Image deblurring using Conditional Adversarial Network (or DeblurGAN)

Amogh Batwal abatwal@iu.edu
Dhruva Bhavsar dbhavsar@iu.edu
Jay Madhu jaymadhu@iu.edu
Ruta Utture rutture@iu.edu

Abstract

Motion blur in videos and images remains a fundamental problem in computer vision which leads to significant degradation of the image, impacting its quality. Many computer vision tasks like image or video analysis, object detection, and recognition rely on local features and descriptors and it is well known that motion blur decreases the performance of the traditional detectors and descriptors. Hence, in this project, we attempt to re-implement the DeblurGAN; an end-to-end learned method for motion deblurring, based on conditional GAN and content loss. We examine the quality of de-blurred images by evaluating it on a real-world problem – Object detection. Code is uploaded at GitHub. `https://github.com/The-GAN-g/DeblurGAN`

1. Introduction

Blur in images and videos arises not only from multiple object motions but also from camera shake, scene depth variation, and due to low-light conditions when the exposure time can often be in the region of several seconds. Traditional deblurring algorithms leverage the physics of the image formation model and use hand-crafted priors. In recent years, significant progress has been made in solving vision problems using deep learning. One such technique is Generative Adversarial Network or GANs. A powerful set of Convolutional Neural Networks, GANs have capabilities to preserve texture details in images and generate solutions that are close to the real ones at the same time, look perceptually convincing. Generative Adversarial Networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other (thus the"adversarial") to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation, and voice generation. Since its introduction in 2014 by Ian Goodfellow, Generative Adversarial Networks (GAN) [5] has been an extremely popular Machine Learning algorithm to solve varied computer vision problems

For this project we re-implement the DeblurGAN [11] which is based on the recent work on image-to-image translation (pix2pix) [6]. Our main goal is to try implement the paper on our own in the best-simplified way possible. Deblurring is treated as a special case of image-to-image translation. The model approach is based on conditional generative adversarial networks and multi-component loss. However, for DeblurGAN, instead of using multi-component loss, we used Wasserstein GAN with the gradient penalty and perceptual loss as suggested in the paper, which promotes solutions that are perceptually difficult to differentiate from real sharp images and allows finer texture information to be restored than when using conventional MSE or MAE as a target for optimisation. According to the paper presenting DeblurGAN, the model obtains the state-of-the-art results, while being 5x times faster than the fastest competitor.

Figure 1. (Example) Top left - Ground truth. Bottom left - Blurred. Right - Deblurred.

2. Background and related work

Over the last few years, with the growth of deep learning approaches based on Convolutional Neural Networks (CNNs) were also developed to solve the problem of image deblurring. However, the breakthrough came when Ian Goodfellow introduced Generative Adversarial Network (GAN), which can preserve textures, patterns, style, and the context in an image, allowing new images to be generated. The idea behind the GAN is to make the two components of it adversarial of each other so that over the

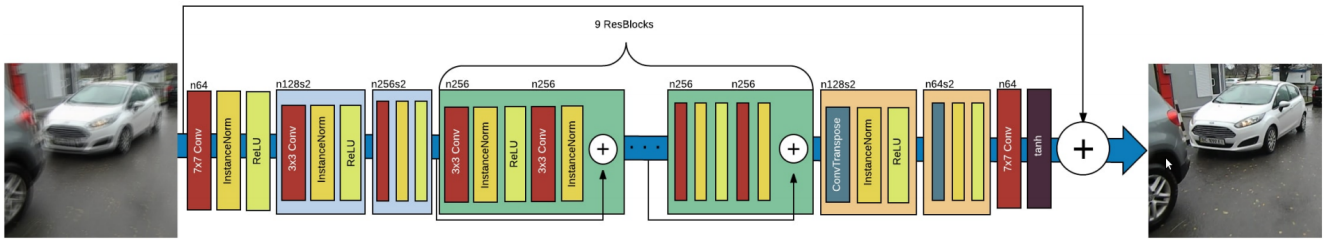

Figure 2. DeblurGAN Generator architecture [11]

timing generator produces better results and discriminator can better distinguish between the image generated by the generator. The game of generator and discriminator can be formulated as a minimax objective function:

$$\min_G \max_D \mathbb{E}_{x\sim\mathbb{P}_r}[log(D(x))] + \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[log(1 - D(\tilde{x}))]$$

However, it is rather challenging to train a GAN model, as people are facing issues like training instability or failure to converge due to vanishing gradients. Hence, for this model, we use WGAN (Wasserstein GAN) [12]. WGAN is intended to improve GAN's training by adopting a smooth metric for measuring the distance between two probability distributions called Wasserstein-1 distance. This is crucial for image deblurring as it allows the use of novel lightweight neural network architectures in contrast to standard Deep ResNet architectures (like ResNet128), previously used for image deblurring. Although GAN models are capable of generating new random plausible examples for a given dataset, there is no way to control the types of images generated other than trying to figure out the complex relationship between the latent space input to the generator and the generated images.

The conditional generative adversarial network, or cGAN for short, is a type of GAN that involves the conditioning of the generation of images by a generator model. Conditional adversarial networks are applied to various images-to-image translation problems like super-resolution, style transfer, photogeneration, etc. An architecture for cGAN as a general solution to image-to-image translation problems called pix2pix, has been defined Isola et al. [7] wherein, the network not only learns the mapping from an input image to output image but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations. The architecture of the DeblurGAN is highly inspired by pix2pix, especially the discriminator.

3. Method

The goal is to get a sharp image as an output for every blurred image as an input. Trained Generator G performs deblurring on the images. Besides, during the training phase, we introduce critic network D and train both networks in an adversarial manner.

3.1. Loss Function

3.1.1 Generator Loss

The generator loss is a combination of content loss and adversarial loss. We have taken the Perceptual Loss as the content loss. The perceptual loss is the difference between the VGG-19 [13] conv3.3 feature maps of the sharp and restored images. The Wasserstein (WGAN) [12] loss is taken as the adversarial loss. It is the negative of the mean critic score on the generated image. The combined loss is the summation of WGAN loss from the critic and the perceptual loss. The λ value is set to 100.

$$L = L_{\text{GAN}} + \lambda L_{\text{x}}$$

The perceptual loss focuses on restoring general content, whereas WGAN loss focuses on texture details. Initially, we experimented with a simple MSE loss to check if the network works, but the results were blurry and so we went ahead and implemented the same losses as the original paper.

3.1.2 Discriminator loss

To train the discriminator we use Wasserstein GAN with the gradient penalty. This helps with restoring the finer details of images. WGAN-GP uses gradient penalty instead of the weight clipping to enforce the Lipschitz constraint. The λ value for gradient penalty is set to 10. The discriminator is trained 5 times for each training iteration of a generator. The discriminator or critic loss is the difference between means of critic score on real image and generated image.

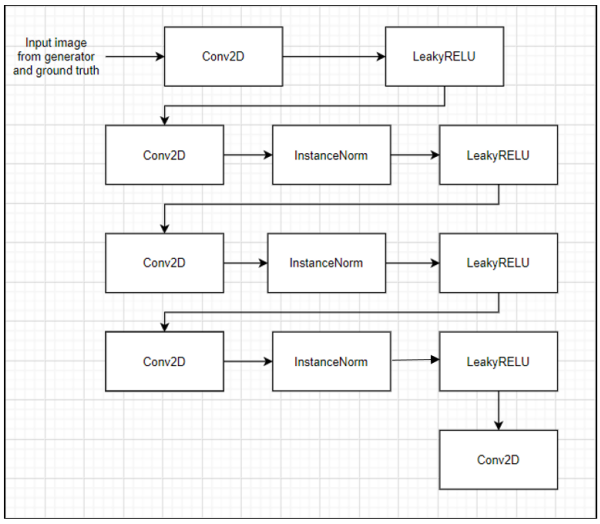

Figure 3. DeblurGAN discriminator architecture

3.2. Network architecture

3.2.1 Generator

The generator has the duty of producing a sharp image from the blurred ones to fool the discriminator in classifying the generated image as the actual (sharp) one. We have implemented the same generator architecture as presented in the paper [11]. However, given the flexibility of using lightweight generator architecture, we implemented a ResNet6 instead of ResNet9 for comparison. However, based on our initial implementation of the architecture, we didn't go with the ResNet6, hence, we decided to go with the architecture presented in the paper. Their generator is highly inspired from pix2pix by Johnson et al. [7] for style transfer. The architecture of the generator is as shown in Figure 2. The architecture consists of two convolution blocks with kernel size 3, stride 2, and padding 1, followed by 9 blocks of Resnet [9] and then two transposed convolution blocks. The ResBlocks consists of convolution layer, batch normalization, and ReLU activation. Dropout is used after the first convolution layer. Apart from the skip connection present in ResBlock, a global skip connection is used. The major change that we made from the original paper is we are using batch normalization instead of instance normalization because we are taking a batch size of 4 for faster training as opposed to 1 taken by the original paper.

3.2.2 Discriminator

The objective is to determine if an input image is artificially created. The discriminator's architecture is also similar to the original paper [11]. Wasserstein GAN [12] with a gradient penalty, also known as WGAN-GP, is used as the critic. The architecture of the discriminator is as shown in Figure 3. A critic approximates the distance between the samples also called Wasserstein distance. The gradient penalty is used as an alternative to a way to enforce the Lipschitz constraint used by WGAN. This method was proposed by Gulrajani et al. [5]. The discriminator follows the architecture of PatchGAN, which was originally used in the pix2pix model by Philip [6]. There are two inputs to the discriminator viz. the deblurred image from the generator and the actual sharp version of that image. All convolution layers except the last are followed by Batch Normalization layer, InstanceNorm, and LeakyReLU [4] (with $\alpha = 0.2$). The major advantage of using WGAN is that it helps to converge better, and so we can use a generator architecture much lighter than ResNet152.

3.3. Training Details

The project implementation was completely done using PyTorch [3] deep learning framework. We used Google Cloud Platform [1] for training our model by creating a VM instance with a single NVIDIA Tesla T4 GPU. The data was loaded for training using a custom data-loader. A special train-test splitting function was implemented which separates images from their original directories and splitting them into the train and test folders, each containing blurred and sharp images in a separate loader. Our data-loader con-

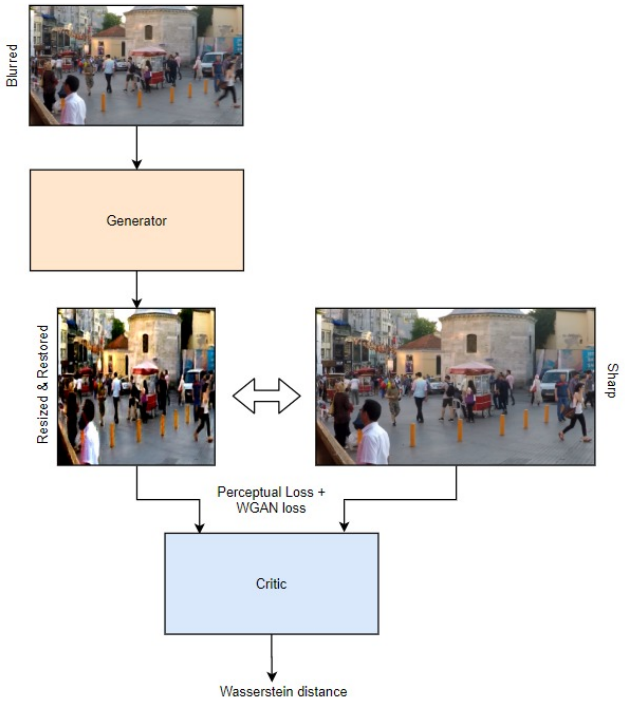

Figure 4. Training [11]. The restored image from generator, and the actual sharp image is fed to the critc network

siders this formation of the image-directories. The dataloader placed images as a pair of blur and corresponding sharp images. All the images were resized to 256x256 sized images. The batch size for data was kept to 4, contrary to 1 used by the original paper for speedy results. The dataset for training consisted of 2103 images. So, for each epoch, the training loop would run for 525 times. We use Adam solver [10] for optimizing generator and discriminator. The value of β_1 is set to 0.5 and β_2 is set to 0.9999. We perform five gradient descent steps for the discriminator for each step of the generator as mentioned in the paper. Initially, we had kept the descent steps equal for both discriminator and generator, which lead to faster training, but the output we got was very pixelated, so we changed it to the ratio of 5:1. We then planned to run the model for 300 epochs. The learning was initially set to 10^{-4} for the first 150 epochs. This rate would then slowly decay to zero over the next 150 epochs. Based on the system configurations of the VM, each epoch took 1530 seconds to run, which would bring the training time for the entire network to a total of 5 days for 300 epochs.

4. Experiments and Results

It endures approximately 5 days to train a DeblurGAN. By the time we were done with our final implementations, we had been 3 days away from the submission, hence we were unable to present the final output. Nonetheless, we saved the trained model after every 20 epochs, and hence we were able to use it to examine the model performance, and show the network is improving epoch by epoch in generating a better result.

4.1. GoPro dataset

We used the GoPro Dataset to train our model. It consists of 3214 sets of images. The images are taken from 240fps videos, shot using GoPro4 Hero Black. The dataset can be found here: `https://github.com/SeungjunNah/DeepDeblur_release`. Out of this, 2103 images were used for training the model on a single GPU. We are using the following metrics to evaluate our model performance:

1. Structural Similarity Index (SSIM): To measure how similar two images are with each other.

2. Peak Signal-to-Noise Ratio (PSNR): To measure the ratio between the maximum possible values of a signal of the ground truth image compared with the deblurred image.

The mean of PSNR and SSIM values over all the images are calculated after every 20 epochs. The results obtained on the trained model are tabulated in Table 1. We receive good results with the authors of DeblurGAN in terms of PSNR.

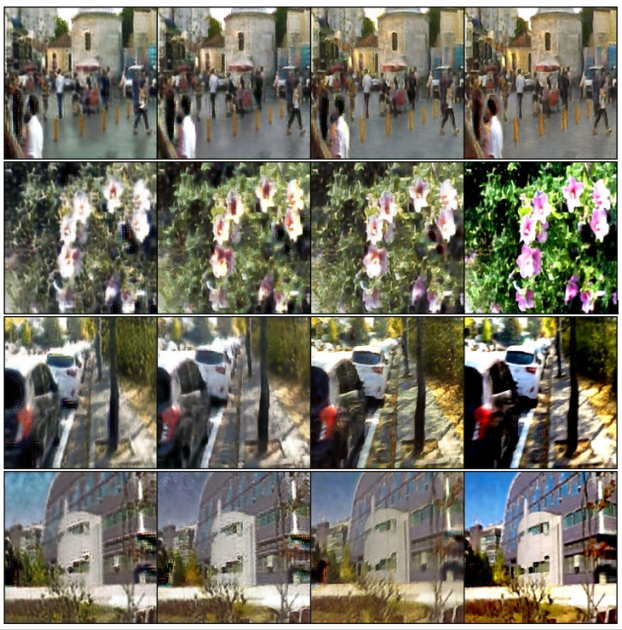

Figure 5. Results of our DeblurGAN after 100 epochs

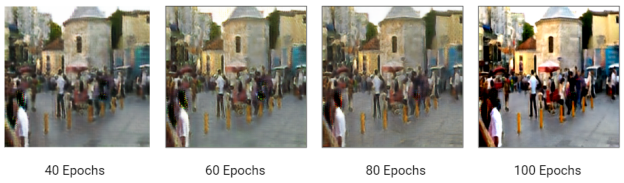

Figure 6. Results of our DeblurGAN performance on blurred images after 40, 60, 80 and 100 epochs

Epoch	PSNR	SSIM
20	28.46	0.599
40	28.52	0.592
60	28.54	0.576
80	28.47	0.586
100	27.88	0.526

Table 1. Average values of PSNR and SSIM over all the images after every 20 epochs.

4.2. YOLO Object Detection

Object Detection is one of the most well-studied problems in computer vision with applications from autonomous driving to defense in various domains. To check the performance of our model results, we run the pretrained YOLO object detection [8] using openCV [2]. We provide the deblurred output from our model and the actual sharp image to the YOLO network for examining the improvement in object detection. The results of our tests can be found in Figure 4.

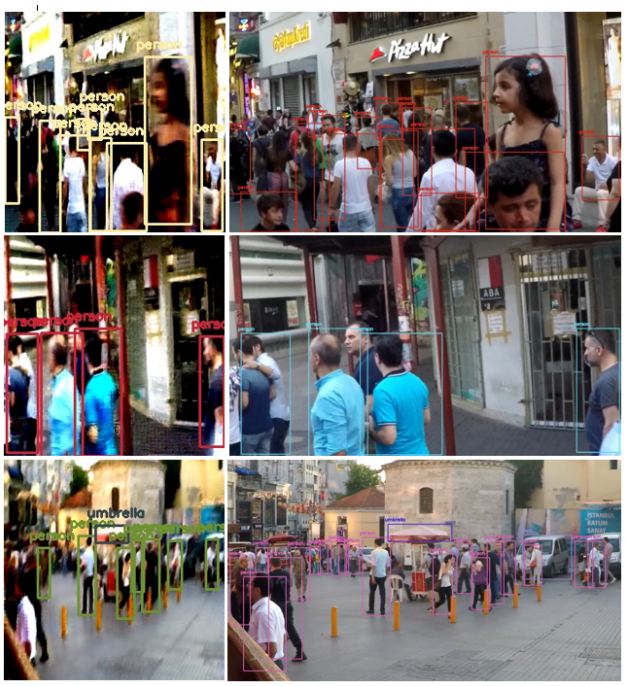

Figure 7. Comparison of YOLO object detection on generated and original image

5. Conclusion

While we successfully re-implemented DeblurGAN, we were not able to present the final output given the time it takes to run the network. By the time we completed this report, the last checkpoint saved was at 100^{th} epoch. Even though we were not able to show the final output, we've presented results generated by DeblurGAN after every 20 epochs, as show in Figure 5. The results suggests that the network is significantly improving after every epoch producing results better the previous. This gives us confidence that our network will generate efficient results after 300 epochs. We do plan to continue running it to see the final results.

References

[1] Google cloud platform. https://cloud.google.com/.

[2] Opencv. https://opencv.org/.

[3] Pytorch. http://pytorch.org/.

[4] T. a. B.Xu, N.Wang. Empirical evaluation of rectified activations in convolutional network. 2015. https://arxiv.org/pdf/1505.00853.pdf.

[5] M. A. V. D. I. Gulrajani, F. Ahmed and A. Courville. Improved training of wasserstein gans., 2017. https://arxiv.org/abs/1704.00028.

[6] P. Isola. Image-to-image translation with conditional adversarial nets, 2017. https://phillipi.github.io/pix2pix/.

[7] A. A. J. Johnson and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution., 2016. https://arxiv.org/pdf/1603.08155.pdf.

[8] R. G. J. Redmon, S. Divvala and A. Farhadi. You only look once: Unified, real-time object detection., 2015. https://arxiv.org/pdf/1506.02640.pdf.

[9] S. R. K. He, X. Zhang and J. Sun. Deep residual learning for image recognition., 2015. https://arxiv.org/pdf/1512.03385.pdf.

[10] D. P. Kingma and J. B. CoRR. Adam: A method for stochastic optimization., 2014. https://arxiv.org/pdf/1412.6980.pdf.

[11] O. Kupyn. Deblurgan: Blind motion deblurring using conditional adversarial networks, 2018. https://arxiv.org/pdf/1711.07064.pdf.

[12] S. C. M. Arjovsky and L. Bottou. Wasserstein gan, 2017. https://arxiv.org/pdf/1701.07875.pdf.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. https://arxiv.org/abs/1409.1556.