Visual Analytics 2021

Gustav Aarup Lauridsen
ID: au593405
Mail: 201804481@post.au.dk

All assignments were produced singlehandedly

Project 1

https://github.com/Guscode/cds-visual-exam/tree/main/Assignment_3

Assignment 3: Finding text using edge detection

The purpose of this assignment is to use computer vision to extract specific features from images. In particular, we're going to see if we can find text. We are not interested in finding whole words right now; we'll look at how to find whole words in a coming class. For now, we only want to find language-like objects, such as letters and punctuation.

Methods

The problem in this assignment is edge detection in relation to identifying text in an image. To address this problem computationally, a key step is to simplify the image, and highlight the structures of interest. This is done by utilizing CV2's image augmentation tools (Bradski, 2021). Firstly, images are converted from RGB-colors to grey-scale, whereafter thresholding is performed to binarize the image. Then, edges are detected with canny detection and overlayed the original image as contours. Similarly, the script includes an option for cropping the image, once again to reduce the amount of noise. For testing the performance, the script also outputs text generated from optical character recognition with Tesseract (Lee, 2021). Tesseract includes multiple page segmentation methods, which allows users to specify the tool for a specific image-type. This is also added as a user-specified argument and included in the script. Furthermore, there are limited use-cases from carrying out automatic edge-detection on single image inputs, why an option to handle multiple images was implemented. In order to test the tool, a dataset with nine images of Danish road signs showing city names was created.

ID: au593405
Mail: 20184481@post.au.dk

Results and Discussion

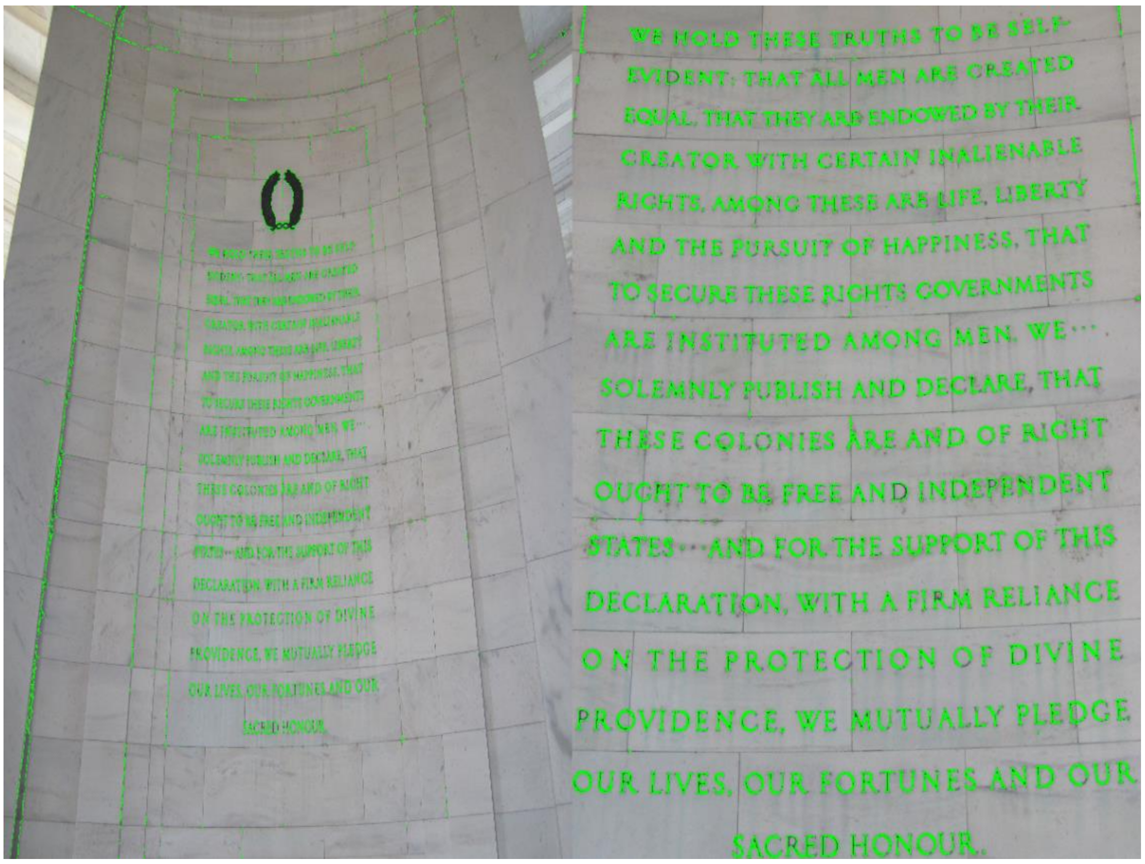

WE HOLD THESE TRUTHS TO BE SELF-
EVIDENT: THAT ALL MEN ARE CREATED
EQUAL, THAT THEY ARE ENDOWED BY THEIR
CREATOR WITH CERTAIN INALIENABLE
RIGHTS, AMONG THESE ARE LIFE, LIBERTY
AND THE PURSUIT OF HAPPINESS, THAT
TO SECURE THESE RIGHTS GOVERNMENTS
ARE INSTITUTED AMONG MEN. WE…
SOLEMNLY PUBLISH AND DECLARE, THAT
THESE COLONIES ARE AND OF RIGHT
OUGHT TO BE FREE AND INDEPENDENT
STATES… AND FOR THE SUPPORT OF THIS
DECLARATION, WITH A FIRM RELIANCE
ON THE PROTECTION OF DIVINE
PROVIDENCE, WE MUTUALLY PLEDGE
OUR LIVES, OUR FORTUNES AND OUR
SACRED HONOUR.

Figure 1: Jefferson image contoured without and with crop

The script was used on an image of the declaration of independence, firstly without cropping and then cropped to the text, as shown above. With the initial goal of finding text in images, it is clear from the results that removing noise around the text area is important. Removing noise around the image data makes thresholding more effective, which is seen in the second image (see figure 1), which shows how the cropped version's contours are almost solely text. On the road signs test (see figure 2), the text is contoured in seven out of nine images, but there is still a lot of contours that are not related to text. Therefore, using the crop-coordinates argument is advised. However, this approach is not robust when dealing with multiple images of different layouts, why implementing a text-detection layer like EAST (Zhou et al., 2017), could improve the performance. On the other hand, this would increase the complexity of the tool, and make it more demanding in terms of computation.

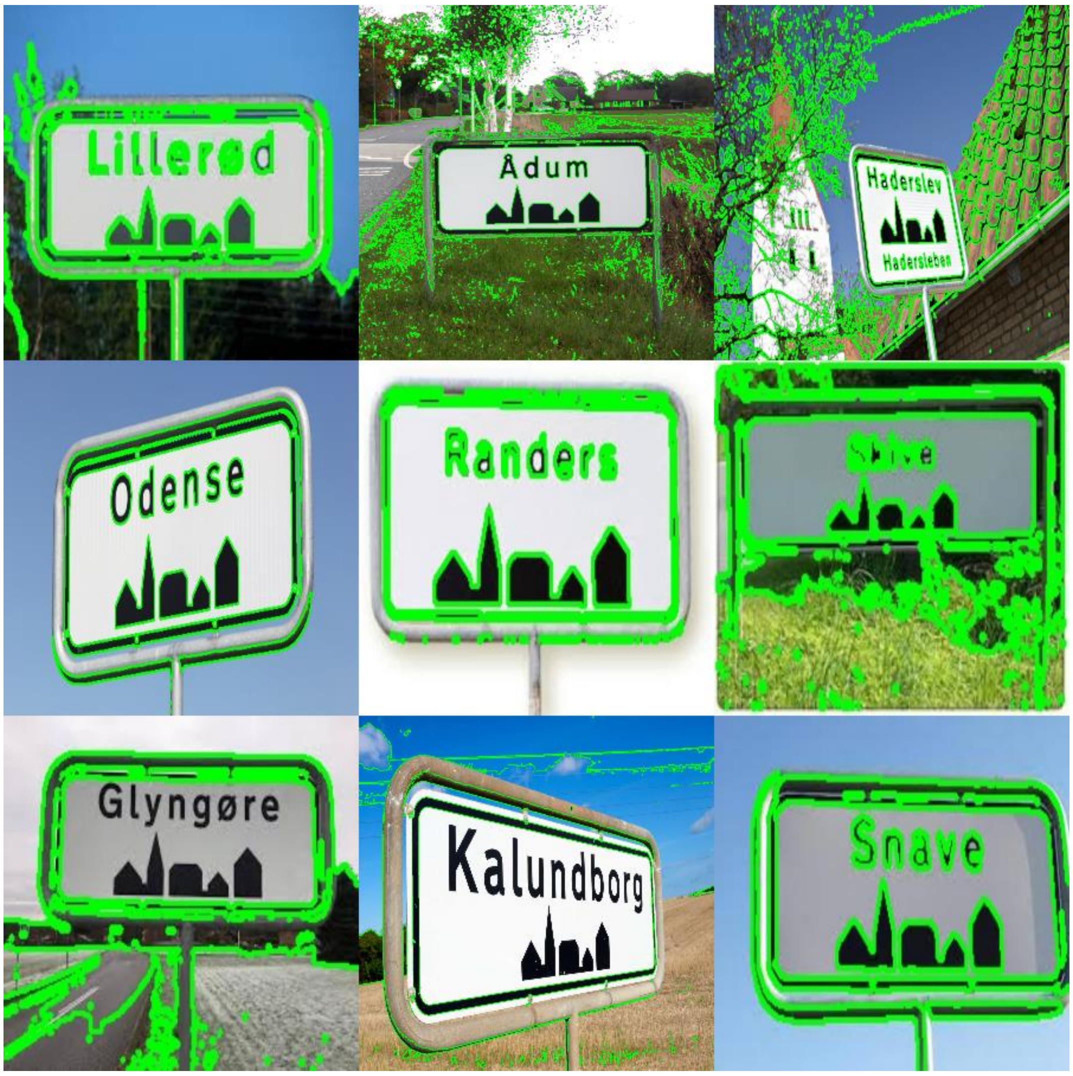


Figure 2: Road signs with contours

How to Use

In order to use the tool from a terminal, a virtual environment has to be set up.

```
$ cd directory/where/you/want/the/assignment
$ git clone https://github.com/Guscode/cds-visual-exam.git
$ cd cds-visual-exam
$ bash create_vis_venv.sh
$ source visvenv/bin/activate
```

The script can then be used with a single image or a folder with multiple images as input.
Single image:

```
$ cd assignment_3
$ python3 detect_edges.py --image jefferson.jpg --crop-coordinates X750X700Y750Y1150 --
psm 5 --output output
```

Folder with multiple images:

```
------------------------------------------------------------------------------------------------------------------------
$ python3 detect_edges.py --image-files signs --output output
------------------------------------------------------------------------------------------------------------------------
```

All user defined arguments:
--image: Path to an image
--output: Path where you want the output files
--crop-coordinates: X and Y coordinates for cropping image in X1X2Y1Y2 format.
--image-files: Path to a folder with images. script takes all files from folder with .jpg, .jpeg or .png
--psm: Specifies page segmentation method. see below for specifications.

See readme on github for further instructions.

Gustav Aarup Lauridsen
Visual Analytics Exam 2021

ID: au593405
Mail: 20184481@post.au.dk

Project 2

https://github.com/Guscode/cds-visual-exam/tree/main/Assignment_4

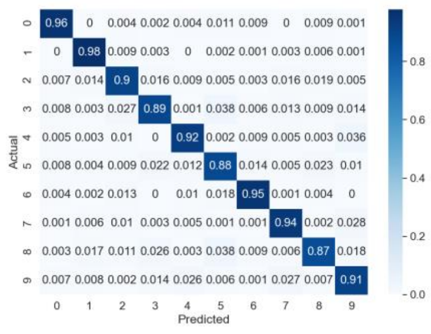
Assignment 4:
Classifier benchmarks using Logistic Regression and a Neural Network

You'll use your new knowledge and skills to create two command-line tools which can be used to perform a simple classification task on the MNIST data and print the output to the terminal. These scripts can then be used to provide easy-to-understand benchmark scores for evaluating these models.
You should create two Python scripts. One takes the full MNIST data set, trains a Logistic Regression Classifier, and prints the evaluation metrics to the terminal. The other should take the full MNIST dataset, train a neural network classifier, and print the evaluation metrics to the terminal.

Methods

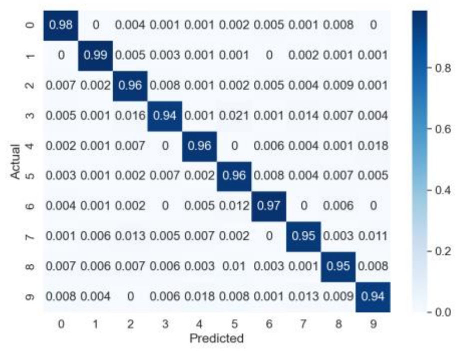
To solve this assignment, two scripts were created. Firstly, lr-mnist.py was created, which performs training and classification on a validation set in order to evaluate the performance and generalizability of a multinomial logistic regression model. The MNIST dataset contains black and white images of handwritten digits with a width and height of 28 pixels. Thereby, each image is represented as a normalized array of 784 values between 0 and 1. A miniature version of the mnist dataset is included in the repository in order to ease computation. Thus, the script allows the user to test any classification task with data that can be represented in an array of numbers. Furthermore, the script allows the user to set their own train/test split, with the default split being 80/20. Following data preprocessing a multinomial logistic regression is fitted to the training data. As the task is to explore the efficiency of the different methods, the script allows the user to change both the norm used in penalization and the solver function. The trained model is then used to predict the validation data, and the script outputs a heatmap confusion matrix and a classification report.
In nn-mnist.py, the only change is substituting the multinomial logistic regression with a neural network. Here, users can specify the number of neurons in the hidden layers, and the number of epochs. The output is the same as in lr-mnist.py, but the nn-mnist.py script allows the user to save the trained model for future use. Both models also include an option of specifying a test image of any kind, which is then predicted by the models.

Discussion



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	0.96	0	0.004	0.002	0.004	0.011	0.009	0	0.009	0.001
1	0	0.98	0.009	0.003	0	0.002	0.001	0.003	0.006	0.001
2	0.007	0.014	0.9	0.016	0.009	0.005	0.003	0.016	0.019	0.005
3	0.008	0.003	0.027	0.89	0.001	0.038	0.006	0.013	0.009	0.014
4	0.005	0.003	0.01	0	0.92	0.002	0.009	0.005	0.003	0.036
5	0.008	0.004	0.009	0.022	0.012	0.88	0.014	0.005	0.023	0.01
6	0.004	0.002	0.013	0	0.01	0.018	0.95	0.001	0.004	0
7	0.001	0.006	0.01	0.003	0.005	0.001	0.001	0.94	0.002	0.028
8	0.003	0.017	0.011	0.026	0.003	0.038	0.009	0.006	0.87	0.018
9	0.007	0.008	0.002	0.014	0.026	0.006	0.001	0.027	0.007	0.91

Figure 4: Multinomial Logistic Regression results



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	0.98	0	0.004	0.001	0.001	0.002	0.005	0.001	0.008	0
1	0	0.99	0.005	0.003	0.001	0.001	0	0.002	0.001	0.001
2	0.007	0.002	0.96	0.008	0.001	0.002	0.005	0.004	0.009	0.001
3	0.005	0.001	0.016	0.94	0.001	0.021	0.001	0.014	0.007	0.004
4	0.002	0.001	0.007	0	0.96	0	0.006	0.004	0.001	0.018
5	0.003	0.001	0.002	0.007	0.002	0.96	0.008	0.004	0.007	0.005
6	0.004	0.001	0.002	0	0.005	0.012	0.97	0	0.006	0
7	0.001	0.006	0.013	0.005	0.007	0.002	0	0.95	0.003	0.011
8	0.007	0.006	0.007	0.006	0.003	0.01	0.003	0.001	0.95	0.008
9	0.008	0.004	0	0.006	0.018	0.008	0.001	0.013	0.009	0.94

Figure 3: Neural Network results

While both models perform with an accuracy above 90%, there is a slight advantage towards using the neural network. The neural network obtains an F1-score of 0.957, while the multinomial logistic regression model obtains an F1-score of 0.922. Similarly, the lowest number-specific accuracy achieved by the neural network is 94% (see figure 3), while the multinomial logistic regression falls to 88% accuracy when classifying the number 5 (see figure 4). These classification results are very robust, which reflects both the complexity and the quality of the dataset. 'Garbage in garbage out' is a popular saying within the field of data-science, and this is a case of the opposite. The data is extremely uniform, and the complexity is limited, why it provides a good baseline measurement when testing methods. It does not, however, reflect most image classification tasks in terms of both complexity and data quality. With this in mind, the results show an advantage to using the neural network, but it is still worth exploring other methods when initiating an image classification project.

How to Use

In order to use the tool from a terminal, a virtual environment has to be set up.

```
$ cd directory/where/you/want/the/assignment
$ git clone https://github.com/Guscode/cds-visual-exam.git
$ cd cds-visual-exam
$ bash create_vis_venv.sh
$ source visvenv/bin/activate
```

To test the logistic regression, in the virtual environment, please run:

```
$ cd assignment_4
$ python lr-mnist.py --mnist download --output outputs
```

To test the neural network, in the virtual environment, please run:

```
$ python nn-mnist.py --mnist download --output outputs
```

The user defined arguments for lr-mnist.py are:

--mnist: Specify either path to a classification dataset or download to perform on mnist data.
--output: Path where you want the output files.
--solver: Specify solver algorithm - can be 'newton-cg', ‘sag’, ‘saga’ and ‘lbfgs’. default='saga'.
--penalty: Specify norm used in penalization, can be 'l2' or None. Default=None.
--test_image: Path to an image on which you wish to test the model.

The user defined arguments for nn-mnist.py are:

--mnist: Specify either path to a classification dataset or download to perform on mnist data.
--output: Path where you want the output files.
--layers: Specify hidden layers, default = 32 16.
--test_split: Spcifies train/test split, default = 0.2.
--epochs: Specify number of epochs, default = 100.
--save_model_path: Path to which you want to save the trained model.
--test_image: Path to an image on which you wish to test the model.

Using all the parameters:

```
$ python src/nn-mnist.py --mnist download --test-split 0.1 -- layers 16 8 --test_image test.png --output outputs --save_model_path outputs
```

See readme on github for further instructions.

Gustav Aarup Lauridsen
Visual Analytics Exam 2021

ID: au593405
Mail: 20184481@post.au.dk

Project 3

https://github.com/Guscode/cds-visual-exam/tree/main/Assignment_5

Assignment 5:
Multi-class classification of impressionist painters

So far in class, we've been working with 'toy' datasets - handwriting, cats, dogs, and so on. However, this course is on the application of computer vision and deep learning to cultural data. This week, your assignment is to use what you've learned so far to build a classifier which can predict artists from paintings. You can find the data for the assignment here:

https://www.kaggle.com/delayedkarma/impressionist-classifier-data

Using this data, you should build a deep learning model using convolutional neural networks which classify paintings by their respective artists. Why might we want to do this? Well, consider the scenario where we have found a new, never-before-seen painting which is claimed to be the artist Renoir. An accurate predictive model could be useful here for art historians and archivists!

Methods

The challenge of creating an image classifier with a convolutional neural network was solved by employing the LeNet (Lecun et al., 1998) architecture for making convolutional neural networks. Thus, the network is built with convolution layers, followed by a pooling layer which downsample the feature maps. Then a flatten layer is utilized, which turns the input into an array with one dimension, which is then passed into a set of dense layers. Lastly a softmax function is employed, which outputs a vector with probabilities of the input belonging to each class. For this project, the script enables the user to define the number of neurons in the hidden layer of the first convolution layer. Similarly, the user can specify the number of epochs and the learning rate. In order to use the convolutional neural network, all images are resized to the same size and represented as a normalized array. The size of the input images is also customizable trough terminal arguments but increasing image size vastly increases computing demands.

Discussion

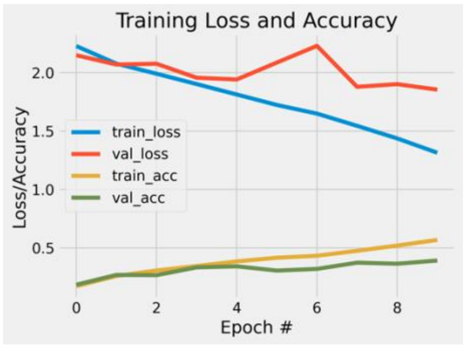


Figure 5: Training history plot

In the classification report available on github, the model shows a macro F1-score of 0.386 and an accuracy of 0.391. These are the results from a 10-epoch run with default settings. The low number of epochs is chosen due to lack of computing power. From figure 5 we see that, at epoch 10, the validation accuracy is still rising, and slightly following the training accuracy, indicating only weak signs of overfitting. Therefore, it is fair to assume that increasing the number of epochs would yield a better validation accuracy. Similarly, including a pre-trained model for image-classification could be beneficial, as it allows fine-tuning on the specific image type, in this case impressionistic art, rather than learning the features from scratch.

How to Use

In order to use the tool from a terminal, a virtual environment has to be set up.

```
$ cd directory/where/you/want/the/assignment
$ git clone https://github.com/Guscode/cds-visual-exam.git
$ cd cds-visual-exam
$ bash create_vis_venv.sh
$ source visvenv/bin/activate
```

The script cnn-artists.py will train a convolutional neural network a dataset with impressionist painters found here:

https://www.kaggle.com/delayedkarma/impressionist-classifier-data

In order to test the script, please download the dataset and place it in the Assignment_5 folder Then run, in the virtual environment:

```
$ python cnn-artists.py --training-path impressionist/training --validation-path
impressionist/validation --output output
```

This will return a classification report in .csv format and a history plot showing the training accuracy, validation accuracy, training loss and validation loss.

The user defined arguments for cnn-artists.py are:

--training-path: Specify path to training data

--validation-path: Specify path to validation data
--epochs: Specify amount of epochs, default = 10
--image-size: Specify image size, default H200W200. Larger sizes will slow down the computation.
--learning-rate: Specify learning rate, default = 0.1
--hidden-layers: Specify neurons in hidden layer, default = 32.
--output: Specify output path

Using all the parameters:

```
-----------------------------------------------------------------------------------------------------------------------
$ python cnn-artists.py --training-path impressionist/training --validation-path
impressionist/validation --output output --epochs 5 --image-size H300W300 --learning-rate 0.001 --
hidden-layers 16
-----------------------------------------------------------------------------------------------------------------------
```

See readme on github for further instructions.

Gustav Aarup Lauridsen
Visual Analytics Exam 2021

ID: au593405
Mail: 20184481@post.au.dk

Project 4

https://github.com/Guscode/GLStyleNet_TF2_Video

Self-assigned project: State-of-the-Art Style Transfer on Video

With inspiration in Erik A. Frandsen's drawings of bike races (see figure 6), I set out to create my favorite bike racing moment on video with Erik A. Frandsen's style, utilizing the strengths of technology to create a cultural output, that would not be possible without it. The assignment consists of incorporating video stylization in a state-of-the-art style-transfer method.

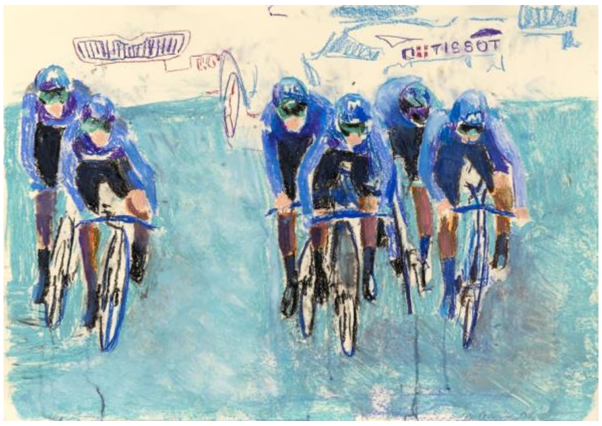


Figure 6: art piece by Erik A. Frandsen

Methods

Firstly, the style-transfer method I decided to work with is GLStyleNet (Wang et al., 2018). GLStyleNet excels in comparison to other style-transfer methods because it allows feature specific style-transfer, by using semantic masks of both content and style images. Thus, GLStyleNet takes global structures and local style into consideration (Wang et al., 2018) when creating the convolutional neural network used for style-transfer. In practice, this means distinguishing between the style of the background and the style of the foreground, which resembles the way most painters work. However, GLStyleNet was built for Tensorflow 1.4, why it was updated to be executable with Tensorflow 2. Similarly, the original tool demanded manual semantic masking, why stylizing multiple pictures was ineffective. Therefore, supporting semantic masking functions were built into the tool. The masking functions aim to create a foreground, a background and an unknown region using thresholding, morphological transformations and image segmentation. With semantic masking built into the tool, functions for handling video input and creating video output was implemented. Finally, the main function was extended to include options for video handling and looping through frames one at a time. Thus, the updated tool can take in a video file, automatically stylize each frame with semantic masking and output a stylized video.

Discussion

For video result see link to video.

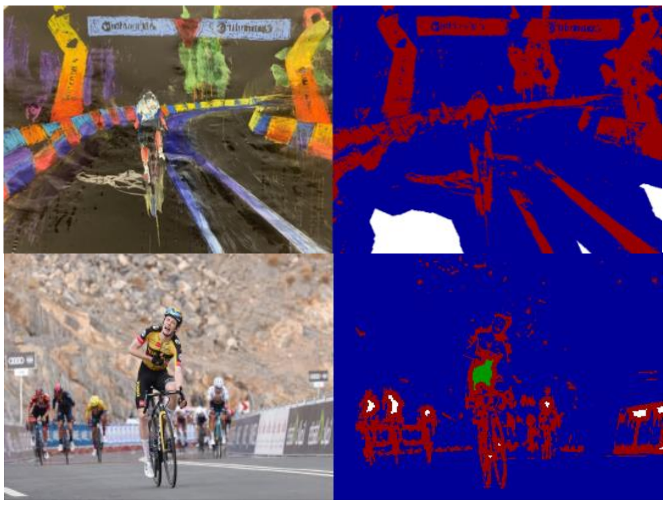


Figure 8: Content and style images with semantic masking

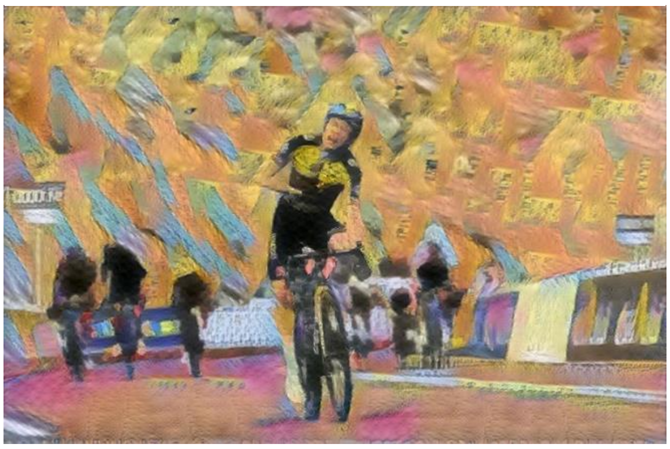


Figure 7: Result of stylization using content and style images and semantic masks from figure 7

The video and figure 7 and 8 show outputs that closely resemble the style of Erik A. Frandsen's art. While evaluating a qualitative output like stylized images is complex, I am personally satisfied and very excited about the results. Similarly, multiple fans of Erik A. Frandsen's work didn't recognize that the picture was created artificially.

With regards to the computational demands of the tool, creating videos requires access to a GPU. The example video was created using a GPU on Google Colab Pro, where stylizing 339 frames had a duration of 13 hours. Similarly, while the output is best on high resolution images, the duration of stylizing and storage needs are highly dependent on image size. Therefore, the tool is of limited accessibility in the sense that a lot of computing power is needed. This is, however, also a problem in the original tool. In terms of output quality, the results lack when the style image and content image do not resemble each other. This can be partially solved by better semantic masking, e.g., implementing pre-trained masking tools into GLStyleNet. Another approach would be to include multiple style images with the given style

ID: au593405
Mail: 20184481@post.au.dk

and perform image similarity search on the semantic masks in order to choose the style image most accurately resembling the content image.
With regards to the use-case of Erik A. Frandsen's art, this is a very direct application of a neural network taking the place of a human brain. As Erik A. Frandsen chooses pictures from bike races and draws them, he is largely performing the same task as GLStyleNet, and the other way around. What GLStyleNet did not do is come up with the style. It is, however, possible for artists to create one style image and generate a lifetime of art from it.

How to Use

In order to use the tool from a terminal, a virtual environment has to be set up.

```
------------------------------------------------------------------------------------------------------------------------
$ cd directory/where/you/want/GLStyleNet_TF2_Video
$ git clone https://github.com/Guscode/GLStyleNet_TF2_Video.git
$ cd GLStyleNet_TF2_Video
$ bash create_stylevenv.sh
$ source stylevenv/bin/activate
------------------------------------------------------------------------------------------------------------------------
```

In order to test the script, style-transfer can be run on the test images supplied on github using this line:

```
------------------------------------------------------------------------------------------------------------------------
$ python GLStyleNet.py --content images/mvdp_win.jpeg  --style images/style.png
------------------------------------------------------------------------------------------------------------------------
```

Running style transfer on videos:

```
------------------------------------------------------------------------------------------------------------------------
$ python GLStyleNet.py --content your_video.mp4  --style your_style_image.png --input-
type video --fps 12
------------------------------------------------------------------------------------------------------------------------
```

The user defined arguments for GLStyleNet.py are:

--content: Specify path to content image or video
--content-mask: Path to content-mask. if None, content mask is created using masking_functions.py
--content-weight: Weight of content
--style: Path to style image
--style-mask: Path to style mask, if None, content mask is created using masking_functions.py
--local-weight: Weight of local style loss.
--semantic-weight: Weight of semantic map channel.
--global-weight: Weight of global style loss.
--output: Path to output
--iterations: number of iterations, default = 100.

--smoothness: Weight of image smoothing scheme.
--input-type: Specify input type, default = image, options ["image", "video"]
--fps: specify desired frames per second in stylized video, default = 12
--init: Image path to initialize, "noise" or "content" or "style".
--device: Specify devices: "gpu"(default: all gpu) or "gpui"(e.g. gpu0) or "cpu"
--class-num: number of semantic classes
--start-at: start at specific frame in video, default = 0.
Using all the parameters:

```
$ python cnn-artists.py --training-path impressionist/training --validation-path
impressionist/validation --output output --epochs 5 --image-size H300W300 --learning-rate 0.001 --
hidden-layers 16
```

See readme on github for further instructions.

References

Bradski, G. (2021). *Opencv/opencv* [C++]. OpenCV. https://github.com/opencv/opencv (Original work published 2012)

Impressionist_Classifier_Data. (n.d.). Retrieved May 18, 2021, from https://kaggle.com/delayedkarma/impressionist-classifier-data

LECUN, Y. (2010). THE MNIST DATABASE of handwritten digits. *Http://Yann.Lecun.Com/Exdb/Mnist/*. https://ci.nii.ac.jp/naid/10027939599/

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lee, M. (2021). *pytesseract: Python-tesseract is a python wrapper for Google's Tesseract-OCR* (0.3.7) [Python]. https://github.com/madmaze/pytesseract

Wang, Z., Zhao, L., Xing, W., & Lu, D. (2018). GLStyleNet: Higher Quality Style Transfer Combining Global and Local Pyramid Features. *ArXiv:1811.07260 [Cs]*. http://arxiv.org/abs/1811.07260

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., & Liang, J. (2017). EAST: An Efficient and Accurate Scene Text Detector. *ArXiv:1704.03155 [Cs]*. http://arxiv.org/abs/1704.03155