

Image inpainting using a generative model.

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 on pictures, which are depicting landscapes or other types of natural sights. We tried to recreate the model of [ref] with some additional changes to see the results we can get out of it, and to examine how the hyper parameteres affect the training.

1. Introduction

referencia ötletek:

Image inpainting has been a topic of interest for a long time. There were approaches that used algorithms to complete this task [1]. Unfortunately these algorithmical solutions could not work properly with narrow areas that needed to be inpainted such as the ones on old photographs.

With the birth of neural networks many more approaches came to complete the task of image inpainting such as [2]. Another example is autoencoders, which were also used to complete this task [3]. Also convolutional autoencoders, so called context encoders were used [4] to inpaint images, it also uses an adversarial type loss which was presented with the generative adversarial networks [5]. With the appearance of GAN many new propositions came for this task. For example a approach which uses GAN to find the best input of the generator network, which would provide the closest picture to the original one [6]. Using There is also an approach that use context encoders as generator networks [7], we tried to make a network very 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 the [reference]. We made some changes however, to make the network a bit simpler and to experiment with the network.

Structure:

The network, we are using basically has 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 patch 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 should be a generated or a original picture, and the output of the network should be 1 if the picture is original or 0 if the picture is fake.

The generator network:

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

The first part of the network is the encoder. 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 also dilated convolutional layers in order to widen the network's receptive field.

The third and final part of the network is the decoder. It increases the resolution of the picture. For this purpose we are using transposed convolutional layers.

The inputs of the network is different from the one in the [reference]. Our inputs are 64x64 pixel RGB pictures with black patches, which are (20-25)x(20-25) pixel big. We didn't 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 picture. The outputs of the network are 64x64 pixel RGB pictures which are hopefully completed by the network.

The discriminator network:

the discriminator network has two parts, the global and the local discriminator, which need different inputs, which then fuse, and provide an output together.

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 discriminators'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 consists of the smaller (20-25)x(20-25) part which by now should be inpainted by generator if it is a generated picture. It is just a part of the picture if the input of the global discriminator was a original picture. This part of the network also only consists of convolutional layers.

The outputs of the global and local discriminators are then merged with a flatten layer or with a global average pooling layer. The global average pooling layer is an addition to the network we propose.

The network also has fully-connected layers after the flattening or pooling. the output of the fully connected layers is also the output of the discriminator network, which determines whether the picture was a original one or a generated one.

Database:

We created our database using the already existing Places 365 database. We generally wanted to use pictures depicting landscapes or other types of natural sights, in order to do that, we went through the 365 classes of the places 365 database and picked 50 classes, which seemed to represent the type of pictures we need. The pictures from these classes depicted natural sights. After this we had approximately 250k pictures.

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 find the pictures with people on them we used a yoloV3 neural network. After this kind of cleaning we had around 200k pictures.

We resized the originally 256x256 pixel pictures to 64x64 pixel pictures, in order to make the training of the network faster. We also distributed the pictures into three groups: train, validation and test pictures. The percentage of each group respectively is 85, 10, and 5 percent which means we have 170k train, 20k validation and 10k test pictures.

We also made the black patches on the pictures. These patches are (20-25)x(20-25) pixel big, which means the patch consists of between the 10 and 15 percent of the original picture. We didn't make uniform sized black patches, so the black patches size could vary. The black patches could vary but the local discriminator needs a uniform input, so we had to make around these (20-25)x(20-25) patches a bigger crop, which also consists of some of the original picture, so not entirely black in the beginning. 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 generated part and the original part of the picture.

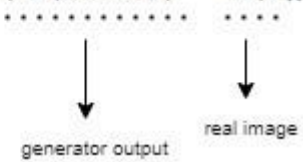
After this process every section of our data: train, validation and test, had three group of pictures. The original pictures are the original 64x64 pixel pictures, the cropped pictures are the original pictures with a (20-15)x(20-25) black patches in them, and the crop pictures are 28x28 pixel pictures which were cropped out from the original pictures, and the inputs of the local discriminator.

Of course the exact coordinates of the (20-25)x(20-25) and the 28x28 crops are saved in a .csv file.

Our database is accessible though a link which can be found in our github repository.

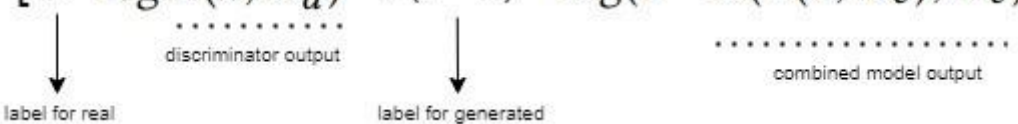
Training in theory:

The training of the network is has three different phases every epoch:

$$L(x, M_c) = \| M_c \odot (C(x, M_c) - x) \|^2$$


1. Figure: Generator loss for phase 1 based in [7]

The first phase is about training the generator. We are only training the generator with the cropped images. During this phase our loss function is a MSE loss between the original picture and the generated one. The MSE loss is supplemented with a binary mask M_c in [ref 4] (which helps to take only the crop part into consideration), however, we decided to set all the values of M_c mask to one. With this approach we expect 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 the following from the network.

$$- [z * \log D(x, M_d) + (1 - z) * \log(1 - D(C(x, M_c), M_c))]$$


2. Figure: Discriminator loss for phase 2 based on [7]

The second phase is about training the discriminator. We are training the discriminator alone with original pictures and generated pictures. The loss function at this part of the training is binary cross-entropy, which is a suitable choice for the loss if we want to do classification with two classes and it is related to the key concept of adversarial training [ian goodfellow eredeti GAN cikk]. It's worth to mention that we used the inverted form of the presented one on the figure to phrase the task as minimization.

$$\begin{aligned}
& - [z * \log D(x, M_d) + (1 - z) * \log(1 - D(C(x, M_c), M_c))] \longrightarrow \text{discriminator loss} \\
& \quad \downarrow \quad \text{discriminator output} \quad \downarrow \quad \text{combined model output} \\
& \quad \text{label for real} \quad \text{label for generated} \\
\\
& [w1 * L(x, M_c) + w2 * [- (1 - z) * \log D(x, M_d) - z * \log(1 - D(C(x, M_c), M_c))]] \longrightarrow \text{generator loss} \\
& \quad \downarrow \quad \downarrow \quad \text{discriminator output} \quad \downarrow \quad \text{combined model output} \\
& \quad \text{MSE loss} \quad \text{label for generated} \quad \text{label for real}
\end{aligned}$$

3. Figure: Losses for phase 3 based on [7]

The third phase is about:

- Train the discriminator similarly to the second phase.
- train the generator with the help of the discriminator: here we take the forward step for the discriminator, but only the generator's weights are changed in the backward step. This is way we train the generator to fool the discriminator. The loss function at this part is a joint loss that contains both the weighted MSE loss used in the first phase and the weighted binary cross entropy loss. It's also worth to mention that we use the same loss function as used for the discriminator, we only swap the labels of real and generated images.

The proportional length of each phase is a hyperparameter of the network.

Training in practice

Our final network has more than 34 million trainable parameters, which makes the training quite time consuming, and requires a great deal of processing power. This kind of processing power was only obtainable for us through google colab. Therefore we optimized our training for google colab and used our google drives to store the required data.

With the processing power of the GPUs of google colab we could make trainings in a reasonable time. Even though now the process was faster, one epoch could still take more than 30 minutes. With the limitation of google colab, that we could only use a GPU constantly for 9 hours, our trainings couldn't be too long. We usually did around 10 or 20 epochs. After each epoch we saved 5 times 5 inpainted picture to be able to check how the network is doing, it was a kind of validation for us.

Hyper parameter optimization and evaluation

The hyper parameter optimization and the evaluation of the network was a quite difficult task. Partly because of the limited access to the google colab GPUs, and also due to the fact that a good loss of the network doesn't necessarily means that the network is good at inpainting images. Of course we realized, that some patterns in the loss of the network could mean that the network is doing good, but the real test that can determine the performance of the network is the image inpainting it is capable of. This could also mean, that the last epoch of the network is not necessarily the best, which makes the saving of the network also a challenge.

Because of the stated reasons, we had to do the hyper parameter optimization by checking the effect of the change of each hyper parameter individually. We did this by changing one

hyper parameter of the original network structure before every training. With the new hyper parameter combination we did 10 epoch each.

Although we found evaluation methods published, we decided that we would evaluate the training's best epoch with the following method: we chose the best epoch subjectively and then looked at the inpainted pictures and gave them scores according to the quality of the inpainting. We used two categories: quality of the inpainted zone, and quality of the whole pictures. From these categories a 10 would mean perfect, and a 1 would mean utterly bad.

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

I. Table: hyper parameter optimization results

| hyper-parameter | local score | global score | average score |
|--------------------|-------------|--------------|---------------|
| target image | 10 | 10 | 10 |
| original structure | 3,67 | 5,33 | 4,5 |
| phase: 0.2-0.5-0.3 | 3,67 | 6 | 4,835 |
| phase: 0.2-0.3-0.5 | 3 | 4,33 | 3,67 |
| MSE:ADV = 10:1 | 4,67 | 6,33 | 5,5 |
| MSE:ADV = 100:1 | 3,66 | 4,33 | 4 |
| batch size: 256 | 3,67 | 5,33 | 4,5 |
| global avg loss | 2,67 | 3,33 | 3 |
| bottleneck: 3 | 3,67 | 4,33 | 4 |

According to our evaluation, changing the ratios of the phases to 0.2-0.5-0.3, and changing the proportion of the joint loss to MSE:ADV = 10:1 would benefit the network. Whereas changing the batch size doesn't have a really big effect on the performance. We could also see, that shortening the bottleneck part of the generator, using global average pooling instead of flattening, changing the ratios of the phases to 0.2-0.3-0.5, or changing the proportion of the joint loss to MSE:ADV = 100:1 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. Final results

The final results of our network could be seen here on some examples:



4. Figure: final results

We could see that the network performs with a good result, even though it is not perfect.

We believe that the network could do even better with a longer training, this is surely something we should consider in the future

Irodalom jegyzék

- [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. *HighResolution 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, Jeff 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]
- [10]