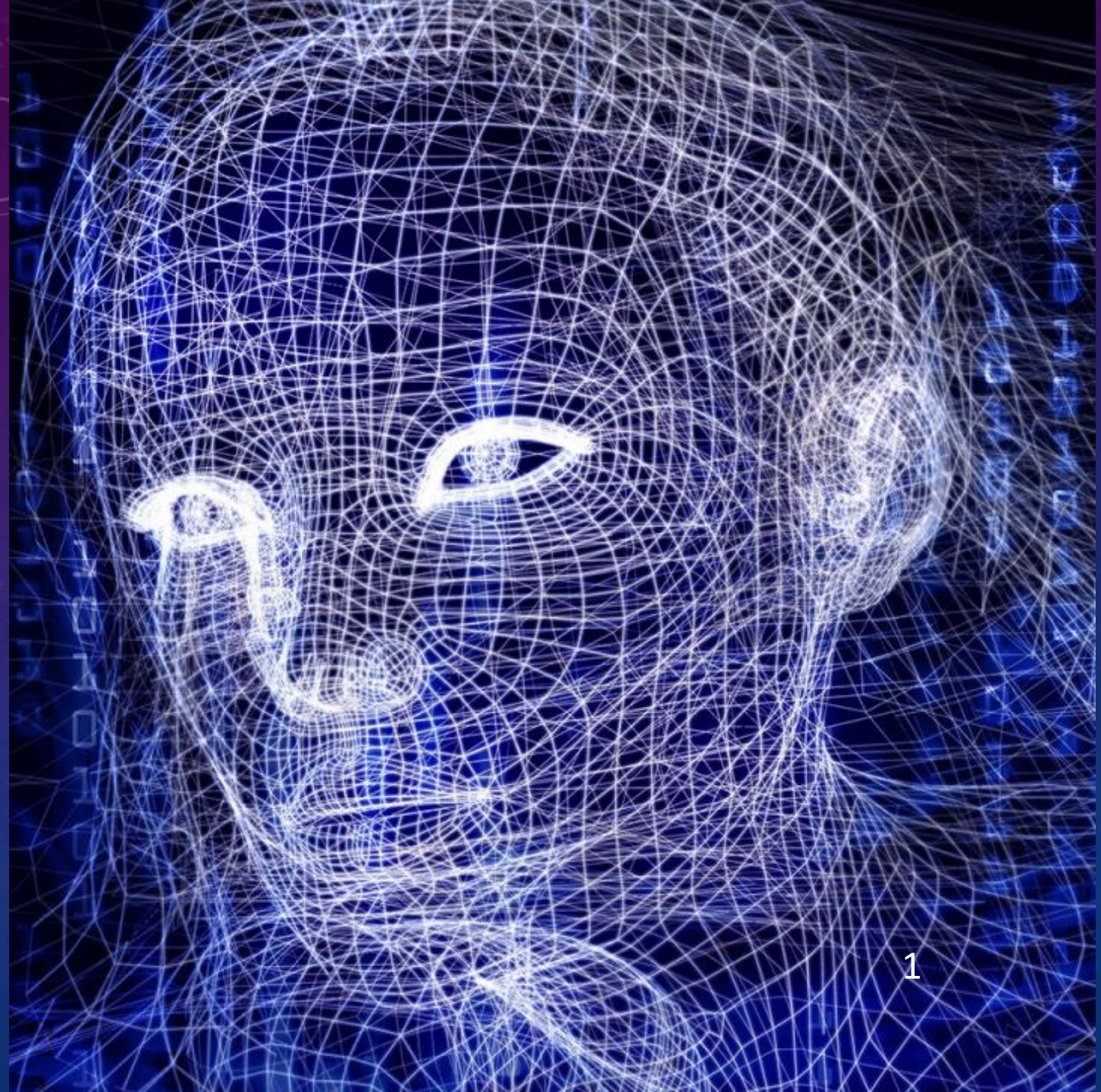
Lecture Series for Computer Vision

Chapter 5
Conditional GAN

徐繼聖

Gee-Sern Jison Hsu

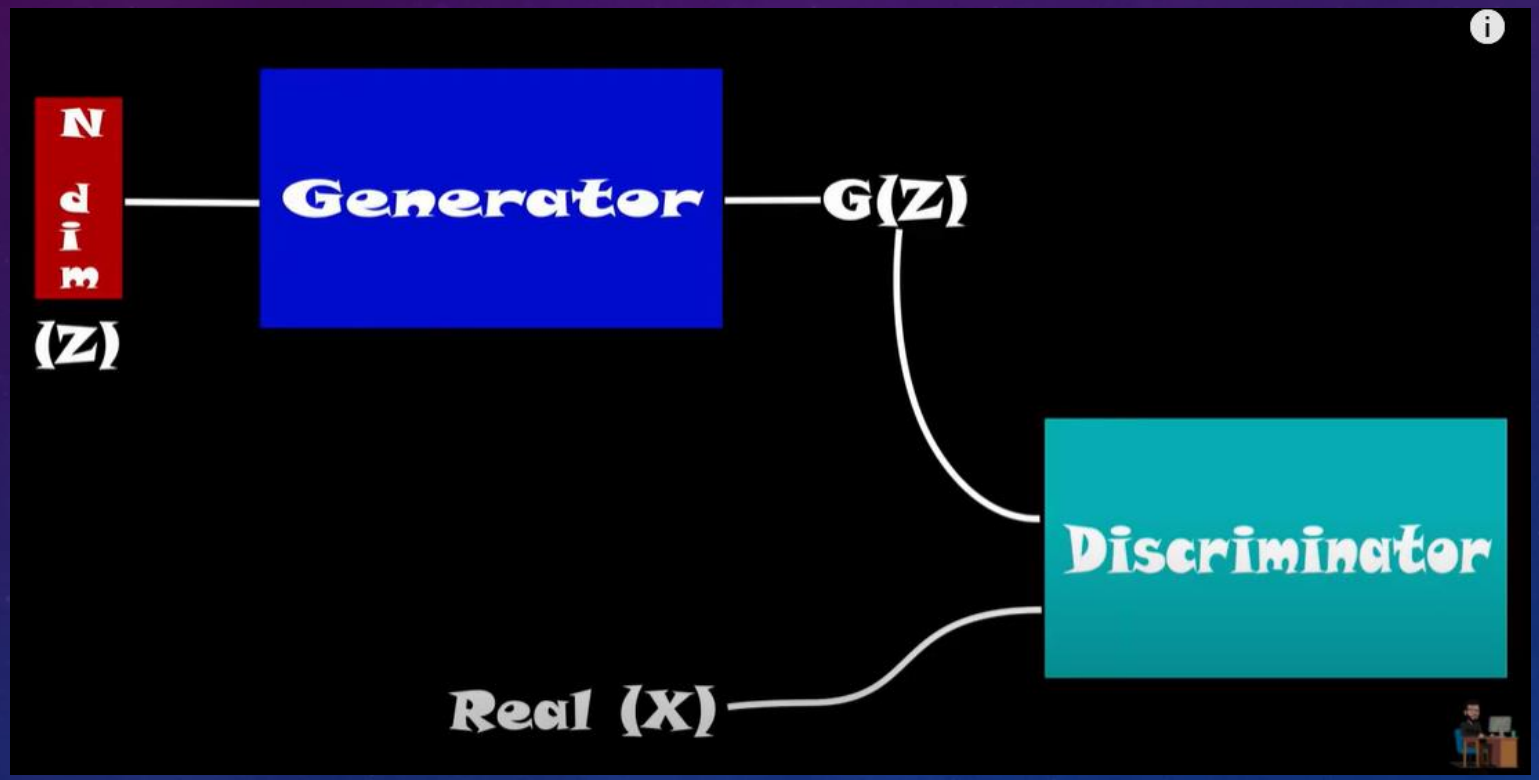
National Taiwan University of Science and Technology



CONDITIONAL GANS

Paper : Conditional Generative Adversarial Nets
Authors : Mirza, Mehdi, and Simon Osindero
Released : ArXiv 2014

CONDITIONAL GENERATE ADVERSARIAL NETWORK || CONDITIONAL GAN || CGAN || GAN



https://www.youtube.com/watch?v=71Wg3GQGyx0&ab_channel=DevelopersHutt [11:40]

GAN LECTURE 2 (2018): CONDITIONAL GENERATION

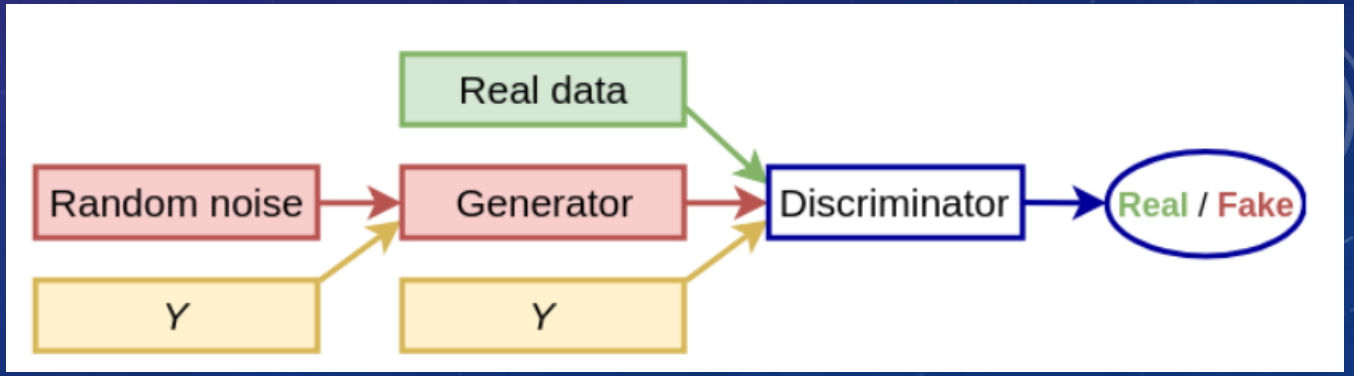
https://www.youtube.com/watch?v=LpyL4nZSuqU&ab_channel=Hung-yiLee [26:31]

What is Conditional GANs (C-GANs)?

- Conditional GANs (C-GANs) are an extension of the GANs model.
- CGANs are allowed to generate images that have **certain** conditions or attributes.

Like DCGANs, **Conditional GANs** also ahs two components:

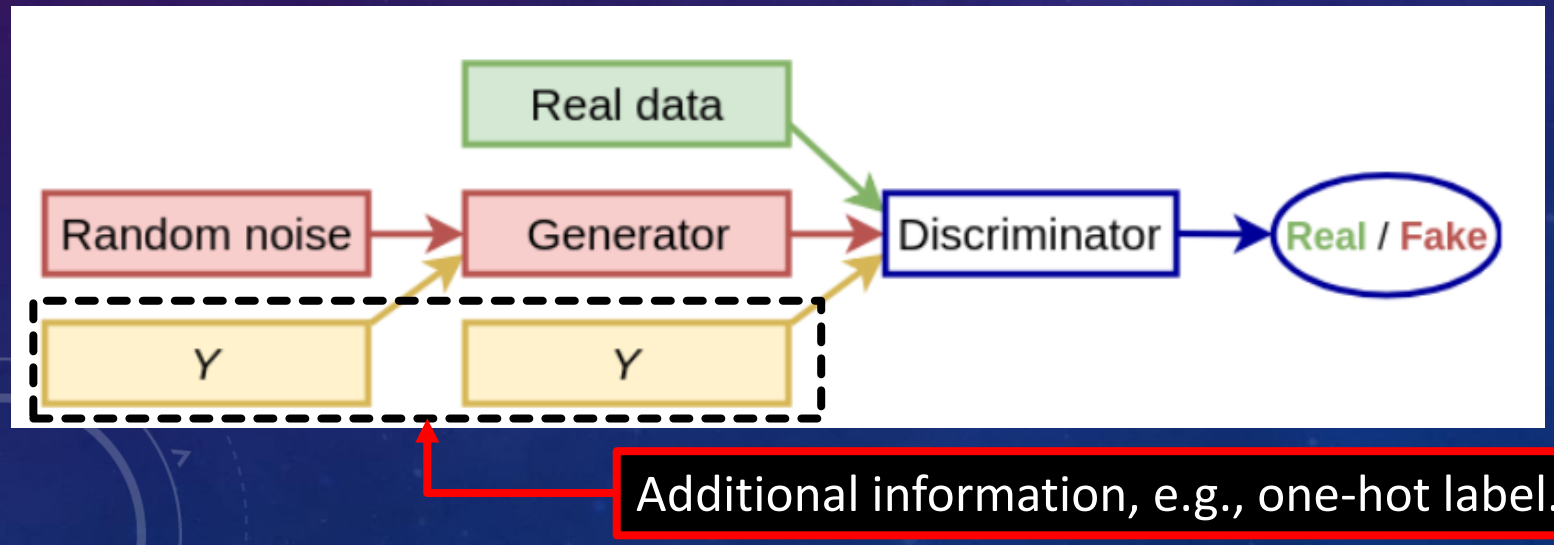
- A Generator Neural Network.
- A Discriminator neural Network.



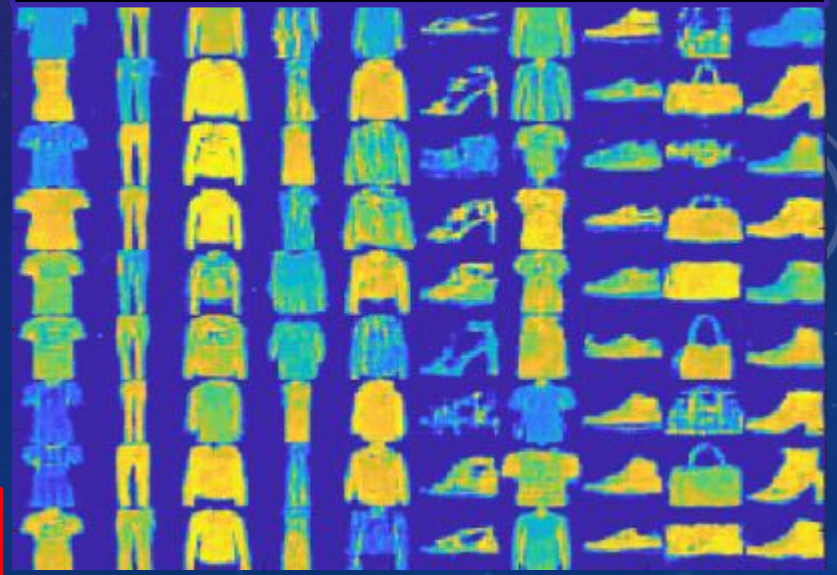
[Ref] https://medium.datadriveninvestor.com/an-introduction-to-conditional-gans-cgans-727d1f5bb011

What is Conditional GANs (C-GANs)?

- GAN whose Generate and Discriminator are conditioned during training by using **additional information**.
- This additional information could be, in theory, anything such as a class label, a set of tags, or even written description.

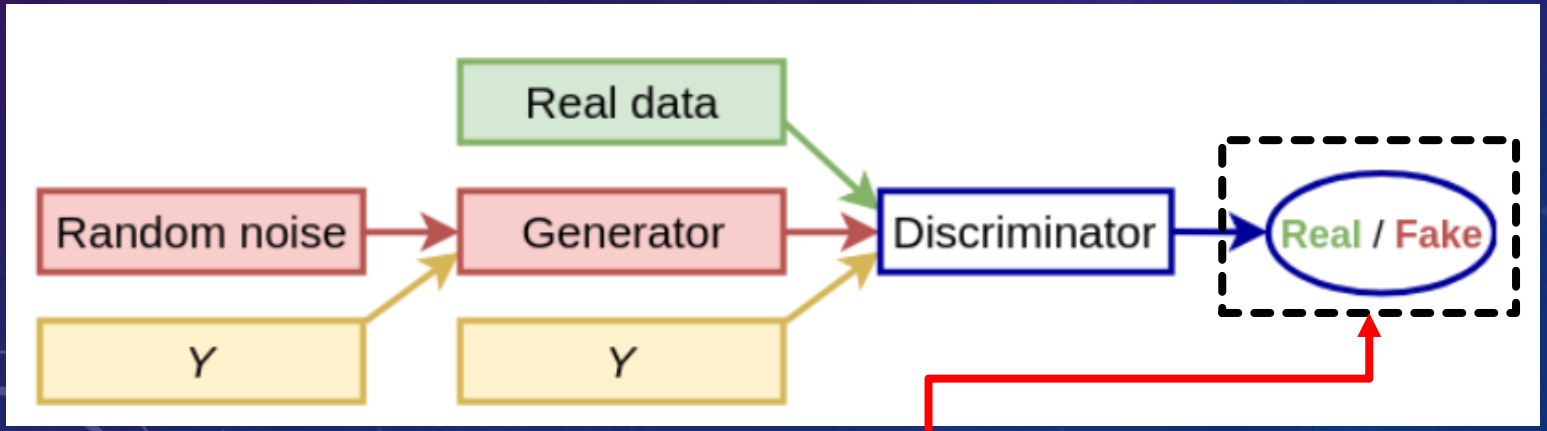


FashionMNIST generated by CGAN



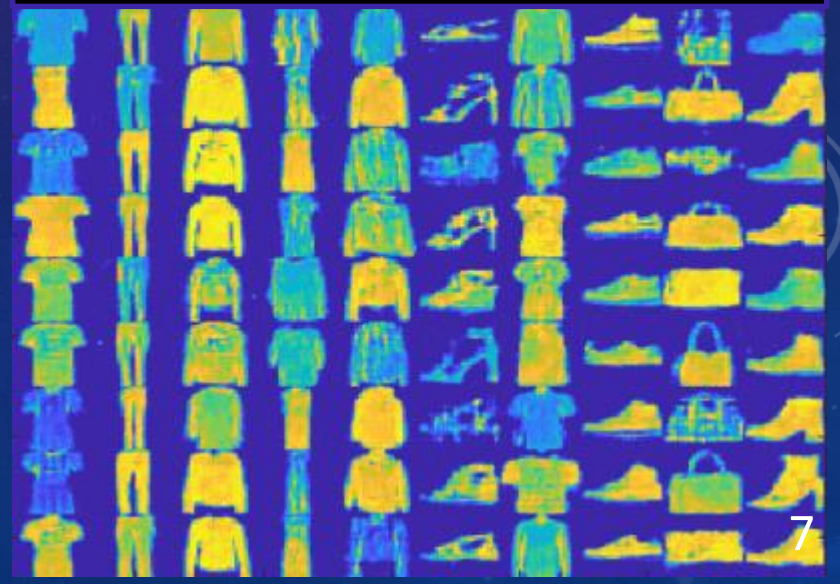
What is Conditional GANs (C-GANs)?

- The Generator learns to produce realistic examples for each label in the training dataset, and the Discriminator learns to **distinguish fake example-labeled pairs from real example-labeled pairs**.
- It learns to accept real, matching pairs while rejecting pairs that are mismatched and pairs in which an example is fake.



Real/Fake "Label" conditioned by additional information.

FashionMNIST generated by CGAN

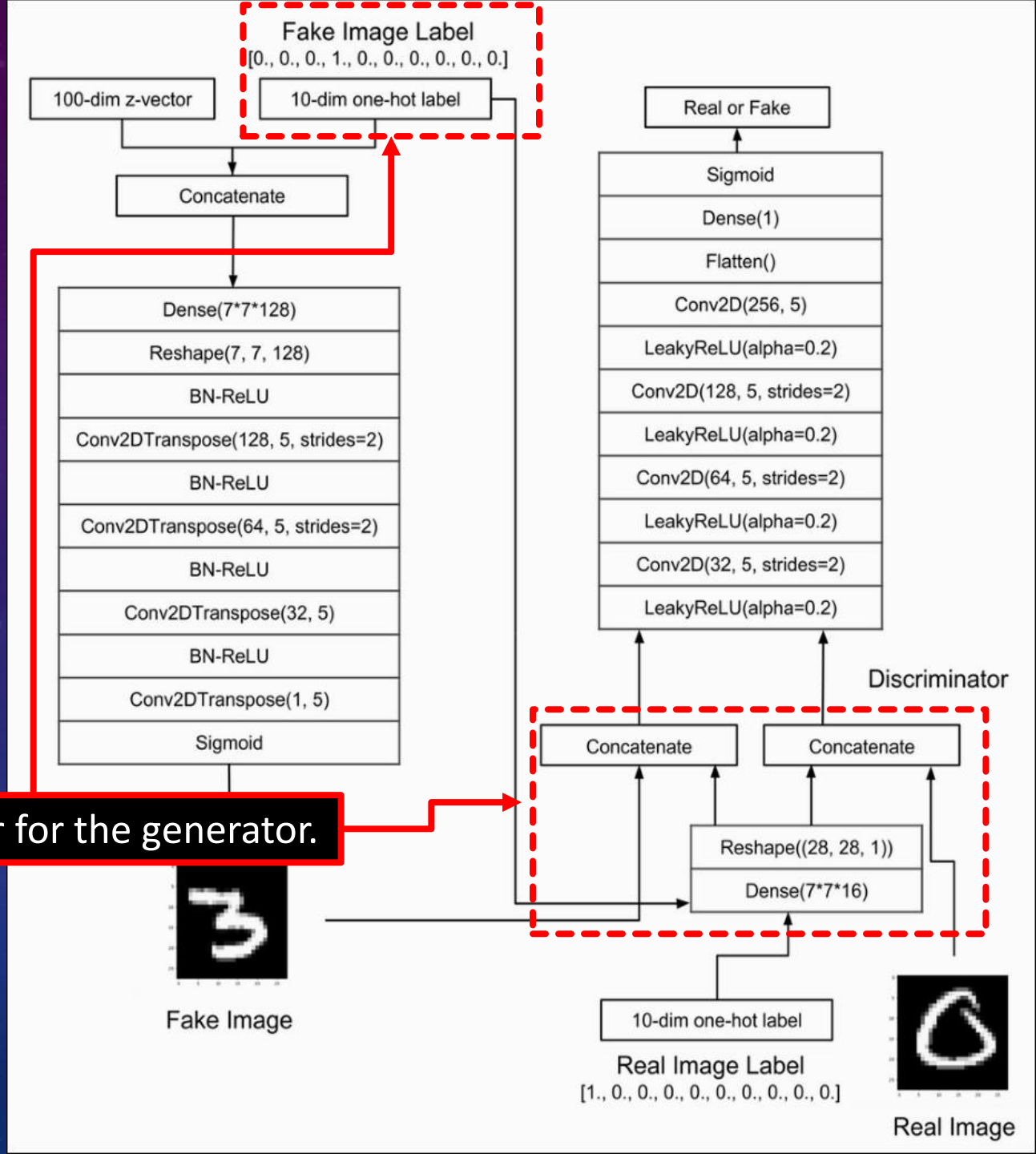


CGAN's Architecture Diagram

Take MNIST as example (10 classes)

The Discriminator's Network (Right)

- The CGAN Discriminator's model is similar to DCGAN Discriminator's model, **except for the one-hot vector.**
- The one-hot vector, which is used to **condition** Discriminator o

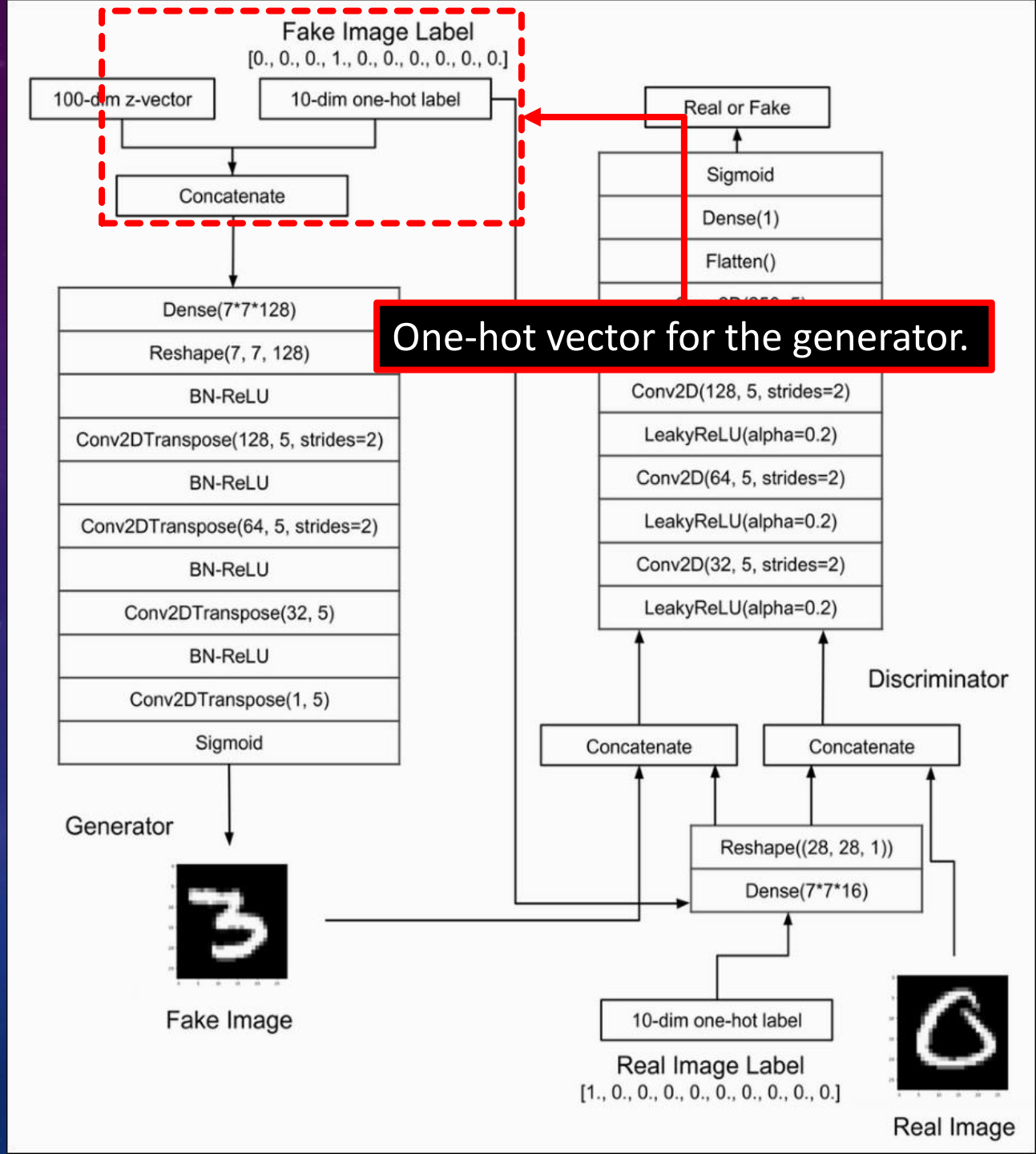


CGAN's Architecture Diagram

Take MNIST as example (10 classes)

The Generator's Network (Left)

- The CGAN Generator's model is also similar to DCGAN Discriminator's model, **except for the one-hot vector.**
- The one-hot vector, which is used to **condition** Generator outputs.



Loss Function (Discriminator)

The Discriminator has two task

- **Discriminator** has to correctly label real images which are coming from training data set as “real”.
- **Discriminator** has to correctly label generated images which are coming from **Generator** as “fake”

The sum of the “fake” and “real” images losses is the overall Discriminator loss, which can be described as follows:

$$\mathcal{L}^{(D)}(\theta^G, \theta^D) = -\mathbb{E}_{x\sim p_{data}} log\mathcal{D}(x|y) - \mathbb{E}_z(1 - \mathcal{D}(\mathcal{G}(z|y')))$$

$\mathcal{D}$	Discriminator	$\mathcal{G}$	Generator
x	Input Image	y	Conditional one-hot vector
z	Input noise of $\mathcal{G}$	y'	Assigned one-hot vector

Loss Function (Generator)

The Generator has one task

- To create an image that looks as “real” as possible to fool **Discriminator**.

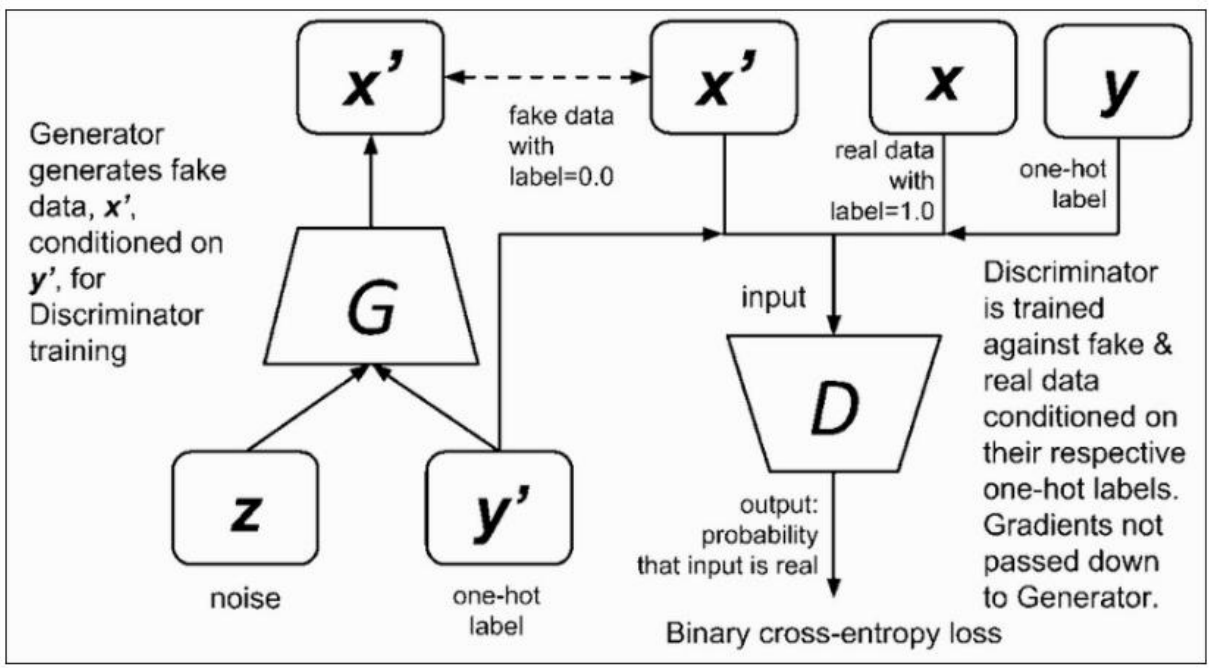
The loss function of the Generator minimizes the correct prediction of the Discriminator on fake images conditioned on the specified one-hot labels

$$\mathcal{L}^{(G)}(\theta^G, \theta^D) = -\mathbb{E}_z \mathcal{D}\big(\mathcal{G}(z|y')\big)$$

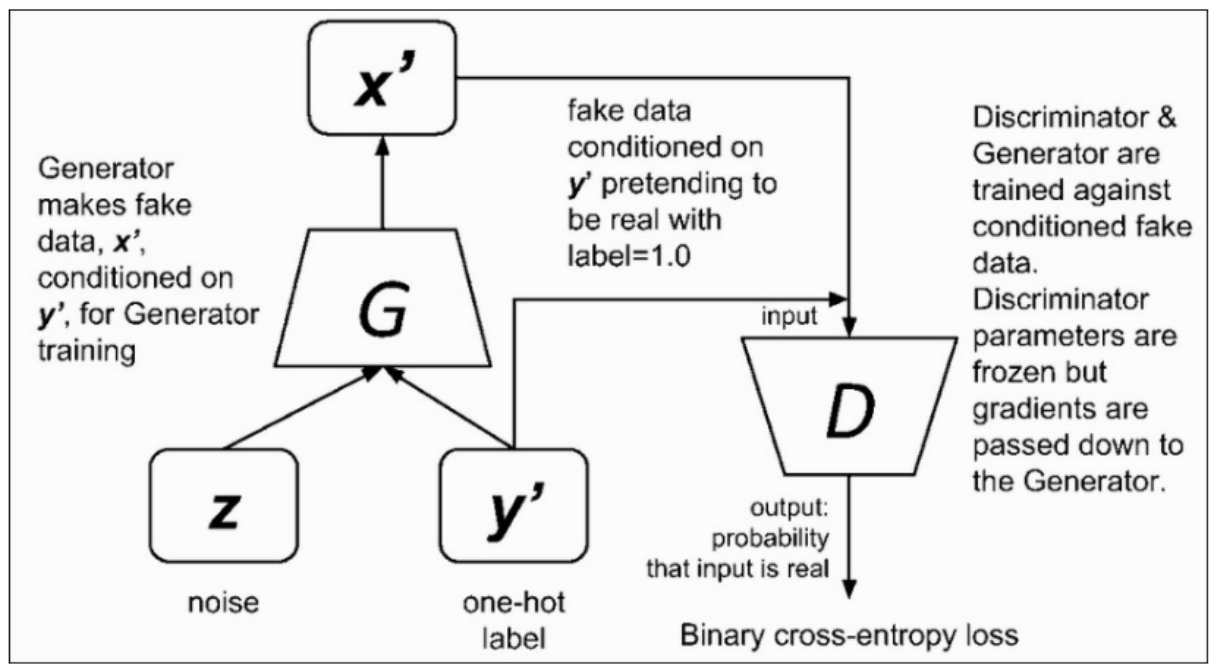
$\mathcal{D}$	Discriminator	$\mathcal{G}$	Generator
z	Input noise of $\mathcal{G}$	y'	Assigned one-hot vector

Training procedure of C-GANs

Discriminator's training procedure



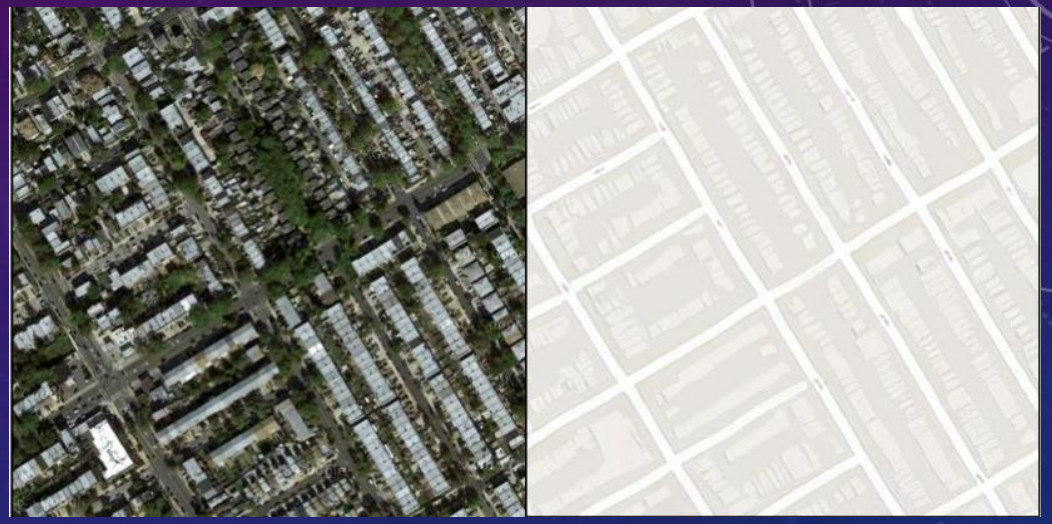
Generator's training procedure



Samples

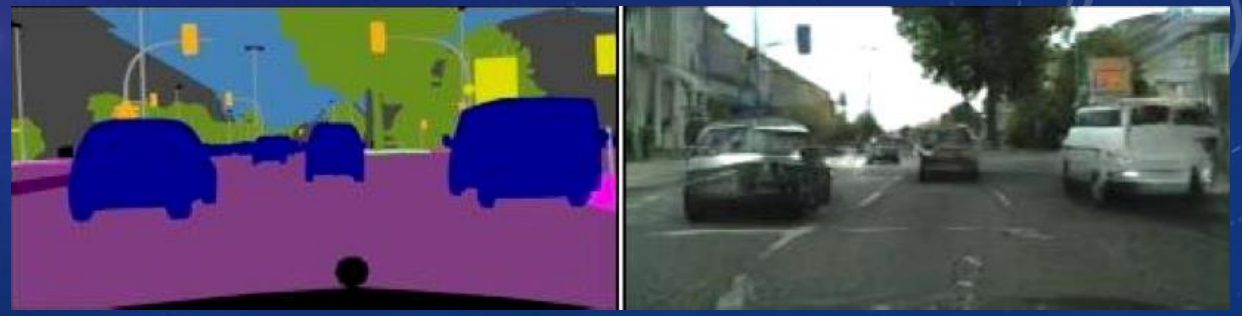
Day to Night

Aerial to Map



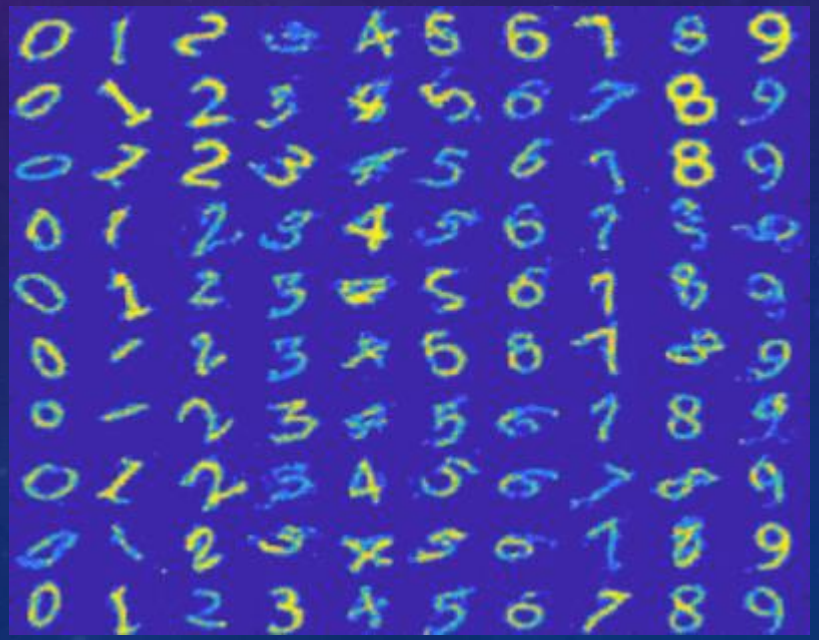
BW to Color

Labels to Street Scene



Example 5.1 - CGAN

- Please download the “exercise_5.1_CGAN.ipynb” from Moodle.
- Upload the Notebook to GoogleColab.
- In the following pages, we will introduce the CGAN structure, show you how to train the CGAN model, and visualize the results with the specified labels.



Example 5.1 - CGAN

```
# training parameters
batch_size = 128
lr = 0.0002
train_epoch = 10

# data_loader
img_size = 32
transform = transforms.Compose([
        transforms.Resize(img_size),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5])
        ])
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=True, download=True, transform=transform),
    batch_size=batch_size, shuffle=True)
```

Define the hyperparameters

Download the Mnist dataset to the folder ‘./data’

Example 5.1 - CGAN

```
G = generator(128)
D = discriminator(128)
```

Build the instance of our classes generator and discriminator model

```
G.weight_init(mean=0.0, std=0.02)
D.weight_init(mean=0.0, std=0.02)
G.cuda()
D.cuda()
```

Weight initialization

```
BCE_loss = nn.BCELoss()
```

“Binary cross-entropy” as loss function

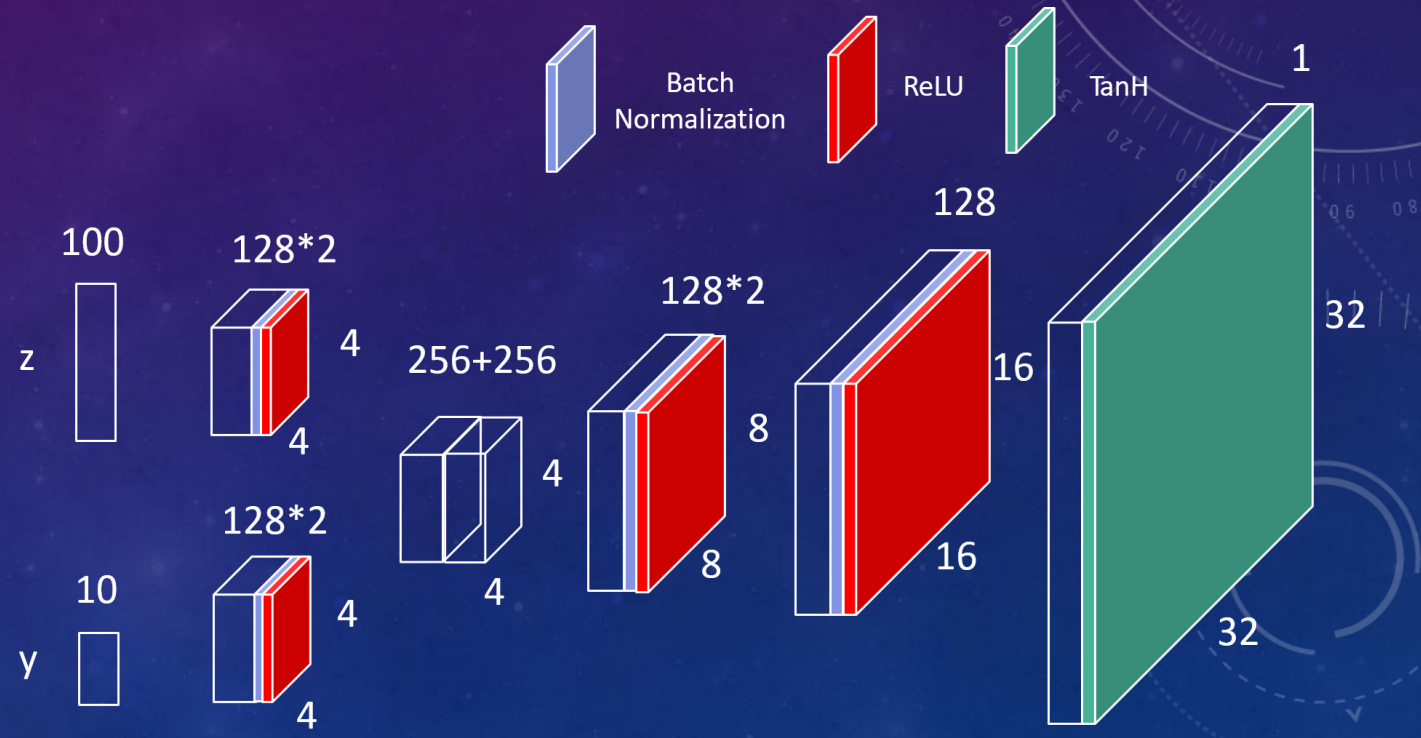
```
G_optimizer = optim.Adam(G.parameters(), lr=lr, betas=(0.5, 0.999))
D_optimizer = optim.Adam(D.parameters(), lr=lr, betas=(0.5, 0.999))
```

Define the “Adam” as a optimizer

Example 5.1 - Generator in the CGAN

The network structure of the generator in Example 5.1

Input Information	
Input Noise (z)	Dimension = 100
Input Label (y)	Dimension = 10
Output Information	
Output	Dimension = 32×32 (FashionMNIST)



Example 5.1 - CGAN

```
class generator(nn.Module):
    def__init__(self, d=128):
        super(generator, self).__init__()
        self.deconv1_1 = nn.ConvTranspose2d(100, d*2, 4, 1, 0)
        self.deconv1_1_bn = nn.BatchNorm2d(d*2)
        self.deconv1_2 = nn.ConvTranspose2d(10, d*2, 4, 1, 0)
        self.deconv1_2_bn = nn.BatchNorm2d(d*2)
        self.deconv2 = nn.ConvTranspose2d(d*4, d*2, 4, 2, 1)
        self.deconv2_bn = nn.BatchNorm2d(d*2)
        self.deconv3 = nn.ConvTranspose2d(d*2, d, 4, 2, 1)
        self.deconv3_bn = nn.BatchNorm2d(d)
        self.deconv4 = nn.ConvTranspose2d(d, 1, 4, 2, 1)
```

Define the architecture

```
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)
```

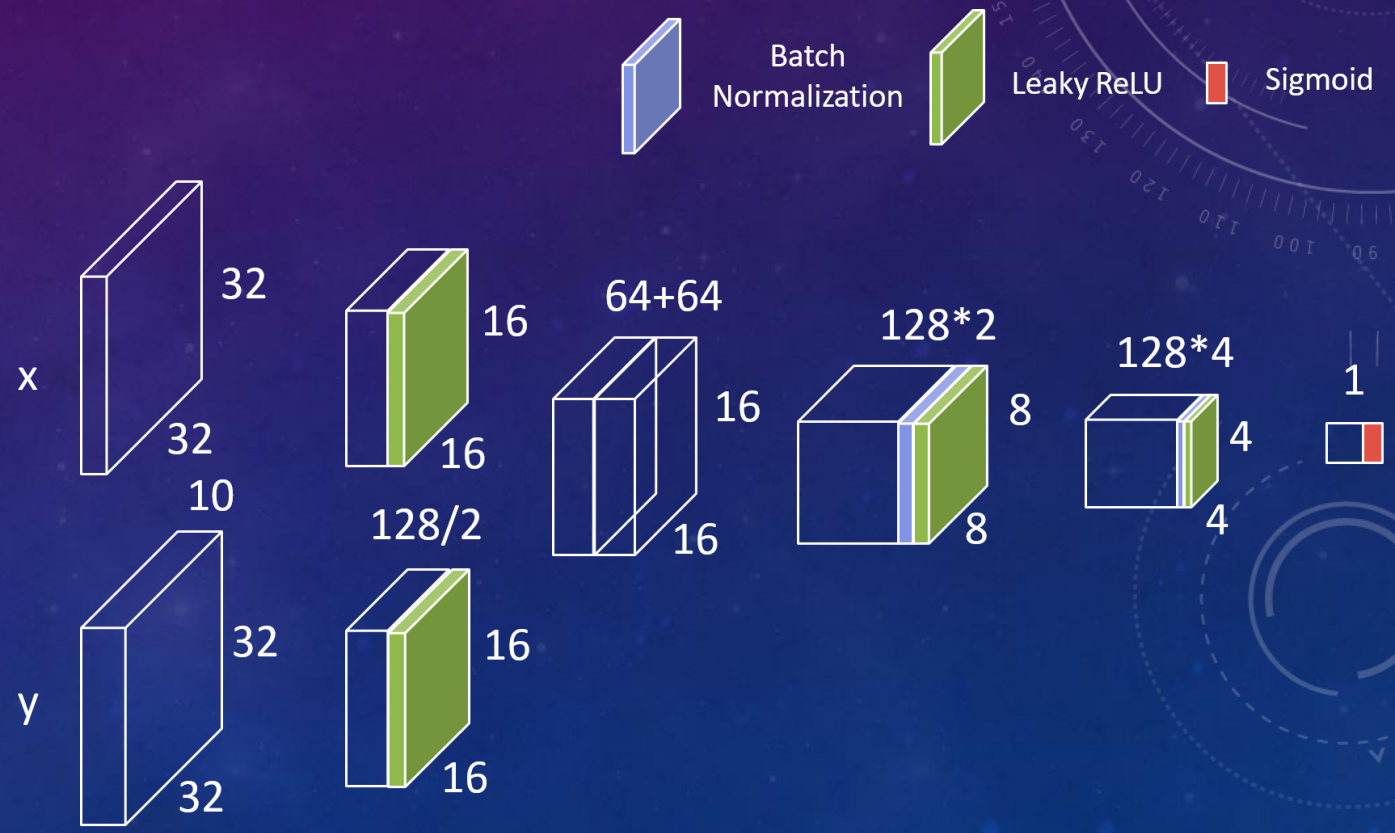
```
def forward(self, input, label):
        x = F.relu(self.deconv1_1_bn(self.deconv1_1(input)))
        y = F.relu(self.deconv1_2_bn(self.deconv1_2(label)))
        x = torch.cat([x, y], 1)
        x = F.relu(self.deconv2_bn(self.deconv2(x)))
        x = F.relu(self.deconv3_bn(self.deconv3(x)))
        x = F.tanh(self.deconv4(x))

        return x
```

Forward pass

Example 5.1 - Discriminator in the CGAN

The network structure of the Discriminator in Example 5.1

Input Information	
Input Image (x)	Dimension = 32×32
Input Label (y)	Dimension = 32×32×10
Output Information	
Output	Dimension = 1



Example 5.1 - CGAN

```
class discriminator(nn.Module):
    def__init__(self, d=128):
        super(discriminator, self).__init__()
        self.conv1_1 = nn.Conv2d(1, d//2, 4, 2, 1)
        self.conv1_2 = nn.Conv2d(10, d//2, 4, 2, 1)
        self.conv2 = nn.Conv2d(d, d*2, 4, 2, 1)
        self.conv2_bn = nn.BatchNorm2d(d*2)
        self.conv3 = nn.Conv2d(d*2, d*4, 4, 2, 1)
        self.conv3_bn = nn.BatchNorm2d(d*4)
        self.conv4 = nn.Conv2d(d * 4, 1, 4, 1, 0)
```

Define the architecture

```
    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)
```

```
def forward(self, input, label):
    x = F.leaky_relu(self.conv1_1(input), 0.2)
    y = F.leaky_relu(self.conv1_2(label), 0.2)
    x = torch.cat([x, y], 1)
    x = F.leaky_relu(self.conv2_bn(self.conv2(x)), 0.2)
    x = F.leaky_relu(self.conv3_bn(self.conv3(x)), 0.2)
    x = F.sigmoid(self.conv4(x))

    return x
```

Forward pass

Example 5.1 - CGAN

Define the fixed noise

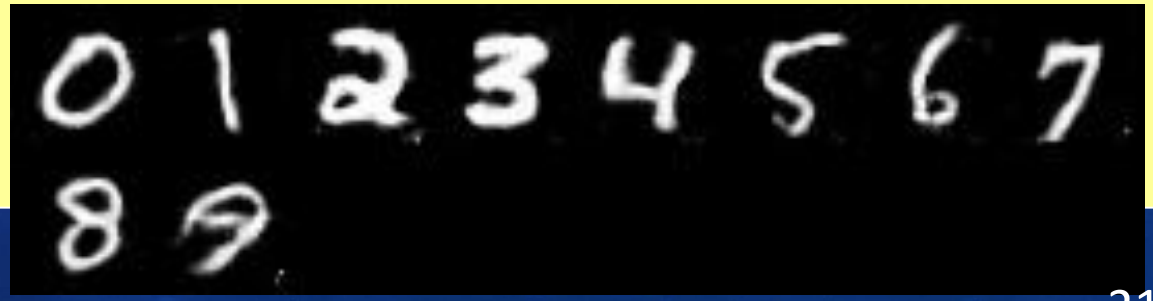
```
# fixed noise & label
fixed _z_2 = torch.randn(10, 100)
fixed_z_2 = fixed_z_2.view(-1, 100, 1, 1)
```

Define the label from 0 to 9

```
one_hot = [[1,0,0,0,0,0,0,0,0,0],
           [0,1,0,0,0,0,0,0,0,0],
           [0,0,1,0,0,0,0,0,0,0],
           [0,0,0,1,0,0,0,0,0,0],
           [0,0,0,0,1,0,0,0,0,0],
           [0,0,0,0,0,1,0,0,0,0],
           [0,0,0,0,0,0,1,0,0,0],
           [0,0,0,0,0,0,0,1,0,0],
           [0,0,0,0,0,0,0,0,1,0],
           [0,0,0,0,0,0,0,0,0,1]]
```

```
one_hot_arr = np.array(one_hot)
fixed_y_label_2 = torch.from_numpy(one_hot_arr)

fixed_y_label_2 = fixed_y_label_2.view(-1, 10, 1, 1)

fixed_y_label_2 = fixed_y_label_2.float()
fixed_z_2 = fixed_z_2.float()

fixed_z_2, fixed_y_label_2 = Variable(fixed_z_2.cuda()),
Variable(fixed_y_label_2.cuda())
```

Result :



Pix2Pix- a conditional GAN for cross-domain knowledge

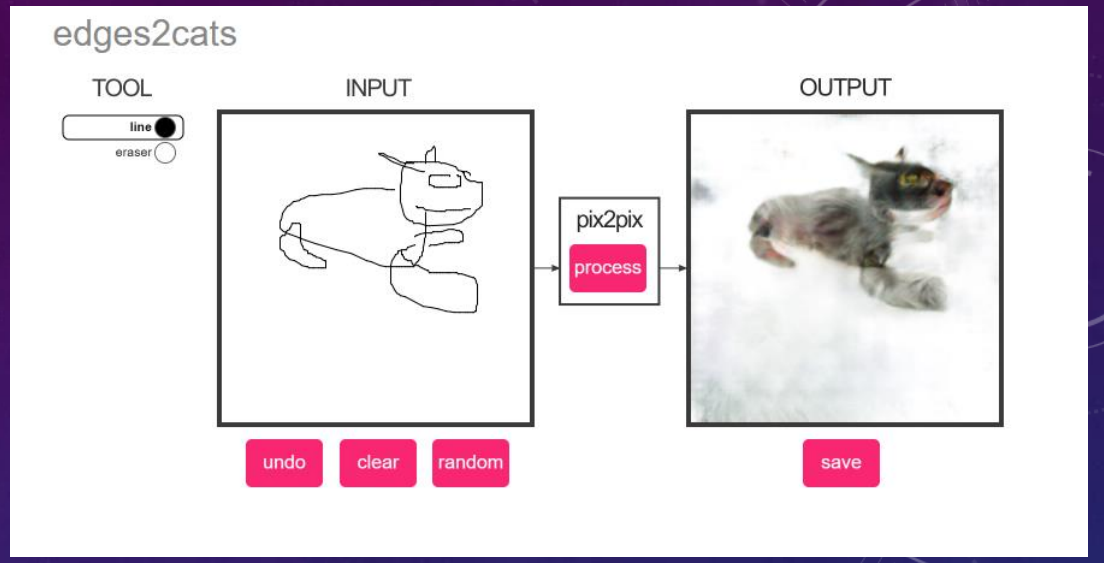
Paper: Image-to-Image Translation with Conditional Adversarial Nets
Authors: Isola et al.
Released: CVPR 2017

Pix2Pix Explained

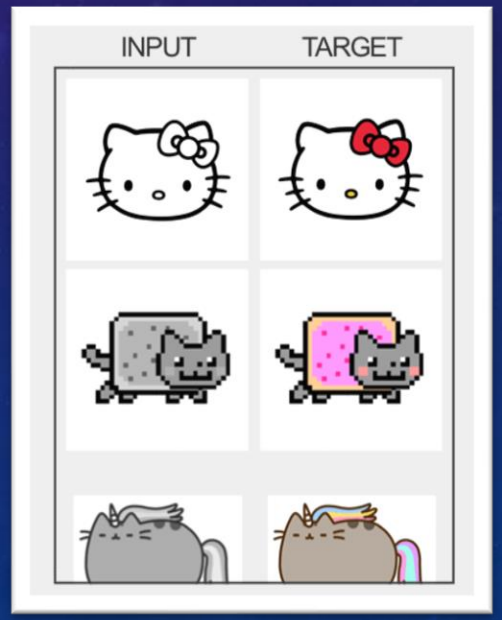
https://www.youtube.com/watch?v=u7kQ5lNfUfg [4:27]

Pix2Pix

- Image-conditional GAN:
  - ➢ allows large images compared to prior GANs
  - ➢ Performs well on a variety of tasks
- An example of a dataset (Upper) would be that the input image is a black and white picture and the target image is the color version of the picture.
- Another example (Lower) is show from the authors demo. We input edges, that transform to a cat. Try it out.



https://affinelayer.com/pixsrv/



https://phillipi.github.io/pix2pix/

Pix2Pix – Examples

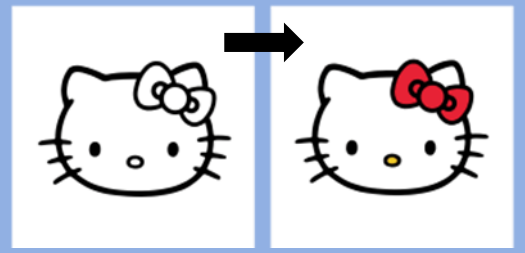
- Map to Aerial
- Day to Night
- Edges to Handbags

Input

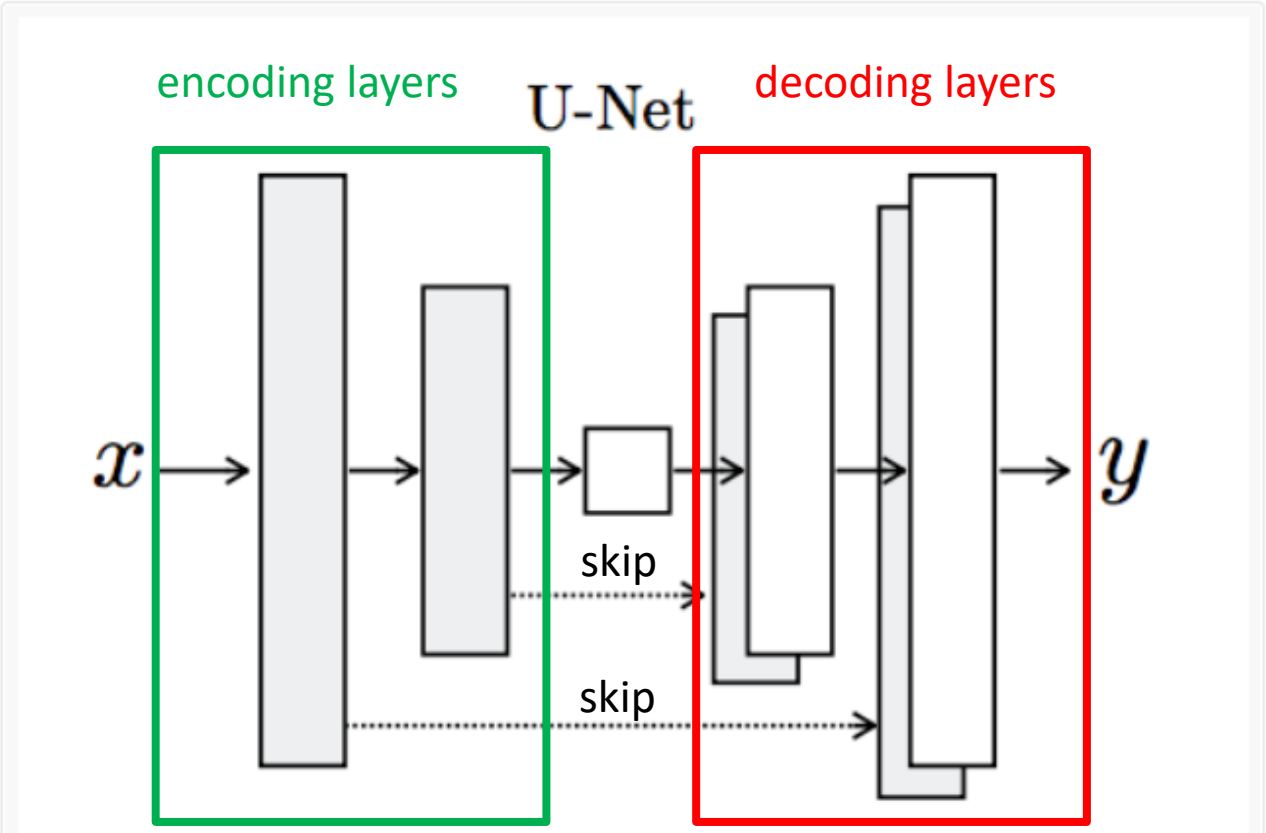
Output

Hello Kitty

activation



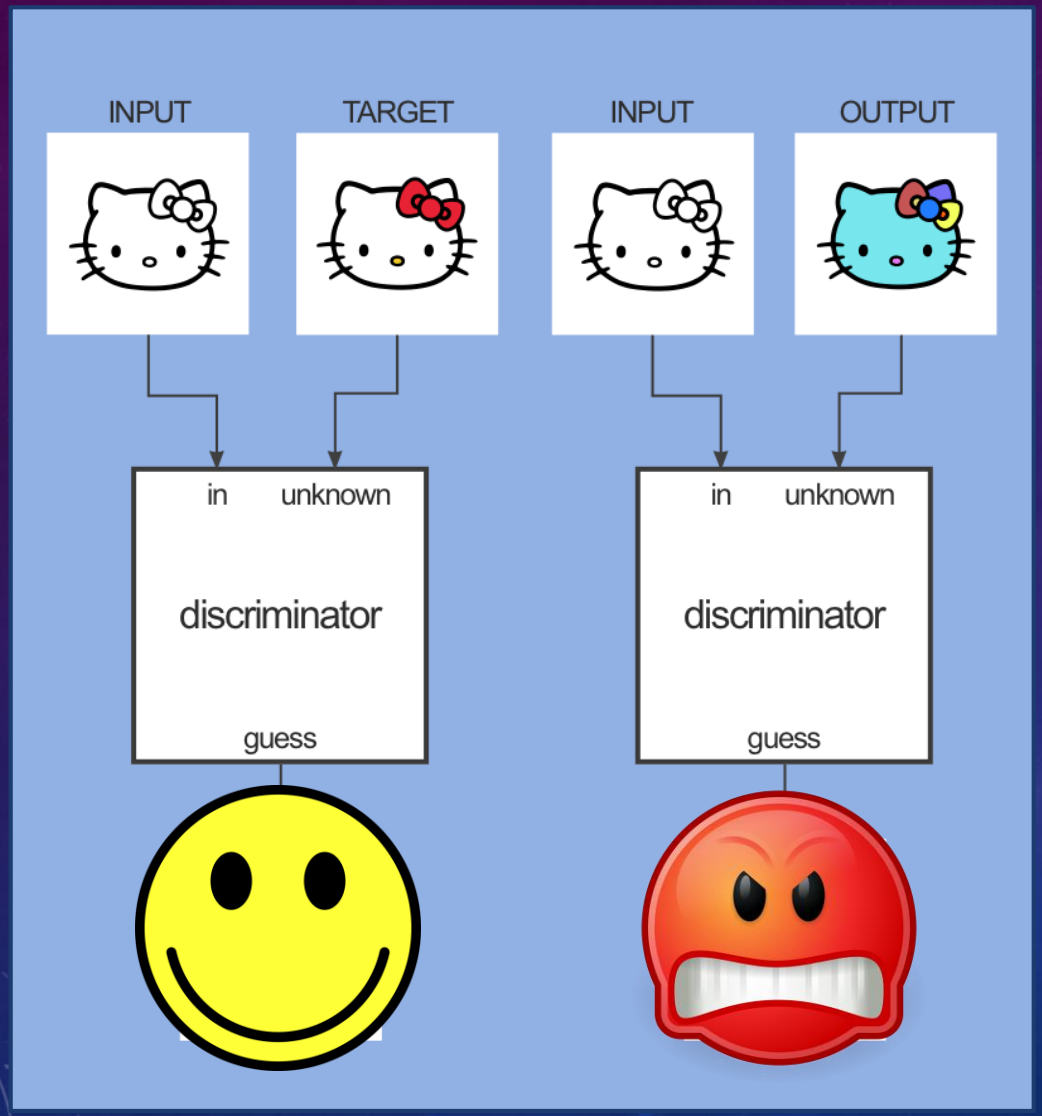
Pix2Pix – Generator is a Encoder-Decoder



Architecture of the U-Net Generator Model
Taken from Image-to-Image Translation With Conditional Adversarial Networks

- [Data] Source/Target sets to define the characteristics of the generated images.
- [Structure] U-net represents the structure which looks like the U-like shape.
- The dotted arrows represent the skip connections that feed the feature layers from the encoding layers to the decoding layers

Pix2Pix – Discriminator



- [Data] Source/Target sets to define the characteristics of the generated images.
- [Structure] U-net represents the structure which looks like the U-like shape.
- The dotted arrows represent the skip connections that feed the feature layers from the encoding layers to the decoding layers

Pix2Pix – Generator has to Satisfy Two Conditions

- There are two losses involved in training the generator
  - G tries to minimize this objective against an adversarial D that tries to maximize

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

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

  - Using L1 distance rather than L2 as L1 encourages less blurring

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1]$$

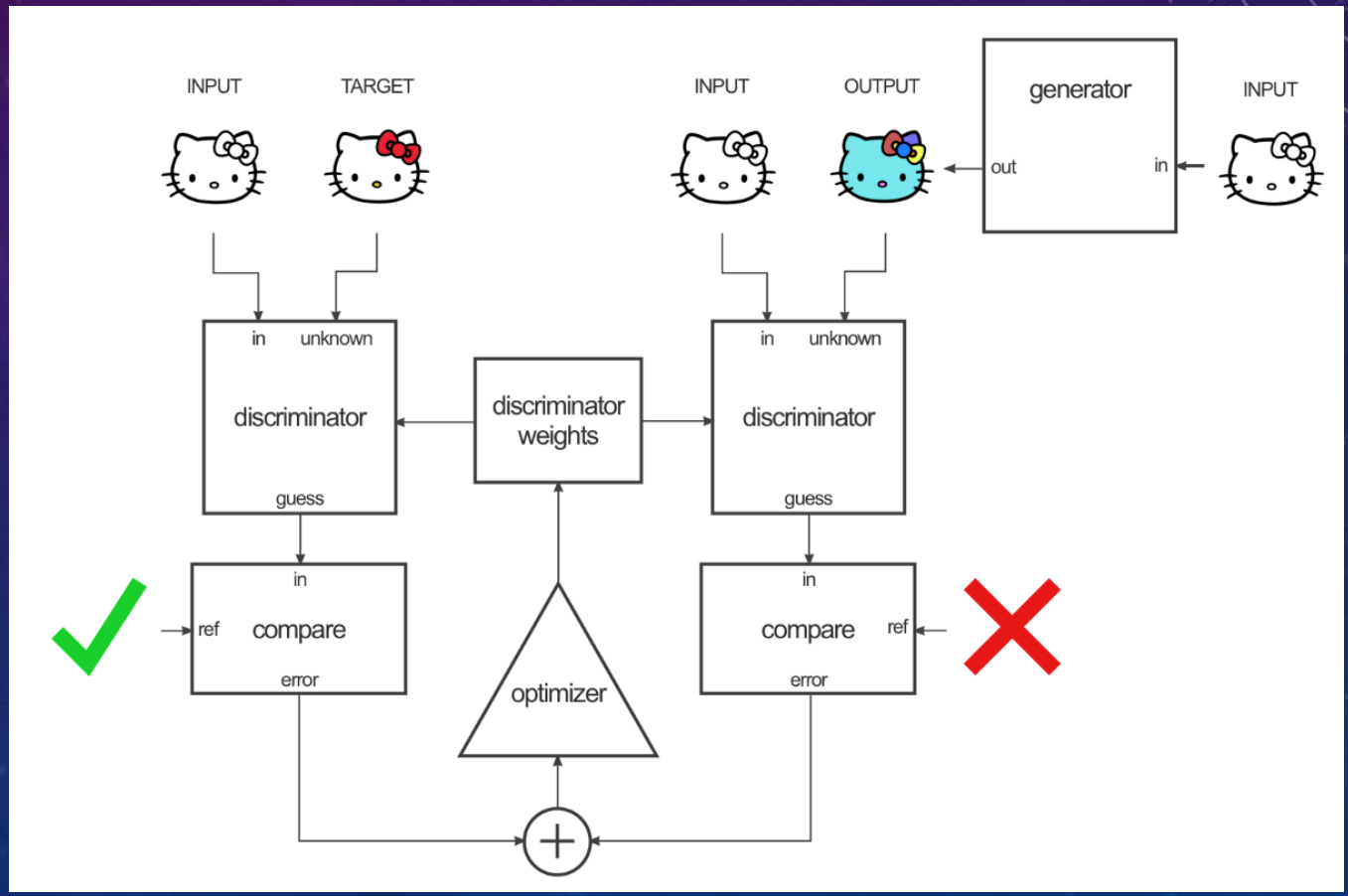
  - The final objective function for optimizing G can be written as follows:

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

Loss Functions of Different Combinations

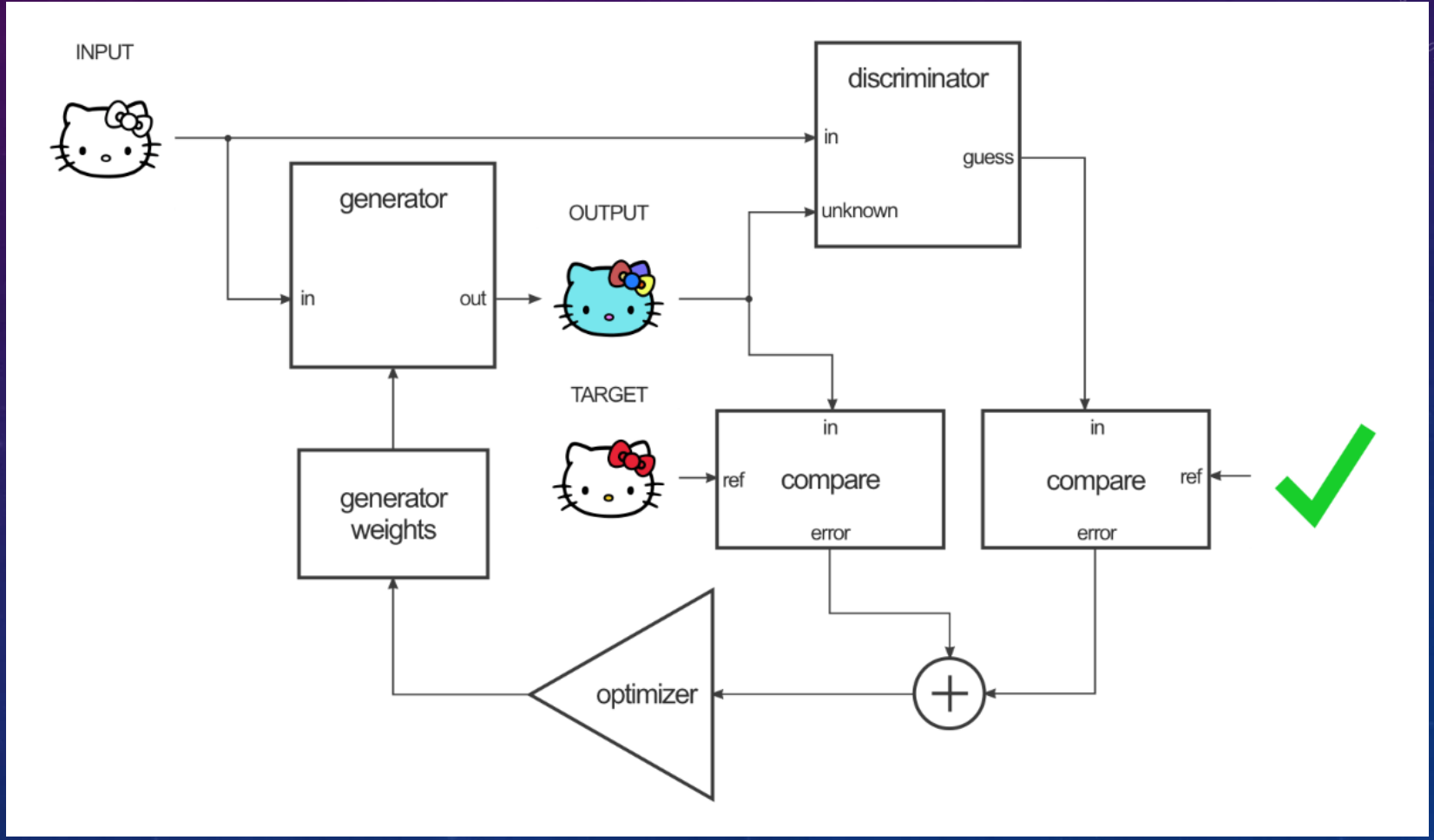
Pix2Pix – Model Overview

- The generator aims to generate the photo-realistic generated images.
- The discriminator aims to discriminate the generated images (INPUT) from the target image (TARGET).



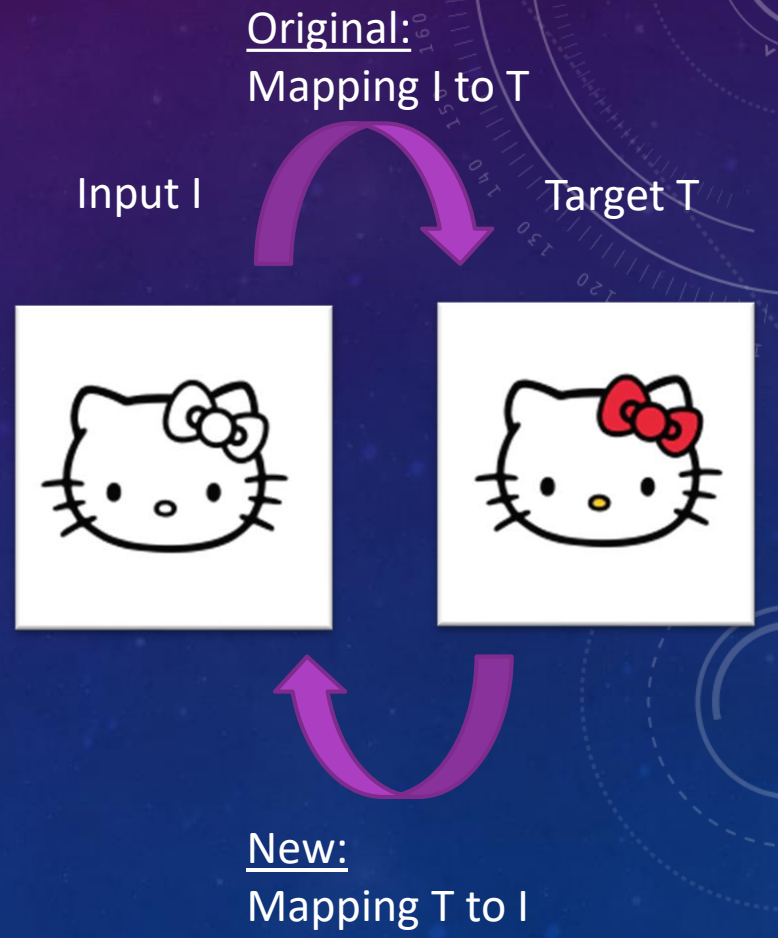
Pix2Pix – Model Overview

- The generator's weights are then adjusted based on the output of the discriminator as well as the difference between the output and target image.



Pix2Pix – Improvement

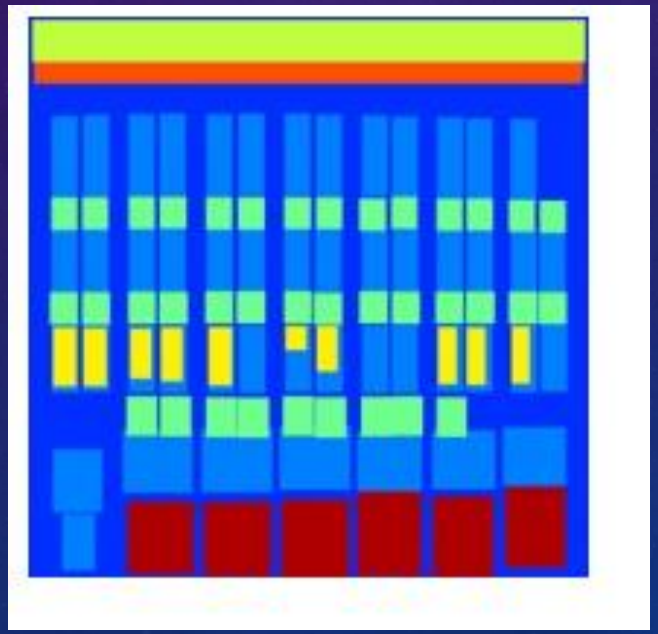
- Pix2Pix only goes from the Input *I* domain to the target *T* domain
- But if we can map from I to T, we should also be able to map from T to I.
- Therefore we can complete a whole cycle
- This is an improvement over the pix2pix method, which was the main idea of cycleGAN, which we will see next



Example 5.2 – Introduction Facades

We want to convert domain A with the color labels to Domain B, which represent real world facades. Therefore, we will train our Conditional GAN with the facades dataset to find a mapping from A to B.

Domain A



Domain B

Example 5.2

[1] For this exercise we are using the github from the authors which is available from the link https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

To clone it into your colab drive we use git clone followed by the github link

```
!git clone https://github.co
```

```
[2] import os
    os.chdir('pytorch-CycleGAN-and-pix2pix/')
```

```
[3] !pip install -r requirements.txt
```

[2] Os.chdir changes the cwd

[3] we install all packages necessary for the github which are saved in the requirements.txt file

Datasets

Download one of the official datasets with:

- `bash ./datasets/download_pix2pix_data`

Or use your own dataset by creating the appropriate folders and adding in the images. Follow the instructions here.

```
[4] !bash ./datasets/download_pix2pix_dataset.sh facades
```

Pretrained models

Download one of the official pretrained models with:

- `bash ./scripts/download_pix2pix_model.sh [edges2shoes, sat2map, map2sat, facades_label2photo, and day2night]`

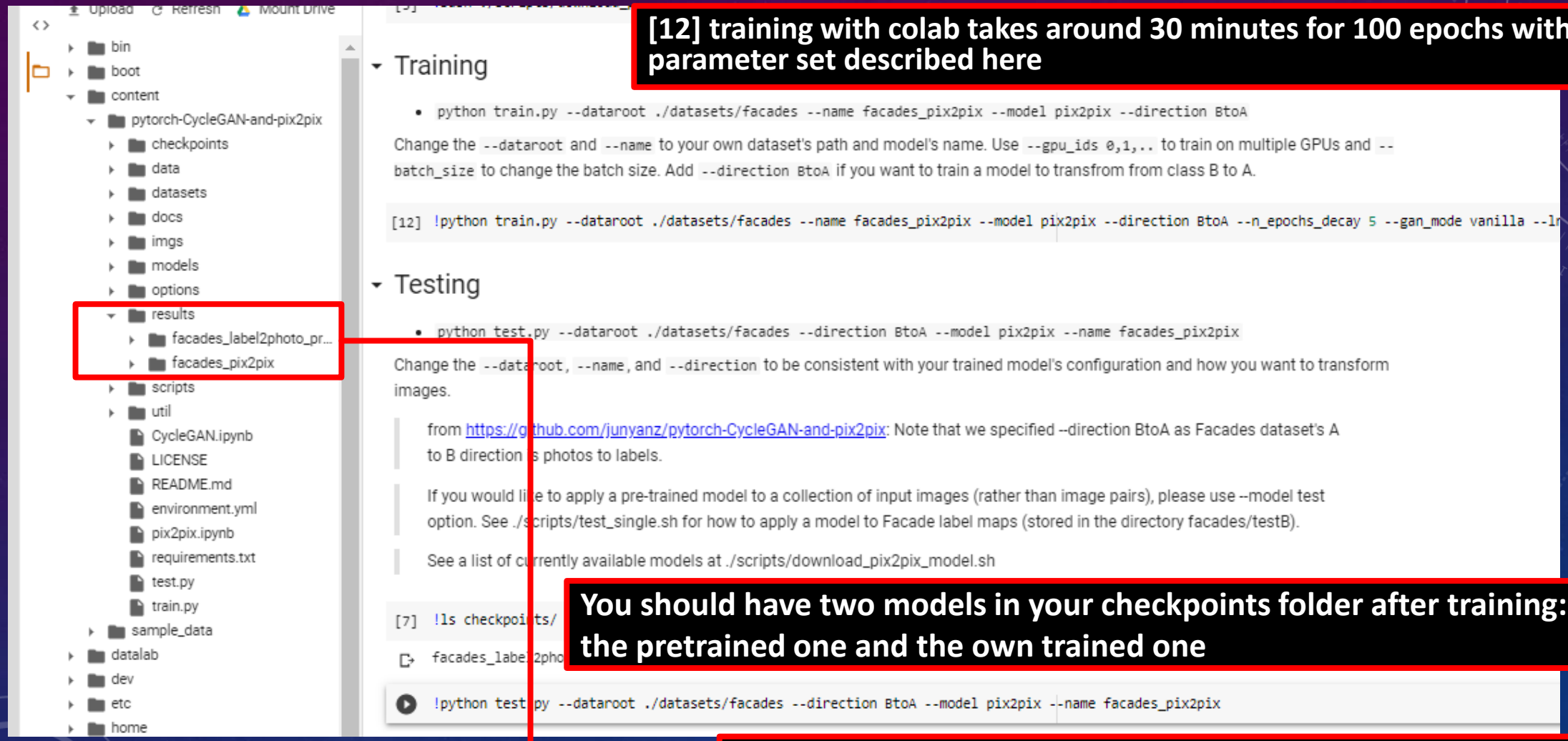
Or add your own pretrained model to `./checkpoints/{NAME}_pretrained/latest_net_G.pt`

```
!bash ./scripts/download_pix2pix_model.sh facades_label2photo
```

[4,5] Bash command executes the commands in .sh files

Here we download the dataset and the pretrained weights

Example 5.2



[12] training with colab takes around 30 minutes for 100 epochs with the parameter set described here

You should have two models in your checkpoints folder after training: the pretrained one and the own trained one

The test.py will output the translated images to the results folder. You can change the weights with the –name argument

Example 5.2 – Training overview

You can train the model by the following commands：

From domain B to domain A

```
!python train.py --dataroot ./datasets/facades --name facades_pix2pix --model pix2pix --direction BtoA --n_epochs_decay 5 --continue_train --gan_mode vanilla --lr_policy step
```

(train.py) You can change the loss function in the following command：

```
parser.add_argument('--gan_mode', type=str, default='lsgan', help='the type of GAN objective. [vanilla| lsgan | wgangp]
```

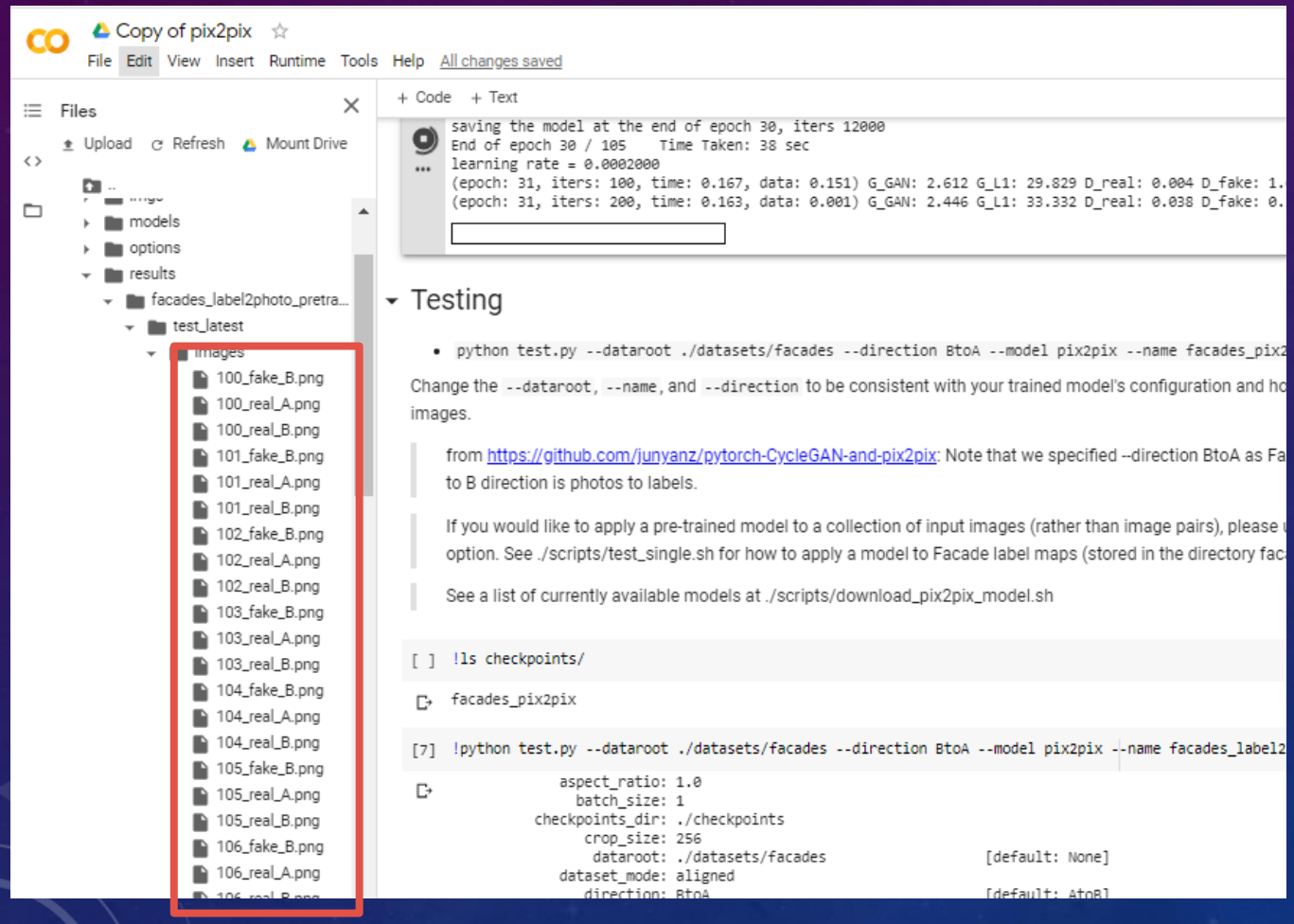
Overview of training

```
----------------- Options ---------------
               batch_size: 1
                    beta1: 0.5
          checkpoints_dir: ./checkpoints
           continue_train: True                          [default: False]
                crop_size: 256
                 dataroot: ./datasets/facades            [default: None]
             dataset_mode: aligned
                direction: BtoA                          [default: AtoB]
              display_env: main
             display_freq: 400
               display_id: 1
            display_ncols: 4
             display_port: 8097
           display_server: http://localhost
          display_winsize: 256
                    epoch: latest
              epoch_count: 1
                 gan_mode: vanilla
                  gpu_ids: 0
                init_gain: 0.02
                init_type: normal
                 input_nc: 3
                  isTrain: True                          [default: None]
              lambda_L1: 100.0
```

Observing the training

```
Setting up a new session...
create web directory ./checkpoints/facades_pix2pix/web...
(epoch: 1, iters: 100, time: 0.165, data: 0.293) G_GAN: 2.553 G_L1: 47.217 D_real: 0.009 D_fake: 0.177
(epoch: 1, iters: 200, time: 0.165, data: 0.001) G_GAN: 3.394 G_L1: 40.468 D_real: 0.030 D_fake: 0.135
(epoch: 1, iters: 300, time: 0.165, data: 0.002) G_GAN: 3.118 G_L1: 40.225 D_real: 0.003 D_fake: 1.685
(epoch: 1, iters: 400, time: 0.339, data: 0.001) G_GAN: 1.706 G_L1: 35.663 D_real: 0.118 D_fake: 0.926
End of epoch 1 / 105     Time Taken: 39 sec
learning rate = 0.0002000
(epoch: 2, iters: 100, time: 0.163, data: 0.108) G_GAN: 2.846 G_L1: 30.274 D_real: 0.021 D_fake: 0.664
(epoch: 2, iters: 200, time: 0.165, data: 0.002) G_GAN: 1.453 G_L1: 27.789 D_real: 1.481 D_fake: 0.160
(epoch: 2, iters: 300, time: 0.166, data: 0.001) G_GAN: 2.581 G_L1: 39.521 D_real: 0.017 D_fake: 0.320
(epoch: 2, iters: 400, time: 0.312, data: 0.001) G_GAN: 2.497 G_L1: 27.504 D_real: 0.568 D_fake: 0.039
End of epoch 2 / 105     Time Taken: 37 sec
learning rate = 0.0002000
```

Example 5.2 - Results



The red textbox shows the folder with all your results. You can later have a look by adjusting the names of the files in the next slide, downloading or just clicking them one by one.

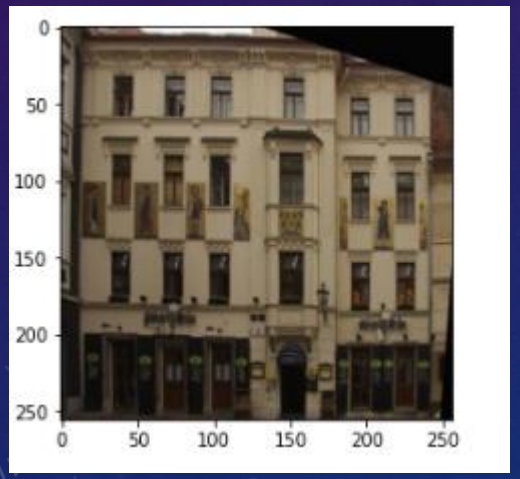
Example 5.2 - Results

The command below will show the images. Remember you can change the output of the imageplot by changing the file name. left is the Fake image on the left side on Domain A,

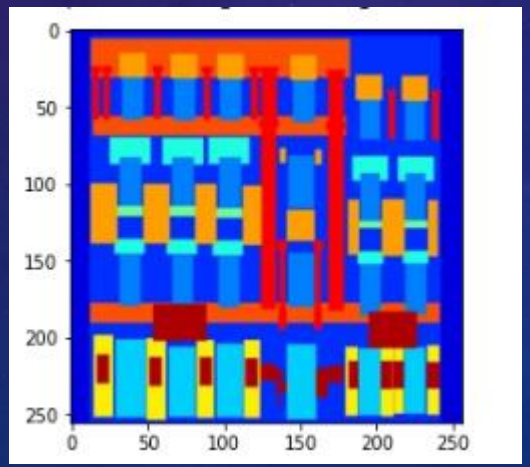
```
img = plt.imread('./results/facades_label2photo_pretrained/test_latest/images/100_fake_B.png')
plt.imshow(img)
```

```
img = plt.imread('./results/facades_label2photo_pretrained/test_latest/images/104_fake_B.png')
plt.imshow(img)
```

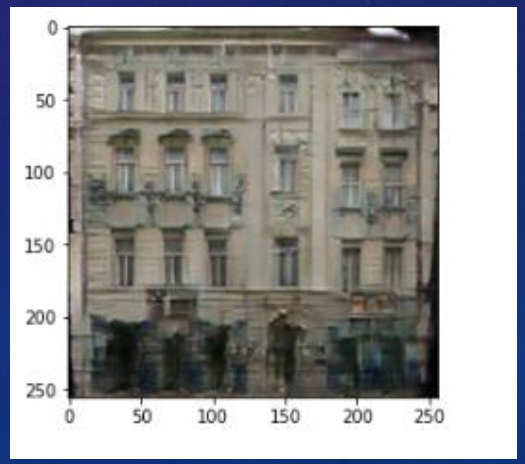
Real Image (Training Set)



Labels of real image (Domain B)



Fake Image from Labels of Domain B (Domain A)



Another fake image