MLiP 24 - Monet Challenge

Bono IJpelaar (s1063058)
Matthijs Neutelings (s4804902)
Saskia de Wit (s1060762)

March 20, 2021

1 Introduction

For the course Machine Learning in Practice we have worked on a submission for the Kaggle competition "I'm Something of a Painter Myself"[1]. The goal of this challenge is to transform everyday images into Monet-esque paintings. This is done by taking a data set of Monet paintings as reference and transfer their style to the pictures from another data set.

2 Method

When we were introduced to the challenge we had several approaches for this challenge. By looking at previous notebooks within the challenge the CycleGAN approach is very dominant. The reason for this dominance can be related to the fact that the challenge description mainly handles GAN's. Also the supplied tutorial by Amy Jang[2] handles the CycleGAN approach step by step.

We researched the literature of different kinds of GAN's: CycleGAN[3], DiscoGAN[4], DualGAN[5] and Gated-GAN[6]. Besides GAN's we also researched a paper that handles style transfer with a Deep Neural Network[7].

Eventually we chose to approach this challenge with the Neural Style Transfer. The reason we chose this method was because in our opinion the results presented in the paper were visually the best compared to the GAN papers. Also this method was not the standard procedure for the challenge, so if we could make it work with the Neural Style Transfer this would be a more unique way of solving the challenge. Proving that this could work on a large scale (7000+ images) would be the first submission to achieve that, to the best of our knowledge.

We used the following algorithm, with each step explained in detail below:

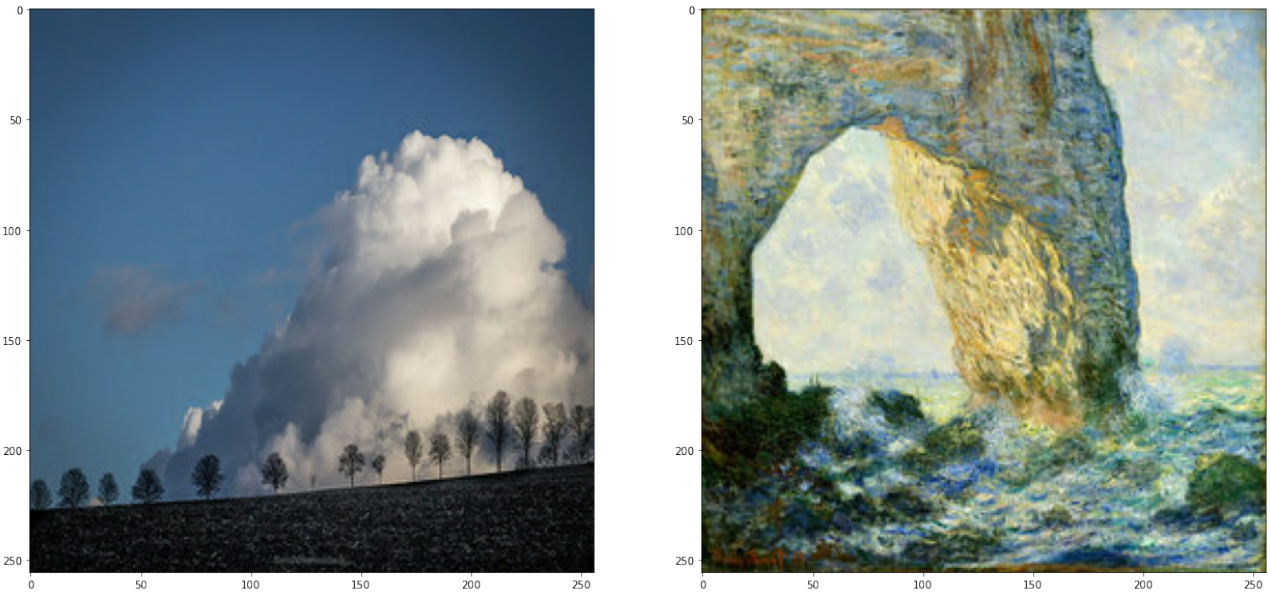
Algorithm 1: Neural Style Transfer

Data: Target photo p, Target Monet m
input : Initial image i, Step size k
output: Modified image i'
Result: i' matches the content features of p and style features of m
$C \longleftarrow$ content features from p;
$S \longleftarrow$ style features from m;
$steps \longleftarrow 0$;
while $steps < k$ **do**
    $steps \longleftarrow steps + 1$;
    calculate loss of i with respect to C and S;
    calculate gradient of loss function;
    apply one step of gradient descent to i;
end

A single iteration of the while loop is called a *train step*.

2.1 Data, input, output, result

Our data consists of a photo and a Monet painting whose features we want the input image to match. The challenge provides data sets of 7038 photos and 300 Monet paintings (figure 1). The resolution of both the photos and the Monet paintings is 256x256 pixels.



(a) Photo from the data set provided by Kaggle

(b) Monet from the data set provided by Kaggle

Figure 1: An example of a photo and a Monet

[8] used a white noise image as the input for their algorithm. For simplicity we use the photo that we want to match as input for the algorithm.

The input image is modified by the train step of the algorithm, and is then returned as output. This output should match the content features if the input photo, and the style features of the Monet painting.

2.2 extracting features

To extract both content and style features, we use a pre-trained Deep Neural Network, VGG-19 [9], created for the task of image recognition. We only need the 16 convolutional layers, and not the 3 fully-connected layers at the end because we will not be doing classification. To classify images, the convolutional layers from VGG-19 represent the content of an image. Activations at each layer represent the contents of the input image, with deeper layers representing the contents increasingly better. For this reason, extracting activations from one of the deeper layers will give us a content representation of an image. Suprisingly, VGG-19 can also be used to extract the style of paintings. [7] showed that we can extract the style representation by looking at the correlation between different layers of VGG-19. Like [7], we extract these correlations using a Gram matrix. In contrary to the content features, the style features can be extracted from any layer in the network.

2.3 train step

The first part of the training step is calculating the loss (that we want to minimize) of the current image. The loss function consists of two parts: a content-loss and a style-loss. For the content part, we extract content features from the target photo and the current image, and look at the mean-squared distance. Then, the loss is weighted by a hyperparameter for content loss. For the style part, we calculate the Gram matrix on five different layers in VGG-19 from both the target Monet painting and the current image. Then for each layer that these Gram matrices are calculated on, we calculate the mean-squared distance between the Gram matrix of the Monet and the current image. We then get five content loss scores. We take a weighted average of these, with the weights being another 5 hyperparameters. Finally, the style loss is weighted by a hyperparameter for style loss, and content-loss and style-loss are added together.

Next, we use a gradient descent algorithm with backpropagation to find the gradient of this loss function with respect to the current image. This gradient is then applied to the current image, in a way that minimizes the loss. The learning rate governs how fast the gradient descent algorithm converges to a local minimum. We use this as another hyperparameter for our experiments.

Type	Steps	L. rate	Content weight	Style weight	Style weights	Monet	Score
GPU	100	0.05	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6844870
GPU	170	0.03	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	285	7362323
GPU	100	0.04	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	2	6919887
GPU	170	0.015	1e4	1e1	{0.45, 0.3, 0.15, 0.05, 0.05}	44	5958377
GPU	100	0.04	1e4	1e-2	{0.45, 0.3, 0.15, 0.05, 0.05}	0	7499393
TPU	1000	0.001	1e3	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6036489
GPU	100	0.04	1e4	1e1	{0.45, 0.3, 0.15, 0.05, 0.05}	0	6933427
TPU	2000	0.0005	1e3	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6703379
TPU	1000	0.001	1	1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6015260
TPU	1000	0.001	1e4	1e1	{0.45, 0.3, 0.15, 0.05, 0.05}	0	6265801
TPU	1000	0.001	1e1	1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6011541
GPU	170	0.04	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	6272901
TPU	100	0.01	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	0	5950373
TPU	100	0.01	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	44	5487280
TPU	10	0.01	1e4	1e1	{0.7, 0.19, 0.24, 0.11, 0.26}	44	5950373
TPU	0	NA	NA	NA	NA	NA	30507079

Table 1: Kaggle results of parameter tuning

2.4 experimental setup

We chose a baseline notebook that is based on the Neural Style Transfer tutorial of TensorFlow[8]. The original state of this notebook transferred a single content image to the style of a chosen style image. We ran the notebook with a GPU that was somewhat slow. A single image transfer took about 2 minutes. We looked into the possibilities Kaggle offered us for processing units and decided to try and rewrite the code to let it use TPU's since these processing units would be faster in bigger batches[10].

Since TPU's work differently from GPU's we started rewriting the structure of the existing code to apply the same work on a TPU. This can be considered as one of the biggest changes we made on the baseline notebook, but this was also easier said than done. The way that the TPU handles its calculations in the process differs from that of a GPU. The TPU does its calculations simultaneously in batches, if the batching strategy is not implemented correctly, the calculations of different images will collide with each other giving strange results.

Eventually we introduced two versions of our code. One being the TPU version, which ran through the 7000 images in the data set in approximately 1.5 hours and a GPU version that almost reaches the maximum of the 5 hours allowed. The original GPU version took about 2 minutes per image, but had 1500 training steps per image. We plotted the loss function for this approach and with that validated that the loss isn't decreasing too much anymore after about 200 steps. We calculated what the maximum steps could be for the 5 hours of run time that was allowed, which gave us about 170 training steps. This was reasonably close to the sweet spot in our loss visualization, so we tried to work with this. We changed the baseline notebook to work with a big data set of 7000+ images and found that the amount of train steps didn't need to be this high in combination with a higher learning rate.

3 Results

Several combinations of submissions were made, to see which parameters had to be tuned for the best results. The parameters we tried to tune are: the number for steps, the learning rate, the content weights, the style weight(s) and the monet. These were tuned for the GPU and the TPU. After running the notebook and looking at it visually it was sometimes hard to see differences, therefore some submissions have been made to see what gives the best results. In table 1 these results can be found. In the row that is marked green, the parameters for the best submission can be found. The row that is marked orange gave some strange results, as the style- and content weight are both 1, but the score is also close to the best score.

We can see some variation in the scores, implying that the parameter tuning is important for the results. In the second row we see that the learning rate is a bit lower than in row one and three, but is does take more steps. Besides that the only change is the monet choice. In the best submission, the learning rate was lower than in other submission, besides that the style weight and monet were changed. In the fifth submission the content weights were set higher, but this resulted in the overall worst score. For the TPU the scores are a bit closer to each other, even though the variation in the parameters seems to be higher.

4 Discussion

Although we expected that calculations on both GPU and TPU should be the same, since we used the same method for both, the inherent workings of GPU and TPU differed in such a way that the same method executed over both methods gave totally different results. this can be seen in Table 1 by comparing the same steps, weights and chosen Monet painting, giving completely different scores. Also on a TPU having 1000 steps with a lower learning rate gave a lower score than the TPU method with 100 steps and a higher learning rate. One thing we can say for sure is that the choice of the Monet does matter. When we chose a Monet painting that with lesser contrasts, the algorithm performed better. Eventually the TPU version performed better under certain circumstances. We assume that the GPU version performed less significant because we were limited in the amount of time a notebook was allowed to run.

5 Conclusion

In this report we tried to apply a Neural Style Transfer method for transferring normal images to a Monet-esque style with Monet paintings as reference. We did this with both a GPU as a TPU version, since the transferring was computationally heavy per train step. In this way we were limited by a run time threshold from Kaggle, which made our GPU version a little bit weak. Our TPU version was faster but gave different results. Eventually we did our best submission with this version by choosing the right Monet painting and weights. Our final notebooks can be found in the appendix attached with this project.

For a future version we were thinking about implementing a search function that compared the set of Monet paintings with the set of content images and find the most suitable Monet painting for each content image. We think that it will improve the algorithm if the style and content image are already similar.

6 Author Contributions

At the start of the project we worked together on fixed moments in the week to first research our options. Then we started with the TPU version for our baseline notebook. In our meetings we looked at the code together to try and fix bugs. In our own time we also tried to do this, Matthijs made the most progress on this. Later on it was a bit harder for Saskia and Bono to catch up, because Matthijs understood his own code the most but we could work forward this. When we finally got our code base working completely, Saskia and Bono did several submissions for the GPU version. Saskia and Matthijs also did several submissions for the TPU version. In the end Bono reviewed the TPU and GPU notebooks and added some documentation to them. We equally contributed on the project report and in general we think there was a fair distribution of the workload for the project as a whole.

7 Evaluation of the Process

Overall, the process went well. At the start of the project we had a clear idea of what we wanted to do, we quickly found a notebook on which we wanted to base our notebook. After a short while we had our first problems. Errors we occurring and it took several weeks to fix these. At one point we had so many different notebooks that it was hard to find the 'right' one. We decided to clean everything and make sure all three of us had access to the working notebooks. Every Thursday we came together to work on the challenge, but usually we also worked on it separately. When we ran into problems we contacted Alex. Looking back, we probably spend too much time on the TPU version where we should have spend time on making the running time of the GPU version better. All three of us had no experience with TPU's and GPU's and therefore we did not know how to work with these. Because we spent so much time on getting the TPU version to work we did not have much time in the end to experiment with all the different parameters of the style transfer.

8 Evaluation of the Supervision

We spent most of our time trying to get our first submission to work. We think the teachers and the coach would have been more useful for questions that were related to the direction of the research instead of code-related questions. Because our questions were code-related, the teacher meetings were not very useful to us. The coach meetings were a bit more useful, because we were able to compare our progress with different groups. The biggest help by far came

from our coach Alex who went with us through our code several times and pointed out where our potential mistakes were.

A downside of the coach meetings was that we were working with TensorFlow and Alex had little experience with that, but instead preferred PyTorch. He did offer us to rewrite our code so it used PyTorch but couldn't seem to get that to work. Eventually Alex had enough general knowledge to help us with debugging and validating our codebase. Another minor drawback for us was that during the teacher meetings- as well as the coach meetings, no one could tell us about the difference between the results for the GPU and the TPU. During the last coach meeting one of the other teams addressed that they got different results for the GPU and TPU, by which it all finally made sense to us. It would have been nice if one of the teachers or our coach could have helped us with that.

References

[1] Kaggle, "I'm something of a painter myself," [Online]. Available: `https://www.kaggle.com/c/gan-getting-started`.

[2] A. Jang, "Monet cyclegan tutorial," [Online]. Available: `https://www.kaggle.com/amyjang/monet-cyclegan-tutorial`.

[3] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," [Online]. Available: `https://arxiv.org/pdf/1703.10593.pdf`.

[4] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," [Online]. Available: `https://arxiv.org/pdf/1703.05192.pdf`.

[5] Z. Yi, H. Zhang, P. Tan, and M. Gong, "Dualgan: Unsupervised dual learning for image-to-image translation," [Online]. Available: `https://arxiv.org/pdf/1704.02510.pdf`.

[6] X. Chen, C. Xu, X. Yang, L. Song, and D. Tao, "Gated-gan: Adversarial gated networks for multi-collection style transfer," [Online]. Available: `https://arxiv.org/pdf/1904.02296.pdf`.

[7] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," [Online]. Available: `https://arxiv.org/pdf/1508.06576.pdf`.

[8] TensorFlow, "Neural style transfer," [Online]. Available: `https://www.tensorflow.org/tutorials/generative/style_transfer`.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," [Online]. Available: `https://arxiv.org/pdf/1409.1556.pdf`.

[10] Kaggle, "Tensor processing units (tpus)," [Online]. Available: `https://www.kaggle.com/docs/tpu`.