Aprendizaje Automático Profundo (Deep Learning)

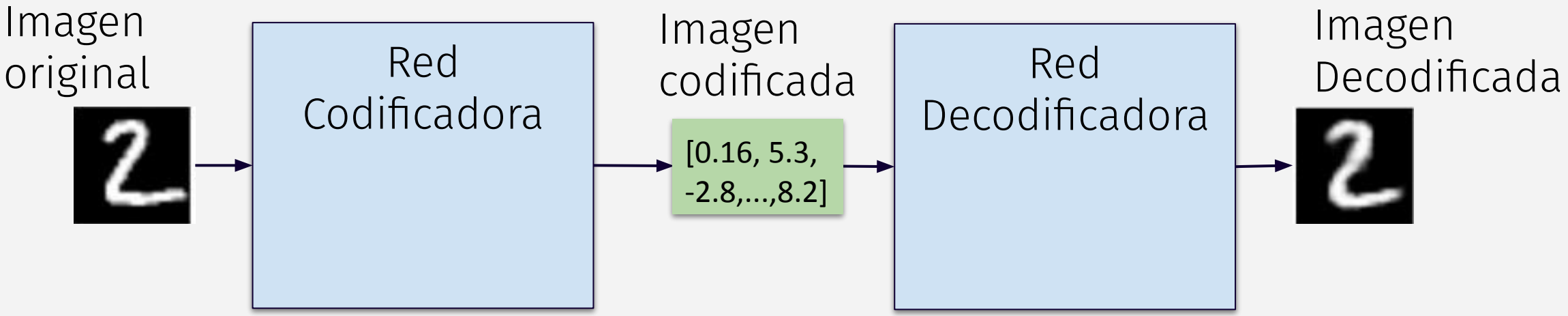
Dr. Facundo Quiroga - Dr. Franco Ronchetti

Autoencoders

Auto codificadores

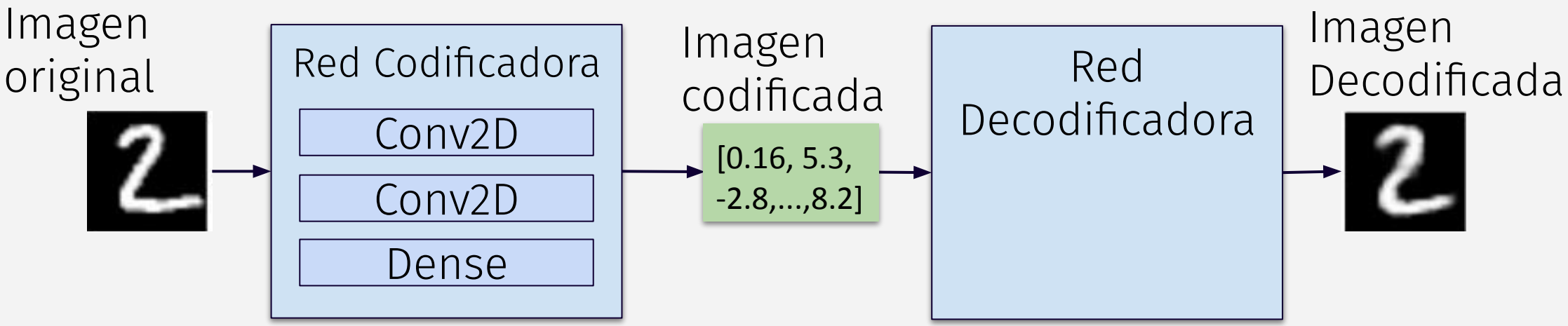
Autoencoders (Auto codificadores)

- Entrenamiento de AutoEncoder
  - Input: Imagen
  - Output: *Misma* imagen!
  - Modelo NO supervisado
    - No utiliza etiquetas
    - Aunque utiliza *mecanismos* supervisados
  - Tres modelos: encoder, decoder, autoencoder = decoder(encoder(x))



Red Codificadora

- Genera un vector con K elementos
  - K es arbitrario
- K << dimensión de la imagen original
- Vector comprime la imagen
  - Autoencoders para compresión
- Es una red común (CNN o Dense)



Red decodificadora

- Genera una imagen
  - Mismo tamaño que imagen de entrada
- *Descomprime* la imagen
  - Entrada: vector de tamaño K
  - Salida: Imagen
- Es una red común (CNN o Dense)
  - Generalmente, diseño espejo a la codificadora

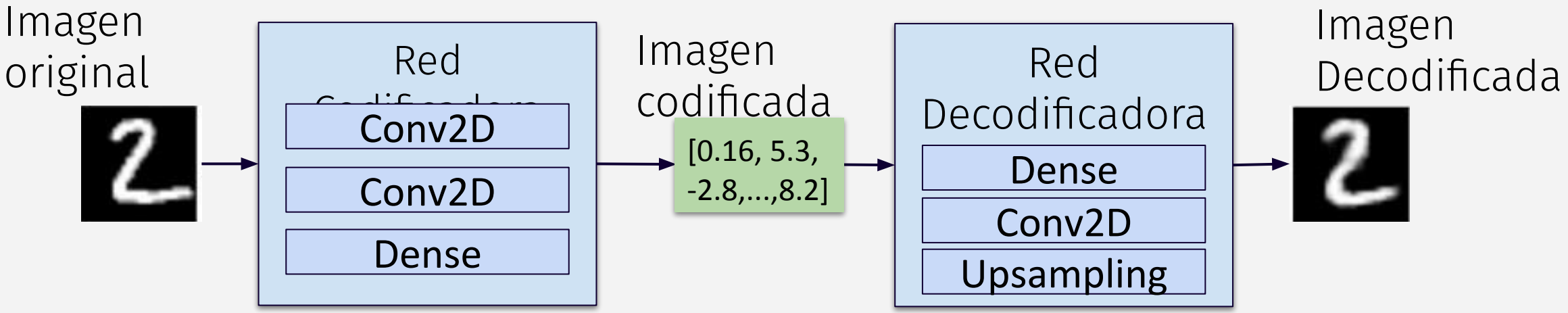
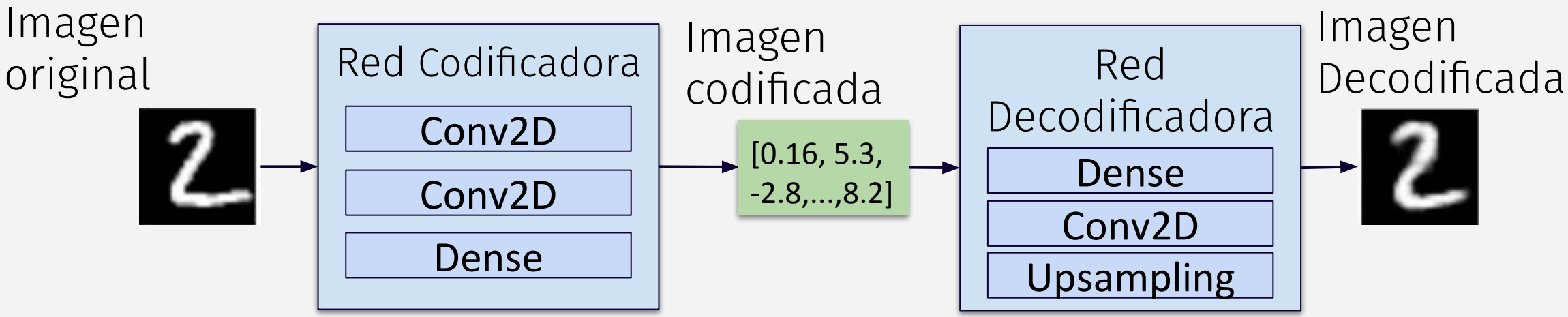


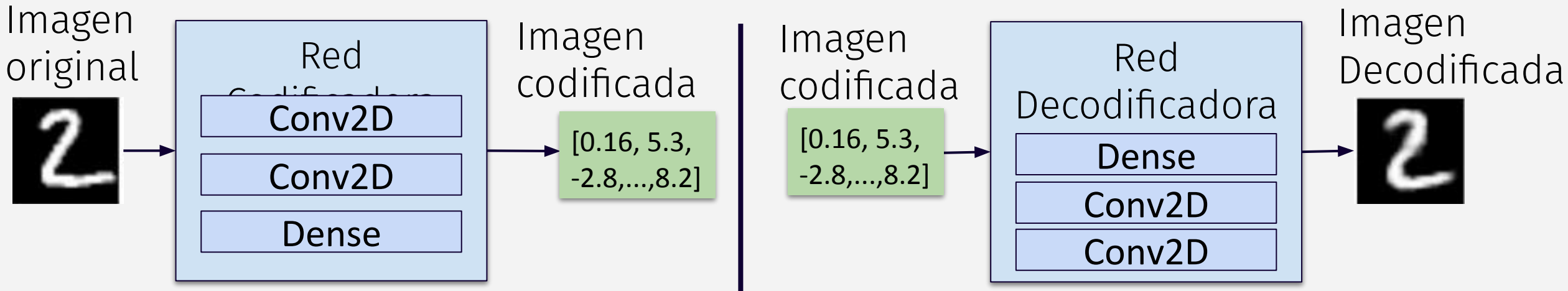
Imagen Codificada

- Se llama vector latente (histórico)
- Vector de tamaño K
- K arbitrario:
  - más grande
    - mayor poder de representación
    - menor compresión



Etapas

- Entrenamiento
  - model.fit(x_train,x_train)
  - Error: mse o entropía cruzada
    - Entre *imágenes*!
- Compresión con red codificadora
  - codigo = encoder.predict(imagen)
- Descompresión con red decodificadora
  - imagen_restaurada = decoder.predict(codigo)



Auto Codificadores para Comprimir (notebook)

- Utiliza el vector latente como compresión de la imagen
- Código solo sirve para el tipo de datos de entrenamiento
  - No es un compresor universal como JPEG o PNG
- Primera versión con modelos tradicionales (capas Dense)

```
def load_data():
  (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
  return x_train,x_test,input_dim
```

Auto Codificadores para Comprimir (notebook)

```
def DenseAutoencoder(input_dim,latent_dim):
  def generate_encoder():
    encoder_input = Input(shape=(input_dim,), name='encoder_input')
    code = Dense(latent_dim, name='latent_vector')(encoder_input)
    encoder = Model(encoder_input, code, name='encoder')
    return encoder,encoder_input

  def generate_decoder():
    latent_input = Input(shape=(latent_dim,), name='decoder_input')
    decoded_image = Dense(input_dim,activation="sigmoid",name='decoder_output')(latent_input)
    decoder = Model(latent_input, decoded_image, name='decoder')
    return decoder

  encoder,encoder_input = generate_encoder()
  decoder = generate_decoder()
  autoencoder = Model(encoder_input, decoder(encoder(encoder_input)), name='autoencoder')
  return autoencoder,encoder,decoder
```

Auto Codificadores para Comprimir (notebook)

```
x_train,x_test,input_dim = load_data()
latent_dim = 64
autoencoder,encoder,decoder=DenseAutoencoder(input_dim,latent_dim)
autoencoder.compile(loss='binary_crossentropy', optimizer='adam')

autoencoder.fit(x_train,x_train,
                validation_data=(x_test, x_test),
                epochs=10, batch_size=128)

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

AC **Convolucionales** para Comprimir (notebook)

- Codificador usa capas Conv2D en lugar de Dense
  - Vector latente tiene tamaño (K, L, M)

```
encoder_input = Input(shape=input_shape, name='encoder_input')

    x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoder_input)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    encoded_shape = K.int_shape(x)[1:]

    encoder = Model(encoder_input, x, name='encoder')
```

AC **Convolucionales** para Comprimir (notebook)

- Decodificador usa capas *Upsample*
  - Capa final con 1 filtro genera imagen de 1 canal

```
latent_input = Input(shape=encoded_shape, name='decoder_input')


    x = Conv2D(8, (3, 3), activation='relu', padding='same')(latent_input)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(16, (3, 3), activation='relu')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)


    decoder = Model(latent_input, x, name='decoder')
```

Denoising Autoencoder (notebook)

- Autoencoder para quitar ruido
  - x_noise = x + np.random.noise(...)
  - model.fit(x_noise,x)
  - Aprende a convertir imágenes con ruido en imágenes sin ruido

Originales

Con ruido

Restauradas

Denoising Autoencoder (notebook)

- Carga de datos y generación de muestras con ruido
- Se modifica a las originales con ruido gaussiano
  - valores de una normal con media 0.5 y desviación 0.1

```
def load_data(noise_location,noise_strength):
  (x_train, _), (x_test, _) = mnist.load_data()

  image_size = x_train.shape[1]
  x_train = np.reshape(x_train, [-1, 28,28, 1])/255.0
  x_test = np.reshape(x_test, [-1, 28,28, 1])/255.0
  input_shape = (28,28, 1)

  # Genera muestras de MNIST corrompidas por el ruido
  # centrado en 0.5 y con fuerza 0.1
  noise = np.random.normal(loc=0.5, scale=0.1, size=x_train.shape)
  x_train_noisy = x_train + noise
  noise = np.random.normal(loc=0.5, scale=0.1, size=x_test.shape)
  x_test_noisy = x_test + noise

  x_train_noisy = np.clip(x_train_noisy, 0., 1.) # restrinjo  al rango 0-1
  x_test_noisy = np.clip(x_test_noisy, 0., 1.) #  restrinjo  al rango 0-1
  return x_train,x_train_noisy,x_test,x_test_noisy,input_shape
```

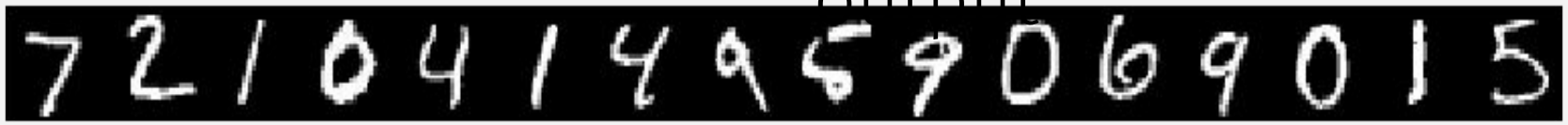
Denoising Autoencoder (notebook)

```
autoencoder.fit(x_train_noisy,
                x_train,
                validation_data=(x_test_noisy,x_test),
                epochs=30,
                batch_size=batch_size)
```

- Entrenamiento del modelo con valores ruidosos como input y originals como output

- Evaluar el modelo con una imagen ruidosa para quitar ruido
- `x_restaurado = autoencoder.predict(x_ruidoso)`

Originales



Con ruido

Restauradas