Final Project Report for the Course
Introduction to Deep Learning in Computer Vision

Anthony Nguyen - V00915696
University of Victoria
Victoria, BC, Canada
phucnm@uvic.ca

Abstract

Raindrops attached to a window or camera lens can reduce pictures quality. Rui et al. [1] addressed this problem by proposing a deep learning method to generate a raindrop-free image from another raindrop degraded image. The authors introduced visual attention to Generative Adversarial Network (GAN). Their underlying idea is the injection of attention maps into GAN can guide the generator and the discriminator to focus more on raindrop regions. This helps generator output better raindrop-free images. The authors collected 1149 pairs of images of size 480×720 *with various background scene and raindrop styles on their own. In this project, I implemented the training part from scratch which was not provided by the authors. I also fixed the source code of the discriminator network provided by the authors on their GitHub repo. In conclusion, the approach demonstrated an effective tool for raindrop removal and background restoration.*

1. Introduction

Raindrop removal is an interesting problem. We often have to capture images under rains and raindrops attached to camera lens or glass windows can reduce image quality severely. Rui et al. [1] published the paper titled "Attentive generative adversarial network for raindrop removal from a single image" to solve the problem. The authors proposed a method based on GAN which is one of the most breakthrough ideas in deep learning in the last decade. There are two key issues the authors have to solve: (i) the raindrop areas are not given, (ii) the background scene is lost for the most part. Although there were some attempts using both deep learning and traditional computer vision tools to solve this problem, they do not work really well on large and dense raindrop images and sometimes images are blurred.

To deal with the above problems efficiently, Rui et al. [1] apply visual attention concept which is expected to guide the Generator and Discriminator to focus on raindrops regions. They use an attentive recurrent neural network to generate attentive masks which indicate raindrop areas and relevant background details before training the Generator and Discriminator.

In this project, I attempt to reproduce the paper [1]. I also attempt to fix flaws in the paper or the official source code if there is any flaw. Since their implementation is only for reference, all layers are in one network and it makes readers hard to read and understand. I remodel the network architecture in a better code structure. From the improved network architecture, I implement the training part from the scratch in order to understand the concepts and reproduce the results.

2. Contributions of the original paper

A few techniques have been proposed by the authors to deal with the raindrop removal problem. Most importantly, they proved the injection of visual attention into the Generator and the Discriminator can help the network focus more on raindrop regions. They also applied skip connections to the Autoencoder to reduce blurred outputs. Along with the visual attention, they defined three custom loss functions which are Attentive Loss, Multiscale Loss and Perceptual Loss. In the following subsections, I will describe their contributions in details.

2.1. Attentive Recurrent Neural Network

Figure 1 shows the achitecture of attentive RNN. This is placed in front of the Autoencoder. It receives input image and generate a sequence of attentive masks. The final mask is then concatenated with the input image and feeded into the Autoencoder. In details, Each recurrent block consists of a few residual blocks, a convolutional LSTM unit, a few convolutional layers which output 2D attentive masks. The higher time step, the more detail attentive masks are produced. The intermediate attentive masks are contributed to the attentive loss function as well. In order to produce the

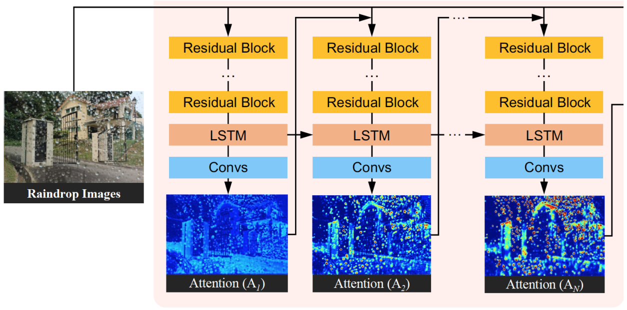

Figure 1. The architecture of Attentive RNN

attentive masks focus on raindrop regions, the author use a binary mask which is subtraction of the degraded image and the clean image. Thus the binary mask indicates raindrop areas.

2.2. Autoencoder

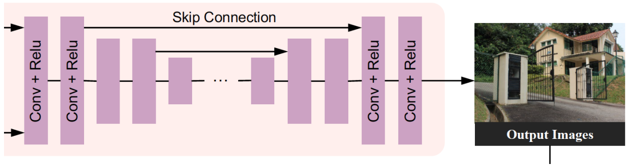

Figure 2. The architecture of Attentive RNN

Figure 2 shows the architecture of the Autoencoder. This subnetworks consists of 16 convolutional-relu blocks. They also introduced skip connections in order to reduce blurred outputs. The underlying idea is similar to ResNet where more information from early layers is contributed to the output layers to avoid gradient vanishing and the network can learn better. They introduced Multiscale Loss which is computed as the summation of mean square errors between output of the Autoencoder with the ground truth image in different scales. They also introduced Perceptual Loss which is mean square error of the features of the output image and the ground truth image extracted by pretrained model VGG16. For that reason, they called this network as Contextual Autoencoder.

2.3. Discriminator

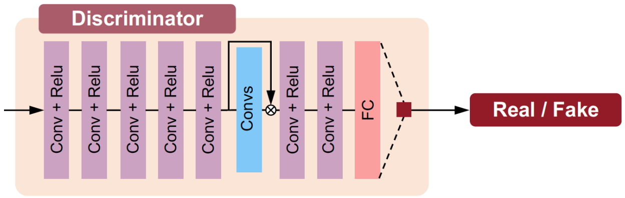

Figure 3. The architecture of the Discriminator

Figure 3 shows the architecture of the Discriminator. This is not much different than a normal GAN. They introduced some convolutional layers in the middle to generate back a 2D attentive mask. This attentive mask is used to compute Discriminative Loss function so that the Discriminator can learn from attentive masks too.

3. Contributions of this project

In this project, I attempted to reproduce the paper by implementing training part. I also found gaps between the paper and their official implementation. *E.g.*, the number of convolutional layers in the Discriminator and dilation parameters of some of convolutional layers in the Autoencoder were not mentioned in the paper. They do not affect the correctness of the paper but it will be difficult for people to reproduce the results. Moreover, I fixed two issues related to image sizes: (i) input size of the final fully-connected layer of their Discriminator is not corrected, (ii) some of images in data set have misaligned dimensions and I had to take care about it to avoid crashes.

3.1. Source code fix

In the source code of the discriminator on their GitHub, the output fully-connected layer did not work as expected. The code snippet below is what was the output layer of the discriminator

```
self.fc = nn.Sequential(
    nn.Linear(32 * 14 * 14, 1024),
    nn.Linear(1024, 1),
    nn.Sigmoid()
    )
```

Input size of the first linear layer is mismatched with actual input data. Thus, Pytorch crashed right away. I calculated the correct input size to $32 * 7 * 11$.

3.2. Dataset image sizes fix

Although most of the images in the training set are 480×720, some of them are 481 in height or 721 in width. I did not realize this until the training process crashed somewhere in the middle because of integer division operations. I had to normalized the input images as following:

```
aligned_w = int(img.shape[0]/4)*4
aligned_h = int(img.shape[1]/4)*4
img = img[0: aligned_w, 0: aligned_h]
```

Where img is the numpy array returned by $matplotlib.image$. I aligned images' dimensions to multiples of 4.

3.3. Training implementation from scratch

Since the authors did not publish their training code, I implemented the training loop from scratch to understand their concepts better. Network architectures of the generator, the discriminator and the use of pretrained model VGG16 were open sourced by the authors at `https://github.com/rui1996/DeRaindrop` though. There was another work done in Tensorflow at `https://github.com/MaybeShewill-CV/attentive-gan-derainnet`. However, I implemented the network in PyTorch and the approach of the two deep learning frameworks is different for the most part. I also wrote checkpoint resuming capability, Tensorboard logging and implemented custom loss functions. In order to control the training process better, I also implemented command line arguments part and added some hyperparameters arguments such as number of time steps N in attentive RNN, θ in attentive loss, γ in discriminative loss. I wrote some image processing helper methods using $torchvision$ and $numpy$ such as the image resize method for the multiscale loss function or the method to compute binary mask from two images for the attentive loss function. The training part can run on either CPU or CUDA.

Regarding training details, I trained the model with batch size 1. That means I only trained one input image at a time. The model itself took over 11GB of memory on GPU. Thus, with single 12GB GPU K80 on Google Cloud Compute Platform, there is no memory left to make use of VGG16 to compute perceptual loss. In order to use VGG16 and have better training speed, I had to simplify the network architecture by removing some convolutional layers in the generator and the discriminator.

I used **Adam** optimizer with learning rate $1e-4$ for both Generator and Discriminator.

Below is core steps of the training loop:

1. Feed the Generator with the input image to get the output image
2. Forward and train the Discriminator with the above output and the corresponding clean image
3. Feed the Generator with the input image to get the output image
4. Train the Generator on the above output image
5. Repeat from Step 1.

In the original GAN, people sample random noise to train the network. In this paper, the authors train with pairs of images because attentive masks are generated and concatenated to the input image to feed as input of the autoencoder. Like other GANs, the network is easy to train but it is really unstable and difficult to optimize. For the network architecture provided by the authors, it take nearly 13s to train a pair of images. The network might need to train over a few days to achieve good results at around 100 000 epochs.

Figure 4 shows loss values of the Generator and Discriminator after 1 epoch.

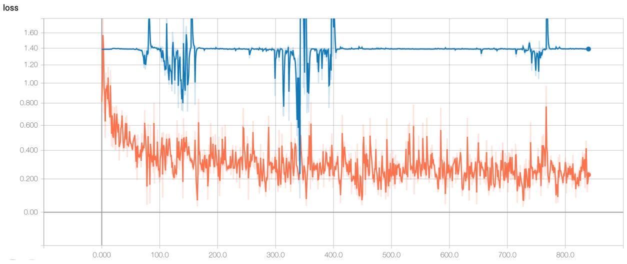

Figure 4. The architecture of the Discriminator

4. Discussions

The injection of visual attention into GAN has obtained excellent results in raindrop removal and background detatils restoration. If I have more time to work on the project, I would train the network with different hyperparameters to search for best network configurations. In addition, I will apply data augmentation such as mirroring and random cropping images. The purpose of this technique is to ' generate more datasets thus we can improve the generalization of the network and avoid overfitting.

In future, there are a few research directions which extend this method. Firstly, there should be a more lightweight method to apply to videos. Secondly, the idea of visual attention in GAN can be applied to other problems such as dirt and haze removal, object removal, image restoration.

References

[1] R. Qian, R. T. Tan, W. Yang, J. Su, and J. Liu. Attentive generative adversarial network for raindrop removal from a single image. *CoRR*, abs/1711.10098, 2017.