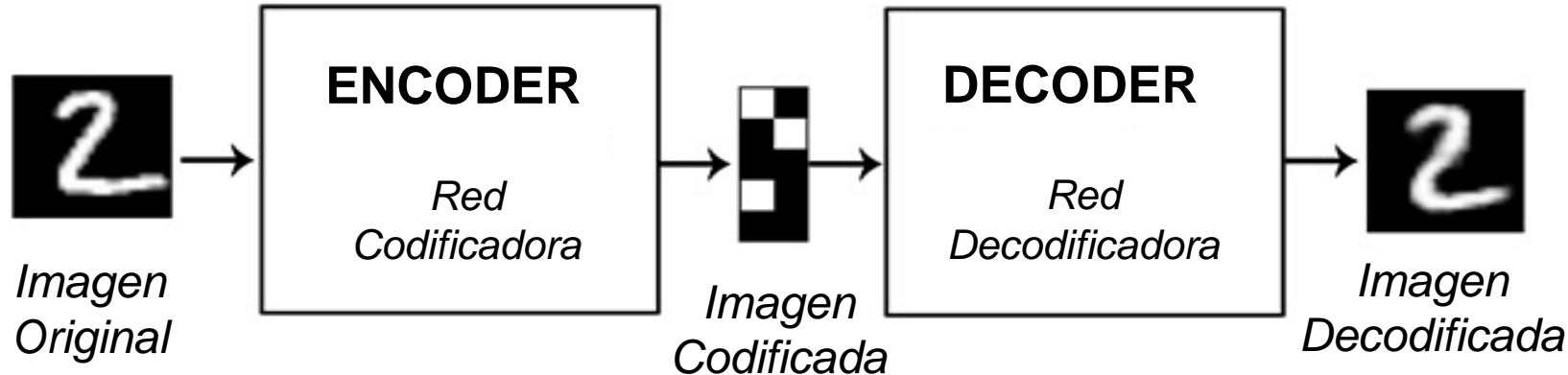


AUTOENCODERS

(AUTO CODIFICADORES)

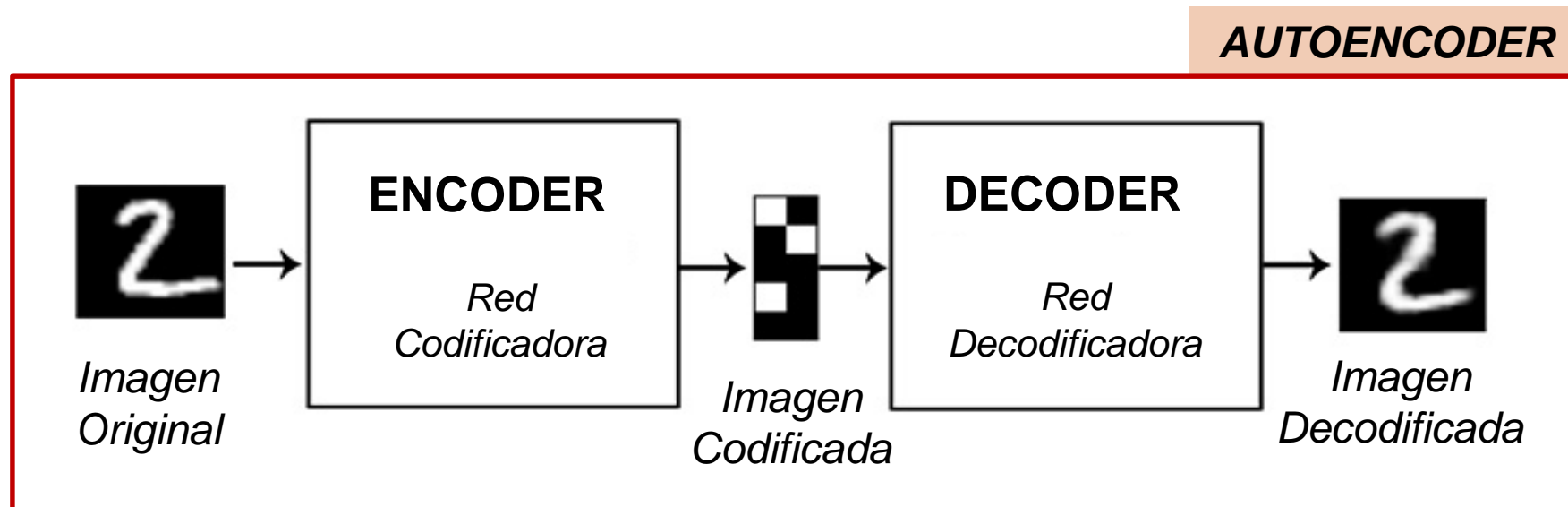
Autoencoding (Auto codificación)

- Es un algoritmo de compresión de datos en el que las funciones de compresión y descompresión son
 - ▣ Específicas de los datos
 - ▣ Con pérdida
 - ▣ Aprendidas automáticamente



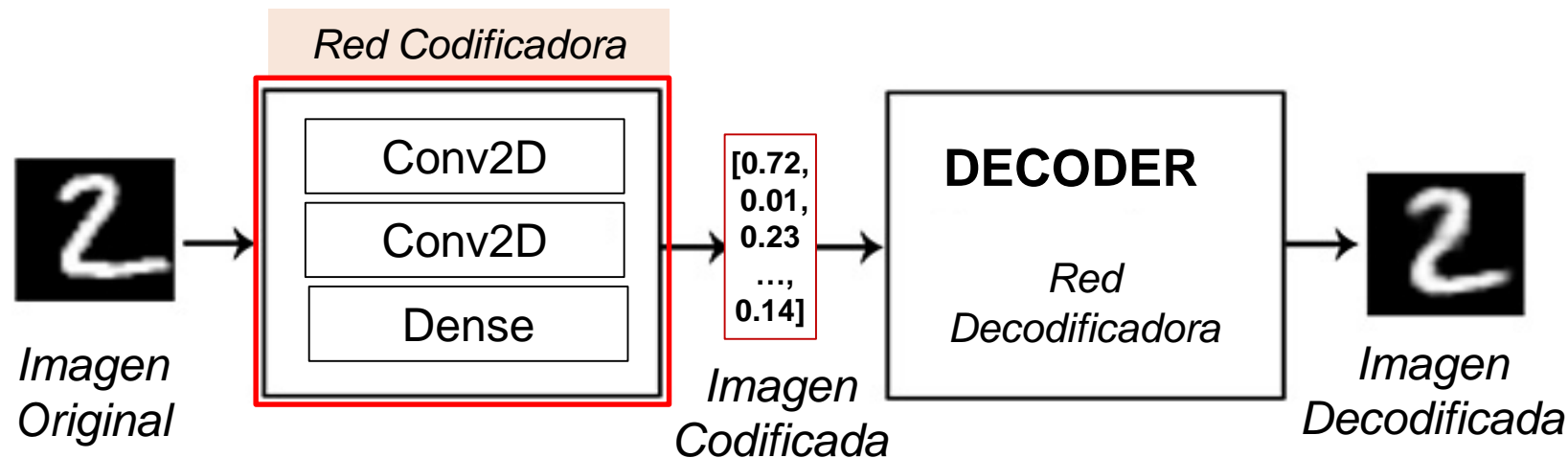
Autoencoder

- Entrenamiento
 - ▣ La entrada y la salida son iguales.
 - ▣ Entrenamiento NO supervisado aunque usa mecanismos supervisados.
- Tres modelos: encoder, decoder, autoencoder=decoder(encoder(x))



Red Codificadora

- Genera una representación vectorial de tamaño K
 - ▣ $K \ll$ tamaño original de la imagen
- La nueva representación comprime la imagen
- Es una red neuronal feedforward o convolucional



Red Decodificadora

- Recibe la imagen codificada y genera una imagen de igual tamaño que la imagen de entrada.
- Descomprime la imagen.
- Es una red neuronal feedforward o convolucional

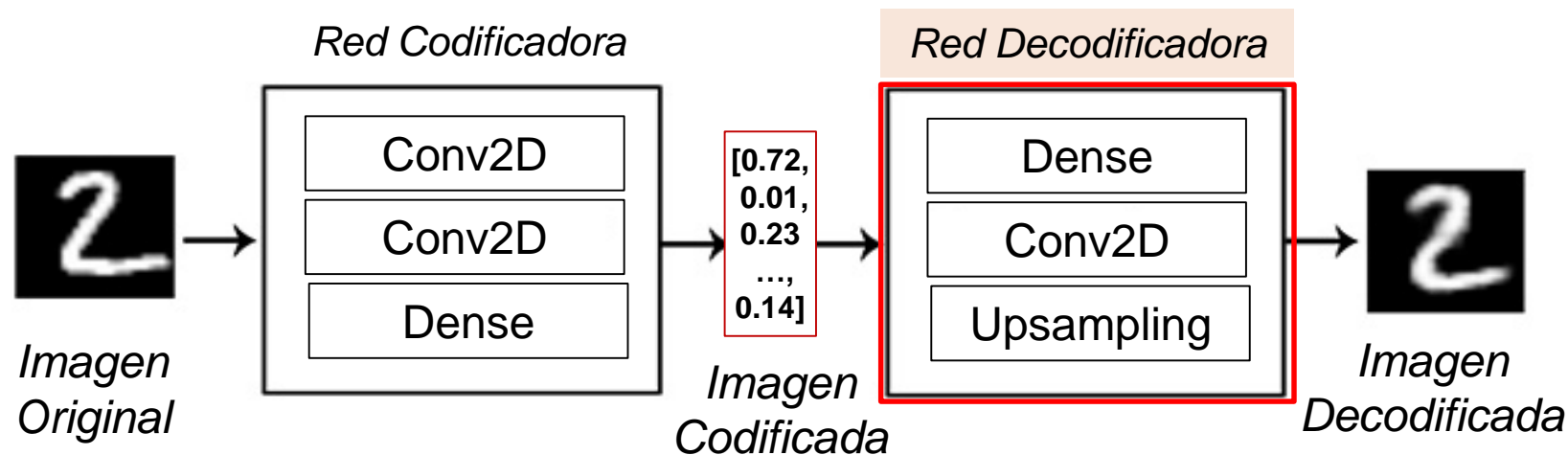
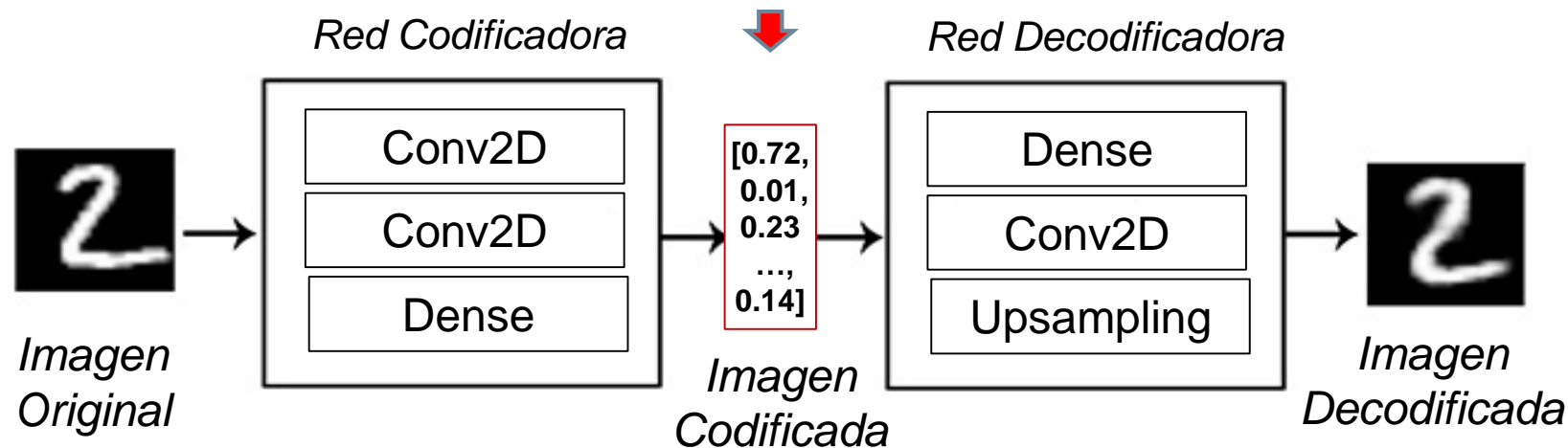


Imagen codificada

- Es un vector numérico de tamaño K (vector latente)
- El valor de K es arbitrario
 - ▣ A mayor valor de K mejora la representación y se reduce la compresión.

Su valor se infiere



Ejemplo

- Construiremos un autoencoder para comprimir los dígitos de MNIST



- Tanto el codificador como el decodificador estarán formados por una única capa densa.

Carga de datos

```
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
```

Nombre ▲	Tipo	Tamaño
input_dim	int	1
x_test	Array of uint8	(10000, 28, 28)
x_train	Array of uint8	(60000, 28, 28)

```
(x_train, _), (x_test, _) = mnist.load_data()
```

```
input_dim = 28*28
```

```
x_train = np.reshape(x_train, [-1, input_dim])/255.0
```

```
x_test = np.reshape(x_test, [-1, input_dim])/255.0
```

```
print(x_train.shape, x_test.shape, input_dim)
```

Nombre ▲	Tipo	Tamaño
input_dim	int	1
x_test	Array of floa...	(10000, 784)
x_train	Array of floa...	(60000, 784)

Modelos Encoder y Decoder

```

encoding_dim = 32  #factor de compresion 24.5, si entrada de 28x28=784
##----- ENCODER -----
encoder_input = Input(shape=(input_dim,), name='encoder_input')
code = Dense(encoding_dim, activation='relu',
              name='latent_vector')(encoder_input)
encoder = Model(encoder_input, code, name='encoder')

```

Model: "encoder"

Layer (type)	Output Shape	Param #
encoder_input (InputLayer)	[(None, 784)]	0
latent_vector (Dense)	(None, 32)	25120
Total params: 25,120		

Modelos Encoder y Decoder

```
encoding_dim = 32  #factor de compresion 24.5, si entrada de 28x28=784
##----- ENCODER -----
encoder_input = Input(shape=(input_dim,), name='encoder_input')
code = Dense(encoding_dim, activation='relu',
              name='latent_vector')(encoder_input)
encoder = Model(encoder_input, code, name='encoder')
##----- DECODER -----
latent_input = Input(shape=(encoding_dim,), name='decoder_input')
decoded_image = Dense(input_dim, activation="sigmoid",
                      name='decoder_output')(latent_input)
decoder = Model(latent_input, decoded_image, name='decoder')
```

Modelos Encoder y Decoder

Model: "decoder"

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	[(None, 32)]	0
decoder_output (Dense)	(None, 784)	25872
Total params: 25,872		

##----- DECODER -----

```
latent_input = Input(shape=(encoding_dim,), name='decoder_input')
decoded_image = Dense(input_dim, activation="sigmoid",
                      name='decoder_output')(latent_input)
decoder = Model(latent_input, decoded_image, name='decoder')
```

Modelo Autoencoder

```
autoencoder = Model(encoder_input, decoder(encoder_input)),  
                    name='autoencoder')  
autoencoder.compile(loss='binary_crossentropy', optimizer='adam')
```

```
Model: "autoencoder"
```

Layer (type)	Output Shape	Param #
encoder_input (InputLayer)	[(None, 784)]	0
encoder (Functional)	(None, 32)	25120
decoder (Functional)	(None, 784)	25872
Total params: 50,992		

Entrenamiento y uso

```
batch_size = 128
autoencoder.fit(x_train, x_train, validation_data=(x_test, x_test),
                epochs=50, batch_size=batch_size)
```

```
# Predicción del Autoencoder
```

```
x_decoded = autoencoder.predict(x_test)
```

```
# Codifica y decodifica algunos dígitos
```

```
encoded_imgs = encoder.predict(x_test)
```

```
decoded_imgs = decoder.predict(encoded_imgs)
```

Después de 50 épocas, el error en entrenamiento y testeo es de aprox. 0.09



Deep Autoencoder

```
input_img = layers.Input(shape=(784,))  
encoded = layers.Dense(128, activation='relu')(input_img)  
encoded = layers.Dense(64, activation='relu')(encoded)  
encoded = layers.Dense(32, activation='relu')(encoded)
```

```
decoded = layers.Dense(64, activation='relu')(encoded)  
decoded = layers.Dense(128, activation='relu')(decoded)  
decoded = layers.Dense(784, activation='sigmoid')(decoded)
```

```
autoencoder = Model(input_img, decoded)
```



Total params: 222,384

Después de 100 épocas, el error en entrenamiento y testeo es de aprox. 0.08

Autoencoder convolucional

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 28, 28, 1)]	0
conv2d_17 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_18 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_8 (MaxPooling2D)	(None, 7, 7, 8)	0
conv2d_19 (Conv2D)	(None, 7, 7, 8)	584
max_pooling2d_9 (MaxPooling2D)	(None, 4, 4, 8)	0
conv2d_20 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_7 (UpSampling2D)	(None, 8, 8, 8)	0
conv2d_21 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_8 (UpSampling2D)	(None, 16, 16, 8)	0
conv2d_22 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_9 (UpSampling2D)	(None, 28, 28, 16)	0
conv2d_23 (Conv2D)	(None, 28, 28, 1)	145
Total params: 4,385		

Autoencoder convolucional

- Resultado del entrenamiento con 10 épocas



Imágenes con ruido

```
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
                                                         size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,
                                                         size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```



Modelo convolucional usado

```
input_img = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# En este punto la representación es de (7, 7, 32)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

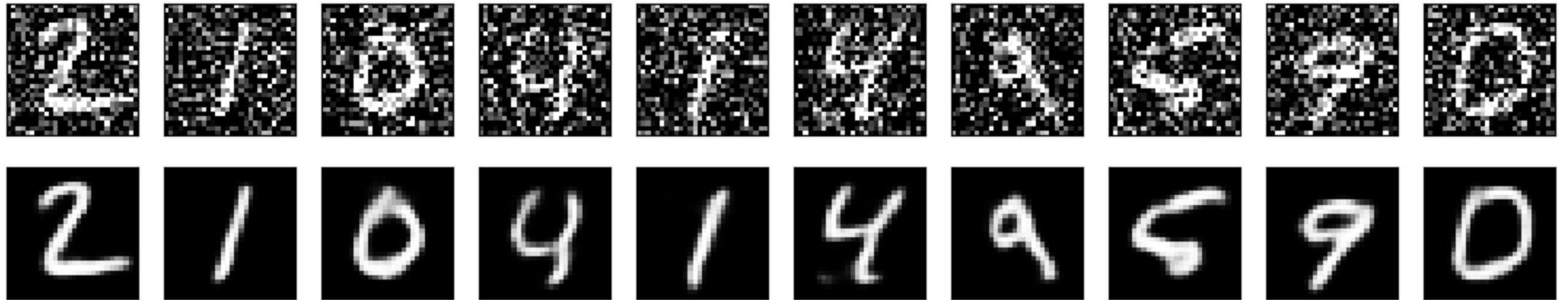
autoencoder = Model(input_img, decoded)
print(autoencoder.summary())
```

Resultado de la decodificación

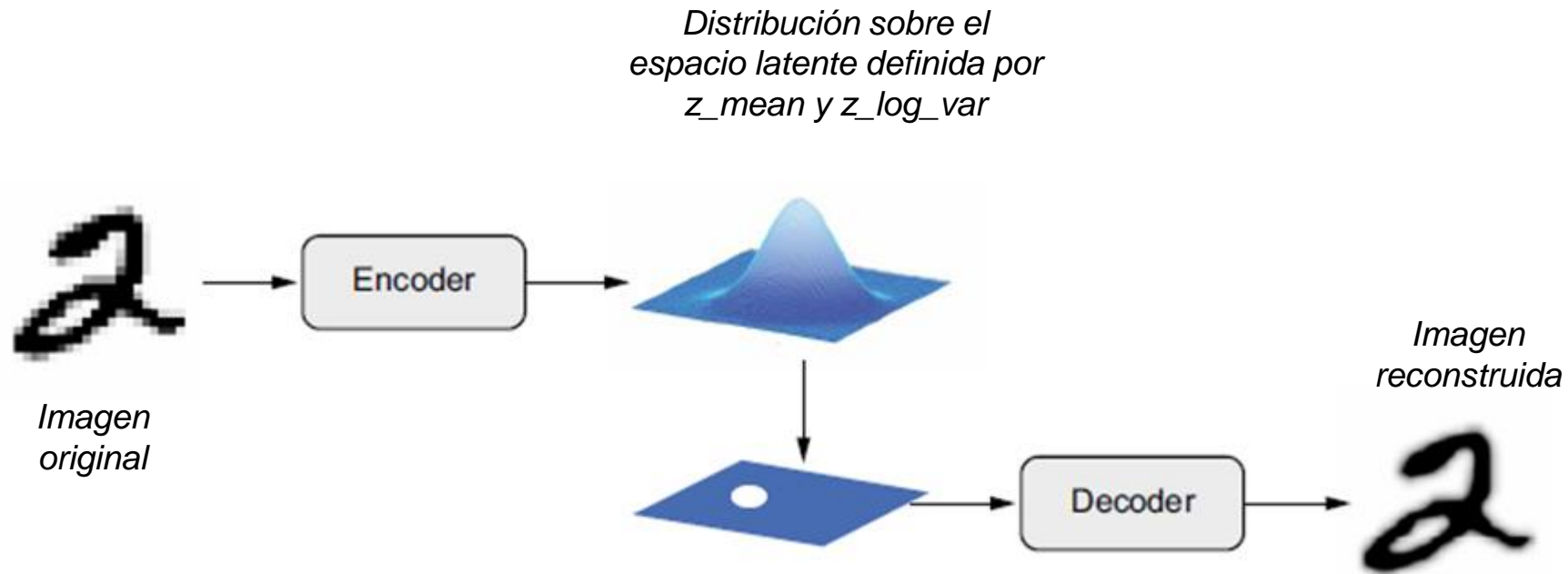
```
autoencoder.fit(x_train_noisy, x_train, epochs=100, batch_size=128,  
               shuffle=True, validation_data=(x_test_noisy, x_test))
```

Predicción del Autoencoder para las imágenes de testeo

```
decoded_imgs = autoencoder.predict(x_test_noisy)
```



Variational autoencoders (VAE)



Variational-autoencoders.ipynb

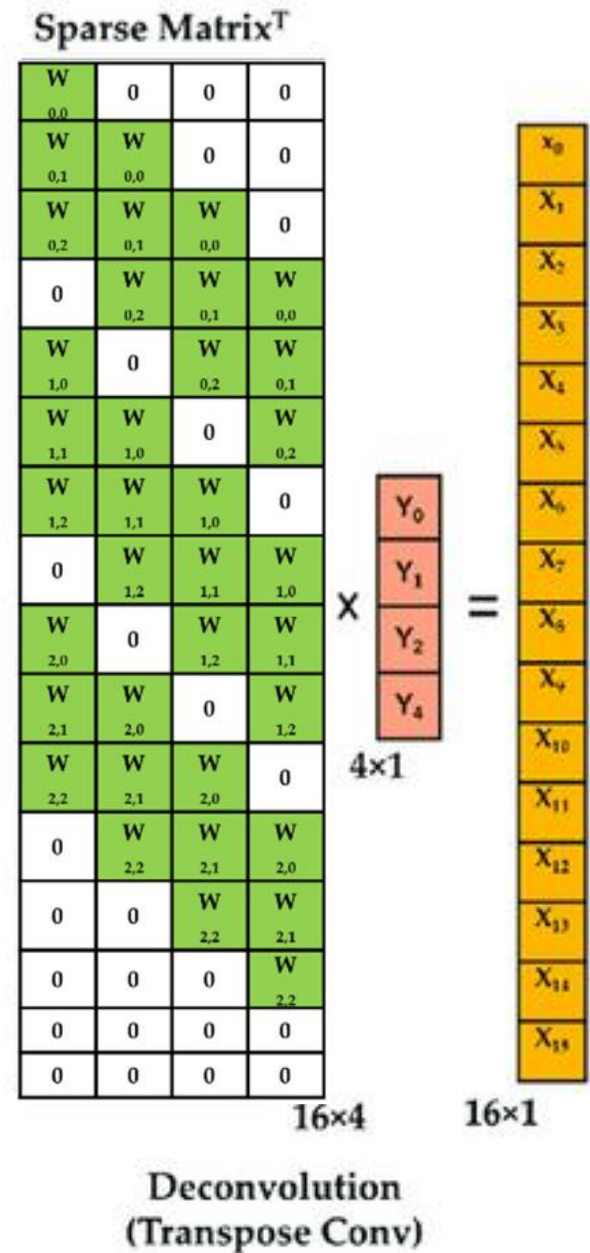
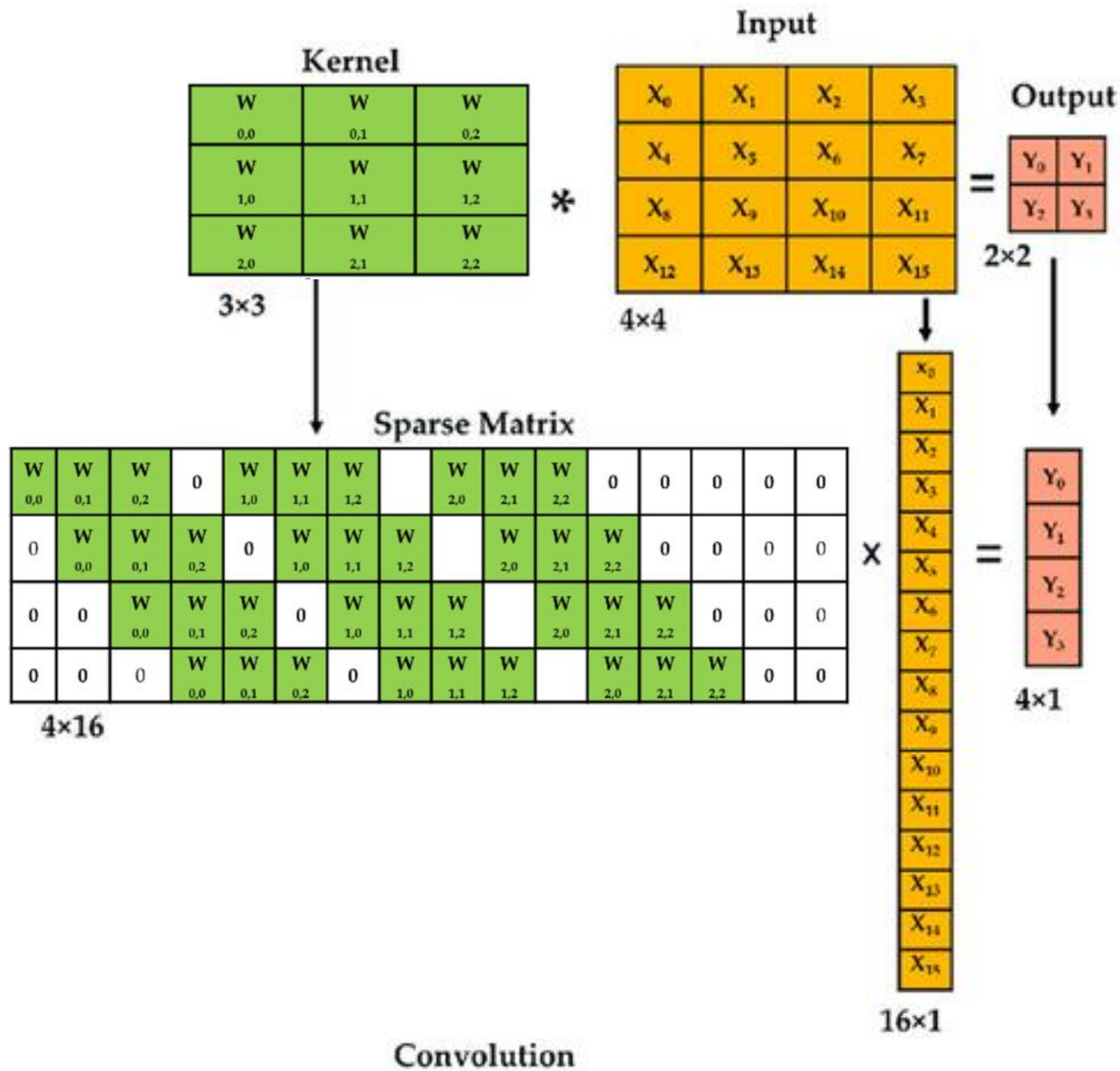
Variational autoencoders (VAE)



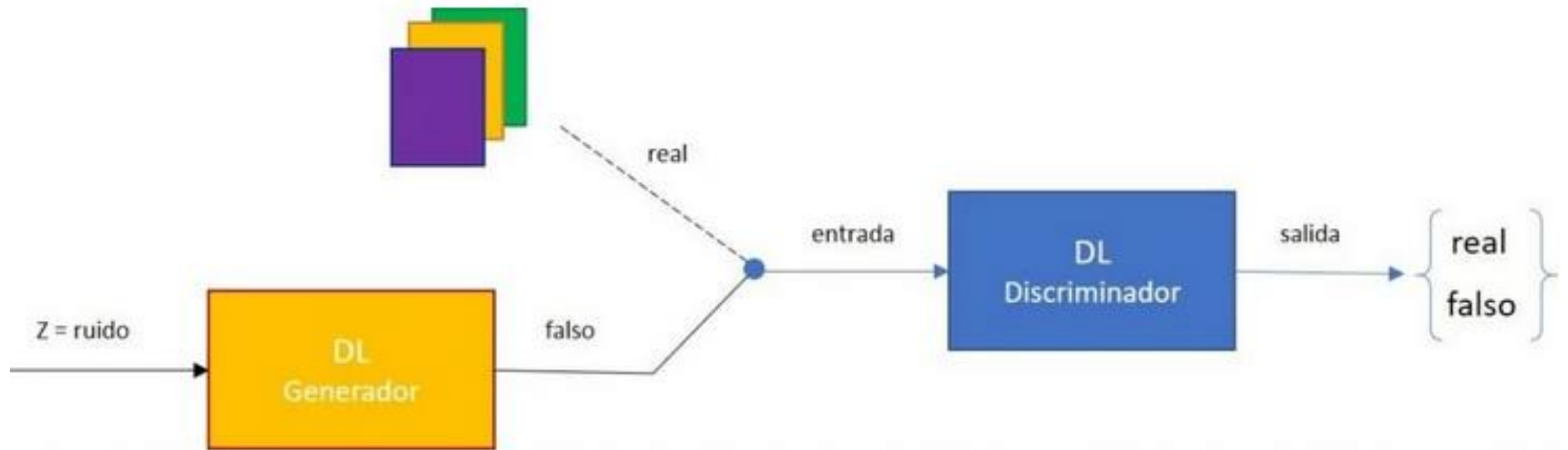
Variational autoencoders (VAE)



Un espacio continuo de caras generadas por Tom White usando VAEs
(Figura 12.14 del libro de Chollet)

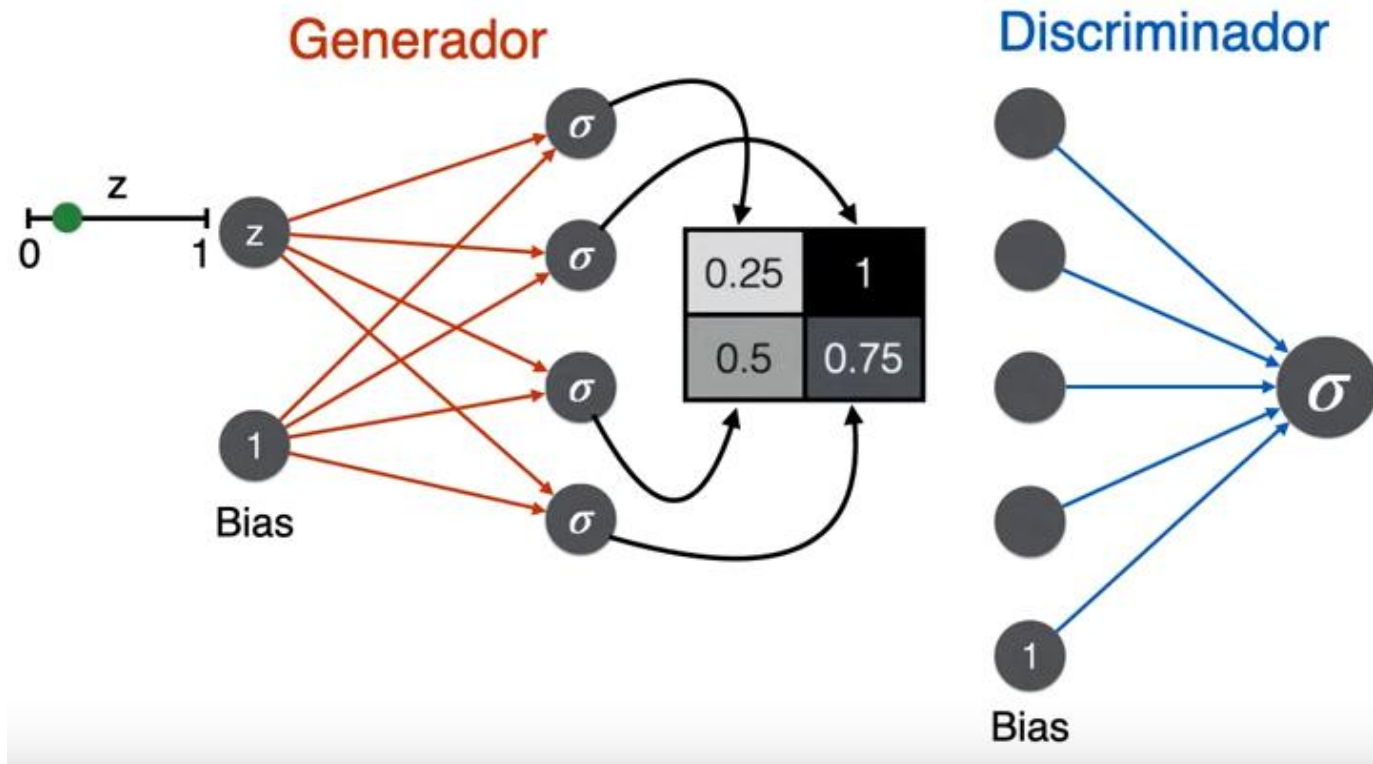


Redes generativas adversarias (GAN)

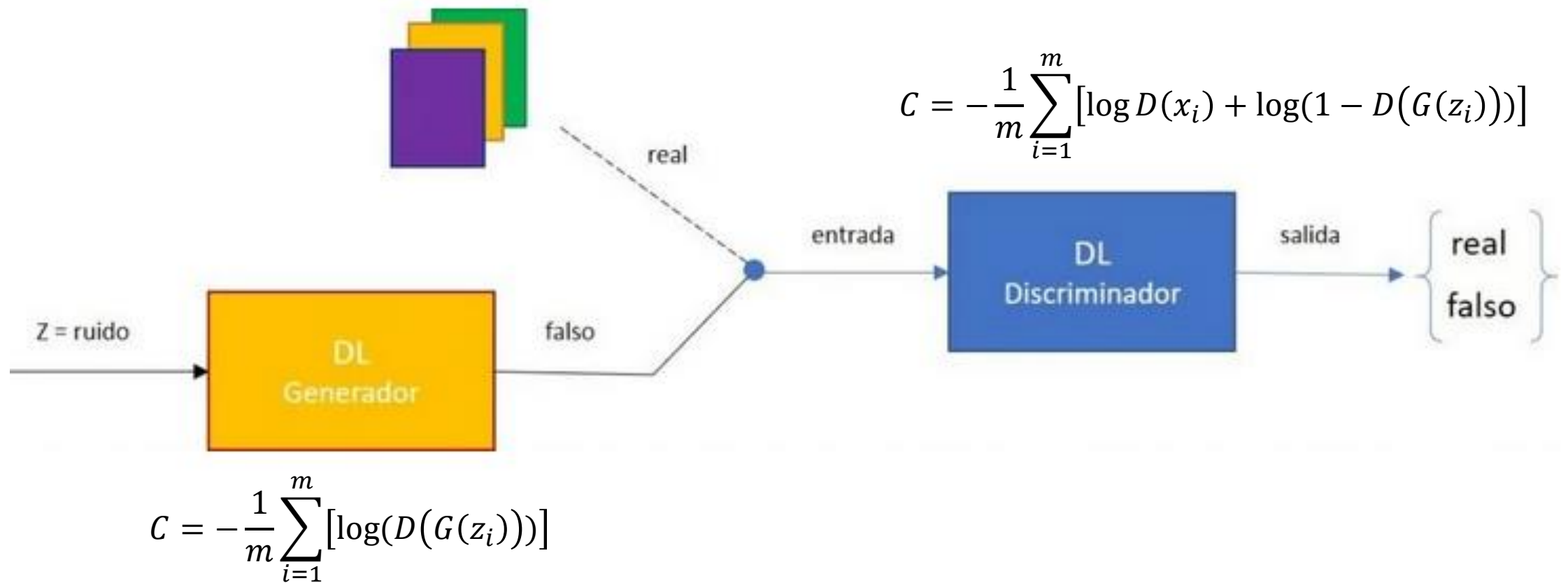


Redes generativas adversarias (GAN)

□ Video de Luis Serrano



Redes generativas adversarias (GAN)



Discriminador

□ Función de costo

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

Predicciones de los
ejemplos reales
(debería ser 1)

Predicciones de los
ejemplos generados
(debería ser 0)

Generador

□ Función de costo

$$C = -\frac{1}{m} \sum_{i=1}^m \left[\log \left(\underbrace{D(G(z_i))}_{\text{Predicciones de los ejemplos generados (debería ser 1)}} \right) \right]$$

Predicciones de los ejemplos generados
(debería ser 1)

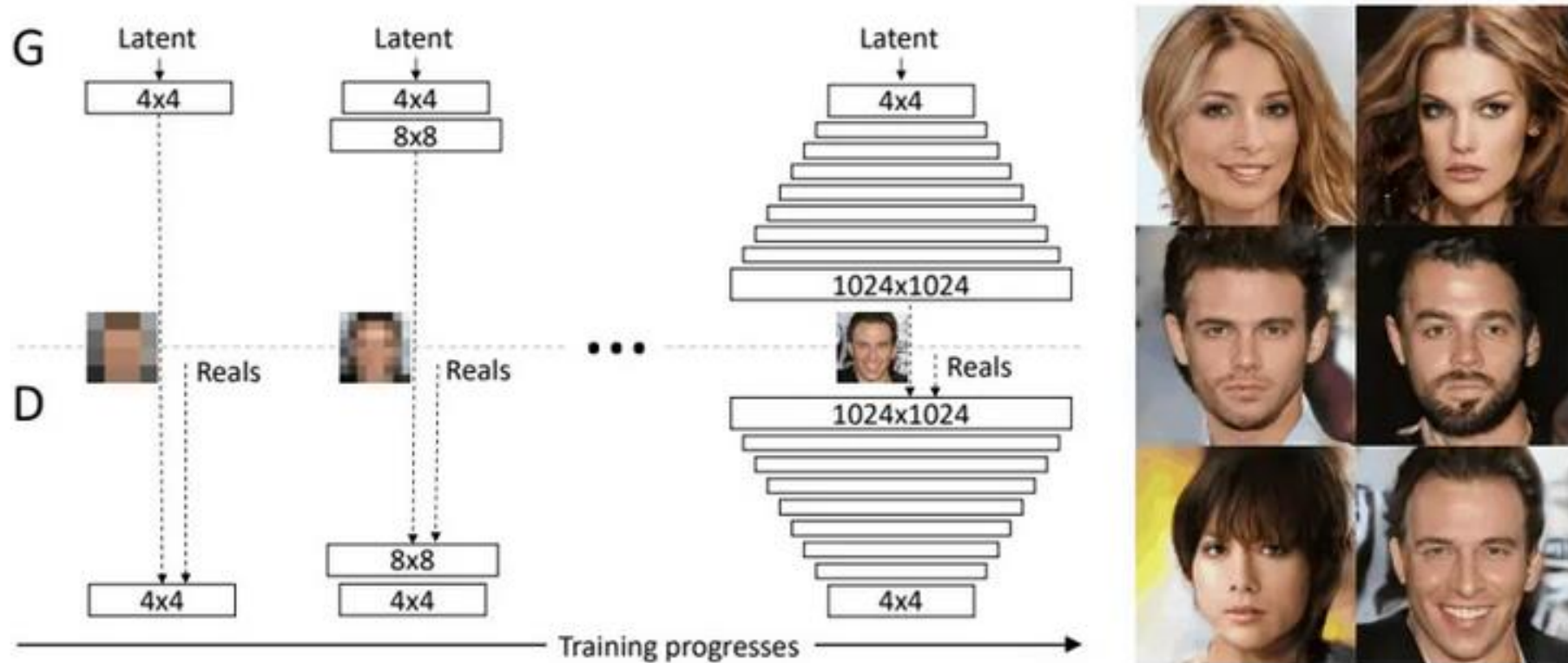
Ejemplo: Generando dormitorios



*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. Aquí tienes un ejemplo de 3 caras que he obtenido. Te animo a que lo pruebes tú también.

