang, X., Ren, S. and Sun, J. (2016*b*), Identity mappings in deep residual networks, *in* 'European conference on computer vision', Springer, pp. 630–645.

Herrmann, F. (2010), 'Randomized sampling and sparsity: Getting more information from fewer samples', *GEOPHYSICS* **75**, WB173–WB187.

Hornik, K. (1991), 'Approximation capabilities of multilayer feedforward networks', *Neural networks* **4**(2), 251–257.

Howard, J. et al. (2018), 'fastai', `https://github.com/fastai/fastai`.

Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K. Q. (2017), Densely connected convolutional networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 4700–4708.

Hubel, D. H. and Wiesel, T. N. (1959), 'Receptive fields of single neurones in the cat's striate cortex', *The Journal of physiology* **148**(3), 574–591.

Hubel, D. H. and Wiesel, T. N. (1968), 'Receptive fields and functional architecture of monkey striate cortex', *The Journal of physiology* **195**(1), 215–243.

Iglovikov, V. and Shvets, A. (2018), 'Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation', *arXiv preprint arXiv:1801.05746* .

Ioffe, S. and Szegedy, C. (2015), 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *arXiv preprint arXiv:1502.03167* .

Isola, P., Zhu, J.-Y., Zhou, T. and Efros, A. A. (2017), Image-to-image translation with conditional adversarial networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1125–1134.

Johnson, J., Alahi, A. and Fei-Fei, L. (2016), Perceptual losses for real-time style transfer and super-resolution, *in* 'European conference on computer vision', Springer, pp. 694–711.

Karras, T., Aila, T., Laine, S. and Lehtinen, J. (2017), 'Progressive growing of gans for improved quality, stability, and variation', *arXiv preprint arXiv:1710.10196* .

Kingma, D. P. and Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kornblith, S., Shlens, J. and Le, Q. V. (2018), 'Do better imagenet models transfer better?', *CoRR* **abs/1805.08974**.
URL: *http://arxiv.org/abs/1805.08974*

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

LeCun, Y. (n.d.), 'The mnist database of handwritten digits', *http://yann. lecun. com/exdb/mnist/* .

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989), 'Backpropagation applied to handwritten zip code recognition', *Neural computation* **1**(4), 541–551.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z. et al. (2017), Photo-realistic single image super-resolution using a generative adversarial network, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 4681–4690.

Lim, B., Son, S., Kim, H., Nah, S. and Mu Lee, K. (2017), Enhanced deep residual networks for single image super-resolution, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition workshops', pp. 136–144.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C. L. (2014), Microsoft coco: Common objects in context, *in* 'European conference on computer vision', Springer, pp. 740–755.

Loshchilov, I. and Hutter, F. (2017), 'Decoupled weight decay regularization', *arXiv preprint arXiv:1711.05101* .

Lu, Y. (2019), The level weighted structural similarity loss: A step away from mse, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 33, pp. 9989–9990.

Maas, A. L., Hannun, A. Y. and Ng, A. Y. (2013), Rectifier nonlinearities improve neural network acoustic models.

Mao, X.-J., Shen, C. and Yang, Y.-B. (2016), 'Image restoration using convolutional auto-encoders with symmetric skip connections', *arXiv preprint arXiv:1606.08921* .

Meng Tang, L., Hong Lim, L. and Siebert, P. (2018), Removal of visual disruption caused by rain using cycle-consistent generative adversarial networks, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 0–0.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G. et al. (2017), 'Mixed precision training', *arXiv preprint arXiv:1710.03740* .

Oliphant, T. (2006–), 'NumPy: A guide to NumPy', USA: Trelgol Publishing. [Online; accessed <today>].
URL: *http://www.numpy.org/*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.
URL: *http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf*

Ramachandran, P., Zoph, B. and Le, Q. V. (2017), 'Searching for activation functions', *arXiv preprint arXiv:1710.05941* .

Ronneberger, O., Fischer, P. and Brox, T. (2015), U-net: Convolutional networks for biomedical image segmentation, *in* 'International Conference on Medical image computing and computer-assisted intervention', Springer, pp. 234–241.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986), 'Learning representations by back-propagating errors', *nature* **323**(6088), 533–536.

Santurkar, S., Tsipras, D., Ilyas, A. and Madry, A. (2018), How does batch normalization help optimization?, *in* 'Advances in Neural Information Processing Systems', pp. 2483–2493.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D. and Wang, Z. (2016), Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1874–1883.

Shi, W., Caballero, J., Theis, L., Huszar, F., Aitken, A., Ledig, C. and Wang, Z. (2016), 'Is the deconvolution layer the same as a convolutional layer?', *arXiv preprint arXiv:1609.07009* .

Simonyan, K. and Zisserman, A. (2014), 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

Smith, L. N. (2017), Cyclical learning rates for training neural networks, *in* '2017 IEEE Winter Conference on Applications of Computer Vision (WACV)', IEEE, pp. 464–472.

Smith, L. N. and Topin, N. (2019), Super-convergence: Very fast training of neural networks using large learning rates, *in* 'Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications', Vol. 11006, International Society for Optics and Photonics, p. 1100612.

Søgaard, J., Krasula, L., Shahid, M., Temel, D., Brunnström, K. and Razaak, M. (2016), 'Applicability of existing objective metrics of perceptual quality for adaptive video streaming', *Electronic Imaging* **2016**(13), 1–7.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The journal of machine learning research* **15**(1), 1929–1958.

Studer, L., Alberti, M., Pondenkandath, V., Goktepe, P., Kolonko, T., Fischer, A., Liwicki, M. and Ingold, R. (2019), 'A comprehensive study of imagenet pre-training for historical document image analysis', *CoRR* **abs/1905.09113**.
URL: *http://arxiv.org/abs/1905.09113*

Su, E. (2017), 'pytorch-ssim', `https://github.com/Po-Hsun-Su/pytorch-ssim`.

Sugawara, Y., Shiota, S. and Kiya, H. (2018), Convolutional neural networks without any checkerboard artifacts, *in* '2018 26th European Signal Processing Conference (EUSIPCO)', IEEE, pp. 1317–1321.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015), Going deeper with convolutions, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1–9.

Thomson, A. M. (2010), 'Neocortical layer 6, a review', *Frontiers in neuroanatomy* **4**, 13.

Ulyanov, D., Vedaldi, A. and Lempitsky, V. (2018), Deep image prior, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 9446–9454.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. (2017), Attention is all you need, *in* 'Advances in neural information processing systems', pp. 5998–6008.

Wang, Z., Bovik, A. C., Sheikh, H. R. and Simoncelli, E. P. (2004), 'Image quality assessment: from error visibility to structural similarity', *IEEE transactions on image processing* **13**(4), 600–612.

Zeiler, M. D. and Fergus, R. (2014), Visualizing and understanding convolutional networks, *in* 'European conference on computer vision', Springer, pp. 818–833.

Zhang, H., Goodfellow, I., Metaxas, D. and Odena, A. (2018), 'Self-attention generative adversarial networks', *arXiv preprint arXiv:1805.08318* .

Zhang, R., Isola, P. and Efros, A. A. (2016), Colorful image colorization, *in* 'European conference on computer vision', Springer, pp. 649–666.

Zhang, R., Zhu, J.-Y., Isola, P., Geng, X., Lin, A. S., Yu, T. and Efros, A. A. (2017), 'Real-time user-guided image colorization with learned deep priors', *ACM Transactions on Graphics (TOG)* **9**(4).

Zhu, J.-Y., Park, T., Isola, P. and Efros, A. A. (2017), Unpaired image-to-image translation using cycle-consistent adversarial networks, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2223–2232.