Motion Deblurring Using Generative Adversarial Networks

Kupyn Orest

computer science @UCU

Who am I?

- Software Engineer at SoftServe
- Student at UCU

Areas of interest:

- Computer Vision
- Artificial Intelligence

Ukrainian Catholic University

csds.ucu.edu.ua

What Is Motion Blur?

Sharp Image:

$$x_0 \mid x_1 \mid x_3 \mid x_4 \mid x_5 \mid x_6 \mid$$

Blurred Image:

$$x_1 + x_0 \mid x_2 + x_1 \mid x_3 + x_2 \mid x_4 + x_3 \mid x_5 + x_4 \mid x_6 + x_5 \mid$$

Deblurred Image:

$$x_1 + x_0 \mid x_2 - x_0 \mid x_3 + x_0 \mid x_4 - x_0 \mid x_5 + x_0 \mid x_6 - x_0 \mid$$

Blur Model

$$G(x) = H(x) * F(x) + N(x)$$

G - Blurred Image

H - Point Spread Function

F - Sharp Image

N - Additive Noise

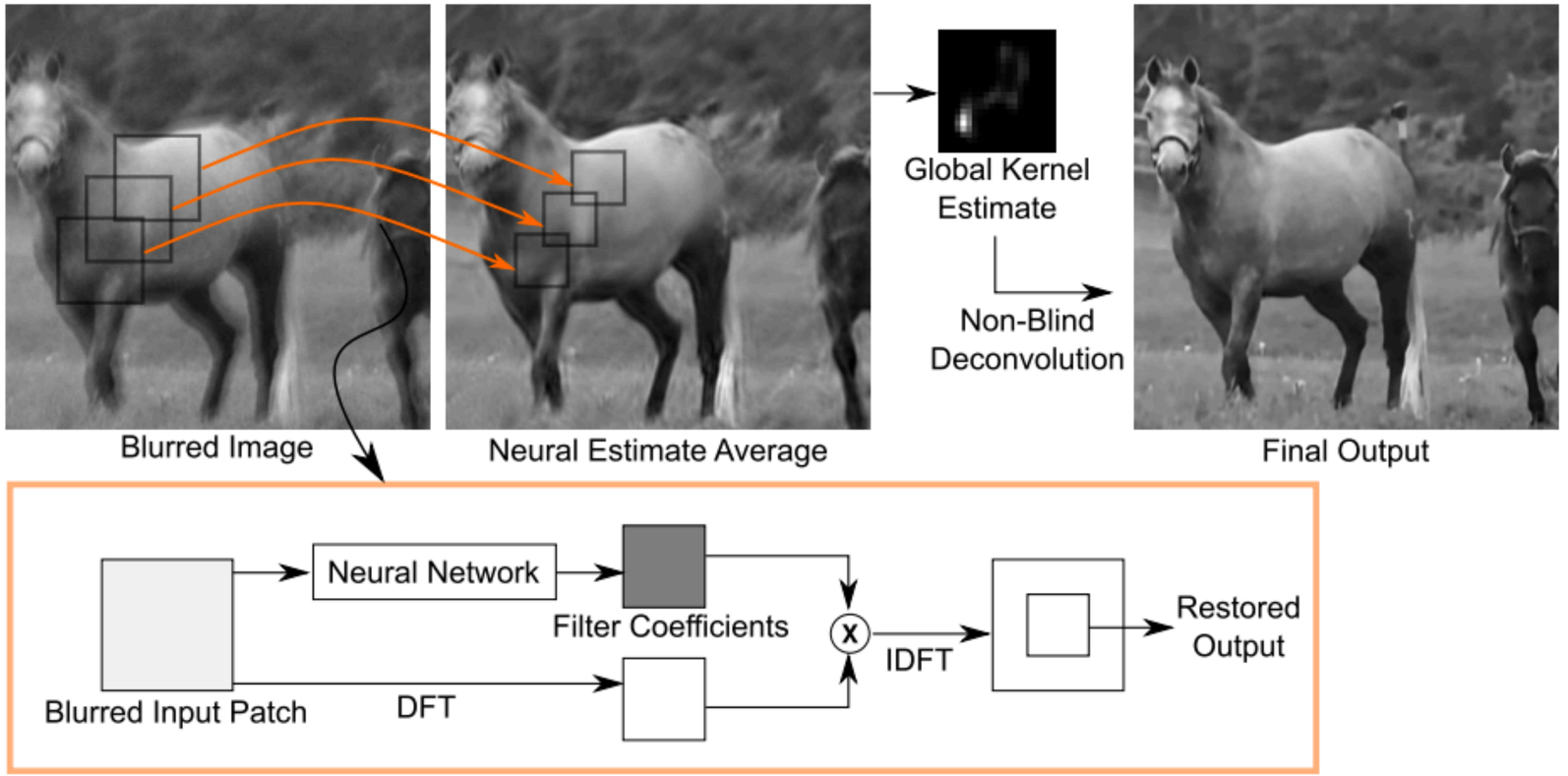
Early Deblurring Models

$$\hat{F}(u,v) = \left(\frac{1}{H(u,v)} \frac{\left|H(u,v)\right|^2}{\left|H(u,v)\right|^2 + S_\eta(u,v) / S_f(u,v)}\right) G(u,v)$$ - Wiener Filter

$$\hat{F}(u,v) = \left(\frac{H^*(u,v)}{\left|H(u,v)\right|^2 + y\left|P(u,v)\right|^2}\right) G(u,v)$$ - Tikhonov Filter

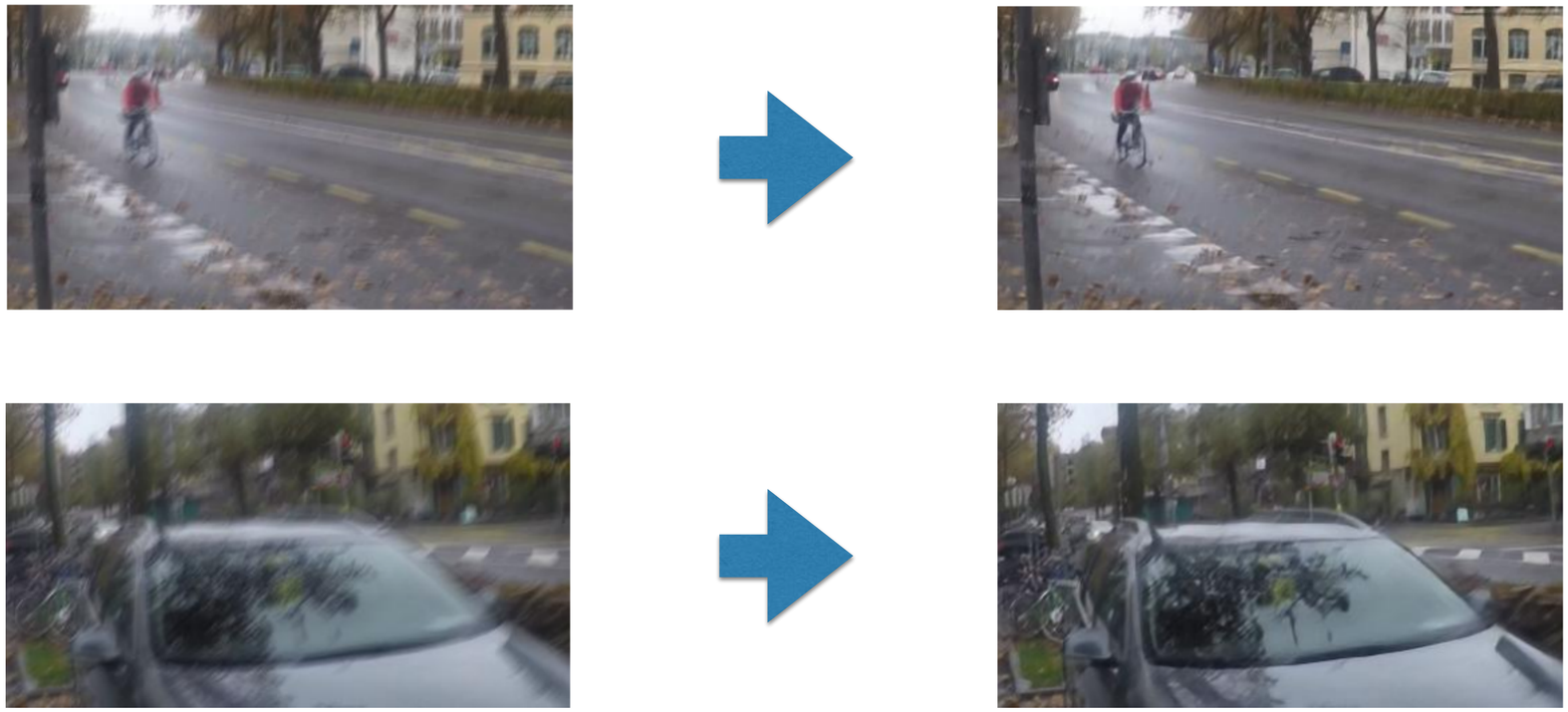
http://sernam.ru/tau_31.php

A Neural Approach to Blind Motion Deblurring



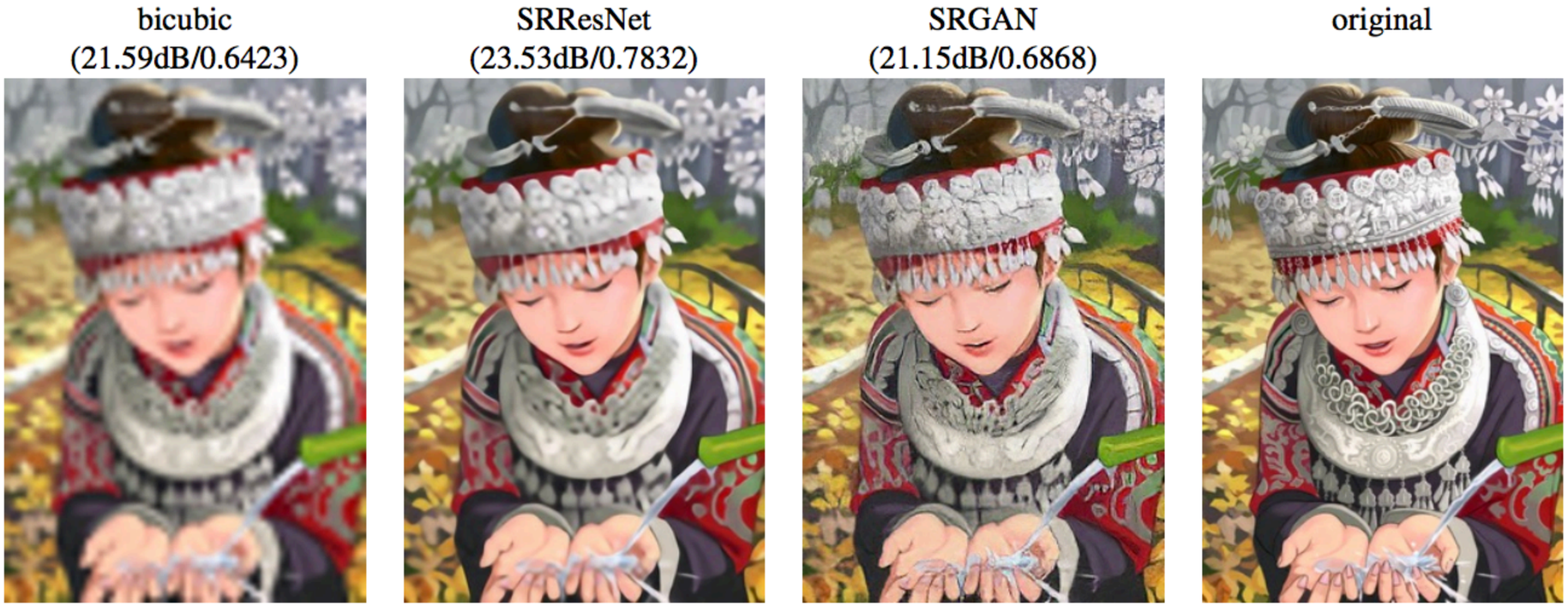
arXiv:1603.04771v2 - 2016

Motion Deblurring In The Wild



arXiv:1701.01486v1 - 2017

SRGAN

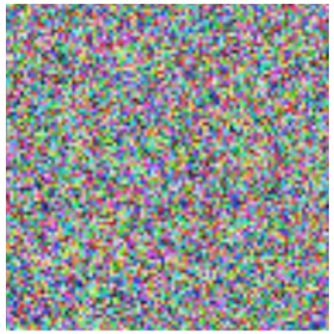


Last version came out 13 April 2017!!!

arXiv:1609.04802v4 - 2017

Generative Adversarial Networks

Noise ~ N(0,1)



Generative Model

arXiv:1406.2661v1 - 2014

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

```python
for epoch in xrange(num_epochs):
    for d_index in xrange(d_steps):
        # 1. Train D on real+fake
        D.zero_grad()

        #  1A: Train D on real
        d_real_data = Variable(d_sampler(d_input_size))
        d_real_decision = D(preprocess(d_real_data))
        d_real_error = criterion(d_real_decision, Variable(torch.ones(1)))  # ones = true
        d_real_error.backward() # compute/store gradients, but don't change params

        #  1B: Train D on fake
        d_gen_input = Variable(gi_sampler(minibatch_size, g_input_size))
        d_fake_data = G(d_gen_input).detach()  # detach to avoid training G on these labels
        d_fake_decision = D(preprocess(d_fake_data.t()))
        d_fake_error = criterion(d_fake_decision, Variable(torch.zeros(1)))  # zeros = fake
        d_fake_error.backward()
        d_optimizer.step()     # Only optimizes D's parameters; changes based on stored gradients from backward()

    for g_index in xrange(g_steps):
        # 2. Train G on D's response (but DO NOT train D on these labels)
        G.zero_grad()

        gen_input = Variable(gi_sampler(minibatch_size, g_input_size))
        g_fake_data = G(gen_input)
        dg_fake_decision = D(preprocess(g_fake_data.t()))
        g_error = criterion(dg_fake_decision, Variable(torch.ones(1)))  # we want to fool, so pretend it's all genuine

        g_error.backward()
        g_optimizer.step()  # Only optimizes G's parameters
```

(Train the detective D)

(Train the forger G)

Tips for GAN training

- Normalize the input (-1;1) and use tanh or sigmoid as last layer in discriminator
- $\max \log(D)$ instead of $\min \log(1-D)$ //helps to solve vanishing gradients problem
- Sample from Gaussian Distribution, not Uniform
- Different mini-batches for real and fake (in batch either all real or all fake)
- Use ADAM (or SGD for discriminator and ADAM for generator)
- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2
- Use ReLU => lower chance of mode collapse
- Track failures early

Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

arXiv:1701.07875v2 - 2017

Wasserstein GAN

- Train Discriminator (Critic) more than Generator
- No log in the loss => no sigmoid as last layer
- Clip weights to satisfy Lipschitz constraint
- Do not use ADAM, use RMSProp instead
- Lower learning rate

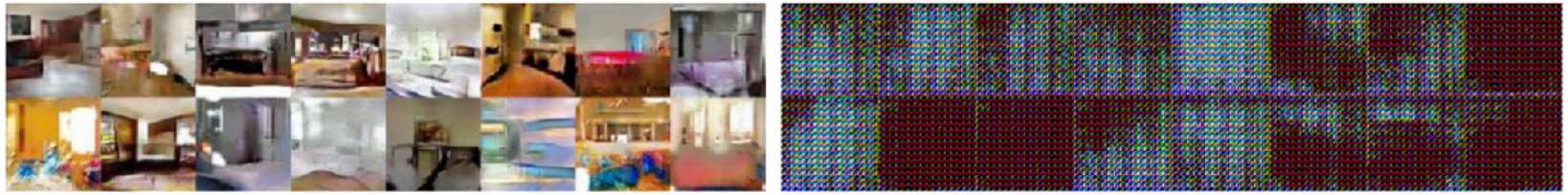


Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [17]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. Left: WGAN algorithm. Right: standard GAN formulation. As we can see the standard GAN failed to learn while the WGAN still was able to produce samples.

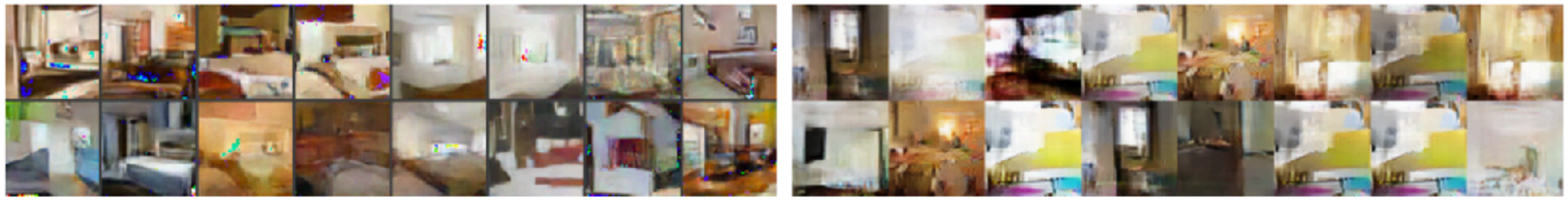
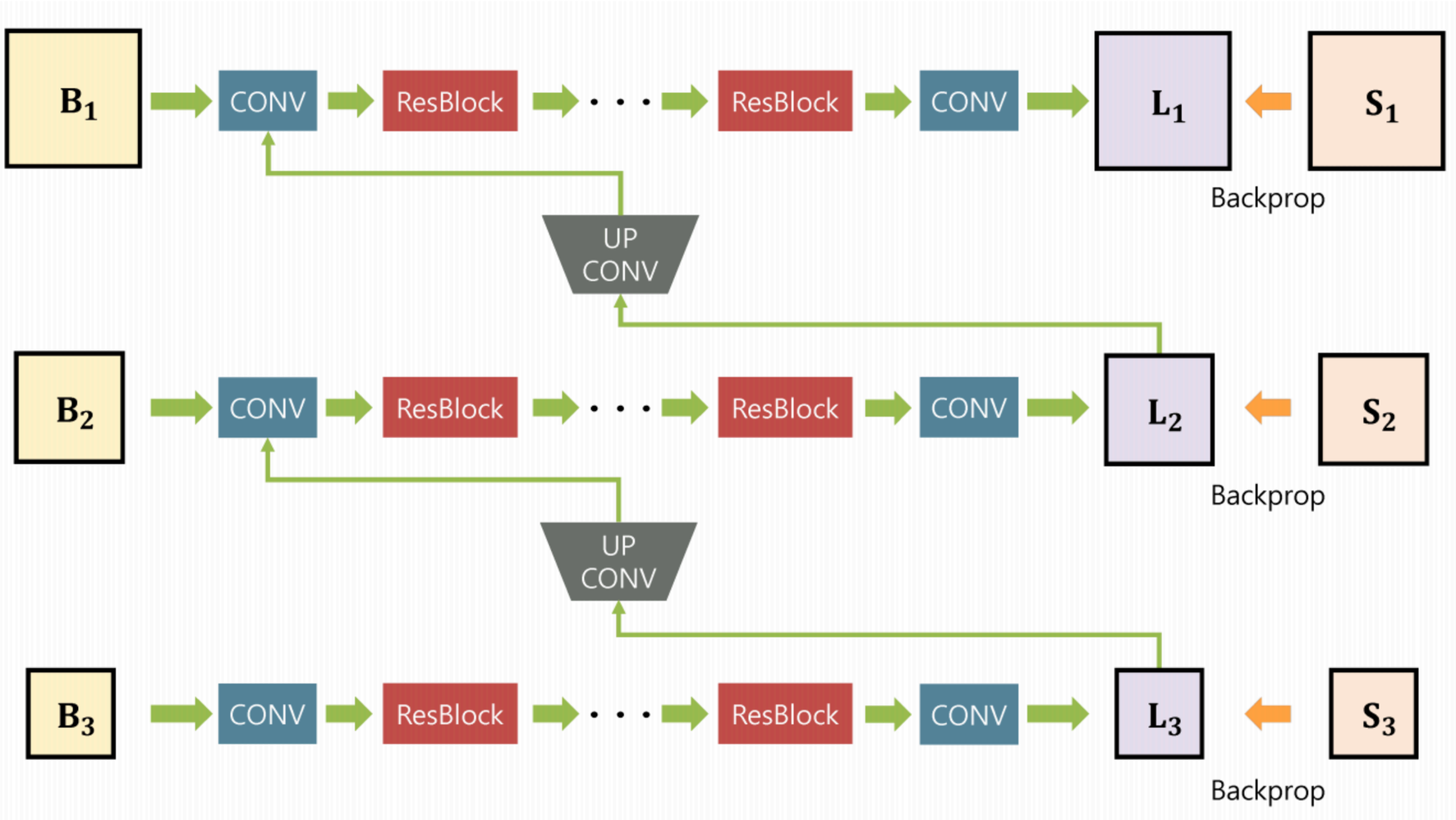


Figure 7: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.

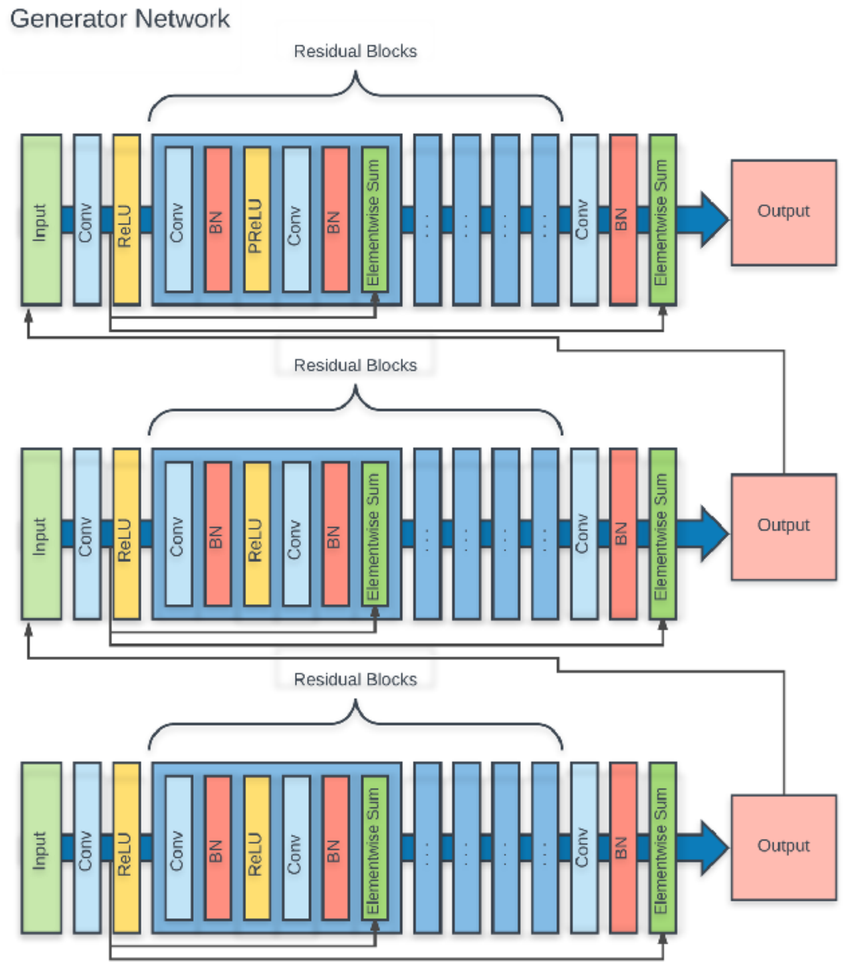
https://github.com/martinarjovsky/WassersteinGAN

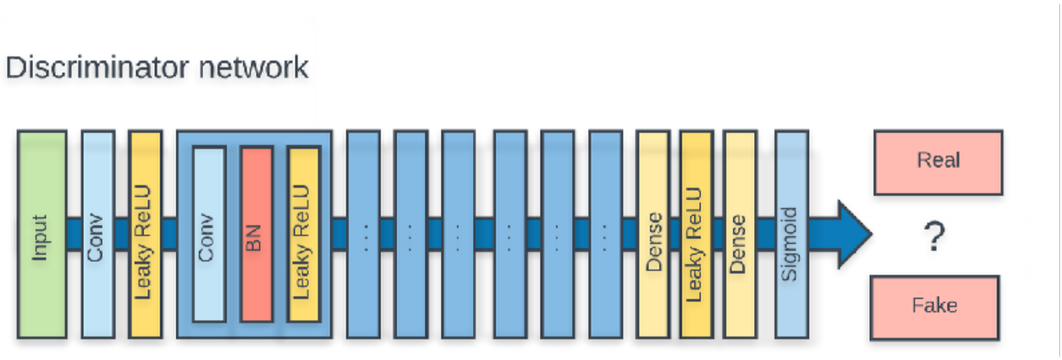
Is it possible to apply GAN to Image Deblurring problem?

Multiscale CNN for Motion Deblurring



Multiscale CNN for Motion Deblurring





What is the tensor size of coarser and finer input? (e.g. Image size = 64x64)

Our Results

Thanks for your attention!

And looking forward to your questions!