Landscape image inpainting using generative adversarial networks.

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

Department of Telecommunications and Media Informatics
2020/2021 Autumn semester

Kovács Richárd – YKPIQV
Dulácska Mátyás – EA2WTZ
Peőcz Krisztián – G904FJ

Our goal was to create a neural network that is capable of filling in blank spaces in images, which are depicting landscapes or other types of natural sights. We tried to recreate the model of [7] with some additional changes to see the results we can get out of it, and to examine how the hyperparameters affect the image quality.

1 Introduction

Image inpainting has been a topic of interest for a long time. Its application is widespread, including photo restoration, and removal of unwanted objects. Despite its significance, image inpainting has remained difficult, and there have been numerous algorithms developed to complete this task [1]. Unfortunately, these algorithmic solutions could only work properly with narrow areas that needed to be restored, such as scratches on old photographs.

With the birth of neural networks, many more approaches came out to complete the task of image inpainting such as [2], as well as autoencoders [3]. Convolutional autoencoders - so called context encoders - were used [4] to inpaint images, and it also uses an adversarial type loss, similarly to the generative adversarial networks (GANs) [5]. With the appearance of GAN, many new propositions came to complete the task of image inpainting and restoration. For example, an approach uses GAN to find the best input of the generator network, which would provide the most similar picture to the original one [6]. There is also an approach that use context encoders as generator networks [7], and we strived to make a network similar to this one. There is a proposition that also uses contextual attention in their network [8].

2 Approach

While constructing the network we relied heavily on [7]. We did make some changes however, to simplify and to experiment with the network.

Structure:

At the most basic level, the network consists of two parts: a generator and a discriminator. The generator network has a kind of autoencoder structure, its purpose is to generate the inpainted picture. The generator's input is a picture with a black region on it, and its output should be the inpainted image.

The discriminator network has two parts: a local and a global discriminator, which together give the final output of the discriminator network. The input of the discriminator network is either a generated or an original picture, and the output of the network represents the discriminator's decision about the images' realness. The output is close to 1 if the input image is original, and 0 if the image has been generated.

The generator network:

The generator network is based on a fully convolutional network, it is similar to an auto-encoder, therefore it has three parts: an encoder, a bottleneck and a decoder.

The first part of the network is the decoder. It reduces the resolution of the picture. For this process instead of using pooling layers, we use strided convolutions. With two strided convolutional layers we decrease the resolution to the quarter of the original resolution.

The second part of the network is the bottleneck, which consists of convolutional and dilated convolutional layers in order to widen the network's perceptive field.

The third and final part of the network is the decoder. The key point here is to increase the tensor's sizes in spatial dimensions so that the output resolution will be equal to the resolution of the input image. For this purpose, we used transposed convolutional layers [9].

The inputs of the network are different from the one in the [4]. Our inputs are 64x64 pixel RGB pictures with black rectangles, which are 20x20 to 25x25 pixel big. We did not make uniform sized black regions, so the size of the part, that needs to be inpainted is approximately between the 10 and 15 percent of the whole image. The output of the network is the reconstructed image, which itself is a 64x64 pixel RGB image – meaning that is has the same dimensions as the input image.

The discriminator network:

The discriminator network has two parts: the global and the local discriminator. These need different inputs, which then are fused together to provide an output.

The global discriminator's input is a 64x64 pixel RGB picture, which can be either generated or an original one. This part of the network consists of convolutional layers.

The local discriminator's input is a 28x28 pixel RGB picture, which is a part of the picture which is given to the global discriminator. This 28x28 pixel part contains the slightly smaller 20x20 to 25x25 part which by now should be inpainted by generator if it is a generated picture. If the input is an original picture, then the local discriminator's input is just a randomly selected region with the appropriate dimensions. This part of the network also only consists of convolutional layers.

Both discriminators' convolutional layers are followed by flatten and fully connected layers in [7]. However, we found the concept of global average pooling intuitive, which provides an intermediate step between convolutional and fully connected layers in classification tasks [10], so alongside using only fully connected layers, we also experimented with global average pooling layers with one fully connected layer (both in the case of local and global discriminators).

In final layers of the discriminator network the outputs of local and global discriminator are concatenated and fused in a final fully connected layer that has only one value as output – it determines whether the picture was original (output of 1) or generated (output of 0).

Dataset:

We created our database using the already existing Places365-Standard database [12]. We generally wanted to use pictures depicting landscapes or other types of natural sights, therefore we went through the 365 classes of the original database and picked 50 classes, which seemed to represent the types of pictures we need. After this we had approximately 250k images.

Unfortunately, many of these pictures had people in them, so we had to take out those pictures, to make the network's task easier. We thought that inpainting people would be too hard for our network. To filter out the pictures with people on them, we used a YOLOv3 [11] neural network. This meant about a 20% reduction in the number of images, leaving us with a total of 203 thousand.

We downscaled the 256x256 images by a factor of 4, resulting in an image dimension of 64x64. This was necessary to make the network training faster on our limited resources. We also distributed the pictures into three groups: train, validation, and test images. The percentage of each group is 85, 10, and 5 percent respectively, which means we have about 170k train, 20k validation and 10k test images.

We also overwritten parts of the images with black patches. These are rectangles, with a size ranging from 20x20 to 25x25 pixels. This means, that about 10 to 15 percent of the original picture is covered. While the black patches could vary in size, the local discriminator needs a uniform input, so the actual input size is 28x28 and contains some of the input image. For this reason, in the beginning, the generated image is not entirely black and allows the network to start learning. With this bigger crop we hope that the local discriminator could not only examine the consistency of the generated part, but also the consistency of the border between the generated and the original part of the image.

Figure 1. Samples from the dataset.
From left to right: original image, image to be inpainted, and the part that has been cut out.

After this process, every section of our data - train, validation, and test - had three groups of pictures. The original 64x64 pixel pictures, the cropped pictures (with the black rectangles in them), and the crops (the 28x28 pixel pictures which were cropped out from the original pictures). The latter provides the inputs of the local discriminator. Finally, the exact coordinates of the black regions and the 28x28 crops are saved in a csv file.

Training in theory:

The training of the network happens in three different phases in each epoch. The proportion of these phases is a hyperparameter.

$$L(x, M_c) = \| M_c \odot (\underbrace{C(x, M_c)}_{\text{Generator output}} - \underbrace{x}_{\text{Real image}}) \|^2 \tag{1}$$

In the first phase the generator is trained with the cropped images. During this phase, our loss function is a MSE loss calculated by the difference between the original picture and the generated one. The MSE loss is supplemented with a binary mask M_c in [7] (which is to take only the cropped part into consideration), however, we decided to set all the values of M_c mask to one. With this approach we want not only the completion of the cropped region but also the reconstruction of the whole image, so we expect an "end-to-end" solution from the network.

$$-[z \cdot \log \underbrace{D(x, M_d)}_{\text{Discriminator output}} + (1 - z) \cdot \log(1 - \underbrace{D(C(x, M_c), M_c)}_{\text{Combined model output}}] \tag{2}$$

Label for real: z — Label for generated: $1 - z$

The second phase is about training the discriminator. We are only training the discriminator, with original pictures and generated pictures. The loss function in this part of the training is

binary cross-entropy, which is a suitable choice if we want to do classification with two classes and it is related to the key concept of adversarial training [5]. (2) shows the exact formula used in [7] as well as in our case for discriminator losses in phases 2 & 3.

$$\left[w1 \cdot \underbrace{L(x, M_c)}_{\text{MSE loss}} + w2 \cdot [-(1 - z) \cdot \log \underbrace{D(x, M_d)}_{\text{Discriminator output}} - z \cdot \log(1 - \underbrace{D(C(x, M_c), M_c)}_{\text{Combined model output}})]\right] \quad (3)$$

Label for generated

Label for real

In Phase 3 we do two things:

- train the discriminator similarly to the second phase
- train the generator with the help of the discriminator

In case of the latter, we take the forward step for the discriminator, but only the generator's weights are changed in the backward step. The generator's loss function here is a joint loss that contains both the weighted MSE loss used in phase 1 and the weighted binary cross entropy loss [7]. Formula (3) shows the exact implementation. It is worth mentioning, that we used the same loss function as for the discriminator, we only swap the labels of real and generated images – this way we train the generator to fool the discriminator.

Training in practice

Our final network has more than 34 million trainable parameters, which makes the training computationally very demanding, and it requires a great deal of processing power. This kind of processing power was only obtainable for us through Google Colaboratory. Therefore, we optimized our training to be used in conjunction with this service.

With the processing power of Google's GPUs, we could train our networks with reasonable efficiency. Even though now the process was significantly faster, one epoch could still take up to 30 minutes. We observed a limitation, that we could only train for the network about 9 hours, before it got shut down. This meant, that the current network parameters had to be stored, so we can continue where the training was halted.

For testing purposes, we usually only did around 10 to 20 epochs. After each epoch, we tested the network with a total of 30 images from the validation dataset and saved the results. For this reason, we were able to check how well the network was doing, and it provided a kind of subjective evaluation.

Hyperparameter optimalization and evaluation

The hyperparameter optimalization and the evaluation of the network was a challenging task. The limited access to high-performance computing resources meant, that we could only investigate a small number of variations. Furthermore, objective evaluation methods proved to be ineffective, because we found, that a good validation loss does not necessary equate to a good quality inpainting – and this makes selecting the best performing epoch non-trivial.

We did notice some patterns in the loss values, that was indicative of the network doing good, but including humans to provide subjective feedback was the only reliable method.

Because of the stated reasons, we had to do the hyperparameter optimalization by observing the effect of the change of each hyperparameter individually. We let each network train for 10 epochs. We are aware of the limitations of this method, but it was a necessary step to continue

development. Low sample size hyperparameter optimalization could be the subject of research at a later date.

Although there are several established ways of rating image quality, we formulated, that in our case assigning a local and a global score would work best. After we chose the seemingly best epoch, we looked at the inpainted pictures and gave them scores according to the quality of the inpainting. Local score was a measure of the inpainted region's quality, while global score described that how well the image as a whole was reconstructed.

Scores were assigned from a scale of 1 to 10, where 10 means that the generated images are perfect and indistinguishable from the original ones. Score of 1 is the opposite, and it is indicative of an awful reconstruction quality.

We defined the base network with the following parameters:

- Ratio of phases: 0.33-0.33-0.34
- MSE:ADV ratio: 1:1
- Batch size = 128
- Discriminator global average pooling: false
- Generator bottleneck = 4

The results of the evaluation could be seen in this table:

Table 1. Subjective ratings of the networks we trained. Scores are 1 to 10, where 10 is indistinguishable from the original image.

Hyperparameter changed	Local score	Global score	Average score
Original	3,67	5,33	4,50
Phases: 0.2-0.5-0.3	3,67	6,00	4,83
Phases: 0.2-0.3-0.5	3,00	4,33	3,67
MSE:ADV = 10:1	4,67	6,33	5,50
MSE:ADV = 100:1	3,66	4,33	4,00
Batch size = 256	3,67	5,33	4,50
Global avg pooling	2,67	3,33	3,00
Bottleneck = 3	3,67	4,33	4,00

According to our evaluation, tweaking the ratios of the phases to 0.2-0.5-0.3, and modifying the proportion of the joint loss as to MSE:ADV = 10:1 would benefit the network. Changing the batch size had no observable effect, and all other tweaks would decrease the performance of the network.

After this evaluation we did one final training, and after that we chose the best model to present the final results with.

3 Results

We configured the network to have the advantageous parameters and we let it train for about 24 hours (220 epochs). Due to resource limitations, we did this in three parts.

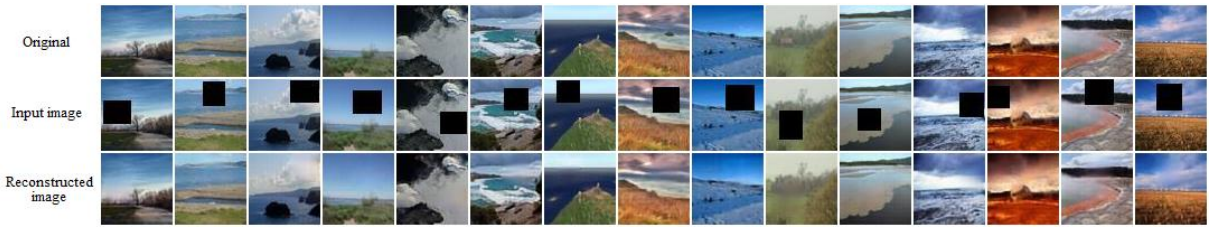

Figure 2. Examples of well reconstructed images from the test dataset.

Figure 2 shows examples, in which the input image has been well reconstructed. In the case of simple, low-noise images, the reconstruction can be quite hard to spot, and is often an impossible task without zooming in. Rectangle shaped slight changes in colour and sharpness are indicative of a reconstructed image versus an original one. However, for those who are uninitiated, these artifacts would go very likely unnoticed.

Figure 3. Failures of the image reconstruction.

Figure 3 shows the limitations of the network in its current form. The network has a hard time reconstructing images with significant or unpredictable noise, such as trees or wildlife in the foreground. Delicate, but contextually important details (such as the trees in the second column) are sometimes lost.

Figure 4. Applications of image reconstruction.

We already mentioned several uses for image reconstruction. In Figure 4, you can see a variety of these applications. The network was used to remove unwanted people, objects, text and faults. In most cases, the results are fairly good. The network is also capable of reconstructing multiple isolated regions, but struggles to properly fill linear faults, like scratches.

4 Summary

Generative adversarial network is one of most important machine learning breakthroughs of the last few years. We compiled a dataset containing natural sights and we were able to train a network, which was capable of inpainting landscape pictures. The effects of different hyperparameters were investigated and measured. We also proved, that our solution works well enough to be used in a variety of use cases.

The network could be improved upon in several ways. The most obvious would be to increase the input size. This would not only make the results more impressive, but also more useful, since we are no longer limited to small images. There is most likely still some untapped

potential in hyperparameter optimalization, which would further increase the performance of the network. Both ideas require more computational power, so we hope, that as computing power becoming more and more abundant, training a GAN network will be more accessible to all.

The source code and further documentation is available on GitHub.

5 References

[1] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. 2000. *Image Inpainting*. In ACM Transactions on Graphics (Proceedings of SIGGRAPH). 417–424.

[2] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. 2017. *High Resolution Image Inpainting using Multi-Scale Neural Patch Synthesis*. In IEEE Conference on Computer Vision and Pattern Recognition.

[3] Junyuan Xie, Linli Xu, and Enhong Chen. 2012. *Image Denoising and Inpainting with Deep Neural Networks*. In Conference on Neural Information Processing Systems. 341–349.

[4] Deepak Pathak, Philipp Krähenbühl, Je Donahue, Trevor Darrell, and Alexei Efros. 2016. *Context Encoders: Feature Learning by Inpainting*. In IEEE Conference on Computer Vision and Pattern Recognition

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. *Generative Adversarial Nets*. In Conference on Neural Information Processing Systems. 2672–2680

[6] Raymond A. Yeh, Chen Chen, Teck Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, Minh N. Do; Proceedings of the IEEE Conference on *Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5485-5493

[7] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. *Globally and Locally Consistent Image Completion*. ACM Trans. Graph. 36, 4, Article 107 (July 2017), 14 pages.

[8] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, Thomas S. Huang; Proceedings of the IEEE Conference on *Computer Vision and Pattern Recognition* (CVPR), 2018, pp. 5505-5514

[9] V Dumoulin, F Visin: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016 - arxiv.org

[10] Min Lin, Qiang Chen, Shuicheng Yan: Network In Network. arXiv preprint arXiv:1312.4400, 2013 - arxiv.org

[11] J.Redmom, A.Farhadi: YOLOv3: An Incremental Improvement. arXiv:1804.02640, 2018 - arxiv.org

[12] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba: A 10 million Image Database for Scene Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017