Deep Learning in Digital Image Processing and Photography

Karol Wojtulewicz Daniel Jonsson Emil Luusua
Tobias Löfgren Joel Oskarsson Daniel Roos

December 2019

1 Introduction

Image manipulation through use of deep neural networks is nothing new in the 21st century. In 2012, when AlexNet [4] demonstrated that deep neural networks are more than suitable for image classification, the trend of using neural networks for various complex problems started. Two examples of such problems in image processing are image denoising and photo enhancement, both of which are explored in this report.

The task of denoising concerns improving the quality of an image by performing inpainting and removal of noise. In this work two state-of-the-art solutions utilizing neural networks, *PixelRL* [2] and *Noise2Noise* (N2N) [5], are evaluated and compared with a simple self-implemented autoencoder.

Photo enhancement is typically performed by professional photographers in order to make their photos more aesthetically pleasing. Since photography is a subjective art form requiring a lot of creativity it is not a task typically suited for machine learning. Therefore successfully applying machine learning on the task would be a great achievement in pushing the boundaries of what artificial intelligence is capable of learning. One attempt at this was made in 2017 by Google researchers Fang and Zhang who developed *Creatism* [1], a system based on deep learning designed to perform photo enhancement at a professional level. They manage to accomplish this by dividing the photo enhancement process into adjustment of several aesthetic aspects, each of which can be optimized individually. This enables neural networks to be trained in order to handle a single straightforward aspect, which they are very well-suited to do. In this work, Creatism has been re-implemented from scratch with some modifications in an attempt to evaluate its effectiveness.

The remaining chapters of this report are organized as follows. Chapter 2 will present used algorithms in further detail as well as the experiments conducted and data sets used to evaluate these. Chapter 3 will present the experimental results and chapter 4 will provide an analysis and discussion of these findings. Chapters 2 and 3 are both be divided into two parts, treating denoising and photo enhancement separately.

2 Methods

Below the different methods used are explained for reproducible purposes. The chapter is divided into two parts: denoising and photo enhancement.

2.1 Denoising

As previously mentioned, we begun by evaluating some existing, state of the art, implementations for denoising: PixelRL [2] and N2N [5].

For evaluation of PixelRL and N2N, pretrained networks trained by the authors of the papers were used. The reason for this is the unreasonable amount of time and computational power required to train these networks on our own. It also gave us a reasonable delimitation for the project as a whole: the pretrained N2N network trained on Gaussian noise allowed for a maximum noising of standard deviation 0.2. Based on this, we limited ourselves to using a Gaussian noiser with standard deviation of 0.1. For evaluation, a unique seed was chosen for the noiser for all algorithms tested, resulting in a comparable result cross-implementation.

2.1.1 PixelRL

PixelRL stands for Reinforcement Learning with Pixel-wise Rewards which uses a fully convolutional network in order to detect and remove noise, restore images and enhance colors in the image [2]. PixelRL extends *deep RL* [6] which is previous work in the area that many people think changed and improved many things in the area of image processing using deep learning. They have made this implementation to work on other various applications, by reason of deep RL. Earlier works have been too limited with regards to how these methods only can execute global actions for the entire image and are limited to simple applications, e.g. image cropping and and global color enhancement.

PixelRL introduces the concept of each pixel in the image, corresponding to a pixel value, having its own agent that can perform 9 possible actions in order to change this and surrounding pixel values. What actions to take is decided by a *Reward Map Convolution*. More details can be read in [2]. Available actions are such as increasing or decreasing pixel value, adding a 5x5 median or gaussian filter etc. The PixelRL implementation only works for grayscale images for image denoising. The PixelRL program and evaluation was performed without `convGRU` or `reward map convolution` on a pretrained network trained on the *BSD68*[1] and *Waterloo*[2] exploration data set. The reason for this was that it was proposed by the authors and simplicity. This means the network was trained and tested on different data sets, which may have an impact on the results. The implementation extends the *A3C* algorithm which is a popular hybrid CPU/GPU implementation. More information and the code is available

[1] https://github.com/clausmichele/CBSD68-dataset
[2] https://ece.uwaterloo.ca/ k29ma/exploration/

on github[3].

2.1.2 Noise2Noise

Noise2Noise is based on the U-Net architecture with channel concatenation between the encoder and decoder (skip architecture) to preserve the image structure [10]. Evaluation of the N2N-implementation was performed using the pretrained
`network_final-gaussian-n2n` network, meaning the network is trained for Gaussian noise using noisy images as ground truth (a speciality of the N2N algorithm; see the paper [5] for more detail).

2.1.3 Custom implementation

Architecture Similarly to N2N, the custom autoencoder follows the U-Net architecture. It is also a fully convolutional network, meaning that it only contains convolutional layers with adaptive filter weights.

Table 1 and Figure 1 show the layer stack of the network.

Training The training was performed using the

$$L_2 = \sum_{i=1}^{n} (y_{true} - y_{pred})^2 \tag{1}$$

loss function and the Adam optimizer. A batch size of 64 was used, and the network was trained for 30 epochs. As training data, the `Cifar10`[4] dataset was used. These images are 32x32 pixles in size, which ends up becoming a delimitation for the network; it cannot in its current implementation handle images larger than this.

The network was implemented in TensorFlow[5] through Google Colab[6], which delivers 24GB ram and more than 15GB of GPU memory.

[3] https://github.com/rfuruta/pixelRL
[4] https://www.cs.toronto.edu/ kriz/cifar.html
[5] https://www.tensorflow.org/
[6] https://colab.research.google.com/

Layer Type	Filter Size	Stride	Activation	Output Shape
Input	-	-	-	3x32x32
Convolutional	5	1	ReLU	32x32x32
Max Pooling	2	1	ReLU	32x16x16
Convolutional	5	1	ReLU	64x16x16
Max Pooling	2	1	ReLU	64x8x8
Convolutional	5	1	ReLU	128x8x8
Max Pooling	2	1	ReLU	128x4x4
Deconvolution	2	1	ReLU	64x8x8
Convolutional	1	1	ReLU	64x8x8
Convolutional	1	1	ReLU	64x8x8
Deconvolution	2	1	ReLU	32x16x16
Concatenation	-	-	-	64x16x16
Convolutional	5	1	ReLU	32x16x16
Convolutional	1	1	ReLU	32x16x16
Deconvolution	2	1	ReLU	32x32x32
Concatenation	-	-	-	64x32x32
Convolutional	5	1	ReLU	32x32x32
Convolutional	5	1	ReLU	3x32x32
Convolutional	1	1	ReLU	3x32x32
Convolutional	1	1	ReLU	3x32x32

Table 1: The custom denoiser architecture

2.2 Photo enhancement

A simplified version of the Creatism [1] system was implemented using the *PyTorch*[7] library. The Creatism system enhances images by sequentially applying a number of image processing operations. These operations all have tunable parameters that determine how they change an image.

The task of choosing these parameters is normally performed by a knowledgeable photographer, but in the Creatism system this is done automatically. The parameters are optimized by using a number of neural networks. Each neural network performs a scoring task, taking an image as input and outputting a score in $[0, 1]$. A different network is used for each image operation, scoring a specific criteria of how the image looks. For example, when applying an operation that changes saturation the saturation-network scores the saturation of the images it is fed.

2.2.1 Enhancement Algorithm

The different operations involved in enhancement are *cropping*, *saturation*, *High Dynamic Range (HDR)* and *tone curves*[8]. Saturation and tone curves are op-

[7] `https://pytorch.org/`

[8] `https://www.mediachance.com/pseam/help/curves.html`

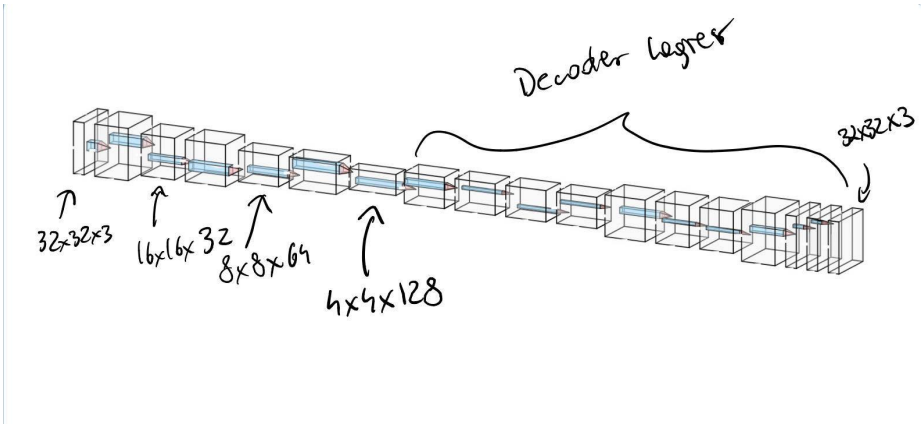


Figure 1: The custom denoiser architecture visualized

erations that change the lighting settings of the image. HDR merges different exposures of an image to get as much intensity details as possible.

After the networks for the different image operations have been trained, enhancement can begin: First off, test images that the network has not trained on are used for enhancement. Secondly, a fixed number of different crops are generated for the image. This is done by trying multiple different aspect ratios and fractions of the original image height. All of these different crop proposals are propagated through the cropping network and the n best scored crops are selected. All other operations are then applied to the selected crops, sequentially optimizing the parameters of each operation by using the networks for scoring. The aesthetic ranking network is finally used in order to rank the images based on their overall look in order to find the best photo. When the best one has been chosen, that parameter configuration is applied to the image at full resolution to generate the final enhanced image.

2.2.2 Training Datasets

Semi-professionally edited lanscape photos were scraped from Flickr[9] using the Flickr API[10]. Used search terms were *beautiful landscape*, *colorful landscape* and *nature landscape* to retrieve 10500 unique photos. After cleaning out non-edited photos and photos with watermarks this resulted in a dataset consisting of 8337 photos. Approximately 90% of these (7399) were used for training and the remaining (938) for validation.

Following the approach utilized by Fang and Zhang [1], each of the photos was randomly perturbed six times to create negative training examples. This procedure was conducted separately for each of the different image operations by randomly sampling its tuning parameters and applying the operation; yielding 4

[9] `https://www.flickr.com/`
[10] `https://www.flickr.com/services/api/`

datasets of 51793 training and 6566 validation examples each (6 perturbations + the original photo). The photos were also resized to a size of 299x299 for cropping and 128x128 for the other 3 datasets.

Since the enhancement algorithm requires the networks to predict a score, targets for the training data needed to be calculated. This was accomplished by assuming the original photos were optimally edited (receiving a score of 1) and measuring how similar the perturbed photo was to the original to determine its score. For cropping, the area ratio defined as $\frac{\text{Area(Perturbed)}}{\text{Area(Original)}}$ was used. For the other three operations, the similarity metric $\text{ReLU}(1 - \frac{\delta}{0.06})$ was used where δ denotes the average percentage pixel difference between the images (yielding a score of zero when the difference exceeds the threshold of 6%).

The aesthetic ranking network was trained on the AVA dataset [7], which contains more than 250000 images with manually annotated scores for photo aesthetics. A target was assigned to each image based on which percentile it would rank in within the dataset (i.e. a target of 0.7 would correspond to the image having a higher average aesthetic score than 70% of images in the dataset).

2.2.3 Training Procedure

In total 5 neural network models were trained, one for each of the enhancement operations and one for aesthetic ranking. Each model was trained using its corresponding dataset, as introduced in section 2.2.2. Apart from this the training loop used was identical for the different networks.

The network architectures used were *InceptionV3* [12], *InceptionTiny128* and *InceptionTiny299*. The last two are scaled down versions of the popular *InceptionV3* network. The full sized network was used in the original Creatism paper [1]. The main differences between the two smaller networks is the input size, one taking images of size 128x128 and the other images of size 299x299. The networks start with a sequence of convolutional layers, max-pooling and batch-normalization. This convolutional part of the network features some of the *Inception-modules* from [12]. This is followed by two fully connected layers. The final layer has only a single unit with a sigmoid activation function, representing the estimated score of an image. A detailed description of the architectures can be found in appendix A.

The networks were trained for regression, trying to predict the score of each perturbed image in the generated datasets. More training details can be found in appendix B.

2.2.4 Experiment setup

Two small datasets were used for evaluating the finished creatism system. The first dataset, referred to as *Gmaps* consists of 29 screenshots from Google Maps streetview[11] panoramas. The *Vacation* datasets consists of a set of vacation photos taken by the authors.

[11] `https://www.google.com/streetview/`

Two enhanced versions of the *Gmaps* dataset were created. The first was enhanced using the implemented Creatism system. The second set was enhanced manually, using only the same operations as Creatism has available and a similar amount of time as the automatic enhancement. A survey was conducted, where participants were shown the two different enhanced versions of an image. They were then asked to select which of the two images they found more aesthetically pleasing.

The *Vacation* dataset was used with two purposes, to showcase the impact of the individual operations as well as to investigate the robustness of the enhancement procedure. The first purpose was fulfilled by simply saving intermediate images after each step in the pipeline and observing these; the second purpose was fulfilled by enhancing the photos using both trained and untrained networks and comparing the results to examine just how much the networks are actually able to learn.

3 Results

3.1 Denoising

Below we present some results for each of the implementations evaluated. Important to note is the lack of a universal metric of evaluation for denoising. Therefore several were considered — e.g. PSNR, SSIM, MSE, NRMSE — but the results were not included in this chapter due to we thinking the images told the clearer story.

3.1.1 Dataset evaluated

As previously mentioned, a common ground truth was chosen for all evaluation. As a sample, Figure 2 shows two of the images from the `Cifar10` dataset used for evaluation. These images were then noised as previously discussed to produce the images in Figure 3.

3.1.2 Noise2Noise

Figure 4 shows the images denoised by Noise2Noise. These results are quite good, especially considering the network has never seen a "clean" image without noise. Especially interesting is how the details in the windshield of the truck are retained after denoising. The truck wheels do however lose some detail.

3.1.3 PixelRL

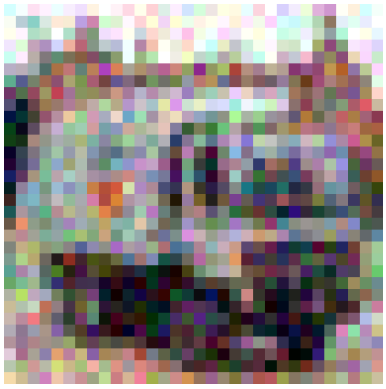
Figure 5 shows the images denoised by PixelRL. As mentioned earlier the implemenation only works for gray scale, which explains why the images looks so different from the others. Otherwise, the results of the pixels are looking relatively accurate when not taking pixel value into regards.

(a) Horse original

(b) Truck original

Figure 2: Before noising

(a) Horse noised



(b) Truck noised

Figure 3: After noising

3.1.4 Own autoencoder

Figure 6 shows the images denoised by our own autoencoder. Compared to N2N, the autoencoder is significantly less aggressive in its denoising, which can for example be seen in the top-right part of the trees in the truck example. The detail in the truck wheel is however (similar to N2N) lost after denoising.

The training and test loss for this implementation can be seen in Figure 7. The indexing is kind of confusing, for two reasons: (1) the indexing starts with 0, meaning the first epoch is epoch 0. (2) the first epoch was excluded from the figure due to its extremely large value compared to the rest of the training. This summarizes to "epoch 5" in the graph actually being the 7^{th} epoch. The epochs will be referred to by their index in this graph below.

No early stopping was implemented, so the network could potentially have continued learning given the time. Most of the improvements was observed

(a) Horse denoised

(b) Truck denoised

Figure 4: Denoised by Noise2Noise

(a) Horse denoised

(b) Truck denoised

Figure 5: Denoised by PixelRL

between the 0^{th} and 2^{nd} epochs, as well as the 4^{th} and 5^{th}. Looking at the progress the network made between these epochs, the majority of the change can be attributed to color correction. The network often started off with no color at all, then gradually added color as the loss was decreasing. Due to how our kernels were implemented, the network sometimes never colored the image at all, resulting in a black-and-white image after training.

Some additional denoising made by our own autoencoder can be seen in Figure 8. Looking at additional these, especially the fourth one, it is clear that our network struggles somewhat with open, monotone, areas with no detail. It performs best on images containing small color variations and clear contours.

(a) Horse denoised (b) Truck denoised

Figure 6: Denoised by our own autoencoder

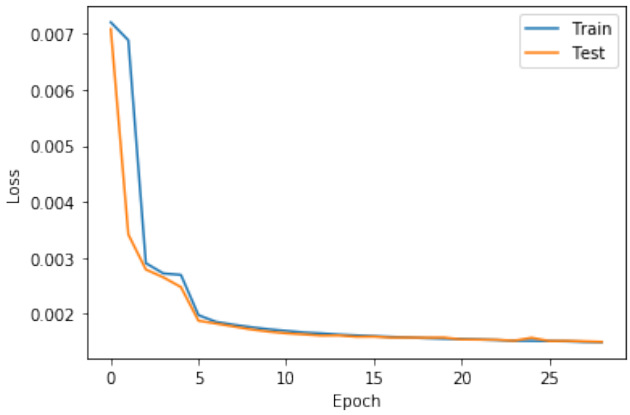


Figure 7: Training and test loss over epochs

3.2 Enhancement

3.2.1 Survey Results

61 persons responded to the survey. The mean value of how many times an individual preferred the Creatism-enhanced version of a photo was 11.84/29 (40.1%), the median 12/29 (41.4%), and the standard deviation 3.16. A histogram depicting the distribution of responses can be seen in figure 9. Here it can be seen that the results vary from 6 to 19 Creatism photos being preferred. Table 6 in appendix D shows the percentage of times each of the photos enhanced by Creatism was preferred, showing that 10/29 (34.5%) of the photos were preferred in a majority of cases.

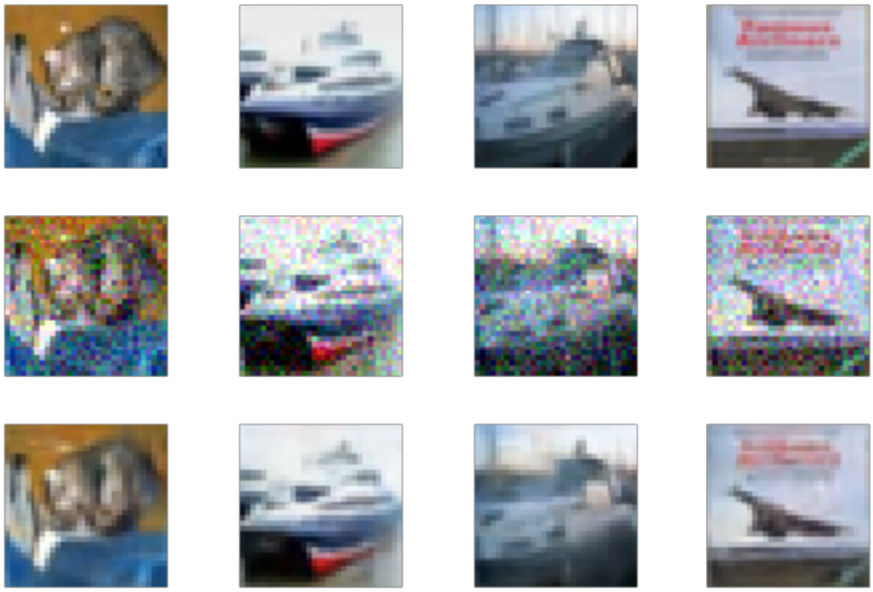


Figure 8: Additional denoising done by our own autoencoder. Note the differences in denoising capabilities for different levels of detail. The first row is the ground truth, the second the noised (Gaussian w/ std = 0.01) versions and the third the denoising done by our network.

3.2.2 Impact of Individual Operations

Figures 10-14 show the same image from the *Vacation* set at each step of the Creatism pipeline. This visualizes the impact of the difference image processing operations. In this particular instance it can be seen that the cropping operation roughly adheres to the rule of thirds by having the sky occupy 1/3 of the vertical space and placing the cliff off-centre. The saturation operation makes the image slightly more colorful and the HDR operation boosts the depth that can be seen in the waves, clouds and the cliff. Lastly the tone curve operation further modifies colors, making the water bluer and sky whiter while simultaneously increasing contrast in the image.

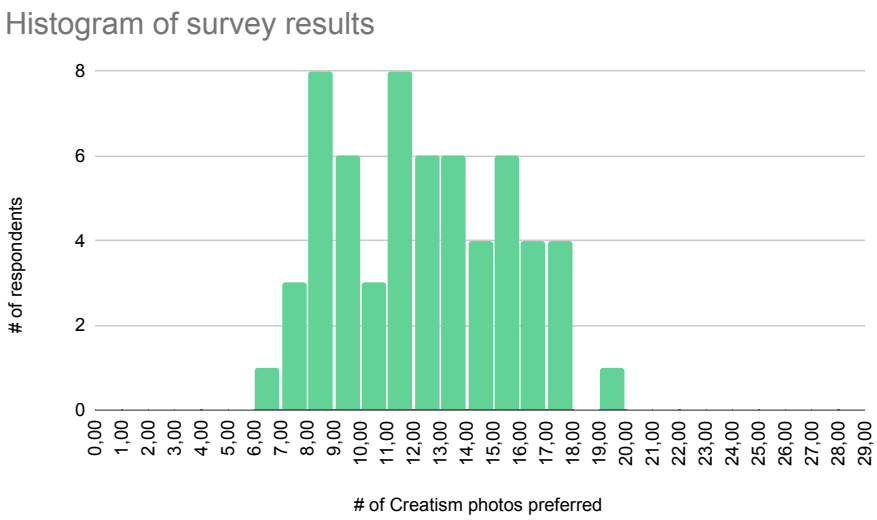


Figure 9: A histogram depicting the survey results.

Figure 10: Original vacation image.

Figure 11: Vacation image after cropping enhancement.

Figure 12: Vacation image after saturation enhancement.

Figure 13: Vacation image after HDR enhancement.

Figure 14: Vacation image after tone curve enhancement.

3.2.3 Enhancement

Table 2 showcases five images from the *Vacation* dataset. It can be seen that the tone curve operation is especially sensitive, since multiple images display an unnatural tint. More enhanced images can be found in appendix C.

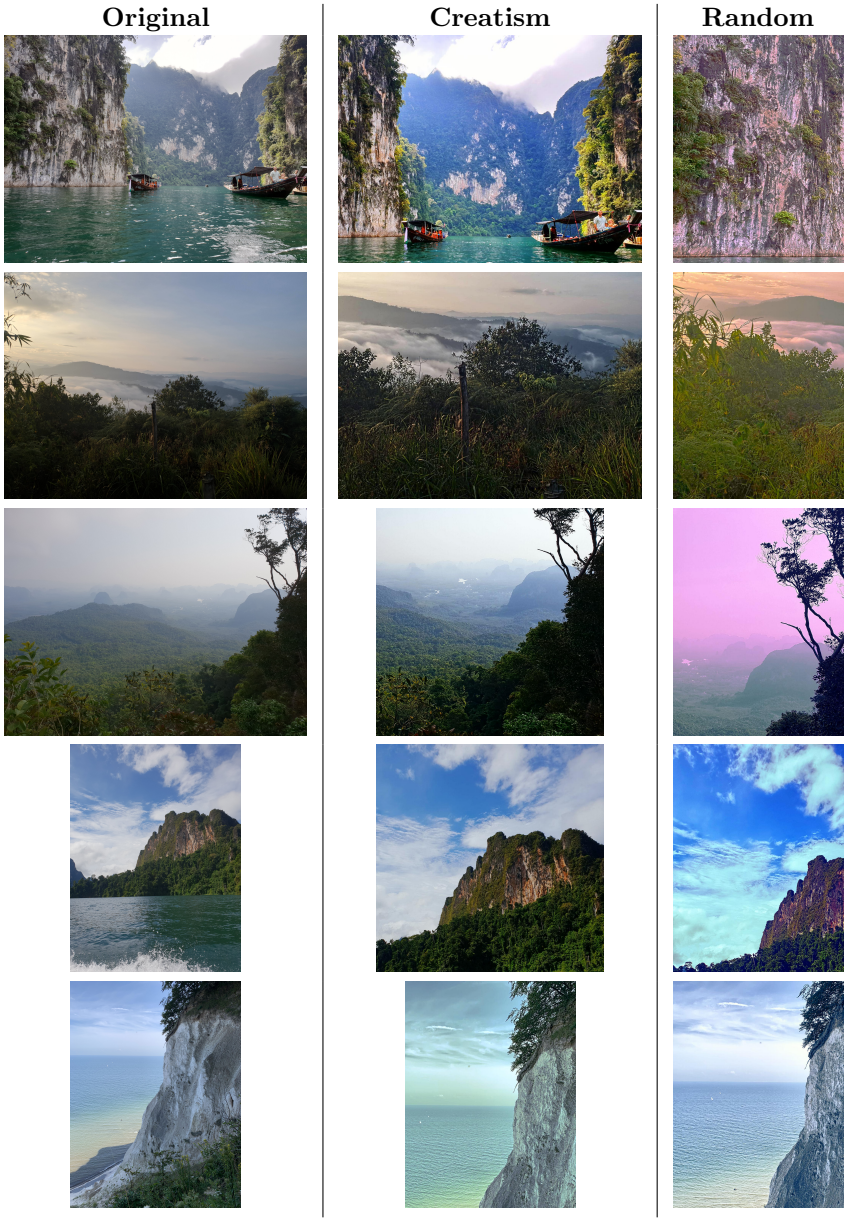


Table 2: *Vacation* images enhanced by Creatism and random parameters.

4 Discussion

This section will discuss different methods and concepts that were used to solve the different problems as well as discussing possible alternations that could be made for further improvement.

4.1 Improvements to the Autoencoder

Even though the autoencoder presented in this report performs well, there are a number of improvements, both in terms of usability and architecture that could be made. There are a number of architectural improvements and regulation methods that could be implemented or have been already tried in this autoencoder. One method, which is known for improving the overall performance of a network is the dropout, where each node instead of being active at all times during training and testing, gets a probability value of it being active during training. This regularization method is known to reduce overfitting by forcing all of the nodes to compensate for the nodes that are turned off at any point in time during training [11]. Dropout was implemented for the autoencoder in the very beginning of the network development.

It did unfortunately result in unclear and blurry images and has therefore not been chosen in the final architecture. What could be considered instead of dropout, that has not been implemented is batch normalization, which normalizes the input to each layer [3]. This method has been proven to decrease the convergence speed as well as improve the accuracy as stated in [3]. The convergence speed could also be improved as well as better results could be obtained by using a non-random weight initialization technique [8].

Another major improvement to the autoencoder would be the ability to input images sized differently than 32x32 pixels. An easy way of implementing this would be to downsize any input image larger than this to 32x32 using for example `skimage.transform.resize`[12].

4.2 Enhancement Results

The results from the conducted survey (section 3.2.1) indicate that the Creatism system performs similar to a human photo editor using the same tools. The survey results give a slight edge to the human enhancer. These results are still quite promising, considering the complexity of creating something that is aesthetically pleasing. Further insight can be gained from the comparison with photos enhanced with random parameters in table 2. It is clear that the networks have learnt some notions of how the operations should be applied to improve the photos.

Observing the changes at each step of the pipeline in figures 10-14 allows for reasoning about the impact of the different observations. The importance of cropping is clear, as would be expected. HDR also performs some substantial

[12]https://scikit-image.org/docs/stable/auto_examples/transform/plot_rescale.html

changes to the image, exaggerating details. Importance of operations varies between images, but these observations extend to most of the *Vacation* dataset.

4.3 Image Processing Operations

The orthogonality of the operations used was not an aspect that was taken into account to a large degree. Saturation, tone curves and HDR were all used and utilizes different techniques for light manipulation in the image. This means that image operations interfere with each other and change the result from previous enhancement. This could be one reason why aesthetic ranking did not show as much promise as hoped.

Creating different perturbations of the HDR images was a challenging task, since we did not have the same access to different HDR filter strengths as in the Creatism work, where they used Snapseed's *HDR filter strength* operations for this purpose. Instead, a fake HDR effect[13] was used that utilizes edge-preserving smoothing filters to generate HDR images [9]. In order to increase this effect on the image, linear interpolation between the original image and the HDR image was used: $\text{HDR}(\alpha) = (1 - \alpha) * image + \alpha * HDR\_image, \alpha \in [0, 1]$. These simplifications might also have had impact on the results.

4.4 Applications and Future Work

Many similar works has been done on the area of image denoising. Given the results and how we were able to create an implementation that is comparable to state of the art works, shows that many improvements are possible given more effort and time available. This works can be applied to many applications, such as surveillance systems or other image processing systems. Future work could further improve how the image denoising system works but also widen the system to work on other applications, such as videos.

The Creatism system provides a basis for many different possible applications. Automatic photo enhancement can be applied on photos taken with mobile phones or digital cameras. This would give users fast access to enhanced versions of their photos without any manual work required. Such automatic enhancement would likely be carried out on a server, so privacy concerns related to the photos would need to be considered.

One of the powers of the Creatism system is that it only applies explicitly defined operations and decides on specific parameters for said operations. This makes the enhancement procedure fully explainable. Contrast this with fully-convolutional neural networks, where the actual changes being made to an image are very hard to decode in an explainable way. Having the system output the best parameters for each operation also opens up for many possible extensions. These parameters can be used to initialize values in photo editing software, allowing a human to then make adjustments. This approach could also be extended to more interactive photo enhancement use cases. For example, a

[13] https://github.com/ray075hl/singleLDR2HDR

dialogue system could be constructed around the Creatism system, allowing a user to give comments like "I think this photo should have a bit higher saturation". Such interactive approaches are interesting directions for future research.

References

[1] Hui Fang and Meng Zhang. Creatism: A deep-learning photographer capable of creating professional work. *arXiv preprint arXiv:1707.03491*, 2017.

[2] Ryosuke Furuta, Naoto Inoue, and Toshihiko Yamasaki. Fully convolutional network with multi-step reinforcement learning for image processing. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.

[6] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[7] Naila Murray, Luca Marchesotti, and Florent Perronnin. Ava: A large-scale database for aesthetic visual analysis. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2408–2415. IEEE, 2012.

[8] Derrick Nguyen and Bernard Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 21–26. IEEE, 1990.

[9] Jae Sung Park, Jae Woong Soh, and Nam Ik Cho. Generation of high dynamic range illumination from a single image for the enhancement of undesirably illuminated images. 2019.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Appendices

A Inception Architectures

Table 3 describes the architecture of the *InceptionTiny128* network. Table 4 describes the architecture of the *InceptionTiny299* network. For the architecture of *InceptionV3* we refer to the original publication [12]. All Inception-modules mentioned refer to the modules used in the original *InceptionV3* network, as described in [12]. Each convolutional layer consist of a 2D-convolution, batch normalization [3] and finally activation function. *InceptionTiny128* consists of 2 932 413 trainable parameters and *InceptionTiny299* consists of 2 941 693 trainable parameters.

Layer Type	**Filter Size**	**Stride**	**Activation**	**Output Shape**
Input	-	-	-	3x128x128
Convolutional	3	1	ReLU	32x126x126
Convolutional	3	1	ReLU	32x124x124
Max Pooling	2	2	-	32x62x62
Convolutional	3	1	ReLU	64x60x60
Convolutional	3	1	ReLU	128x58x58
Max Pooling	2	2	-	128x28x28
InceptionModuleA[1]	-	-	-	256x28x28
InceptionModuleB	-	-	-	736x14x14
InceptionModuleC[2]	-	-	-	768x14x14
Adaptive Average Pooling	-	-	-	768x1x1
Flatten	-	-	-	768
Fully Connected	-	-	ReLU	350
Fully Connected		-	Sigmoid	1

Table 3: List of layers in the *InceptionTiny128* network. [1]Using 32 pooling features [2]Using 128 7x7 channels

Layer Type	Filter Size	Stride	Activation	Output Shape
Input	-	-	-	3x299x299
Convolutional	3	2	ReLU	32x149x149
Convolutional	3	1	ReLU	32x147x147
Convolutional	3	1	ReLU	32x145x145
Max Pooling	3	2	-	32x72x72
Convolutional	3	1	ReLU	64x70x70
Convolutional	3	1	ReLU	128x68x68
Max Pooling	3	3	-	128x22x22
InceptionModuleA[1]	-	-	-	256x22x22
InceptionModuleB	-	-	-	736x11x11
InceptionModuleC[2]	-	-	-	768x11x11
Adaptive Average Pooling	-	-	-	768x1x1
Flatten	-	-	-	768
Fully Connected	-	-	ReLU	350
Fully Connected		-	Sigmoid	1

Table 4: List of layers in the *InceptionTiny299* network. [1]Using 32 pooling features [2]Using 128 7x7 channels

B Creatism Training Details

The networks were trained using the Adam-optimizer with a learning rate of 0.001 and batch size 64, except for the HDR-model that used batch size 16. The networks were regularized by using dropout in the layer between the convolutional and fully connected part. Dropout probability was set to 0.5 during training. Additionally, l_2-regularization with a weighting of 0.0005 was applied.

Operation	Network Model
Cropping	*InceptionTiny299*
Saturation	*InceptionTiny128*
HDR	*InceptionV3*
Curves	*InceptionTiny128*
Aesthetic Ranking	*InceptionTiny299*

Table 5: List of which network model was used for scoring of the different operations.

Table 5 describes which model was used for scoring each operation. The networks were trained for 100 epochs on an Nvidia GTX 1060 GPU. Each epoch included training on each image in the training set once. After each epoch the network was validated on the validation set. This was done by calculating the average absolute error between the network output and the true score for all

samples in the validation set. Which model to be used in the enhancement procedure was manually selected by studying the validation error for different epochs.

C Enhanced Images

Figures 15-26 show *Vacation* photos enhanced by Creatism, with the original photo to the left.

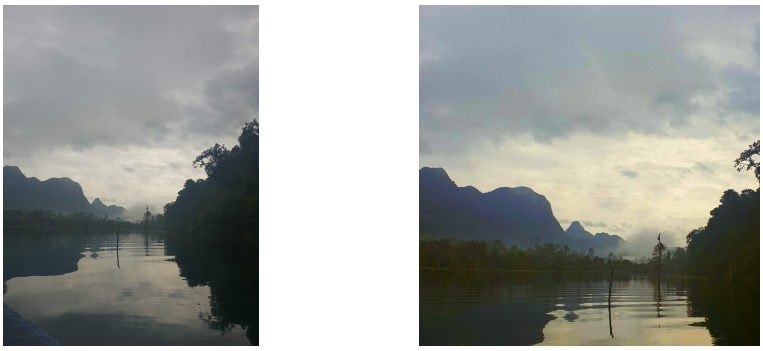


Figure 15: Photo from Khao Sok, Thailand.

Figure 16: Photo from Battambang, Cambodia.

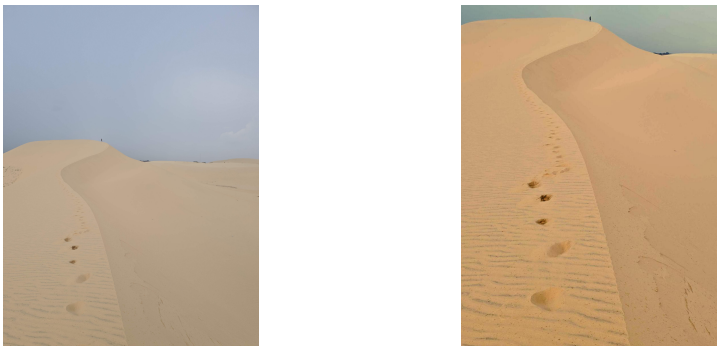


Figure 17: Photo from Mui Ne, Vietnam.

Figure 18: Photo from Mui Ne, Vietnam.

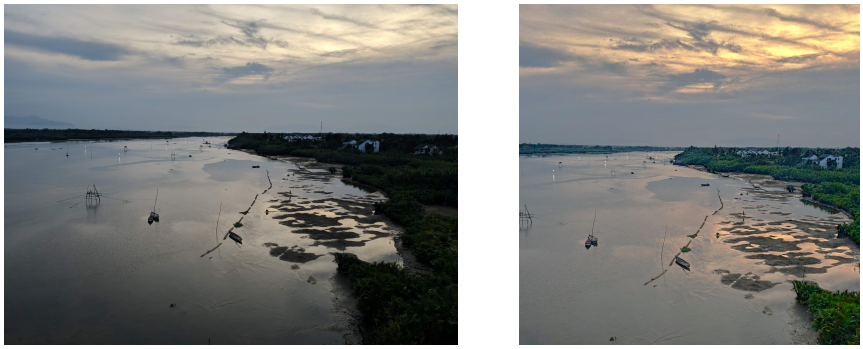


Figure 19: Photo from Hoi An, Vietnam.

Figure 20: Photo from Hoi An, Vietnam.

Figure 21: Photo from Ha Giang, Vietnam.

Figure 22: Photo from Ha Giang, Vietnam.

Figure 23: Photo from Ha Giang, Vietnam.

Figure 24: Photo from Gunung Ijen, Indonesia.

Figure 25: Photo from Railay, Thailand.

Figure 26: Photo from Khao Sok, Thailand.

D Individual Photo Results from Survey

Table 6 describes the results for each individual photo in the survey.

Photo	Creatism preferred (%)
1	14.8
2	37.8
3	59.0
4	9.8
5	73.8
6	62.3
7	32.8
8	11.5
9	16.4
10	72.1
11	80.3
12	50.8
13	29.5
14	18.0
15	6.6
16	78.7
17	26.2
18	85.2
19	42.6
20	16.4
21	4.9
22	40.1
23	47.5
24	88.5
25	16.4
26	65.6
27	11.4
28	45.9
29	37.7

Table 6: Percentage of times each Creatism photo was preferred.

E List of Contributions

- Karol Wojtulewicz
  - Worked on the denoising part of the project
  - Dived into the Noise2Noise paper
  - Dived into the PixelRL paper
  - Has written introduction in the report
  - Has written significant part of discussion for the denoising part of the report
  - Has created the architecture for the denoising autoencoder inspired by the U-Net architecture
- Daniel Jonsson
  - Worked on the denoising part of the project
  - Evaluated the (already trained) PixelRL network
  - Wrote the denoising parts of the paper
- Emil Luusua
  - Worked on the Creatism part of the project
  - Gathered training data from Flickr
  - Created the *Vacation* dataset
  - Implemented perturbation for cropping, the tone curves operation and some of the enhancement pipeline
  - Wrote parts of the Creatism part and introduction in the report
  - Presented the mid-term presentation
- Tobias Löfgren
  - Worked on the Creatism part of the project
  - Implemented perturbation for most image operations and some of the enhancement pipeline
  - Overlooked training of a few networks
  - Wrote parts of the Creatism part in the report
  - Presented final presentation
- Joel Oskarsson
  - Worked on the Creatism part of the project
  - Implemented most of the training loop, the network architectures, some of the enhancement pipeline and pre-processing for aesthetic ranking

  - Overlooked training of multiple networks
  - Wrote parts of the Creatism part in the report and final presentation
- Daniel Roos
  - Worked on the denoising part of the project
  - Evaluated the (already trained) Noise2Noise network
  - Evaluated and assisted Karol in implementation of our own autoencoder
  - Wrote major parts of chapter 3.1