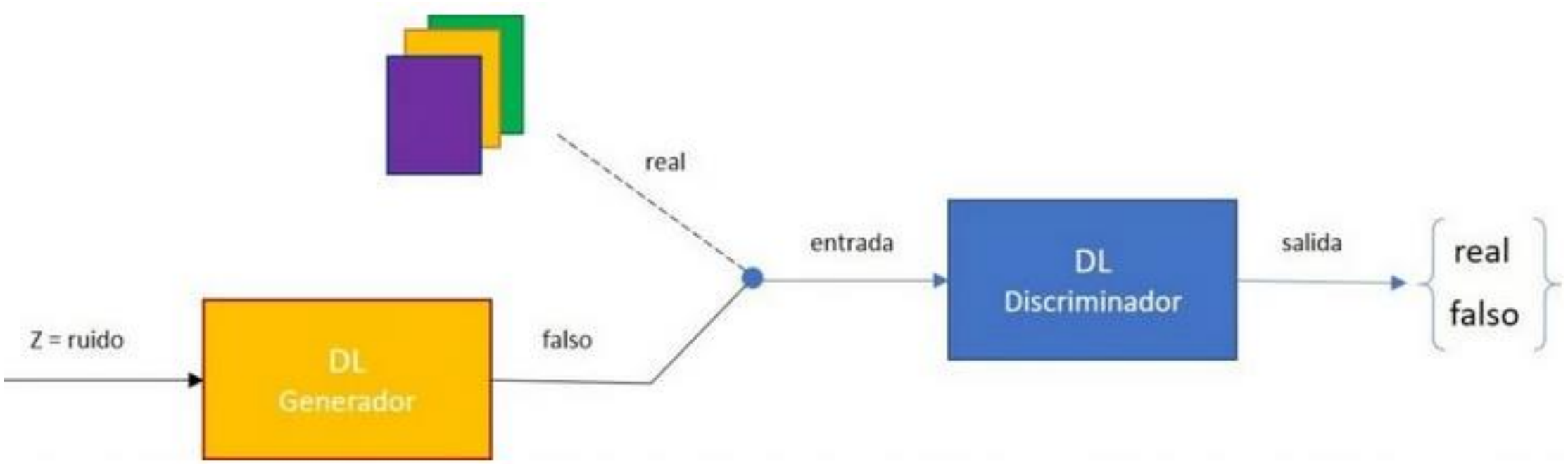
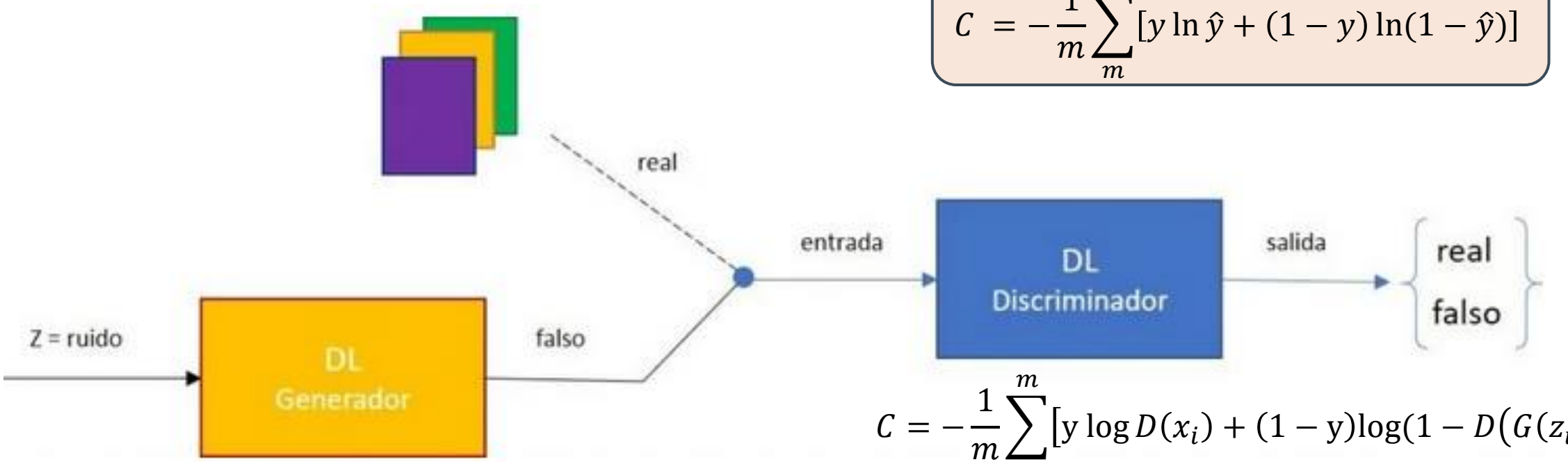
Redes generativas adversarias (GAN)



Redes generativas adversarias (GAN)

Entropía cruzada binaria

$$C = -\frac{1}{m}\sum_{m}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})]$$

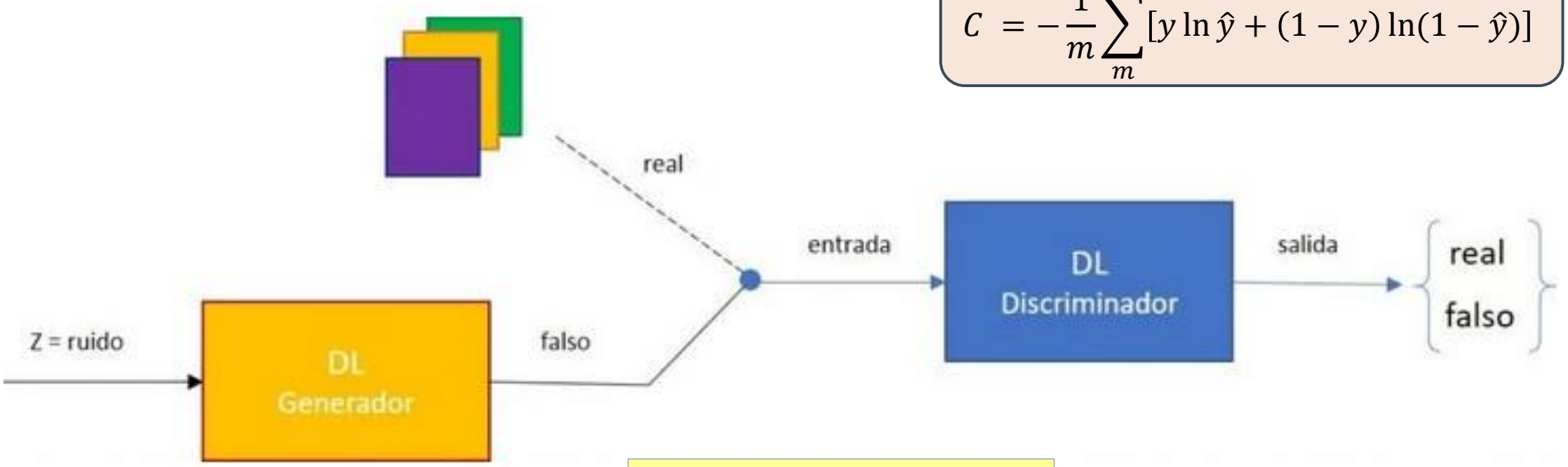


$$C = -\frac{1}{m}\sum_{i=1}^{m}\left[\mathrm{y}\log D(x_i) + (1-\mathrm{y})\log\left(1 - D\left(G(z_i)\right)\right)\right]$$

Redes generativas adversarias (GAN)

Entropía cruzada binaria

$$C = -\frac{1}{m}\sum_{m}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})]$$



$$C = -\frac{1}{m}\sum_{i=1}^{m}[\log(D(G(z_i)))]$$

Considera sólo cuando el discriminador clasifica incorrectamente las muestras generadas

Discriminador

```
def build_discriminator(img_shape):
    model = Sequential()
    model.add(Flatten(input_shape=img_shape))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten (Flatten)            (None, 784)               0
 dense (Dense)                (None, 512)               401920
 leaky_re_lu (LeakyReLU)      (None, 512)               0
 dense_1 (Dense)              (None, 256)               131328
 leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
 dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
```

```
img_shape = (28, 28, 1)

# Crear y compilar el discriminador
discriminator = build_discriminator(img_shape)
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
```

Generador

```
def build_generator(latent_dim):
    model = Sequential()
    model.add(Dense(256, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(1024))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(28 * 28 * 1, activation='tanh'))
    model.add(Reshape((28, 28, 1)))
    return model

latent_dim = 100
# Crear el generador
generator = build_generator(latent_dim)
```

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 256)	25856
leaky_re_lu_2 (LeakyReLU)	(None, 256)	0
batch_normalization	(None, 256)	1024
dense_4 (Dense)	(None, 512)	131584
leaky_re_lu_3 (LeakyReLU)	(None, 512)	0
batch_normalization_1	(None, 512)	2048
dense_5 (Dense)	(None, 1024)	525312
leaky_re_lu_4 (LeakyReLU)	(None, 1024)	0
batch_normalization_2	(None, 1024)	4096
dense_6 (Dense)	(None, 784)	803600
reshape (Reshape)	(None, 28, 28, 1)	0

Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584

Modelo combinado. Generador+Discriminador

```
latent_dim = 100

# El generador toma ruido como entrada y genera imágenes
z = Input(shape=(latent_dim,))
img = generator(z)

# En el modelo combinado solo se entrena el generador
discriminator.trainable = False

# Rta del discriminador
valid = discriminator(img)
```

```
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 100)]             0
 sequential_1 (Sequential)    (None, 28, 28, 1)         1493520
 sequential (Sequential)      (None, 1)                 533505

=================================================================
Total params: 2,027,025
Trainable params: 1,489,936
Non-trainable params: 537,089
```

```
# Modelo combinado (stacked generador y discriminador)
combined = Model(z, valid)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
```

Entrenamiento del modelo

```
for epoch in range(epochs):
```

Discriminador

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Seleccionar un conjunto aleatorio de imágenes reales

Entrenamiento del modelo

```
for epoch in range(epochs):
```

Discriminador

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Generar un conjunto de imágenes falsas

Entrenamiento del modelo

```
for epoch in range(epochs):
```

Discriminador

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Entrenar el discriminador

Entrenamiento del modelo

```
for epoch in range(epochs):
```

Discriminador

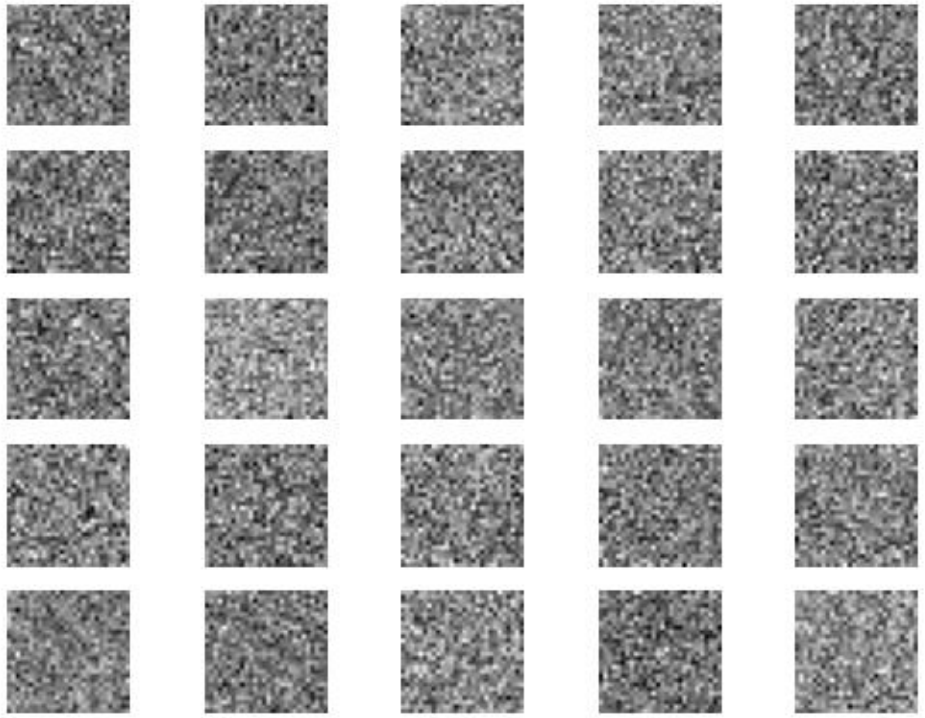
```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Entrena el generador para engañar al discriminador

GAN_MNIST.ipynb

Evolución del entrenamiento

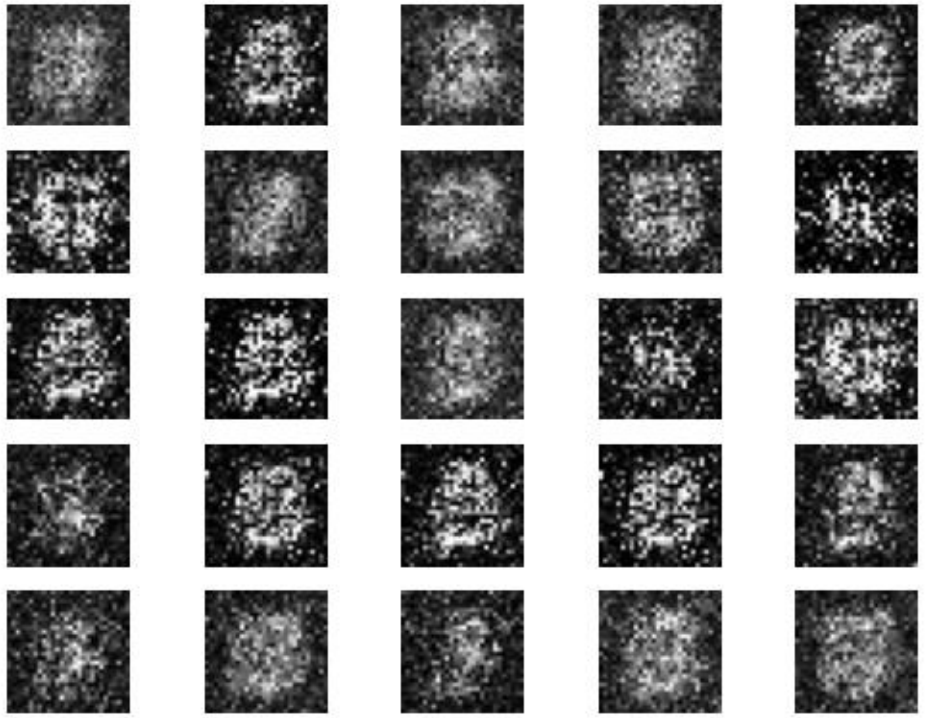
- Época 0



GAN_MNIST.ipynb

Evolución del entrenamiento

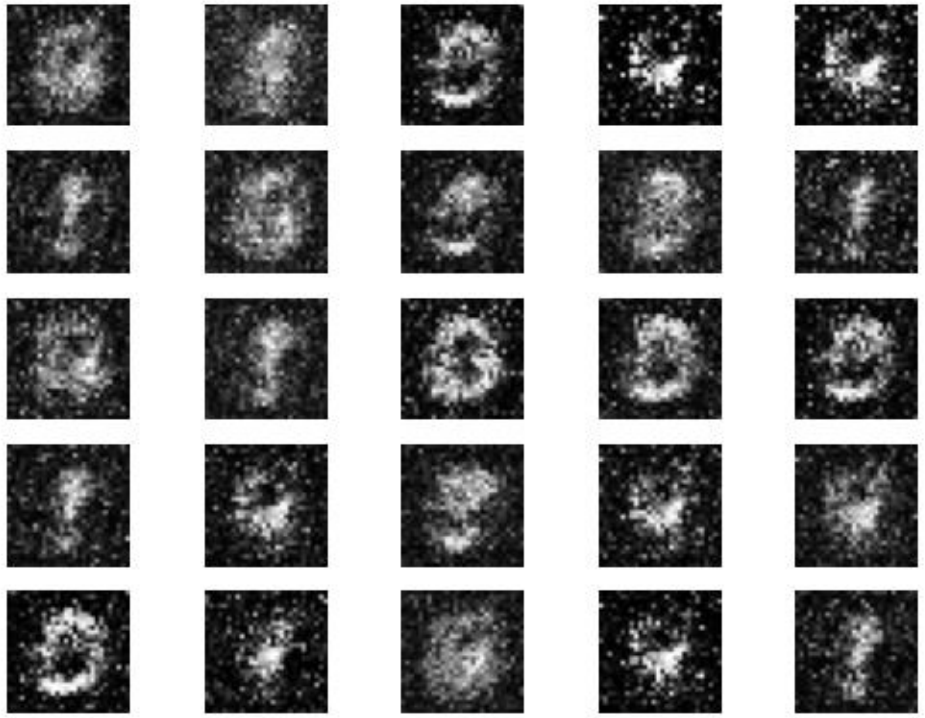
- Época 400



GAN_MNIST.ipynb

Evolución del entrenamiento

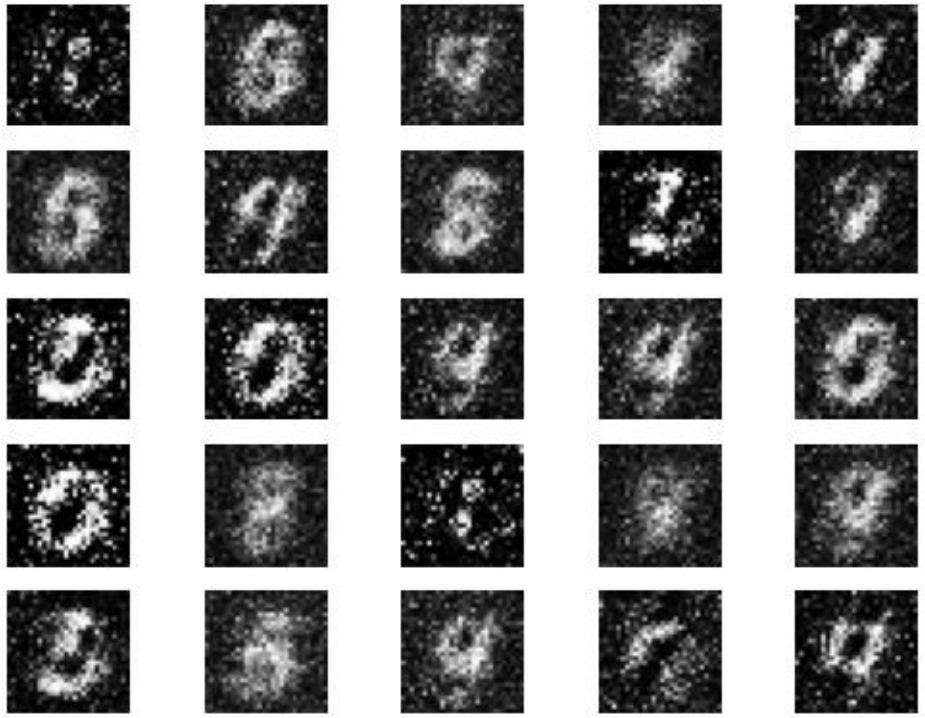
- Época 600



GAN_MNIST.ipynb

Evolución del entrenamiento

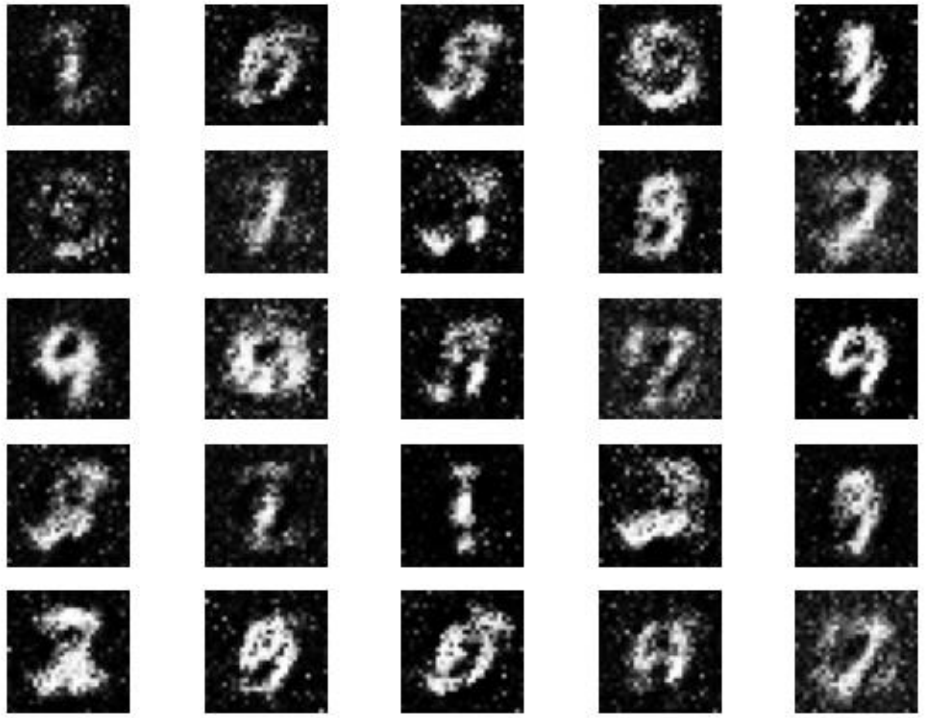
- Época 800



GAN_MNIST.ipynb

Evolución del entrenamiento

- Época 1600



GAN_MNIST.ipynb

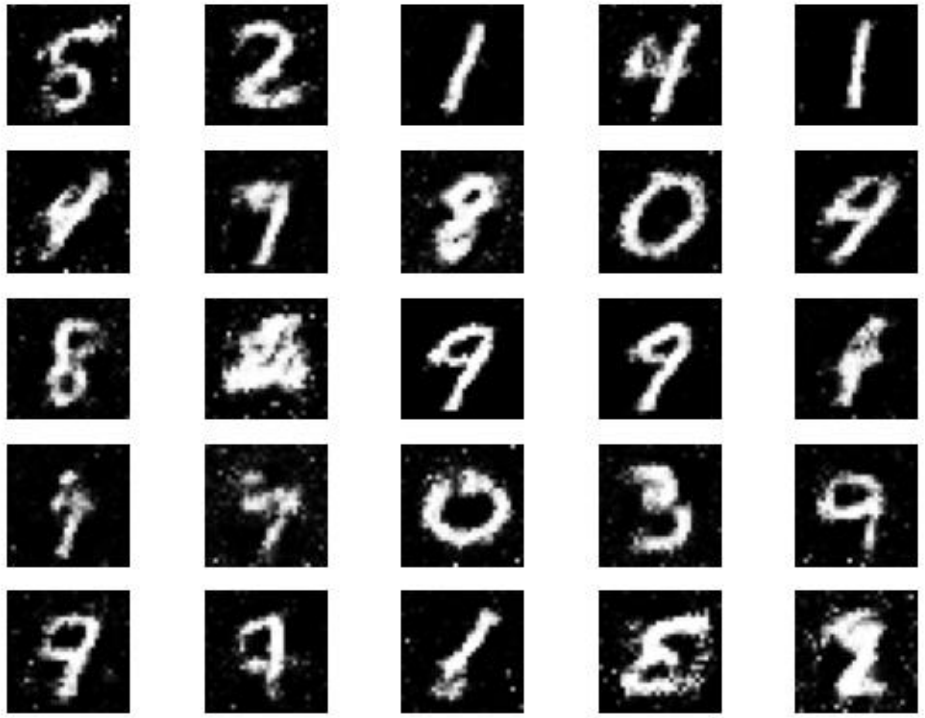
Evolución del entrenamiento

- Época 3200

GAN_MNIST.ipynb

Evolución del entrenamiento

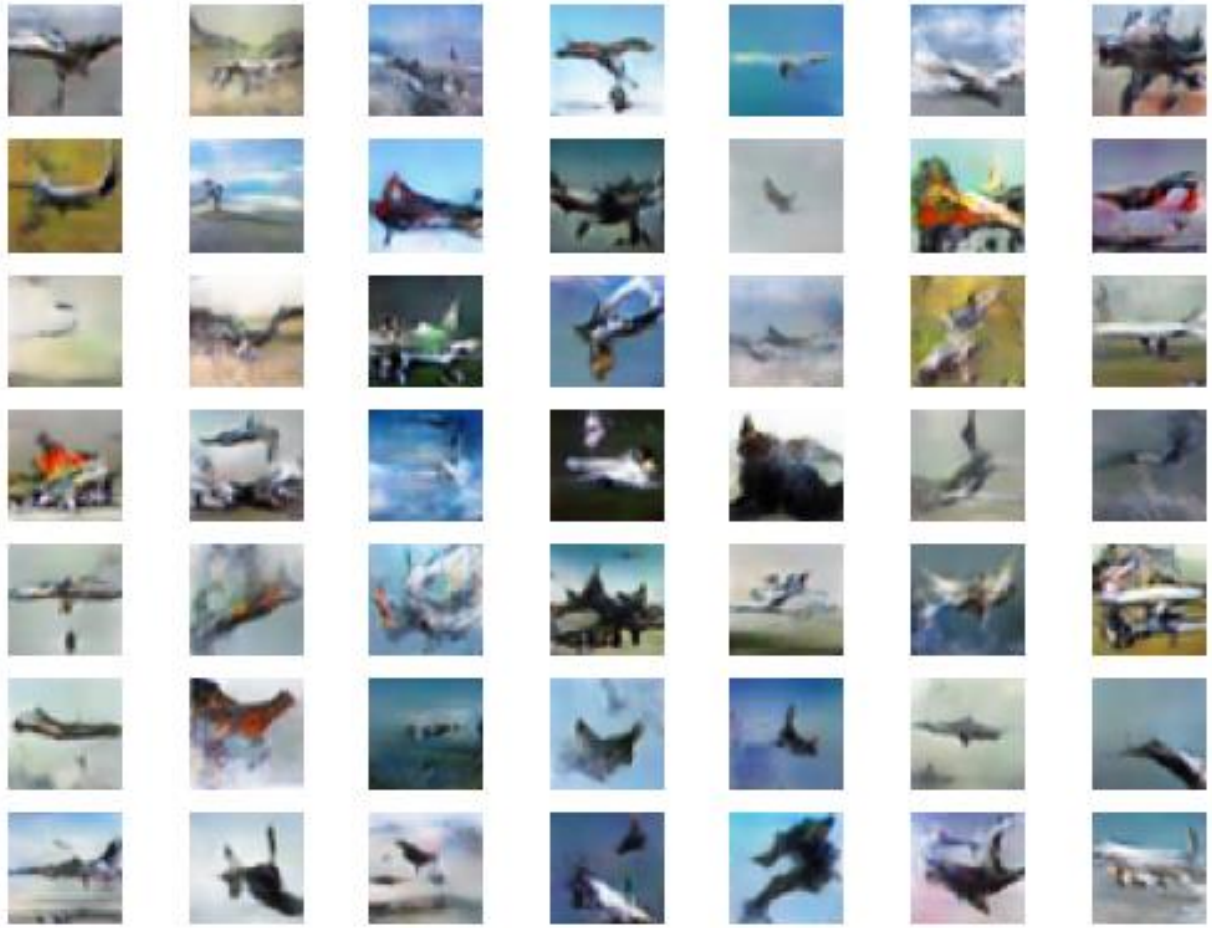
- Época 5200



GAN_MNIST.ipynb

GAN - Ejemplo

- En **GAN_CIFAR10.ipynb** se utiliza una red GAN para generar imágenes de aviones a partir de la clase 0 de CIFAR10.
- Esta es la respuesta del generador después de entrenar la GAN durante 100 épocas.

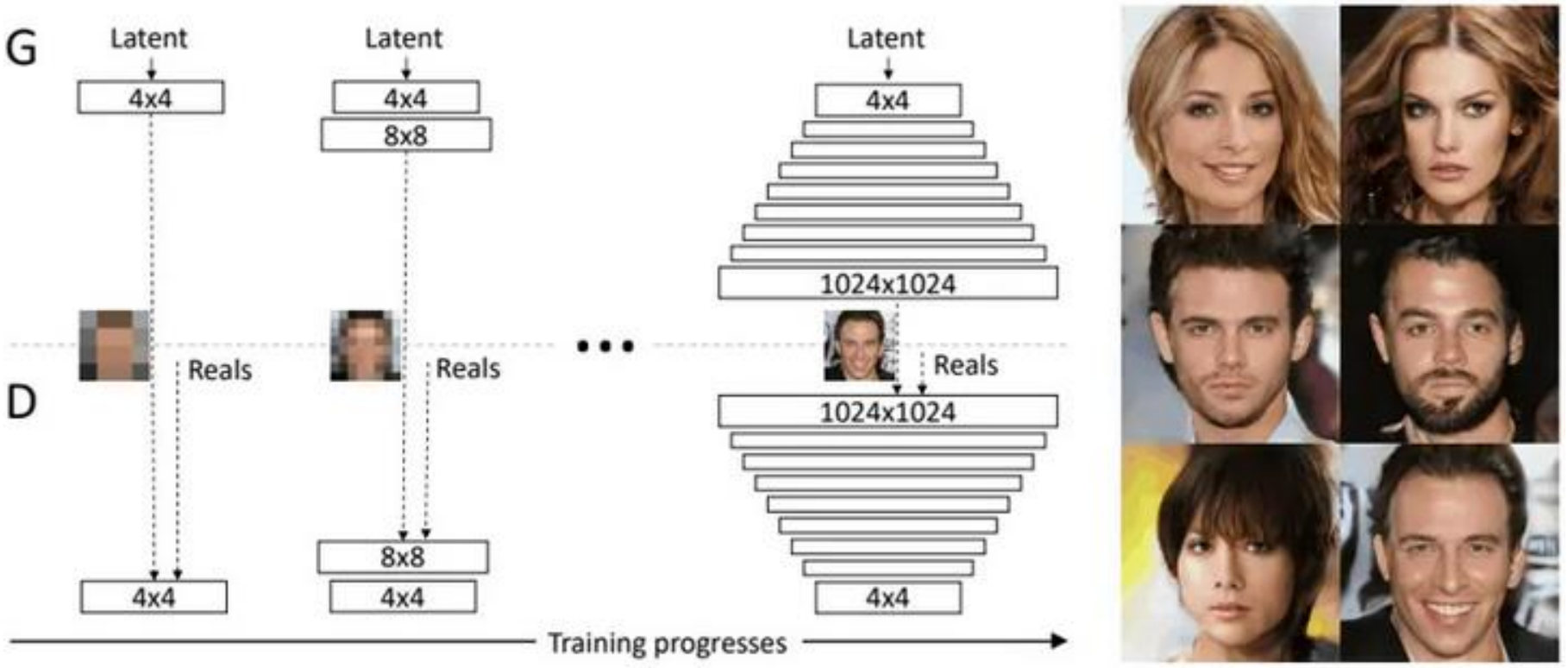


Ejemplo: Generando dormitorios

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (2016)

Redes generativas progresivas

- La calidad generativa del modelo mejora incrementando el tamaño de las imágenes en forma progresiva



thispersondoesnotexist.com

- thispersondoesnotexist.com es una web creada por nvidia que crea caras en alta definición de personas que no existen.
- Ejemplo: 3 caras generadas