GANs

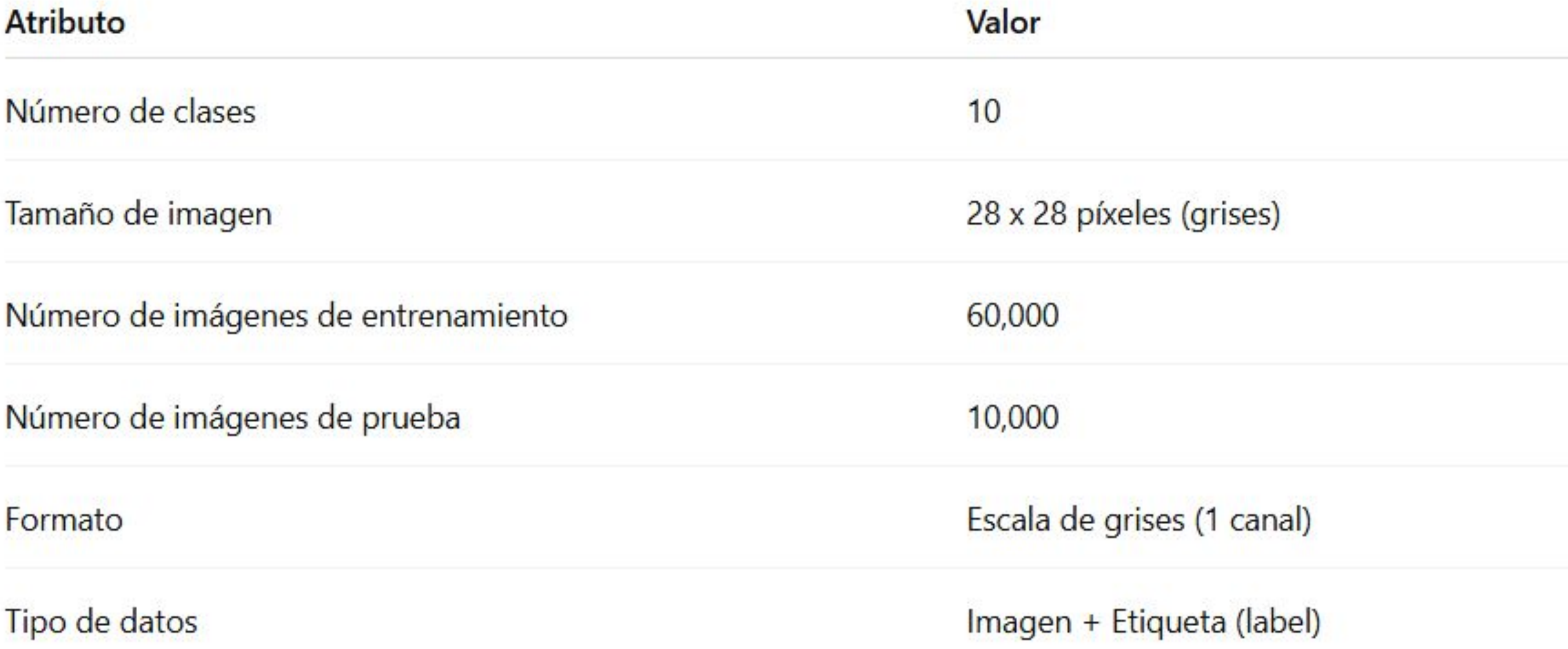
Adán González Rodríguez
Sara Porto Álvarez

Código:

Link al código

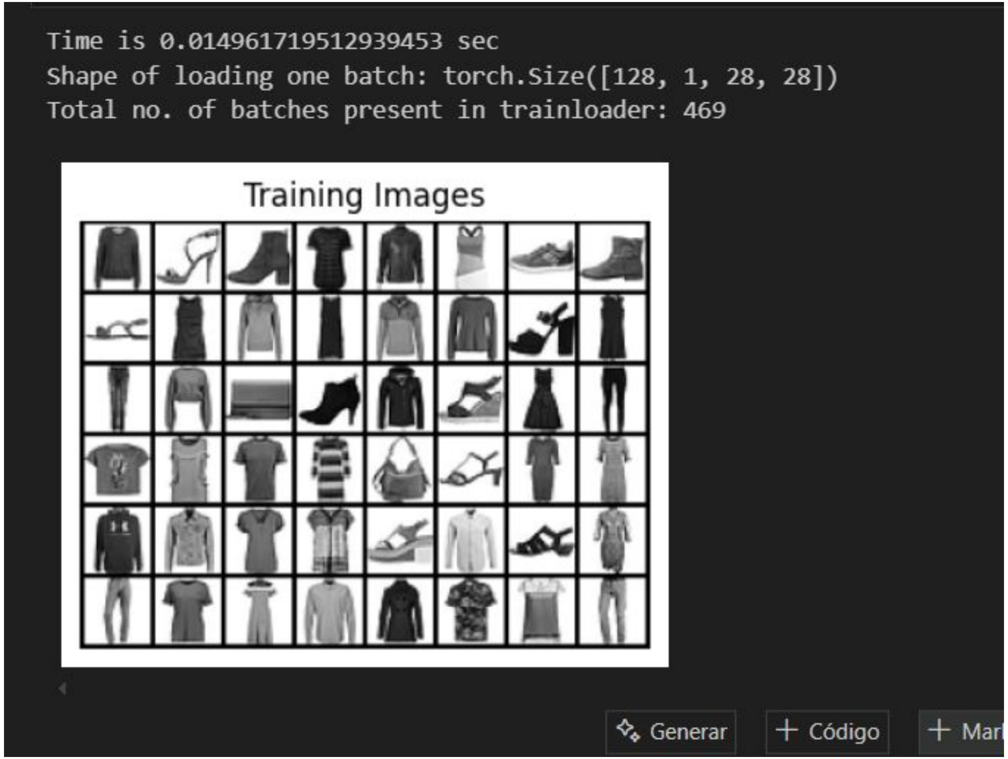
Dataset

Explicación del Dataset:



Atributo	Valor
Número de clases	10
Tamaño de imagen	28 x 28 píxeles (grises)
Número de imágenes de entrenamiento	60,000
Número de imágenes de prueba	10,000
Formato	Escala de grises (1 canal)
Tipo de datos	Imagen + Etiqueta (label)

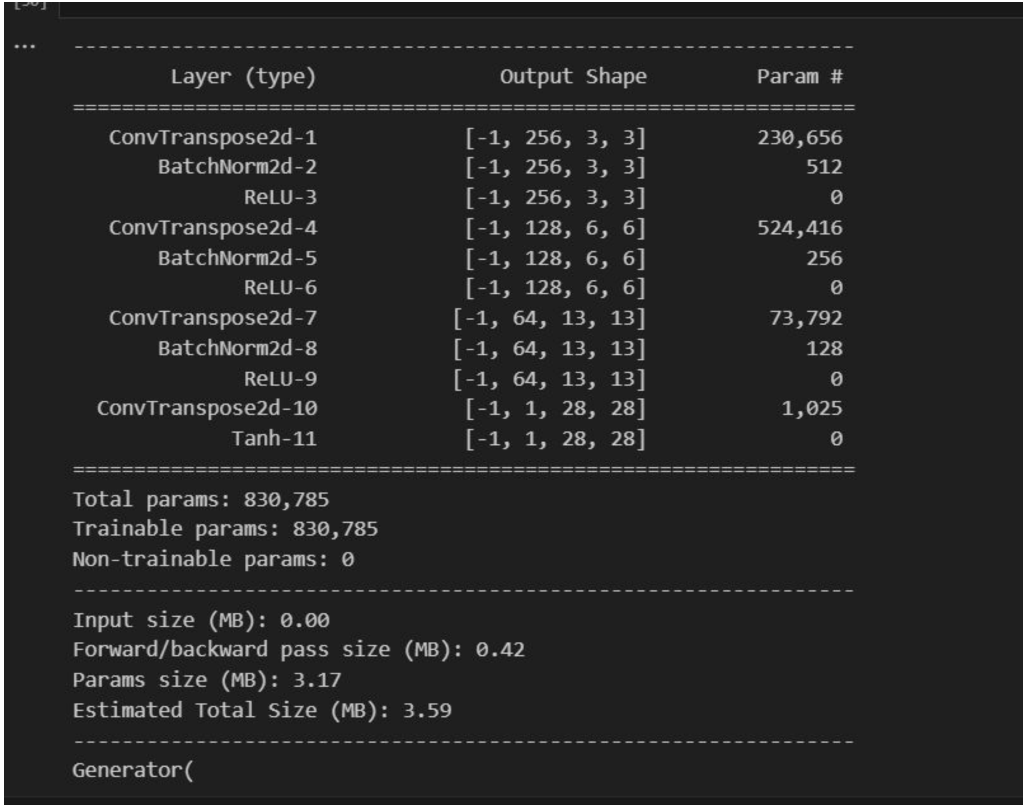
Dataset:



```
Time is 0.014961719512939453 sec
Shape of loading one batch: torch.Size([128, 1, 28, 28])
Total no. of batches present in trainloader: 469
```

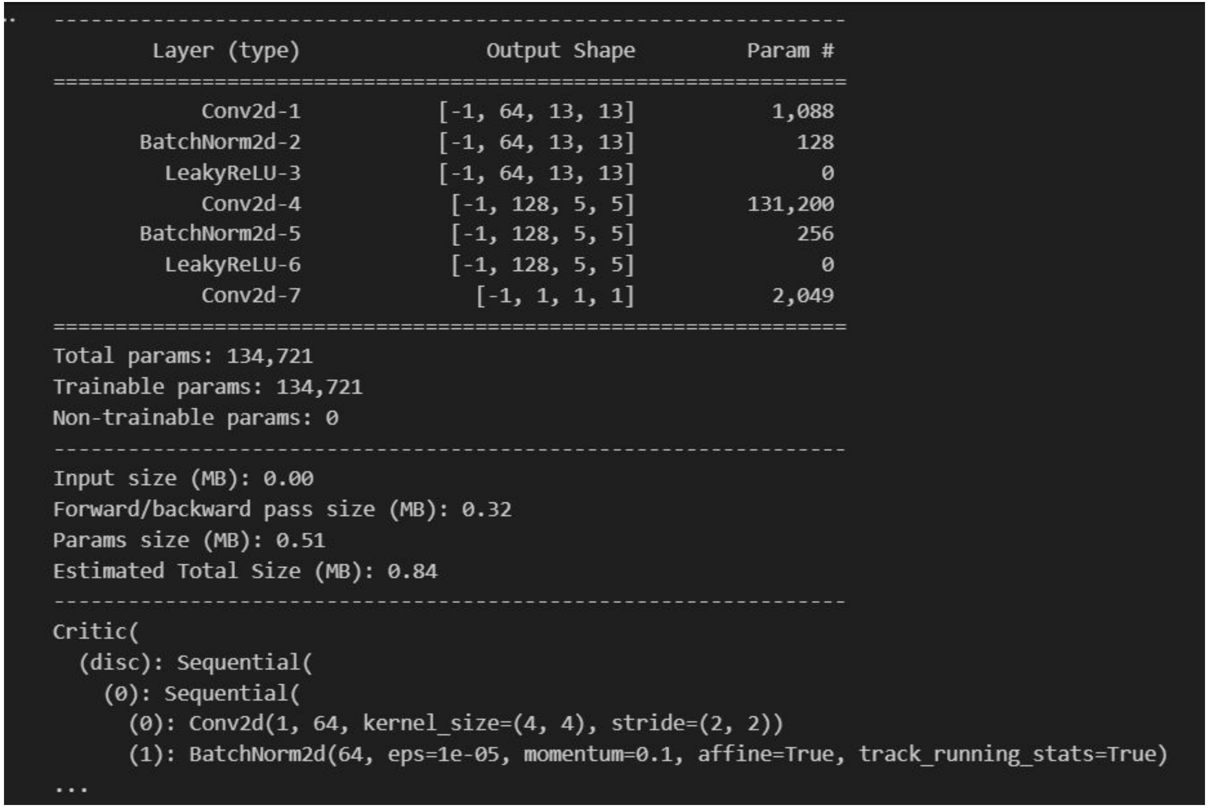
GAN

Red neuronal del Generator:



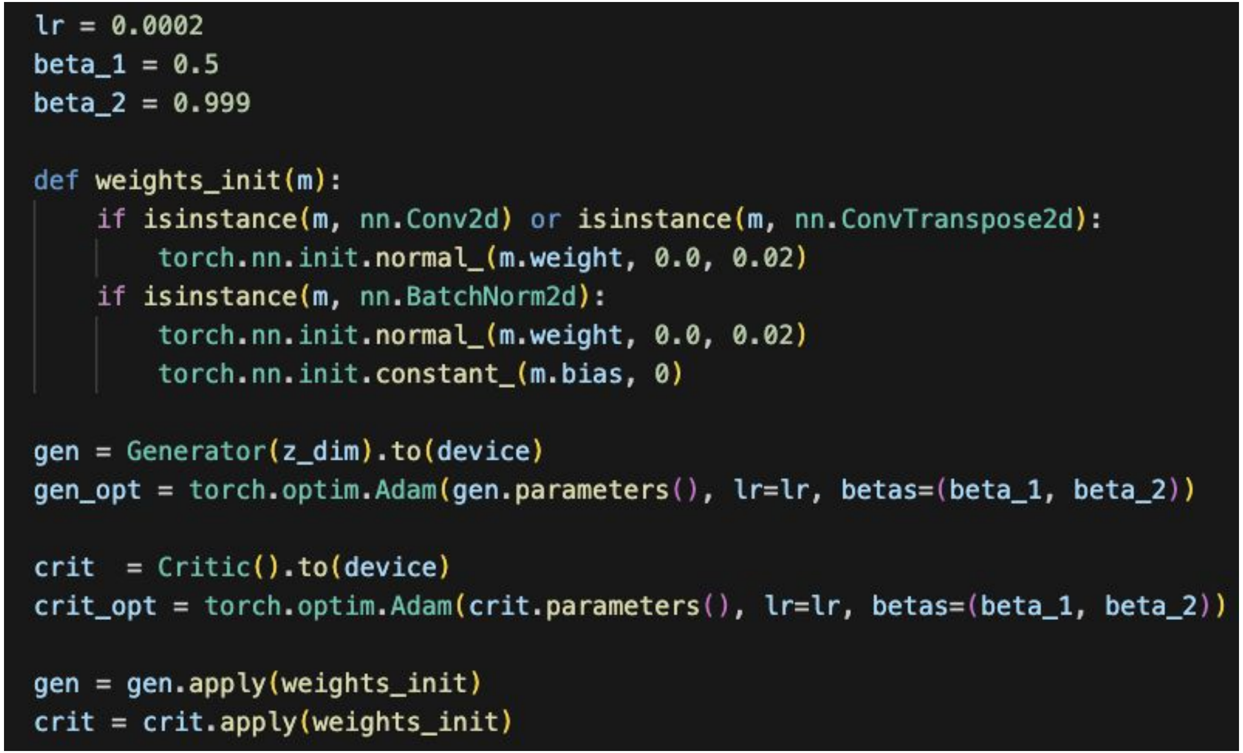
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1          [-1, 256, 3, 3]         230,656
       BatchNorm2d-2          [-1, 256, 3, 3]             512
              ReLU-3          [-1, 256, 3, 3]               0
   ConvTranspose2d-4          [-1, 128, 6, 6]         524,416
       BatchNorm2d-5          [-1, 128, 6, 6]             256
              ReLU-6          [-1, 128, 6, 6]               0
   ConvTranspose2d-7         [-1, 64, 13, 13]          73,792
       BatchNorm2d-8         [-1, 64, 13, 13]             128
              ReLU-9         [-1, 64, 13, 13]               0
  ConvTranspose2d-10          [-1, 1, 28, 28]           1,025
           Tanh-11          [-1, 1, 28, 28]               0
================================================================
Total params: 830,785
Trainable params: 830,785
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.42
Params size (MB): 3.17
Estimated Total Size (MB): 3.59
----------------------------------------------------------------
Generator(
```

Red neuronal del Crítico:



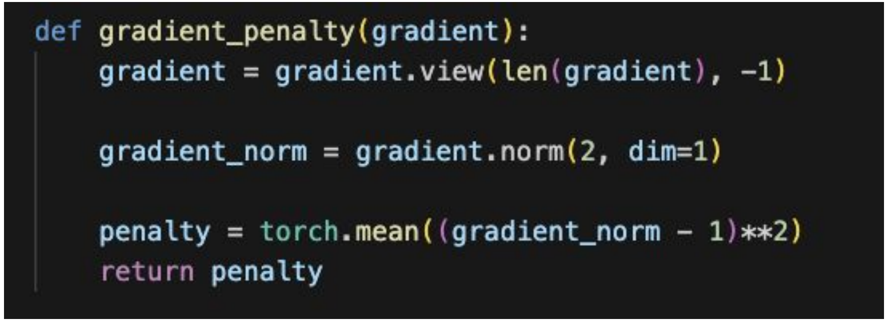
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 13, 13]           1,088
       BatchNorm2d-2           [-1, 64, 13, 13]             128
         LeakyReLU-3           [-1, 64, 13, 13]               0
            Conv2d-4            [-1, 128, 5, 5]         131,200
       BatchNorm2d-5            [-1, 128, 5, 5]             256
         LeakyReLU-6            [-1, 128, 5, 5]               0
            Conv2d-7              [-1, 1, 1, 1]           2,049
================================================================
Total params: 134,721
Trainable params: 134,721
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.32
Params size (MB): 0.51
Estimated Total Size (MB): 0.84
----------------------------------------------------------------
Critic(
  (disc): Sequential(
    (0): Sequential(
      (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
...
```

Optimizador:



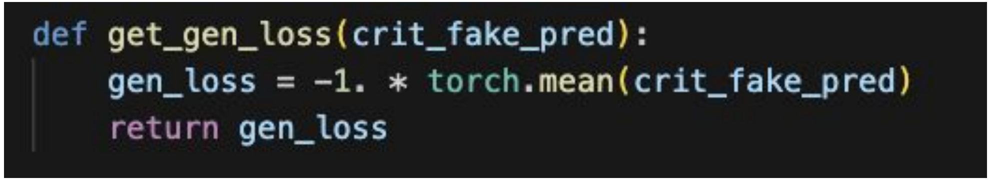
```python
lr = 0.0002
beta_1 = 0.5
beta_2 = 0.999

def weights_init(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
    if isinstance(m, nn.BatchNorm2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
        torch.nn.init.constant_(m.bias, 0)

gen = Generator(z_dim).to(device)
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr, betas=(beta_1, beta_2))

crit  = Critic().to(device)
crit_opt = torch.optim.Adam(crit.parameters(), lr=lr, betas=(beta_1, beta_2))

gen = gen.apply(weights_init)
crit = crit.apply(weights_init)
```

Gradient penalty:

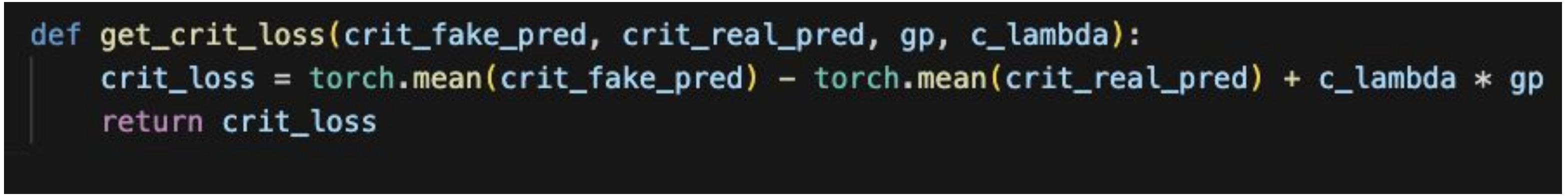


```python
def gradient_penalty(gradient):
    gradient = gradient.view(len(gradient), -1)

    gradient_norm = gradient.norm(2, dim=1)

    penalty = torch.mean((gradient_norm - 1)**2)
    return penalty
```

Loss:



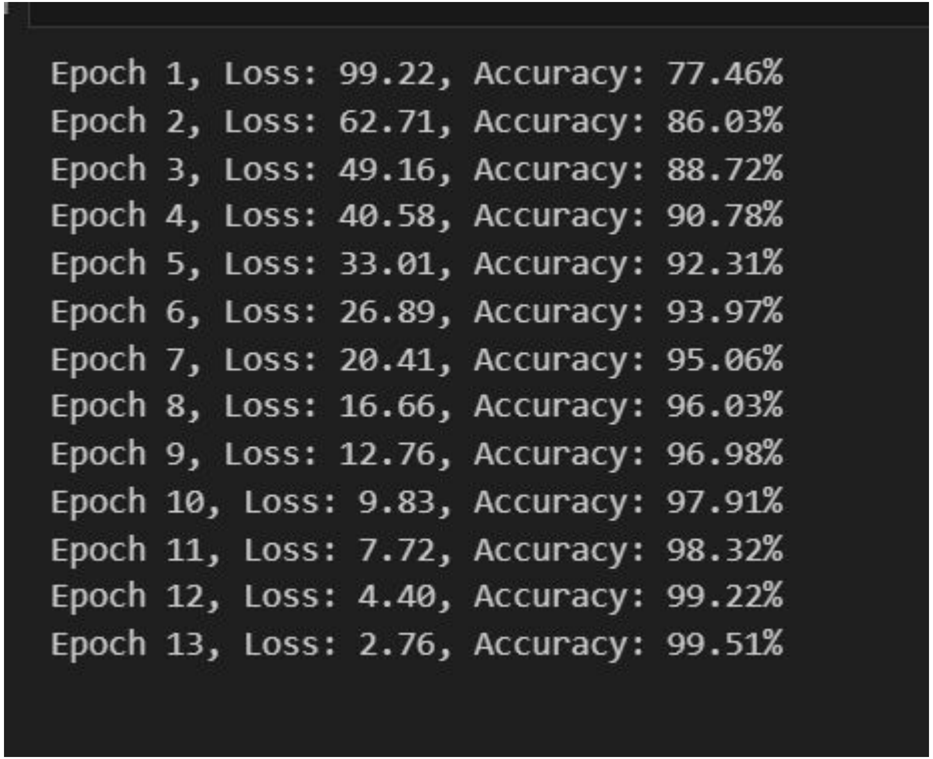
```python
def get_gen_loss(crit_fake_pred):
    gen_loss = -1. * torch.mean(crit_fake_pred)
    return gen_loss
```



```python
def get_crit_loss(crit_fake_pred, crit_real_pred, gp, c_lambda):
    crit_loss = torch.mean(crit_fake_pred) - torch.mean(crit_real_pred) + c_lambda * gp
    return crit_loss
```

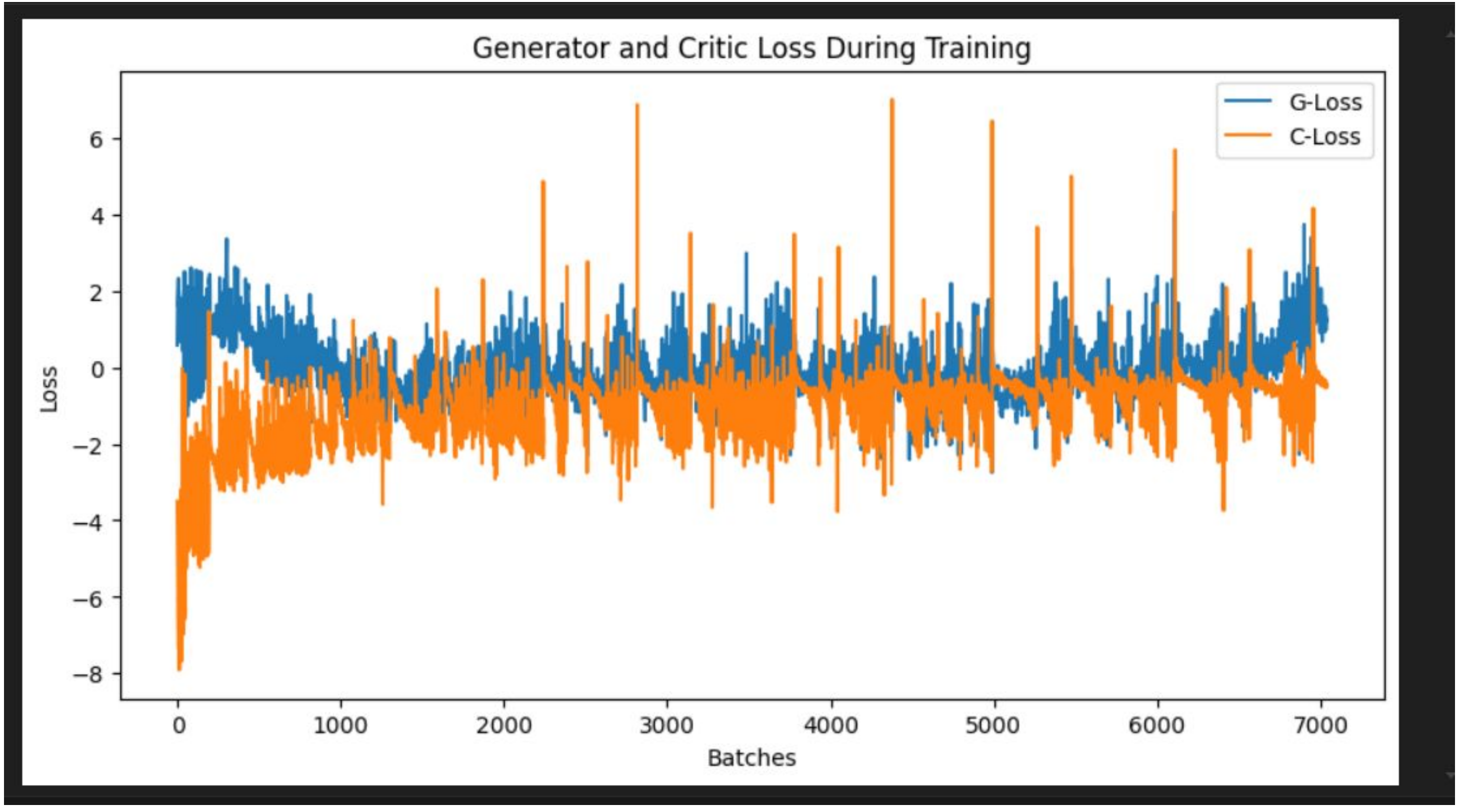
Resultados

Resultados de una ejecución:

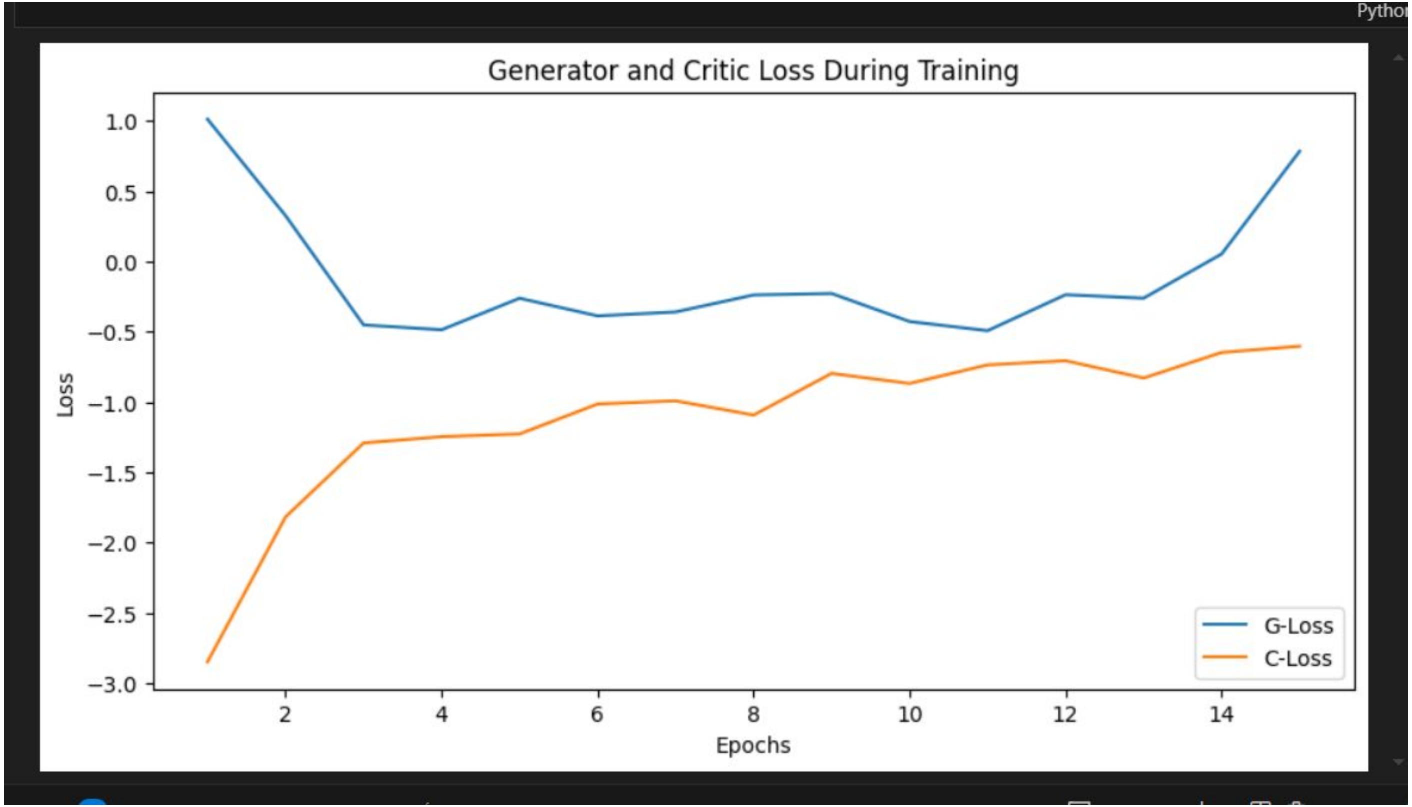


```
Epoch 1, Loss: 99.22, Accuracy: 77.46%
Epoch 2, Loss: 62.71, Accuracy: 86.03%
Epoch 3, Loss: 49.16, Accuracy: 88.72%
Epoch 4, Loss: 40.58, Accuracy: 90.78%
Epoch 5, Loss: 33.01, Accuracy: 92.31%
Epoch 6, Loss: 26.89, Accuracy: 93.97%
Epoch 7, Loss: 20.41, Accuracy: 95.06%
Epoch 8, Loss: 16.66, Accuracy: 96.03%
Epoch 9, Loss: 12.76, Accuracy: 96.98%
Epoch 10, Loss: 9.83, Accuracy: 97.91%
Epoch 11, Loss: 7.72, Accuracy: 98.32%
Epoch 12, Loss: 4.40, Accuracy: 99.22%
Epoch 13, Loss: 2.76, Accuracy: 99.51%
```

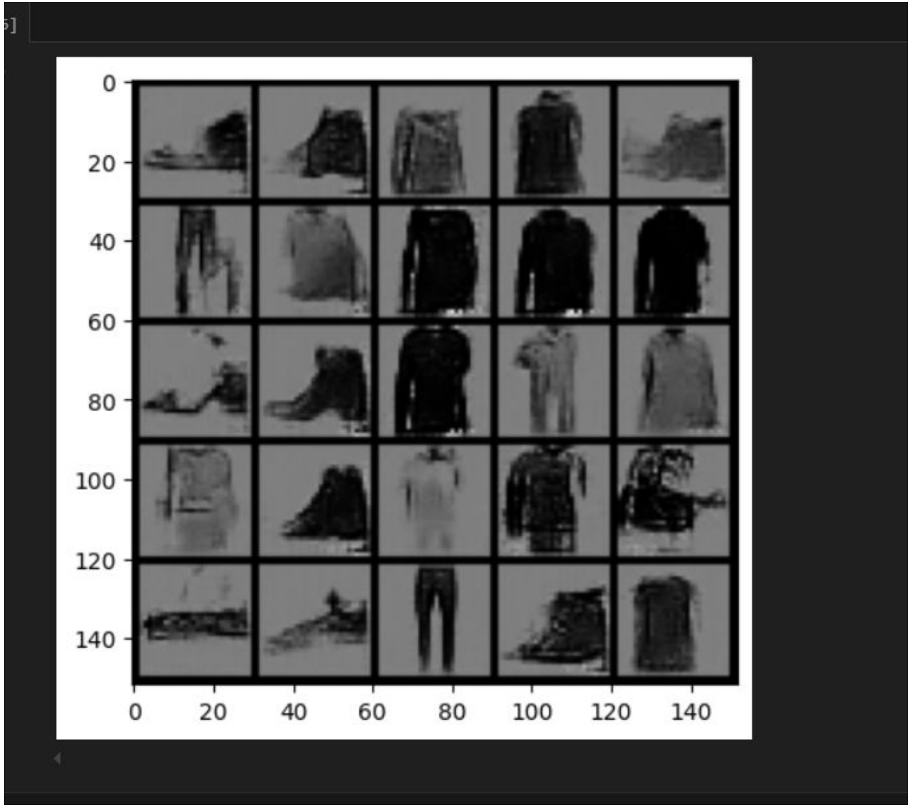
Training Loss (Generator and Critic) Batches:



Training Loss (Generator and Critic) Epochs:

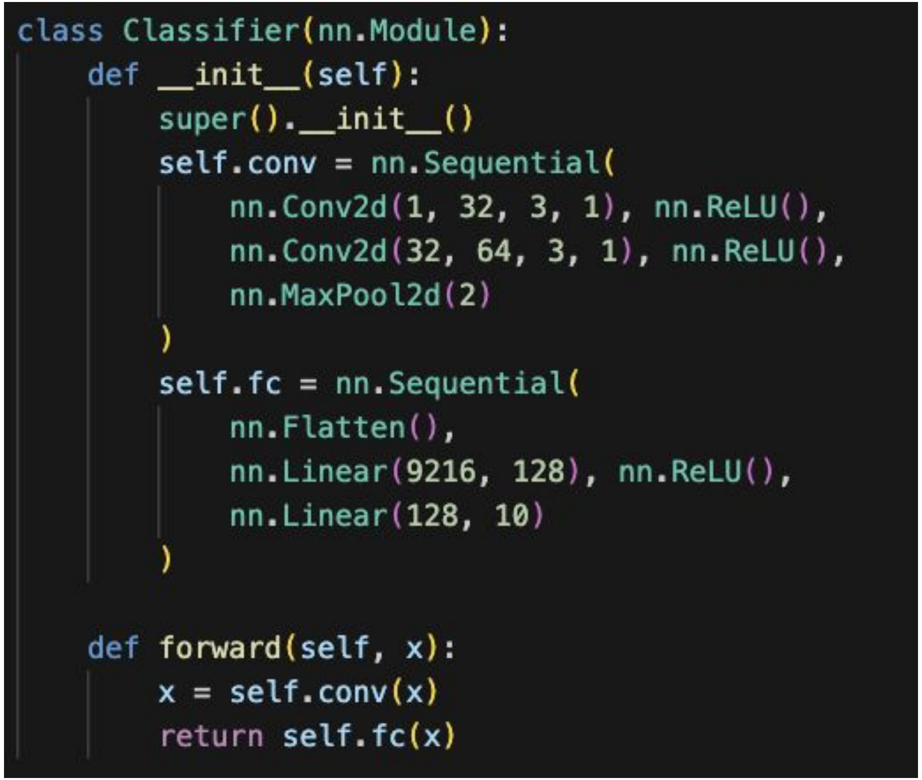


Ejemplo de Generación de Imágenes con la GAN:



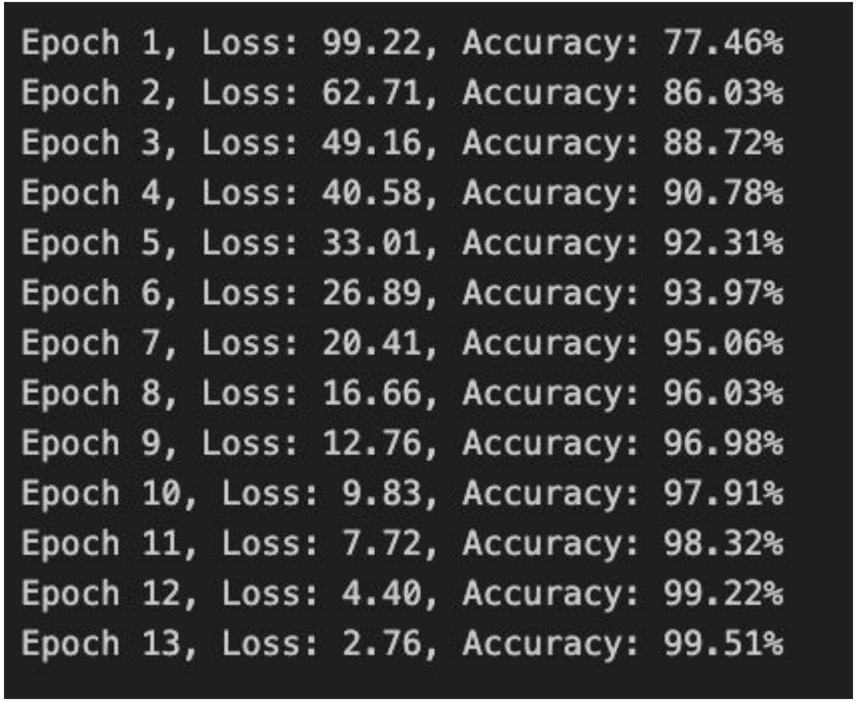
Entrenamiento del Clasificador

Clasificador:



```python
class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, 3, 1), nn.ReLU(),
            nn.Conv2d(32, 64, 3, 1), nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(9216, 128), nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.conv(x)
        return self.fc(x)
```

Entrenamiento del clasificador:



```
Epoch 1, Loss: 99.22, Accuracy: 77.46%
Epoch 2, Loss: 62.71, Accuracy: 86.03%
Epoch 3, Loss: 49.16, Accuracy: 88.72%
Epoch 4, Loss: 40.58, Accuracy: 90.78%
Epoch 5, Loss: 33.01, Accuracy: 92.31%
Epoch 6, Loss: 26.89, Accuracy: 93.97%
Epoch 7, Loss: 20.41, Accuracy: 95.06%
Epoch 8, Loss: 16.66, Accuracy: 96.03%
Epoch 9, Loss: 12.76, Accuracy: 96.98%
Epoch 10, Loss: 9.83, Accuracy: 97.91%
Epoch 11, Loss: 7.72, Accuracy: 98.32%
Epoch 12, Loss: 4.40, Accuracy: 99.22%
Epoch 13, Loss: 2.76, Accuracy: 99.51%
```

Distribución por clases del resultado del clasificador:

0 – T-shirt/top

1 – Trouser

2 – Pullover

3 – Dress

4 – Coat

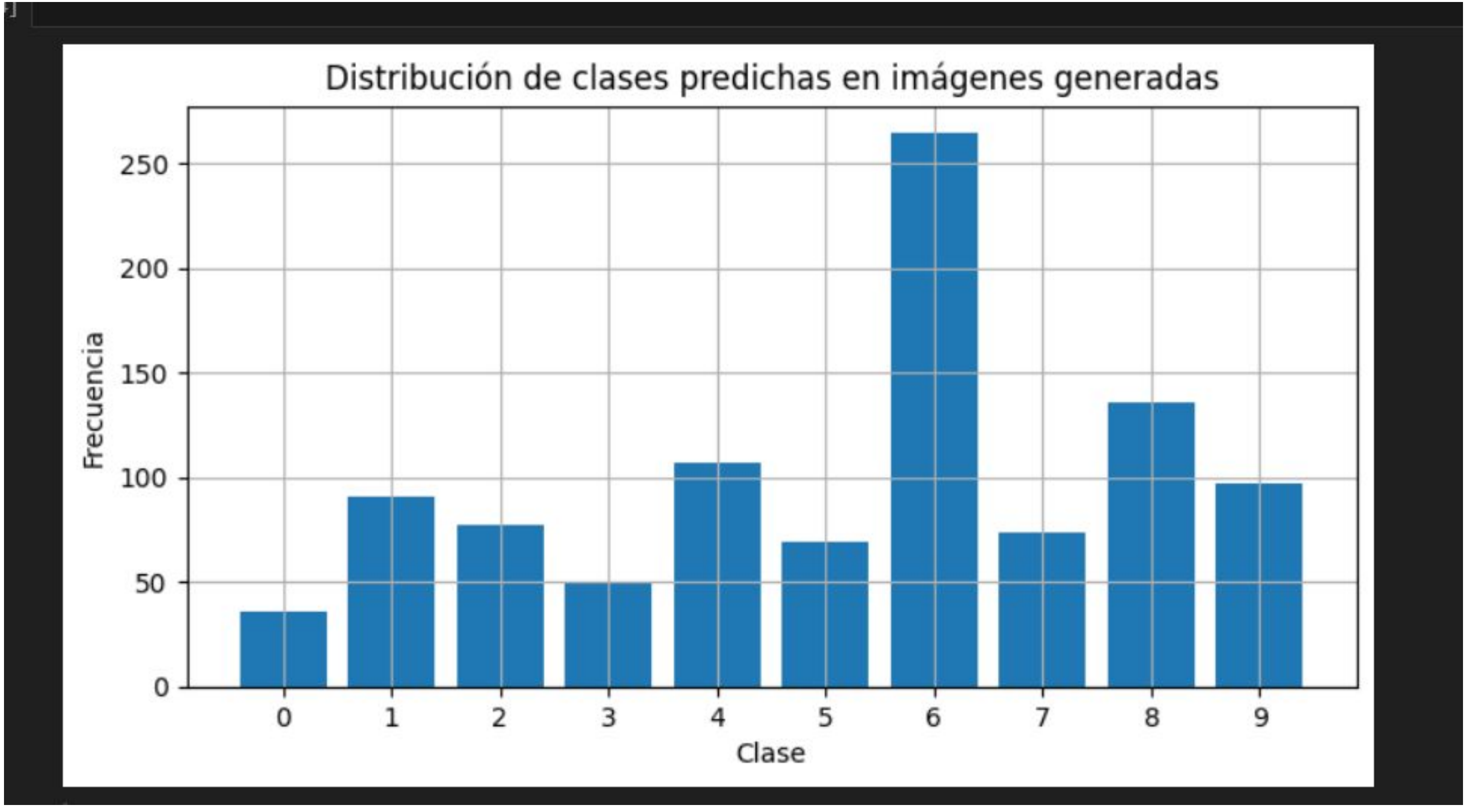
5 – Sandal

6 – Shirt

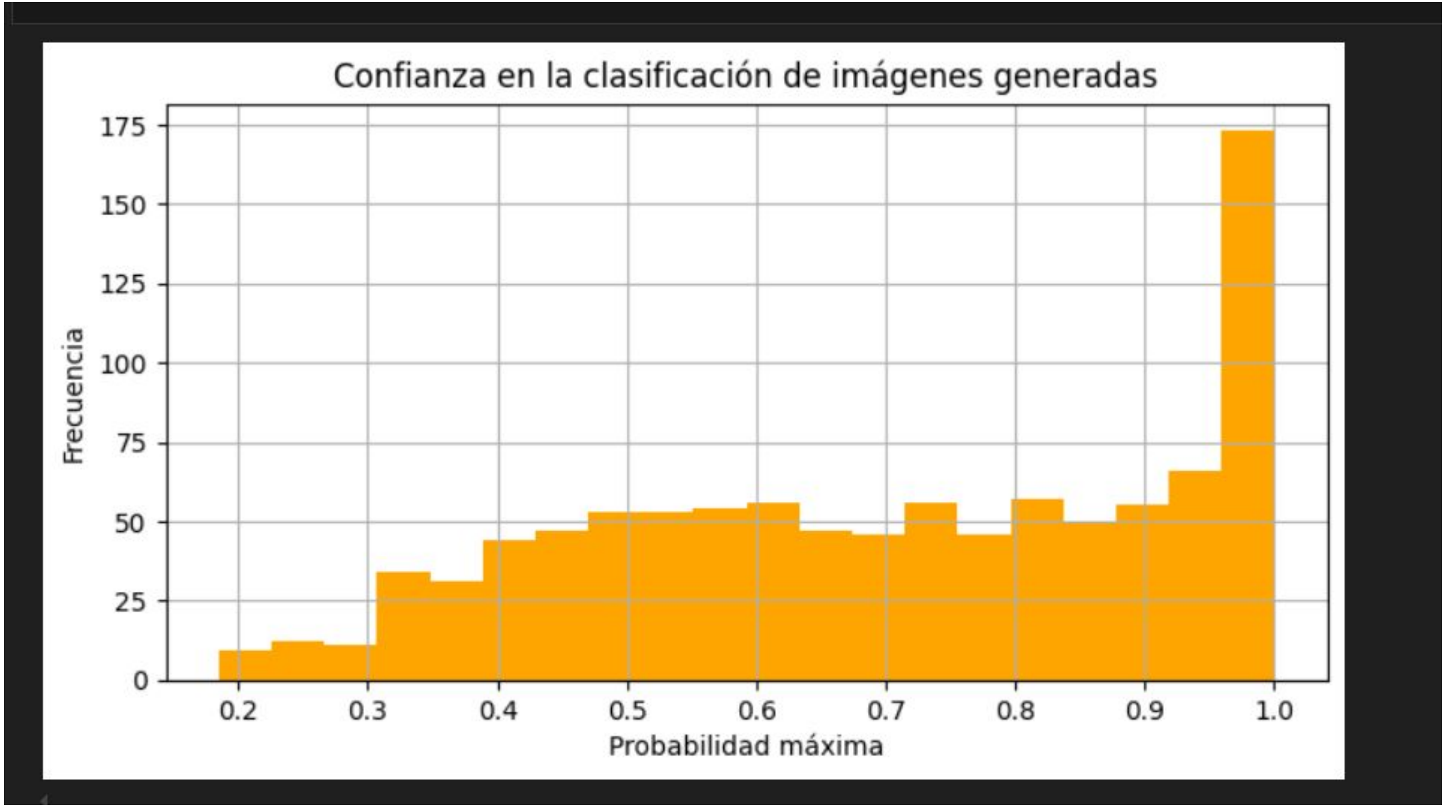
7 – Sneaker

8 – Bag

9 – Ankle boot



Confianza Obtenida de la generación de imágenes:



Resultados adicionales:

Link a resultados adicionales

Fin