Autoencoders

Adán González Rodríguez
Sara Porto Álvarez

Dataset y preparación

Dataset PneumoniaMNIST de MNIST: *hold-out*

```python
# Inicializa datasets
train_dataset = PneumoniaMNIST(split='train', transform=transform, download=True)
val_dataset   = PneumoniaMNIST(split='val', transform=transform, download=True)
test_dataset  = PneumoniaMNIST(split='test', transform=transform, download=True)

# Mostrar info para cada split
show_info("Train", train_dataset)
show_info("Validation", val_dataset)
show_info("Test", test_dataset)
```

```
Using downloaded and verified file: /root/.medmnist/pneumoniamnist.npz
Using downloaded and verified file: /root/.medmnist/pneumoniamnist.npz
Using downloaded and verified file: /root/.medmnist/pneumoniamnist.npz

--- TRAIN ---
Clase 1: 3494 imágenes
Clase 0: 1214 imágenes
Tamaño de imagen: torch.Size([1, 28, 28])

--- VALIDATION ---
Clase 1: 389 imágenes
Clase 0: 135 imágenes
Tamaño de imagen: torch.Size([1, 28, 28])

--- TEST ---
Clase 1: 390 imágenes
Clase 0: 234 imágenes
Tamaño de imagen: torch.Size([1, 28, 28])
```

Mejorando la división del dataset

Entrenamiento con imágenes sanas (label = 0)

```
### ENTRENAMIENTO SOLO CON IMÁGENES SANAS
# Obtener imágenes sanas (label == 0)
healthy_images = [img for img, label in train_dataset if label.item() == 0]
healthy_tensor = torch.stack(healthy_images)

# Dataset y dataloader
healthy_dataset = TensorDataset(healthy_tensor)
train_loader = DataLoader(healthy_dataset, batch_size=64, shuffle=True)
```

Conjunto balanceado de validación y test

```
Subconjunto validación: 270 imágenes (135 sanos + 135 con neumonía)
Subconjunto test: 468 imágenes (234 sanos + 234 con neumonía)
```

Autoencoders

Link al código de los Autoencoders

Autoencoder:

Encoder

```
# Encoder
self.encoder = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1),  # Aumentar los filtros
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),

    nn.Conv2d(32, 16, kernel_size=3, padding=1),  # Aumentar la profundidad
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),

    nn.Conv2d(16, 8, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, stride=1)
)
```

Decoder

```
# Decoder
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(8, 8, kernel_size=2, stride=2),     # -> [B, 8, 12, 12]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 8, kernel_size=2, stride=2),     # -> [B, 8, 24, 24]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 16, kernel_size=2, stride=2),    # -> [B, 16, 48, 48]
    nn.ReLU(),
    nn.Conv2d(16, 1, kernel_size=5, padding=2),            # -> [B, 1, 48, 48]
    nn.Upsample(size=(28, 28), mode='bilinear'),          # Asegura tamaño de salida
    nn.Sigmoid()
)
```

Entrenamiento del autoencoder

```python
### TRAIN
num_epochs = 200

for epoch in range(num_epochs):
    # ENTRENAMIENTO
    autoencoder.train()
    train_loss = 0.0
    for imgs, _ in train_loader:
        imgs = imgs.to(device)

        # Forward
        outputs = autoencoder(imgs)
        loss = criterion(outputs, imgs)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * imgs.size(0)

    train_loss /= len(train_loader.dataset)
```

```python
    # VALIDACIÓN
    autoencoder.eval()
    val_loss = 0.0
    with torch.no_grad():
        for imgs, _ in val_loader:
            imgs = imgs.to(device)
            outputs = autoencoder(imgs)
            loss = criterion(outputs, imgs)
            val_loss += loss.item() * imgs.size(0)

    val_loss /= len(val_loader.dataset)
    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}")
```

Resultados del entrenamiento del autoencoder

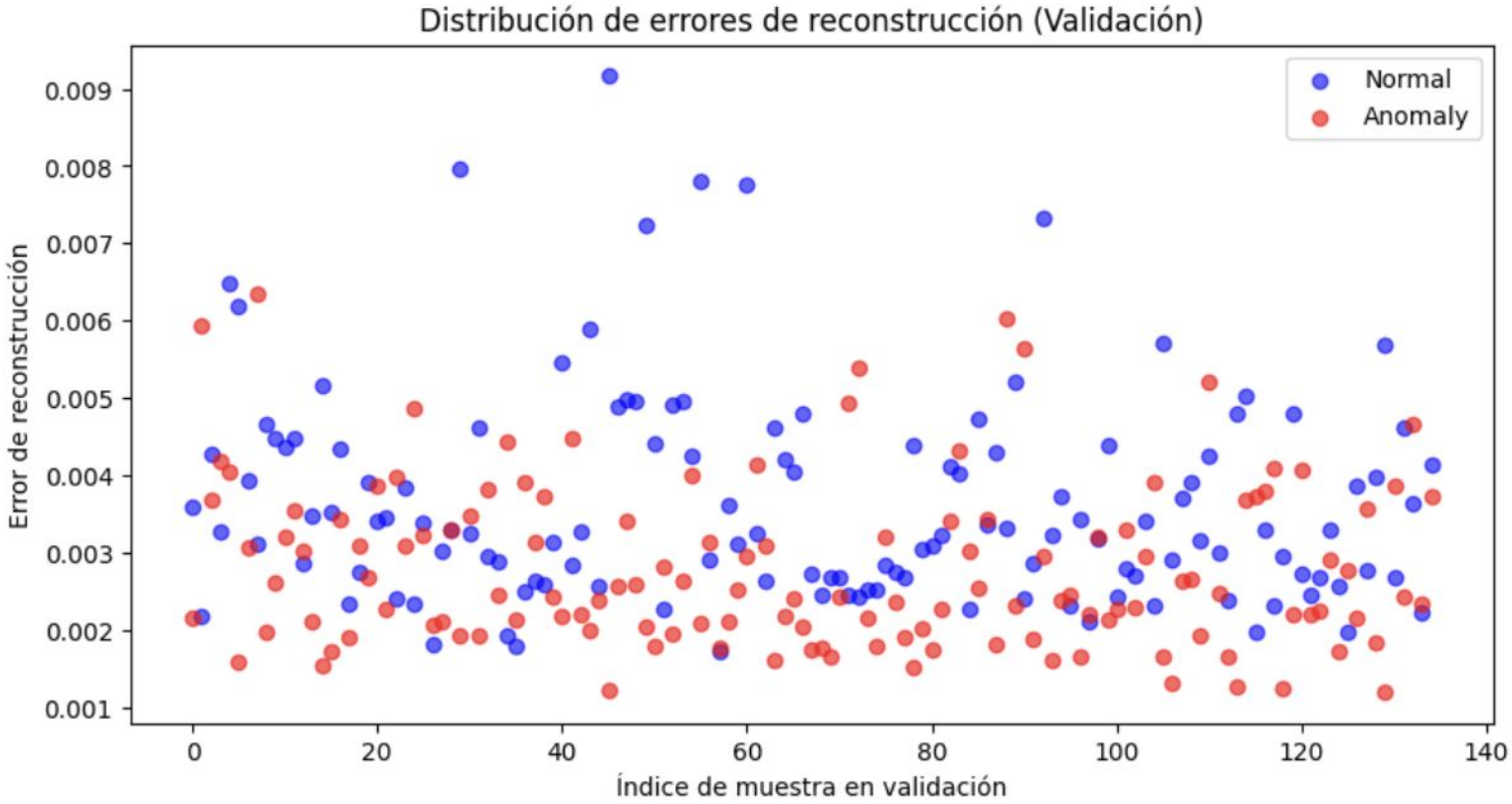
```
Epoch [1/200], Train Loss: 0.0340, Val Loss: 0.0299
Epoch [2/200], Train Loss: 0.0335, Val Loss: 0.0300
Epoch [3/200], Train Loss: 0.0324, Val Loss: 0.0257
Epoch [4/200], Train Loss: 0.0241, Val Loss: 0.0154
Epoch [5/200], Train Loss: 0.0175, Val Loss: 0.0129
Epoch [6/200], Train Loss: 0.0157, Val Loss: 0.0117
Epoch [7/200], Train Loss: 0.0145, Val Loss: 0.0109
Epoch [8/200], Train Loss: 0.0134, Val Loss: 0.0100
Epoch [9/200], Train Loss: 0.0125, Val Loss: 0.0097
Epoch [10/200], Train Loss: 0.0117, Val Loss: 0.0091
Epoch [11/200], Train Loss: 0.0111, Val Loss: 0.0087
Epoch [12/200], Train Loss: 0.0106, Val Loss: 0.0084
Epoch [13/200], Train Loss: 0.0102, Val Loss: 0.0082
Epoch [14/200], Train Loss: 0.0096, Val Loss: 0.0078
Epoch [15/200], Train Loss: 0.0093, Val Loss: 0.0075
Epoch [16/200], Train Loss: 0.0090, Val Loss: 0.0073
Epoch [17/200], Train Loss: 0.0087, Val Loss: 0.0074
Epoch [18/200], Train Loss: 0.0085, Val Loss: 0.0070
Epoch [19/200], Train Loss: 0.0082, Val Loss: 0.0068
Epoch [20/200], Train Loss: 0.0080, Val Loss: 0.0067
Epoch [21/200], Train Loss: 0.0078, Val Loss: 0.0065
Epoch [22/200], Train Loss: 0.0076, Val Loss: 0.0063
Epoch [23/200], Train Loss: 0.0075, Val Loss: 0.0063
Epoch [24/200], Train Loss: 0.0073, Val Loss: 0.0061
Epoch [25/200], Train Loss: 0.0072, Val Loss: 0.0060
...
Epoch [197/200], Train Loss: 0.0038, Val Loss: 0.0032
Epoch [198/200], Train Loss: 0.0037, Val Loss: 0.0033
Epoch [199/200], Train Loss: 0.0037, Val Loss: 0.0032
Epoch [200/200], Train Loss: 0.0037, Val Loss: 0.0032
```

Estadísticas sobre conjunto de validación

```python
errors = np.array(errors).flatten()
labels = np.array(labels).flatten()

error_normal = errors[labels == 0]
error_anomaly = errors[labels == 1]

mean_normal = np.mean(error_normal)
std_normal = np.std(error_normal)

print(f"Media (normales): {mean_normal:.4f}")
print(f"Desviación estándar (normales): {std_normal:.4f}")
print(f"Media (anomalías): {np.mean(error_anomaly):.4f}")
print(f"Desviación estándar (anomalías): {np.std(error_anomaly):.4f}")
```

```
Media (normales): 0.0036
Desviación estándar (normales): 0.0014
Media (anomalías): 0.0028
Desviación estándar (anomalías): 0.0011
```

Distribución de errores de reconstrucción



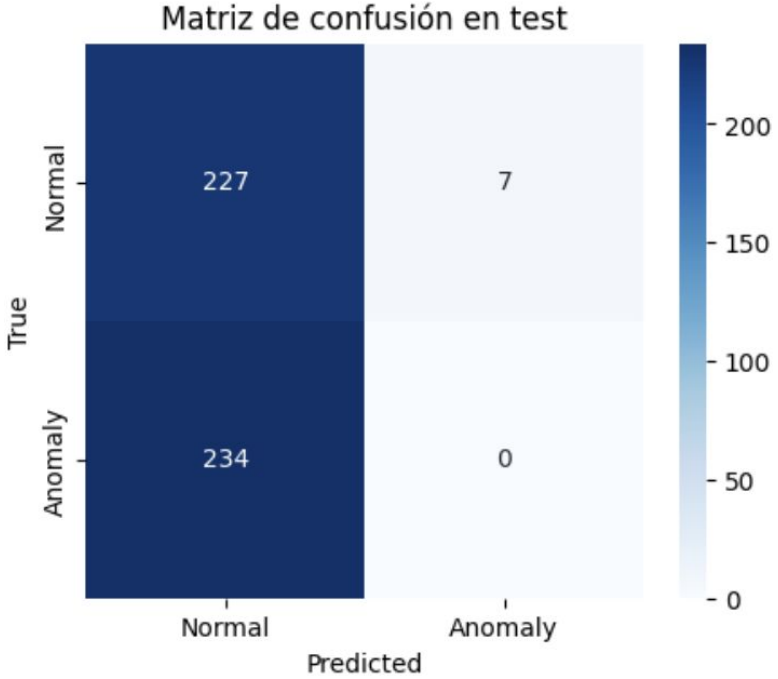
Mejor umbral (maximizando accuracy)

```python
thresholds = np.linspace(errors.min(), errors.max(), 100)
best_acc = 0
best_th = 0

for th in thresholds:
    preds = (errors > th).astype(int)  # 1 = anomalía
    acc = accuracy_score(labels, preds)
    if acc > best_acc:
        best_acc = acc
        best_th = th

print(f"\n🔍 Mejor umbral: {best_th:.4f} con accuracy: {best_acc:.4f}")
```

```
Mejor umbral: 0.0092 con accuracy: 0.5000
```

Matriz de confusión

```
Tasa de falsos positivos (FP rate): 0.0299
Tasa de falsos negativos (FN rate): 1.0000
```




```
Accuracy en test: 0.4850
```

❌**mal funcionamiento!**

Más pruebas de autoencoders

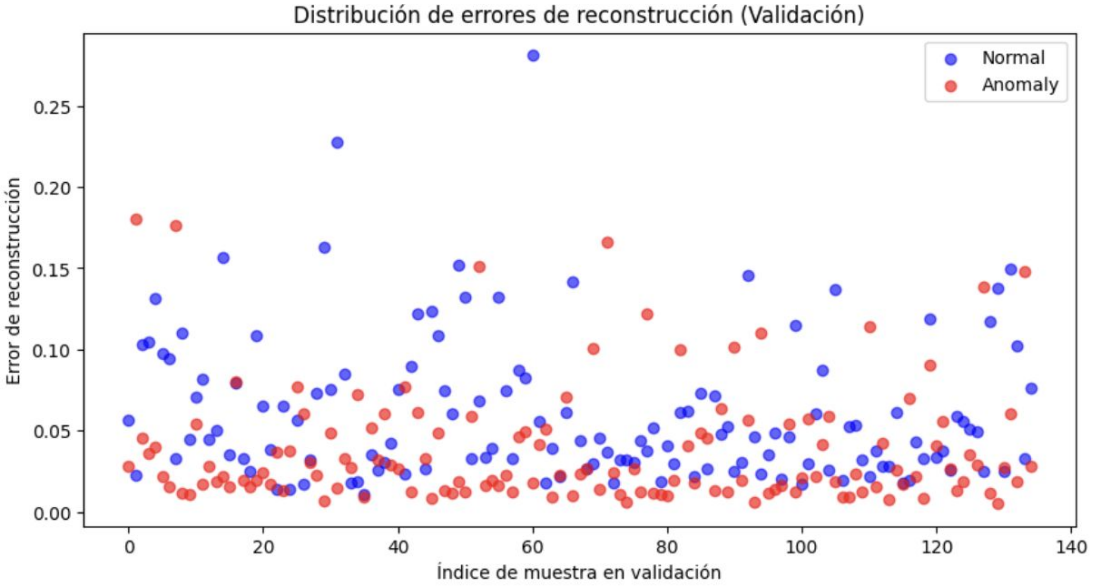
Link al código de pruebas adicionales de los autoencoders

Cambios en la red: la red

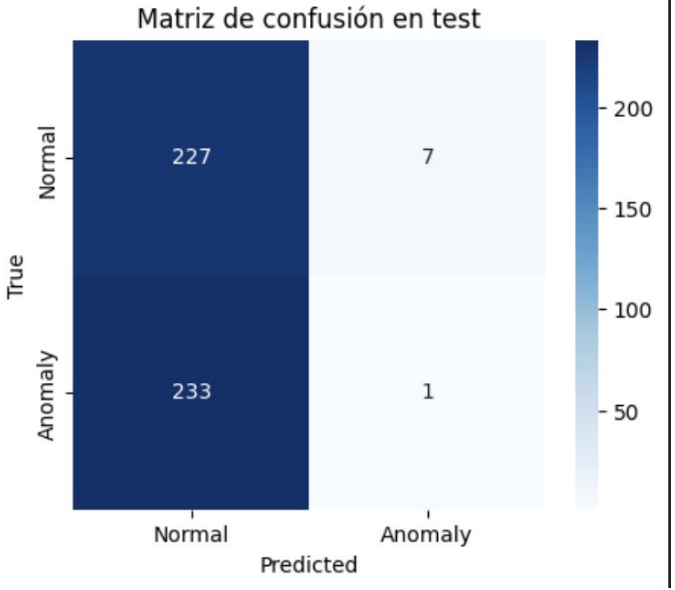
```python
# Encoder más relajado, pero con algo de capacidad
self.encoder = nn.Sequential(
    nn.Conv2d(1, 16, kernel_size=3, stride=2, padding=1),  # [B, 16, 14, 14]
    nn.ReLU(),
    nn.Conv2d(16, 8, kernel_size=3, stride=2, padding=1),  # [B, 8, 7, 7]
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Conv2d(8, 4, kernel_size=3, stride=2, padding=1),   # [B, 4, 4, 4]
    nn.ReLU()
)

self.decoder = nn.Sequential(
    nn.ConvTranspose2d(4, 8, kernel_size=2, stride=2),     # [B, 8, 8, 8]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 8, kernel_size=2, stride=2),     # [B, 8, 16, 16]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 16, kernel_size=2, stride=2),    # [B, 16, 32, 32]
    nn.ReLU(),
    nn.Conv2d(16, 1, kernel_size=3, padding=1),            # [B, 1, 32, 32]
    nn.Upsample(size=(28, 28), mode='bilinear'),
    nn.Sigmoid()
)
```

Cambios en la red: resultados

sigue teniendo mal funcionamiento!




```
Tasa de falsos positivos (FP rate): 0.0299
Tasa de falsos negativos (FN rate): 0.9957
```



Resultados adicionales Autoencoders:

Link a resultados adicionales de AEs

Variational Autoencoders

Link al código del VAE

VAE:

```
# Encoder
self.encoder = nn.Sequential(
    nn.Conv2d(1, 16, kernel_size=3, padding=1),   # -> [B, 16, 28, 28]
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),                    # -> [B, 16, 14, 14]

    nn.Conv2d(16, 8, kernel_size=3, padding=1),   # -> [B, 8, 14, 14]
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),                    # -> [B, 8, 7, 7]

    nn.Conv2d(8, 8, kernel_size=3, padding=1),    # -> [B, 8, 7, 7]
    nn.ReLU(),
    nn.MaxPool2d(2, stride=1)                     # -> [B, 8, 6, 6]
)

# Capa para calcular la media y log-variancia
self.fc_mu = nn.Linear(8 * 6 * 6, 128)   # Media del espacio latente
self.fc_logvar = nn.Linear(8 * 6 * 6, 128)  # Log de la varianza

# Decoder
self.decoder_fc = nn.Linear(128, 8 * 6 * 6)  # Decodificador
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(8, 8, kernel_size=2, stride=2),     # -> [B, 8, 12, 12]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 8, kernel_size=2, stride=2),     # -> [B, 8, 24, 24]
    nn.ReLU(),
    nn.ConvTranspose2d(8, 16, kernel_size=2, stride=2),    # -> [B, 16, 48, 48]
    nn.ReLU(),
    nn.Conv2d(16, 1, kernel_size=5, padding=2),            # -> [B, 1, 48, 48]
    nn.Upsample(size=(28, 28), mode='bilinear'),          # Asegura tamaño de salida
    nn.Sigmoid()
)
```

VAE: funciones

```python
def encode(self, x):
    # Pasamos por el encoder
    x = self.encoder(x)
    x = x.view(x.size(0), -1)  # Aplanamos la salida
    mu = self.fc_mu(x)  # Media
    logvar = self.fc_logvar(x)  # Log de la varianza
    return mu, logvar

def reparameterize(self, mu, logvar):
    # Reparameterization trick
    std = torch.exp(0.5 * logvar)  # Convertimos logvar a std
    eps = torch.randn_like(std)  # Muestreamos epsilon
    return mu + eps * std  # Generamos el espacio latente

def decode(self, z):
    # Decodificador
    z = self.decoder_fc(z)
    z = z.view(z.size(0), 8, 6, 6)  # Volvemos a dar forma a la salida
    return self.decoder(z)

def forward(self, x):
    # Pass hacia adelante
    mu, logvar = self.encode(x)
    z = self.reparameterize(mu, logvar)
    decoded = self.decode(z)
    return decoded, mu, logvar
```

Función de **pérdida** para el VAE

```python
# Función de pérdida
def vae_loss(recon_x, x, mu, logvar):

    # Error de reconstrucción
    BCE = F.binary_cross_entropy(recon_x.view(-1, 28 * 28), x.view(-1, 28 * 28), reduction='sum')

    # Divergencia KL
    MSE = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return BCE + MSE
```

Entrenamiento del autoencoder:

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Inicialización de modelo, optimizador y loss function
vae = VAE().to(device)
optimizer = optim.Adam(vae.parameters(), lr=1e-3)
num_epochs = 200

# Entrenamiento
def train(epoch):
    vae.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = vae(data)
        loss = vae_loss(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
    print(f"Train Epoch: {epoch} \tLoss: {train_loss / len(train_loader.dataset)}")

# Validación
def validate():
    vae.eval()
    val_loss = 0
    with torch.no_grad():
        for data, _ in val_loader:
            data = data.to(device)
            recon_batch, mu, logvar = vae(data)
            val_loss += vae_loss(recon_batch, data, mu, logvar).item()
    val_loss /= len(val_loader.dataset)
    print(f"Validation Loss: {val_loss}")

# Entrenamiento
for epoch in range(1, num_epochs + 1):
    train(epoch)
    validate()
```

Resultados del entrenamiento del autoencoder

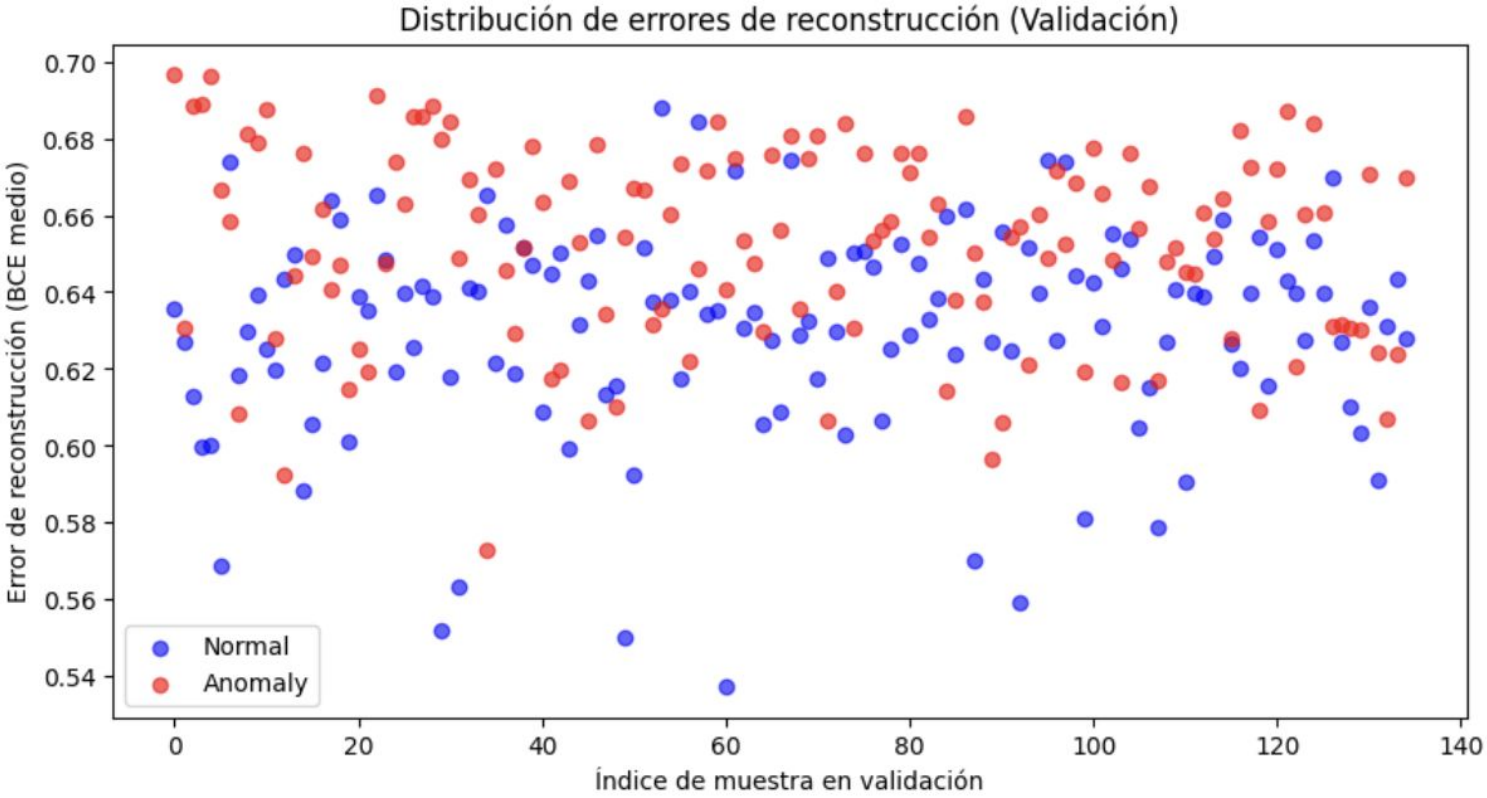
```
Train Epoch: 1  Loss: 540.866624407949
Validation Loss: 537.1332049334491
Train Epoch: 2  Loss: 539.3694269975289
Validation Loss: 537.19609375
Train Epoch: 3  Loss: 538.8164448620264
Validation Loss: 536.1902615017361
Train Epoch: 4  Loss: 536.0169732032537
Validation Loss: 530.0725423177083
Train Epoch: 5  Loss: 526.044038624897
Validation Loss: 524.0979727285879
Train Epoch: 6  Loss: 519.975220732084
Validation Loss: 521.6557816116898
Train Epoch: 7  Loss: 516.9502837983937
Validation Loss: 520.4124927662037
Train Epoch: 8  Loss: 515.5300578536861
Validation Loss: 519.7522569444444
Train Epoch: 9  Loss: 514.555828163612
Validation Loss: 519.1778085214121
Train Epoch: 10         Loss: 513.395001029654
Validation Loss: 518.4007631655093
Train Epoch: 11         Loss: 512.7580779576812
Validation Loss: 517.3221987123843
Train Epoch: 12         Loss: 512.3266867020182
Validation Loss: 517.7019205729167
Train Epoch: 13         Loss: 512.0882204360585
...
Train Epoch: 199        Loss: 501.1320129092875
Validation Loss: 507.54920247395836
Train Epoch: 200        Loss: 500.95173979355434
Validation Loss: 507.73188114872687
```

Estadísticas sobre conjunto de validación

```python
errors = np.array(errors).flatten()
labels = np.array(labels).flatten()

error_normal = errors[labels == 0]
error_anomaly = errors[labels == 1]

mean_normal = np.mean(error_normal)
std_normal = np.std(error_normal)

print(f"Media (normales): {mean_normal:.4f}")
print(f"Desviación estándar (normales): {std_normal:.4f}")
print(f"Media (anomalías): {np.mean(error_anomaly):.4f}")
print(f"Desviación estándar (anomalías): {np.std(error_anomaly):.4f}")
```

```
Media (normales): 0.6305
Desviación estándar (normales): 0.0270
Media (anomalías): 0.6526
Desviación estándar (anomalías): 0.0253
```

Distribución de errores de reconstrucción



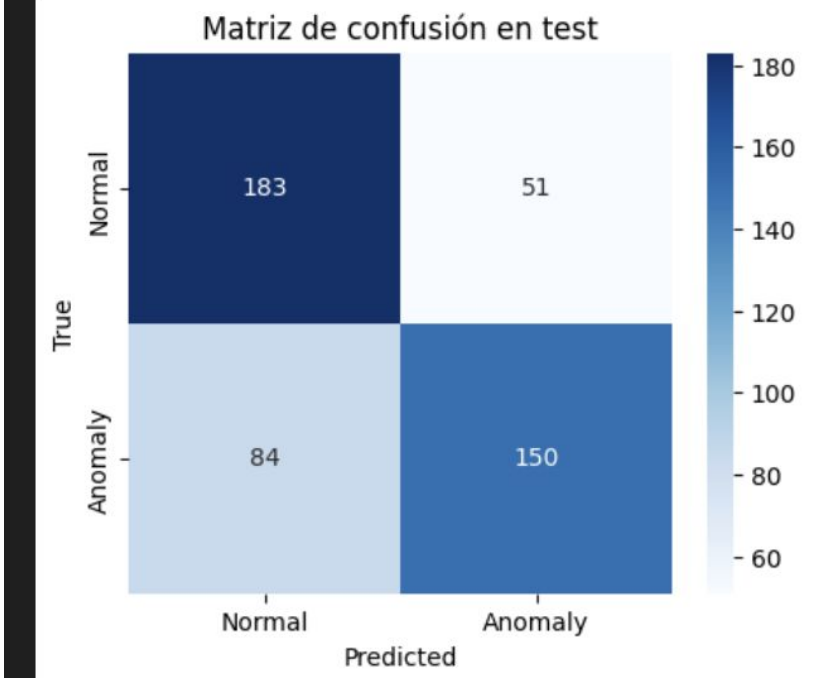
Mejor umbral (maximizando accuracy)

```python
thresholds = np.linspace(errors.min(), errors.max(), 200)
best_acc, best_th = 0, 0

for th in thresholds:
    preds = (errors > th).astype(int)
    acc = accuracy_score(labels, preds)
    if acc > best_acc:
        best_acc, best_th = acc, th

print(f"\n🔍 Mejor umbral: {best_th:.4f}  —  Accuracy: {best_acc:.4f}")
```

```
Mejor umbral: 0.6517  —  Accuracy: 0.7000
```

Matriz de confusión

```
🚨 Tasa de falsos positivos (FP rate): 0.2179
⚠️ Tasa de falsos negativos (FN rate): 0.3590
```




```
Accuracy en test: 0.7115
```

✅mejor funcionamiento!

Resultados adicionales VAEs:

Link a resultados adicionales de VAEs