Visual Analytics Portfolio

Malte Højmark-Bertelsen
201809226@post.au.dk

Cultural Data Science
Aarhus University
2021

Table of Contents

Table of Contents **1**

Assignment 3 - Edge Detection **2**
- Project Description 2
- Methods 2
- Usage 3
- Discussion of Results 3

Assignment 4 - Classification Benchmarks **6**
- Project Description 6
- Methods 6
- Usage 6
- Discussion of Results 7

Assignment 5 - CNNs on cultural image data **10**
- Project Description 10
- Methods 10
- Usage 11
- Discussion of Results 12

Self Assigned Portfolio - Neural Style Transfer **14**
- Project Description 14
- Methods 14
- Usage 15
- Discussion of Results 16

References **18**

Assignment 3 - Edge Detection

GitHub link for general repository:
https://github.com/MalteHB/visual_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/edge_detection.py

Project Description

Description for 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.

Download and save the image at the link below:

https://upload.wikimedia.org/wikipedia/commons/f/f4/%22We_Hold_These_Truths%22_at_Jefferson_Memorial_IMG_4729.JPG

Using the skills you have learned up to now, do the following tasks:

- *Draw a green rectangular box to show a region of interest (ROI) around the main body of text in the middle of the image. Save this as image_with_ROI.jpg.*
- *Crop the original image to create a new image containing only the ROI in the rectangle. Save this as image_cropped.jpg.*
- *Using this cropped image, use Canny edge detection to 'find' every letter in the image*
- *Draw a green contour around each letter in the cropped image. Save this as image_letters.jpg"* - description for assignment 3.

Methods

To detect the letters from the picture of one of the pieces of texts from the Jefferson Memorial I used the image processing and computer vision software OpenCV (Bradski, 2000). Here I used OpenCV to first crop the image to only contain an image with the letters. Afterwards, I created a gray-scaled version of the image, applied a gaussian blur to mitigate high frequency edges, used Canny Edge Detection, an edge detection algorithm developed by John F. Canny (Canny, 1986), and drew the green contours detected by the Canny algorithm.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/visual_analytics_cds.git
cd visual_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

The package *opencv* is also a requirement for the script to work, however, installing it through pip might cause issues and have different dependencies depending on the operating system used. I, therefore, recommend installing it by using the 'conda' command:

```
# Conda install packages
conda install opencv -y
```

Finally to run the scripts, run the following lines:

```
# Run the neural style transfer script.
# See the available flags by using the '--help' after the command.
python src/edge_detection.py
```

Discussion of Results

The edge detection resulted in three images, including the image with a green region of interest (Image 1), the cropped image (Image 2) and the image with the green contours drawn around the detected letters (Image 3). I do not think that the edge detection was completely successful. It managed to capture the majority of the letters, however, it seems that it has difficulties capturing the inner regions of letters with encapsulated areas such as 'O' and 'D'. The Canny Edge Detection has since the creation in 1986 been improved many times, and if one wanted to try to fully detect all the letters, one could try to employ gravitational field intensity instead of the image gradient-based approach as shown in later research (Rong et al., 2014).

Image 1: Image from Jefferson Memorial with a green region of interest.

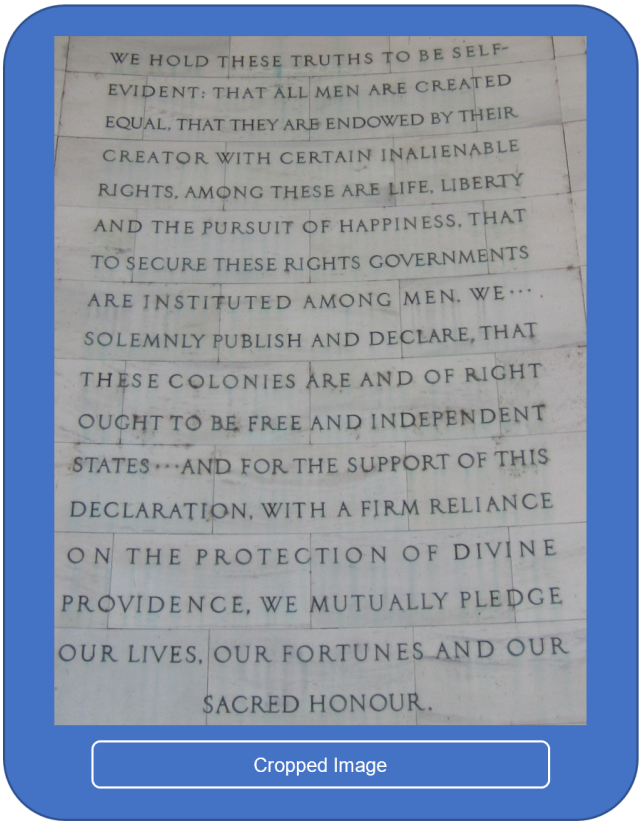

Image 2: Cropped image.

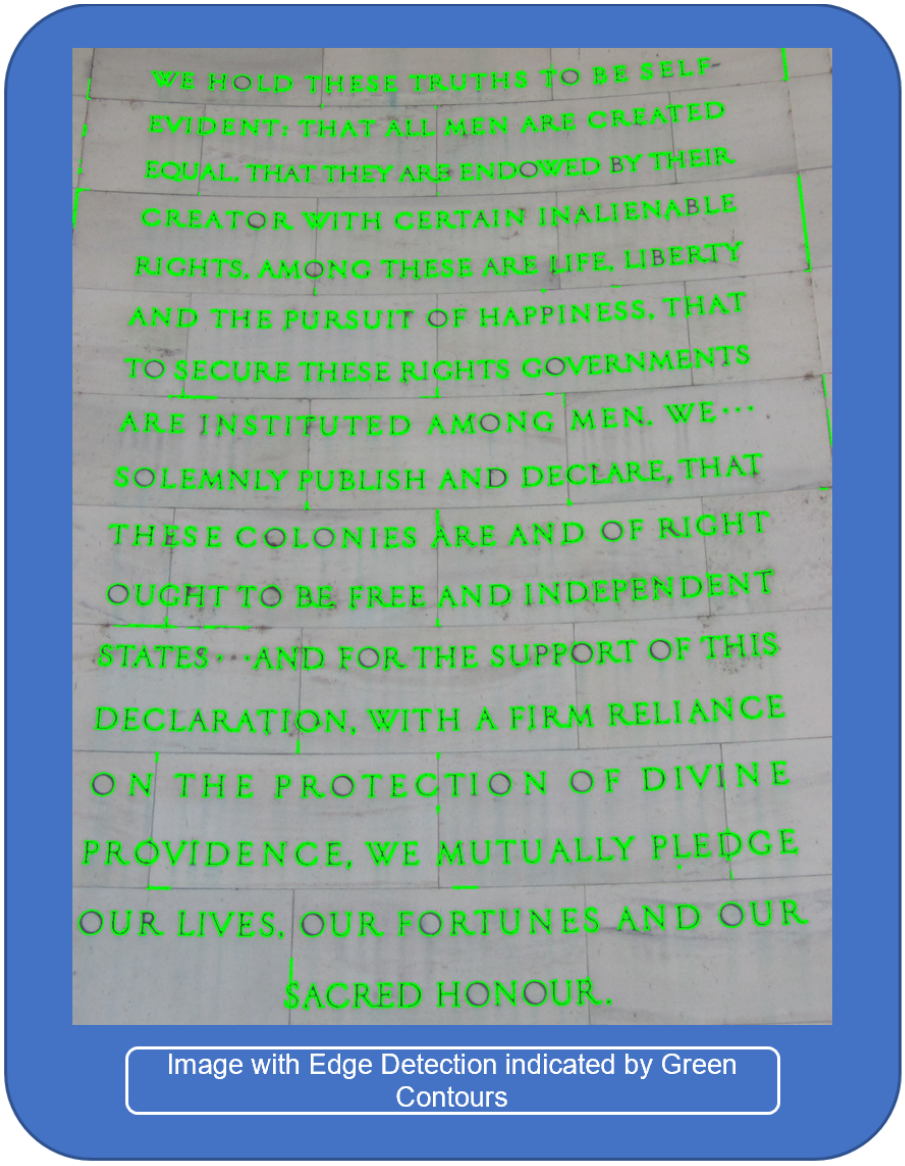

Image 3: Image with the letters detected by using the Canny Edge Detection algorithm.

Assignment 4 - Classification Benchmarks

GitHub link for general repository:
https://github.com/MalteHB/visual_analytics_cds

GitHub link for the specific python scripts:
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/lr-mnist.py
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/nn-mnist.py

Project Description

Description for assignment 4:

"This assignment builds on the work we did in class and from session 6.

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." - description for assignment 4.

Methods

To train a logistic regression classifier to predict the MNIST dataset (LeCun & Cortes, 2010) I employed the Python package Scikit-Learn (Pedregosa et al., 2011). For the neural network we were provided with a standard neural network implementation using NumPy (Harris et al., 2020). While the implementation was sublime, I further implemented the option to use early stopping in it. This was due to a long training time, and to mitigate the problem of substantial overfitting. In the assessment of the models a classification report was created, including the precision, recall and F1-scores on the validation set for each digit, as well as an overall macro-averaged F1 score, and the overall accuracy.

Usage

To run the scripts developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/visual_analytics_cds.git
cd visual_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

The package *opencv* is also a requirement for the script to work, however, installing it through pip might cause issues and have different dependencies depending on the operating system used. I, therefore, recommend installing it by using the 'conda' command:

```
# Conda install packages
conda install opencv -y
```

Finally to run the scripts, run the following lines:

```
# Run the neural style transfer script.
# See the available flags by using the '--help' after the command.
python src/lr-mnist.py
Python src/nn-mnist.py
```

Discussion of Results

The two scrips concluded in a slew of results seen in Table 1 and Table 2. The results showed that both the logistic regression classifier and the neural network are very good at categorizing the digits from the MNIST dataset. It is, however, evident that the neural network achieves the best performance with an F1-score = 0.96, while the logistic regression classifier achieves F1-score = 0.91. While the default amount of epochs is set to 100, implementation of early stopping stopped the training after 40 epochs, and it does, thus, not seem that further training could lead to performance enhancement. To enhance performance, one could implement larger convolutional neural networks, and other deep learning methods, could possibly also lead to a higher performance.

Classification Report for the Logistic Regression	**Precision**	**Recall**	**F1-score**	**Number of test pictures**
0	0.95	0.96	0.96	213

1	0.93	0.97	0.95	301
2	0.91	0.89	0.90	265
3	0.91	0.89	0.90	261
4	0.89	0.89	0.89	217
5	0.91	0.88	0.89	231
6	0.93	0.96	0.94	233
7	0.92	0.92	0.92	271
8	0.87	0.87	0.87	225
9	0.88	0.87	0.88	285
Accuracy			0.91	2500
Macro Average	0.91	0.91	0.91	2500
Weighted Average	0.91	0.91	0.91	2500

Table 1: Classification report for the Logistic Regression Classifier on the MNIST dataset.

Classification Report for the Neural Network	**Precision**	**Recall**	**F1-score**	**Number of test pictures**
0	0.98	0.97	0.98	213
1	0.97	0.98	0.98	301
2	0.95	0.94	0.94	265
3	0.94	0.95	0.95	261
4	0.92	0.96	0.94	217
5	0.96	0.95	0.95	231
6	0.95	0.98	0.97	233
7	0.95	0.97	0.96	271
8	0.96	0.92	0.94	225
9	0.97	0.92	0.94	283
Accuracy			0.96	2500
Macro Average	0.96	0.96	0.96	2500

Weighted Average	0.96	0.96	0.96	2500

Table 2: Classification report for the Neural Network on the MNIST dataset.

Assignment 5 - CNNs on cultural image data

GitHub link for general repository:
https://github.com/MalteHB/visual_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/cnn-artists.py

Project Description

Description for 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!

For this assignment, you can use the CNN code we looked at in class, such as the ShallowNet architecture or LeNet. You are also welcome to build your own model, if you dare - I recommend against doing this.

Perhaps the most challenging aspect of this assignment will be to get all of the images into format that can be fed into the CNN model. All of the images are of different shapes and sizes, so the first task will be to resize the images to have them be a uniform (smaller) shape.

You'll also need to think about how to get the images into an array for the model and how to extract 'labels' from filenames for use in the classification report." - description for assignment 5.

Methods

To solve the task, being a multilabel classification task, i chose to use a convolutional neural network (CNN) with three convolutional layers with 3 x 3 kernels, a stride of 1 and a padding of

1 to make sure that the the outputs for each layer will have the same dimensions as the input. After each convolution layer both a ReLU (Nair & Hinton, 2010) activation and a max pooling operation is performed. Finally, the output from the last convolutional layer is flattened and passed through a fully-connected layer with a ReLU activation function and then onto the final classification layer with the 10 units corresponding to the number of classes in the impressionist image data set. Both the CNN and all the data preprocessing was done using Tensorflow (Martín Abadi et al., 2015). The batch size was set to 32, the optimizer used was Adam (Da, 2014) with an initial learning rate of 0.001, and a total of 10 epochs was run. To evaluate the model, the precision, recall and F1-scores were calculated on the validation set for each artist, as well as an overall macro-averaged F1 score, and the overall accuracy. Furthermore, the loss for both the training and the validation set was calculated during training to assess overfitting.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/visual_analytics_cds.git
cd visual_analytics_cds
```

Moreover, I recommend installing Anaconda (“Anaconda Software Distribution,” 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the ‘conda’ command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the ‘requirements.txt’ file:

```
# Install requirements
pip install -r requirements.txt
```

The package *opencv* is also a requirement for the script to work, however, installing it through pip might cause issues and have different dependencies depending on the operating system used. I, therefore, recommend installing it by using the ‘conda’ command:

```
# Conda install packages
conda install opencv -y
```

Before running the script you need to download the data and extract it to the data folder in your cloned ‘visual_analytics_cds’ folder. The data can be downloaded from the following link: https://www.kaggle.com/delayedkarma/impressionist-classifier-data

Finally to run the script run the following line:

```
# Run the neural style transfer script.
# See the available flags by using the '--help' after the command.
python src/cnn-artists.py
```

Discussion of Results

The precision, recall and F1-scores and the overall accuracy can be seen in Table 3 while the loss and accuracy curves for both the training and the validation set, during training, can be seen in Image 4. It is evident that the model has not learned to thoroughly distinguish between the individual artistic features from the different impressionist artists, in that the macro-averaged F1-score is 0.41 and the accuracy is 0.41. The model is best at predicting Matisse paintings correctly, F1 = 0.48, while its worst artist is Gauguin, F1 = 0.33. This could possibly be due to a substantial amount of overfitting, which the loss and training curves also indicate (see Image 4). To increase the performance of the model, one could try to increase the size of the neural network, or perhaps try to include some dropout layers. Other methods, such as increasing the amount of training data, either by collecting more paintings, or through more innovative methods such as data augmentation, could possibly also lead to better validation data metrics.

Classification Report for the Convolutional Neural Network	**Precision**	**Recall**	**F1-score**	**Number of test pictures**
Cezanne	0.30	0.39	0.34	99
Degas	0.48	0.30	0.37	99
Gauguin	0.34	0.32	0.33	99
Hassam	0.32	0.47	0.39	99
Matisse	0.51	0.45	0.48	99
Monet	0.43	0.42	0.43	99
Pissarro	0.49	0.44	0.47	99
Renoir	0.46	0.39	0.43	99
Sargent	0.45	0.43	0.44	99
Van Gogh	0.38	0.40	0.39	99
Accuracy			0.41	990
Macro Average	0.42	0.41	0.41	990

Weighted Average	0.42	0.41	0.41	990

Table 3: Table X: Showing the metrics from the CNN after 10 epochs of training

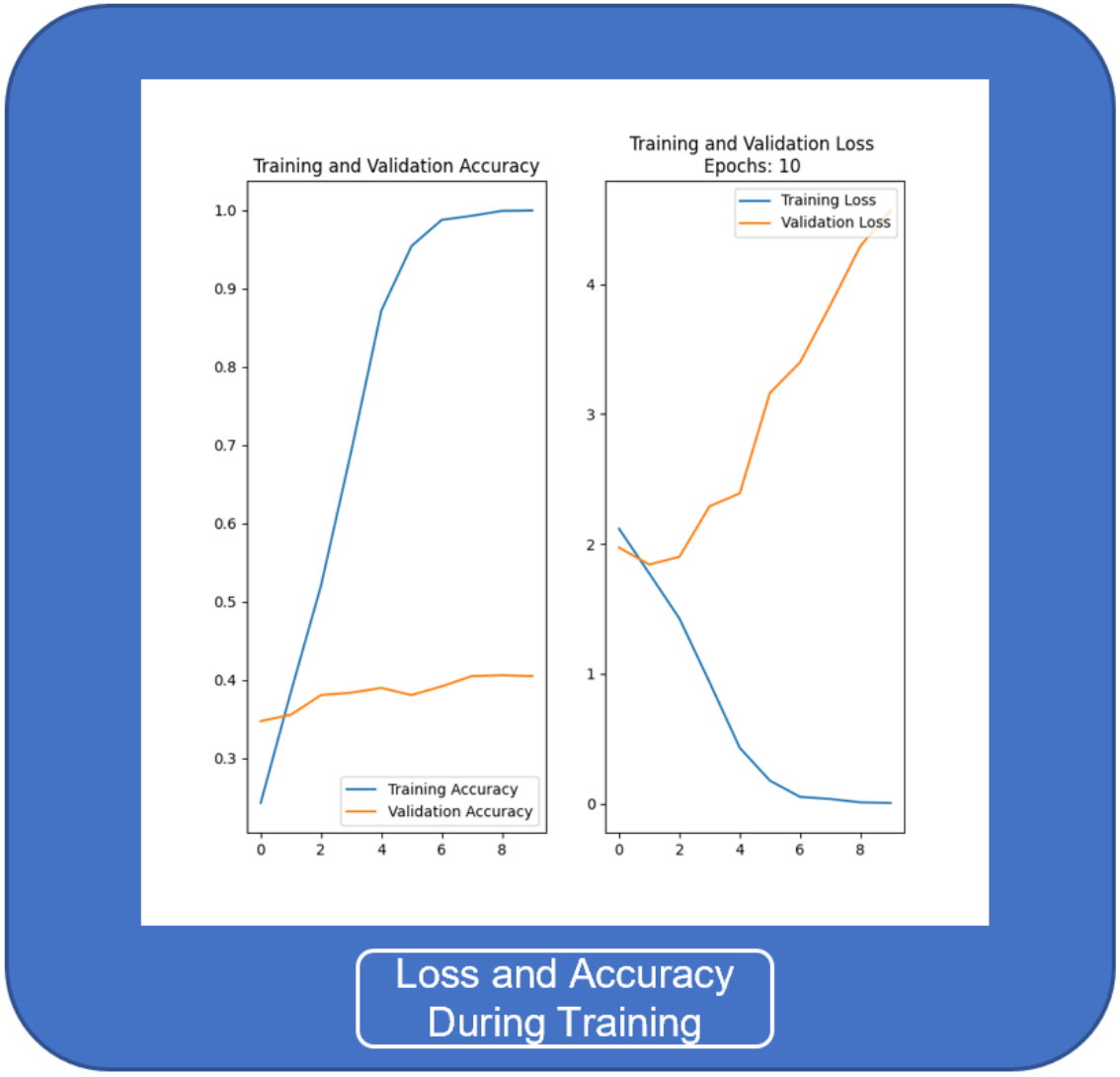

Image 4: Showing the training and validation accuracy and loss during training.

Self Assigned Portfolio - Neural Style Transfer

GitHub link for general repository:
https://github.com/MalteHB/visual_analytics_cds

GitHub link for the specific python script:
https://github.com/MalteHB/visual_analytics_cds/blob/main/src/neural_style_transfer.py

Project Description

For the self assigned portfolio assignment I wanted to assess the capabilities of utilizing pretrained models for neural style transfer. Specifically, to investigate the quality of the two different transfer learning style transfer methods. One being simply downloading and instantiating a pretrained style transfer model from the TensorFlow Model Hub, while the other is to instantiate the weights of a large pretrained image recognition model, and finetune it to create a stylized image.

Methods

For the pretrained style transfer model I chose the arbitrary image stylization model from the open source project, Magenta. Magenta was created by a team of researchers and engineers from Google Brain to explore the creative and artistic powers of machine learning ("Magenta," 2021; *Magenta GitHub*, 2021). As mentioned, I wanted to finetune my own neural style transfer model and this was done by using the pretrained weights from intermediate layers from the VGG19 model architecture pretrained on ImageNet data (Deng et al., 2009; Simonyan & Zisserman, 2015). Specifically, I used block five from the second convolution layer as the content layer and block 1-5 from the first convolution layer as the style layer. I chose to run the finetuning for 25 epochs with 100 training steps for each epoch and a learning rate at 0.005. There are multiple hyperparameters that could be optimized, and this is certainly a limitation of this project, however, implementing hyperparameter optimization was deemed as being outside the scope. The content image, being a selfie of the undersigned, and the style image, being *Cubist 9* by the artist Thomas C. Fedro (Fedro, n.d.), can be seen in Image 5.

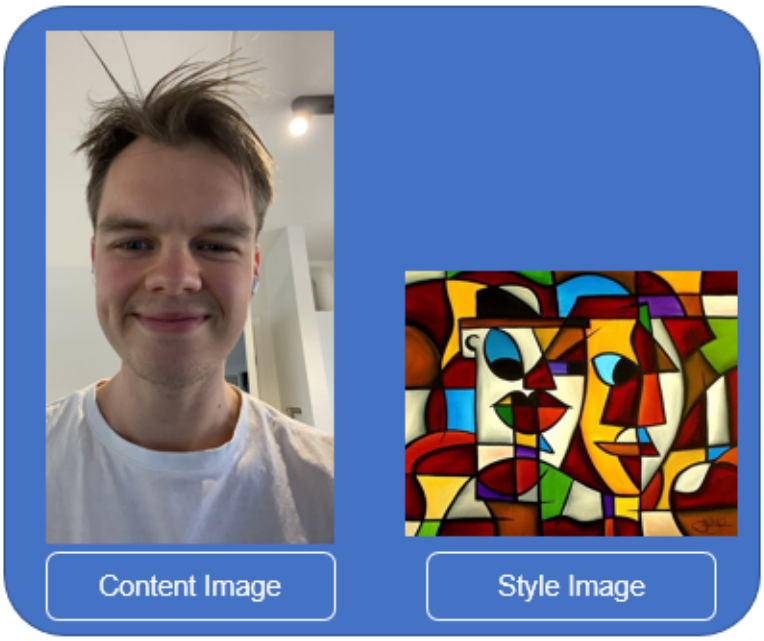

Image 5: The content image and the style image used for the neural style transfer.

Usage

To run the script developed for this project start by cloning the github repository:

```
# Clone the GitHub
git clone https://github.com/MalteHB/visual_analytics_cds.git
cd visual_analytics_cds
```

Moreover, I recommend installing Anaconda ("Anaconda Software Distribution," 2020) from their website (https://docs.anaconda.com/anaconda/install/) and using this to create a virtual environment through the 'conda' command:

```
# Create and activate conda environment:
conda create -n cds python=3.8
conda activate cds
```

Then install the requirements from the 'requirements.txt' file:

```
# Install requirements
pip install -r requirements.txt
```

The package *opencv* is also a requirement for the script to work, however, installing it through pip might cause issues and have different dependencies depending on the operating system used. I, therefore, recommend installing it by using the 'conda' command:

```
# Conda install packages
conda install opencv -y
```

Finally to run the script run the following lines:

```
# Run the neural style transfer script.
# See the available flags by using the '--help' after the command.
python src/neural_style_transfer.py

# To use the pretrained style transfer model from Magenta use the flag
'--pretrained'
python src/neural_style_transfer.py --pretrained
```

Discussion of Results

The two pictures produced by the arbitrary image stylization model from Magenta and by the finetuned VGG19 model can be seen in Image 6*, and the loss can be seen in* Image 7. According to the loss curve, it seems that, with the current hyperparameters, the model could not be much further optimized. When addressing the pictures, it seems that the Magenta model provides a better style transfer, in that the picture is much more close to the style of cubism and *Cubist 9*, while the picture the finetuned VGG19 model created does not even seem reminiscent of cubist art. Furthermore, while the VGG19 took a total of 3 hours, 21 minutes and 15 seconds on the computer used, the Magenta model only took 10 seconds. This investigation, therefore, concludes that, with the current images, the pretrained model for arbitrary image style transfer from Magenta provides higher quality image style transfer, compared to the finetuned VGG19. Future research should delve into the realm of hyperparameter optimization and, moreover, investigate how different styles of art could provide different quality outcomes.

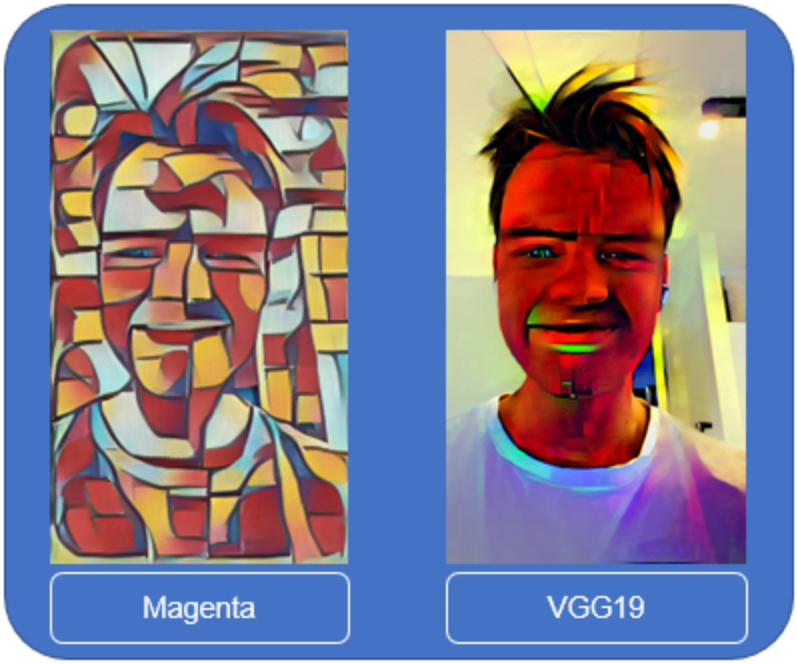

Image 6: The stylized image produced by the pretrained neural style transfer model from Magenta and the image produced by finetuning the specific layers from VGG19 for 25 epochs.

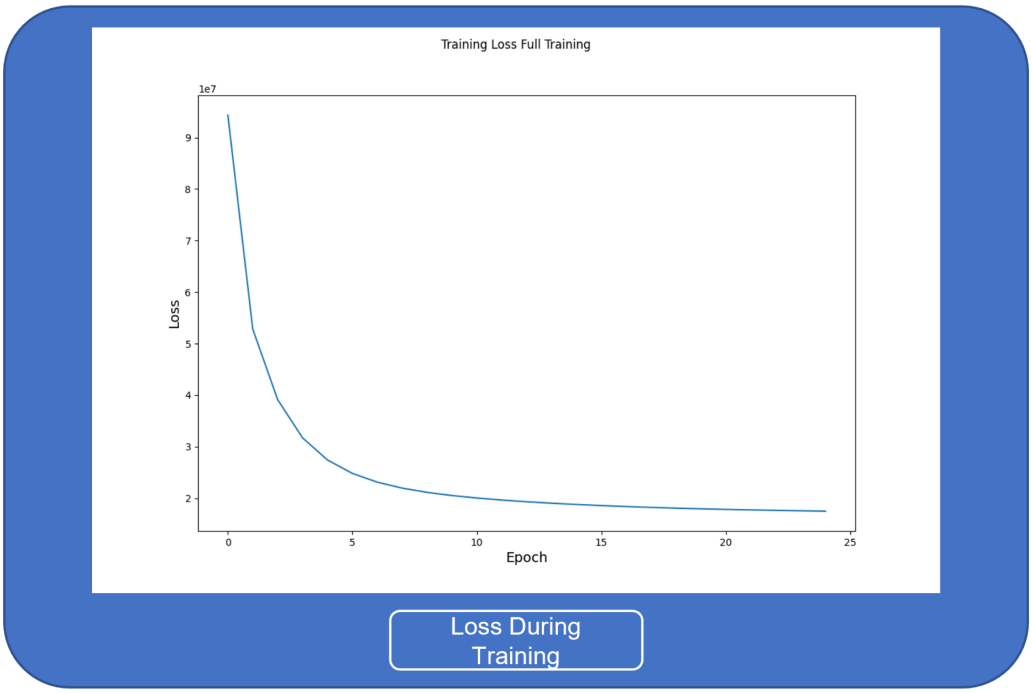

Image 7: Loss curve of the finetuning of the VGG19 model.

References

Anaconda Software Distribution. (2020). In *Anaconda Documentation* (Vers. 2-2.4.0) [Computer software]. Anaconda Inc. https://docs.anaconda.com/

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

Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, *8*(6), 679–698. https://doi.org/10.1109/TPAMI.1986.4767851

Da, K. (2014). A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Fedro, T. C. (n.d.). *Cubist 9*. Retrieved May 7, 2021, from https://www.ebsqart.com/Art-Galleries/Contemporary-Cubism/43/Cubist-9/204218/

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

LeCun, Y., & Cortes, C. (2010). *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/

Magenta. (2021, May 6). *Magenta*. https://magenta.tensorflow.org/

Magenta GitHub. (2021).

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale*

Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Icml*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rong, W., Li, Z., Zhang, W., & Sun, L. (2014). An improved Canny edge detection algorithm. *2014 IEEE International Conference on Mechatronics and Automation*, 577–582. https://doi.org/10.1109/ICMA.2014.6885761

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs]*. http://arxiv.org/abs/1409.1556