Deep Learning

Deep Convolutional Generative Adversarial Networks (Lab)

U Kang
Seoul National University

In This Lecture

- Deep Convolutional Generative Adversarial Networks
  - Generator (Deconvolution layers)
  - Discriminator (Convolution layers)
  - Minimax problem

Outline

- **Deep Convolutional GAN**
- Data Preparation

Deep Convolutional GAN

- Deep Convolutional GAN (DCGAN)
  - Unsupervised machine learning
  - Generative model
  - An adversarial system of two convolutional neural networks

Generative Adversarial Networks

- Generative Adversarial Networks (GAN)
  - **Generator**: generate fake samples, tries to fool the Discriminator
  - **Discriminator**: tries to distinguish between real and fake samples
  - Train them against each other
  - Repeat this procedure and we get better Generator/Discriminator

Generative Adversarial Networks

- GAN's architecture

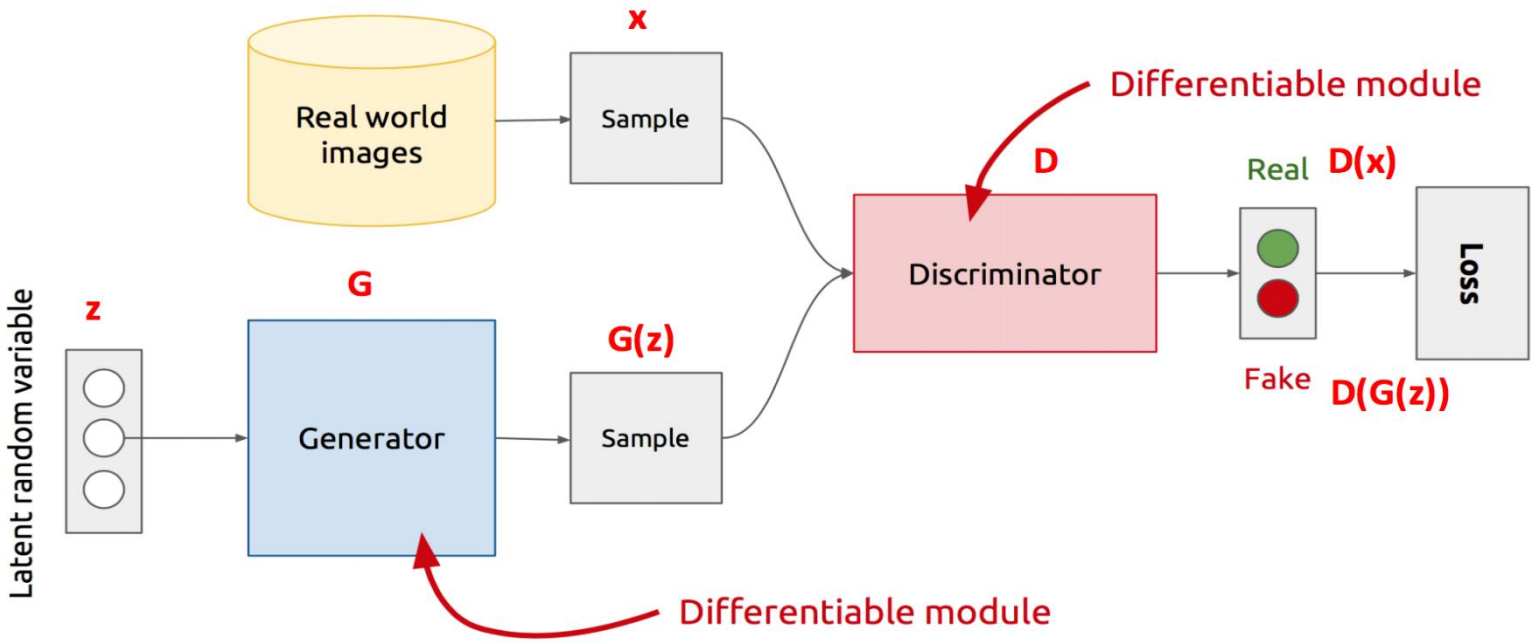
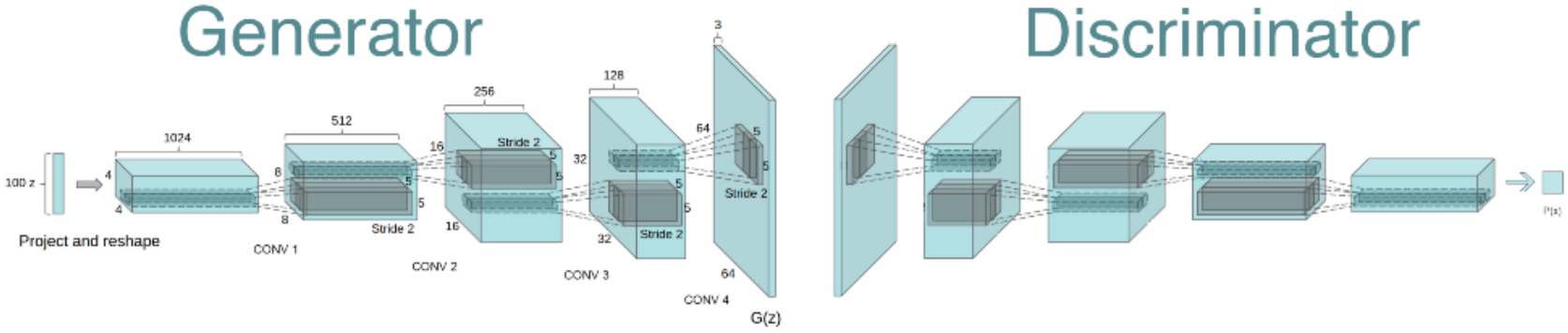


Figure from: https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

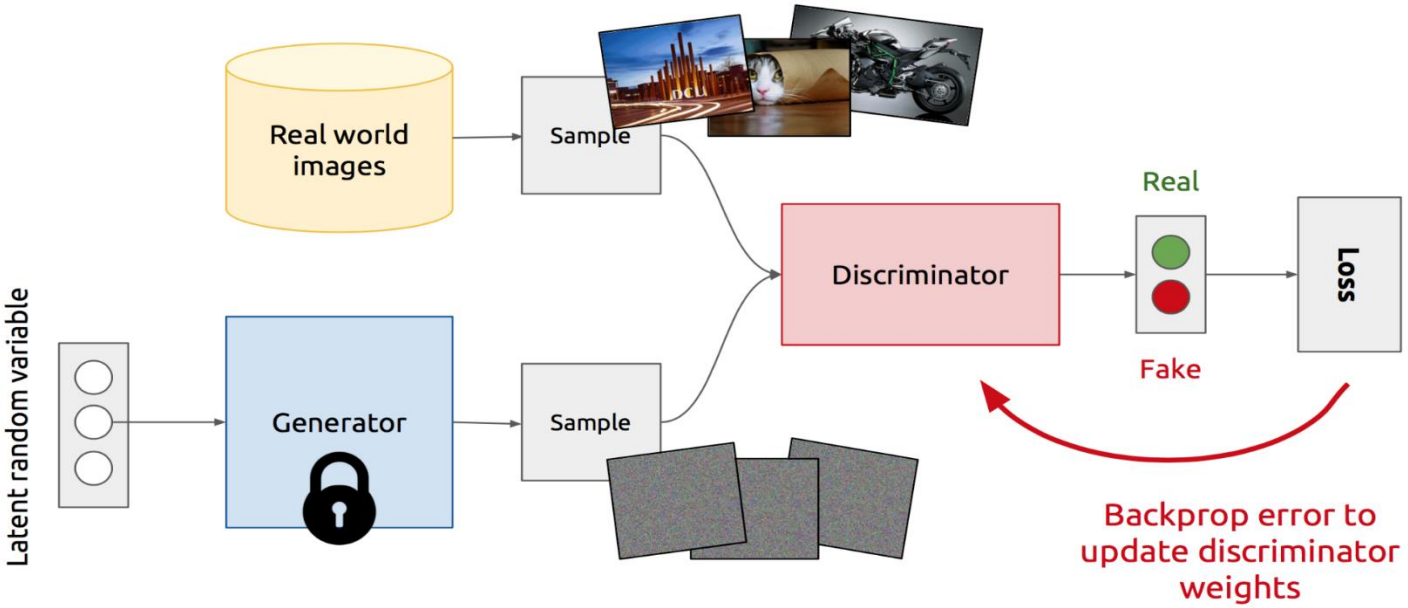
Deep Convolutional GAN

- DCGAN's architecture
  - Only differences from naïve GAN:
    - Generator is a deconvolutional neural network
    - Discriminator is a convolutional neural network



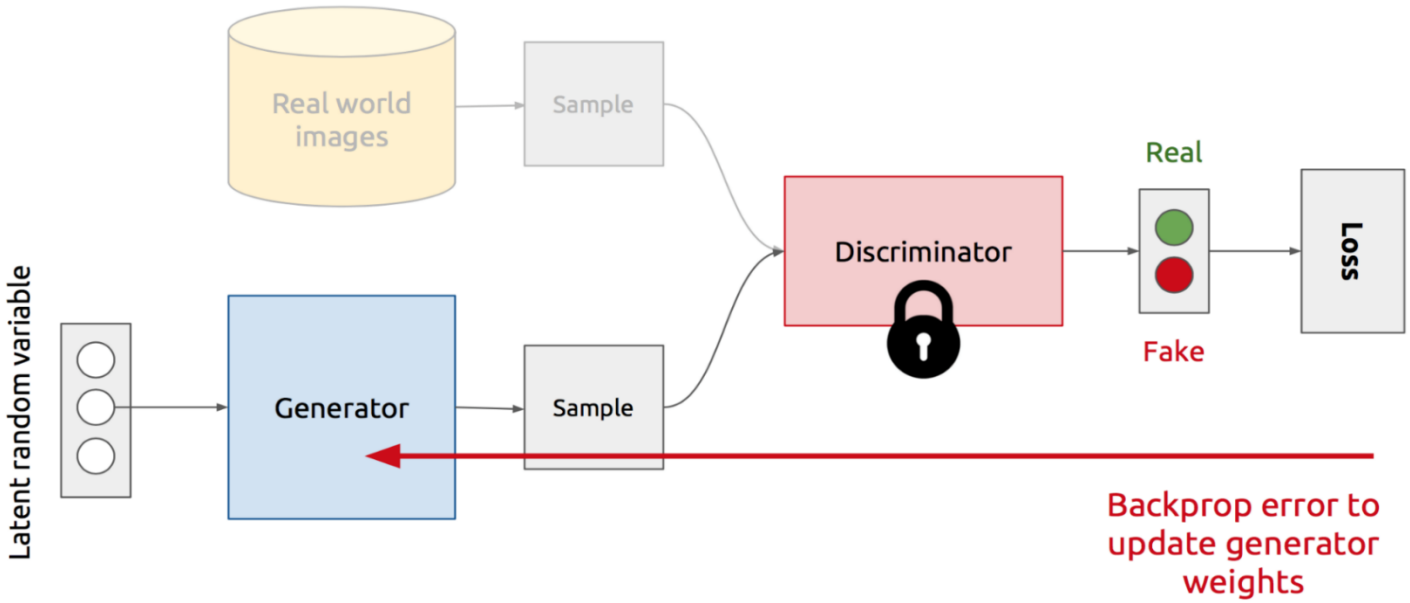
Deep Convolutional GAN

- Training Discriminator
  - The Discriminator determines if a sample is fake or real



Deep Convolutional GAN

- Training Generator
  - The Generator aims to fool the Discriminator



Deep Convolutional GAN

- DCGAN's loss function

$$\min_{G} \max_{D} V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[logD(x)] + \mathbb{E}_{z\sim q(z)}[\log(1 - D(G(z))]$$

- The Discriminator maximizes its reward $V(D,G)$
- The Generator minimizes Discriminator's reward
- The equilibrium
  - $\forall x, P_{data}(x) = P_{gen}(x)$
  - $\forall x, D(x) = \frac{1}{2}$

Outline

- ☑ Deep Convolutional GAN
- ☐ **Data Preparation**

Dataset (CelebA)

- CelebFaces Attributes Dataset (CelebA)
  - Number of images: 202,599
  - Size: (178 X 218) pixels
    - We will crop and resize the images for training
  - Sample images





Download the dataset

- If you already prepared the dataset, skip to preprocessing
  - It takes about an hour to download and extract the dataset
- Import:

```
import os
import requests
import tarfile
from tqdm import tqdm
import time
```

Download the dataset

- Download the tar file from Google Drive
- Then extract the tar file

```
def download_celeb_a(dirpath):
    dirpath = './data'
    data_dir = 'celebA'

    if os.path.exists(os.path.join(dirpath, data_dir)):
        print('Found Celeb-A - skip')
        return

    if not os.path.exists(dirpath):
        os.makedirs(dirpath)

    filename, drive_id  = "celebA.tgz", "1VipMDvxWXPO7gqZFR_ShA8seuVYM1rsJ"
    save_path = os.path.join(dirpath, filename)

    if os.path.exists(save_path):
        print('[*] {} already exists'.format(save_path))
    else:
        download_file_from_google_drive(drive_id, save_path)

    print("Start extracting tar file.")
    with tarfile.open(save_path) as open_tarfile:
        open_tarfile.extractall(dirpath)
    os.remove(save_path)
    print("Download CelebA End.")
```

Download the dataset

- Sub-functions for downloading the tar file

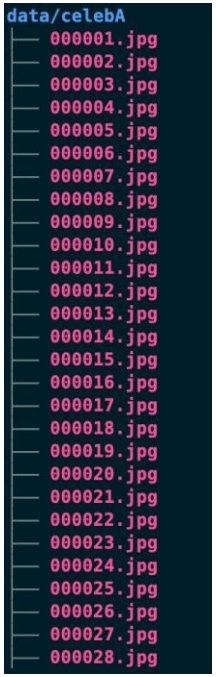
```
def get_confirm_token(response):
    for key, value in response.cookies.items():
        if key.startswith('download_warning'):
            return value
    return None

def save_response_content(response, destination, chunk_size=32*1024):
    total_size = int(response.headers.get('content-length', 0))
    with open(destination, "wb") as f:
        for chunk in tqdm(response.iter_content(chunk_size), total=total_size,
                          unit='B', unit_scale=True, desc=destination):
            if chunk: # filter out keep-alive new chunks
                f.write(chunk)

def download_file_from_google_drive(id, destination):
    print("Downloading into ./data/... Please wait.")
    URL = "https://docs.google.com/uc?export=download"
    session = requests.Session()

    response = session.get(URL, params={ 'id': id }, stream=True)
    token = get_confirm_token(response)

    if token:
        params = { 'id' : id, 'confirm' : token }
        response = session.get(URL, params=params, stream=True)

    save_response_content(response, destination)
    print("Download Done.")
```

Download the dataset

- Now, set the data path and prepare the dataset

```
dirpath = './data'
download_celeb_a(dirpath)
```

- Tree of the dataset:



Preprocess the dataset

- Import:

```
%matplotlib inline
import os
import math
import numpy as np

from glob import glob
from matplotlib import pyplot as plt
from PIL import Image
```

- Configuration:

```
# Image configuration
import random

IMAGE_HEIGHT = 64
IMAGE_WIDTH = 64
data_files = glob(os.path.join(dirpath, 'celebA/*.jpg')) # list of all '.jpg' files
data_shape = (len(data_files), IMAGE_WIDTH, IMAGE_HEIGHT, 3)
```

  - The images will be resized to (64 X 64) pixels

Preprocess the dataset

- Read, crop, and resize an image

```
def get_image(image_path, width, height, mode):
    """
    Read an image from image_path, and crop/resize the image
    """
    image = Image.open(image_path)

    if image.size != (width, height):
        # Remove pixels that aren't part of a face
        face_width = face_height = 108
        j = (image.size[0] - face_width) // 2
        i = (image.size[1] - face_height) // 2
        image = image.crop([j, i, j + face_width, i + face_height])
        image = image.resize([width, height], Image.BILINEAR)

    return np.array(image.convert(mode))
```

  - The resultant size of the image is (64 X 64) which will be given
  - “mode” will be given (e.g., ‘RGB’)

Data batch

- Get image files and preprocess them

```
def get_batch(image_files, width, height, mode='RGB'):
    """
    Get a single batch
    """
    data_batch = np.array(
        [get_image(sample_file, width, height, mode)
         for sample_file in image_files]).astype(np.float32)

    # Make sure the images are in 4 dimensions
    if len(data_batch.shape) < 4:
        data_batch = data_batch.reshape(data_batch.shape + (1,))

    return data_batch
```

  - The shape of each batch should be:
    - (num_batch, width, height, channel)

Data batch

- Split the entire dataset by batch size

```
def get_batches(batch_size, data_files, data_shape):
    """
    Generate batches
    """
    IMAGE_MAX_VALUE = 255

    cur_index = 0
    while cur_index + batch_size <= data_shape[0]:
        # print(f"This batch: [{cur_index}/{data_shape[0]}]")
        data_batch = get_batch(data_files[cur_index:cur_index + batch_size],
                               *data_shape[1:3])
        cur_index += batch_size
        yield data_batch / IMAGE_MAX_VALUE - 0.5
```

  - data_files: list of all data samples
  - Normalize the pixels of the images into [-0.5, 0.5]

Visualization

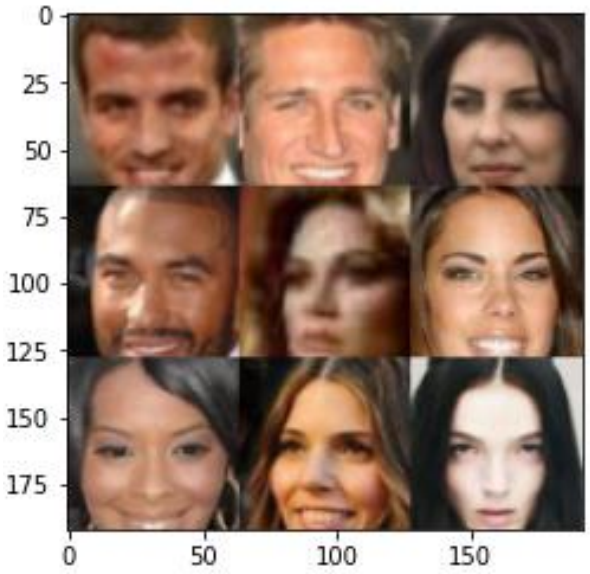
- Draw images in a grid plot

```
def images_square_grid(images, mode='RGB'):
    """
    Helper function to save images as a square grid (visualization)
    """
    # Get maximum size for square grid of images
    save_size = math.floor(np.sqrt(images.shape[0]))
    # Scale to 0-255
    images = (((images - images.min()) * 255) / (images.max() - images.min())).astype(np.uint8)
    # Put images in a square arrangement
    images_in_square = np.reshape(
            images[:save_size*save_size],
            (save_size, save_size, images.shape[1], images.shape[2], images.shape[3]))
    # Combine images to grid image
    new_im = Image.new(mode, (images.shape[1] * save_size, images.shape[2] * save_size))
    for col_i, col_images in enumerate(images_in_square):
        for image_i, image in enumerate(col_images):
            im = Image.fromarray(image, mode)
            new_im.paste(im, (col_i * images.shape[1], image_i * images.shape[2]))

    return new_im
```

Visualization

- Let's sample some images and draw them in the plot

```
test_images = get_batch(glob(os.path.join(dirpath, 'celebA/*.jpg'))[:10], data_shape[1], data_shape[2])
plt.imshow(images_square_grid(test_images))
```



- Images are cropped to focus more on the faces

Questions?

Reference

- Reference
  - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
    - https://arxiv.org/pdf/1511.06434.pdf