Laboratorio

Ejercicios propuestpos

- Generación de letras (cGAN) - MLP
- Generación de imagen médica (cGAN) – MLP
- Generación de imagen médica (cGAN) – DCGAN (Grises)
- Generación de imagen médica (cGAN) – DCGAN (Color)
- Generación de personajes de anime (GAN) - DCGAN

Generación de letras

- Entrenamiento de una cGAN basada en MLP con un conjunto de datos ofrecido por pytorch --> EMNIST

Generación de letras

- Obtengan el conjunto de datos usando pytorch
- Utilizen el siguiente código para mostrar los datos

```
def show_samples(data_loader):
    images, labels = next(iter(data_loader))
    plt.figure(figsize=(8,8))
    for i in range(64):
    ax = plt.subplot(8, 8, i + 1)
    image = torch.squeeze(images[i])
    class_label = int(labels[i])
    plt.imshow(image, cmap='gray')
    plt.axis('off')

num_classes = len(train_dataset.classes)
print(f"Number of classes in EMNIST: {num_classes}")

show_samples(train_dataloader)
```

Generación de letras (cGAN)

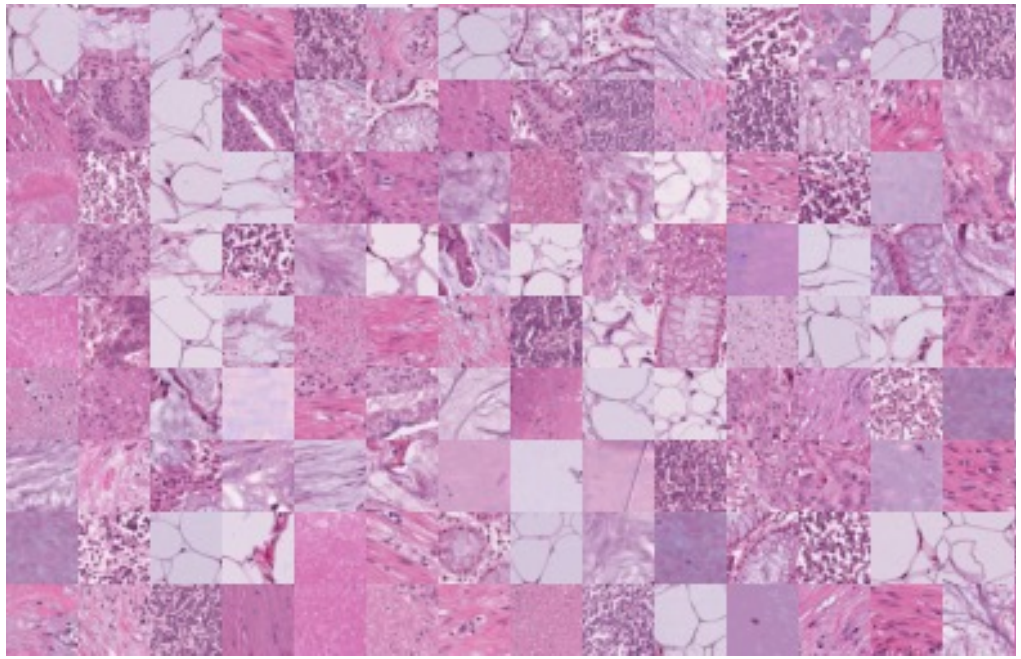
- Creen una arquitectura que entrene una cGAN
  - Implemente los modelos que sigan las siguientes instrucciones.
    - The **discriminator** should bea able to discriminate among 28x28 grayscale images, between four and five layers, using LeakyReLU as activation function, and Dropout.
      - The class has only one input: number of classes
    - The **generator** should be able to create 28x28 grayscale images, between four and five layers and using LeakyReLU as activation function.
      - The class has two inputs: size of z and number of classes

Generación de letras (cGAN)

- Configuren e implemente el entrenamiento con las siguientes características comunes
  - Size of z is 100
  - Binary Cross Entropy as loss funciton
  - Learning rate of 0.0001

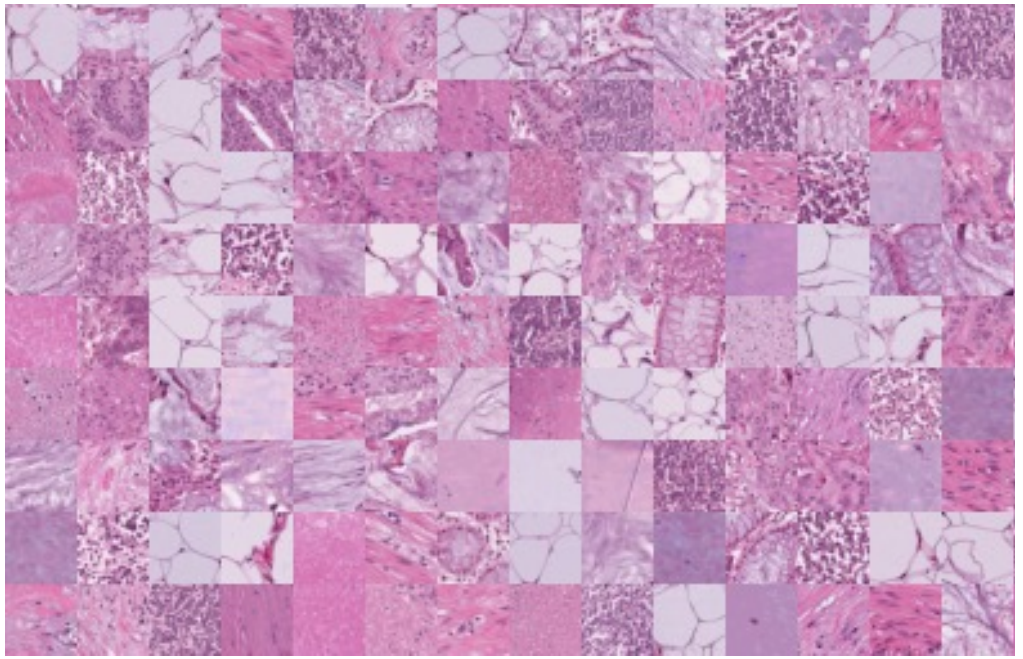
cGAN MedMnist. Patología de Colon - MLP

- Entrenamiento de una cGAN basada en MLP con un conjunto de datos ofrecido no ofrecido por pytorch --> MedMnist
- La cGAN será la definida en el ejercicio anterior.



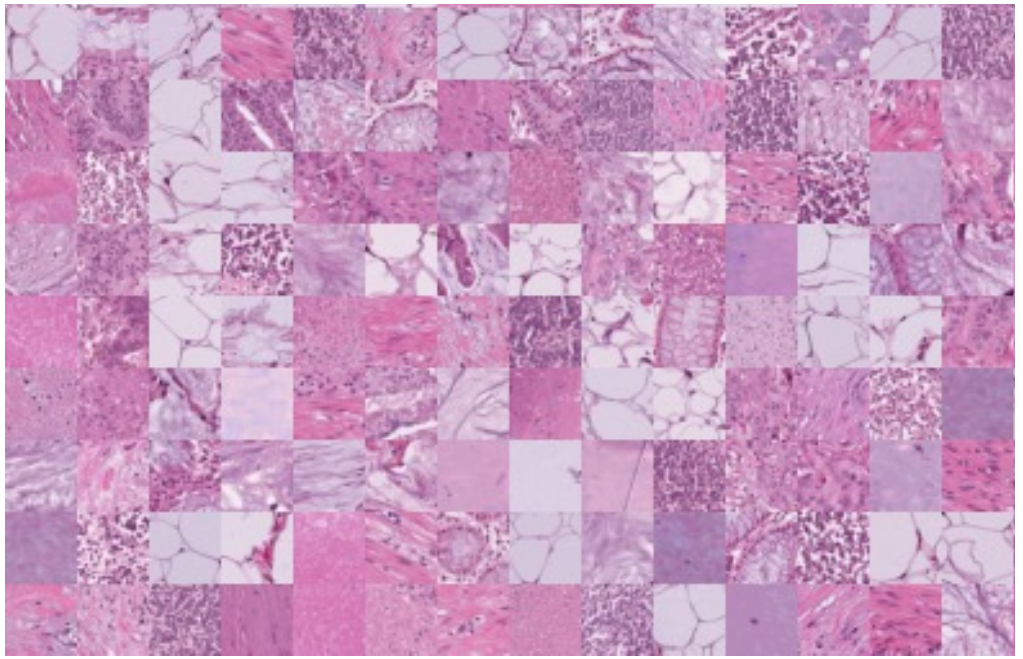
cGAN MedMnist. Patología de Colon (Grises) - DCGAN

- Entrenamiento de una cGAN basada en DCGAN con un conjunto de datos ofrecido no ofrecido por pytorch --> MedMnist
- La cGAN estará definida por una DCGAN.



cGAN MedMnist. Patología de Colon (Color) - DCGAN

- Entrenamiento de una cGAN basada en DCGAN con un conjunto de datos ofrecido no ofrecido por pytorch --> MedMnist
- La cGAN estará definida por una DCGAN.



Vanilla GAN Personajes de anime - DCGAN

- Entrenamiento de una GAN basada en DCGAN con un conjunto de datos definido por un conjunto de imágenes descargados.
- La GAN estará definida por una DCGAN.

Vanilla GAN Personajes de anime - DCGAN

- Para obtener los datos descárguense los mismos utilizando wget (copien el siguiente comando):

```
!wget --load-cookies /tmp/cookies.txt
"https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-
cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate
'https://docs.google.com/uc?export=download&id=1KIgyF4vpFDea-
Itq6Nz22vrWyouLCz87' -O- | sed -rn 's/.*confirm=([0-9A-Za-
z_]+).*/\1\n/p')&id=1KIgyF4vpFDea-Itq6Nz22vrWyouLCz87" -O anime-faces-
dataset.zip && rm -rf /tmp/cookies.txt
```

- Descompriman el archivo

```
!unzip anime-faces-dataset.zip -d dataset
```

Vanilla GAN Personajes de anime - DCGAN

- Para obtener los datos descárguense los mismos utilizando wget (copien el siguiente comando):

```
!wget --load-cookies /tmp/cookies.txt
"https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-
cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate
'https://docs.google.com/uc?export=download&id=1KIgyF4vpFDea-
Itq6Nz22vrWyouLCz87' -O- | sed -rn 's/.*confirm=([0-9A-Za-
z_]+).*/\1\n/p')&id=1KIgyF4vpFDea-Itq6Nz22vrWyouLCz87" -O anime-faces-
dataset.zip && rm -rf /tmp/cookies.txt
```

- Descompriman el archivo

```
!unzip anime-faces-dataset.zip -d dataset
```

Vanilla GAN Personajes de anime - DCGAN

- Definan las transformaciones y luego usen el siguiente código para crear la clase Dataset. El Dataloader se crea igual que siempre.

```
dataroot = "./dataset/" # Root directory for dataset

dataset = ImageFolder(root=dataroot,
transform=data_transforms)
```

Vanilla GAN Personajes de anime - DCGAN

- Muestre algunas imágenes usando el siguiente código.

```
def show_images(images):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(images.detach(), nrow=8).permute(1, 2, 0))

def show_batch(dl):
    for images, _ in dl:
        show_images(images)
    break

show_batch(dataloader)
```

Vanilla GAN Personajes de anime - DCGAN

- Dado el siguiente código para crear el generador y el discriminador agregue el código que haga falta para completar las dos clases.
- Defina el entrenamiento para esta GAN no supervisada.

Vanilla GAN Personajes de anime - DCGAN

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d(100, 64 * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(64 * 8),
            nn.ReLU(True),
            # state size. ``(64*8) x 4 x 4``
            nn.ConvTranspose2d(64 * 8, 64 * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64 * 4),
            nn.ReLU(True),
            # state size. ``(64*4) x 8 x 8``
            nn.ConvTranspose2d( 64 * 4, 64 * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64 * 2),
            nn.ReLU(True),
            # state size. ``(64*2) x 16 x 16``
            nn.ConvTranspose2d( 64 * 2, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            # state size. ``(64) x 32 x 32``
            nn.ConvTranspose2d( 64, 3, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. ``(nc) x 64 x 64``
            )
```

Vanilla GAN Personajes de anime - DCGAN

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
    super(Discriminator, self).__init__()
    self.main = nn.Sequential(
        # input is ``(3) x 64 x 64``
        nn.Conv2d(3, 64, 4, 2, 1, bias=False),
        nn.LeakyReLU(0.2, inplace=True),
        # state size. ``(ndf) x 32 x 32``
        nn.Conv2d(64, 64 * 2, 4, 2, 1, bias=False),
        nn.BatchNorm2d(64 * 2),
        nn.LeakyReLU(0.2, inplace=True),
        # state size. ``(ndf*2) x 16 x 16``
        nn.Conv2d(64 * 2, 64 * 4, 4, 2, 1, bias=False),
        nn.BatchNorm2d(64 * 4),
        nn.LeakyReLU(0.2, inplace=True),
        # state size. ``(ndf*4) x 8 x 8``
        nn.Conv2d(64 * 4, 64 * 8, 4, 2, 1, bias=False),
        nn.BatchNorm2d(64 * 8),
        nn.LeakyReLU(0.2, inplace=True),
        # state size. ``(ndf*8) x 4 x 4``
        nn.Conv2d(64 * 8, 1, 4, 1, 0, bias=False),
        nn.Sigmoid()
        )
```