Presentation (Design Project)

Applications of GAN and it's variants

Presented By:

- Rigvita Sharma: 2016A7PS0067P

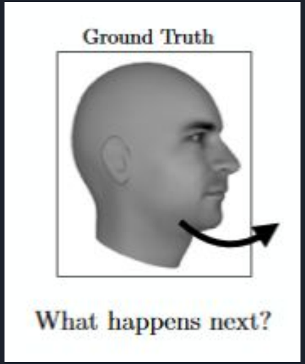
GAN

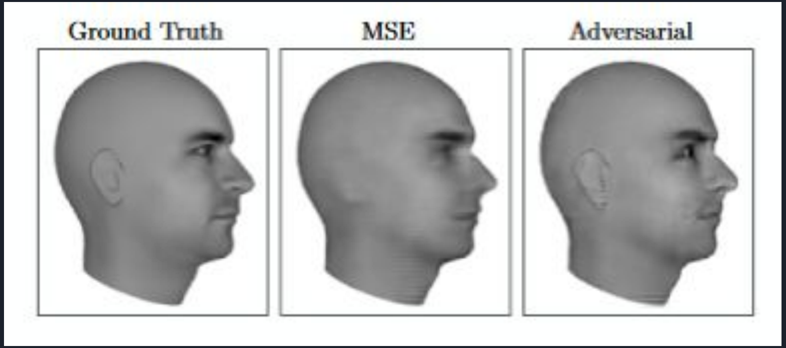
- First introduced in 2014 by Ian Goodfellow.
- Have been used to generate image synthesis, semantic image editing, style transfer, image super-resolution and classification.
- The GAN framework is a powerful way to do unsupervised learning.

*"Generative Adversarial Networks is the **most interesting idea in the last 10 years** in machine learning." - Yann Lecun, Director, Facebook AI*

Applications

Predicting the next frame in a video (Lotter et al 2015)





- Here the model is trained to predict the next frame in video sequence.
- Video depicts movement of 3D model of head.
- The left image depicts the actual frame and hence the ideal result.
- The center image is the result of training the model using mean squared error between the actual next frame and predicted next frame. (Taking average over multiple possible outputs to produce a single output)
- The right image uses GAN loss to produce a sharp and recognizable image as compared to center image where features are blurred.

Applications

Increasing Resolution of an image (Ledig et al. 2016)

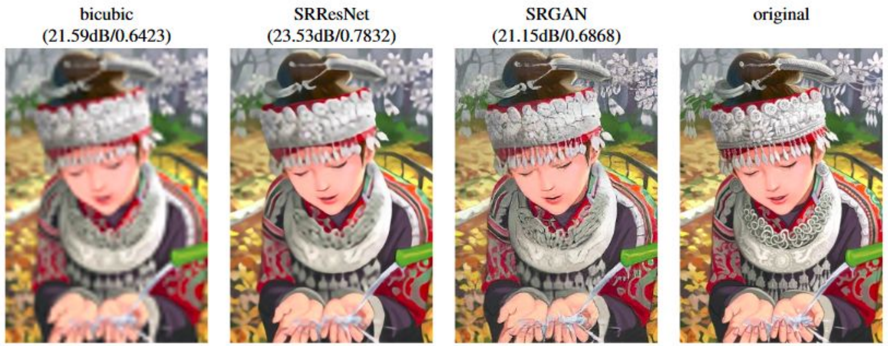
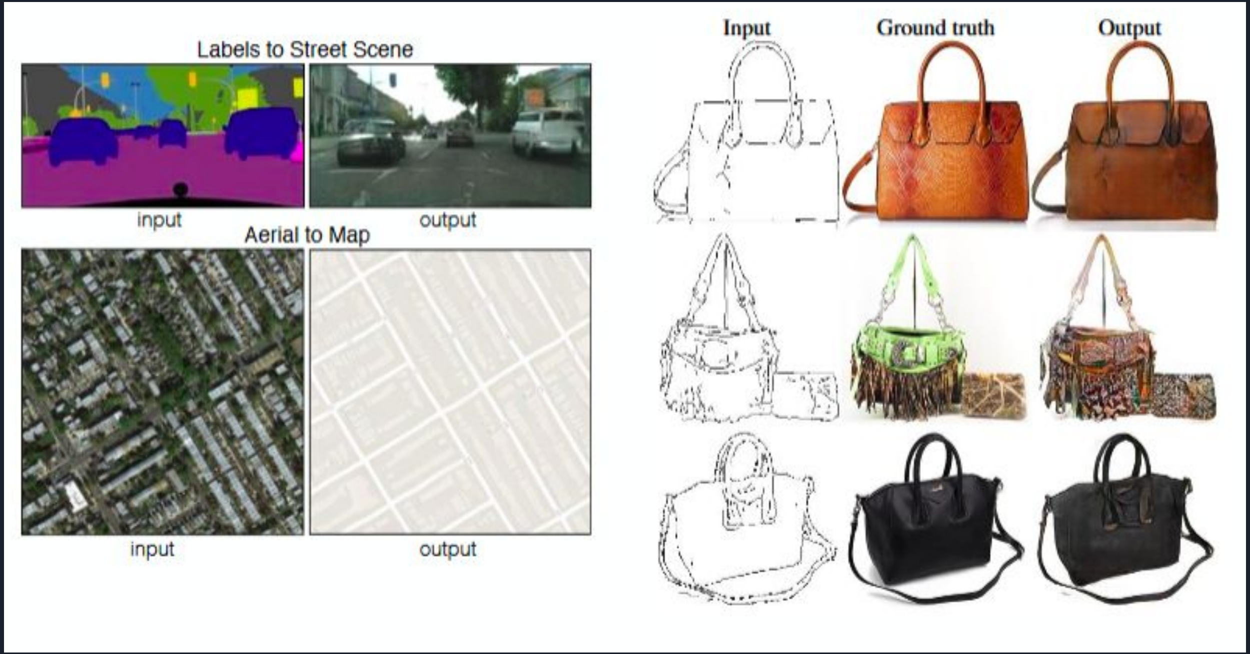


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

- The goal is to take a low resolution image and synthesize a high resolution equivalent.
- The paper shows the importance of using generative models to generate realistic images.
- The original image is downsampled to create a low resolution image.
- Bicubic method is simple interpolation method which doesn't use any statistics of the image.
- SRResNet is neural network trained with mean square error.
- SRGAN is GAN based neural network which improves over SRResNet.

Applications

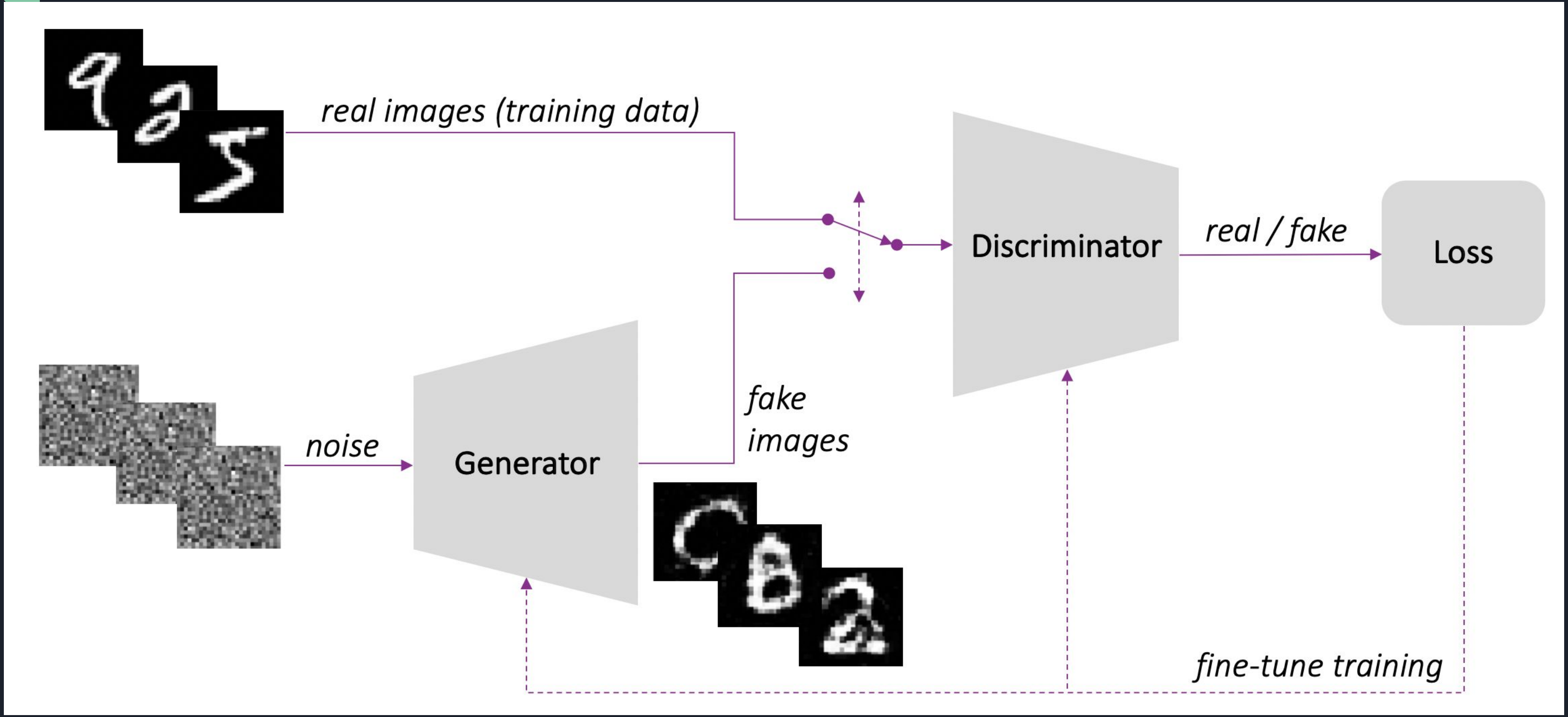
Image to Image Translation (Isola et al. 2016)



GAN

- GAN is a minmax game with two neural net components. First net(Generator G(z)) generates data and other net(Discriminator D(x)) tries to tell the difference between real data and fake data generated by first net.
- A common analogy is to think of one network as an art forger, and the other as an art expert. The forger (generator G), creates forgeries, with the aim of making realistic images. The expert (discriminator D), receives both forgeries and real (authentic) images, and aims to tell them apart. Both are trained simultaneously, and in competition with each other.

Architecture (Vanilla GAN)



Training Procedure

- Feed noise z from random distribution to Generator G to produce fake sample G(z).
- We feed the fake sample G(z) and real data x to discriminator alternatively.
- The discriminator is binary classification which calculates loss for both fake sample and real data. It finally calculate the D loss by summing both the real_data and fake_data loss.
- The generator also calculates its loss from the noise.
- Parameters are adjusted for both the networks in backpropagation step.
- Optimization algorithm is applied.
- Repeat the process for certain number of epochs.

Training

- Objective Function

$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- The first term is entropy that the data from real distribution (pdata(x)) passes through the discriminator. The discriminator tries to maximize this to 1.
- The second term is entropy that the data from random input (p(z)) passes through the generator, which then generates a fake sample which is then passed through the discriminator to identify the fakeness. Discriminator tries to maximize it to 0 .
- So overall, the discriminator is trying to maximize our function V.
- While the generator tries to minimize the function V.

Pdata(x) -> the distribution of real data
X -> sample from pdata(x)
P(z) -> distribution of generator
Z -> sample from p(z)
G(z) -> Generator Network
D(x) -> Discriminator Network

Training Procedure

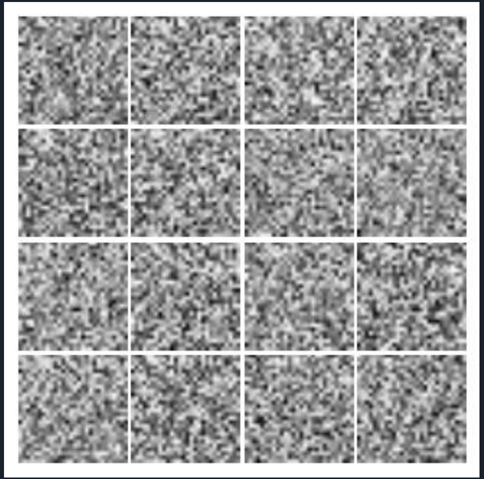
- Two simultaneous SGD on two minibatches -
  - A minibatch of training examples
  - A minibatch of generated samples
- Cost Functions:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

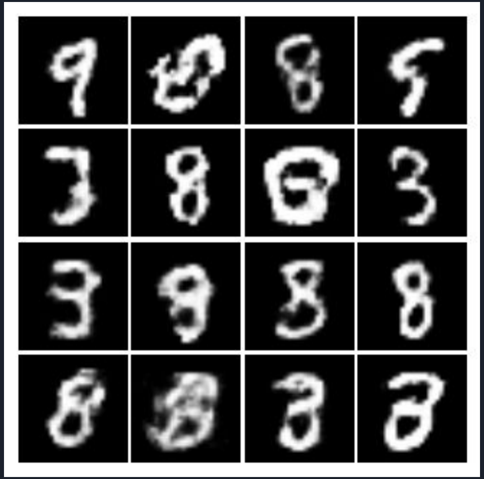
$$J^{(G)} = -J^{(D)}$$

- First term in J(D) represents feeding the actual data to the discriminator. The discriminator would want to maximize the log probability of predicting one, indicating that the data is real.
- The second term represents the samples generated by G. Here, the discriminator would want to maximize the log probability of predicting zero, indicating the the data is fake.
- The generator, on the other hand tries to minimize the log probability of the discriminator being correct.

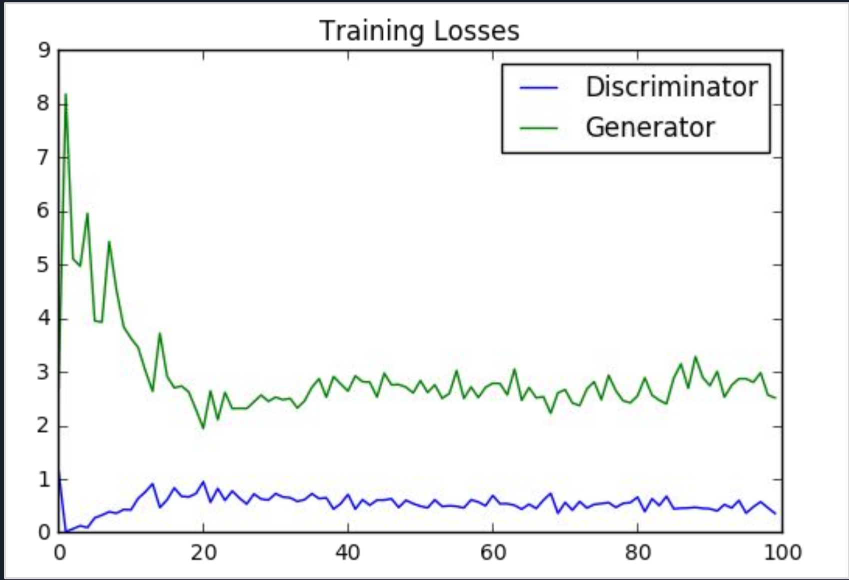
Results from GAN trained



Input Noise

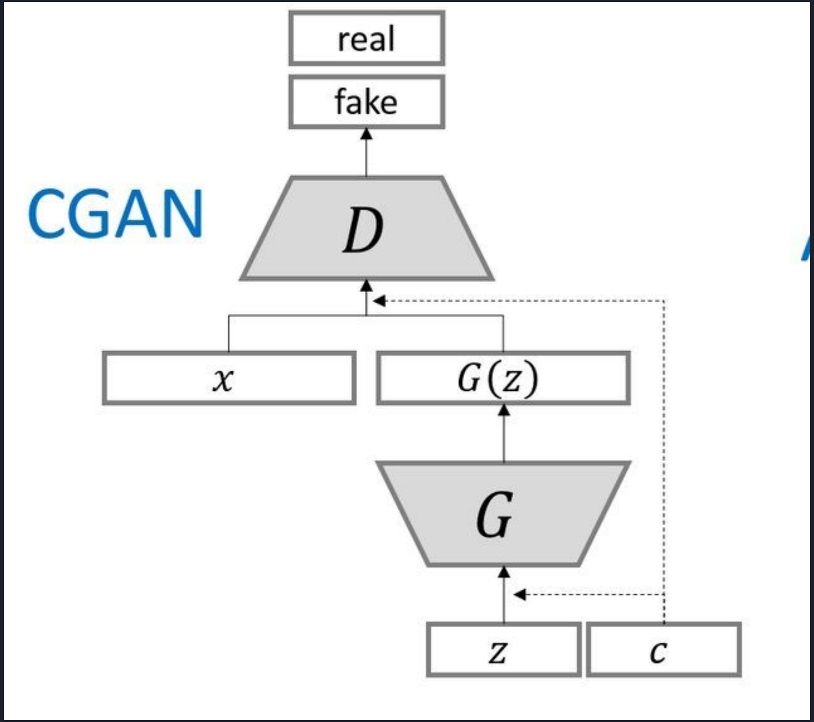


After epoch 99000



Conditional GAN

CGAN Architecture



X - real image
Z - random noise
C- conditioned label
G-Generator
D- Discriminator

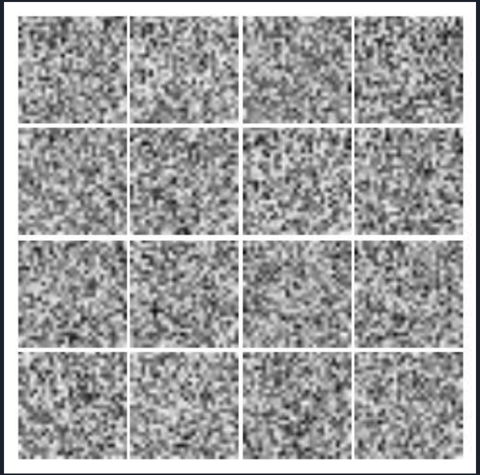
Conditional GAN

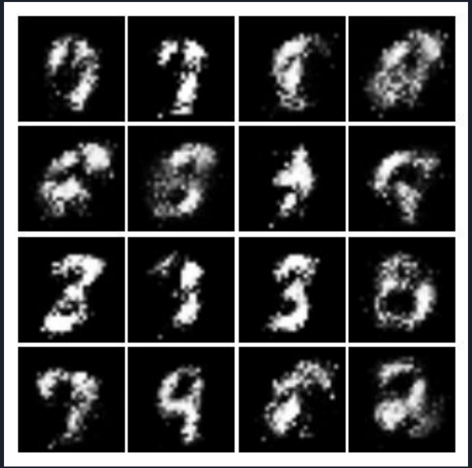
- It uses labels to improve the performance of GAN.
- We condition the G(x) and D(x) networks with some vector y.
- We feed y as an input to both the network.
- Objective function:

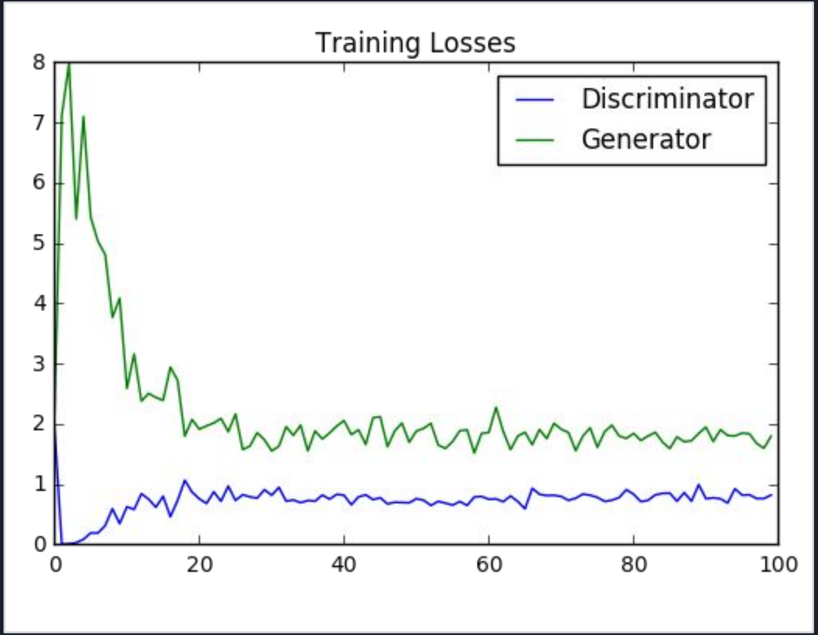
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

- Only difference from GAN is the additional input layer in both the generator and discriminator layers.

Results from CGAN trained


Input Noise


After epoch 5000



Problems in training GAN

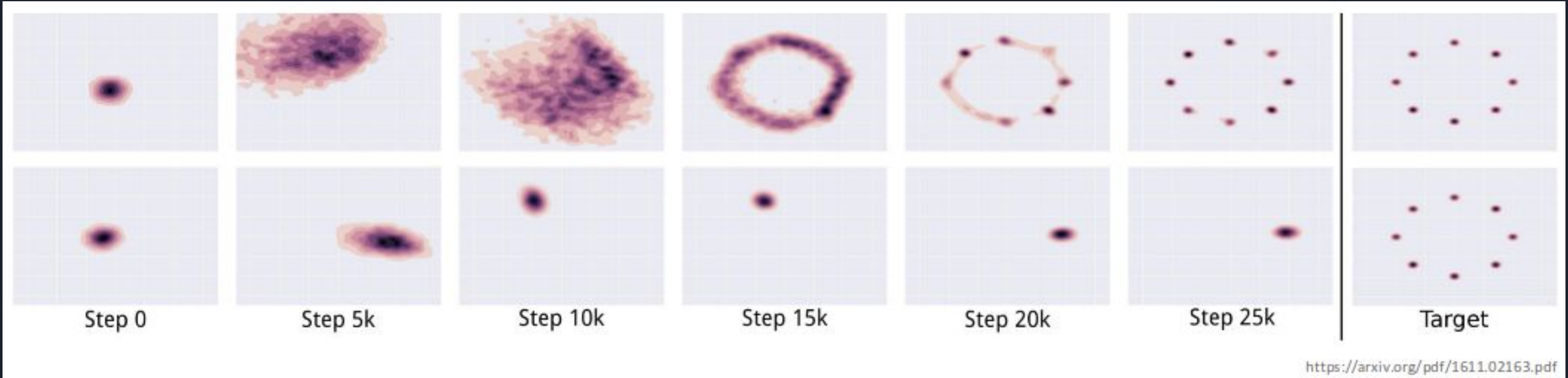
- Non-Convergence
  - Optimization algorithms tries to approach local minima instead of global minima.
  - Hence, the algorithm might not converge
  - May lead to oscillations.
- Mode Collapse
  - Generator collapses producing limited varieties of samples
- Vanishing Gradient
  - This is common problem is deep learning but in GAN since the gradient of Discriminator propagates back to the generator, it gets stronger. Hence, no stability in training.

Non-Convergence

- GANs require finding the equilibrium to a game with two players.
- Even if player successfully moves downhill in player's update, same update might make the other player move uphill.
- Sometimes, the simultaneous gradient descent converges, but otherwise, players continue to undo each others progress and never converge.
- In practice, GANs often seem to oscillate.

Mode Collapse

- Mode Collapse occurs when generator learns to map several input z to a small set of output points.
- Mode collapse may arise because the maxmin solution to the GAN game is different from the minimax solution.

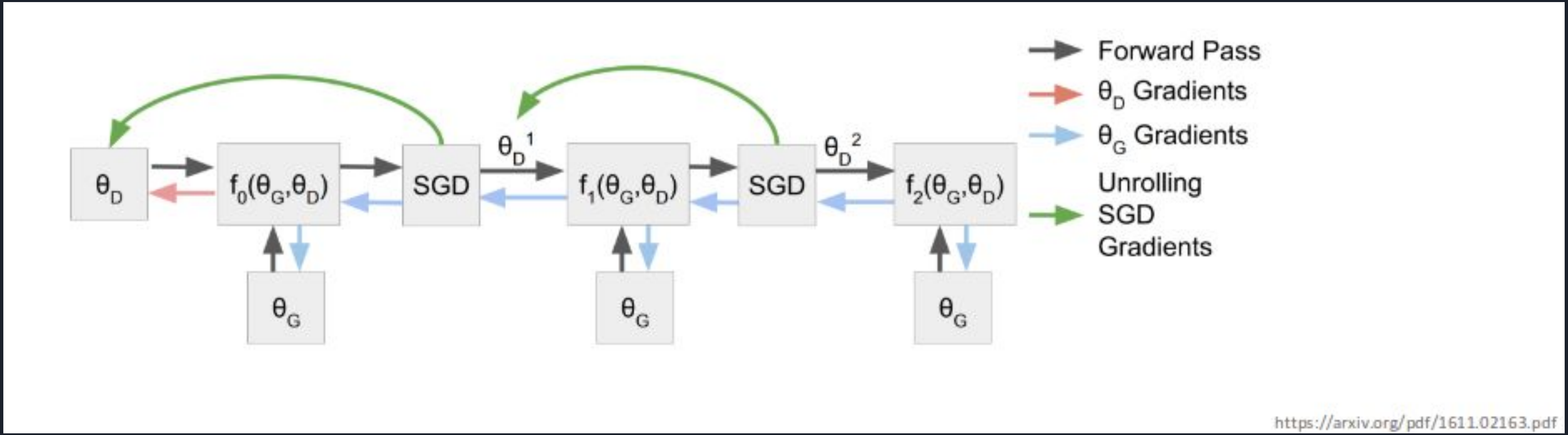


Mode Collapse: Minibatch Features

- One approach to address the issue is Minibatch Features. The basic idea of minibatch features is to allow the discriminator to compare an example to a minibatch of generated samples and a minibatch of real samples.

Mode Collapse: Unrolled GAN

- Another approach is Unrolled GAN. In unrolled GAN, we build a computational graph describing k steps of learning in the discriminator, then backpropagate through all k of these steps of learning when computing the gradient on the generator.



Parzen Window Log Likelihood Estimation

Non-Parametric Estimation

- Probability distribution is unknown.
- We have labeled data, which is used to determine the Probability distribution Function
- These procedures are used with any distribution and without any assumptions about the underlying data.
- There are two types of Non-Parametric procedures:
  - Parzen Window (Region size is constant)
  - Nearest Neighbours (Number of samples in Region are constant)

Density Estimation

- Probability of vector x falling in region R $P(a < x < b) = \int_a^b p(x)dx$
- Assuming n samples in total, by binomial equation, the probability of k samples falling in R is $P_k = \binom{n}{k} P^k (1-P)^{n-k}$
- The Expected value for k, E(k) = nP.
- If n is large, k/n is a good estimate for P.
- If p(x) is continuous and assuming, R to be small such that p doesn't vary significantly, $P = \int_{\Re} p(x')dx' \cong p(x)V$ where V is volume of R.
- Hence, p(x) = (k/n)/V. (Estimate of number of points falling in the volume V)

Density Estimation

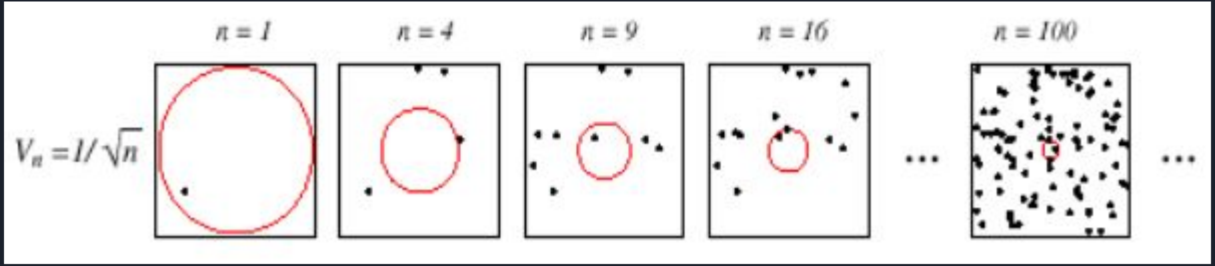
- If we allow the number of samples n to go to infinity, however, we are able to establish convergence of the previous equation to equality.
- For convergence of $p_n(\vec{X}) = \dfrac{k_n/n}{V_n}$

  following conditions must meet:

$$\lim_{n\to\infty} V_n = 0$$
$$\lim_{n\to\infty} k_n = \infty$$
$$\lim_{n\to\infty} k_n/n = 0$$

Parzen Window

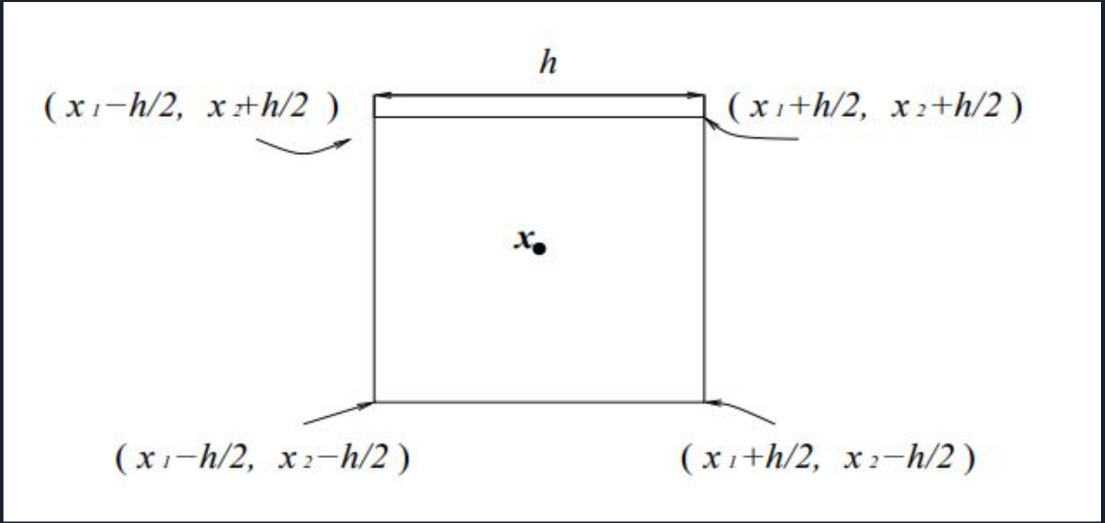
- To satisfy the above conditions, one of the method is Parzen window.
- Here the volume is fixed based on n.



- In this approach, the region is considered either as a hypercube or gaussian distribution.

Parzen Window

- We define a window function, that tells us if the sample is in R:
    - We assign 1 to the sample if it lies within ½ of the edges of the hypercube and 0 if outside.



$$\varphi(u)=\begin{cases}1 & |u_j|\le \frac{1}{2} \quad j=1,\dots,d\\ 0 & \text{otherwise}\end{cases}$$

Parzen Window

- Extending the approach to n samples we get, $\varphi(\frac{x - x_i}{h}) = \begin{cases} 1 & |x - x_i| \le \frac{h}{2} \quad j = 1, \dots, d \\ 0 & \text{otherwise} \end{cases}$

  where x is a sample inside the hypercube of length h centered at xi.

- Since, p(x) = (k/n)/V. We have,

$$k = \sum_{i=1}^{i=n} \varphi\left(\frac{x - x_i}{h}\right) \qquad p_\varphi(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

Parzen Window

- We can generalize the idea for other distributions too. If gaussian function is used, we have $$p(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x_i - x)^2}{2\sigma^2}\right)$$
- This is simply the average of n Gaussian functions with each data point as a center. Σ needs to be predetermined here.

Example: Suppose we have 5 points, x1 = 2, x2 = 2.5, x3 = 3, x4 = 1 and x5 = 6. Now, using gaussian function σ = 1as window function, the parzen probability density function at x = 3 will be calculated by finding contribution of each data point.

Pdf function = 1/√(2Π) exp(-(x1-x)^2/2)

For x = 2, pdf = 0.2420

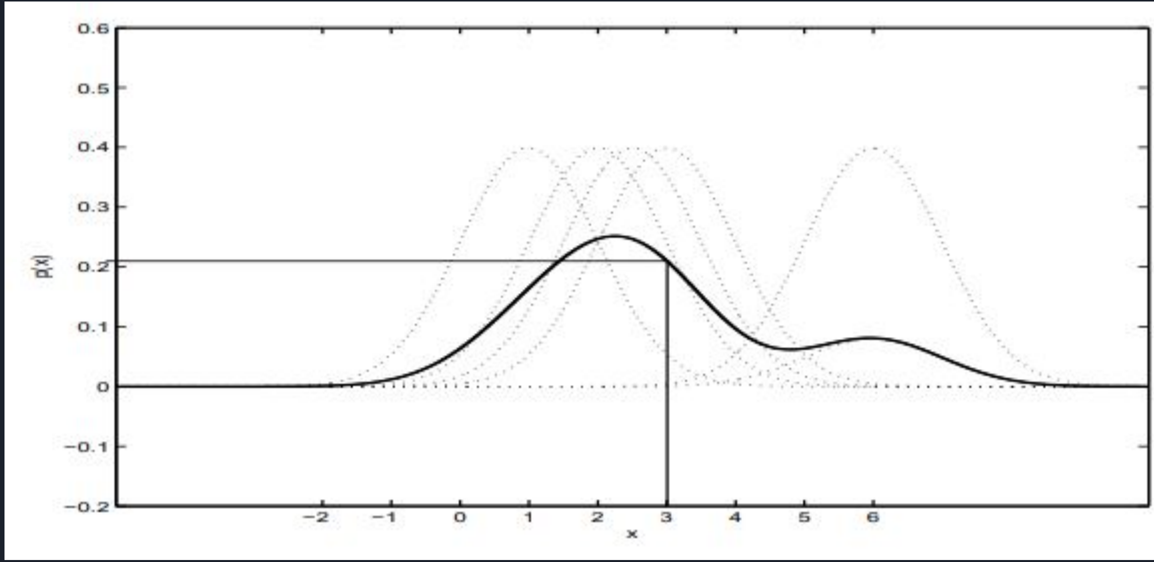
For x = 2.5 pdf = 0.3521

For x = 3, pdf = 0.3989

For x = 1, pdf =0.0540

For x = 6, pdf =0.0044

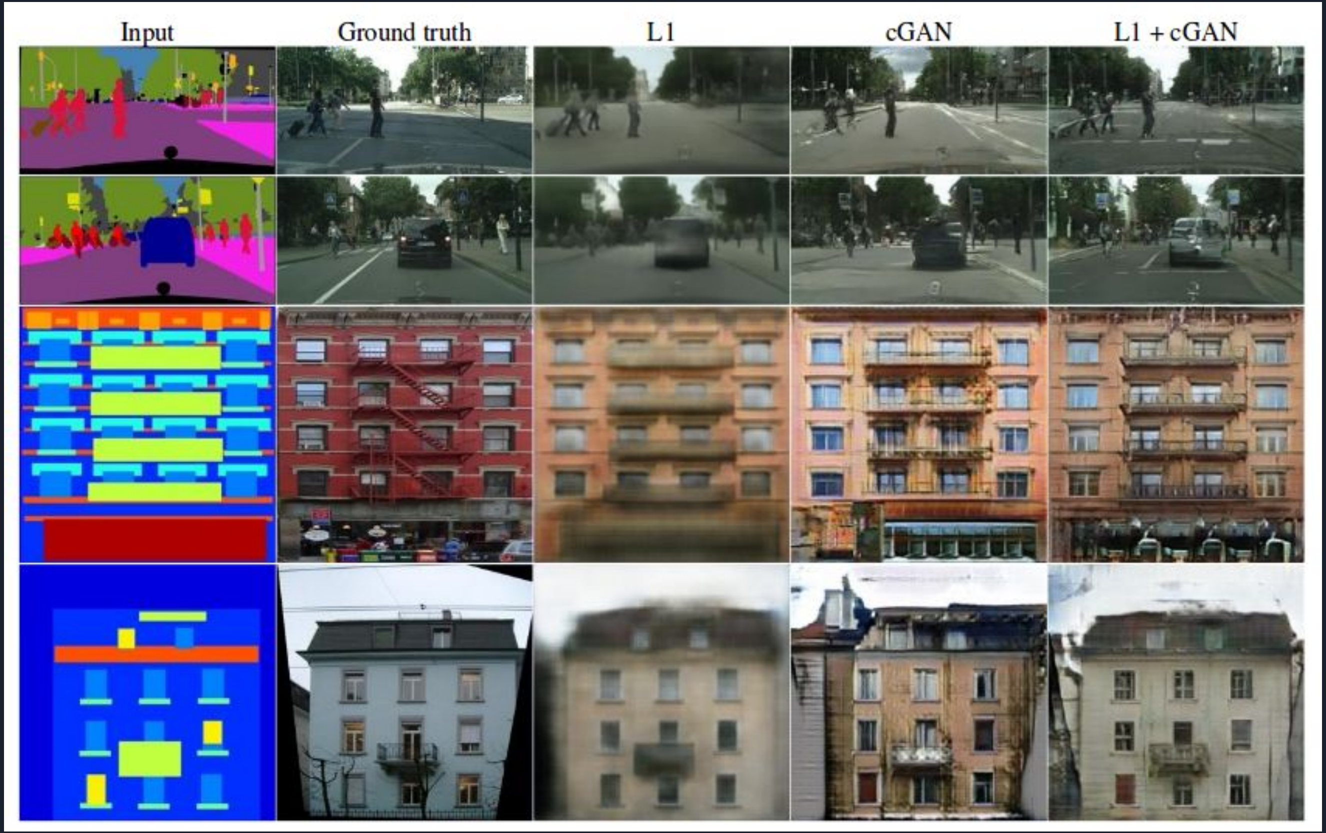
Hence p(x=3) = (0.2420+0.3521+0.3989 +0.0540+ 0.0044)/5 = 0.2103



Dotted lines are Gaussian functions centered at the 5 data points. Black line is the pdf at the given point x = 3.

Ref: http://www.personal.reading.ac.uk/~sis01xh/teaching/CY2D2/Pattern2.pdf

Image to Image Translation

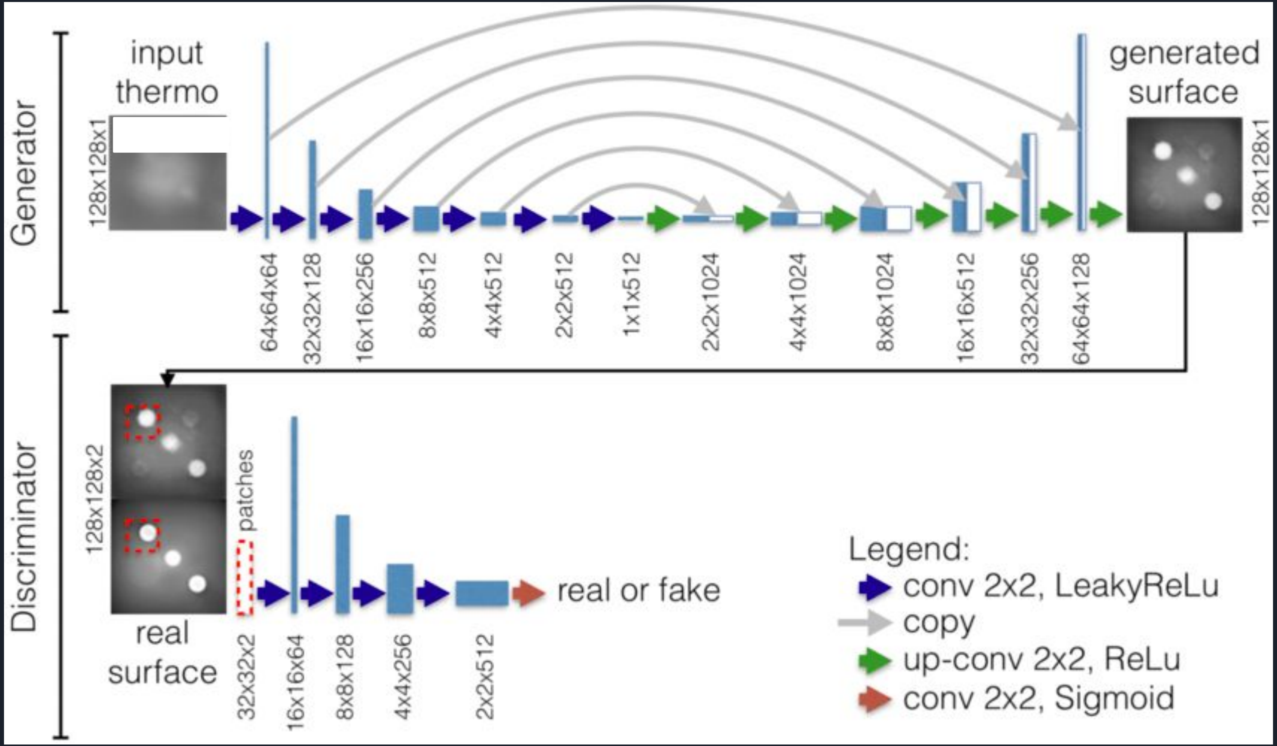


Results from 611.07004.pdf

Image to Image Translation

- Network is trained to map from input pictures (images) to output pictures (images).
- It uses CGAN.
- Generator - U-Net Based
- Discriminator - PatchGAN Based
- Objective: The discriminator's job remains unchanged, but the generator is tasked to not only fool the discriminator but also to be near the ground truth output in an (L1) sense.

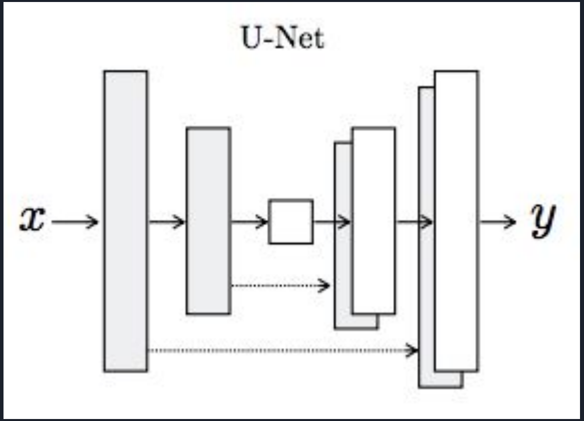
Network Architecture



Generator is U-NET architecture. Discriminator is a PatchGAN

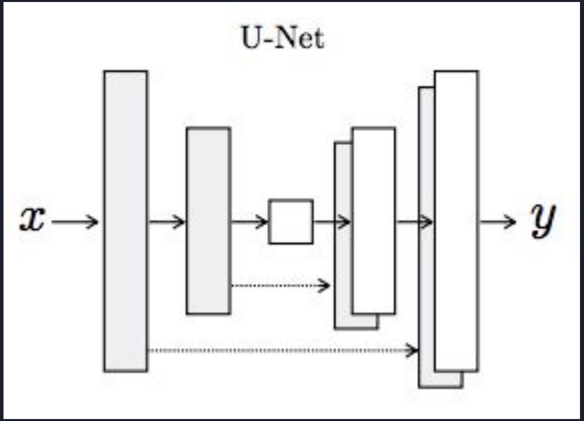
Network Architecture (Generator)

- Basic Architecture is Encoder-Decoder.
- The encoder and decoder both are composed of a stack of convolutional and max pooling layers, whose main components are as follows:
    - Multi-Head Self-Attention Mechanism
    - Position-Encoding and Position-Wise Feed Forward NNs
- The decoder is used to enable precise localization using transposed convolutions.



Network Architecture (Generator)

- The input passes through a series of layers which progressively downsample, until a bottleneck layer, after which the process is expansive.
- To utilise the low level information shared in the layers, U-Net adds "skip connection".
- Skip connections are added between matching layer pairs on either side of the bottleneck.



U-Net architecture. The dotted lines denotes the skip connections

Network Architecture (Discriminator)

- The discriminator architecture is PatchGAN which only penalizes structure at the scale of patches.
- The discriminator tries to classify if each NxN patch in an image is real or fake.
- The discriminator is run across the image and the output D is average of patch responses.
- Hence, Pix2pix can produce sharp images

Network Architecture

- Loss Function for Pix2Pix utilises Loss Function from CGAN and L1 or L2 Loss.

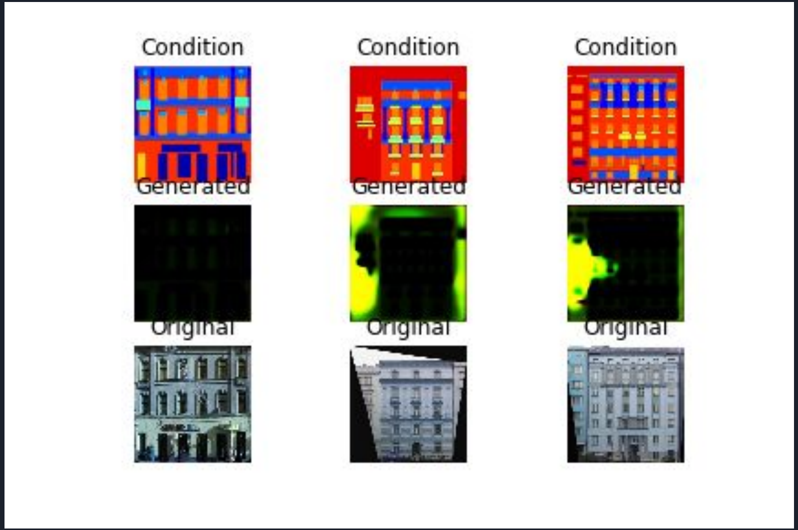
$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))],$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]. \qquad \mathcal{L}_{L_2}(G) = \mathbb{E}_{x,y,z}\left(y - G(x, z)\right)^2$$

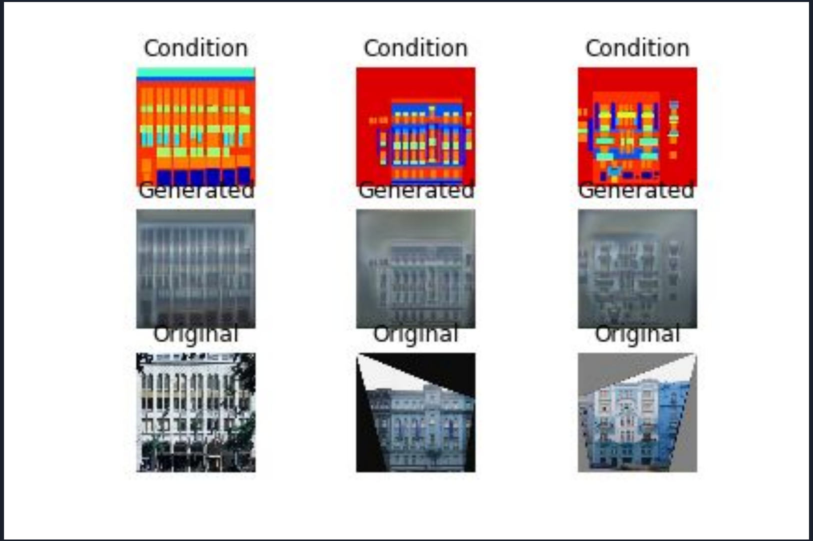
- As the generator G is not only trying to reduce the loss from discriminator but also trying to move the fake distribution close to real distribution by using L1 loss.Hence, Loss function for the generator

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$$

Results from Training

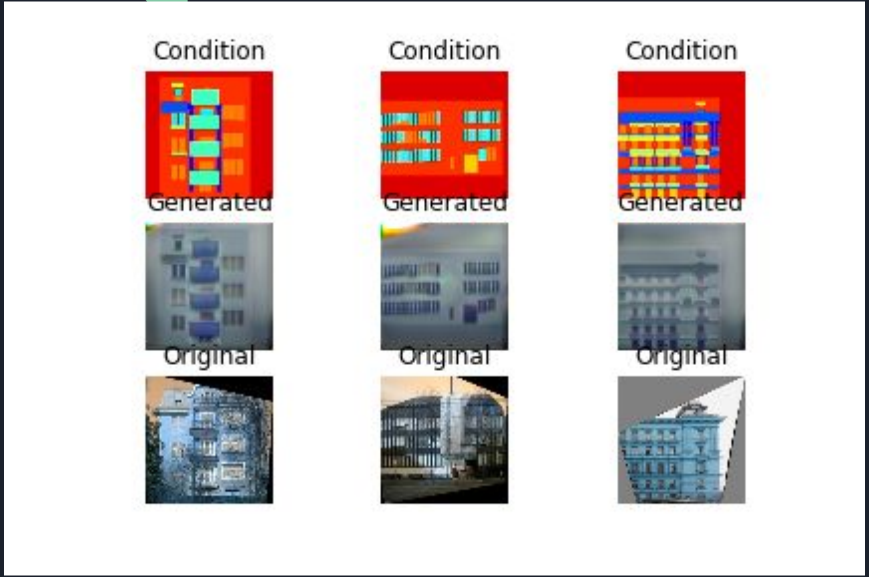


Pix2Pix model trained with only adversarial loss and adversarial loss calculated as cross-entropy.

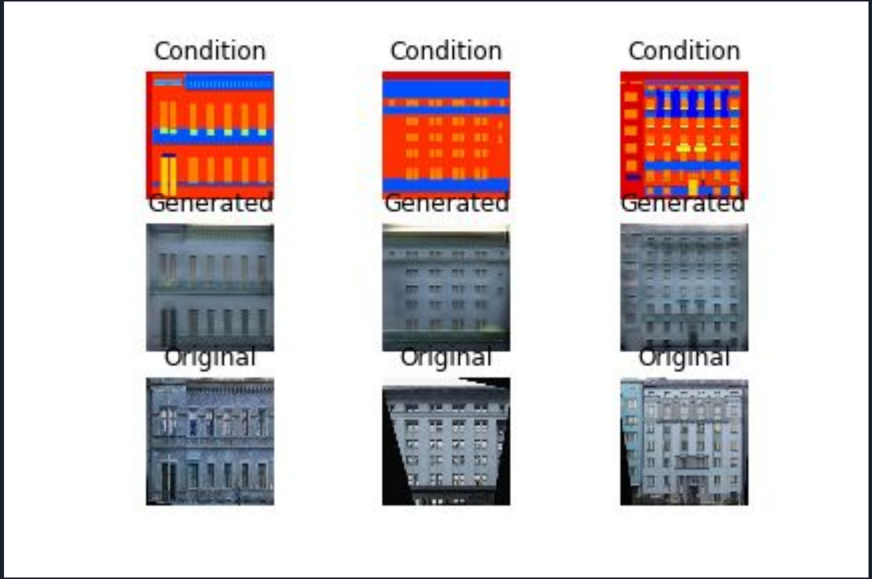


Pix2Pix model trained with a combination of adversarial loss and L2 loss. adversarial loss calculated as cross-entropy.

Results from Training

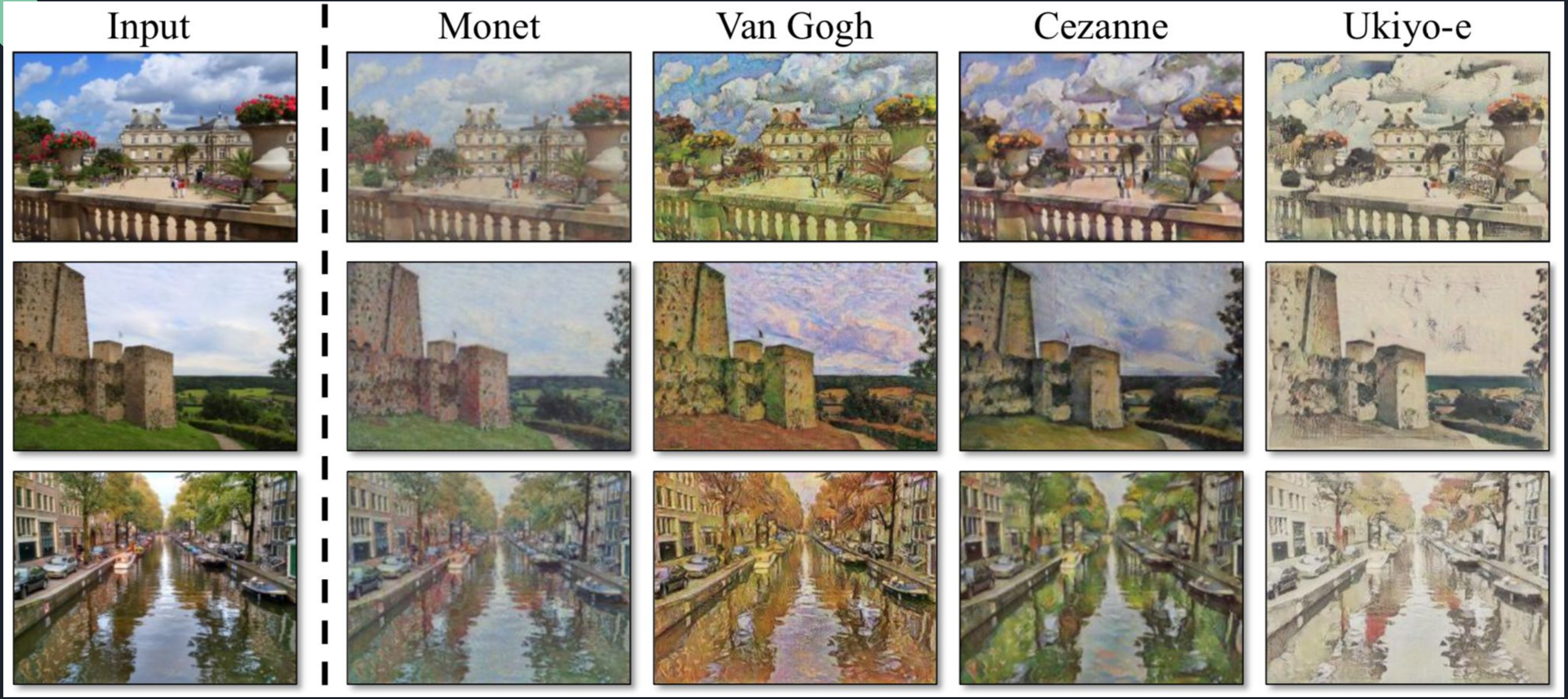


Pix2Pix model trained with just adversarial loss. adversarial loss calculated as mean squared error.



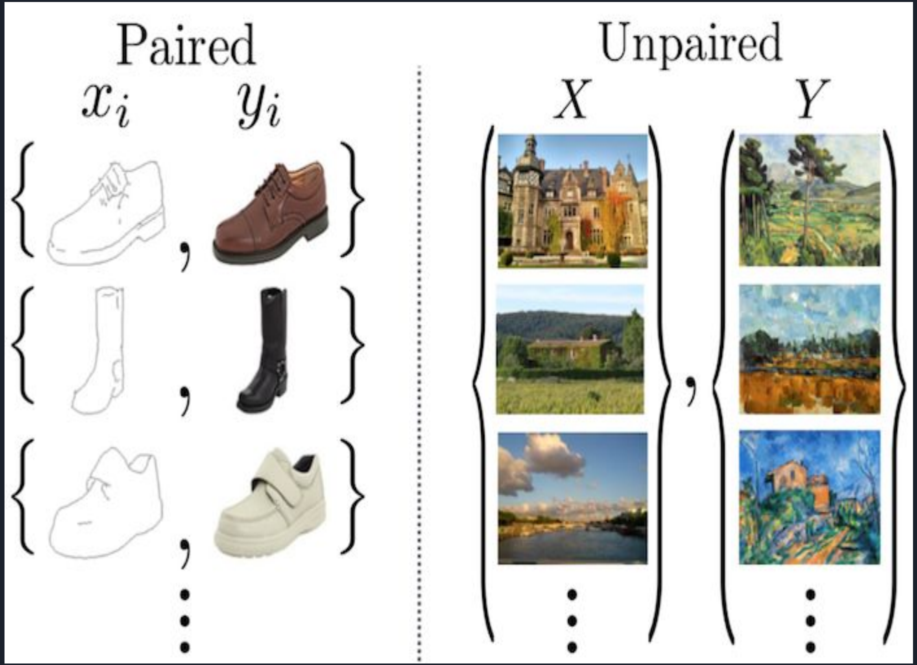
Pix2Pix model trained with adversarial loss and L1 loss as in the Pix2Pix paper. adversarial loss calculated as mean squared error.

CycleGAN



CycleGAN can be used for collection style transfer, where the entire works of an artist are used to train the model. *Image taken from paper.

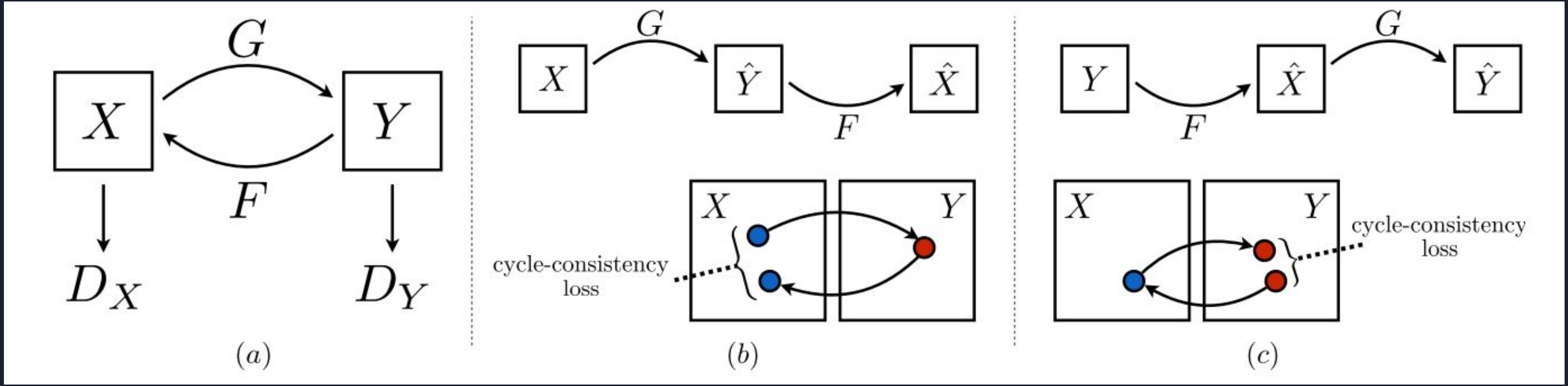
Cycle GAN



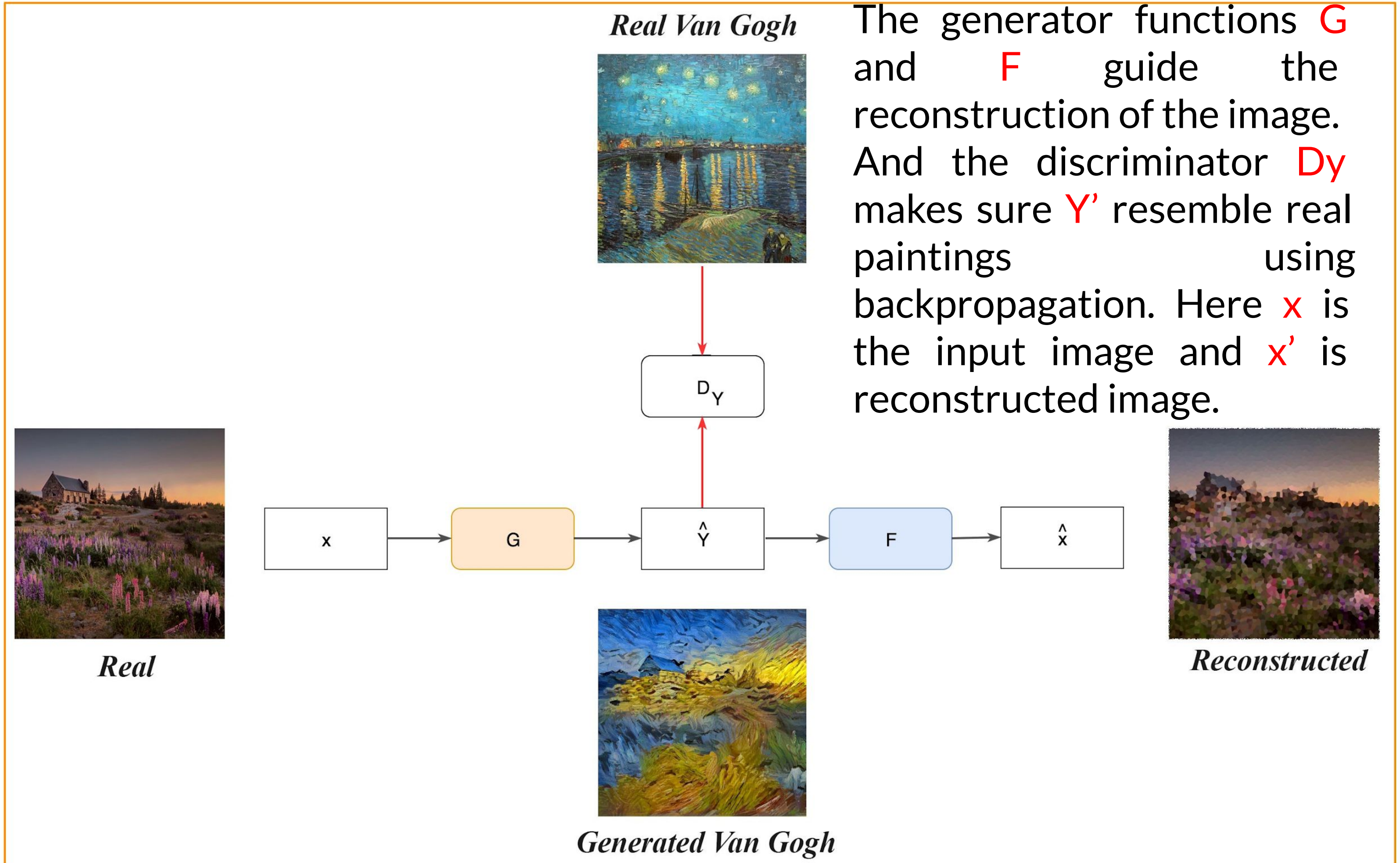
- Pix2Pix uses paired training data for image to image translation.
- But, generally, it is difficult to gather paired training data.
- CycleGAN is an approach for learning the cross-domain translation without the paired training samples.
- CycleGAN learns the style of images as a whole and applies it to other types of images.

Cycle GAN

- We use a deep network G to convert image x to y.
- We reverse the process with another deep network F to reconstruct the image.
- Then, we use a mean square error MSE to guide the training of G and F.
- D acts as a critic between the training samples and the generated images.



a)Network Architecture contains two Generators and two Discriminators. b)&c) The generators G and F are cycle consistent. L1 metric is used to calculate the cycle-consistency loss.



The generator functions G and F guide the reconstruction of the image. And the discriminator Dy makes sure Y' resemble real paintings using backpropagation. Here x is the input image and x' is reconstructed image.

Network Design

- We build three networks.
  - A generator G to convert a real image X to a style picture y.
  - A generator F to convert a style picture y to a real image X.
  - A discriminator D to identify real or generated style pictures.
- For the reverse direction, we just reverse the data flow and build an additional discriminator Dx to identify real images.

Cost Function

- For training, utilize the adversarial loss for generator and discriminator.

$$\mathcal{L}_{\text{GAN}}(G, D, X, Y)$$

$$\textit{For } G, \text{ minimize } \mathbb{E}_{x \sim p_{\text{data}}(x)}[(D(G(x)) - 1)^2]$$

$$\textit{For } D, \text{ minimize } \mathbb{E}_{y \sim p_{\text{data}}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(G(x))^2]$$

Cost Function

- But, mapping is underconstrained (i.e. the pairing might not be meaningful) . Also, it generally leads to problem like mode-collapse.
- Hence, we couple it with “cycle consistency loss” (calculate L1 norm) - We take another mapping F: Y->X and this loss to reinforce F(G(x)) = x.
- G: X-> Y and F: Y->X such that F and G are bijections.

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

Cost Function

- Final Objective Function:

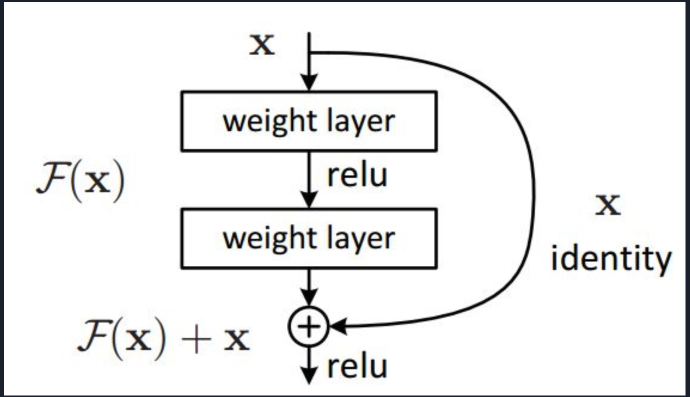
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

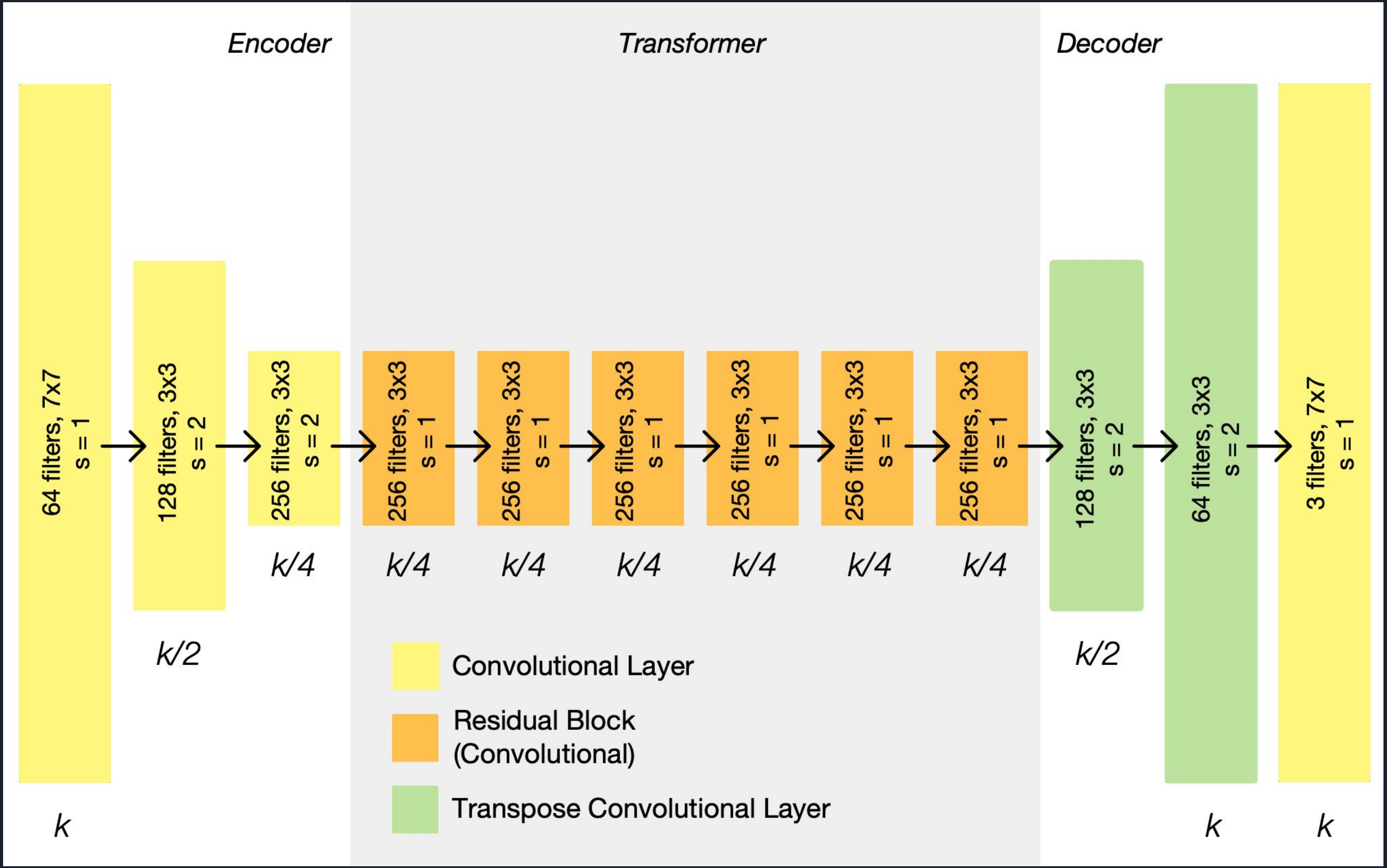
Generator Architecture

- CycleGAN generator has three sections: an encoder, a transformer, and a decoder.
- The encoder shrinks the representation size while increasing the number of channels of the input. The encoder is composed of three convolution layers.
- The resulting activation is then passed to the transformer, a series of six residual blocks.
- It is then expanded again by the decoder, which uses two transpose convolutions to enlarge the representation size, and one output layer to produce the final image in RGB.

Transformer Architecture

- In traditional neural networks, each layer feeds into the next layer.
- In a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away.
- The idea here is to learn a context vector (say U), which gives us global level information on all the inputs and tells us about the most important information .



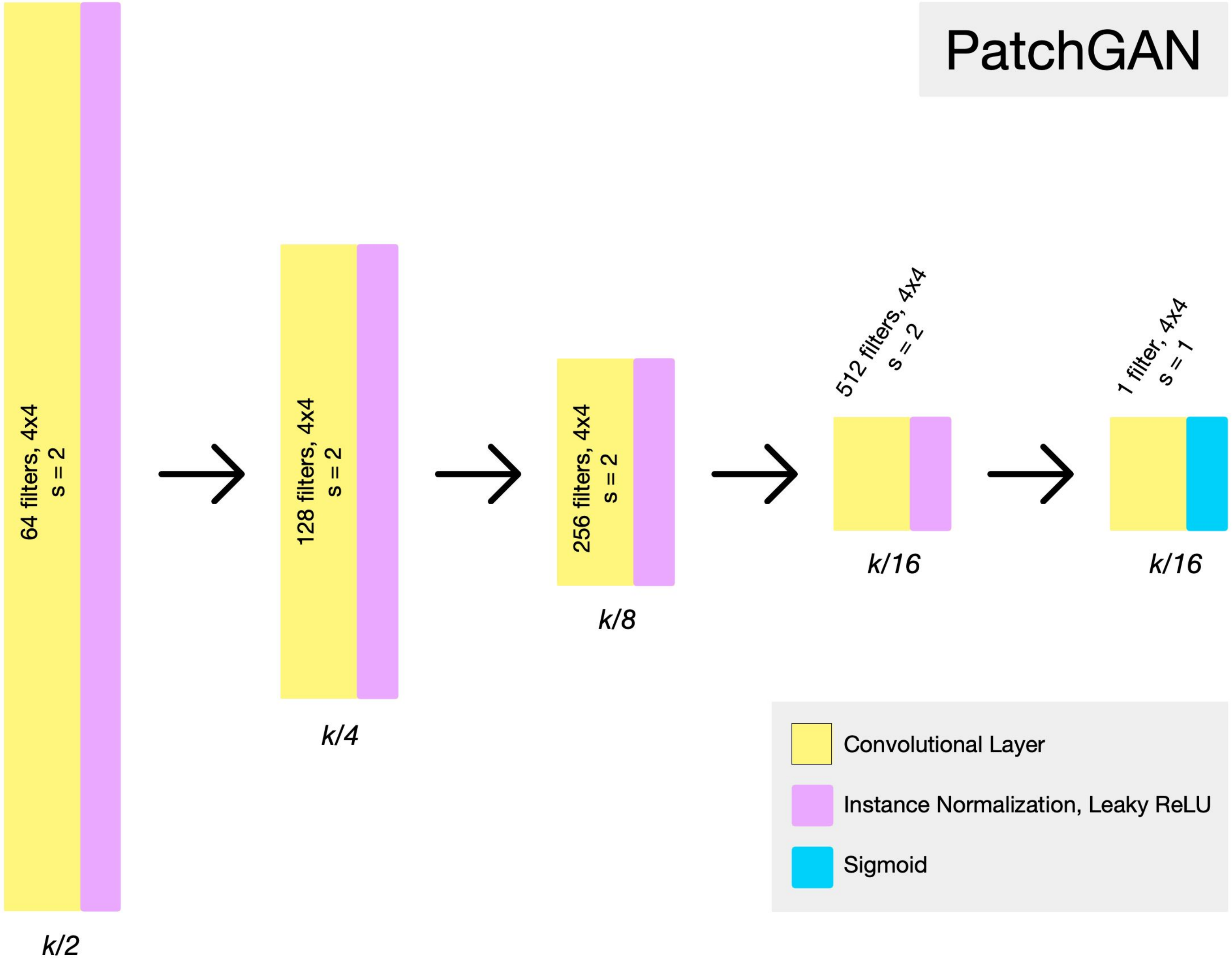


Generator Architecture. Contains three parts: a) Encoder b)Transformer c) Decoder

Discriminator Architecture

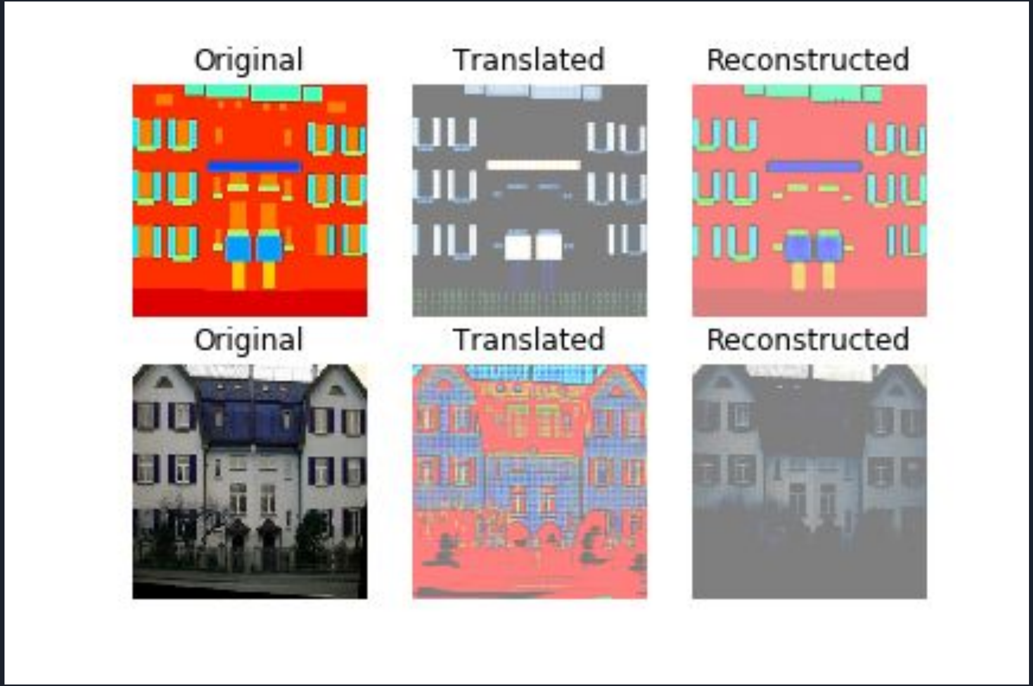
- The discriminators are PatchGANs, convolutional neural networks that look at a “patch” of the input image, and output the probability of the patch being “real”.
- It allows the discriminator to focus on more surface-level features, like texture.

PatchGAN



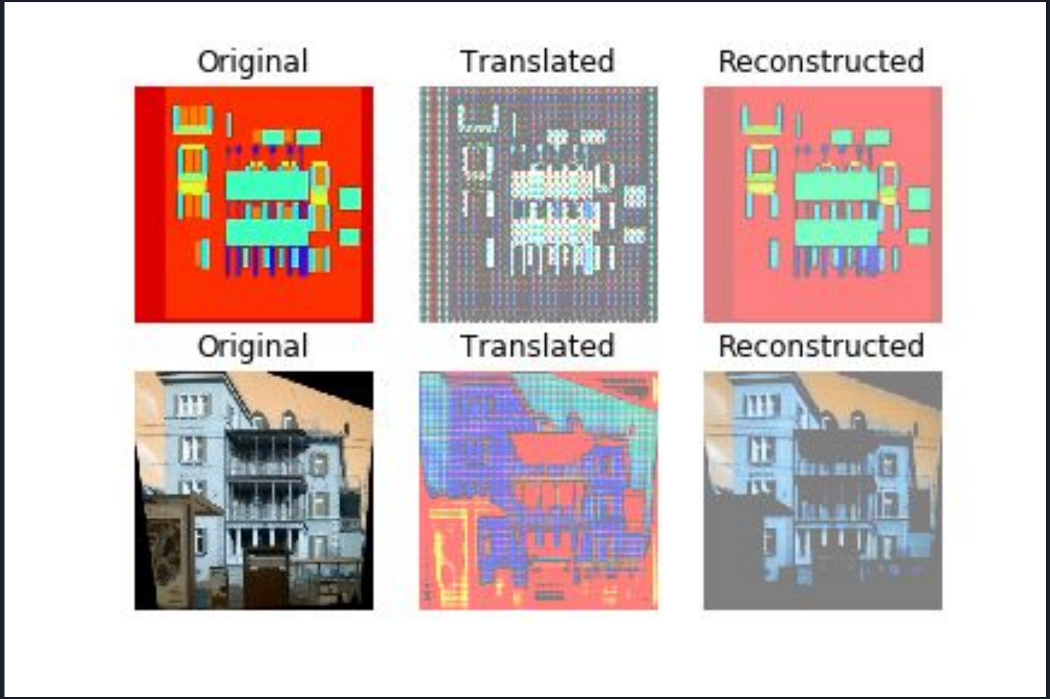
64 filters, 4x4
s = 2
k/2
128 filters, 4x4
s = 2
k/4
256 filters, 4x4
s = 2
k/8
512 filters, 4x4
s = 2
k/16
1 filter, 4x4
s = 1
k/16
Convolutional Layer
Instance Normalization, Leaky ReLU
Sigmoid

Results from Training



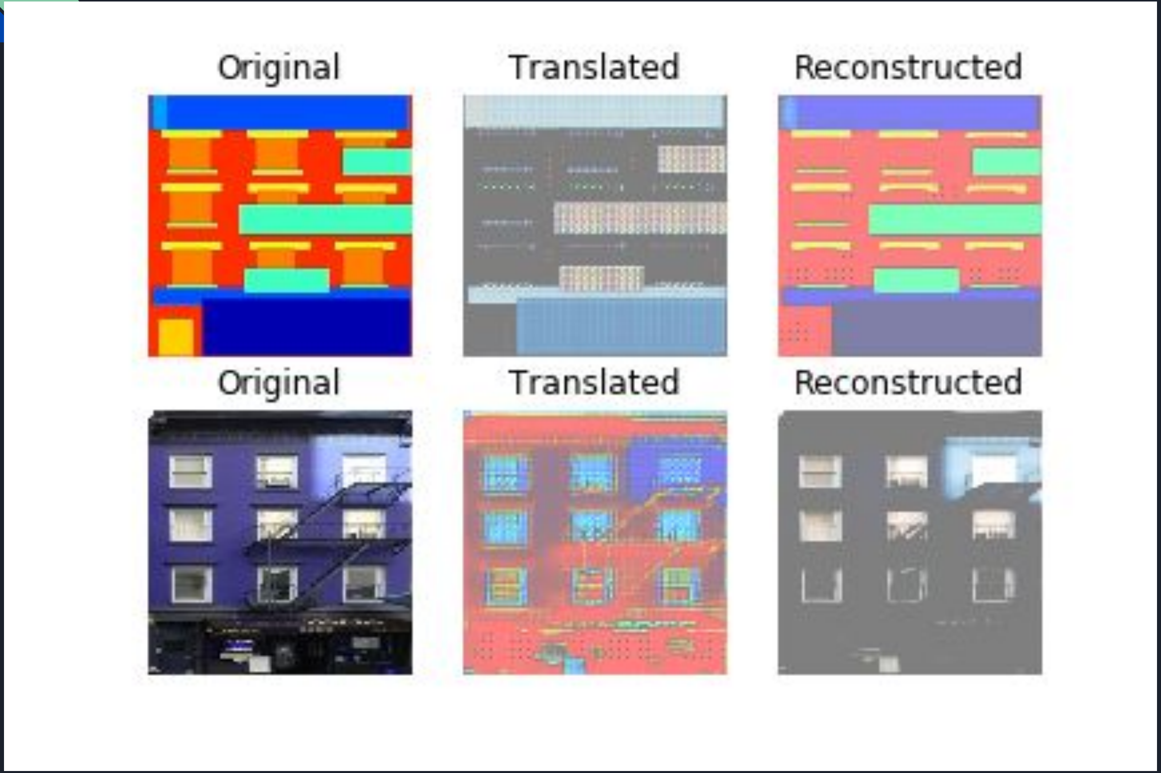
CycleGAN trained with lambda = 10.

Results from Training



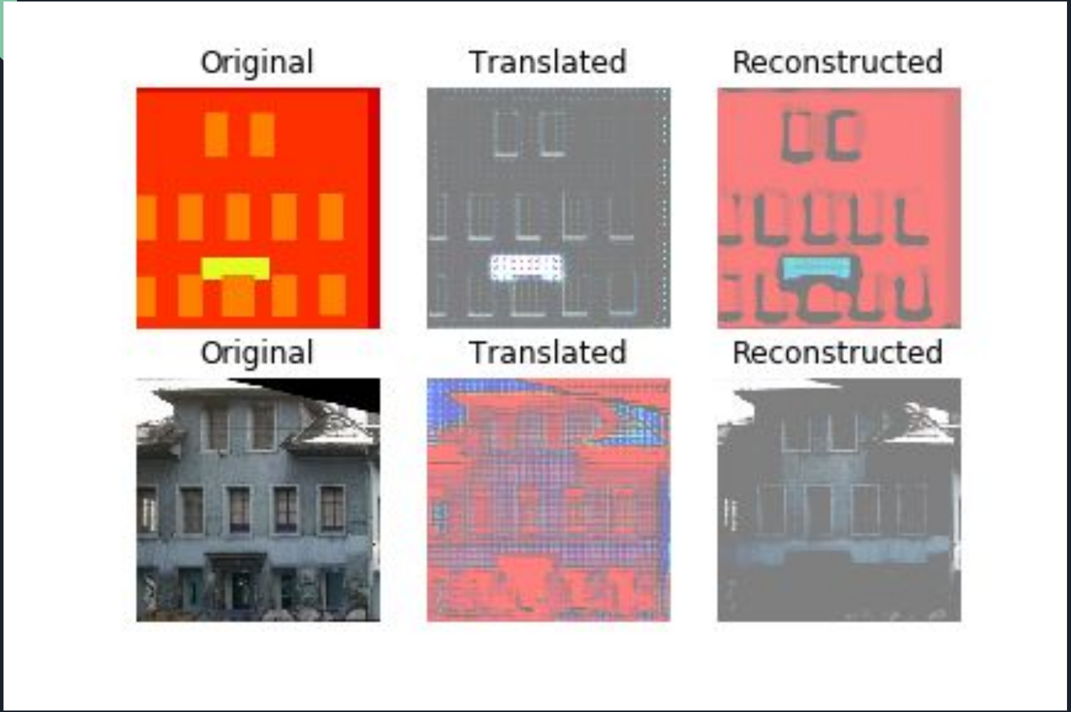
CycleGAN trained with lambda = 1 .

Results from Training



CycleGAN trained with lambda = 5.

Results from Training



CycleGAN trained with lambda = 15 .

Limitations

- Works well for problems which involves colour or texture.
- However, tasks that require substantial geometric changes to the image, such as cat-to-dog translations, usually fail.
- Translations on the training data often look substantially better than those done on test data.

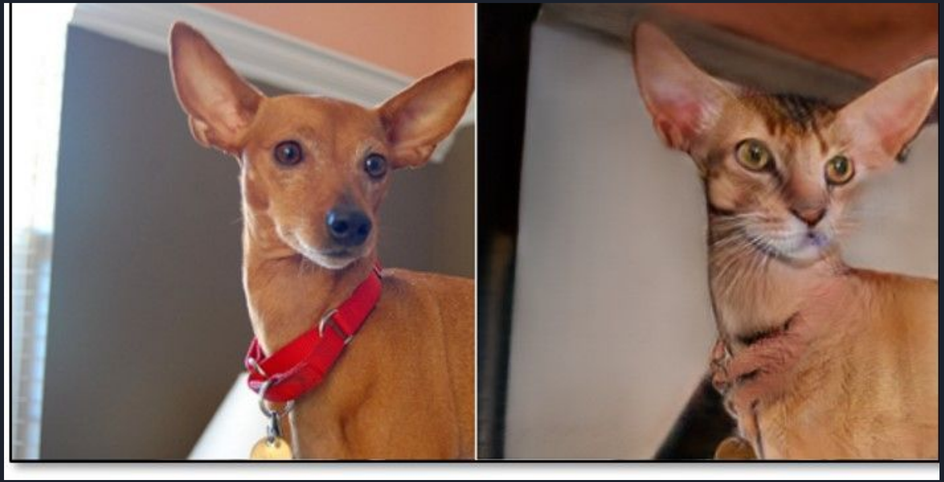


Image from project page for Dog->Cat translation using CycleGAN

ThankYou

Readings

- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In NIPS'2014.
- Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks, M. Mirza and S. Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.
- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks.
- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros
- Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation.
- Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks, Chuan Li and Michael Wand

Readings

- Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
- Sequence to Sequence Learning with Neural Networks, Ilya Sutskever, Oriol Vinyals, Quoc V. Le
- Fully Convolutional Networks for Semantic Segmentation, Jonathan Long, Evan Shelhamer, Trevor Darrell
- Perceptual Losses for Real-Time Style Transfer and Super-Resolution, Justin Johnson, Alexandre Alahi, Li Fei-Fei
- Improved Techniques for Training GANs. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Alec Radford, Vicki Cheung, Xi Chen.
- Pros and Cons of GAN Evaluation Measures, Ali Borj
- Pattern Classification by Richard O. Duda, David G. Stork, Peter E.Hart

Readings

- http://www.csd.uwo.ca/~olga/Courses/CS434a_541a/Lecture6.pdf
- https://github.com/yassineELMAAZOUZ/Parzen-Window-Kernel-Density-Estimation/blob/master/projet_MAP.pdf
- https://www.projectrhea.org/rhea/index.php/Parzen_Window_Density_Estimation
- www.personal.reading.ac.uk/~sis01xh/teaching/CY2D2/Pattern2.pdf
- https://milania.de/blog/Introduction_to_kernel_density_estimation_%28Parzen_window_method%29
- http://ssg.mit.edu/cal/abs/2000_spring/np_dens/density-estimation/parzen62.pdf
- https://sebastianraschka.com/Articles/2014_kernel_density_est.html
- Classification with Deep Belief Networks, Hussam Hebbo, Jae Won Kim.
- 2007 NIPS Tutorial on: Deep Belief Nets, Geoffrey Hinton.
- Hugo Larochelle, youtube videos on RBM and DBN.
-