# Universidad del Valle de Guatemala
# Data Science
# Laboratorio6

## Integrantes:
## - Christopher García 20541
## - Andrea Lam 20102

In [ ]:
```
# Se importan librerías
import random
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.utils import save_image
import torchvision.datasets as dset
import torchvision.transforms as transforms
import numpy as np
```

## Se accede a Kaggle para obtener datos

In [ ]:
```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in c:\users\andre\onedrive\documentos\github\u
vg_ds_lab6\.venv\lib\site-packages (1.5.16)
Requirement already satisfied: six>=1.10 in c:\users\andre\onedrive\documentos\githu
b\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in c:\users\andre\onedrive\documentos\github
\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (2023.7.22)
Requirement already satisfied: python-dateutil in c:\users\andre\onedrive\documentos
\github\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in c:\users\andre\onedrive\documentos\github
\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in c:\users\andre\onedrive\documentos\github\uvg
_ds_lab6\.venv\lib\site-packages (from kaggle) (4.66.1)
Requirement already satisfied: python-slugify in c:\users\andre\onedrive\documentos
\github\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in c:\users\andre\onedrive\documentos\github
\uvg_ds_lab6\.venv\lib\site-packages (from kaggle) (2.0.6)
Requirement already satisfied: bleach in c:\users\andre\onedrive\documentos\github\u
vg_ds_lab6\.venv\lib\site-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in c:\users\andre\onedrive\documentos\gi
thub\uvg_ds_lab6\.venv\lib\site-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in c:\users\andre\onedrive\docume
ntos\github\uvg_ds_lab6\.venv\lib\site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\andre\onedrive\d
ocumentos\github\uvg_ds_lab6\.venv\lib\site-packages (from requests->kaggle) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\andre\onedrive\documentos\gi
thub\uvg_ds_lab6\.venv\lib\site-packages (from requests->kaggle) (3.4)
Requirement already satisfied: colorama in c:\users\andre\onedrive\documentos\github
\uvg_ds_lab6\.venv\lib\site-packages (from tqdm->kaggle) (0.4.6)
```

In [ ]:
```
import os

# Obtener el directorio actual donde se encuentra el archivo de Jupyter Notebook
current_directory = os.path.dirname(os.path.abspath('__file__'))

# Ruta completa al directorio .kaggle en el directorio actual
kaggle_directory = os.path.join(current_directory, ".kaggle")

# Crear el directorio .kaggle si no existe
if not os.path.exists(kaggle_directory):
    os.mkdir(kaggle_directory)
```

In [ ]:
```
!copy .kaggle\kaggle.json %userprofile%\.kaggle\
```

```
        1 file(s) copied.
```

In [ ]:
```
! chmod 600 ~/.kaggle/kaggle.json
```

```
'chmod' is not recognized as an internal or external command,
operable program or batch file.
```

In [ ]:
```
!kaggle datasets download -d jessicali9530/celeba-dataset
```

```
celeba-dataset.zip: Skipping, found more recently modified local copy (use --force t
o force download)
```

In [ ]:
```
import os
import zipfile


# Obtener la ubicación actual de trabajo
current_directory = os.getcwd()

# Definir una ubicación para la extracción
extract_path = os.path.join(current_directory, "celeba-dataset")

# Crear la carpeta de extracción si no existe
if not os.path.exists(extract_path):
    os.makedirs(extract_path)

# Extraer el archivo ZIP
zip_ref = zipfile.ZipFile('celeba-dataset.zip', 'r')
zip_ref.extractall(extract_path)
zip_ref.close()
```

In [ ]:
```
transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Redimensiona las imágenes a 128x128 píxeles
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

In [ ]:
```
dataset = dset.ImageFolder(
    root="./celeba-dataset/img_align_celeba",
    transform=transform  # Utiliza la transformación modificada
)
```

# Preparacion de datos

In [ ]:
```
# Define las variables
ngpu = 1
ngf = 64
nc = 3
nz = 100
lr = 0.0002
beta1 = 0.5
batch_size = 128
num_epochs = 5
workers = 2
```

In [ ]:
```
# Especifica la ruta al directorio de datos
data_dir = 'celeba-dataset/img_align_celeba/'

# Define la transformación para preprocesar las imágenes
transform = transforms.Compose([
    transforms.Resize(128),            # Redimensiona las imágenes a 128x128 píxele
    transforms.CenterCrop(128),        # Recorta las imágenes al centro
    transforms.ToTensor(),             # Convierte las imágenes en tensores
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normaliza los valores
])

# Carga el conjunto de datos
dataset = dset.ImageFolder(root="celeba-dataset/img_align_celeba", transform=transf

# Crea un DataLoader para facilitar el acceso a los datos en lotes
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=Tr

# Define el dispositivo (CPU o GPU)
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cp
```

# Implementacion de la GAN

In [ ]:
```
# Define el generador y el discriminador
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            # Capa de entrada: Z dimension -> ngf*8 dimension
            nn.ConvTranspose2d(nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),

            # Capa 2: ngf*8 dimension -> ngf*4 dimension
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),

            # Capa 3: ngf*4 dimension -> ngf*2 dimension
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
```

```
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),

            # Capa 4: ngf*2 dimension -> ngf dimension
            nn.ConvTranspose2d(ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),

            # Capa de salida: ngf dimension -> nc canales (3 canales en RGB)
            nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)
```

In [ ]:
```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            # Capa de entrada: 3 canales de color
            nn.Conv2d(nc, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            # Capa de salida: un solo canal
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

**# Definicion perdida y optimizadores**

In [ ]:
```
# Define el generador y el discriminador
generator = Generator().to(device)
discriminator = Discriminator().to(device)


# Define la función de pérdida para la GAN
criterion = nn.BCELoss()

# Define los optimizadores para el generador y el discriminador
```

```
optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(beta1, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(beta1, 0.999))
```

# Entrenamiento de la GAN

In [ ]:
```
real_labels = torch.ones((batch_size, 1, 1, 1), device=device)
fake_labels = torch.zeros((batch_size, 1, 1, 1), device=device)
```

In [ ]:
```
# Entrenamiento de la GAN
for epoch in range(num_epochs):
    for i, data in enumerate(dataloader, 0):
        real_images, _ = data  # Obtén un lote de imágenes reales y sus etiquetas (

        # Inicializa los gradientes del discriminador y el generador
        optimizer_D.zero_grad()

        # Entrena al discriminador con imágenes reales
        output = discriminator(real_images.to(device))
        loss_D_real = criterion(output, real_labels)
        loss_D_real.backward()

        # Genera un lote de imágenes falsas
        fake_images = generator(torch.randn(batch_size, nz, 1, 1, device=device))

        # Entrena al discriminador con imágenes falsas
        output = discriminator(fake_images.detach())
        loss_D_fake = criterion(output, fake_labels)
        loss_D_fake.backward()

        # Pérdida total del discriminador
        loss_D = loss_D_real + loss_D_fake

        # Actualiza los pesos del discriminador
        optimizer_D.step()

        # Inicializa los gradientes del generador
        optimizer_G.zero_grad()

        # Entrena al generador para engañar al discriminador
        output = discriminator(fake_images)
        loss_G = criterion(output, real_labels)
        loss_G.backward()

        # Actualiza los pesos del generador
        optimizer_G.step()

        # Imprime estadísticas durante el entrenamiento (puedes personalizar esto)
        if i % 100 == 0:
            print(f'Epoch [{epoch}/{num_epochs}] Batch [{i}/{len(dataloader)}] Loss

        # Guarda imágenes generadas por el generador al final de cada época (opcion
        if (i + 1) == len(dataloader):
            with torch.no_grad():
```

```
                fake = generator(torch.randn(64, nz, 1, 1, device=device)).detach()
            save_image(fake, f"images/epoch_{epoch}.png", normalize=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\andre\OneDrive\Documentos\GitHub\UVG_DS_Lab6\Laboratorio6.ipynb Cell 22 line 1
      <a href='vscode-notebook-cell:/c%3A/Users/andre/OneDrive/Documentos/GitHub/UVG_DS_Lab6/Laboratorio6.ipynb#X62sZmlsZQ%3D%3D?line=8'>9</a> # Entrena al discriminador con imágenes reales
     <a href='vscode-notebook-cell:/c%3A/Users/andre/OneDrive/Documentos/GitHub/UVG_DS_Lab6/Laboratorio6.ipynb#X62sZmlsZQ%3D%3D?line=9'>10</a> output = discriminator(real_images.to(device))
---> <a href='vscode-notebook-cell:/c%3A/Users/andre/OneDrive/Documentos/GitHub/UVG_DS_Lab6/Laboratorio6.ipynb#X62sZmlsZQ%3D%3D?line=10'>11</a> loss_D_real = criterion(output, real_labels)
     <a href='vscode-notebook-cell:/c%3A/Users/andre/OneDrive/Documentos/GitHub/UVG_DS_Lab6/Laboratorio6.ipynb#X62sZmlsZQ%3D%3D?line=11'>12</a> loss_D_real.backward()
     <a href='vscode-notebook-cell:/c%3A/Users/andre/OneDrive/Documentos/GitHub/UVG_DS_Lab6/Laboratorio6.ipynb#X62sZmlsZQ%3D%3D?line=13'>14</a> # Genera un lote de imágenes falsas

File c:\Users\andre\OneDrive\Documentos\GitHub\UVG_DS_Lab6\.venv\lib\site-packages\torch\nn\modules\module.py:1518, in Module._wrapped_call_impl(self, *args, **kwargs)
   1516     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1517 else:
-> 1518     return self._call_impl(*args, **kwargs)

File c:\Users\andre\OneDrive\Documentos\GitHub\UVG_DS_Lab6\.venv\lib\site-packages\torch\nn\modules\module.py:1527, in Module._call_impl(self, *args, **kwargs)
   1522 # If we don't have any hooks, we want to skip the rest of the logic in
   1523 # this function, and just call forward.
   1524 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1525         or _global_backward_pre_hooks or _global_backward_hooks
   1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
   1529 try:
   1530     result = None

File c:\Users\andre\OneDrive\Documentos\GitHub\UVG_DS_Lab6\.venv\lib\site-packages\torch\nn\modules\loss.py:618, in BCELoss.forward(self, input, target)
    617 def forward(self, input: Tensor, target: Tensor) -> Tensor:
--> 618     return F.binary_cross_entropy(input, target, weight=self.weight, reduction=self.reduction)

File c:\Users\andre\OneDrive\Documentos\GitHub\UVG_DS_Lab6\.venv\lib\site-packages\torch\nn\functional.py:3113, in binary_cross_entropy(input, target, weight, size_average, reduce, reduction)
   3111     reduction_enum = _Reduction.get_enum(reduction)
   3112 if target.size() != input.size():
-> 3113     raise ValueError(
   3114         "Using a target size ({}) that is different to the input size ({}) is deprecated. "
   3115         "Please ensure they have the same size.".format(target.size(), input.size())
   3116     )
   3118 if weight is not None:
   3119     new_size = _infer_size(target.size(), weight.size())
```

```
ValueError: Using a target size (torch.Size([128, 1, 1, 1])) that is different to th
e input size (torch.Size([128, 1, 5, 5])) is deprecated. Please ensure they have the
same size.
```