Deep Learning - Final Course Project

Ron Eliav, Yosef-Natan Berebi

Submitted as final project report for the DL course, BIU, Semester A, 2022

1 Introduction

We have the honor to present a smart context-encoder. A model that fill an image masked automatically. We did many experiments on many architectures until we got to our best results. We hope you'll enjoy to read our experiments and our results as well.

1.1 Related Works

The main paper we based on, is this paper. We tried many architecture, but finally we chose UNET architecture with skip connection. We also did style loss, and this paper helped us to understand the approach for implementing style loss. However, in this paper he take face photos, so it was nice to see the results on Monet photos.
In the PathcGAN (patch loss) we were helped in the paper, but we also checked if it's better to train the patch loss from the beginning or only after some epochs on the auto-encoder.
In the style loss part we helped by Neural Style Transfer from this paper, and also by this paper.We also check to train the auto encoder first for a few epochs and than the all model.

2 Solution

2.1 General approach

First of all, we tried many architectures for auto-encoder and for the discriminator. We tried to add more convolution layers, fully connected layers, and different smart fully connected etc, until we got a DNN that learn, Good results we got from UNET architecture.
We also tried to do transfer learning UNET or ResNet for the encoder alone. We built a model that use the same layers for the encoder and for the decoder. At the end, the best architecture was not pre-trained UNET, but with skip connections between the encoder and the decoder. Actually we tried to train

the auto-encoder on ImageNet but we hadn't enough GPU and time for that. All the way we thought about how much deep we want our model to be. Deeper net means bigger receptive field. At the beggining we thpught that we need a shallow net, because the context encoder regarding mainly on the close pixels. But then we ubderstood the a perfect completion must based on the whole photo, therefor we tried deep NN such as UNET and ResNet.

We built a discriminator model, as a classifier one. Also for the discriminator we tried ResNet and another architectures and pre-trained model, but nothing were better than our own architecture, same as UNET encoder architecture.

In order to support any mask shape, the model should return an image in the size of the input image. We conducted experiments when the model returns a full color image, and when the model return only the filled mask with white padding at the around of the mask. On this we calculates MSE. We got better results on the second experiment. However, we felt pretty bad about it. The mask size was small regarding all the prediction, so the model thought he was justify almost at all, when the generation didn't was so good.
Therefore, we decided to implement custom loss. Giving the masks, we calculated the loss regarding the mask place. i.e, we gave higher value for the mask pixels, that the model had to fill, than the pixels the model should return 'white' on them. Unfortunately, the results were not good, so we didn't use our custom loss for the final training.

We detected that if we train our model from the beginning with the auto-encoder and the discriminator, the model learns worse that we train the model some epochs only with the auto-encoder, and then we add the discriminator for fine-tuning for getting better results.

Regarding the patch-loss: First of all we trained our model without regular, means that the discriminator returned only one number. Then, we trained the discriminator in many sizes of patches, and so a big improvement, and the best one, on patches in size o 7*7. It's also makes sense, because so that the receptive field of each patch is focused on another part of the completion.

We trained our model on all of the three types of masks that had shown in the paper. We got the best results on 8 random blocks in size of 32*32.
The random region was the hardest one to implement.
We decided to do it manual by create many random masks on the PC and in the model we did data augmentation, by distorted the masks, for training the model.

Then, we started to think about more creative and smart models for models pipeline. First of all we decided to create a Monet classifier in order to detect if the input image is Monet's or a regular photo. We didn't know if it's necessary for the exercise, but we want to build something generic.

After that, we had doubts if it's better to divide our model to 3 parts, one model for each mask type, or to build one model for all of the masks. We checked both of the options. On the first option we built a classifier model that classify each image to its mask type so the program 'send' the input image to its correlate model. For the second option, we built a model on training set that contains masked images with all of the 3 masks types. We saw that the mask type classifier gets very high results, and the special models for each mask type are little better than the one big model. However, if we had more times, we were trained the classifiers better, maybe even do classifier for central block and for blocks, so if the mask isn't one of them, it's probably a region. The reason for this is that the region in a random shape, so it's difficult to the model to learn.

Style loss experiments: At the beginning, we tried to learn about how to implement the style loss in our model, so we stared with adding the style loss to the auto encoder itself, without the all model. we tried training only the auto-encoder with the style loss. We did not know how much weight to give to the style loss and the content loss, so we try different hyper parameters until we got the best results.
Another thing is the load weights. At the beginning we trained the model with random initialized and then we tried to load weights that we trained on the regular photos. We saw that the second option gave better results, so we use the weight from the regular photos as the initialization weights.

Also, one of the problem we had to deal with is the small amount of photo we are given. After we trained the model we thought about more ways to improve our results. We try two things:
A: We started to make augmentations on the Monet photos. We use a few different functions to increase our amount, like zoom, brightness, horizontal flip, and also we built a combine of this function to create more images. After we had hundreds and even thousand photos we start training the model again, we did not see a extraordinary of improvement, but we still got a good results so we used this.
B: another thing that we tried but unfortunately without much success was to generate style images from the photos - meaning to take each of the Monet photo, to warp it with a random style image, and create a new image with style. However, the results didn't improved.

2.2 Design

Our final pipeline goes like this: The user insert paths of images to the pipeline. Then, he decided which images he want to fill its mask. The pipeline detect the style of the image, i.e if it is a regular photo or a Monet photo. This model based on UNET architecture model, in binary cross-entropy loss and ADAM optimizer on 30 epochs, when we took 270 Monet photos and 270 random regular photos and trained the classifier.

Then, the pipeline classify the mask type (central block, blocks or region) according the mask the model gets. This model also trained on UNET encoder architecture. trained on 6500 mask gets high percent of accuracy.

After we have the style class and the mask type, we 'send' the image to the relevant context encoder so the generation will be the best.

2.3 Experimental results

Our goal was to get the best model, but with tendency to generic solution. the style Classifier got 86 percent accuracy on our test set. It's definitely enough, because the Monet models have good results on regular photos and vice versa, so it's not so bad to do a mistake in this classify.

The mask type classifier got 100 percent of accuracy, even on region that he didn't saw before.
All the context encoders models got the approximately the same loss and the same 'native' likelihood.

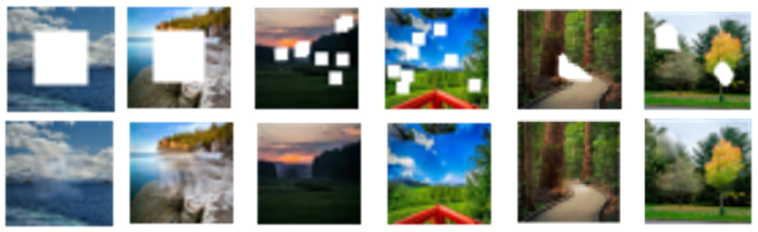

Figure 1: Results on our test set.
In the first line you can see the masked images. is the second line there are the input images with the generation injected

3 Discussion

Our first important insight is when you build a model, you should first want your model to learn something. At the beginning we tried to implement all the big and complicated architecture together. The model, off course, didn't worked, and we didn't understand the reason. So, we decided to make a progress step by step. To start with auto-encoder that learn, and to add more layers and complicated features during the training.
The second thing we learn is that generic code donates for building models.

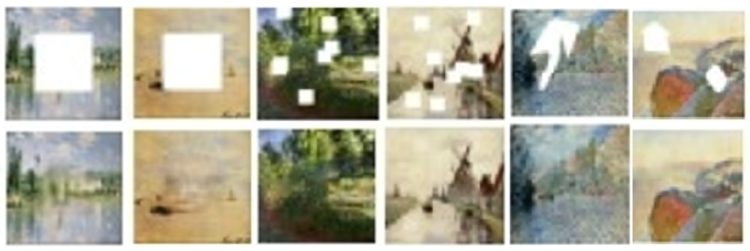

Figure 2: Results on our Monet test set.
In the first line you can see the masked images. is the second line there are the input images with the generation injected

Each model training is different in something from the other trainings. So for more efficient work, you want to work as generic as possible in order to not waste time each training.

Maybe the most important insight we got is divide a problem to many sub-problems can make the model predict better, and we'll explain. A deep neural network is a very smart and very stupid thing. DNN can learn everything, but sometimes we want it to learn something pretty easy, and it's not going well. In most of the times, the reason is that the problem isn't so easy for the model, despite the fact it's easy for humans. However, split the big problems to many sub-problems (e.g, divide images for 3 different masks types or divide to two different style) are making the model learning better. It's useful for almost every model someone can built. We always should ask ourselves if we can divide the problem to many mini-problems, and to build a model for each mini-model. The last but not least important insight is that not everything that looks correct and helpful for us, will definitely help training the model. It happened for us in the custom loss, when we were sure that it'll got the highest score. Maybe if we continue to try in this way, we were found the way to improve results with custom loss. However, we earned that you must try all the things you think that can help the model, but sometimes it's not the situation. Also, in the style loss - like we said earlier,we did not know how to implement the style loss function in the entire model ,so we started step by step - at the begin we only use it for the auto encoder, and then for the all model.

4 Code

Here is the link for the directory of the project. Inside the folder you'll find 'pipeline models' notebook in 'notebooks' folder, for fill an input masked image. You'll find there also notebooks of the training track. In addition, there is a data folder with all of the relevant data.