High-pass Filters Preprocessing in Image Tracing with Convolutional Autoencoders

Andreas Fischer and Zineddine Bettouche
Deggendorf Institute of Technology
Dieter-Görlitz-Platz 1
94469 Deggendorf
E-Mail: andreas.fischer@th-deg.de, zineddine.bettouche@th-deg.de

***Abstract*—Image tracing describes the task of converting a raster image into a vector format. This paper investigates different processing pipelines that can extract an abstract representation of an image by means of high-pass filtering, autoencoding, and vectorization. Results indicate that reconstructing an image using Autoencoders, then filtering it with high-pass filters, and finally vectorizing it, can represent the image more abstractly while improving the efficiency of the vectorization process.**

***Index Terms*—image quality, vector graphics, neural networks, autoencoders, high-pass filters, vectorization, complexity theory, information technology.**

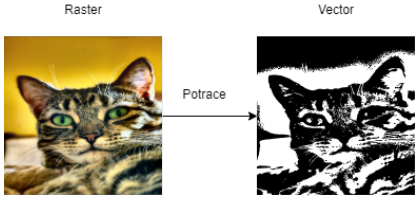

Figure 1. Potrace vectorization

I. Introduction

Graphical information can be represented digitally in two ways: raster-oriented or vector-oriented. The visual information in raster images is encoded as a 2D pixel array (or a bitmap). This is the case in the well-known Portable Network Graphics (PNG) or JPEG File Interchange Format (JFIF, commonly known just as JPEG). However, vector graphics represent data in a set of mathematical primitives such as lines and circles. A widely used format for vector graphics is Scalable Vector Graphics (SVG).

There are many use-cases for conversion between the two ways of representation. Vector images have to be rasterized in order to display on a raster-oriented monitor, and raster images have to be vectorized, if one wants to obtain a scale-free representation of the image. However, vectorization is not as accurate as rasterization. The main obstacle of converting a raster image into a vector graphic is the identification of mathematical primitives in a way that is appropriately fitting. On the one hand, a literal representation produces too much noise and fails to capture the relationships between image areas. On the other hand, the mapped image should not vary much from its original version.

This paper investigates different processing pipelines that can extract an abstract representation of an image by means of high-pass filtering, autoencoding, and vectorization. It extends previous work by Fischer and Amesberger [1], investigating the interplay between autoencoders and high-pass filters in the vectorization process. The cat dataset by Zhang et al. [2] is used to demonstrate the applicability of our approach.

The remainder of this paper is structured as follows: In Section II a brief introduction is given on image tracing, autoencoders and high-pass filters. Section IV introduces the methodology of this paper. This includes the evaluation methods used and the reason why they have been chosen. Section V presents the experiments and their results. This is the part that attempts to eliminate inefficient processing algorithms, so that only a few pipelines that score closely are put forwared to further evaluation. Section VI includes the evaluation of the different processing pipelines built, and closes with a summarizing interpretation. Section III discusses related work. Finally, Section VII concludes the paper and discusses future work.

II. Background

In this section, the three main techniques for image processing used in this paper are discussed. Image tracing is used to produce vector graphics from a raster image. Autoencoders and high-pass filters are used as pre-processing steps to reduce the complexity in the image with the goal of achieving an abstract representation of the image in its vector format.

A. Image Tracing

Image tracing is the process of vectorization raster images. It works by using edge detection mechanisms to identify areas in a raster image to be represented as mathematical objects such as polygons. The vectorization program used in this work is Potrace by Peter Selinger [3]. It traces the images by first converting them into black-and-white and then extracting Bézier-bounded polygons (cf. Figure 1).

B. Autoencoder

Autoencoders are artificial neural networks that can be trained to reconstruct the input by passing it through an

 ISBN: 978-1-61208-954-6

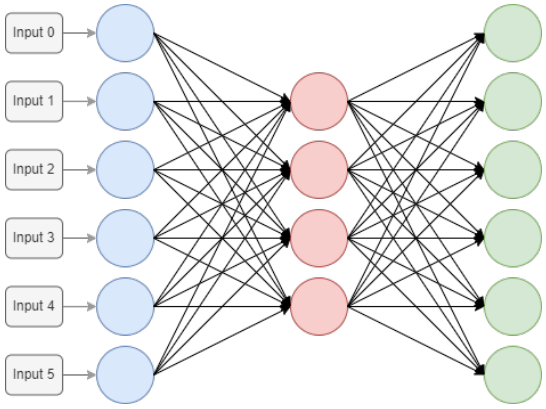

Figure 2. Example structure of an autoencoding network

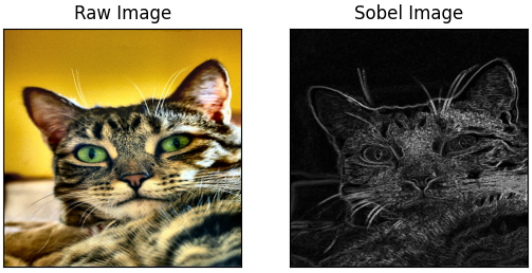

Figure 3. Applying Sobel derivatives on a random image

information bottleneck. The network learns how to preserve the input by an efficient reconstruction. This preservation of information can reduce the input complexity.

A basic autoencoder architecture is shown in Figure 2. The input layer is the layer that is the farthest on the left. The number of neurons becomes smaller the closer the layer is to the center, and this builds the information bottleneck. When the information reaches the center layer, it is at its highest density. By mirroring the architecture to the right, the encoded data from the center layer gets decoded again to its original size. The training process encourages the output layer to match the information that was passed to the input layer.

C. High-pass Filters

A high-pass filter can be used to make an image appear sharper. These filters (e.g., Sobel [4] and Canny [5]) emphasize fine details in the image. High-pass filtering works with the change in intensity. If one pixel is brighter than its immediate neighbors, it gets boosted. Figure 3 shows the result of applying a high-pass filter (Sobel) on a random image.

III. Related Work

In a paper done in MIT, Solomon and Bessmeltsev [6] explored the use of frame fields. The general idea of their method is to find a smooth frame field on the image plane, where at least one direction is aligned with nearby contours of the drawing. Around X- or T-shaped junctions, the two directions of the field will be aligned with the two intersecting contours. Then, the topology of the drawing is extracted by tracing the frame field and grouping traced curves into strokes. Finally, they created with the extracted topology a vectorization aligned with the frame field.

Lacroix [7] analyzed some problems of R2V conversion, and a strategy has been proposed involving a preprocessing stage generating a mask, from which edges are removed and lines are kept. A clustering is then performed while considering only the pixels of the mask. A new algorithm, the medianshift, has been proposed in this context. Then comes the labeling process which should also take the pixel type into account. The last step involves a regularization procedure. The importance of the pre-processing ignoring edge pixels while keeping lines has been shown on some examples. Tests also showed the superiority of the median-shift over the mean-shift, and over the clustering method used by Vector-Magic. This paper also showed that a better line vectorization can be obtained from enabling the extraction of dark lines, which can support the use of high-pass filters as a preprocessing stage to put further emphasis on those dark lines.

Xie et al. [8] designed a novel approach, which achieves a performance comparable to traditional linear sparse coding algorithm on the simple task of denoising additive white Gaussian noise. They use autoencoders to reduce image noise in the area of repairing damaged images.

Gong et al. [9] presented an algorithm that successfully completes the automatic extraction and vectorization of the road network. The main obstacles in road extraction in remote sensing images are: first, different scales and strong connectivity; second, complex backgrounds and occlusions; and third, high resolution and a small proportion of roads in the image. The process of road vectorization in this paper is mainly divided into road network extraction and vectorization preservation. This work also shows the advantages of using dense dilation convolution, which points to the possibility of using autoencoding models for vectorization preservation.

Fischer and Amesberger [1] showed that preprocessing the raster image with an autoencoder neural network, can reduce complexity by over 70% while keeping reasonable image quality. They proved that autoencoders perform significantly better compared to PCA in this task. We base our our work on this previous work, having a closer look at the effect of high-pass filters on autoencoding in an image vectorization pipeline.

IV. Methodology

This section describes the methodology of this paper. The programming implementation is first introduced along with the autoencoder structure. Then, the common grounds of processing pipelines are demonstrated. In addition, the evaluation methods are presented with the rationale behind their selection.

A. Autoencoder Structure & Software Implementation

The test/evaluation framework was implemented in Python. The autoencoder was implemented with TensorFlow [10] and Keras [11]. The convolutional neural network was built with convolution and pooling layers in three steps to a 32×32

 ISBN: 978-1-61208-954-6

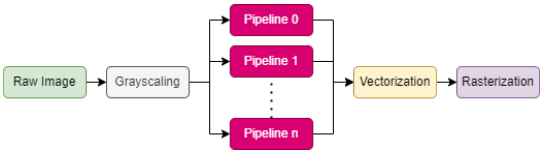

Figure 4. General processing approach

bottleneck. The decoder mirrors this structure with three steps of transposed convolutional layers and batch normalization layers. The autoencoder input is set to a 255x255 image (gray-scaled). The high-pass filters used in this paper are the standard implementations in OpenCV [12].

B. General Approach of Processing

Regardless of the path an image takes in any pipeline that will be built, the first processing stage is always going to be converting the image into gray-scale. The focus of this work is around single-channel images; however, it can be extended in the future for multi-channel (RGB) processing. Therefore, when a pipeline is demonstrated visually, the initial version of the image displayed is going to be gray-scale, but this is implying that the raw RGB images were all grayscaled, which will be a common branch for all the pipelines built in this work.

After an image is grayscaled, it will be put through a certain cascade of processing stages. In this paper, the concerned stages are: High-pass Filtering, Autoencoding, and Vectorization. The experiments of this work are going to tune the different parameters that these stages can take. More importantly, the outputs of all pipelines possible are going to be in a vector format; due to the fact that we are attempting to enhance the vectorization process, while aiming for an abstract representation of the image. Therefore, a rasterization stage is going to always be placed at the end of every pipeline. Converting images back into their raster format is mandatory to perform a comparison between the grayscaled image that was initially fed to a pipeline, and its resulting vector format. Hence, we rasterize the vector output to be able to evaluate the efficiency of the pipeline. A general processing approach for the different pipelines is shown in Figure 4.

C. Evaluation Methods

The case at hand deals with both vector and raster images. Therefore, for a comparison to take place, a comparison method for each format needs to be selected.

- **Vector:** Various methods can be used to measure the level of the complexity in a vector image. One can be the size of the file, which can be used to infer the length of the entire path entries in the file. Furthermore, investigating the reduction of complexity can be done through analyzing the longest path tags. The number of path tags can be taken as a characteristic value of the complexity. In this paper, it is assumed that the number of SVG path entries is directly related to the complexity.
- **Raster:** There are mainly two common ways of comparing raster images. The first one is comparing images based on the Mean Squared Error (MSE) [13]. The MSE value denotes the average difference of the pixels all over the image. A higher MSE value designates a greater difference between the original image and processed image. Nonetheless, it is indispensable to be extremely careful with the edges. A major problem with the MSE is that large differences between the pixel values do not necessarily mean large differences in content in the images. The Structural Similarity Index (SSIM) [14] is used to account for changes in the structure of the image rather than just the perceived change in pixel values across the entire image. The implementation of the SSIM used is contained in the Python library Scikit-image [15]. The SSIM method is significantly more complex and computationally intensive than the MSE method, but essentially the SSIM tries to model the perceived change in the structural information of the image, while the MSE actually estimates the perceived errors.

In the experiments conducted for this paper, the results of MSE and SSIM drive to the same conclusion. Therefore, in order to avoid redundancy, only the SSIM graphs are displayed in this paper.

V. Experiments

Experiments in this paper are essential to tune the different parameters a pipeline can have. In this section, some of the important experiments undergone are going to be presented.

Nevertheless, other unmentioned experiments also helped in cutting down the number of possible pipelines for evaluation. For instance, at the beginning of the work, three high-pass filters were selected for their commonality: Gaussian, Sobel, and Canny. However, the Gaussian filter introduced noise onto the images, which was not negligible. An experiment was done that attempted to reduce such noise by applying the Grain-extract or the Difference filters, but the results were not acceptable. Therefore, the Gaussian filter was removed from the set of filters. Another experiment was conducted to compare the scores of Sobel and Canny filtering. Both SSIM and MSE indicated that the two filters were so close to each other that there was not a decisive choice between them.

The two experiments mentioned below deal with the effect of features' color on the autoencoding and vectorization stages. The need for experimentation concerning such effect has risen when it was found that the conventional implementation of filters in the programming libraries resulted in images having a dark background with light features (lines and shapes), which was perceived as counter-intuitive.

The filters that were succeeded with the word **-direct** are the ones that follow the conventional implementation (dark background with white lines). On the opposite side, filters with **-inverse** as suffix refer to the ones that are constituted of white background with black lines.

 ISBN: 978-1-61208-954-6

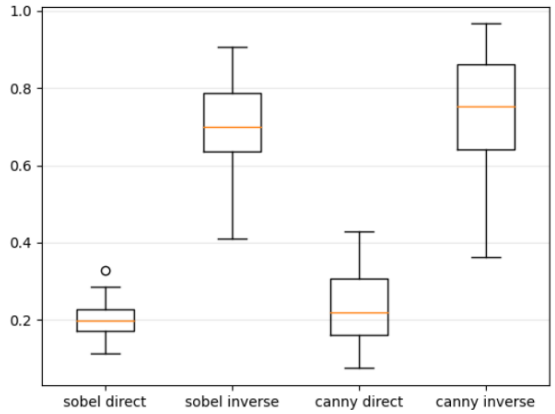

Figure 5. Similarity of autoencoded images in relation to the color of their features

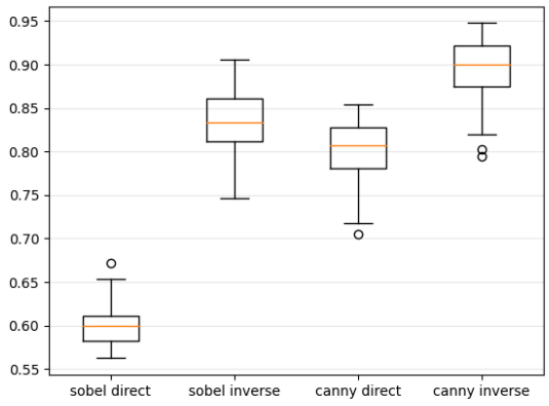

Figure 6. Similarity of vectorized images in relation to the color of their features

A. Features' Color Effect on Autoencoding

This experiment was done to obtain the difference between training an autoencoder with images whose lines are drawn in black on white background, and training it with the same images but inverted. Therefore, four models of autoencoders were trained with 5000 epochs. 50 images were reconstructed, in order to make the measurement more generalized.

From the plotted results in Figure 5 we conclude that autoencoders respond better when the training images have darker features.

B. Features' Color Effect on Vectorization

This experiment aims to display the effect of high-pass filters on reconstructed-images vectorization. We took 50 random images, reconstructed them and filtered each image with Canny and Sobel filters (direct and inverse: 4 versions per image). Finally, all of the images were vectorized, and then compared (after rasterization) with their versions pre-vectorization.

From the box-plots in Figure 6, we conclude that the filters brought more definition to the lines in the images, which made the shapes appear clearer, and this has lead to a better vectorization. Therefore, white images with black lines get vectorized better when compared to the darker images.

C. Results of Experimentation

Concerning autoencoding, for the sobel-direct, the mean and standard-deviation values were 0.202 and 0.044, respectively. Whereas their inverse scored 0.699 and 0.124, respectively. For the canny-direct, the mean and standard-deviation values were 0.234 and 0.090, respectively. Whereas the inverse scored 0.741 and 0.150, respectively. These values further states a better learning rate for the autoencoder when the most important features of an image are darker than its other contained data. Therefore, we can conclude that when training an autoencoder, the semi-supervised neural network responds better when the training images have darker lines in their important features.

Following the same pattern, the box-plots show a better fitness of white images with black lines when compared to the darker images in vectorization.

VI. Evaluation

Evaluation is concerned with how abstract the resulting images are. As there are two pre-processing blocks (filtering and autoencoding), four different pipelines can be built: autoencoding, filtering, autoencoding-filtering, and filtering-autoencoding. After one of these selections is fed the images, a vectorization process is always cascaded at the end.

First, all of the resulting images are going to be evaluated based on their path count (size) and similarity to the input images. Then, a summary of evaluation is going to be introduced for each of the pipelines individually.

Before engaging in the evaluation, it is good to elaborate on the column naming of the upcoming plots:

- default: the default image.
- sobel, canny: the filtered version of the image by the respective filter.
- dec: the decoded version.
- vect: the vectorized version.
- A combination of two or more indicates the case of cascaded stages. A default-dec-sobel label represents the following: the default image is reconstructed with the autoencoder then filtered with the sobel filter.

A. Evaluating the size of the produced images

To evaluate the size of the image, we count the number of path objects generated in the SVG file. From Figure 7 (note that the graph is in logarithmic scale) we see that the autoencoder (*-dec-*) significantly reduced the size of images, as it keeps only the most important features. The reconstructed filtered images (canny-dec, sobel-dec) had a similar path count. Although it was much smaller than the ones that did not go through that step, it was still above the default images that were reconstructed and vectorized without any filtering. Finally, when filters were applied onto the default images that were put through an autoencoding stage (default-dec-sobel,

 ISBN: 978-1-61208-954-6

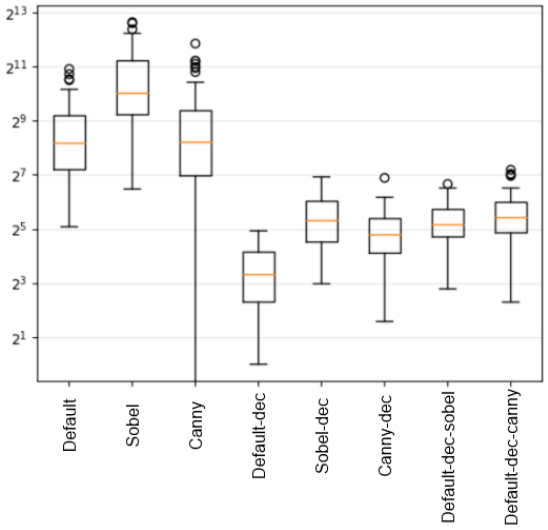

Figure 7. Path count of the resulted groups of vector images

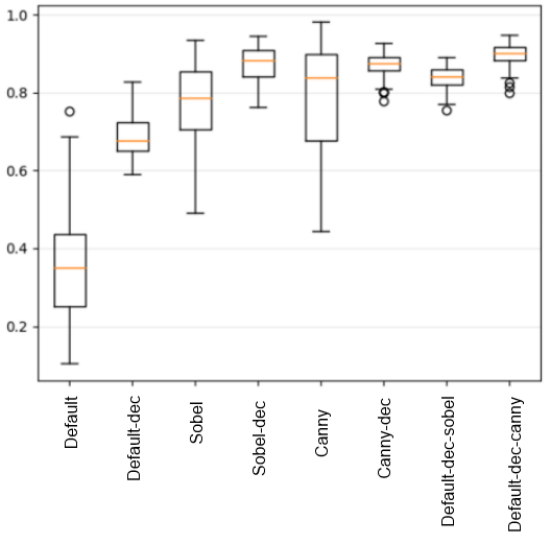

Figure 8. Vectorization accuracy of different pipelines

default-dec-canny), these images scored in size calculations very similarly to the filtered images when only reconstructed (canny-dec, sobel-dec).

B. Evaluating the quality of the produced images

A more accurate way of examining the efficiency of the vectorization process of each pipeline, is to compare the images and their vector versions (Figure 8). The pipeline of autoencoding-filtering-vectorization (two last groups on the most-right) seems to experience the highest SSIM, which indicates its fitness in vectorization. It made more sense for the autoencoder to reconstruct the images and then for the filters to come afterwards, putting emphasis on the important features of each image.

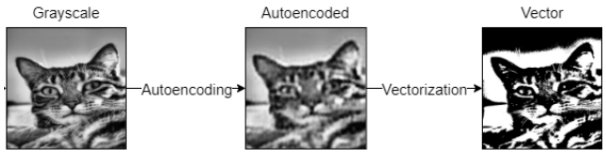

Figure 9. Autoencoding-vectorization pipeline

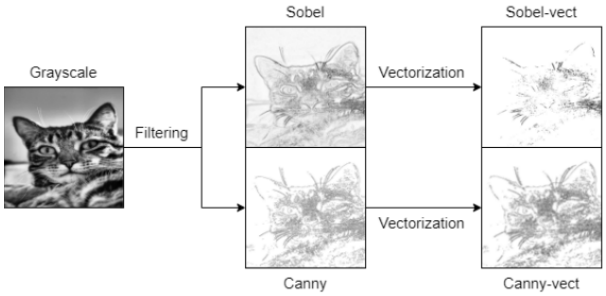

Figure 10. Filtering-vectorization pipeline

C. Implemented Pipelines: an evaluation summary

This is a summary of results evaluation for each of the pipelines individually.

- **Autoencoding-Vectorization:** This pipeline was based on the work of Fischer and Amesberger [1]. However, the implementation was different, and the evaluation was about the abstractness of the results. The quality of the vectorization is acceptable only in terms of general similarity. However, an abstract representation of the image is not achieved (Figure 9).
- **Filtering-Vectorization:** In this pipeline (Figure 10), the vectorization algorithm finds a difficulty in vectorizing the filtered images. This is due to the noises caused by the applied filters. Although the experiments showed that the quality of the vectorization increased when the images where taken as a light background with dark features, the noise involved created an obstacle for Potrace to convert thoroughly the images into a vector format, which resulted into losing data.
- **Filtering-Autoencoding-Vectorization:** This pipeline was built as an attempt to enhance the *Autoencoding-Vectorization* pipeline. Although the autoencoding stage was efficient in reducing the size of the images, it did not result in an abstract view of the image features. Therefore, a filtering stage was placed prior the autoencoding process. Unfortunately, this pipeline does not achieve the result intended. The autoencoding stage was supposed to reconstruct the filtered images in a lower complexity; but the case at hand is that, the autoencoding model is attempting to smooth the images, canceling the effect of the high-pass filters. This has resulted in a significant drop in the quality of the vector images, which is seen in Figure 11.
- **Autoencoding-Filtering-Vectorization:** Due to the results in the *Filtering-Autoencoding-Vectorization*

 ISBN: 978-1-61208-954-6

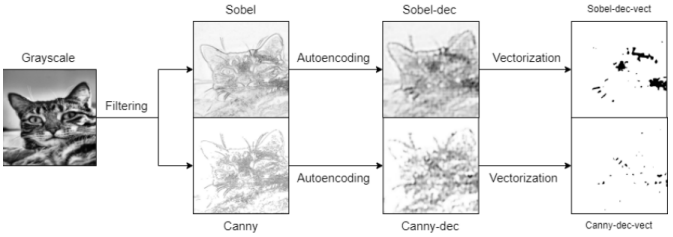

Figure 11. Filtering-autoencoding-vectorization pipeline

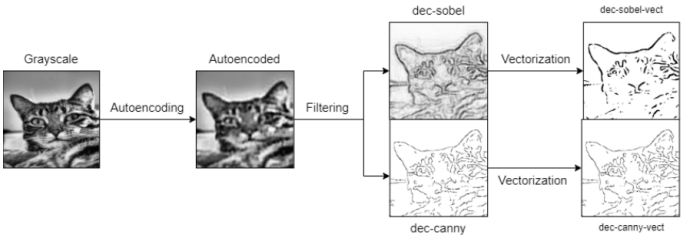

Figure 12. Autoencoding-filtering-vectorization pipeline

pipeline, it was clear that the filtering stage would act in a more proper way if it succeeded the autocoding process, rather than preceding it. This was concluded when the autoencoding model was seen to reduce the complexity of the images while introducing a smoothing effect. The filters were placed after the reconstruction stage to preserve the important features of the reduced-complexity image. This cascade shows an acceptable vectorization quality while resulting in the intended abstract representation of the images as shown in Figure 12.

As for providing more visualizations of the results that can be obtained with this pipeline, Figure 13 shows some random images that were fed to the Autoenconding-filtering-vectorization pipeline along with their respective output images. As can be seen, the features of the cats are extracted very clearly in all examples.

VII. Conclusion

This paper overall discussed the use of high-pass filters in vectorization pipelines, along with the autoencoding stage. It is concluded in this chapter that high-pass filters can enhance the training of an autoencoder, which in return make the vectorization process more efficient by preserving the important features of an image.

After evaluating the efficiency of the vectorization algorithm in every pipeline, it was clear that the images that went through the cascade of autoencoding-filtering, scored the highest in similarity, and the lowest in error. This points to the fact that the images that were reconstructed, preserved the most important features, which were brought up even more by the filtering part succeeding the reconstruction, which leads to not only a better vectorization but also a more abstract representation of the image.

This cascade of autoencoding-filtering gave decent results that matched the initial expectations; however, further work must be put into the structure of the models built, and their training dataset.

Concerning this future work, experiments showed that dark features on a light background in images can improve both the training of autoencoder models and the process of vectorization, which can be a good candidate for further investigation. From another side, the work in this paper deals with single-channel images (gray-scale); however, the vectorization of multi-channel images can be put through future experimentation.

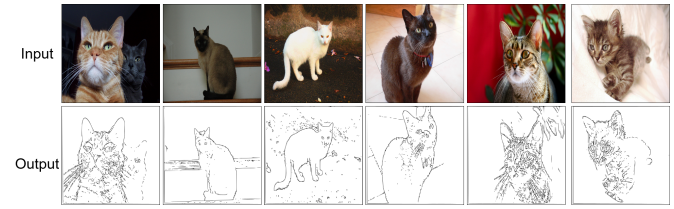

Figure 13. Some of the output images along with their input images of the pipeline Autoenconding-filtering-vectorization

References

[1] A. Fischer and M. Amesberger, "Improving image tracing with artificial intelligence," in *2021 11th International Conference on Advanced Computer Information Technologies (ACIT)*, 2021, pp. 714–717.

[2] W. Zhang, J. Sun, and X. Tang, "Cat head detection - how to effectively exploit shape and texture features," in *ECCV*, 2008.

[3] P. Selinger, "Potrace : a polygon-based tracing algorithm," 2003.

[4] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[5] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[6] M. Bessmeltsev and J. Solomon, "Vectorization of line drawings via polyvector fields," 2018.

[7] V. Lacroix, "Raster-to-vector conversion: Problems and tools towards a solution a map segmentation application," 03 2009, pp. 318 – 321.

[8] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[9] Z. Gong, L. Xu, Z. Tian, J. Bao, and D. Ming, "Road network extraction and vectorization of remote sensing images based on deep learning," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 303–307.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[11] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[12] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[13] C. Sammut and G. I. Webb, "Mean squared error," *Encyclopedia of Machine Learning*, no. 4, pp. 653–653, 2010.

[14] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[15] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

 ISBN: 978-1-61208-954-6