Deblur GAN: Blind Motion Deblurring Using Conditional Adversarial Networks

E6040.2018Spring.DGAN.report
Yanjun Lin yl3829, Yao Chou yc3332, Yilin Lyu yl3832
Columbia University

Abstract

This project is a implementation and replication of the ideas by paper Deblur GAN: Blind Motion Deblurring Using Conditional Adversarial Networks. [2] The learning is based on a conditional GAN and the content loss. We implemented Deblur GAN in a TensorFlow approach. One of the technical challenges we met is sudden switch of colors of the deblurred images when training the GAN in Tensorflow, and was solved by adjusting different weights to components of the loss function. To evaluate our own model, we used the PSNR approach, which was mentioned by the author in the original paper. We also evaluated our model with blurred images created by different kernels to show how our model perform on different kernels.Our model also performs well on clinical image deblurring, showing great generalizing ability.

1. Introduction

The impressive achievements of Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision in recent years. GANs are known for an ability to preserve high texture details in images and create solutions that are close to real image manifold and look perceptually more convincing. Image deblurring, in the meanwhile, could be treated as a special case of such image-to-image translation. The idea of Deblur GANs by Kupyn and Budzan 2017 [2] are innovative and creative. It is an approach based on Conditional Generative Adversarial Networks and a multi-component loss function. Using Wasserstein GAN [3] with Gradient penalty with perceptron loss, the Deblur GANs could get solutions which are perceptually hard to distinguish from real sharp images and allows to restore finer texture details in contrast to using traditional MSE or MAE as an optimization target.

Our objectives in this project were, first, to successfully implement the proposals of the original paper in TensorFlow and Keras, second, to replicate the improved training results on blurred images reported by the authors, third, to compare and test the sensitivity of Deblur GANs using different pre-trained models, and finally, to test the generalization and improvement of the optimal trained model. There were a number of challenges we encountered while implementing these novel proposals. The first was conceptual: making sure we properly understood the model architecture and the components of the loss function. The second challenge involved when we actually implementing the Deblur GANs in Tensorflow. Implementing the model with complicated loss function correctly required careful attention to detail. During the process, we met challenges including the limitation of our computation ability and gradient vanishing when training the model. By taking carefully look on the code provided by the authors, we implemented the model step by step and made sure it match the ideas proposed by the original paper, especially to add gradient penalty to the loss function. To efficiently utilize our computation ability, we optimized our methods to organize datasets, to fetch and to use the data, making sure our training process went well.

2. Original Paper
2.1 Methodology of the Original Paper

The goal is to recover sharp image given only a blurred image as an input, so no information about the blur kernel is provided. To do that, the original paper proposed and trained a CNN G as the Generator. For each blurred image it estimates corresponding sharp image. In addition, during the training phase, the authors introduced critic function D as the discriminator and train both networks in an adversarial manner. In the original paper, two kinds of loss is applied to the generator. One is WGAN-GP loss and the second is Perceptual loss.
The model the original paper uses is a Conditional Wasserstein GAN with Gradient Penalty and Perceptual loss based on pretrained VGG-19 activations. Such architecture also gives good results on other image-to-image translation problems (super resolution, colorization, inpainting, dehazing etc.)

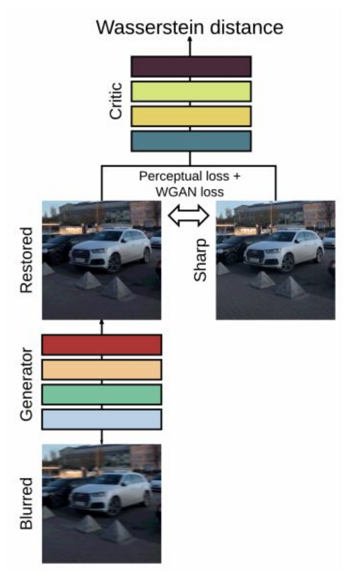

Fig. 1. Conditional GAN for motion deblurring

2.2 Key Results of the Original Paper

The key result of the original paper is that the Deblur GANs performed better results than previous method on GoPro test dataset of 1111 images [4]. Two main metrics are PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index), the Deblur GANs achieved 28.72 and 0.958 on them respectively, and the time of deblurring an image is 0.85s.

DeblurGAN shows superior results in terms of structured self-similarity, is close to state-of-the-art in peak signal-to-noise-ratio and provides better looking results by visual inspection. It can handle blur caused by camera shake and object movement, does not suffer from usual artifacts in kernel estimation methods and at the same time has more than 6x fewer parameters comparing to Multi-scale CNN , which heavily speeds up the inference.

3. Methodology

Since the model architecture and loss function proposed in the original paper are somewhat complicated and hard to understand, so we will give step-by-step detailed descriptions in the following sections of implementation. Here we only proposed what our works are different from the original paper and the expected result.
We first implemented the novel proposals of Deblur GAN in both Tensorflow and trained on a larger dataset including more dblur-sharp pairs which was expected to improve the performance of the model.
To balance the components of the complex loss function, we searched and set different parameters and it actually made a great difference.
The original paper proposed the training details that firstly trained on discriminator and then the generator. We changed the training recipe firstly trained a naive generator through several epochs which produces more confusing samples to the discriminator. The novel idea is expected to save much training time and gives similar deblurring performance.
To test the sensitivity of deblurring results, we slightly changed the model especially the pre-trained VGG19 model which is used for calculating the perceptual loss. Quantitatively and visually evaluations were made on internal data and external data, especially images with different blur kernel.

3.1 Objectives and Technical Challenges

There are basically two technical challenges we met throughout our projects.
Firstly, the limitation of computational capacity. We slightly change the training sequence as mentioned before and it saves much time by visually evaluation. Meanwhile, we optimize our method for data storing, fetching and using so that it saved much time for loading data to train our model.
Secondly, the colour of the deblurred images often switch suddenly and there are sometimes weird spots in the images. We resolved this challenge by searching and setting different parameters for the components of the loss function and giving the perceptual loss greater weight. This made the perceptual loss and Wasserstein GAN [2] with gradient penalty loss, which refers to WGAN-GP loss in roughly same scale. Meanwhile, we changed the output of the generator, from clip to average.

4. Implementation

In this section, we give descriptions to the Deblur GAN in details step-by-step. Firstly introduce the model architecture of both the generator and discriminator, followed by loss function and training details..we trust that these proposals accurately reflect the author's' intentions.

4.1 Deep Learning Network

4.1.1 Generator

The first two blocks are convolution block with stride ½. Next it is followed by nine residual blocks [6] (ResBlocks) and two transposed convolution blocks. Each ResBlock consists of a convolution layer, instance normalization layer, and ReLU activation. Dropout regularization with a probability of 0.5 is added after the first convolution layer in each ResBlock. The generator also introduces a global skip connection between the input and the last output of the transposed convolution block.

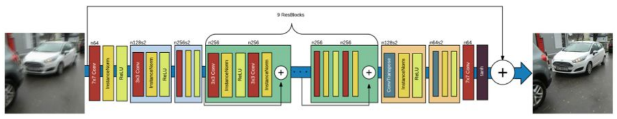

Fig.2 Architecture of the DeblurGAN generator network

4.1.2 Discriminator

Discriminator contains 4 convolution blocks. All convolutional layers contains batch normalization except the first convolutional layer and all are followed by leky_ReLU activation. Discriminator is WGAN-GP.

4.2 Loss

For GANs, the goal of generator and discriminator is to minmax the objective:

$$\min_G \max_D E_{x\sim P_r}[\log(D(x))] + E_{\tilde{x}\sim P_g}[\log(1 - D(\tilde{x}))]$$

where P_r is the data distribution and P_g is the model distribution implicitly defined by $\tilde{x} = G(z),\ z \sim P(z)$ (the input z to the generator is sampled from the blurry images.

However, the divergences defined above which GANs typically minimize are potentially not continuous with respect to the generator's parameters, leading to training difficulty. To solve this problem, Wasserstein GANs is proposed. In this method, the Wassertain-I distance $W(q,p)$ is used instead. Here $W(q,p)$ is continuous everywhere and differentiable almost everywhere. The goal for generator and discriminator becomes:

$$\min_G \max_{D\in\square} E_{x\sim P_r}[D(x)] + E_{\tilde{x}\sim P_g}[D(\tilde{x})]$$

where $\square$ is the set of 1-Lipschitz function. To enforce the Lipschitz constraint on the critic, one needs to clip the weights of the critic to lie within a compact space [-c,c]. Clipping the weights in critic would lead to optimization difficulties, and even when optimization succeeds the resulting critic can have a pathological value surface. Therefore, gradient penalty is introduced, an alternative way to enforce the Lipschitz constraint. A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, so we consider directly constraining the gradient norm of critic;'s output with respect to its input. The new object is:

$$E_{x\sim P_r}[D(x)] - E_{\tilde{x}\sim P_g}[D(\tilde{x})] + \lambda E_{\hat{x}\sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$

where $P_{\hat{x}}$ is defined as sampling uniformly along straight lines between pairs of points sampled from the data distribution P_r and the generator distribution P_g. From experiments in [5], we use $\lambda = 10$, which is believed to work well across a variety of architectures and datasets. The goal of discriminator is to minimize the whole objective. As for the generator, it tries to minimize $-E_{x\sim P_r}[D(x)]$.

Beside the GANs loss, the generator also need to minimize perceptual loss, which is a simple L2-loss, but based on the difference of the generated and target image CNN feature maps.

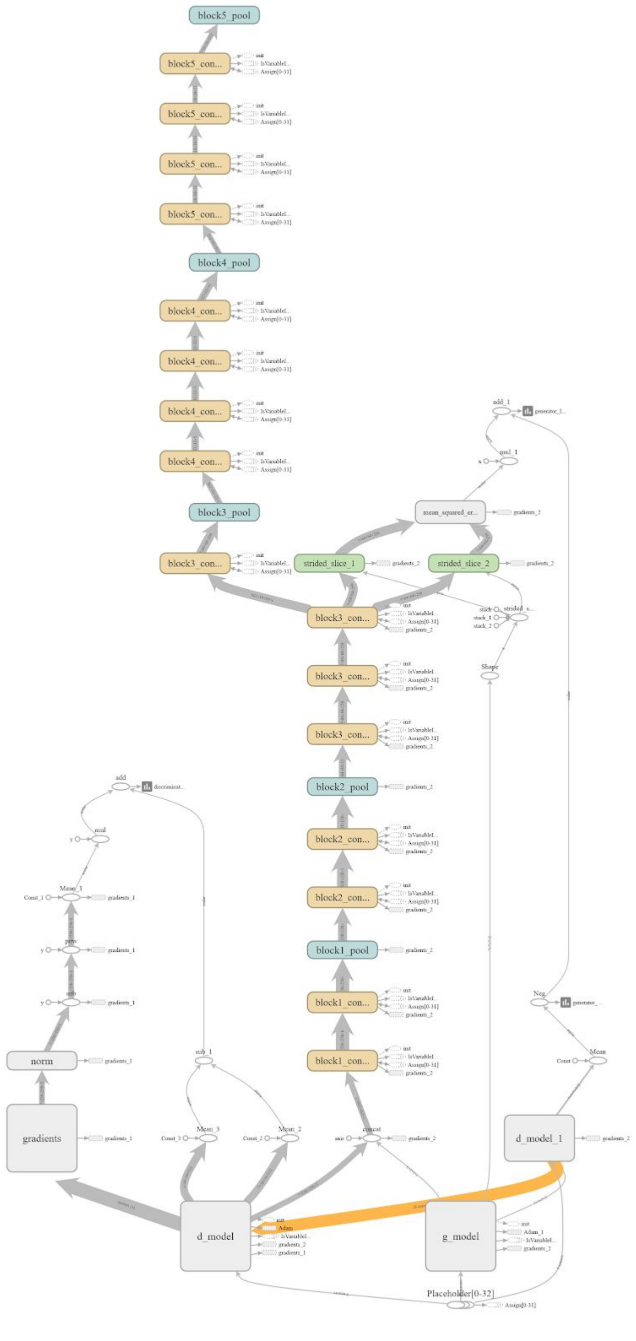

Fig. 3 TensorBoard Computational Graph

4.3 Training Details

Our Deblur GAN model was trained on a random crops of size 256x256 from GoPro dataset images. The GoPro dataset consists of 2103 pairs of blurred and sharp images in 720p quality, taken from various scenes.
For the parameters, different from the original paper, we adjusted and set greater weights for perceptual loss to make the losses in relatively similar scale, which showed empirically better results on validation.
As the models are fully convolutional and are trained on image patches, they can be applied to images of arbitrary size. For optimization we firstly performed 5 gradient descent steps on D, then one step on G, using Adam optimizer with beta1 equals to 0.5 as the solvers. We also trained our model by firstly training the G and then performing previous recipe. After comparison, the new recipe saves much training time to get similar results by visually evaluation. The learning rate was set to 10^{-4} for both generator and critic. We set the batch sizes as 2 and the epoch numbers as 20 due to computation limitation. The training phase took roughly 11 hours for training one Deblur GAN network.

4.4. Software Design

We implemented this method using Tensorflow and Keras, and we monitor all the losses and their corresponding component using tensorboard. Fig.3 is the computational graph from tensorboard.

5. Results

In this section, we will discuss the results of our experiments.

5.1 Project Results

We firstly trained naive models to find the weighting parameters to maintain the losses in similar scale and then use the parameter to fully train GAN models for deblurring.
We fully trained two Debur GAN models in our project. The first model is trained using VGG19 as pre-trained model, consistent with the original paper, to calculate the perceptual loss and take weighting parameter as 100000 inside the loss function.
The other model is trained using VGG16 as pre-trained model and take the same weighting parameter inside the loss function.
Evaluation using the metric PSNR (Peak Signal To Noise Ratio) are donc to search for the parameters and to test the ability of our model. The best model we gained hits a PSNR of 26.77 in the GoPro test dataset of 1111 images.
Visual evaluations of customized images deblurring also have been done using the optimal model to test the generalization of it.

Another major component is the analyses of performance on images with different blur kernels. We created two different kinds of blurred images using Gaussian blur kernel and Median filter kernel. We also changed kersels sizes to 5, 9, 13. We found our model had different performance on both kernels types and kernel sizes, which will be discussed in 5.3.

5.2 Comparison of Results

5.2.1 Result Comparison

Table 1: Mean PSNR measure on GoPro test dataset of 1111 images.

	Our Implementation		**Original Paper**
Pre-trained Model	VGG19	VGG16	VGG19
PSNR	**26.77**	25.06	**28.72**
Process Time	0.97s		**0.85s**

Our implementation basically replicates the performance of the original Deblur GAN. Notice that we only trained 20 epoches for our optimal mode. Considering the authors trained 160 epoches for six days, our model is expected to get similar performance as the training process going. This also indicate our rescale method and novel training procedure has made an improvement.

5.2.2 Visual Evaluation

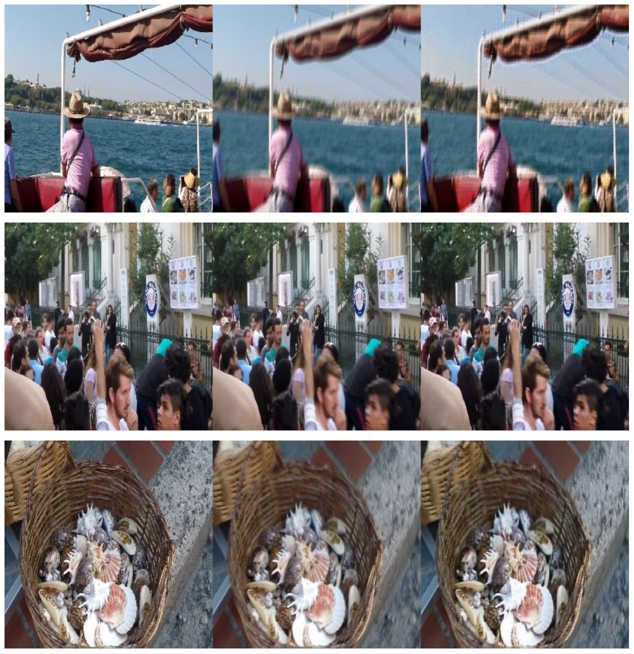
Fig.4 Result of the testing data. From left to right: Sharp, Blurred, Deblurred.

The above pictures showed the performances of our implementation on testing data. It seems that our Deblur GAN has already have the ability to detect blur on irregular boundaries, for example the human face at the bottom of the second picture and the edge of the shells in the third picture.
The following pictures showed the performance of the model using data from external data, clinical images and our own blurred images.

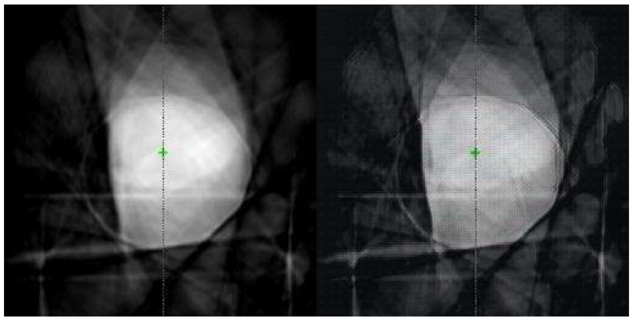
Fig.5 Deblurring a CT image of an egg.

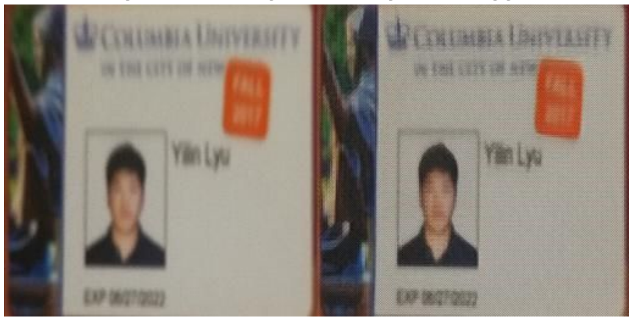
Fig.6 Deblurring a CUID card

5.3 Model Analysis on Different Kernels

During the training process, we used sharp and blurred images from GOPRO dataset. The blurred images from GOPRO datasets was generated by simulates realistic and complex blur kernels. The algorithms was detailed described in the original paper.

Algorithm 1 Motion blur kernel generation.
Parameters:
M = 2000 – number of iterations,
Lmax = 60 – max length of the movement,
ps = 0.001 – probability of impulsive shake,
I – inertia term, uniform from (0,0.7),
pb – probability of big shake, uniform from (0,0.2),
pg – probability of gaussian shake, uniform from (0,0.7),
φ – initial angle, uniform from (0,2π),
x – trajectory vector.

```
1: **procedure** BLUR(Img, M, Lmax, ps)
2: v0 ← cos(φ) + sin(φ) ∗ i
3: v ← vo ∗ Lmax/(M − 1)
4: x = zeros(M, 1)
5: for t = 1 to M − 1 do
6:      if randn < pb ∗ ps then
7:          nextDir ← 2 · v · e i∗(π+(randn−0.5)))
8:      else:
9:          nextDir ← 0
10:     dv ← nextDir + ps ∗ (pg ∗ (randn + i ∗ randn) ∗ I ∗ x[t] ∗ (Lmax/(M − 1))
11:     v ← v + dv
12:     v ← (v/abs(v)) ∗ Lmax/(M − 1)
13:     x[t + 1] ← x[t] + v
14: Kernel ← sub pixel interpolation(x)
15: Blurred image ← conv(Kernel, Img)
16: return Blurred image
```

In this part, we analyzed our model with two different kinds of kernel with different sizes.

Gaussian blur

The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in one dimension is

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

in two dimensions, it is the product of two such Gaussians, one in each dimension:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution.Values from this distribution are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight (having the highest Gaussian value) and neighboring pixels receive smaller weights as their distance to the original pixel increases.

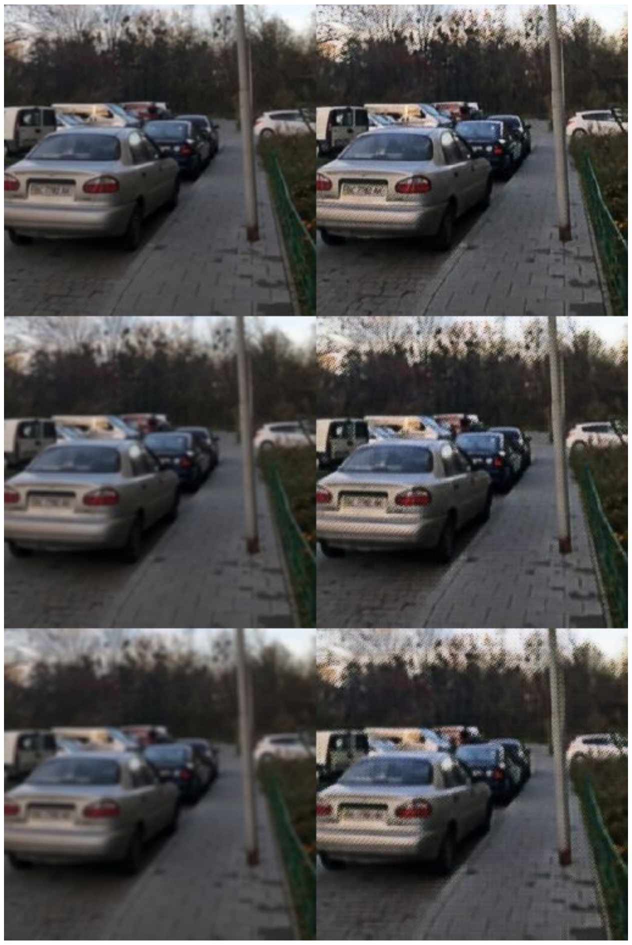

Figure.7 Top: Gaussian blurred image(ksize=5), before and after deblur; Middle: Gaussian blurred image(ksize=9), before and after deblur; Bottom: Gaussian blurred image(ksize=13), before and after deblur

Median Filter
The median filter is a nonlinear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise (but see discussion below), also having applications in signal processing.The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called the "window", which slides, entry by entry, over the entire signal. For 1D signals, the most obvious window is just the first few preceding and following entries, whereas for 2D (or higher-dimensional) signals such as images, more complex window patterns are possible (such as "box" or "cross" patterns). Note that if the window has an odd number of entries, then the median is simple to define: it is just the middle value after all the entries in the window are sorted numerically.

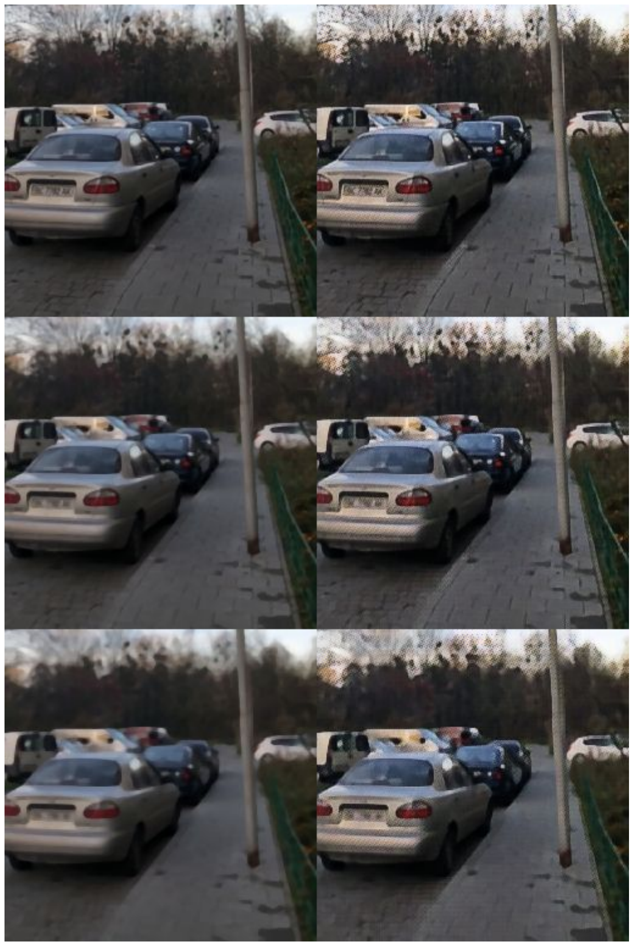

Figure.8 Top: Median blurred image(ksize=5), before and after deblur; Middle: Median blurred image(ksize=9), before and after deblur; Bottom: Median blurred image(ksize=13), before and after deblur

Result
To show how our model perform on different blurred kernels, instead of analyzing the difference between the original sharp image and the deblurred image, we determined a relative difference, that is we analyze the difference between blurred images and deblurred images. This is because deblur effect is a relative effect. It is very likely that the larger the kernel size, the more blurred the image, the more difference between the sharp and deblurred image. Here, we chose Gaussian and Median kernels and set kernel size to 5, 9 and 13 respectively.

Results showed that, our model performs better on Gaussian kernels since the distance between blurred and deblurred images are larger than those images created by Median kernels. Meanwhile, our model also showed that, for Gaussian kernel blurred images the larger the kernel size, the more deblur effect, since as the kernel size became larger, the distance between the blurred and deblurred images increased. However, for Median kernel blurred images, the effect remained the same with different kernel sizes.

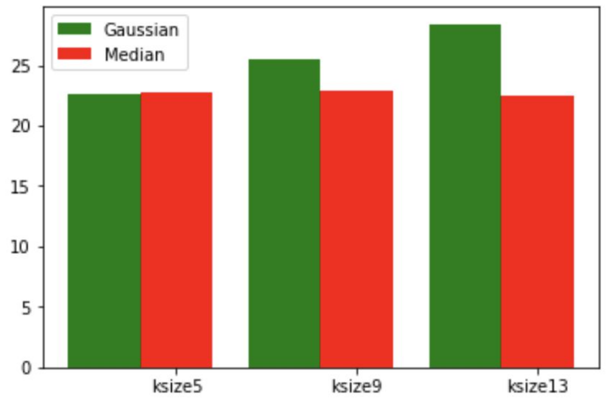

Fig.9 Deblur effect with respect to kernel size

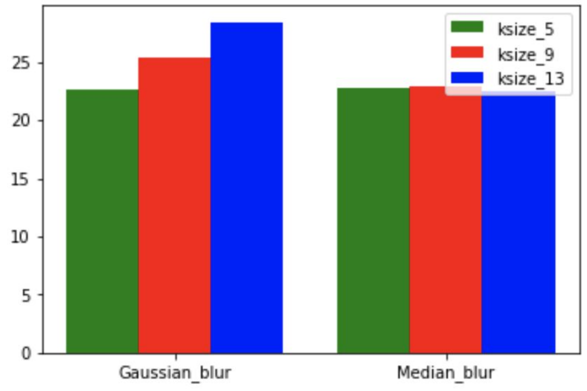

Fig.10 Deblur effect with respect to kernel type

5.4 Discussion of Insights Gained

We would say the implementation and replication of the novel ideas from the original Deblur GAN paper is a enjoyable and illuminating process. Starting from zero, understanding of the model architecture and complicated components of loss function with gradient penalty took time but are really educational. The most exciting thing through our project was to babysitting the model and to see it become better and better, finally showed great performance on deblurring images.
Our team did not have experience of GANs previously so that we learnt a lot while exploring how the GANs make sense. One of the novel idea of the paper is to treat the image deblurring problem as a image to image generation process, which has greatly broadened our insights about GANs and the applications of them.
While training and replicating the Deblur GAN, we gradually notice that it was very sensitive to subtle changes of loss function's parameters. It made the choosing and searching of parameters a tricky task which finally lead to our better understanding of WGAN loss and residual learning.
Our training process did not go well at the very beginning and we encountered some technical challenges including the sudden switch of colors of output images, limitation of computation and difference between PyTorch and Tensorflow. To resolve those problems, carefully debugging and fully understanding of the paper's ideas is important useful, leading us to the conclusion that transferring tech to product requires mastery of both theoretical and practical knowledges.
It seems that the Deblur GAN performs better if the images are more blurring and having irregular borders. A plausible explanation is that the deblurring process have similar pattern with features extraction in CNN and firstly learn weights better if the boundary is regular. In the original paper, the authors trained the model for 160 epoches and it took six weeks. Therefore we expect the model could be better after more epoches. Meanwhile, by generating testing data with different blur kernels, we discover that our model gives better performance for specific kernel like Gaussian kernel. One possible reason is that Gaussian blur algorithm is more similar to the original blur algorithm used to create blurred images for training.
Through experiments, our model showed great deblurring cabacity and generalizing ability for images shot by us, licences and clinical images, thus we believe that the model could have broad applications in the future.

6. Conclusion

We replicated and the kernel-free blind motion deblurring learning approach and implemented DeblurGAN which is a Conditional Adversarial Network that is optimized using a multi-component loss function. Despite not matching the authors' PSNR exactly, we did show our implementation have great performance on image deblurring. Besides that, we optimize the training procedure and save much training time. Moving forward, we test the sensitivity of the model using different pre-trained model and the it's performance on different blur kernels.

7. References

[1]Bibitbucket link:
*https://bitbucket.org/ecbm6040_2018spring/2018_e6040_*

yl3832/src/ed66b6abe81e/project_DGAN_yl3832_yl3829_yc3332%20/?at=master
GitHub link:
https://github.com/yl3829/deblur_tf
[2] Kupyn, Orest, et al. "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks." *arXiv preprint arXiv:1711.07064* (2017).
[3] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." *arXiv preprint arXiv:1701.07875* (2017).
[4] "The GoPro dataset," *online*: https://drive.google.com/file/d/1H0PIXvJH4c40pk7ou6nAwoxuR4Qh_Sa2/view.
[5] Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." *Advances in Neural Information Processing Systems*. 2017.
[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015. 3, 5

8. Appendix

Individual student contributions

TABLE 2
Contributions of each team member to the overall project.

	yl3829	**yc3332**	**yl3832**
Last name	Lin	Chou	Lyu
Fraction of contribution	1/3	1/3	1/3
Contribution 1	Built discriminator in Tensorflow, and implemented WGAN-GP loss, perceptual loss.	Implemented the VerySimpleDeglurGAN for simple use	Data organization, saving, reusing, preprocess, and deprocess
Contribution 2	Trained models and analysis the generated images	Built generator in Tensorflow	Trained model and parameter searching and tuning
Contribution 3	Proposed to first train generator than the discriminator to save more training time.	Blurred and analyzed images using different kernels	Model training, testing and image generating in Tensorflow Results evaluations

All team members contributed to the final write-up.