How to Develop a Pix2Pix GAN for Image-to-Image Translation

Date : 2022-01-19

KwangWoob University Visual Informatics Lab

김동욱, 박형빈, 배홍섭, 정태민, 최은영

Overview

- What is Pix2Pix?
- Prerequisites
  - GAN, Conditional GAN
- Pix2Pix
  - Objection Function
  - Network Architecture
  - Generator with skip connection
  - Discriminator with PatchGAN
  - Optimization and inference
- Experiments with source code
- Conclusion

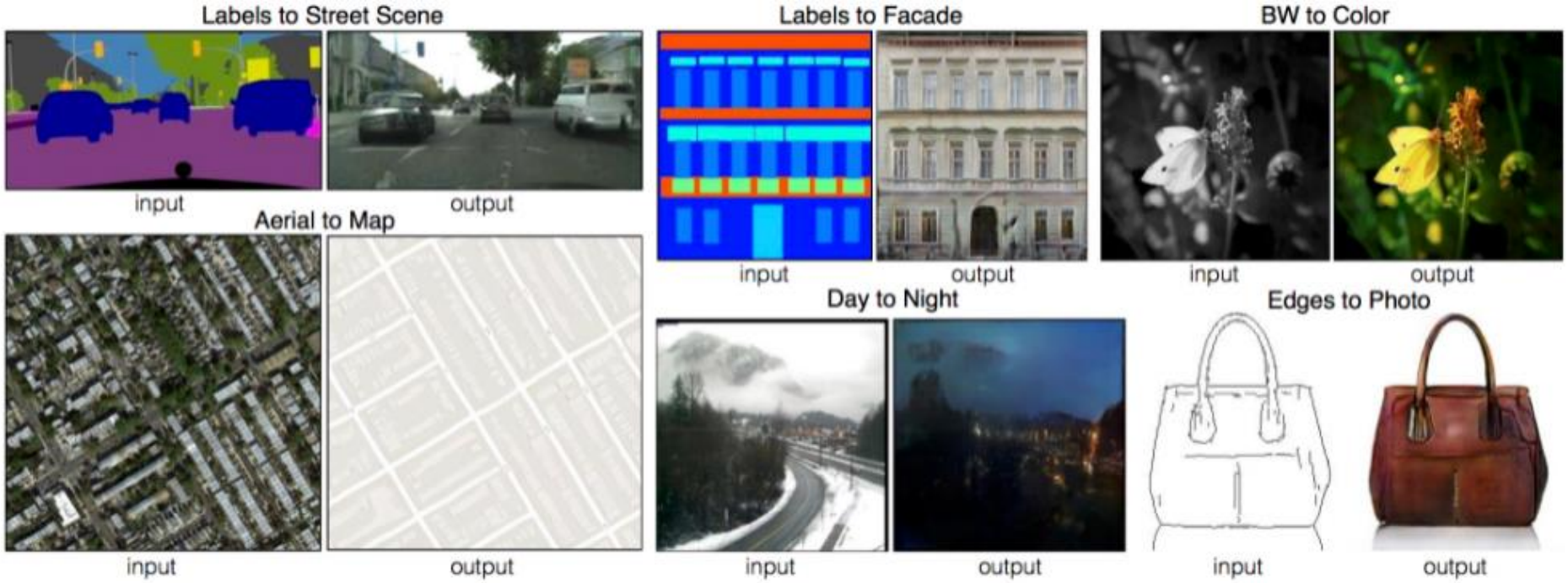
Official Reference

You can check the detailed contents about pix2pix GAN model in the link below

- GitHub
  https://github.com/phillipi/pix2pix
- Paper : Image-to-Image Translation with Conditional Adversarial Networks
  https://arxiv.org/abs/1611.07004
- Projects
  https://phillipi.github.io/pix2pix/

What is Pix2Pix?

- Image-to-Image translation is the task that translates images from one domain to another by learning a mapping between the input and output images.
- And the models representing these fields(image-to-image translation) are **pix2pix** and CycleGAN.
- **Pix2Pix GAN** further extends the idea of CGAN, where the images are translated from input to an output image, conditioned on the input image.

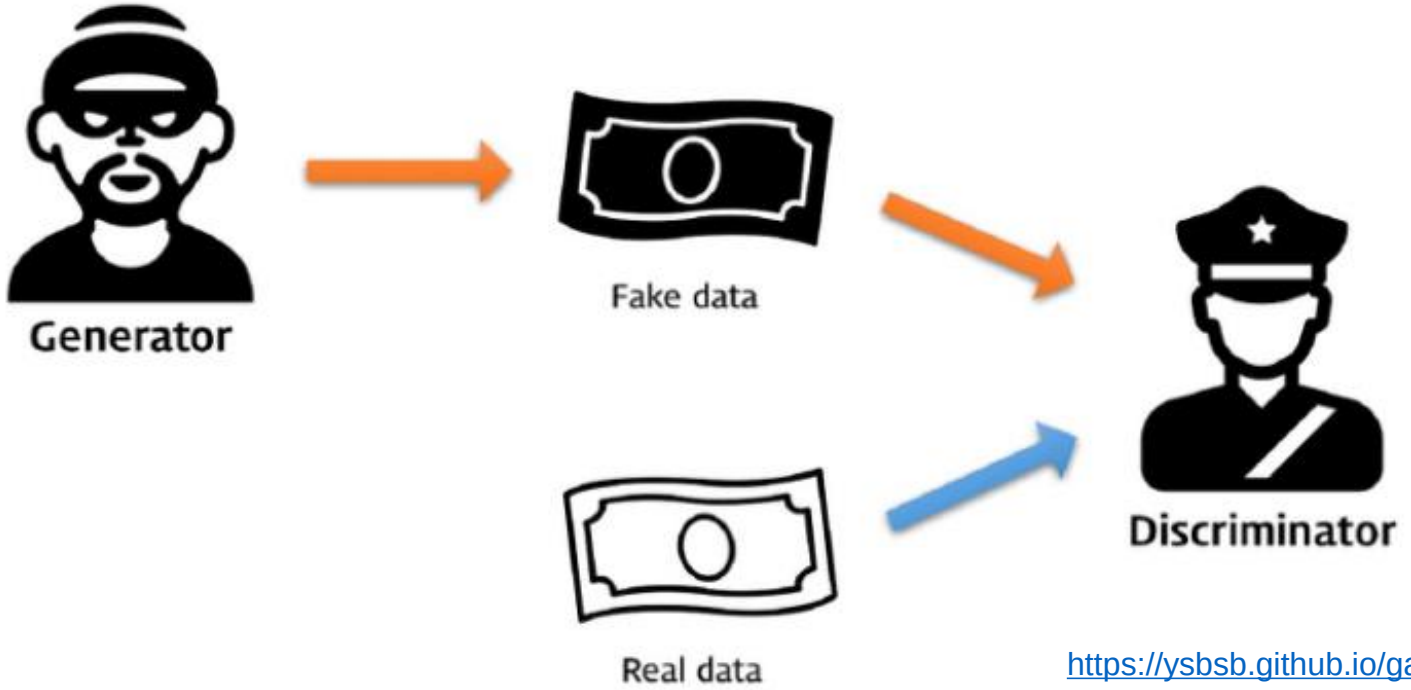


Example results on several image-to-image translation problems. In each case we use the same architecture and objective, simply training on different data.

https://ysbsb.github.io/gan/2020/06/17/GAN-newbie-guide.html

Prerequisites : GAN

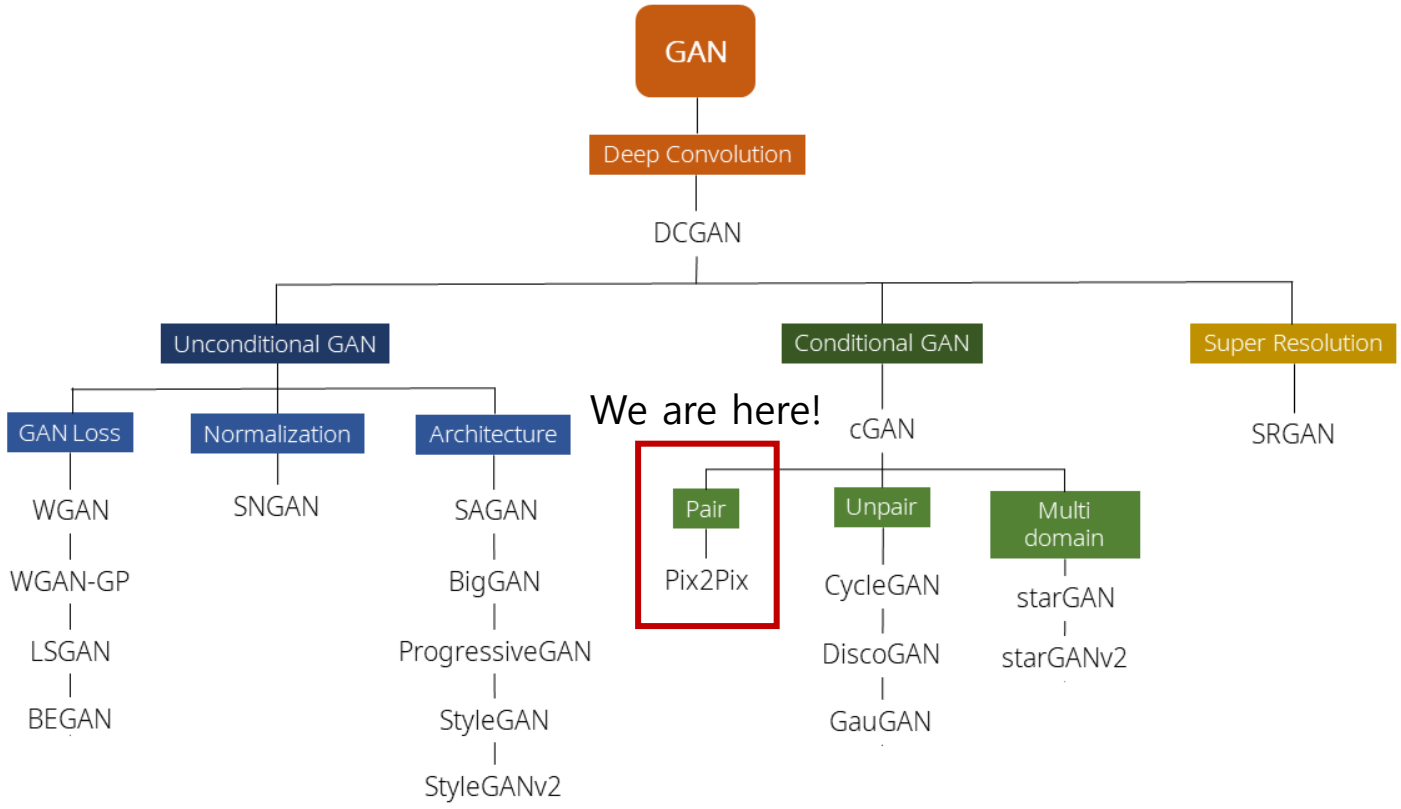
- Prior to learning up pix2pix modeling, it is easier to understand with basic knowledge of **GAN** and **CGAN**. Especially, authors of the paper emphasizes **Conditional Adversarial Networks, CGAN**.
- Generative adversarial network, or GAN for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks(CNN).
- The **generator model** that we train to generate new examples, and the **discriminator model** that tries to classify examples as either real(from the domain) or fake(generated).



https://ysbsb.github.io/gan/2020/06/17/GAN-newbie-guide.html

Prerequisites : CGAN

- Prior to learning up pix2pix modeling, it is easier to understand with basic knowledge of GAN and CNN. Especially, authors of the paper emphasizes Conditional Adversarial Networks, CGAN.
-



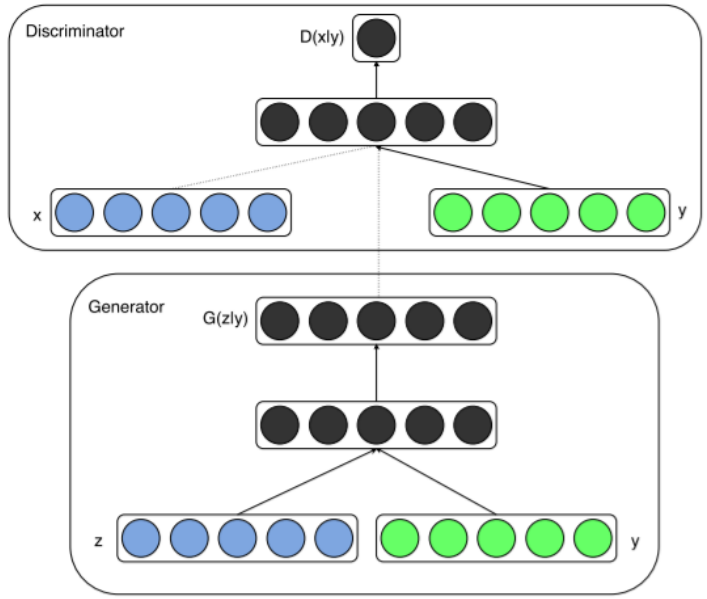


Figure 1: Conditional adversarial net

https://ysbsb.github.io/gan/2020/06/17/GAN-newbie-guide.html

Objective

- In the objective of a conditional GAN, G tries to minimize objective against an adversarial D that tries to maximize it, i.e. $\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x,z))]$,
- Previous approaches have found it beneficial to mix the GAN objective with a more traditional loss, such as L2 distance(or L1 distance). So paper also explore this option, using L1 distance rather than L2 distance. Because L1 encourages less blurring.
- *This motivates restricting the GAN discriminator to only model high-frequency structure, relying on an L1 term to force low-frequency correctness.*

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

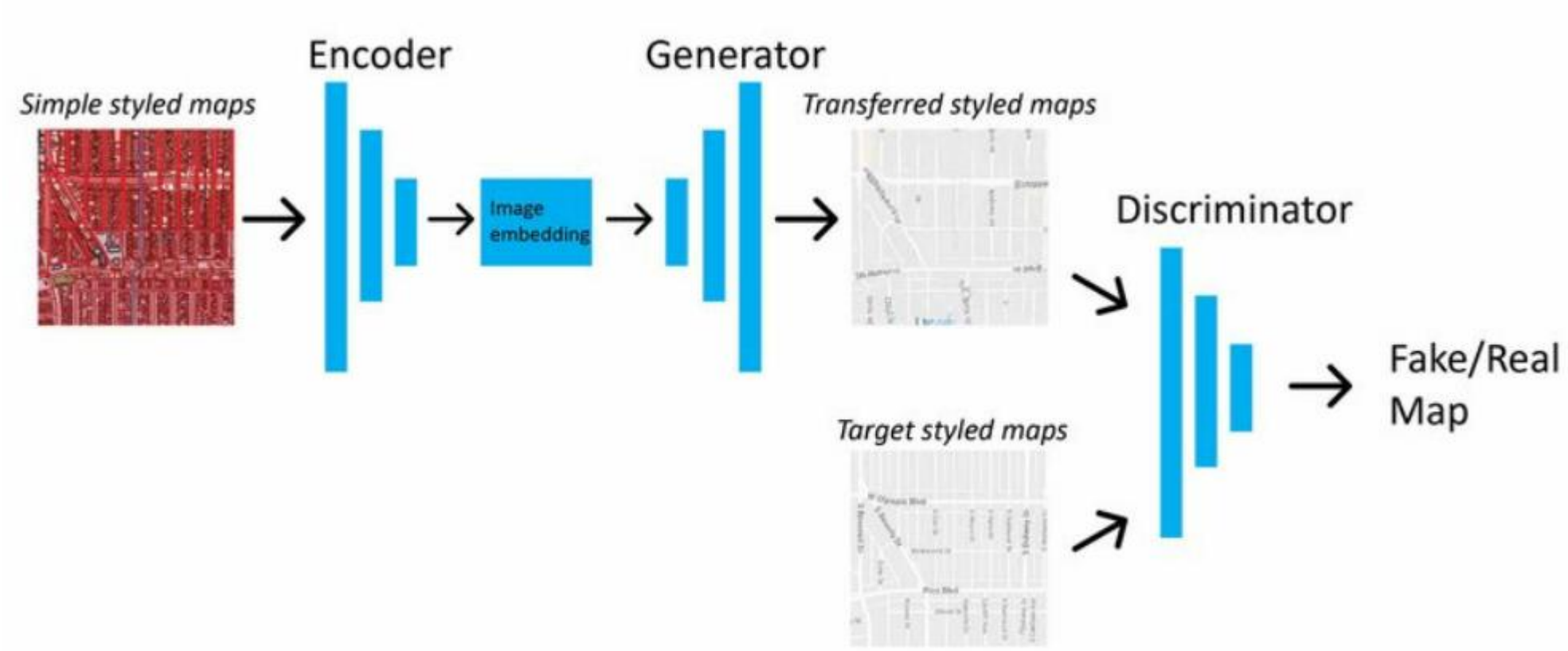
$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x,z))], \quad \mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1].$$

$$\text{GAN: } \min_G \max_D V(D,G) = \mathbb{E}_{x \sim P_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim P_z(z)}[log(1 - D(G(z)))]$$

https://arxiv.org/abs/1611.07004

Network Architecture

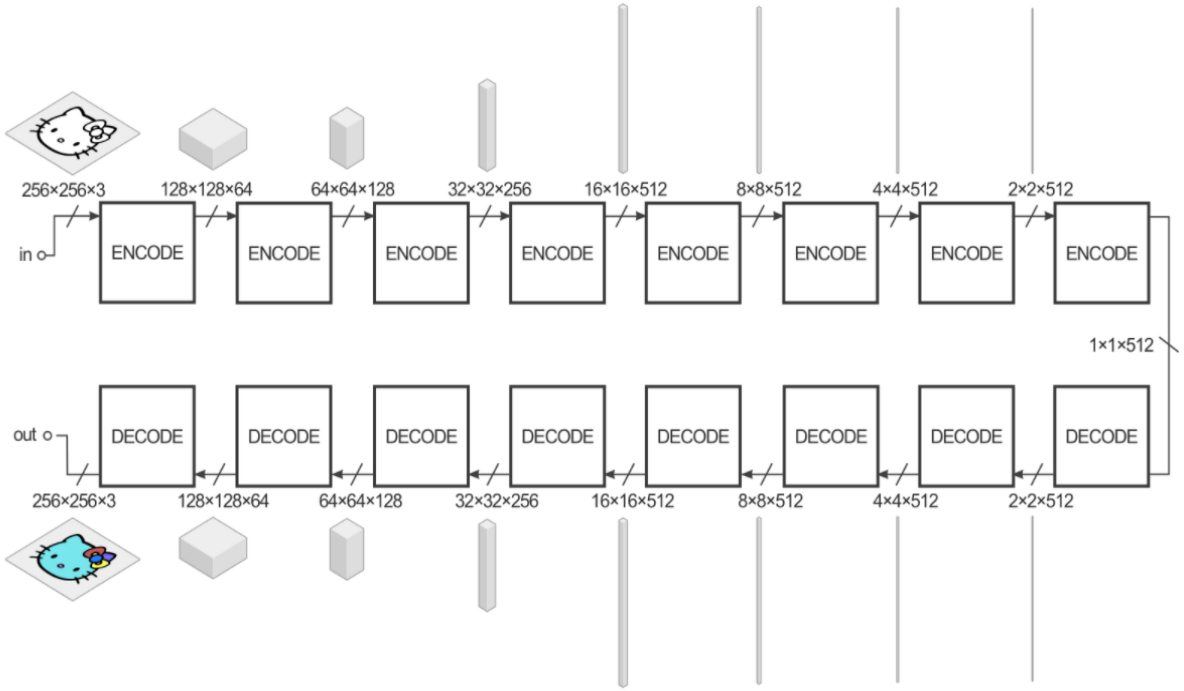
- Adapt generator and discriminator architectures from Unsupervised representation learning with deep convolutional generative adversarial networks
- Both generator and discriminator use modules of the form **convolution-BathNorm-ReLu**



https://www.researchgate.net/figure/Data-flow-of-Pix2Pix-in-this-research_fig2_332932603

Generator with skip connection

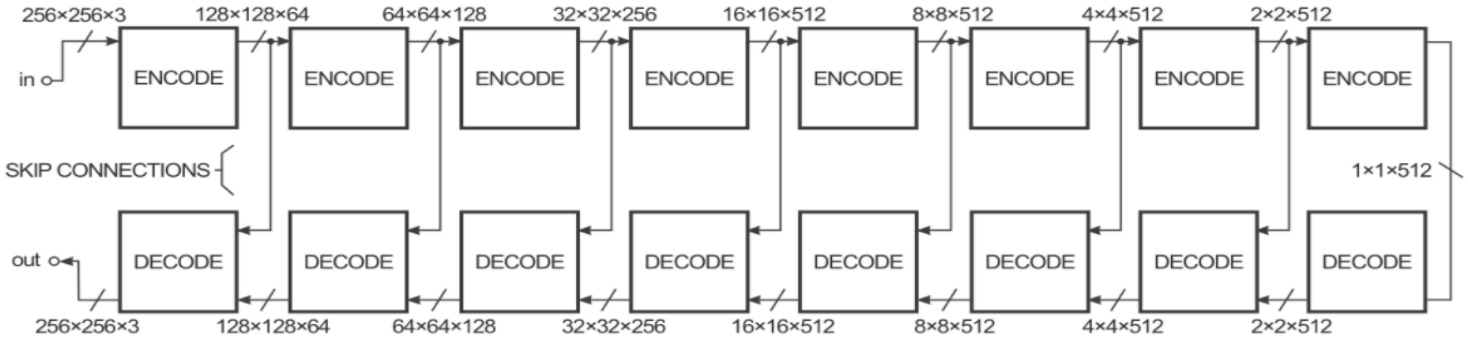
- The generator takes some input and tries to reduce it with a series of encoders (convolution + activation function) into a much smaller representation. The idea is that by compressing it this way we hopefully have a higher level representation of the data after the final encode layer.
- The decode layers do the opposite (deconvolution + activation function) and reverse the action of the encoder layers

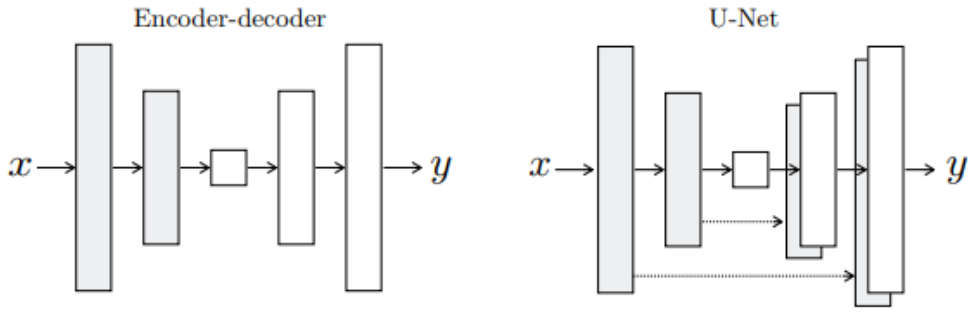


https://affinelayer.com/pix2pix/

Generator with skip connection

- In order to improve the performance of the image-to-image transform in the paper, the authors used a "U-Net" instead of simple encoder-decoder architecture. This is the same thing, but with "skip connections" directly connecting encoder layers to decoder layers.
- The skip connections give the network the option of bypassing the encoding/decoding part so that share the low-level information between the input and output.

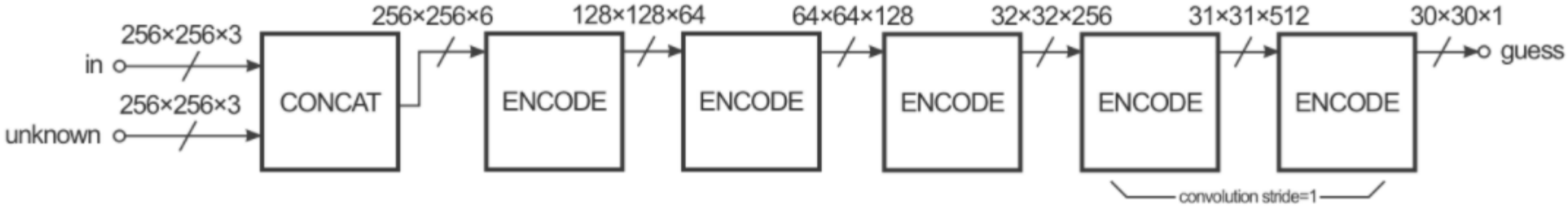




https://affinelayer.com/pix2pix/

Discriminator with PatchGAN

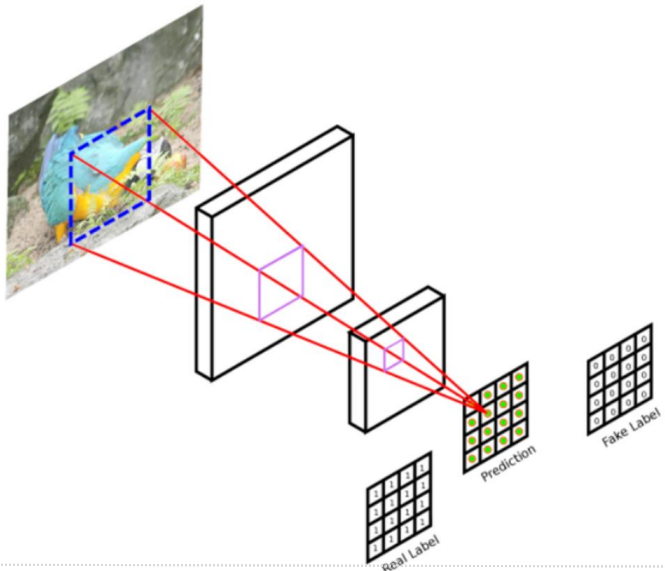
- The Discriminator has the job of taking two images, an input image and an unknown image (which will be either a target or output image from the generator), and deciding if the second image was produced by the generator or not.



https://affinelayer.com/pix2pix/

Discriminator with PatchGAN

- The Discriminator concentrate on high-frequency structure (relying on an L1 term to force low-frequency correctness), it is sufficient to restrict our attention to the structure in local image patches.
- Discriminator tries to classify if each N ×N patch in an image is real or fake. We run this discriminator convolutionally across the image, averaging all responses to provide the ultimate output of D.



- Advantage of PatchGAN

  - A smaller PatchGAN has fewer parameters

  - Can be applied to arbitrarily large images

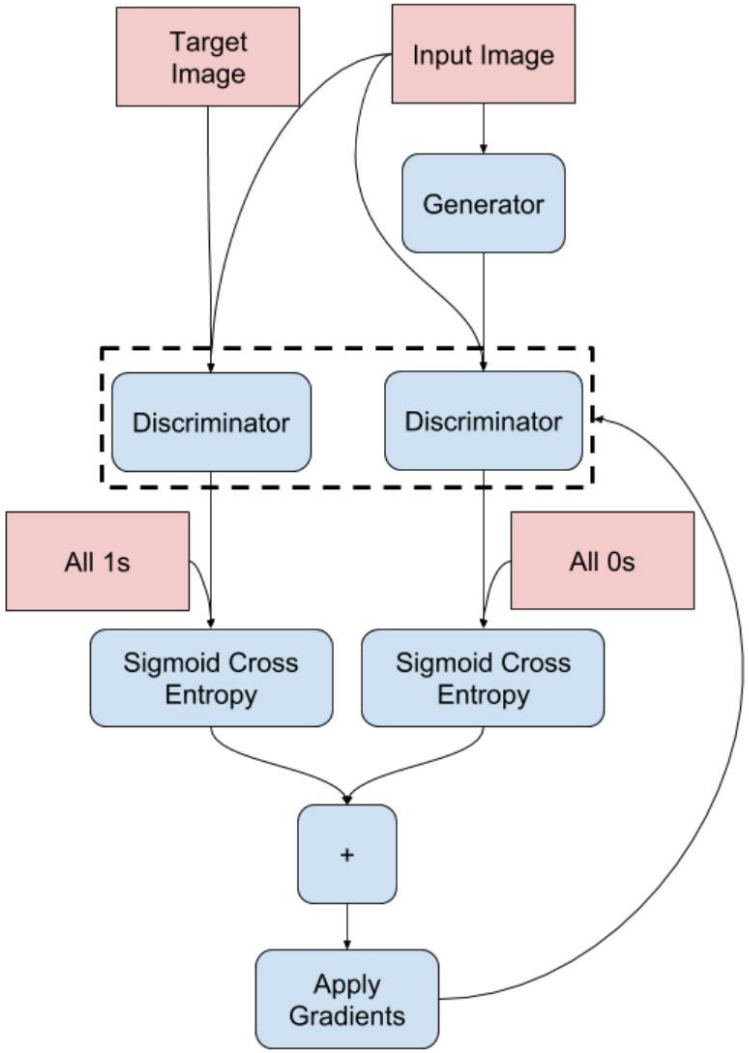
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=laonple&logNo=221366130381

Optimization and inference

- Alternate between one gradient descent step on D, then one step on G. D-> G -> D -> G…
- Train to maximize $\log D(x, G(x, z))$ rather than minimize $\log(1 - D(x, G(x, z))$
- Use **minibatch SGD** and apply **the Adam solver** with a **learning rate of 0.0002**, and **momentum parameters** $\beta_1 = 0.5, \beta_2 = 0.999$. (Check this out in later code)
- Divide the objective by 2 while optimizing D, which slows down the rate at which D learns relative to G.
- At inference time, run the generator net in exactly the same manner as during the training phase.
- In experiments, use batch sizes between 1 and 10 depending on the experiment.

Optimization and inference

- Discriminator

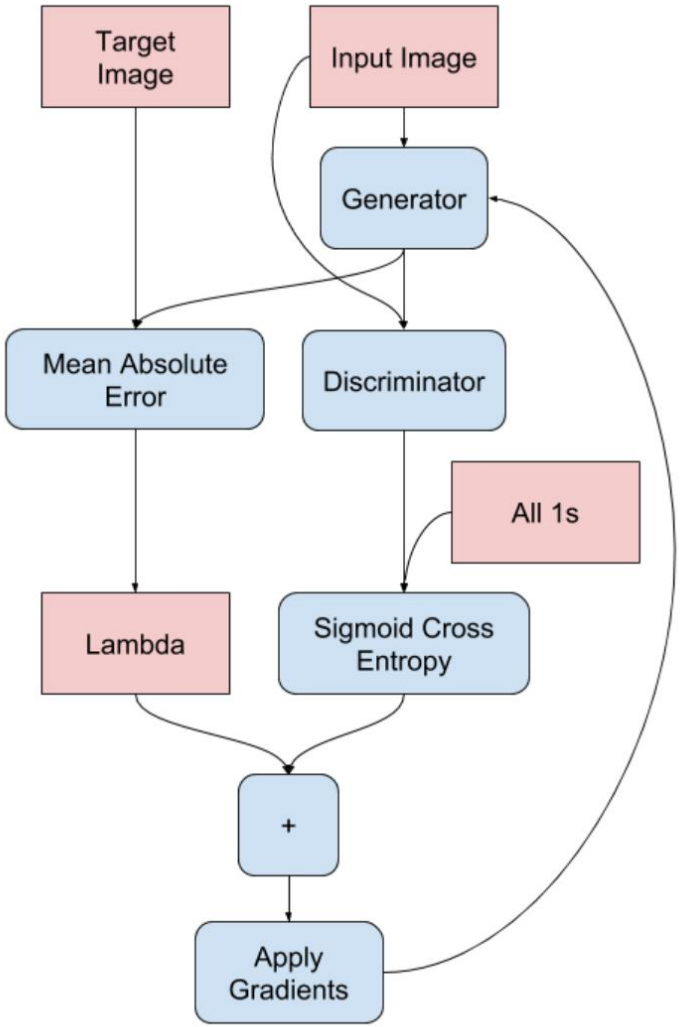


- The discriminator looks at the input/target pair and the input/output(which Generator create) pair and produces its guess about how realistic they look.
- The update parameter by using correct guesses (which discriminator guesses input/target as true OR vise versa)

https://www.tensorflow.org/tutorials/generative/pix2pix

Optimization and inference

- Generator



- The generator model is trained via the discriminator model. It is updated to minimize the loss predicted by the discriminator for generated images marked as “real.”
- The generator is also updated to minimize the L1 loss or mean absolute error between the generated image and the target image.

https://www.tensorflow.org/tutorials/generative/pix2pix

Experiments with source code

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Load Image

- In this tutorial, we will use the so-called “maps” dataset used in the Pix2Pix paper. This is a dataset comprised of satellite images of New York and their corresponding Google maps pages. Each image is 1,200 pixels wide and 600 pixels tall and contains both the satellite image on the left and the Google maps image on the right.

Download dataset : http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/maps.tar.gz

Load Image

- We can prepare this dataset for training a Pix2Pix GAN model in Keras. We will just work with the images in the training dataset. Each image will be loaded, rescaled, and split into the satellite and Google map elements. The result will be 1,097 color image pairs with the width and height of 256×256 pixels.

[1]
Copying

```
from os import listdir # Returns a list containing the names of the entries in the directory given by path
from numpy import asarray # Convert the input to an array
from numpy import vstack # Stack arrays in sequence vertically (row wise)
from keras.preprocessing.image import img_to_array # Converts a PIL Image instance to a Numpy array
from keras.preprocessing.image import load_img # Loads an image into PIL format
from numpy import savez_compressed # Save several arrays into a single file in compressed .npz format
def load_images(path, size=(256,512)):
  src_list, tar_list = list(), list()
  for filename in listdir(path):
    pixels = load_img(path + filename, target_size=size)
    pixels = img_to_array(pixels)
    sat_img, map_img = pixels[:, :256], pixels[:, 256:]
    src_list.append(sat_img)
    tar_list.append(map_img)
  return [asarray(src_list), asarray(tar_list)]
path = 'maps/train/'
[src_images, tar_images] = load_images(path)
print('Loaded: ', src_images.shape, tar_images.shape)
filename = 'maps_256.npz'
print('Saved dataset: ', filename)
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Load Image

- This function enumerates the list of images in a given directory, loads each with the target size of 256×512 pixels, splits each image into satellite and map elements and returns an array of each.

```
def load_images(path, size=(256,512)):
  src_list, tar_list = list(), list()
  for filename in listdir(path):
    pixels = load_img(path + filename, target_size=size)
    pixels = img_to_array(pixels)
    sat_img, map_img = pixels[:, :256], pixels[:, 256:]
    src_list.append(sat_img)
    tar_list.append(map_img)
  return [asarray(src_list), asarray(tar_list)]
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Load Image

- Running the example loads all images in the training dataset, summarizes their shape to ensure the images were loaded correctly, then saves the arrays to a new file called maps_256.npz in compressed NumPy array format.

```
path = 'maps/train/'
[src_images, tar_images] = load_images(path)
print('Loaded: ', src_images.shape, tar_images.shape)
filename = 'maps_256.npz'
print('Saved dataset: ', filename)
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Discriminator

[2]
Copying

```
from numpy import load
from numpy import zeros
from numpy import ones
from numpy.random import randint
from keras.optimizers import Adam
from keras.initializers import RandomNormal
from keras.models import Model
from keras.models import Input
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Activation
from keras.layers import Concatenate
from keras.layers import Dropout
from keras.layers import BatchNormalization
from keras.layers import LeakyReLU
from matplotlib import pyplot
```

[3]
Copying

```
def define_discriminator(image_shape):
  init = RandomNormal(stddev=0.02)
  in_src_image = Input(shape=image_shape)
  in_target_image = Input(shape=image_shape)
  merged = Concatenate()([in_src_image, in_target_image])
  d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
  d = LeakyReLU(alpha=0.2)(d)
  d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
  patch_out = Activation('sigmoid')(d)
  model = Model([in_src_image, in_target_image], patch_out)
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
  return model
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Discriminator

- Model weights were initialized via random Gaussian with a mean of 0.0 and standard deviation of 0.02. Images input to the model are 256×256.

```
init = RandomNormal(stddev=0.02)
```

- The model takes two images as input, specifically a source and a target image. These images are concatenated together at the channel level, e.g. 3 color channels of each image become 6 channels of the input.

```
in_src_image = Input(shape=image_shape)
in_target_image = Input(shape=image_shape)
merged = Concatenate()([in_src_image, in_target_image])
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Discriminator

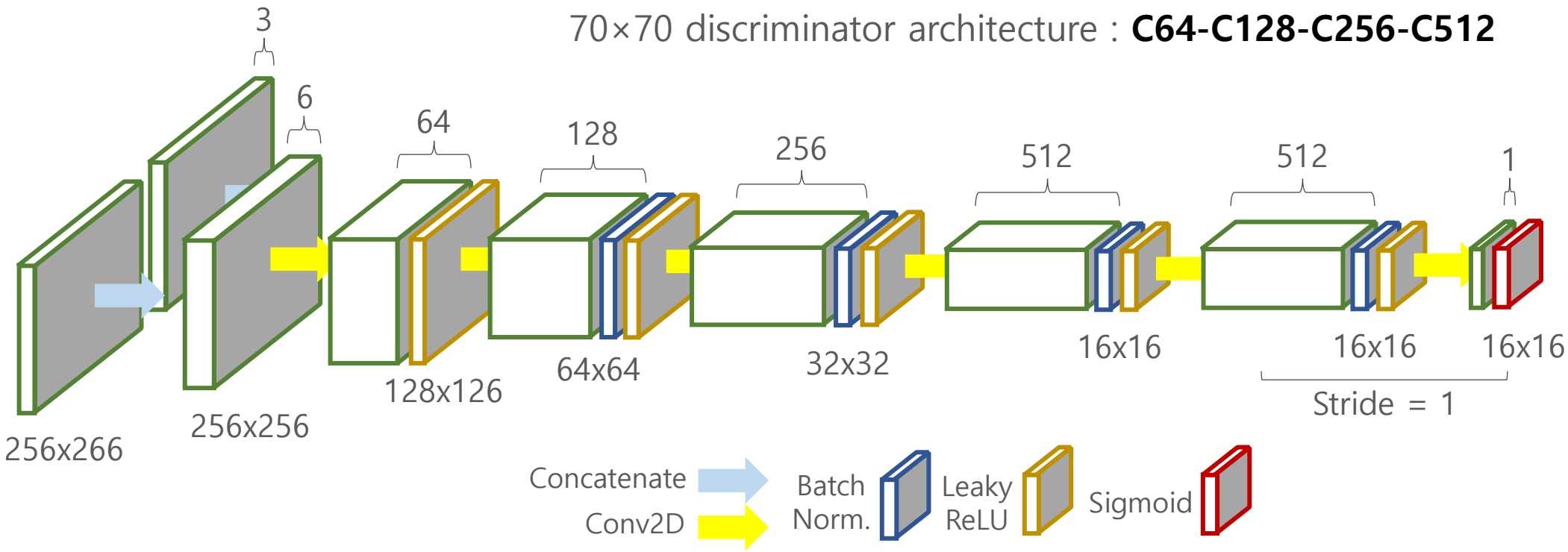
- A convolution is applied after the last layer to map to a **1-dimensional** output, followed by a **Sigmoid function**. BatchNorm is not applied to the first C64 layer. All **ReLUs** are leaky, with slope **0.2**. The kernel size is fixed at **4×4** and a stride of **2×2** is used on all but the last 2 layers of the model.

```
d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
d = LeakyReLU(alpha=0.2)(d)
d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
d = BatchNormalization()(d)
d = LeakyReLU(alpha=0.2)(d)
d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
d = BatchNormalization()(d)
d = LeakyReLU(alpha=0.2)(d)
d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
d = BatchNormalization()(d)
d = LeakyReLU(alpha=0.2)(d)
d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
d = BatchNormalization()(d)
d = LeakyReLU(alpha=0.2)(d)
d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
patch_out = Activation('sigmoid')(d)
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Discriminator

- The 70×70 discriminator architecture is defined using a shorthand notation as: C64-C128-C256-C512, where C refers to a block of Convolution-BatchNorm-LeakyReLU layers and the number indicates the number of filters.



Define Discriminator

- The model is trained with a batch size of one image and the Adam version of stochastic gradient descent is used with a small learning range and modest momentum. The loss for the discriminator is weighted by 50% for each model update.

```
model = Model([in_src_image, in_target_image], patch_out)
opt = Adam(lr=0.0002, beta_1=0.5)
model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
return model
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Generator(encoder_block)

- Encoder_block captures the context of an image.
- Encoder_block contains previous layer, number of filters, batchnorm(boolean) as parameter
- Encoder_block is made up with (Conv-> Batchnorm-> Leaky ReLU)

[4]
Copying

```
def define_encoder_block(layer_in, n_filters, batchnorm=True):
  init = RandomNormal(stddev=0.02)
  g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
  if batchnorm:
    g = BatchNormalization()(g, training=True)
  g = LeakyReLU(alpha=0.2)(g)
  return g
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Generator(decorder_block)

- Decoder_block is Opposition of encoder_block, it localizes feature map
- Similar parameters with encoder, but additionally use skip connection layer and dropout(boolean)
- Decoder_block is made of with (Transposed Conv-> Batchnorm-> Dropout (first 3 blocks) -> ReLU)

[5]
Copying

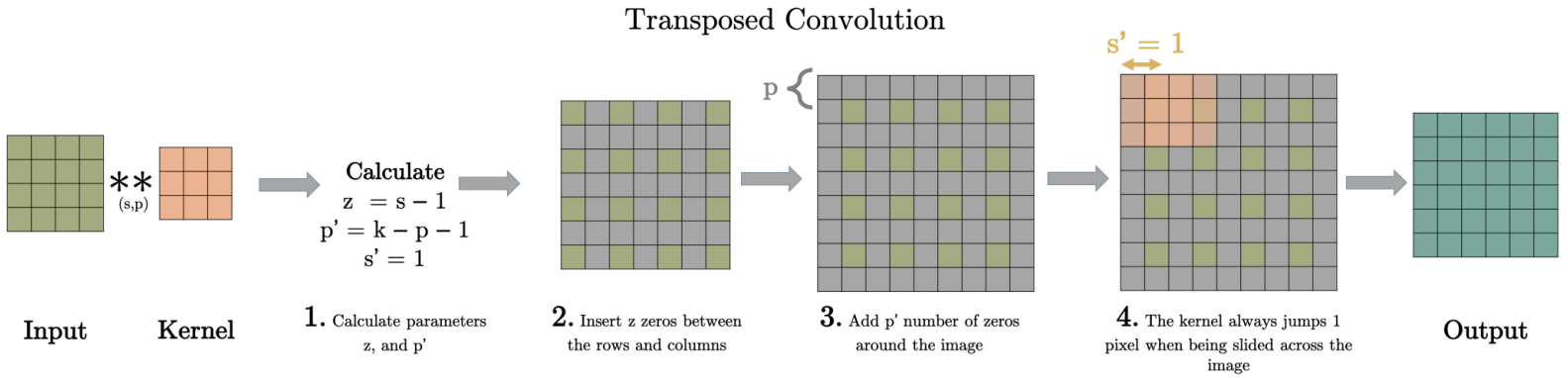
```
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
  init = RandomNormal(stddev=0.02)
  g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
  g = BatchNormalization()(g, training=True)
  if dropout:
    g = Dropout(0.5)(g, training=True)
  g = Concatenate()([g, skip_in])
  g = Activation('relu')(g)
  return g
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Generator(decorder_block)

- Conv2DTranspsose func work this way!!

```
g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
```



- For U-net skip architecture concatenate up-conv feature map and feature map in encoding stage

```
g = Concatenate()([g, skip_in])
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Generator

- Implement generator using encoder and decoder block
- Generator outputs pair of in_image, out_image

[6]
Copying

```
def define_generator(image_shape=(256,256,3)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # image input
  in_image = Input(shape=image_shape)
  # encoder model
  e1 = define_encoder_block(in_image, 64, batchnorm=False)
  e2 = define_encoder_block(e1, 128)
  e3 = define_encoder_block(e2, 256)
  e4 = define_encoder_block(e3, 512)
  e5 = define_encoder_block(e4, 512)
  e6 = define_encoder_block(e5, 512)
  e7 = define_encoder_block(e6, 512)
  # bottleneck, no batch norm and relu
  b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
  b = Activation('relu')(b)
  # decoder model
  d1 = decoder_block(b, e7, 512)
  d2 = decoder_block(d1, e6, 512)
  d3 = decoder_block(d2, e5, 512)
  d4 = decoder_block(d3, e4, 512, dropout=False)
  d5 = decoder_block(d4, e3, 256, dropout=False)
  d6 = decoder_block(d5, e2, 128, dropout=False)
  d7 = decoder_block(d6, e1, 64, dropout=False)
  # output
  g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
  out_image = Activation('tanh')(g)
  # define model
  model = Model(in_image, out_image)
  return model
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define GAN

- The define_gan() function below implements this, taking the already-defined generator and discriminator models as arguments and using the Keras functional API to connect them together into a composite model.
- Both loss functions are specified for the two outputs of the model and the weights used for each are specified in the loss_weights argument to the compile() function.

[7]
Copying

```
def define_gan(g_model, d_model, image_shape):
  for layer in d_model.layers:
    if not isinstance(layer, BatchNormalization):
      layer.trainable = False
  in_src = Input(shape=image_shape)
  gen_out = g_model(in_src)
  dis_out = d_model([in_src, gen_out])
  model = Model(in_src, [dis_out, gen_out])
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=[1,100])
  return model
```

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define generate_real_samples

- The generate_real_samples() function below will prepare a batch of random pairs of images from the training dataset, and the corresponding discriminator label of class=1 to indicate they are real.

[8]
Copying

```
def generate_real_samples(dataset, n_samples, patch_shape):
  trainA, trainB = dataset
  ix = randint(0, trainA.shape[0], n_samples)
  X1, X2 = trainA[ix], trainB[ix]
  # generate 'real' class labels (1)
  y = ones((n_samples, patch_shape, patch_shape, 1))
  return [X1, X2], y
```

- trainA = dataset[0]
- tranB = dataset[1]
- TrainA.shape = [1096, 256, 256, 3]
- n_samples = number of select images

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define generate_fake_samples

- The generate_fake_samples() function below uses the generator model and a batch of real source images to generate an equivalent batch of target images for the discriminator.

[9]
Copying

```
def generate_fake_samples(g_model, samples, patch_shape):
  X = g_model.predict(samples)
  # create 'fake' class labels (0)
  y = zeros((len(X), patch_shape, patch_shape, 1))
  return X, y
```

- samples = value of return 'generate_real_samples' X1
- That means create fake samples from real samples

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Train

- The *train()* function below implements this, taking the defined generator, discriminator, composite model, and loaded datas et as input. The number of epochs is set at 100 to keep training times down, although 200 was used in the paper. A batch size of 1 is used as is recommended in the paper.
- Training involves a fixed number of training iterations. There are 1,097 images in the training dataset. One epoch is one ite ration through this number of examples, with a batch size of one means 1,097 training steps. The generator is saved and e valuated every 10 epochs or every 10,970 training steps, and the model will run for 100 epochs, or a total of 109,700 traini ng steps.

[10]
Copying

```
def train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch=1):
  n_patch = d_model.output_shape[1]
  trainA, trainB = dataset
  bat_per_epo = int(len(trainA) / n_batch)
  n_steps = bat_per_epo * n_epochs
  for i in range(n_steps):
    [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
    X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
    d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
    d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
    g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
    print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss))
    if (i+1) % (bat_per_epo * 10) == 0:
      summarize_performance(i, g_model, dataset)
```

```
# unpack dataset
trainA, trainB = dataset
# calculate the number of batches per training epoch
bat_per_epo = int(len(trainA) / n_batch)
# calculate the number of training iterations
n_steps = bat_per_epo * n_epochs
```

- d.model.output_shape : (None, 16, 16, 1) -> n_patch = 16 왜..?
- trainA, trainB = dataset -> dataset[0] is sat_img , dataset[1] is map_im

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Define Train

[11]
Copying

```
for i in range(n_steps):
  # select a batch of real samples
  [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
  # generate a batch of fake samples
  X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
  # update discriminator for real samples
  d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
  # update discriminator for generated samples
  d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
  # update the generator
  g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
  # summarize performance
  print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss))
```

- d_loss1 = gap of target and source -> real pair
- d_loss2 = gap of fake_target(create by generator) and source -> fake pair
- Source : 위성 사진
- Target : 지도 사진
- Fake target : generator가 source로 부터 생성한 fake

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/

Play train

[12]
Copying

```
# load image data
dataset = load_real_samples('maps_256.npz')
print('Loaded', dataset[0].shape, dataset[1].shape)
# define input shape based on the loaded dataset
image_shape = dataset[0].shape[1:]
# define the models
d_model = define_discriminator(image_shape)
g_model = define_generator(image_shape)
# define the composite model
gan_model = define_gan(g_model, d_model, image_shape)
# train model
train(d_model, g_model, gan_model, dataset)
```

```
>1082, d1[0.006] d2[0.048] g[15.425]
>1083, d1[0.508] d2[0.151] g[8.600]
>1084, d1[0.165] d2[0.123] g[10.576]
>1085, d1[0.055] d2[0.172] g[9.696]
>1086, d1[0.030] d2[0.212] g[9.571]
>1087, d1[0.059] d2[0.084] g[21.962]
>1088, d1[0.009] d2[0.060] g[11.934]
>1089, d1[0.247] d2[0.317] g[7.782]
>1090, d1[0.001] d2[0.122] g[16.818]
>1091, d1[0.002] d2[0.041] g[15.183]
>1092, d1[0.422] d2[0.227] g[13.804]
>1093, d1[0.001] d2[0.044] g[14.583]
>1094, d1[0.002] d2[0.055] g[17.279]
>1095, d1[0.371] d2[1.282] g[17.911]
>1096, d1[0.498] d2[0.315] g[12.592]
WARNING:tensorflow:Compiled the loaded model, but the cc
l you train or evaluate the model.
>Saved: plot_001096.png and model_001096.h5
```

This example epochs = 1 , n_patch =1

https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/