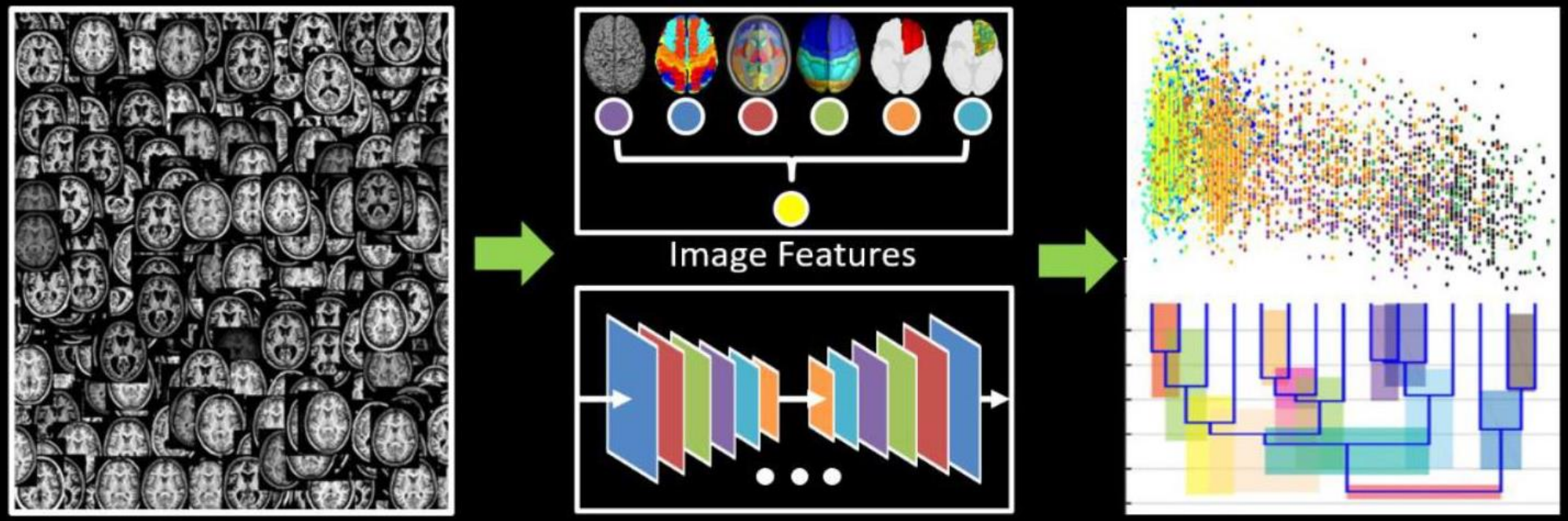


Generative Adversarial Network

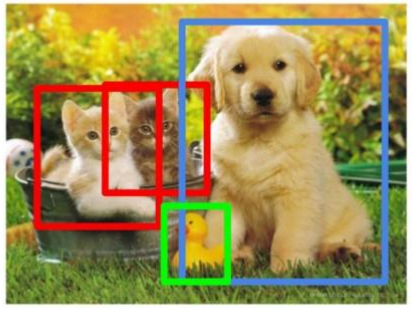
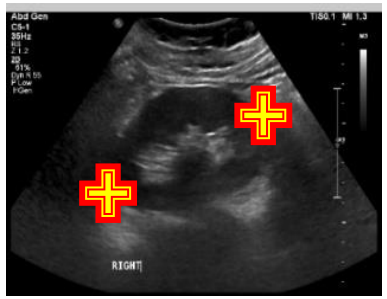
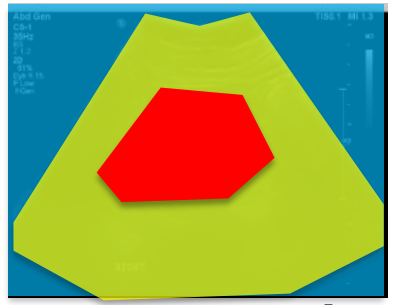
[Spring 2020 CS-8395 Deep Learning in Medical Image Computing]

Instructor: Yuankai Huo, Ph.D.
Department of Electrical Engineering and Computer Science
Vanderbilt University

Topics

- Review
- Semantic Segmentation
- U-Net
- Instance Segmentation

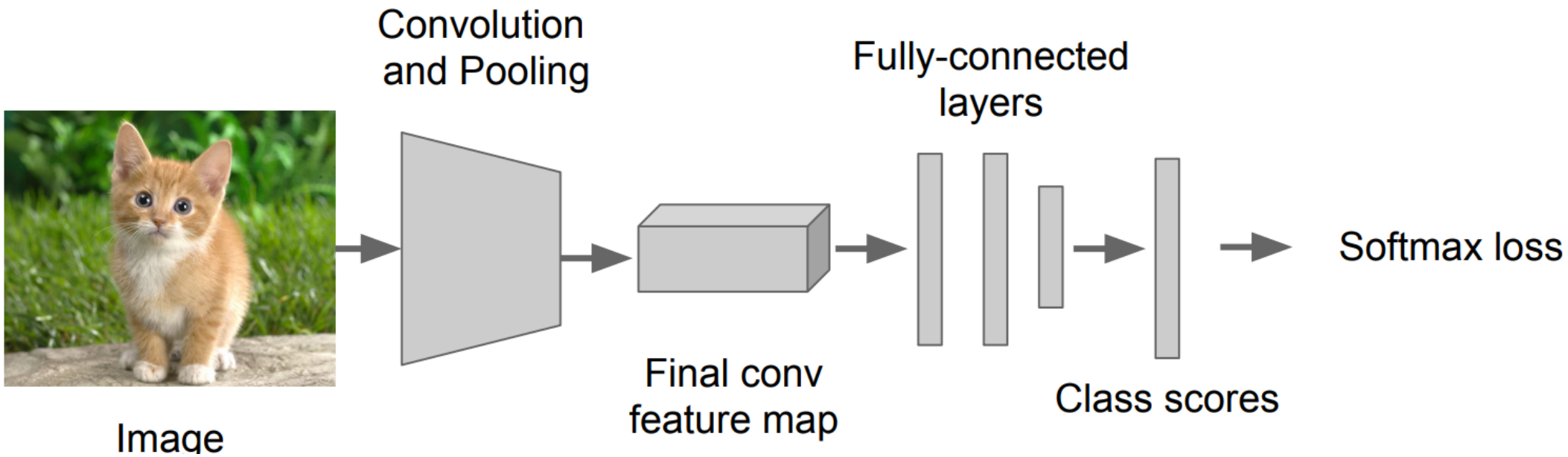
Tasks

	Classification	Classification + Localization	Object Detection	Instance Segmentation	Synthesis
Computer Vision	 CAT	 CAT	 CAT, DOG, DUCK	 CAT, DOG, DUCK	
Medical Image Computing	 Input	 Classification	 Detection	 Segmentation	 Synthesis

https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

Simple Recipe for Classification

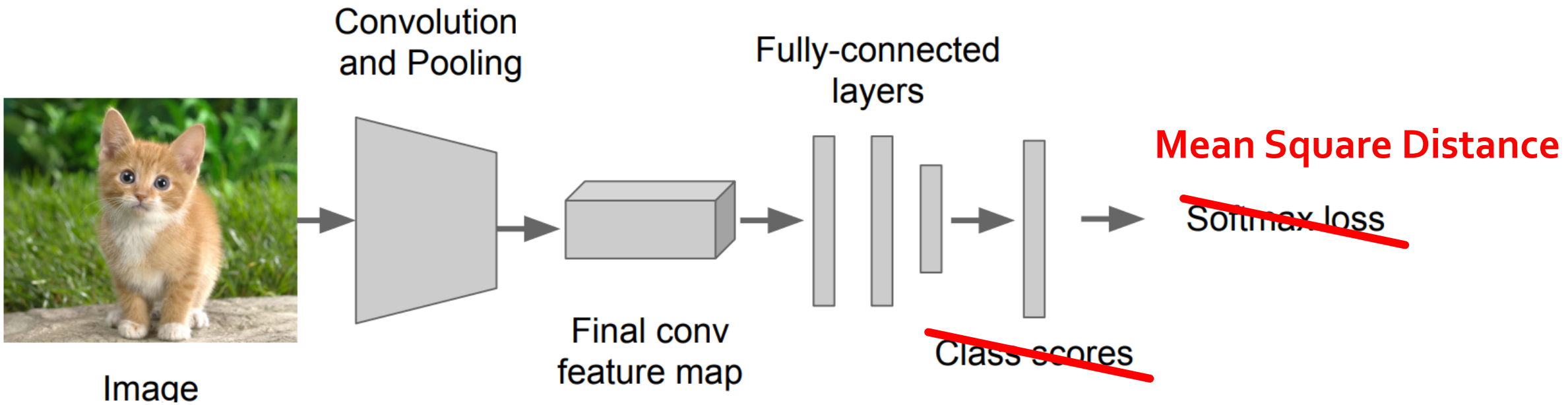
"CONV-POOL-FC"



http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Simple Recipe for Localization

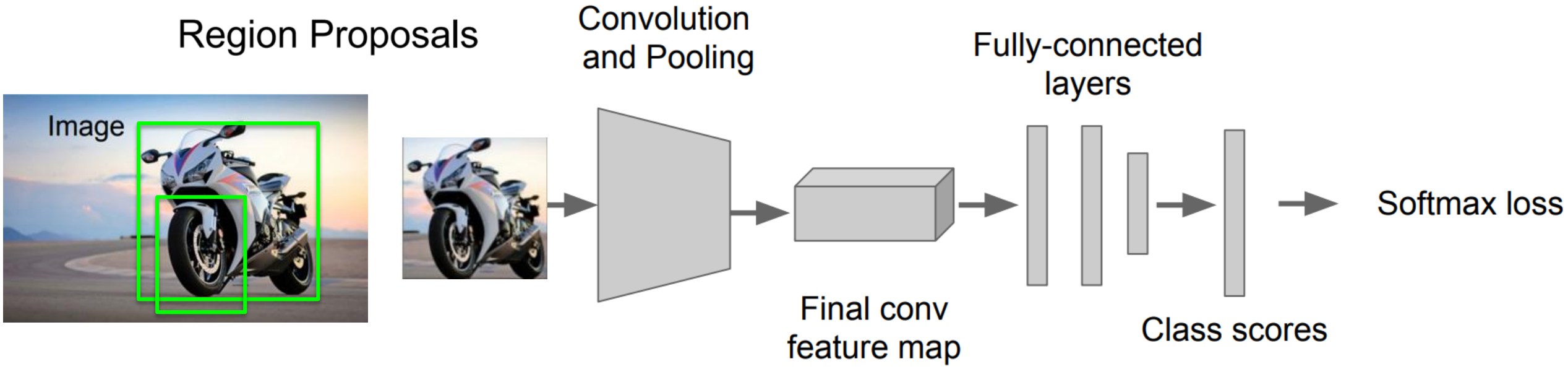
"CONV-POOL-FC"



http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Detection

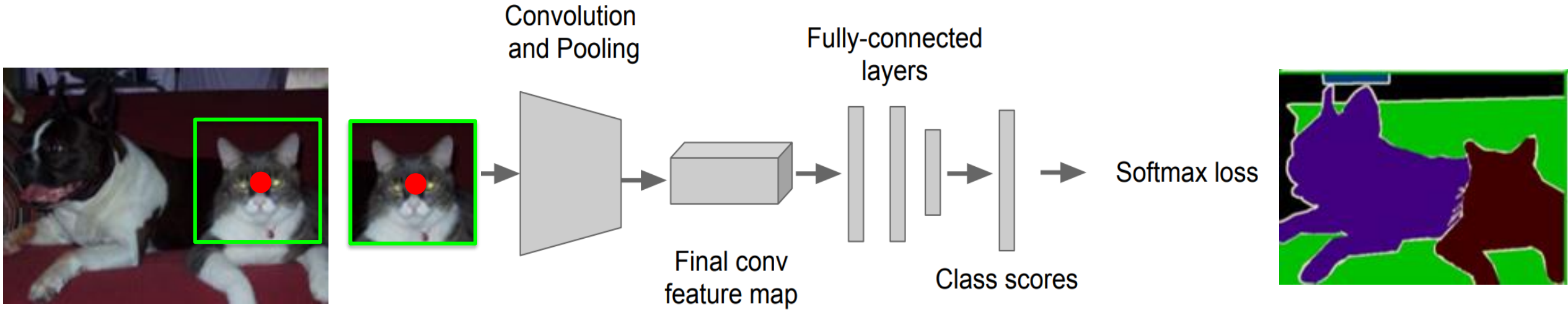
"CONV-POOL-FC"



http://slazebni.cs.illinois.edu/spring17/lec07_detection.pdf
https://blog.csdn.net/sum_nap/article/details/80388110
http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Segmentation

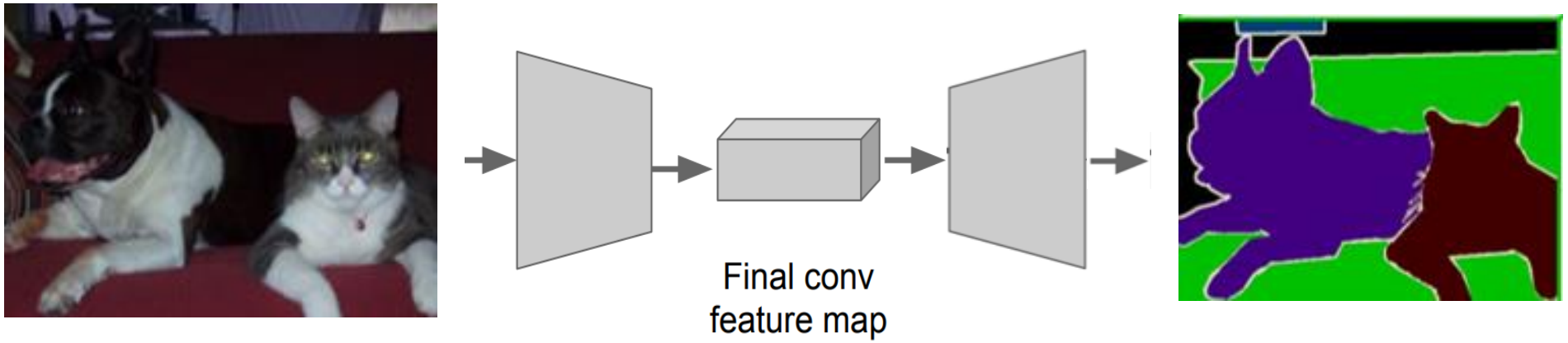
“CONV-POOL-FC”



http://slazebni.cs.illinois.edu/spring17/lec07_detection.pdf
https://blog.csdn.net/sum_nap/article/details/80388110
http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

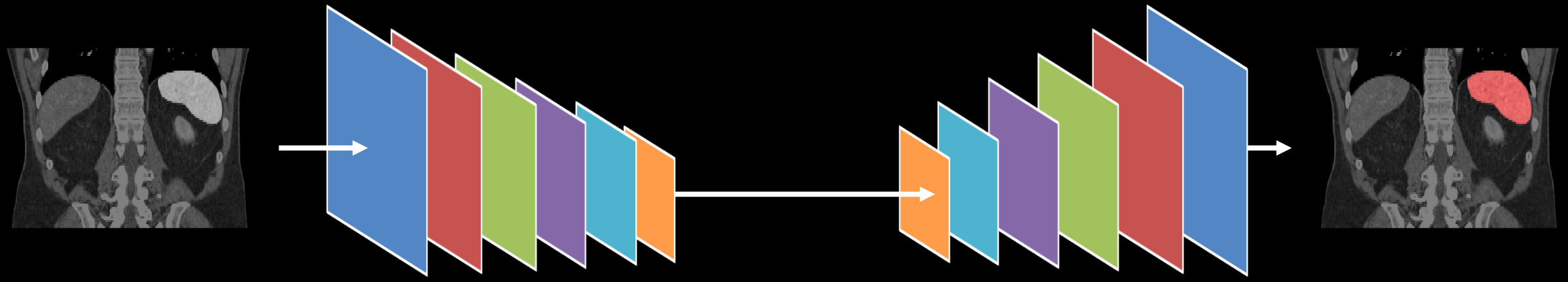
Segmentation

“Fully Conv Net”

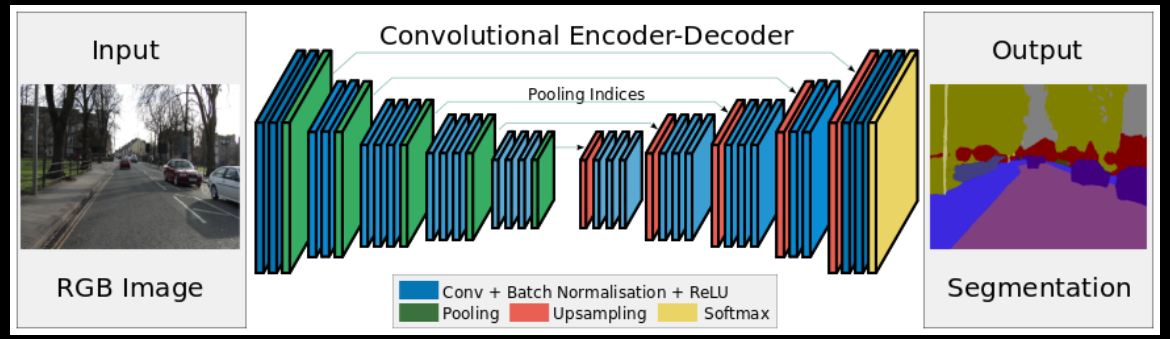


http://slazebni.cs.illinois.edu/spring17/lec07_detection.pdf
https://blog.csdn.net/sum_nap/article/details/80388110
http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

Encoder – Decoder Strategy

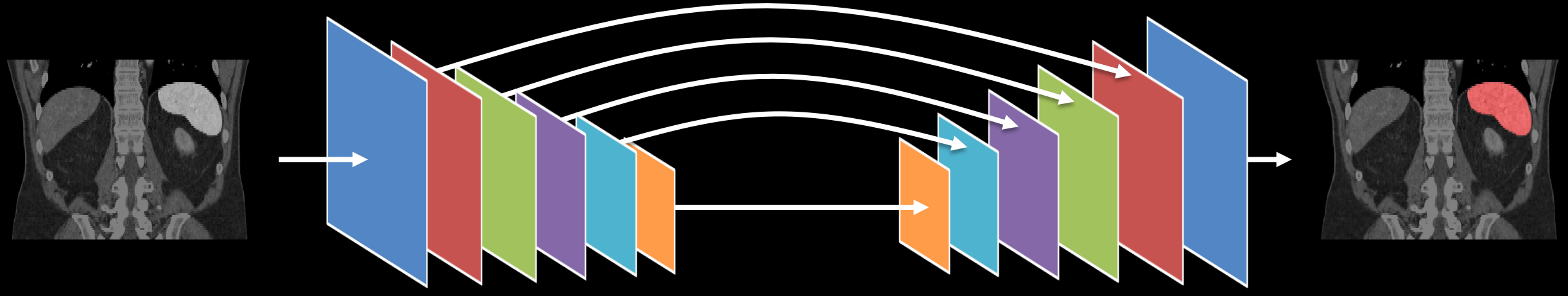


SegNet:



Badrinarayanan, et al. arXiv:1511.00561(2015)

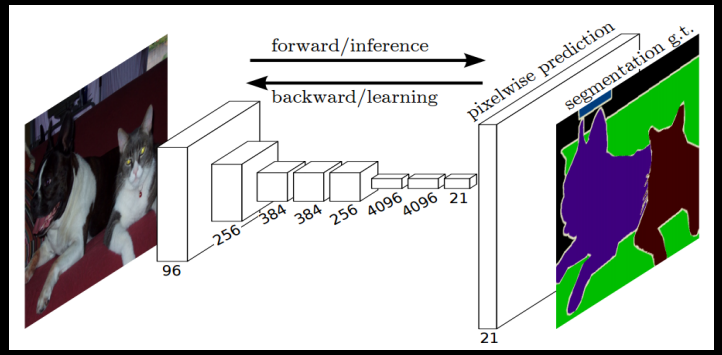
Fully Convolutional Network (FCN)



Encoder

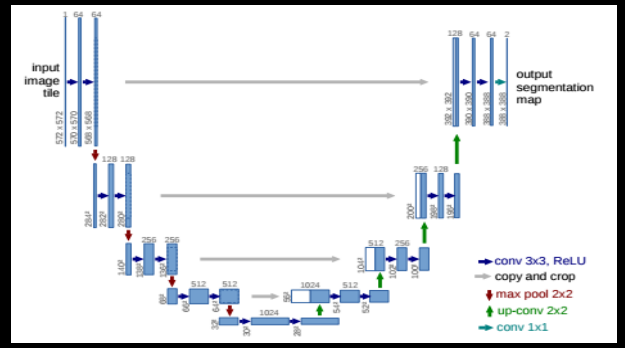
Decoder

FCN:



Long, et al. CVPR (2015)

Unet:



Ronneberger, et al. MICCAI (2015)

All Code for U-Net

Encoder

```
layer1 = F.relu(self.conv1_input(x))
layer1 = F.relu(self.conv1(layer1))

layer2 = F.max_pool2d(layer1, 2)
layer2 = F.relu(self.conv2_input(layer2))
layer2 = F.relu(self.conv2(layer2))

layer3 = F.max_pool2d(layer2, 2)
layer3 = F.relu(self.conv3_input(layer3))
layer3 = F.relu(self.conv3(layer3))

layer4 = F.max_pool2d(layer3, 2)
layer4 = F.relu(self.conv4_input(layer4))
layer4 = F.relu(self.conv4(layer4))

layer5 = F.max_pool2d(layer4, 2)
layer5 = F.relu(self.conv5_input(layer5))
layer5 = F.relu(self.conv5(layer5))
```

Decoder

```
layer6 = F.relu(self.conv6_up(layer5))
layer6 = torch.cat((layer4, layer6), 1)
layer6 = F.relu(self.conv6_input(layer6))
layer6 = F.relu(self.conv6(layer6))

layer7 = F.relu(self.conv7_up(layer6))
layer7 = torch.cat((layer3, layer7), 1)
layer7 = F.relu(self.conv7_input(layer7))
layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8_up(layer7))
layer8 = torch.cat((layer2, layer8), 1)
layer8 = F.relu(self.conv8_input(layer8))
layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9_up(layer8))
layer9 = torch.cat((layer1, layer9), 1)
layer9 = F.relu(self.conv9_input(layer9))
layer9 = F.relu(self.conv9(layer9))
layer9 = self.final(self.conv9_output(layer9))

return layer9
```

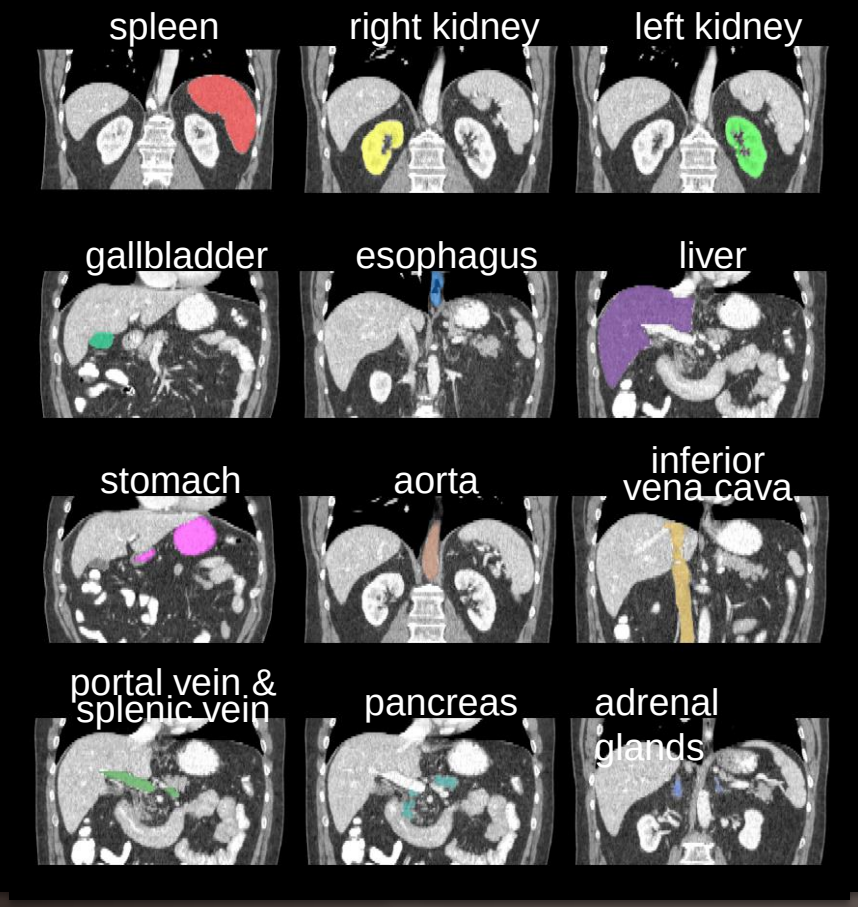
Parameters

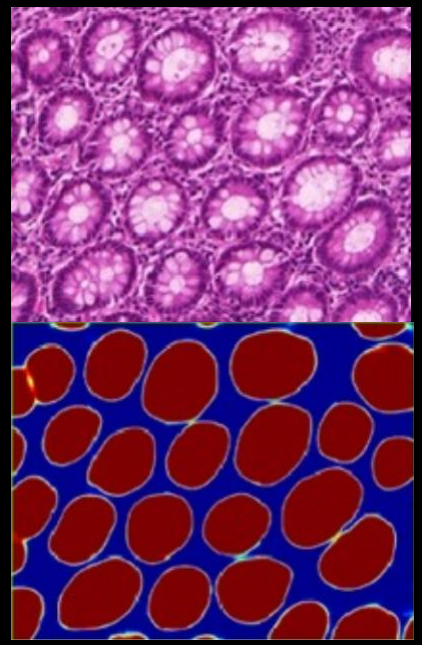
```
self.conv1_input =      nn.Conv2d(1, 64, 3, padding=1)
self.conv1 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv2_input =      nn.Conv2d(64, 128, 3, padding=1)
self.conv2 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv3_input =      nn.Conv2d(128, 256, 3, padding=1)
self.conv3 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv4_input =      nn.Conv2d(256, 512, 3, padding=1)
self.conv4 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv5_input =      nn.Conv2d(512, 1024, 3, padding=1)
self.conv5 =            nn.Conv2d(1024, 1024, 3, padding=1)

self.conv6_up =         nn.ConvTranspose2d(1024, 512, 2, 2)
self.conv6_input =      nn.Conv2d(1024, 512, 3, padding=1)
self.conv6 =            nn.Conv2d(512, 512, 3, padding=1)
self.conv7_up =         nn.ConvTranspose2d(512, 256, 2, 2)
self.conv7_input =      nn.Conv2d(512, 256, 3, padding=1)
self.conv7 =            nn.Conv2d(256, 256, 3, padding=1)
self.conv8_up =         nn.ConvTranspose2d(256, 128, 2, 2)
self.conv8_input =      nn.Conv2d(256, 128, 3, padding=1)
self.conv8 =            nn.Conv2d(128, 128, 3, padding=1)
self.conv9_up =         nn.ConvTranspose2d(128, 64, 2, 2)
self.conv9_input =      nn.Conv2d(128, 64, 3, padding=1)
self.conv9 =            nn.Conv2d(64, 64, 3, padding=1)
self.conv9_output =     nn.Conv2d(64, 2, 1)
```

https://github.com/jonnedtc/U-Net-PyTorch/blob/master/networks.py

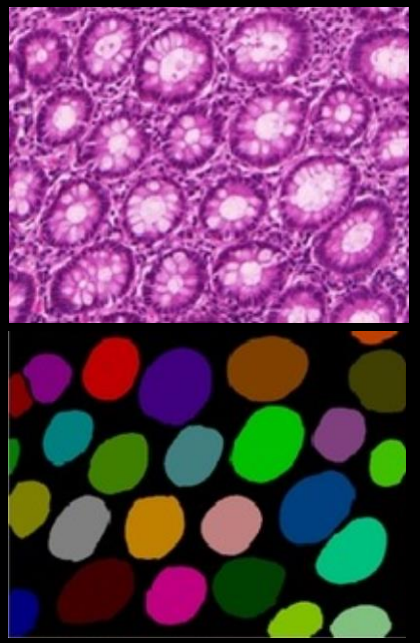
Instance Segmentation

Semantic Segmentation



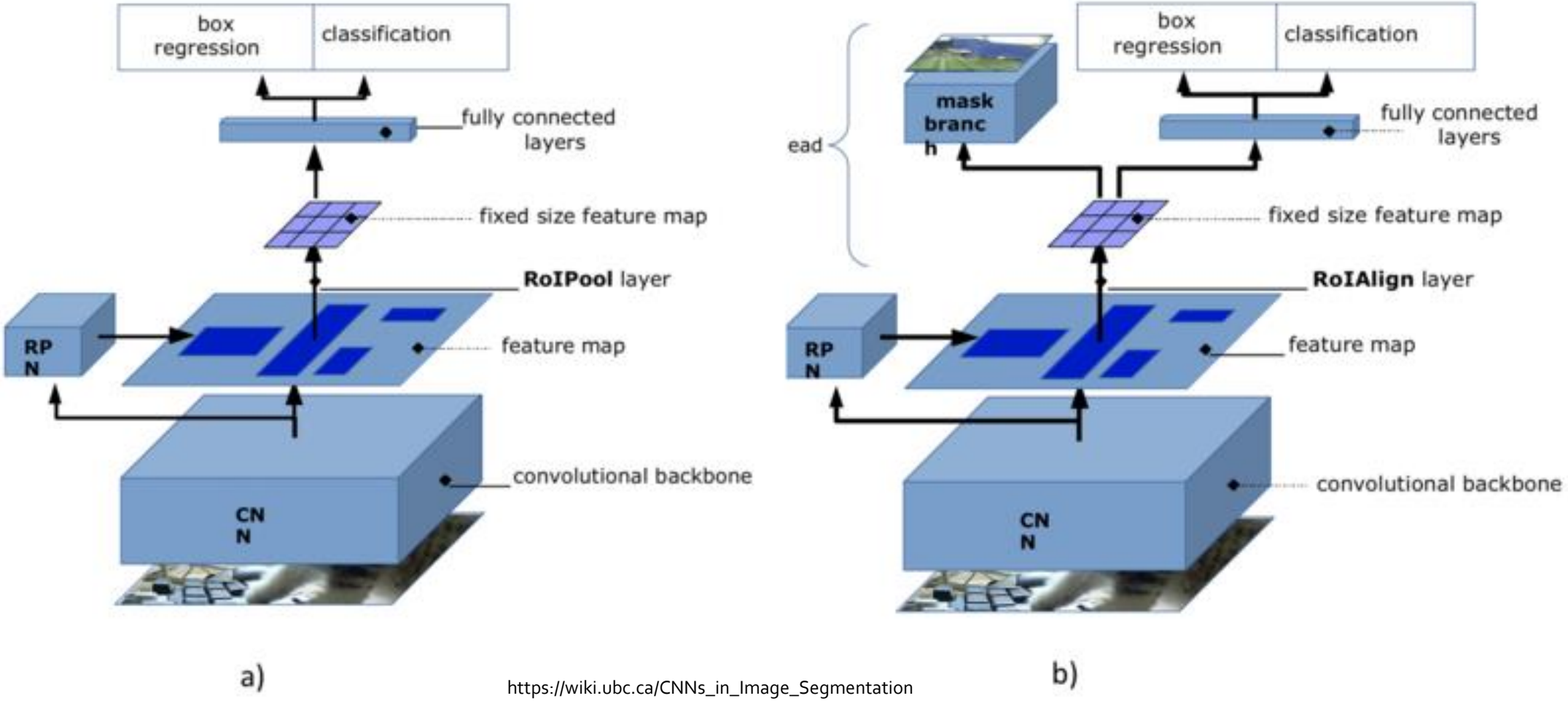


Instance Segmentation



https://www.sciencedirect.com/science/article/pii/S1361841516302043

Faster R-CNN vs. Mask R-CNN



https://wiki.ubc.ca/CNNs_in_Image_Segmentation

Image Synthesis

https://junyanz.github.io/CycleGAN/

More Examples

Converting Monet into Thomas Kinkade

Resurrecting Ancient Cities

Animal Transfiguration

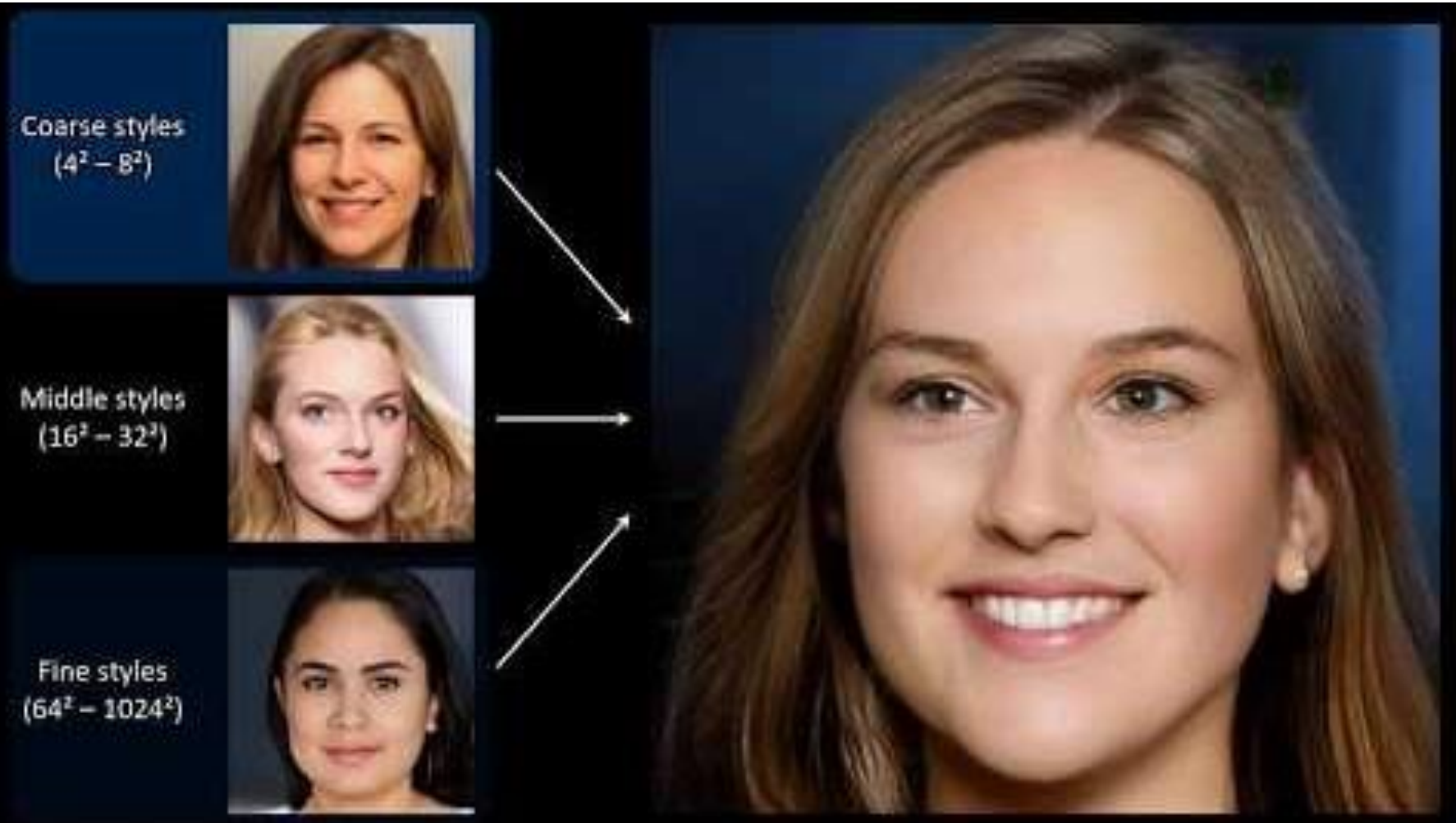
Portrait to Dollface

Face ↔ Ramen

Colorizing legacy photographs

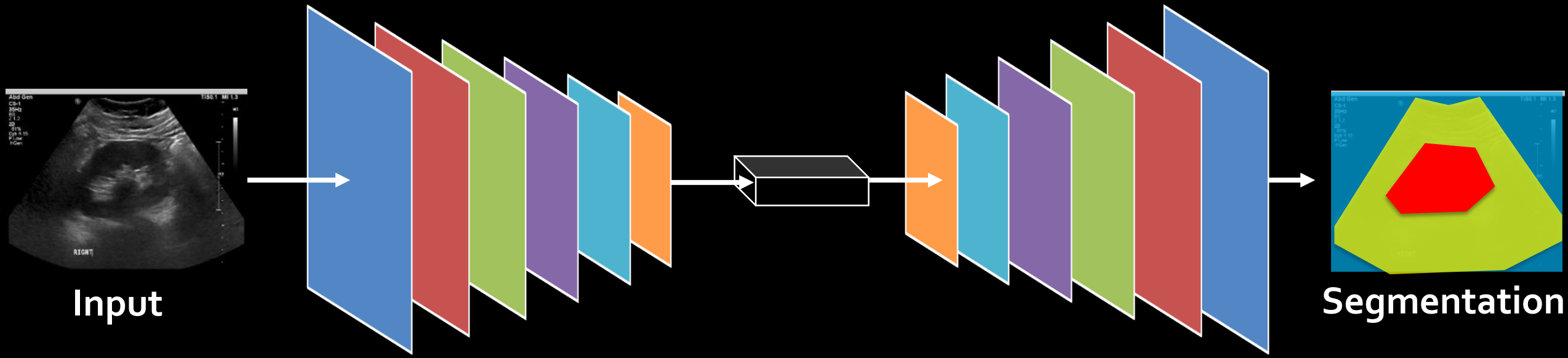
https://junyanz.github.io/CycleGAN/

State-of-the-art

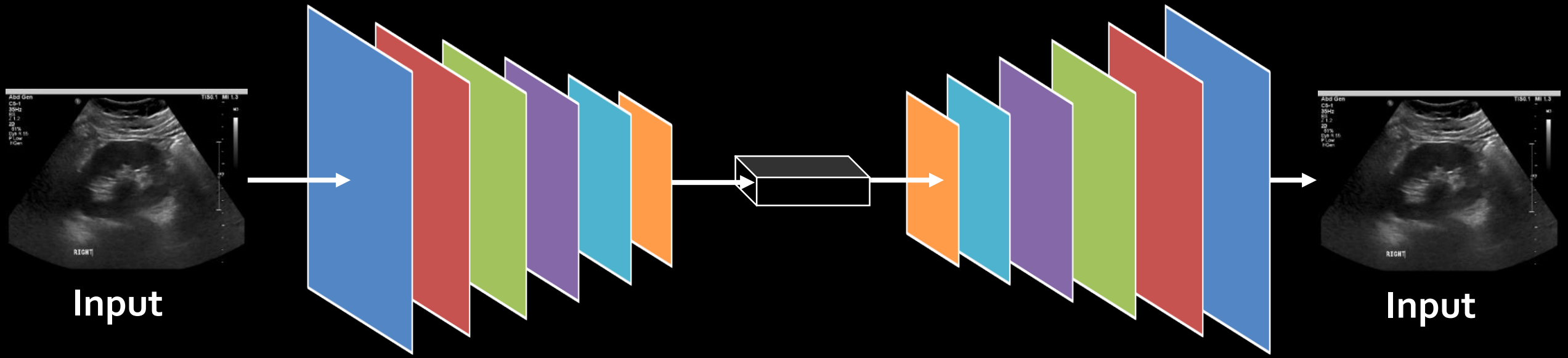


https://www.youtube.com/watch?v=kSLJriaOumA

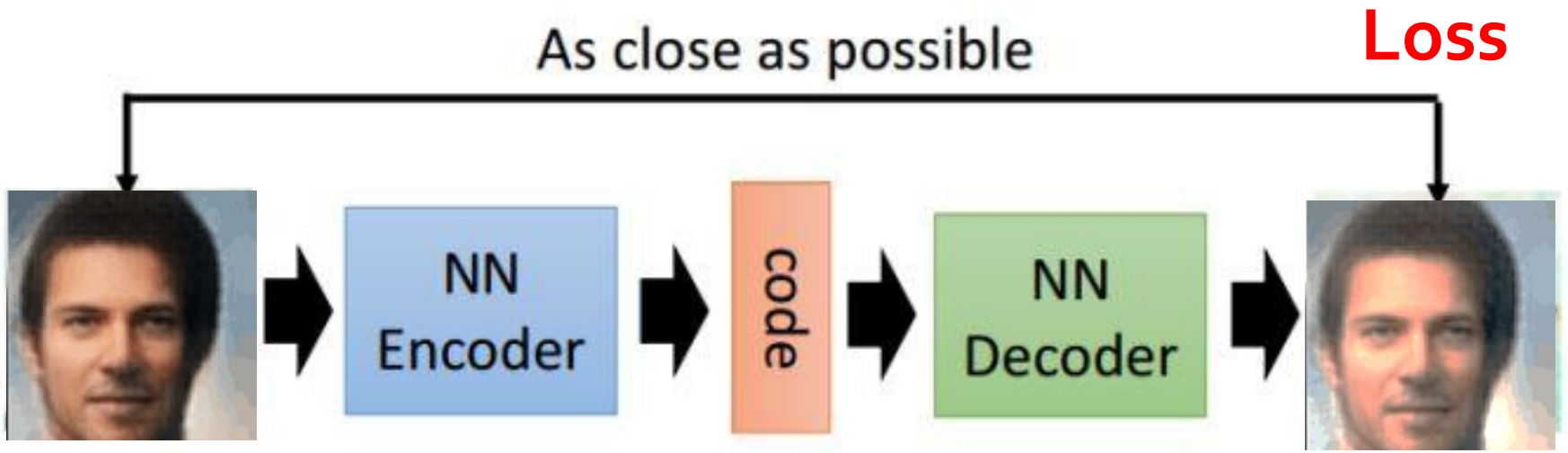
Segmentation (Image to Image)



Auto Encoder

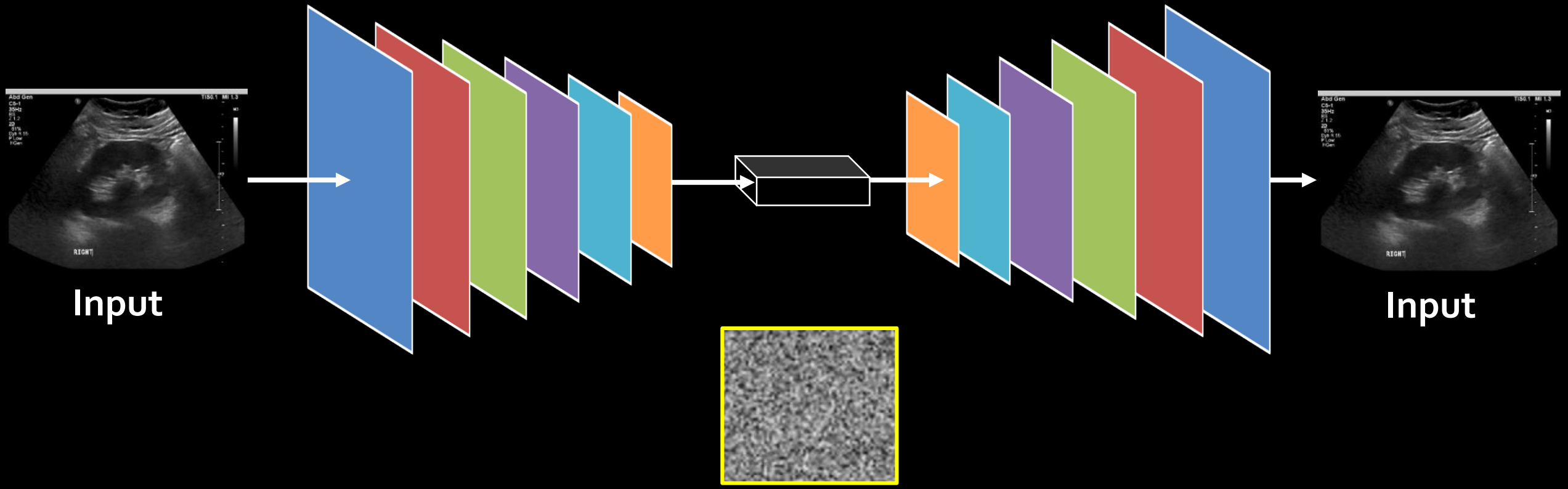


Let Deep Network Do It!



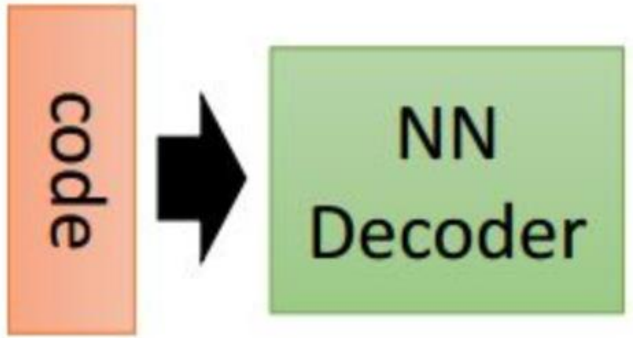
https://zhuanlan.zhihu.com/p/27549418

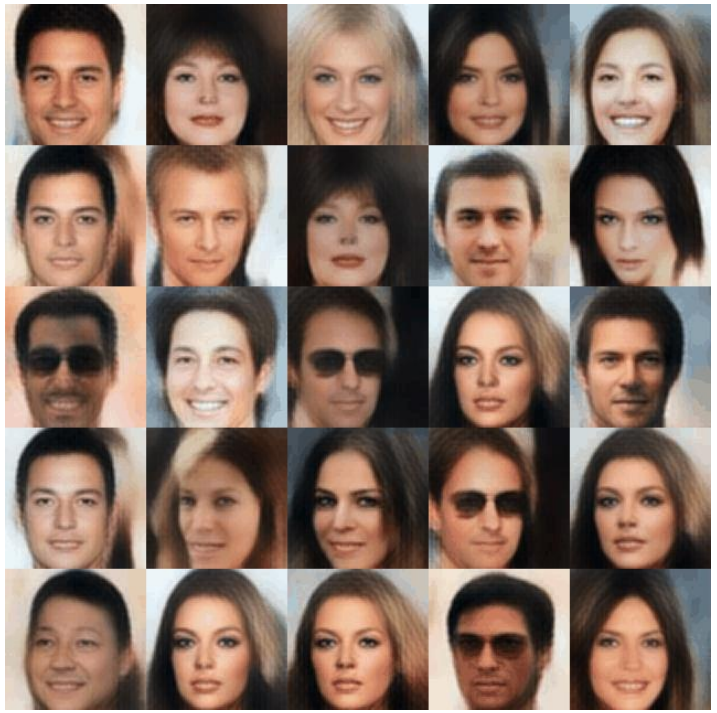
Auto encoder



Auto-encoder

Randomly generate a vector as code

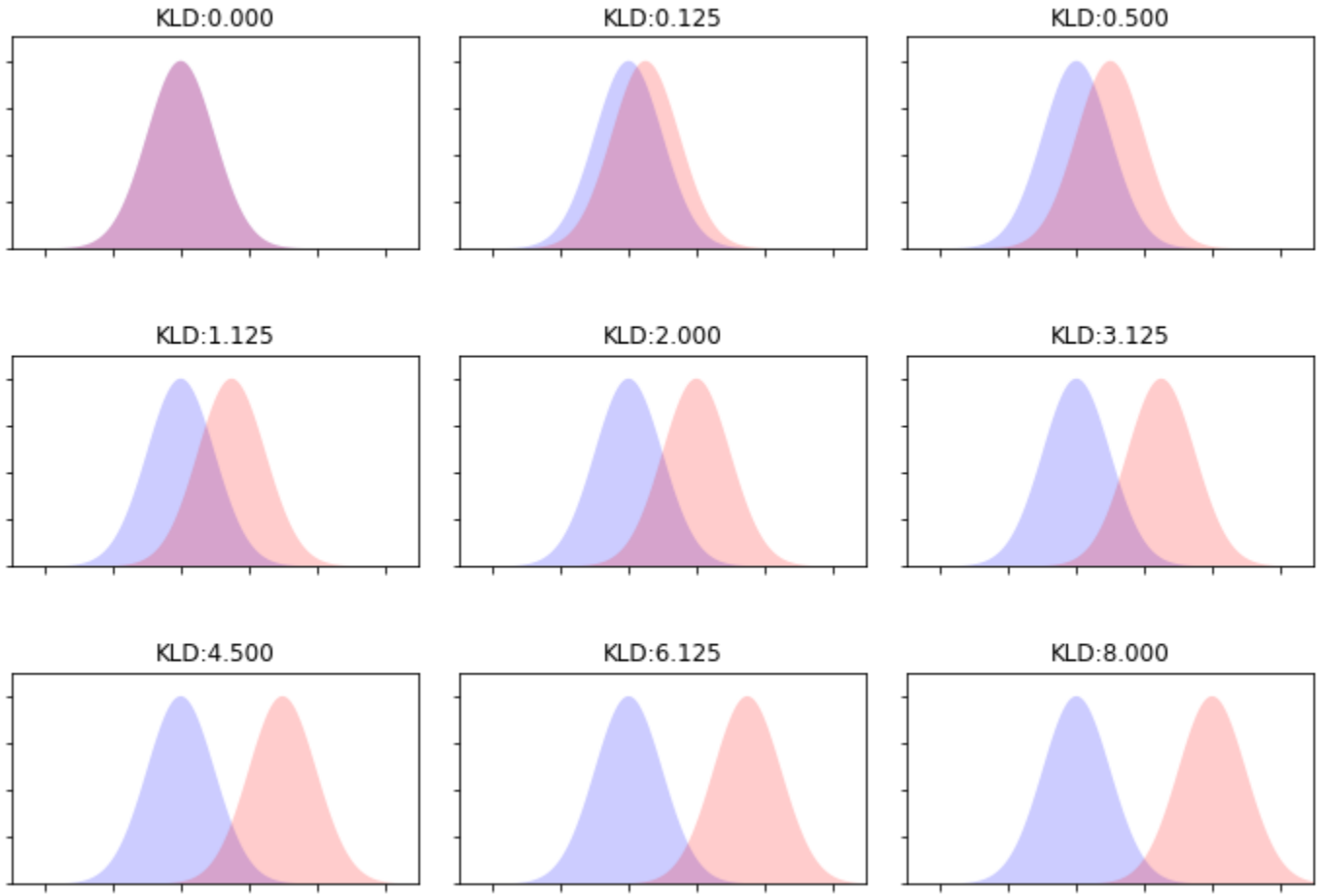




https://zhuanlan.zhihu.com/p/27549418

https://houxianxu.github.io/assets/project/dfcvae

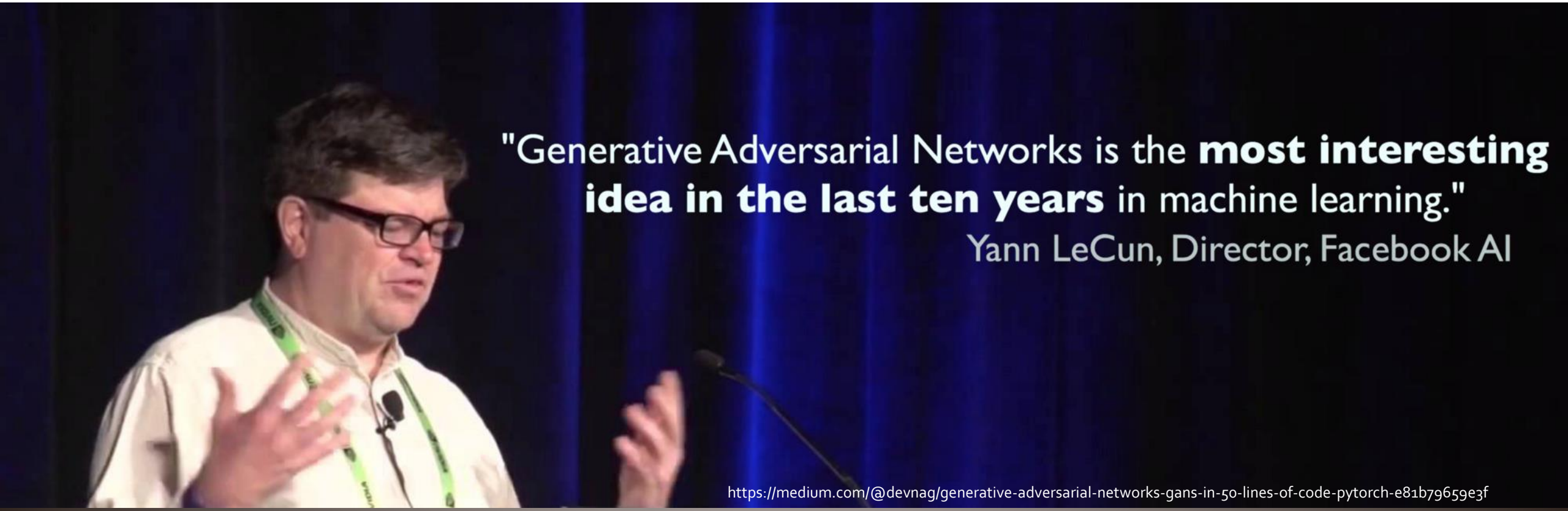
KL divergence



http://yusuke-ujitoko.hatenablog.com/entry/2017/05/07/200022

GAN

(Generative Adversarial Network)



https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

The GAN Father

Ian Goodfellow

Goodfellow obtained his B.S. and M.S. in computer science from Stanford University and his Ph.D. in machine learning from the Université de Montréal, under the supervision of Yoshua Bengio and Aaron Courville. After graduation, Goodfellow joined Google as part of the Google Brain research team. He then left Google to join the newly founded OpenAI institute. He returned to Google Research in March 2017.

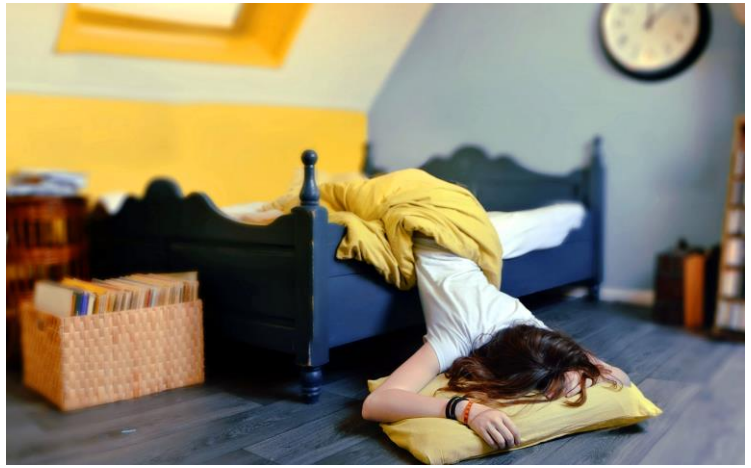
"I invented generative adversarial networks"

http://tellmenothing.com/2017/05/13/couple-sleep-memes-night-owls/

Story of How the GAN was Invented

https://www.peekyou.com/ian_goodfellow

http://bigbangtheory.wikia.com/wiki/File:Pos8.jpg



http://tellmenothing.com/2017/05/13/couple-sleep-memes-night-owls/

Example

https://consumerist.com/2007/04/25/hey-bank-of-america-your-atm-gave-me-a-fake-20/

Example

https://consumerist.com/2007/04/25/hey-bank-of-america-your-atm-gave-me-a-fake-20/

Example

https://www.wisebread.com/how-to-spot-counterfeit-money

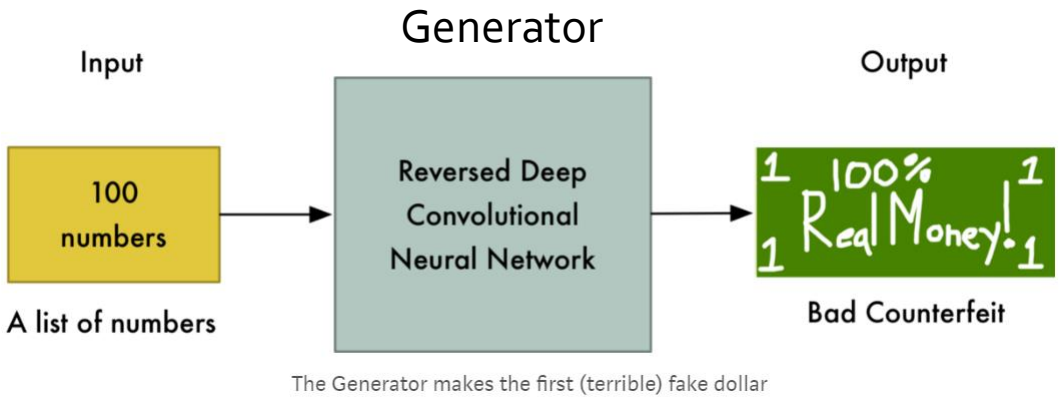
Next Iteration

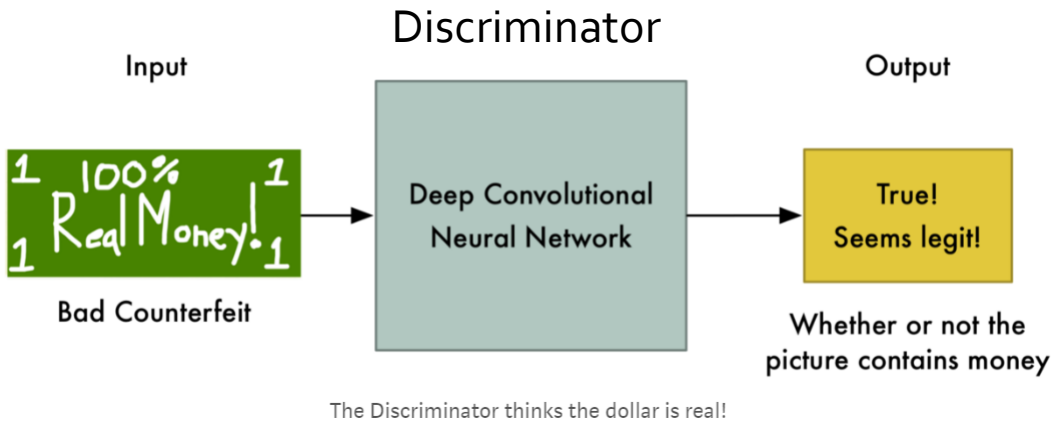
https://www.youtube.com/watch?v=oOxo6RbJ6is

Next Iteration

https://www.ebay.com/bhp/counterfeit-money

Example

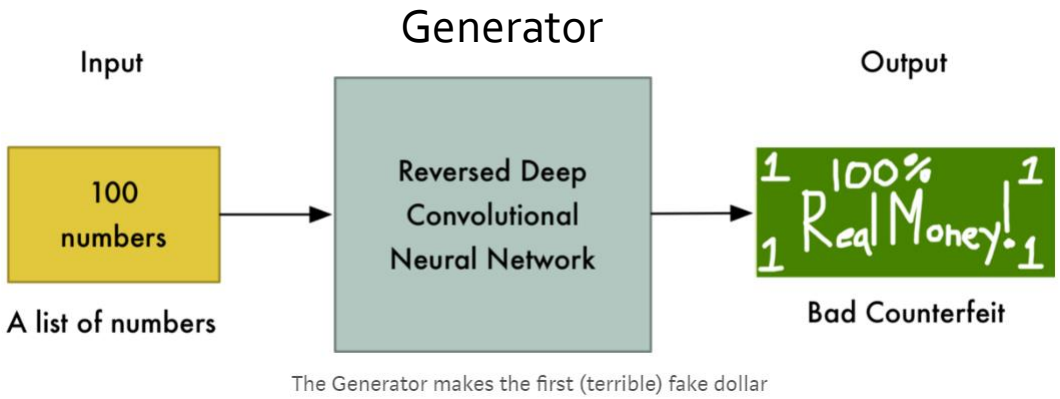




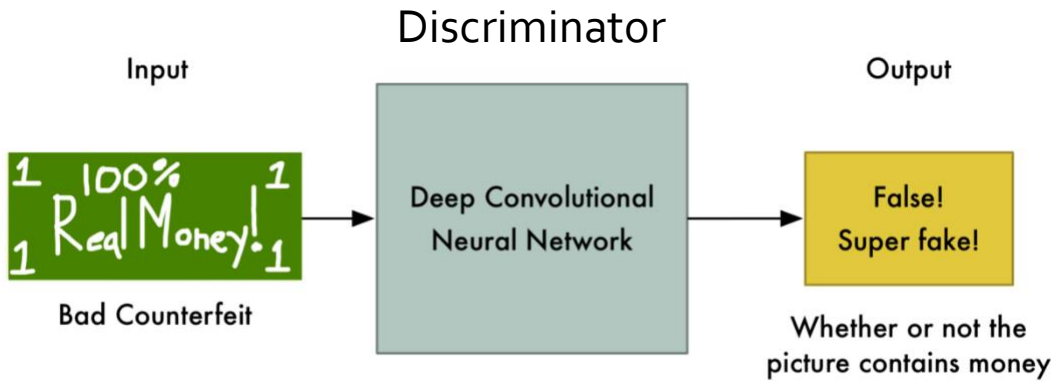
We tell Discriminator that this one is fake, real money has a picture of a person on it and the fake money doesn't.

https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Update Discriminator



The Generator makes the first (terrible) fake dollar

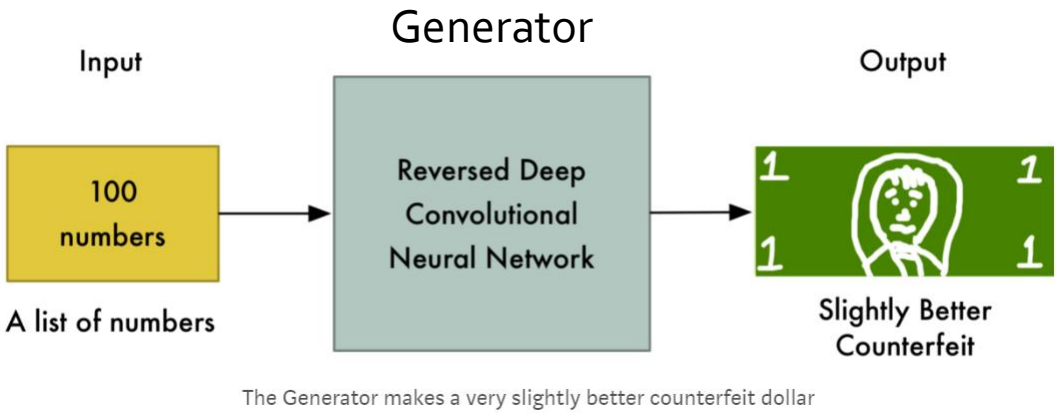


The Discriminator levels up! It now can spot very bad fake dollars.

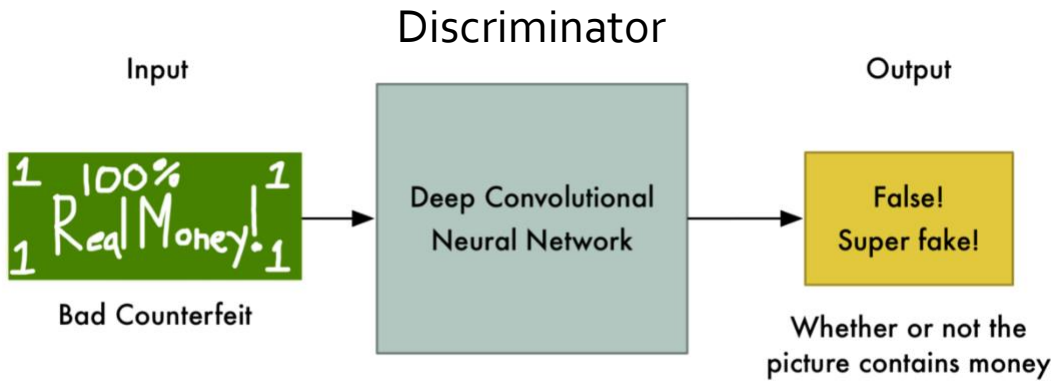
We tell the Generator that it's money images are suddenly getting rejected as fake so it needs to step up it's game. We also tell it that the Discriminator is now looking for faces

https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Update Generator



The Generator makes a very slightly better counterfeit dollar

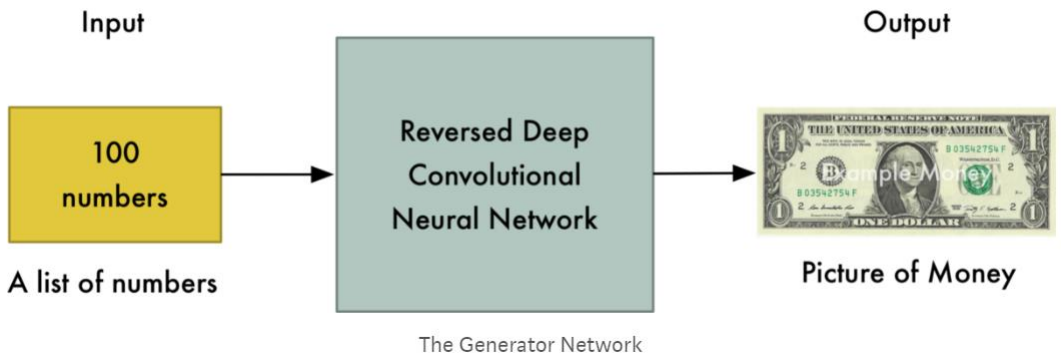


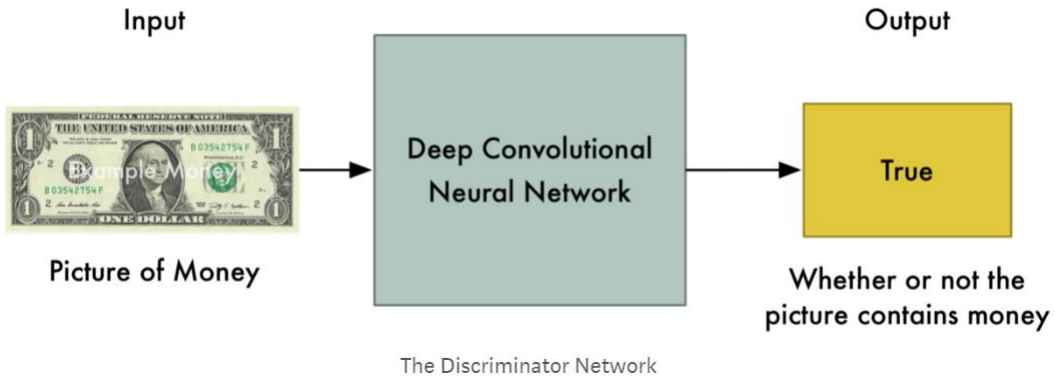
The Discriminator levels up! It now can spot very bad fake dollars.

And the fake bills are being accepted as valid again! So now the Discriminator has to look again at the real dollar and find a new way to tell it apart from the fake one.

https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

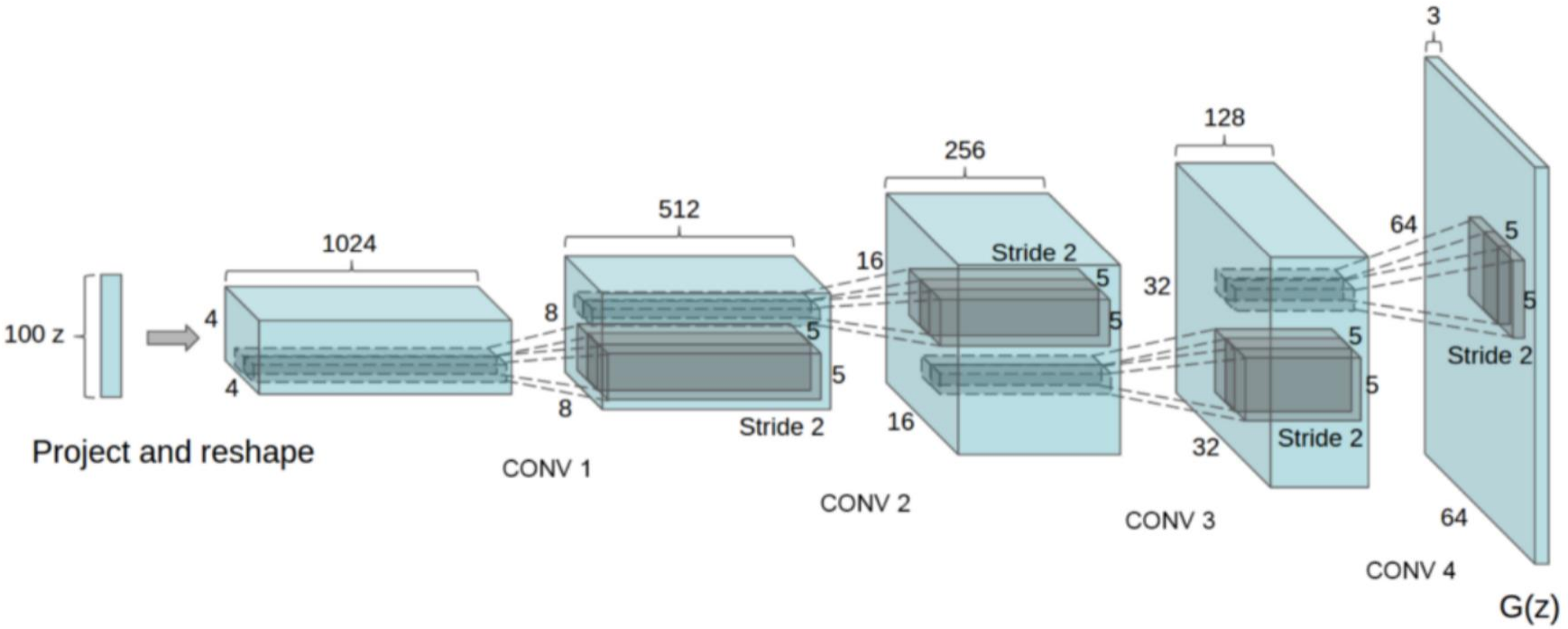
Final Purpose





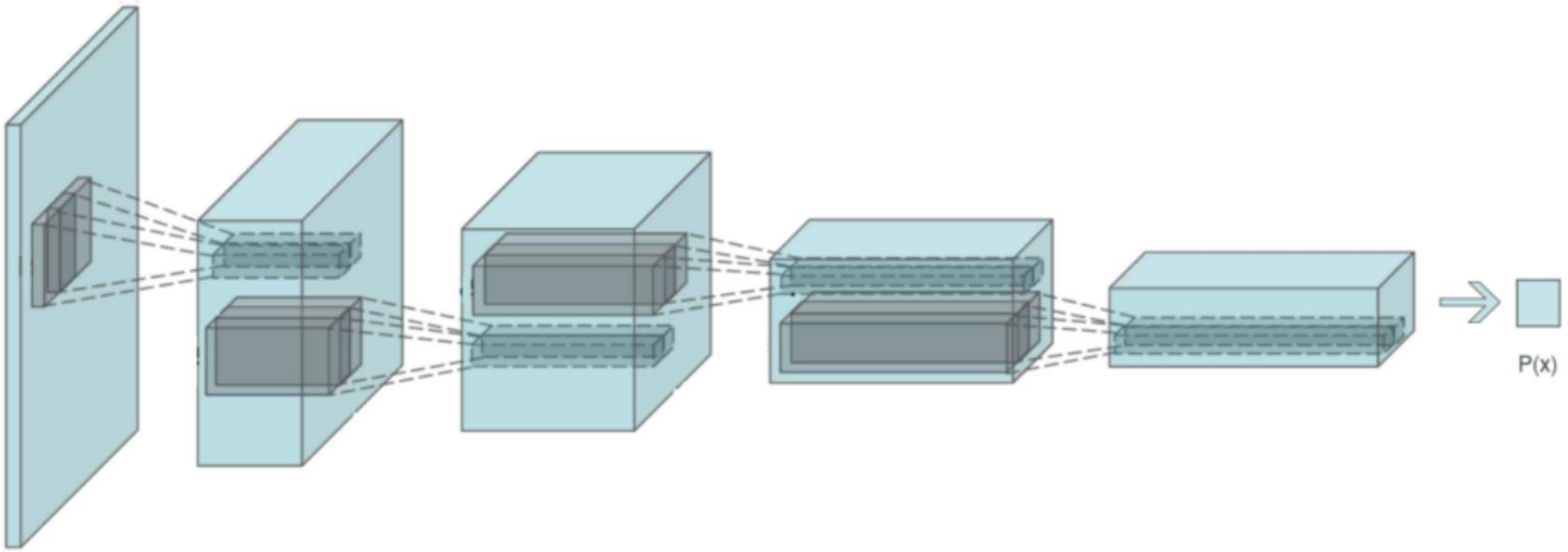
https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Generator



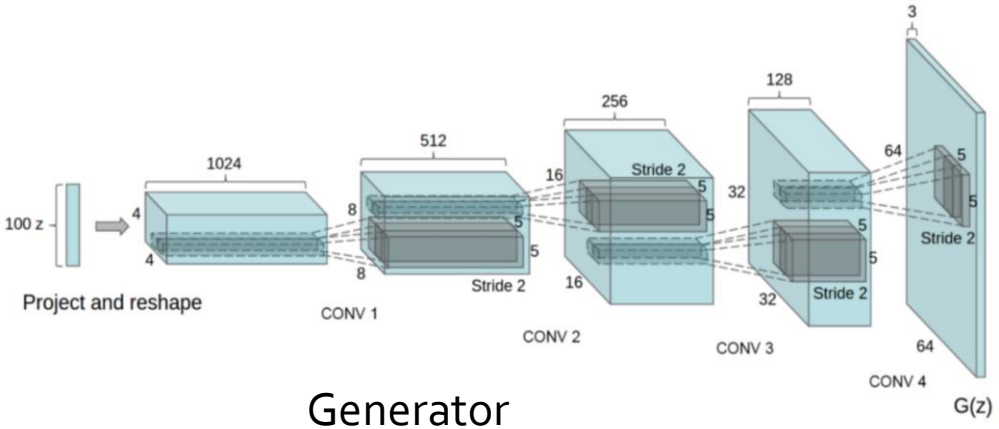
https://arxiv.org/abs/1511.06434

Discriminator



https://arxiv.org/abs/1511.06434

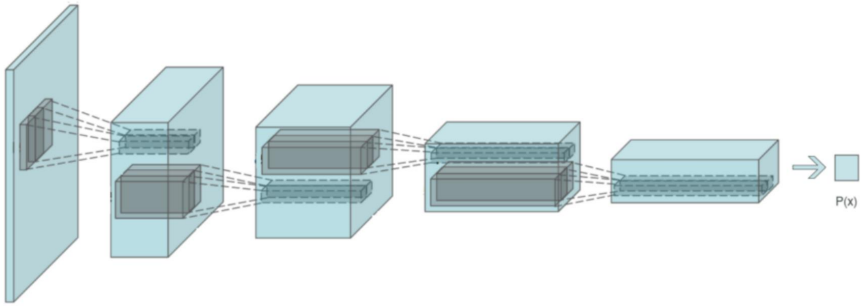
GAN



Generator

Fake

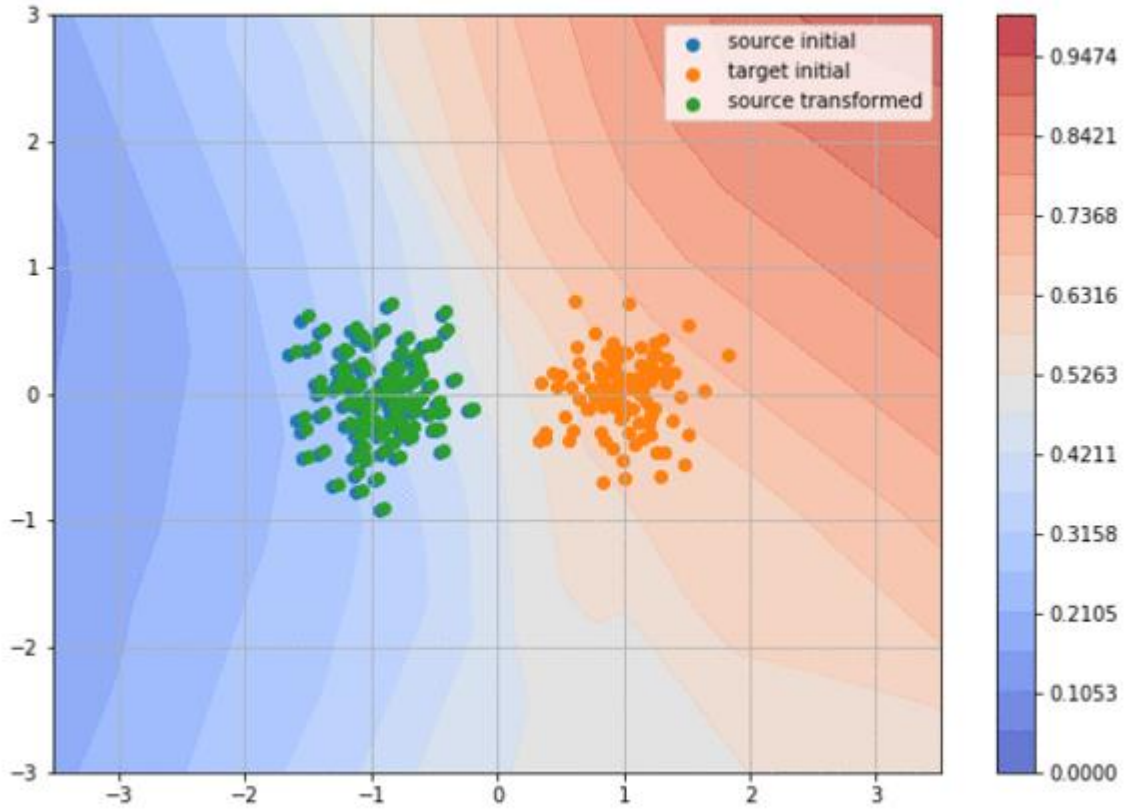
True



Discriminator

https://arxiv.org/abs/1511.06434

Domain Adaptation



GAN-based adversarial domain adaptation for two Gaussian domains. The discriminator (background) tries to separate the green distribution from the orange distribution, and the generator modifies the green distribution to fool the discriminator.

https://artix41.github.io/static/domain-adaptation-in-2017/index.html

VAE vs. GAN

Entropy:
$H(\mathbf{p}) = -\Sigma_i p_i \log_2(p_i)$

Cross-Entropy:
$H(\mathbf{p}, \mathbf{q}) = -\Sigma_i p_i \log_2(q_i)$

KL Divergence:
$D_{KL}(\mathbf{p} \,||\, \mathbf{q}) = H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p})$

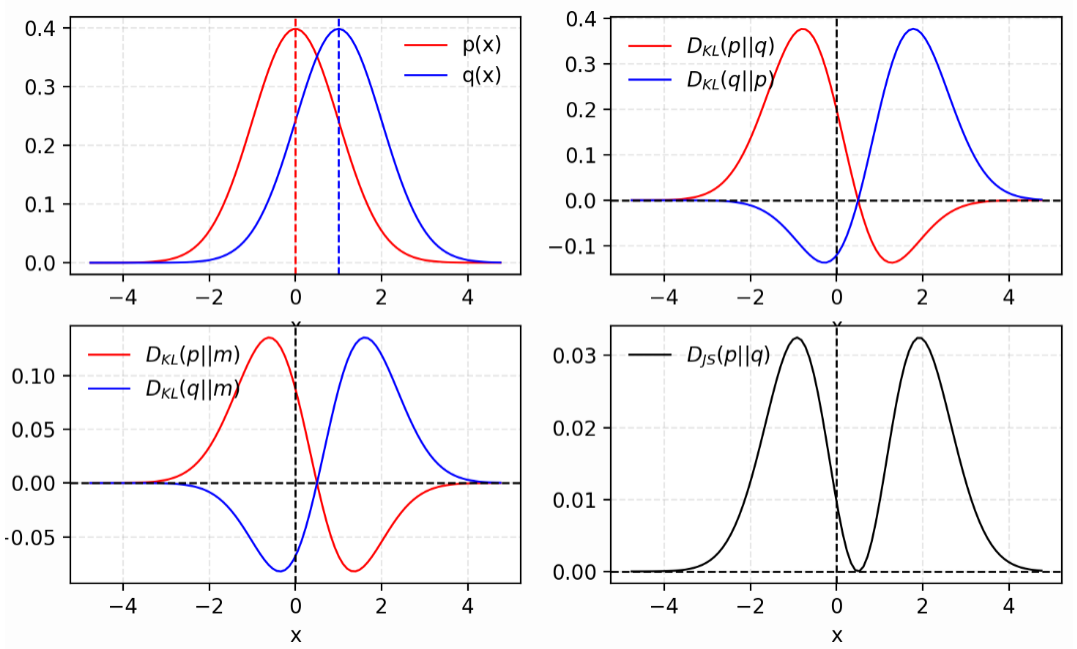
Cross-Entropy = Entropy + KL Divergence

VAE:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

GAN:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

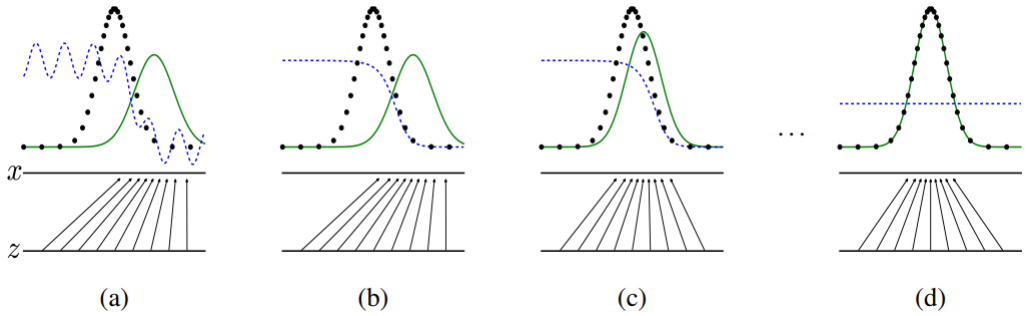


https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
https://www.youtube.com/watch?v=ErfnhcEV1O8

GAN paper

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7



In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

JS Divergence

Entropy:
$H(\mathbf{p}) = -\Sigma_i p_i \log_2(p_i)$

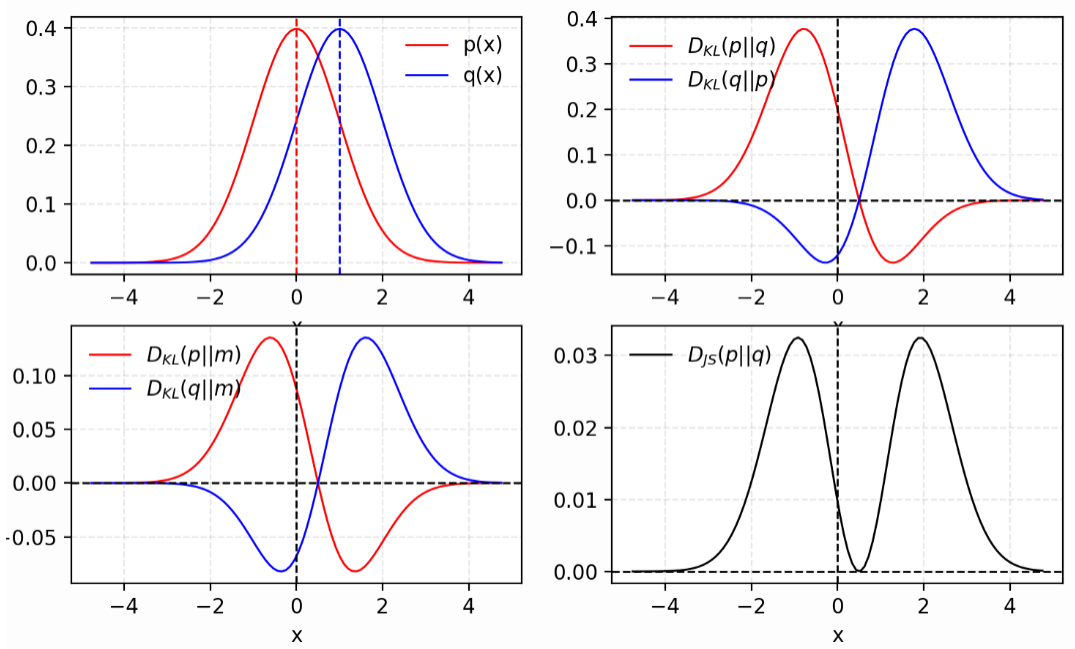
Cross-Entropy:
$H(\mathbf{p}, \mathbf{q}) = -\Sigma_i p_i \log_2(q_i)$

KL Divergence:
$D_{KL}(\mathbf{p} \,||\, \mathbf{q}) = H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p})$

Cross-Entropy = Entropy + KL Divergence

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

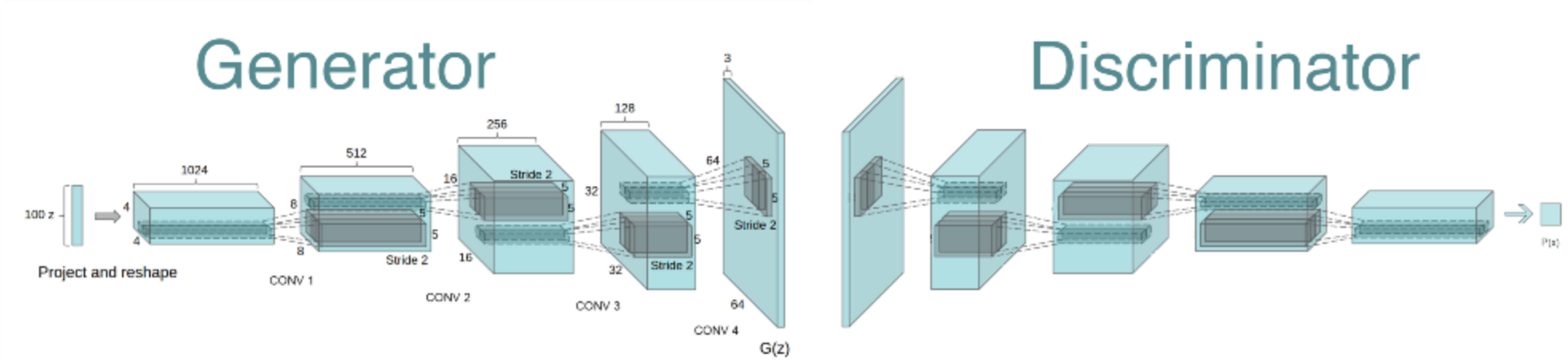
$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2})$$



https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
https://www.youtube.com/watch?v=ErfnhcEV1O8

DCGAN

rings convolution operation into GAN



It gets rid of max-pooling which destroys spatial information and hurts the image quality. Its main design includes: Replace all max pooling with convolutional stride, Use transposed convolution for upsampling, Eliminate fully connected layers, and Use Batch normalization **BN**

https://arxiv.org/abs/1511.06434

WGAN

Instead of adding noise, Wasserstein GAN (WGAN) proposes a new cost function using Wasserstein distance that has a smoother gradient everywhere. WGAN learns no matter the generator is performing or not.

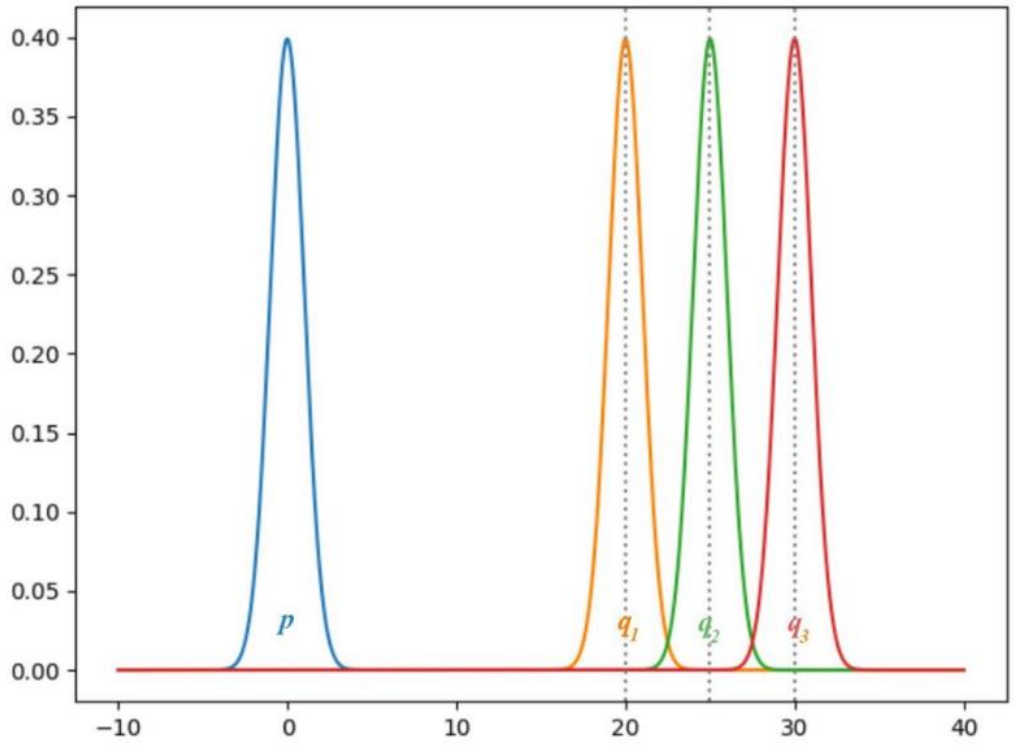
M. Arjovsky, S. Chintala, L. Bottou "Wasserstein GAN" 2016

GAN: minimize Jensen-Shannon divergence between p_X and $p_{G(Z)}$

$$JS(p_X||p_{G(Z)}) = KL(p_X||\frac{p_X + p_{G(Z)}}{2}) + KL(p_{G(Z)}||\frac{p_X + p_{G(Z)}}{2})$$

WGAN: minimize earth mover distance between p_X and $p_{G(Z)}$

$$EM(p_X, p_{G(Z)}) = \inf_{\gamma \in \prod(p_X, p_{G(Z)})} E_{(x,y)\sim\gamma}[||x - y||]$$



https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

LSGAN

More Stable than GAN, converge faster than WGAN

Still use a classifier but replace cross-entropy loss with Euclidean loss.

Discriminator

GAN $\min_D E_{x\sim p_X}[-\log D(x)] + E_{z\sim p_Z}[-\log(1 - D(G(z)))]$

LSGAN $\min_D E_{x\sim p_X}[(D(x) - 1)^2] + E_{z\sim p_Z}[D(G(z))^2]$

Generator

GAN $\min_G E_{z\sim p_Z}[-\log D(G(z))]$

LSGAN $\min_G E_{z\sim p_Z}[(D(G(z)) - 1)^2]$

61

https://www.slideshare.net/mlreview/tutorial-on-theory-and-application-of-generative-adversarial-networks

EBGAN

The diagram of an EBGAN with an auto-encoder discriminator is depicted in figure 1. The generator (G) takes random vector z as input, and transform it into a sample $G(z)$, for example an image. The discriminator (D), whose output is a scalar energy, takes either real or generated images, and estimates the energy value E accordingly, where $E \in \mathbb{R}$.

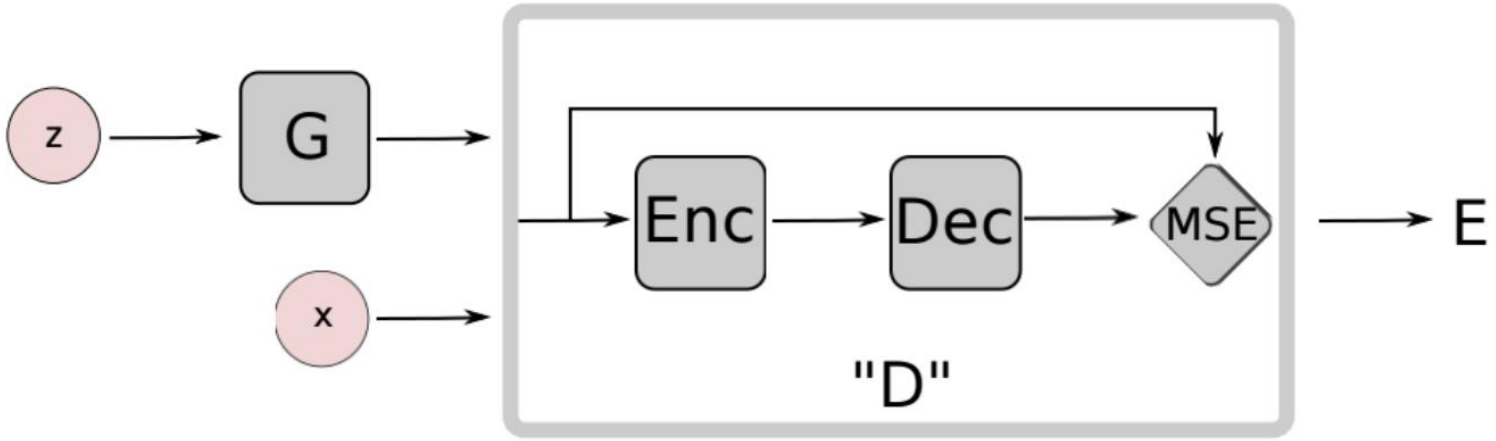
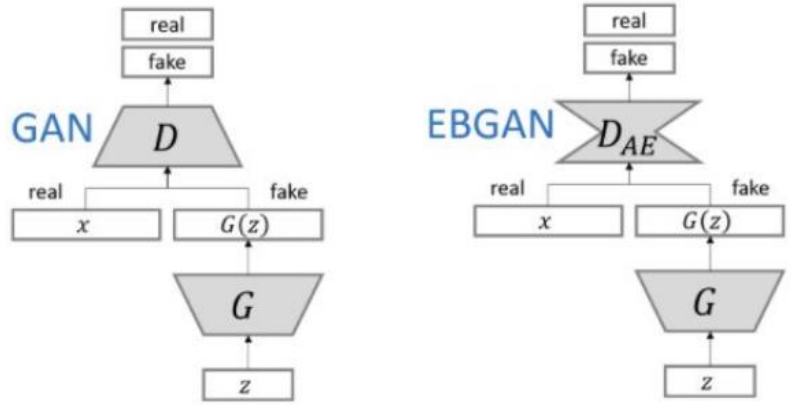
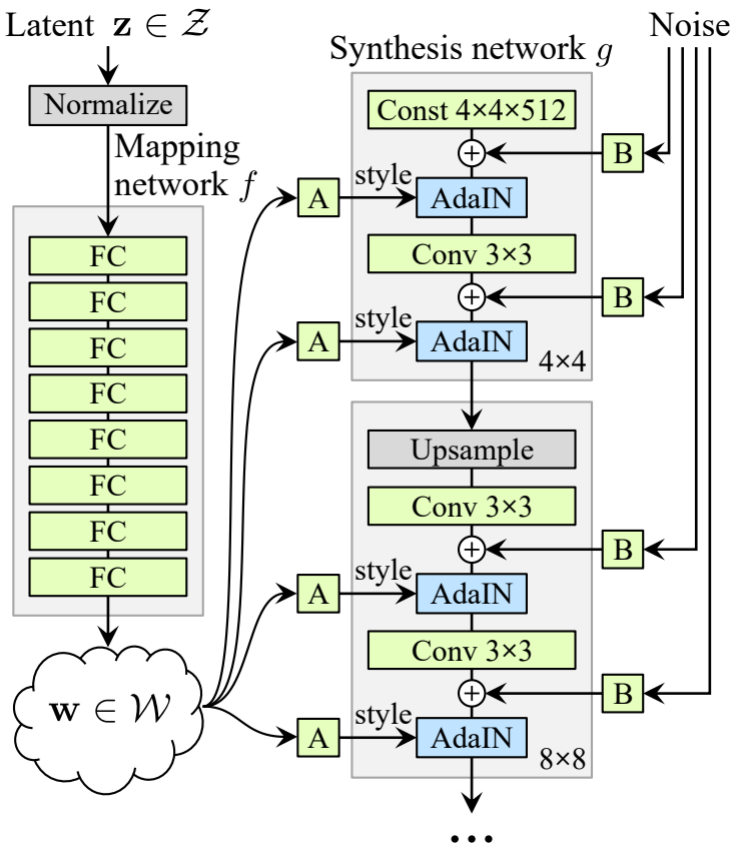


Figure 1: EBGAN architecture.



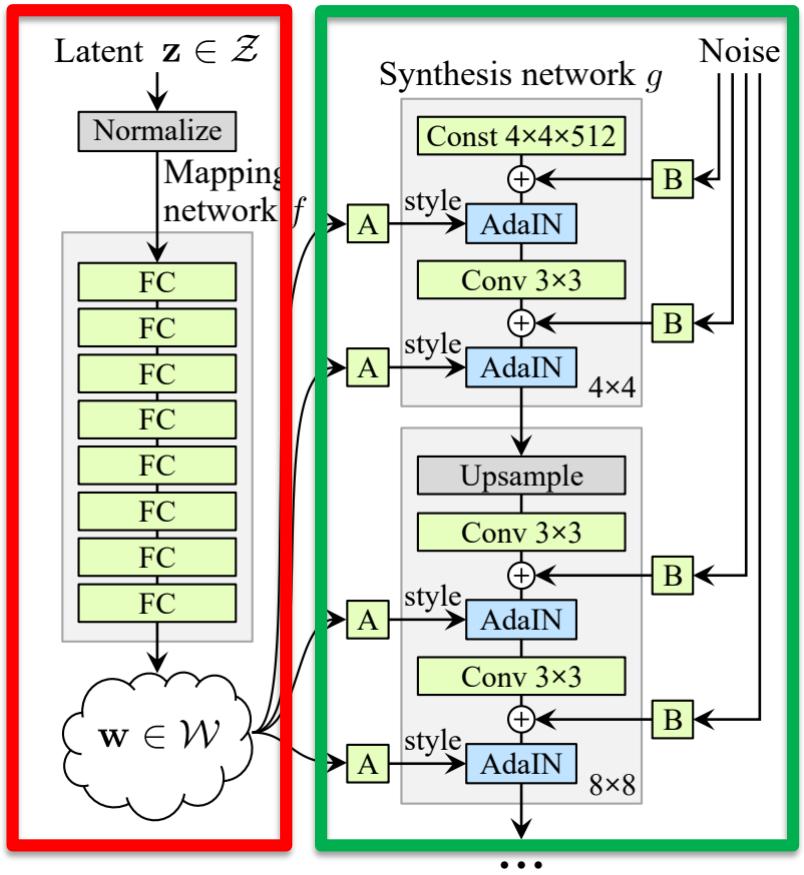
https://arxiv.org/abs/1609.03126

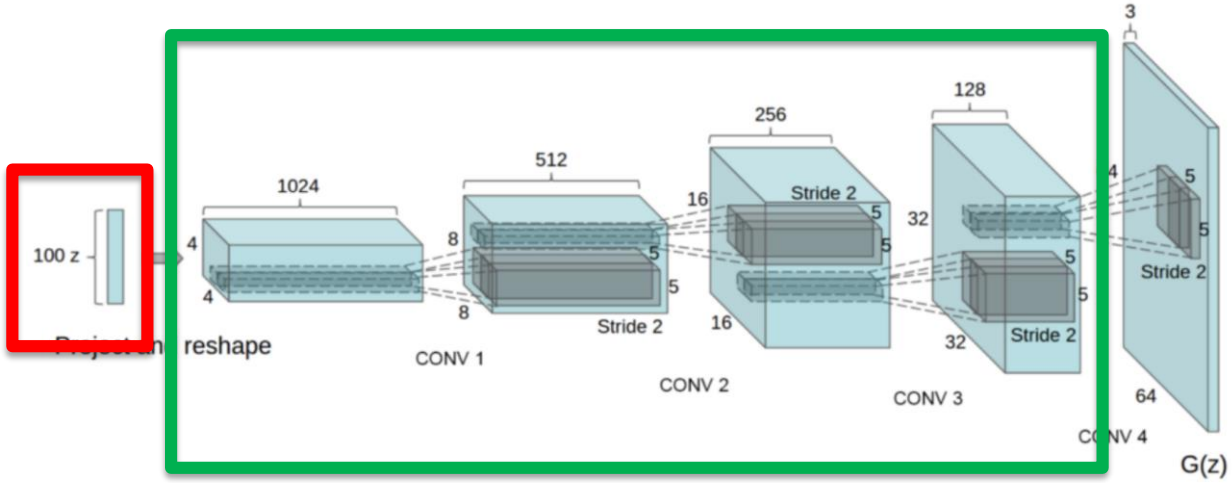
StyleGAN



https://arxiv.org/abs/1812.04948

StyleGAN





https://arxiv.org/abs/1812.04948

Noise to Medical Image

Learning Implicit Brain MRI Manifolds with Deep Learning

Camilo Bermudez*[a], Andrew J. Plassard[b], Larry T. Davis[c], Allen T. Newton[c], Susan M Resnick[b], Bennett A. Landman[a]

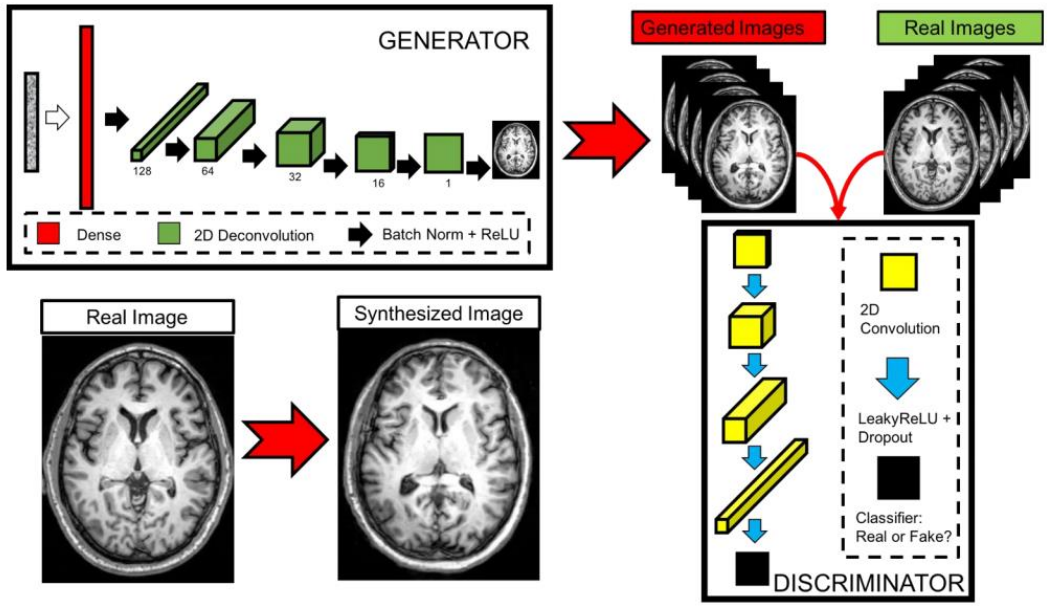


Figure 1. Pipeline and network architecture for Generative Adversarial Network used for image synthesis. The generator is a 2D deconvolutional neural network that takes noise as input and generates a 2D image of the brain. The discriminator is a convolutional neural network that takes real and synthetic images and learns to classify them.

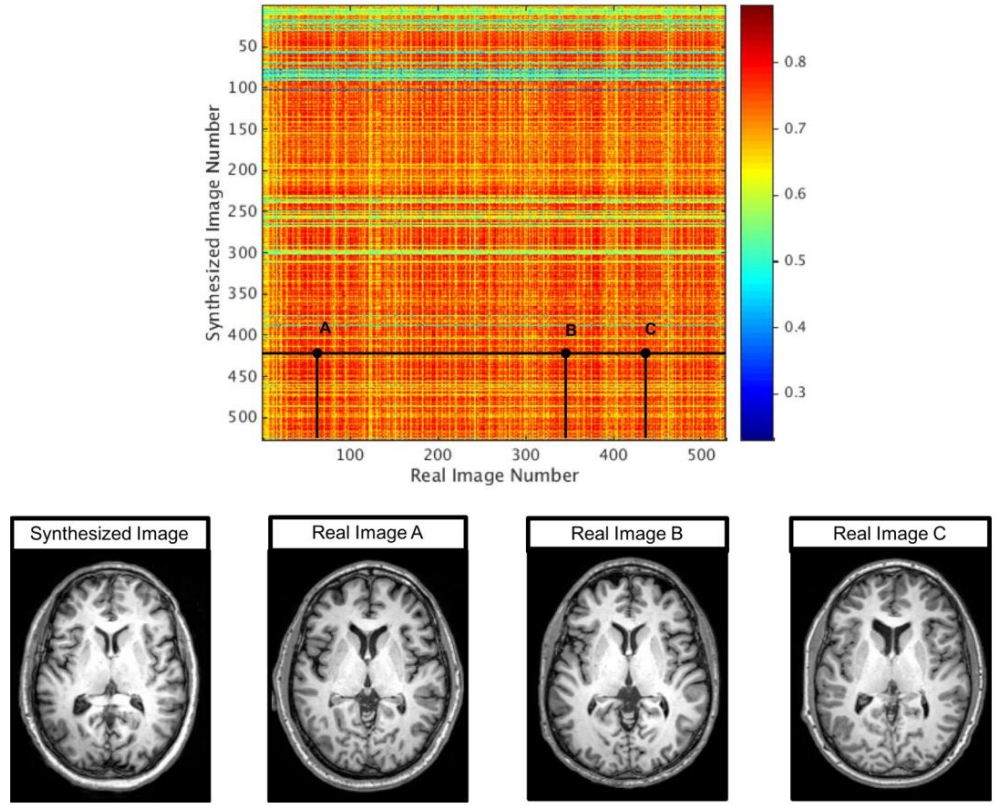
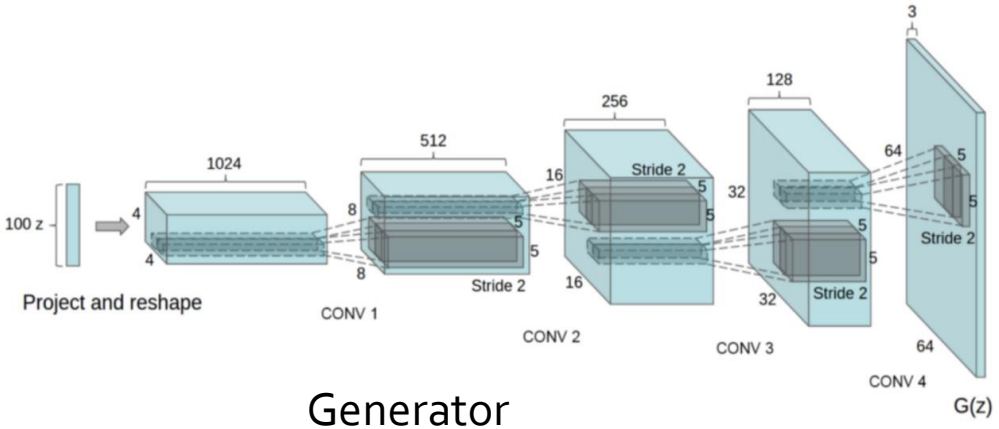


Figure 3. Correlation map between synthesized images and real images. Representative examples a synthetic image (top left) and three real images with the highest correlation values: Real Image A (rho = 0.76), Real Image B (rho = 0.77), and Real Image C (rho = 0.78).

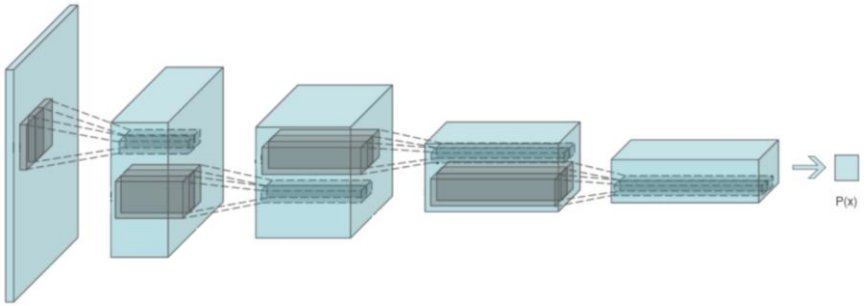
GAN



Generator

Fake

True



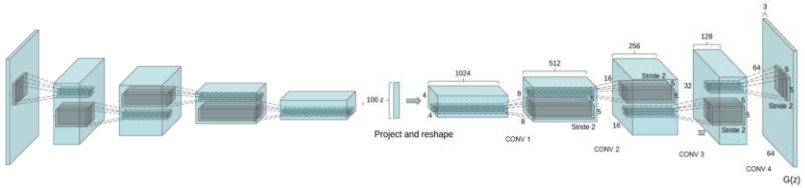
Discriminator

https://arxiv.org/abs/1511.06434

How?

How?

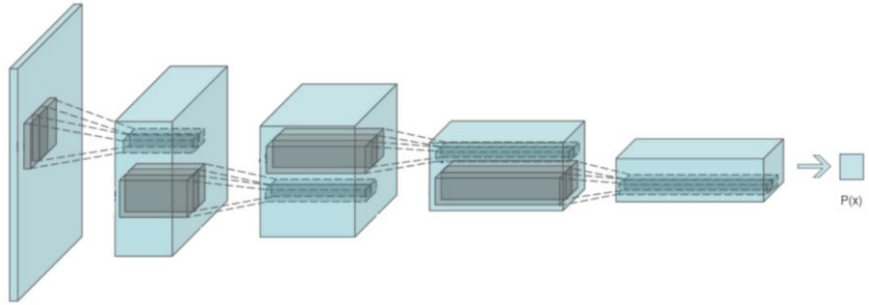
Image to Image GAN (Conditional GAN)



Generator

Fake

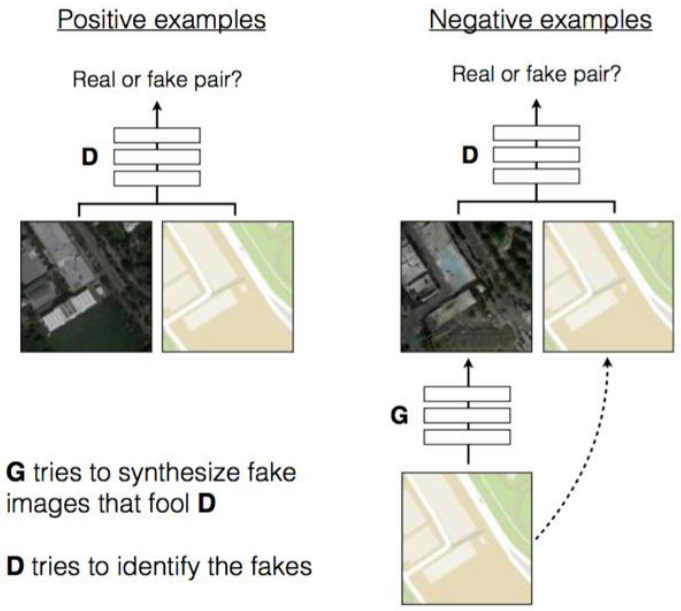
True



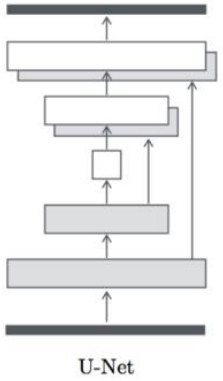
Discriminator

https://arxiv.org/abs/1511.06434

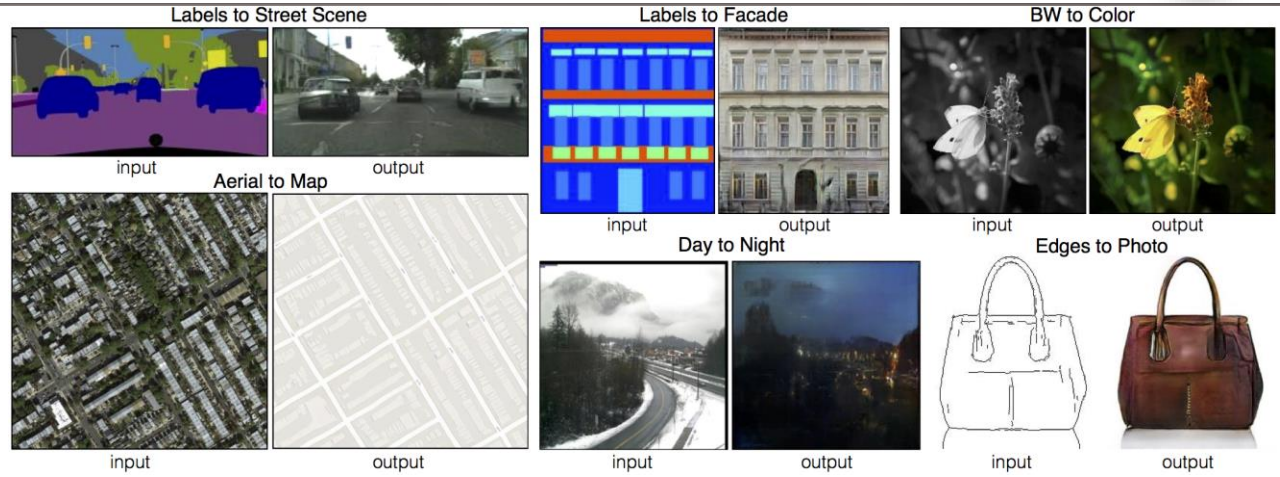
Pix-2-Pix GAN



Discriminator

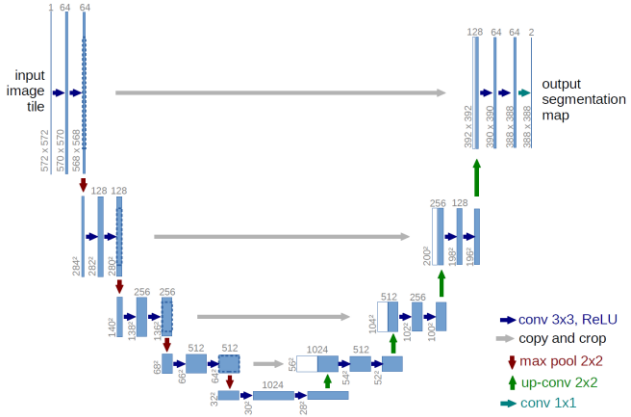


Generator



https://arxiv.org/abs/1611.07004 https://tensorflow.blog/tag/pix2pix/

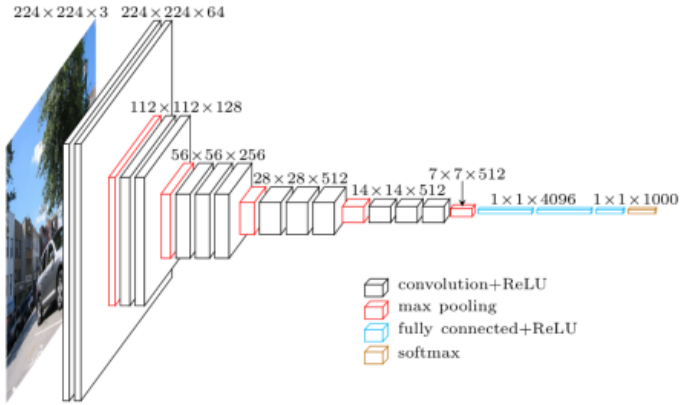
How To



Generator

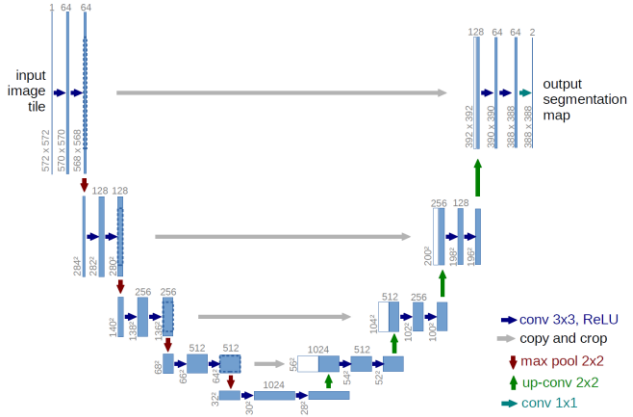
Fake

True



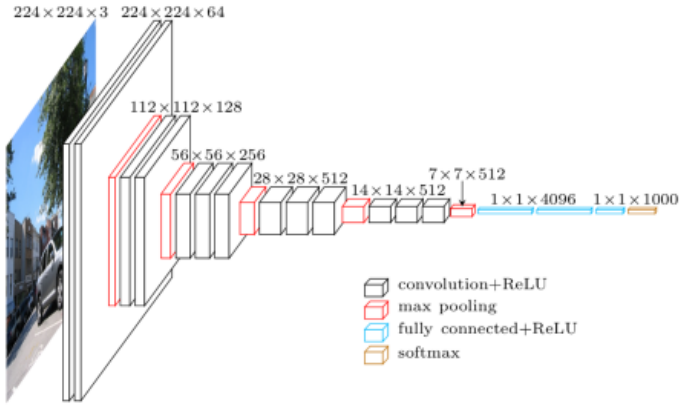
Discriminator

How To



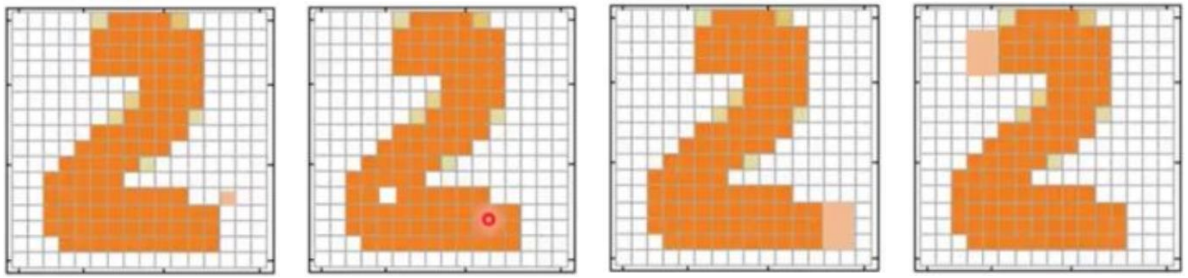
Generator

Fake



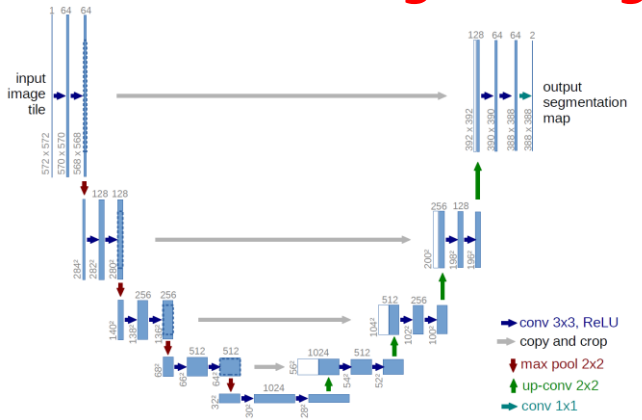
Discriminator

True



How To

Replace Feature Engineering!

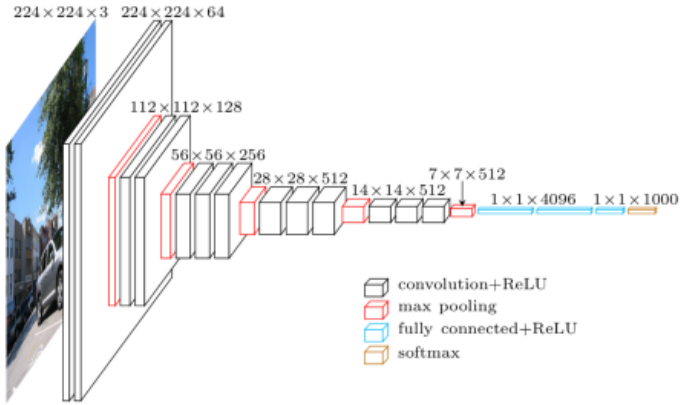


Generator

Fake

True

Replace Loss Engineering!

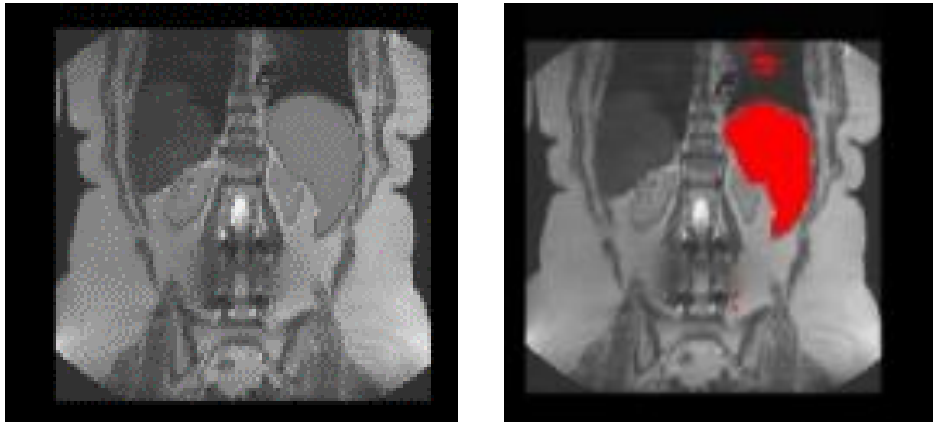


Discriminator

Patch GAN

Isola et al, Archive 2017

Image GAN

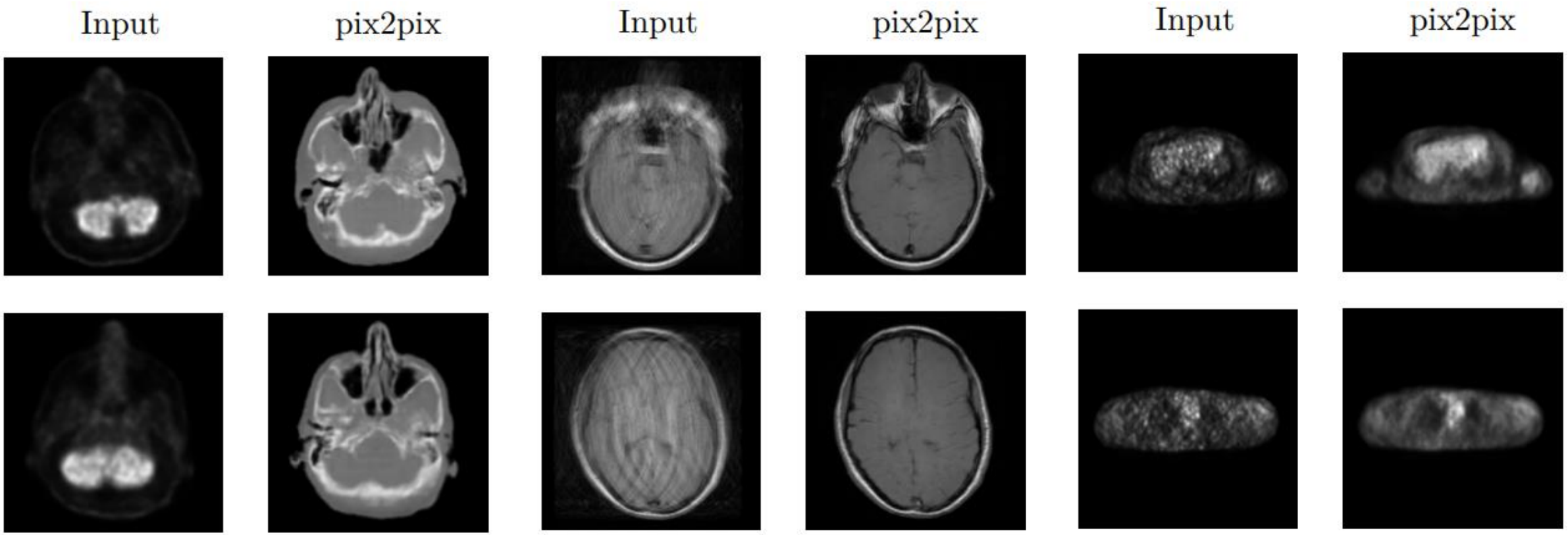


0/1

Patch GAN

0/1	0/1	0/1
0/1	0/1	0/1
0/1	0/1	0/1

Pix2Pix on Medical Image

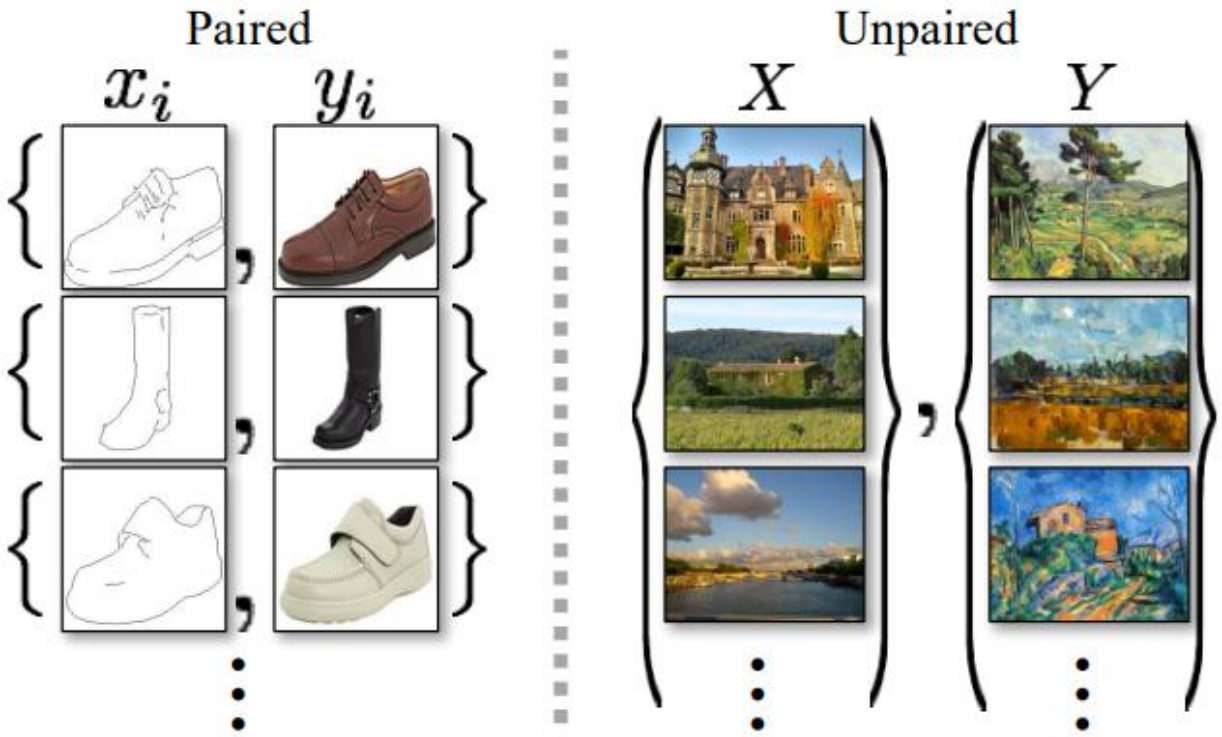


PET-CT translation

MR motion correction

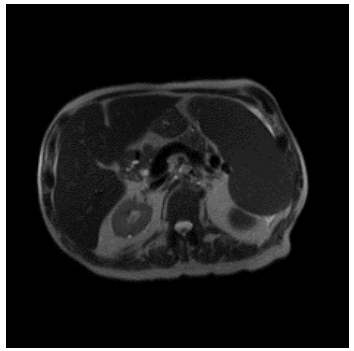
PET denoising

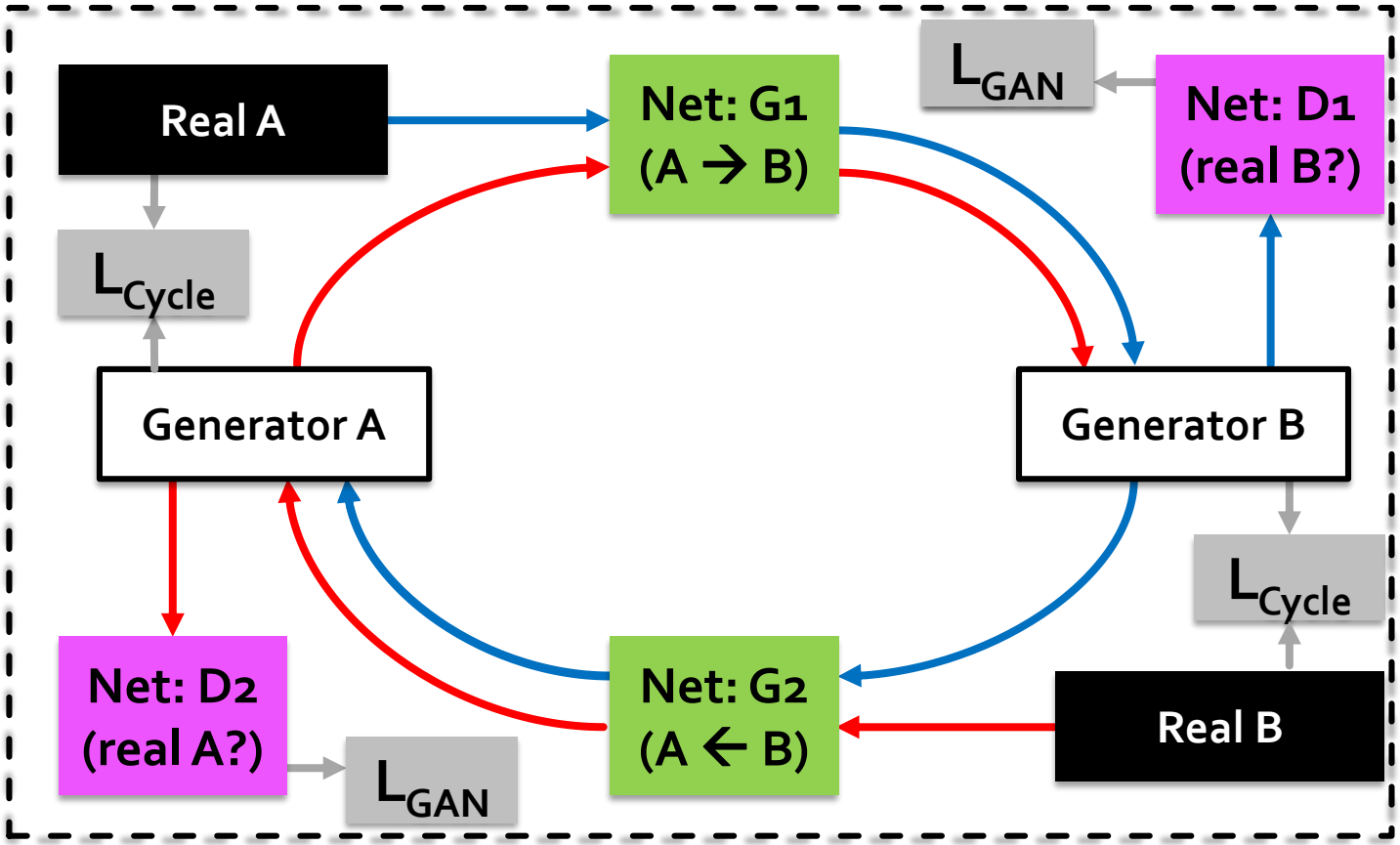
Unpaired Synthesis



https://arxiv.org/pdf/1703.10593.pdf

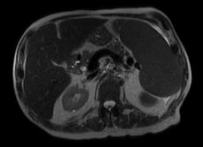
CycleGAN



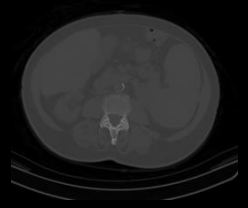


Synthesis

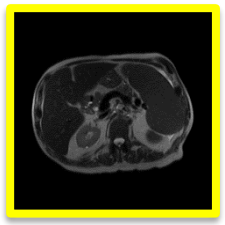
Cycle 1



A: MRI



B: CT



Real A

L_{Cycle}

Generator A

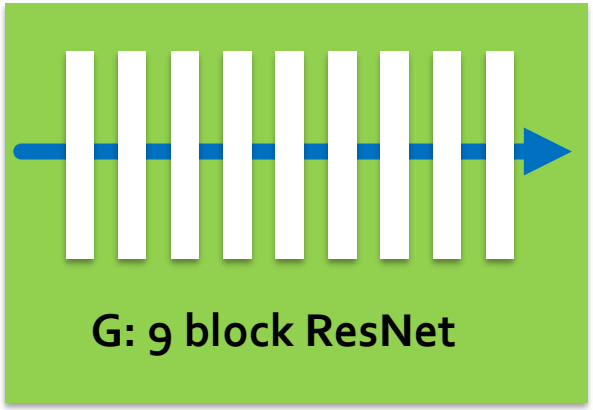
Net: G1 (A → B)

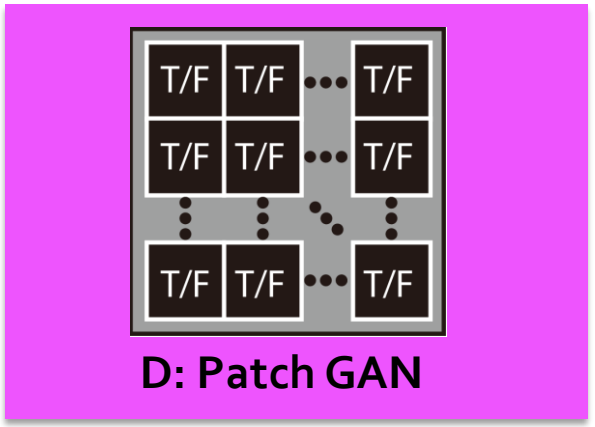
L_{GAN}

Net: D1 (real B?)

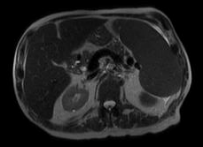
Generator B

Net: G2 (A ← B)

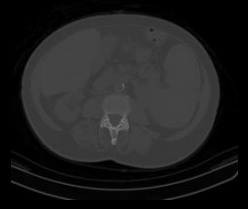




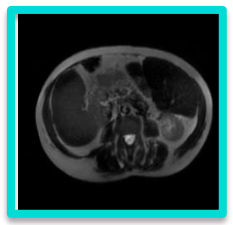
Cycle 2



A: MRI



B: CT



Real A

L_{Cycle}

Generator A

Net: G_1 (A → B)

L_{GAN}

Net: D_1 (real B?)

Generator B

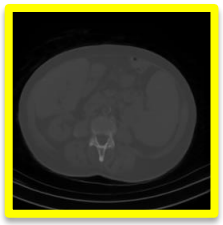
L_{Cycle}

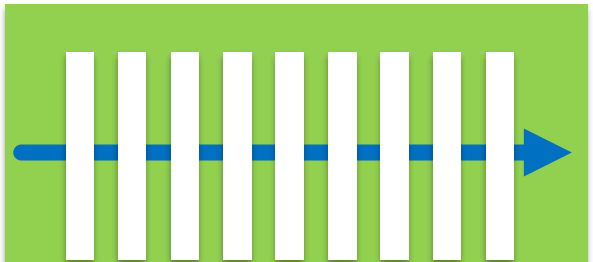
Real B

Net: G_2 (A ← B)

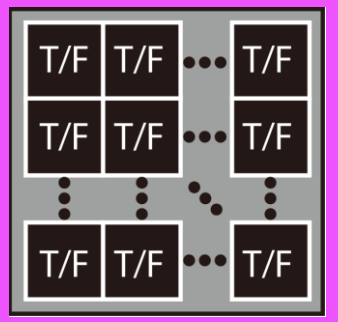
Net: D_2 (real A?)

L_{GAN}



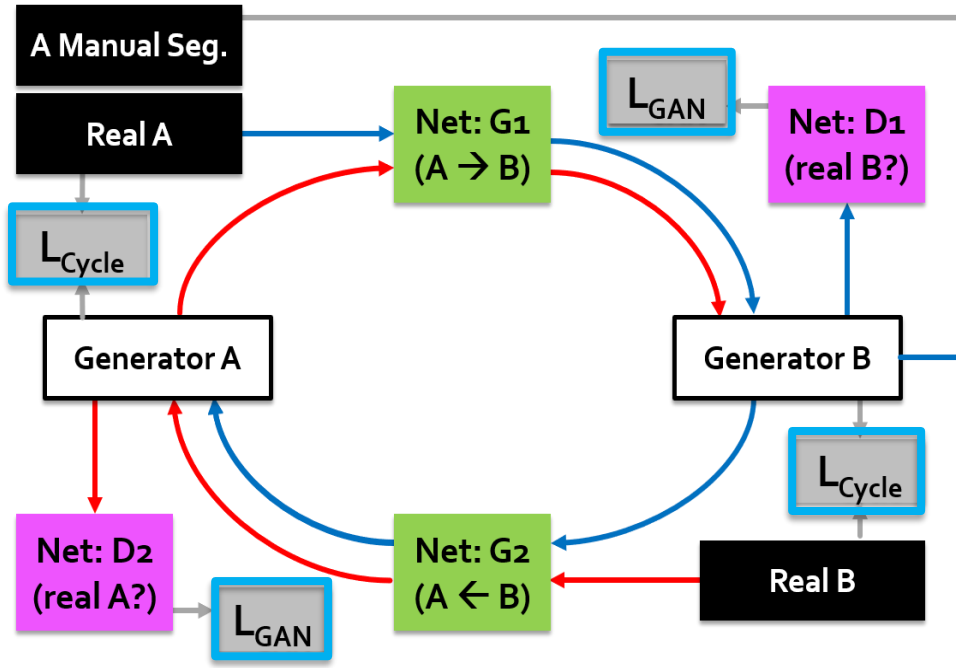


G: 9 block ResNet



D: Patch GAN

Loss Functions



$$\mathcal{L}_{\text{GAN}}(G_1, D_1, A, B) = E_{y\sim B}[\log D_1(y)] + E_{x\sim A}[\log(1 - D_1(G_1(x)))]$$

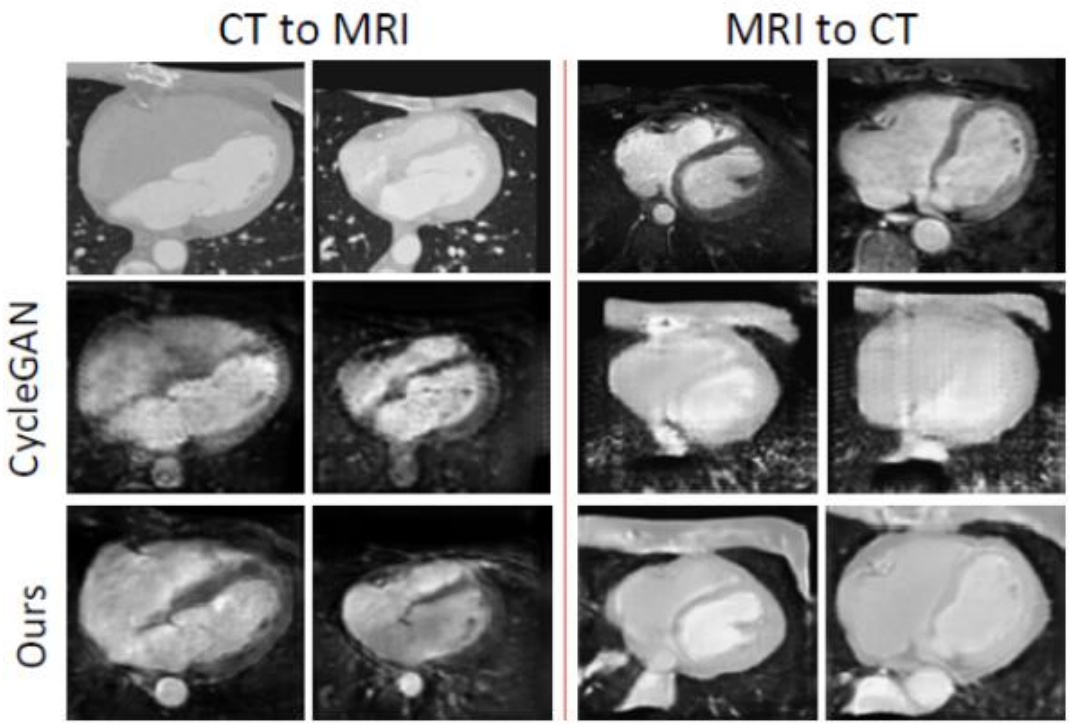
$$\mathcal{L}_{\text{GAN}}(G_2, D_2, B, A) = E_{x\sim A}[\log D_2(x)] + E_{y\sim B}[\log(1 - D_2(G_2(y)))]$$

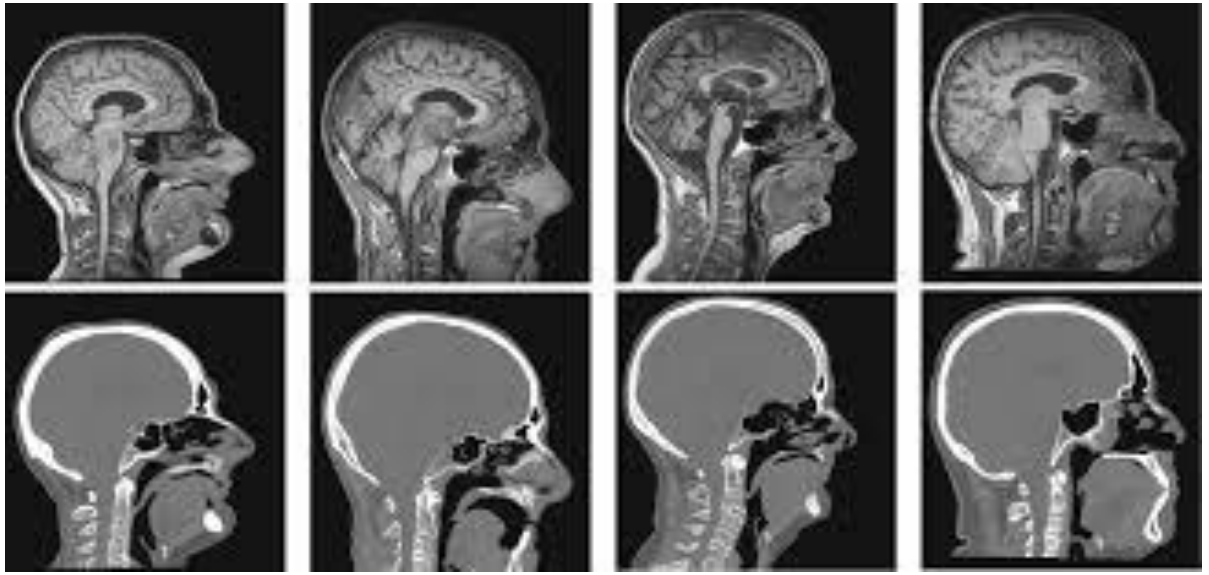
$$\mathcal{L}_{\text{cycle}}(G_1, G_2, A) = E_{x\sim A}[\|G_2(G_1(x)) - x\|_1]$$

$$\mathcal{L}_{\text{cycle}}(G_2, G_1, B) = E_{y\sim B}[\|G_1(G_2(y)) - y\|_1]$$

$$\mathcal{L}_{\text{total}} = \lambda_1 \cdot L_{\text{GAN}}(G_1, D_1, A, B) + \lambda_2 \cdot \mathcal{L}_{\text{GAN}}(G_2, D_2, B, A) + \lambda_3 \cdot \mathcal{L}_{\text{cycle}}(G_1, G_2, A) + \lambda_4 \cdot \mathcal{L}_{\text{cyle}}(G_2, G_1, B)$$

On Medical Image





https://towardsdatascience.com/cvpr-2018-paper-summary-translating-and-segmenting-multimodal-medical-volumes-with-cycle-and-e6381b4a2690

Author

Image-to-Image Translation with Conditional Adversarial Nets

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

Univerity of California, Berkeley
In CVPR 2017

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu* Taesung Park* Phillip Isola Alexei A. Efros

UC Berkeley

In ICCV 2017

Outstanding Doctoral Dissertation Award: Jun-Yan Zhu
SIGGRAPH 2018

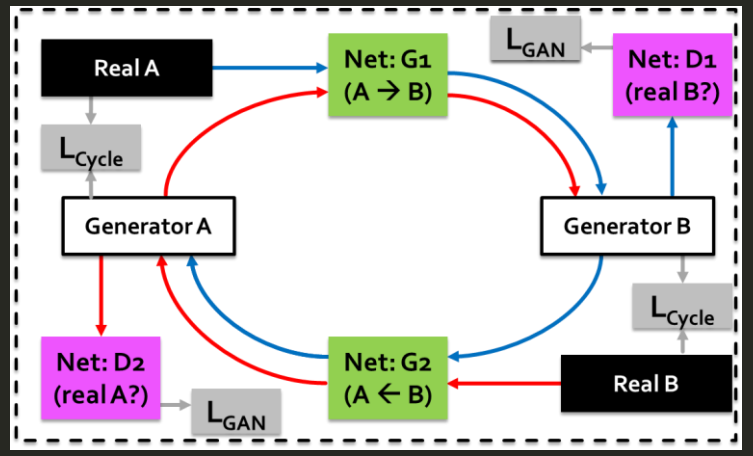
Code

train.py

```
model = create_model(opt)
model.setup(opt)
visualizer = Visualizer(opt)
total_steps = 0

for epoch in range(opt.epoch_count, opt.niter + opt.niter_decay + 1):
    epoch_start_time = time.time()
    iter_data_time = time.time()
    epoch_iter = 0

    for i, data in enumerate(dataset):
        iter_start_time = time.time()
        if total_steps % opt.print_freq == 0:
            t_data = iter_start_time - iter_data_time
        visualizer.reset()
        total_steps += opt.batch_size
        epoch_iter += opt.batch_size
        model.set_input(data)
        model.optimize_parameters()
```

Code

cycle_gan_model.py




```
# load/define networks
# The naming conversion is different from those used in the paper
# Code (paper): G_A (G), G_B (F), D_A (D_Y), D_B (D_X)
self.netG_A = networks.define_G(opt.input_nc, opt.output_nc, opt.ngf, opt.netG, opt.norm,
                                not opt.no_dropout, opt.init_type, opt.init_gain, self.gpu_ids)
self.netG_B = networks.define_G(opt.output_nc, opt.input_nc, opt.ngf, opt.netG, opt.norm,
                                not opt.no_dropout, opt.init_type, opt.init_gain, self.gpu_ids)

if self.isTrain:
    use_sigmoid = opt.no_lsgan
    self.netD_A = networks.define_D(opt.output_nc, opt.ndf, opt.netD,
                                    opt.n_layers_D, opt.norm, use_sigmoid, opt.init_type, opt.init_gain, self.gpu_ids)
    self.netD_B = networks.define_D(opt.input_nc, opt.ndf, opt.netD,
                                    opt.n_layers_D, opt.norm, use_sigmoid, opt.init_type, opt.init_gain, self.gpu_ids)

if self.isTrain:
    self.fake_A_pool = ImagePool(opt.pool_size)
    self.fake_B_pool = ImagePool(opt.pool_size)
    # define loss functions
    self.criterionGAN = networks.GANLoss(use_lsgan=not opt.no_lsgan).to(self.device)
    self.criterionCycle = torch.nn.L1Loss()
    self.criterionIdt = torch.nn.L1Loss()
    # initialize optimizers
    self.optimizer_G = torch.optim.Adam(itertools.chain(self.netG_A.parameters(), self.netG_B.parameters()),
                                        lr=opt.lr, betas=(opt.beta1, 0.999))
    self.optimizer_D = torch.optim.Adam(itertools.chain(self.netD_A.parameters(), self.netD_B.parameters()),
                                        lr=opt.lr, betas=(opt.beta1, 0.999))
    self.optimizers = []
    self.optimizers.append(self.optimizer_G)
    self.optimizers.append(self.optimizer_D)
```

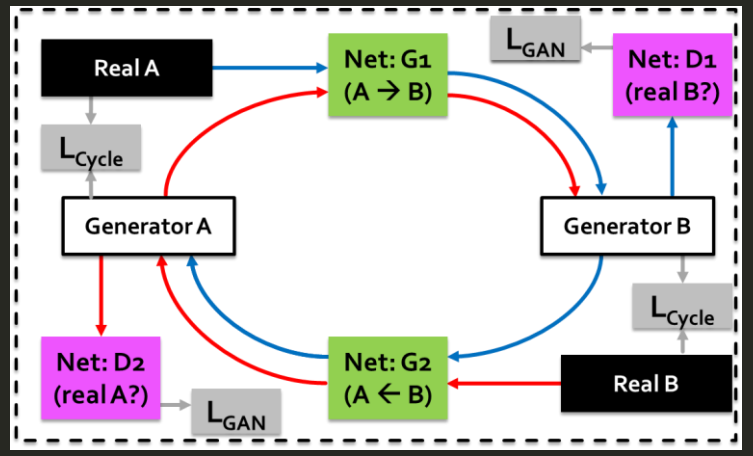
Code

unaligned_dataset.py

```python
def __getitem__(self, index):
    A_path = self.A_paths[index % self.A_size]
    if self.opt.serial_batches:
        index_B = index % self.B_size
    else:
        index_B = random.randint(0, self.B_size - 1)
    B_path = self.B_paths[index_B]
    # print('(A, B) = (%d, %d)' % (index_A, index_B))
    A_img = Image.open(A_path).convert('RGB')
    B_img = Image.open(B_path).convert('RGB')

    A = self.transform(A_img)
    B = self.transform(B_img)
    if self.opt.direction == 'BtoA':
        input_nc = self.opt.output_nc
        output_nc = self.opt.input_nc
    else:
        input_nc = self.opt.input_nc
        output_nc = self.opt.output_nc
```

Code

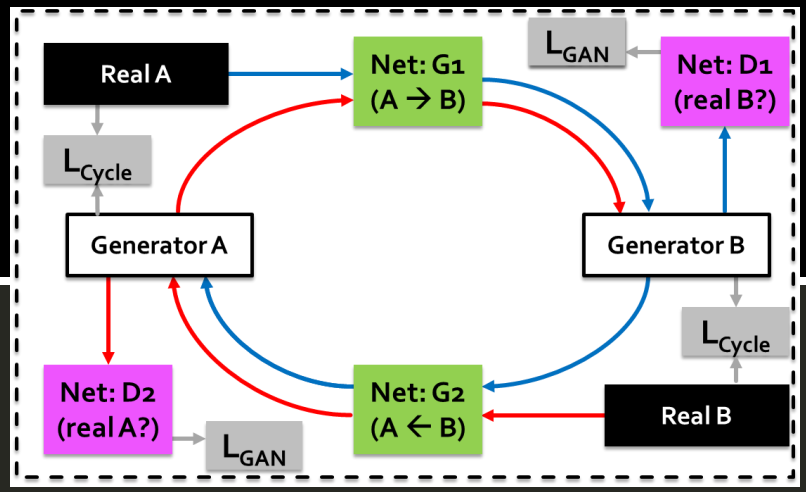
cycle_gan_model.py




```
def optimize_parameters(self):
    # forward
    self.forward()
    # G_A and G_B
    self.set_requires_grad([self.netD_A, self.netD_B], False)
    self.optimizer_G.zero_grad()
    self.backward_G()
    self.optimizer_G.step()
    # D_A and D_B
    self.set_requires_grad([self.netD_A, self.netD_B], True)
    self.optimizer_D.zero_grad()
    self.backward_D_A()
    self.backward_D_B()
    self.optimizer_D.step()

def forward(self):
    self.fake_B = self.netG_A(self.real_A)
    self.rec_A = self.netG_B(self.fake_B)

    self.fake_A = self.netG_B(self.real_B)
    self.rec_B = self.netG_A(self.fake_A)
```

Code

cycle_gan_model.py

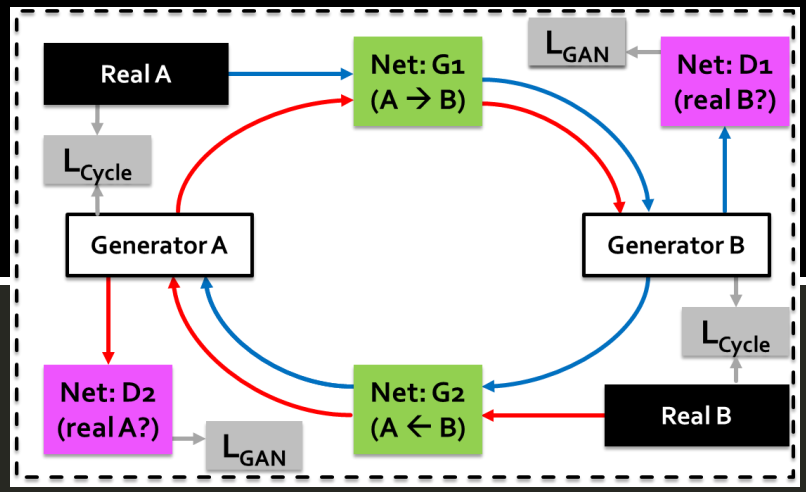



```python
def optimize_parameters(self):
    # forward
    self.forward()
    # G_A and G_B
    self.set_requires_grad([self.netD_A, self.netD_B], False)
    self.optimizer_G.zero_grad()
    self.backward_G()
    self.optimizer_G.step()
    # D_A and D_B
    self.set_requires_grad([self.netD_A, self.netD_B], True)
    self.optimizer_D.zero_grad()
    self.backward_D_A()
    self.backward_D_B()
    self.optimizer_D.step()
```

```python
def backward_G(self):
    lambda_idt = self.opt.lambda_identity
    lambda_A = self.opt.lambda_A
    lambda_B = self.opt.lambda_B

    # GAN loss D_A(G_A(A))
    self.loss_G_A = self.criterionGAN(self.netD_A(self.fake_B), True)
    # GAN loss D_B(G_B(B))
    self.loss_G_B = self.criterionGAN(self.netD_B(self.fake_A), True)
    # Forward cycle loss
    self.loss_cycle_A = self.criterionCycle(self.rec_A, self.real_A) * lambda_A
    # Backward cycle loss
    self.loss_cycle_B = self.criterionCycle(self.rec_B, self.real_B) * lambda_B
    # combined loss
    self.loss_G = self.loss_G_A + self.loss_G_B + self.loss_cycle_A +
        self.loss_cycle_B + self.loss_idt_A + self.loss_idt_B

    self.loss_G.backward()
```

Code

cycle_gan_model.py




```
def optimize_parameters(self):
    # forward
    self.forward()
    # G_A and G_B
    self.set_requires_grad([self.netD_A, self.netD_B], False)
    self.optimizer_G.zero_grad()
    self.backward_G()
    self.optimizer_G.step()
    # D_A and D_B
    self.set_requires_grad([self.netD_A, self.netD_B], True)
    self.optimizer_D.zero_grad()
    self.backward_D_A()
    self.backward_D_B()
    self.optimizer_D.step()
```

```
def backward_D_basic(self, netD, real, fake):
    # Real
    pred_real = netD(real)
    loss_D_real = self.criterionGAN(pred_real, True)
    # Fake
    pred_fake = netD(fake.detach())
    loss_D_fake = self.criterionGAN(pred_fake, False)
    # Combined loss
    loss_D = (loss_D_real + loss_D_fake) * 0.5
    # backward
    loss_D.backward()
    return loss_D

def backward_D_A(self):
    fake_B = self.fake_B_pool.query(self.fake_B)
    self.loss_D_A = self.backward_D_basic(self.netD_A, self.real_B, fake_B)

def backward_D_B(self):
    fake_A = self.fake_A_pool.query(self.fake_A)
    self.loss_D_B = self.backward_D_basic(self.netD_B, self.real_A, fake_A)
```