Practica 3:
Redes Convolucionales

Ana Paula Morresi

Aprendizaje profundo con aplicación a visión artificial

7 de noviembre de 2025

1. Problemas

Ejercicio 1.

Usando la base de datos del MNIST, se entrenó y evaluó el rendimiento de una red convolucional. Para armar la red se uso *Keras* con *tensorflow* como backend.

En la Fig. 1 se pueden observar algunas imágenes de la base de datos MNIST con su clasificación (etiqueta) correspondiente. Esta base de datos cuenta con dígitos del 0 al 9 con dimensiones de 28 x 28 pixeles y en escala de grises (1 canal). De esta forma, tenemos 10 clases y dimensión de entrada (28, 28, 1). El rango de valores de las imágenes va de 0 a 255, por lo que se la reescaleo a valores entre 0 y 1.

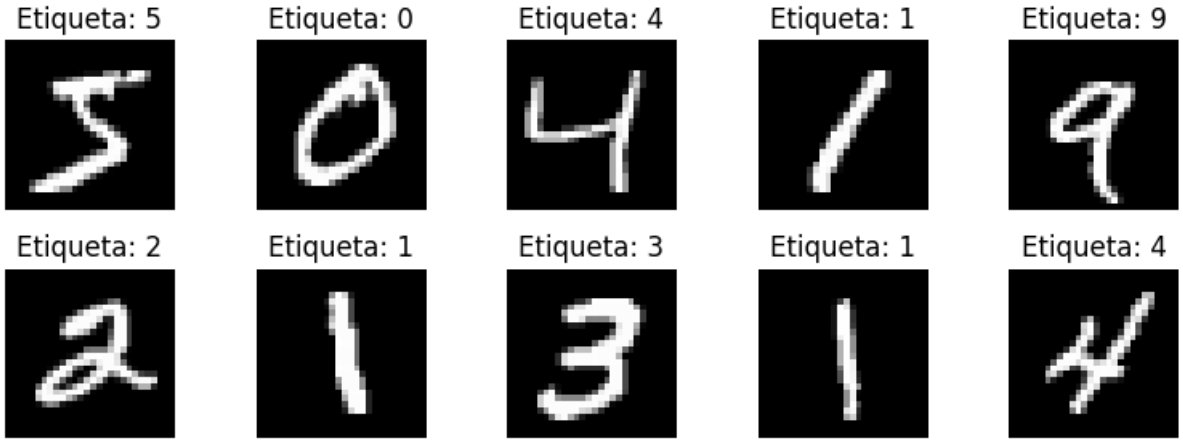

Figura 1: Caption

Usando *keras.Sequential* se creo el modelo intercalando capas Conv2D y MaxPooling. Al final se termino con una capa GlobalAveragePooling2D seguida de una capa densa. En las capas convolucionales se utilizó la función de activación *ReLU* para introducir no linealidad, mientras que en la capa de salida se empleó *Softmax* para obtener probabilidades asociadas a cada una de las 10 clases posibles.

En la Tab. 1 se puede ver el resumen del modelo implementado.

Layer (type)	**Output Shape**	**Param #**
conv2d_12 (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d_6 (MaxPooling2D)	(None, 13, 13, 8)	0
conv2d_13 (Conv2D)	(None, 11, 11, 16)	1,168
global_average_pooling2d_6 (GlobalAveragePooling2D)	(None, 16)	0
dense_6 (Dense)	(None, 10)	170

Cuadro 1: Resumen de la arquitectura del modelo convolucional.

Total params: 1,418 (5.54 KB)
Trainable params: 1,418 (5.54 KB)
Non-trainable params: 0 (0.00 B)

Usando la perdida *CategoricalCrossEntropy*, el optimizador *Adam* y la métrica *accuracy*, se entreno el modelo. Se uso un *batch size=200*, *learning rate=5e-3* y un *early stopping* con paciencia de 3 épocas en la loss de validación para evitar sobreentrenamiento. Se obtuvo de valores de accuracy finales:

Accuracy en train: 0.9374 || Accuracy en validation: 0.9453 || Accuracy en test: 0.9436

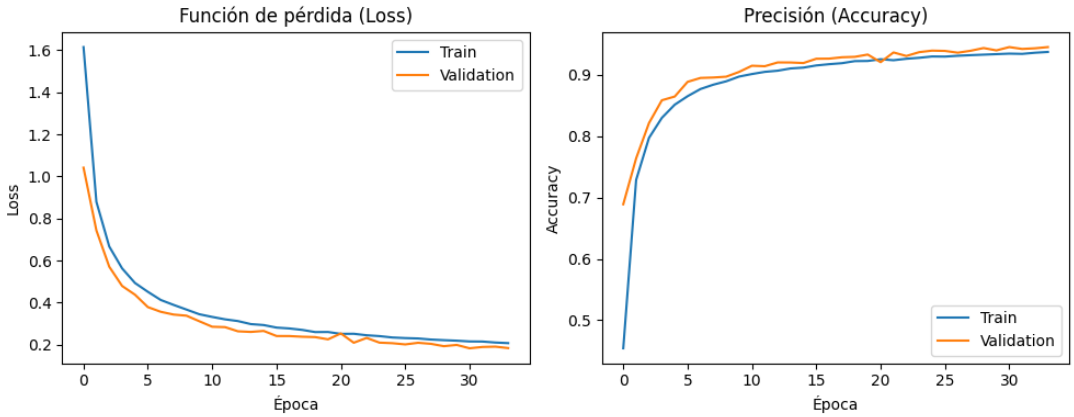

Figura 2: Curvas de loss y accuracy.

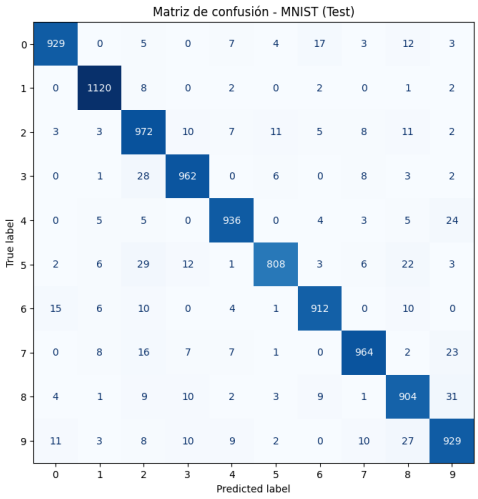

Figura 3: Matriz de confusión de los datos de test.

Se puede ver que el accuracy obtenido en el set de test supera el de train. La arquitectura utilizada es suficiente para capturar las características de los dígitos.

Ejercicio 2.

Con la misma base de datos del MNIST se entrenó y evaluó la performance de una red convolucional realizada con *pytorch*.

Para este caso se creo el modelo combinando capas *Conv2D* y *Linear*. En el forward se combino funciones *relu*, *max_pool2d*, *adaptive_avg_pool2d* y *xview*, esta ultima para poner la matriz en un vector. En la salida final se aplico *log_softmax*.

Esto, comparado con la sintaxis de *keras*, lleva mas lineas de código para su implementación. Ademas, es necesario especificar las dimensiones de entrada a cada capa, lo cual *keras* hace automáticamente.

En la Tab. 2 se puede ver el resumen del modelo implementado. En el forward se uso *relu* en las dos salidas de las capas *Conv2d*. A la salida de la primera capa convolucional, también se aplicó, luego de la función de activación, una *max_pool2d* y, después de la segunda capa convolucional, una *adaptive_avg_pool2d*.

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 8, 26, 26]	80
Conv2d-2	[-1, 16, 11, 11]	1,168
Linear-3	[-1, 10]	170

Cuadro 2: Resumen de la arquitectura del modelo convolucional en PyTorch.

Total params: 1,418
Trainable params: 1,418
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.01
Estimated Total Size (MB): 0.06

Usando la perdida *NLLLoss* y el optimizador *Adam*, se entreno el modelo. Se uso un *batch size=200*, *learning rate=1e-3* y un *early stopping* con paciencia de 3 épocas en la loss de validación. Se obtuvo de valores de accuracy finales:

Accuracy en train: 0.9303 || Accuracy en validation: 0.9274 || Accuracy en test: 0.9387

En la Fig. 4 se pueden ver las curvas de perdida y accuracy, y en la Fig. 5 se puede observar la matriz de confusión sobre los datos de test. En este caso, se puede observar que la red logro una buena performance en el aprendizaje, de forma similar al problema anterior.

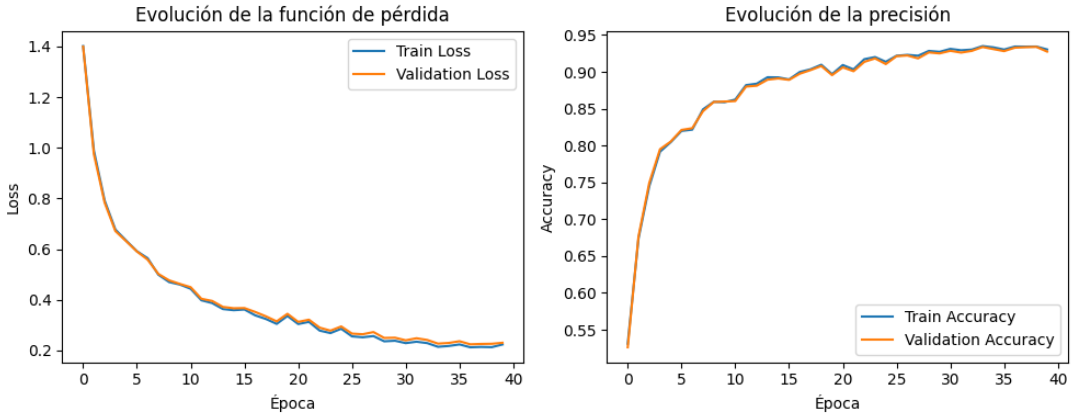

Figura 4: Curvas de loss y accuracy.

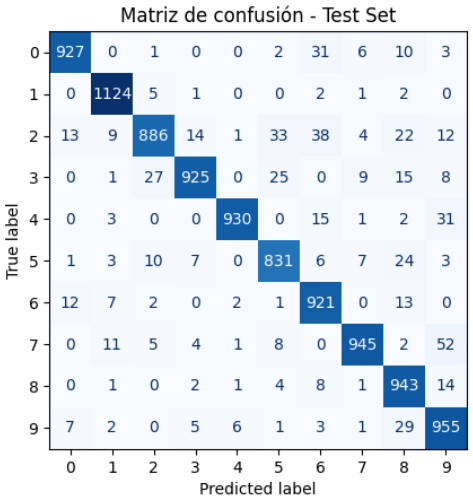

Figura 5: Matriz de confusión de los datos de test.

Ejercicio 3.

Se entrenó y evaluó la performance de una red con arquitectura U-net que realiza una tarea de segmentación semántica. Para esto se usaron imágenes de *oxford-it-pet*.

La base de datos cuenta con un total de 7390 imágenes y mascaras de diferentes especies de gatos y perros. Las mascaras tiene anotaciones por pixel de animal (1), fondo (2) y no clasificado/borde (3). Con esto se entreno una arquitectura U-net como la que se encuentra en la Fig. 10, donde se reescalearon las imágenes a tamaños de 128 x 128. Como optimizador se uso *Adam*, con función de loss *Sparse Categorical Crossentropy* y las métricas *accuracy* y *IoU*. En el entrenamiento se uso ademas un *early stopping* con una paciencia de 5 épocas en la loss de validación.

Se obtuvo de valores de accuracy finales:

Accuracy en train: 0.8949 || Accuracy en validation: 0.8692 || Accuracy en test: 0.8721

Y como valores finales de IoU finales:

IoU en train: 0.7216 || IoU en validation: 0.6810 || IoU en test: 0.6855

En la Fig. 6 se pueden ver las curvas de loss y las metricas accuracy y IoU en los dataset de train y validation. Se puede observar que en el valor de accuracy obtenido en test es bueno, el valor de IoU es mas bajo, aunque sigue siendo mayor a un 68 %.

La metrica IoU se utilizo ya que esta se define como:

$$IoU_i = \frac{Pred_i \cap True_i}{Pred_i \cup True_i}$$

para la clase i.

Esto es mas correcto para medir que tan buena fue la performance del modelo en la segmentación. El valor finalmente calculado como IoU es el promedio sobre las tres clases.

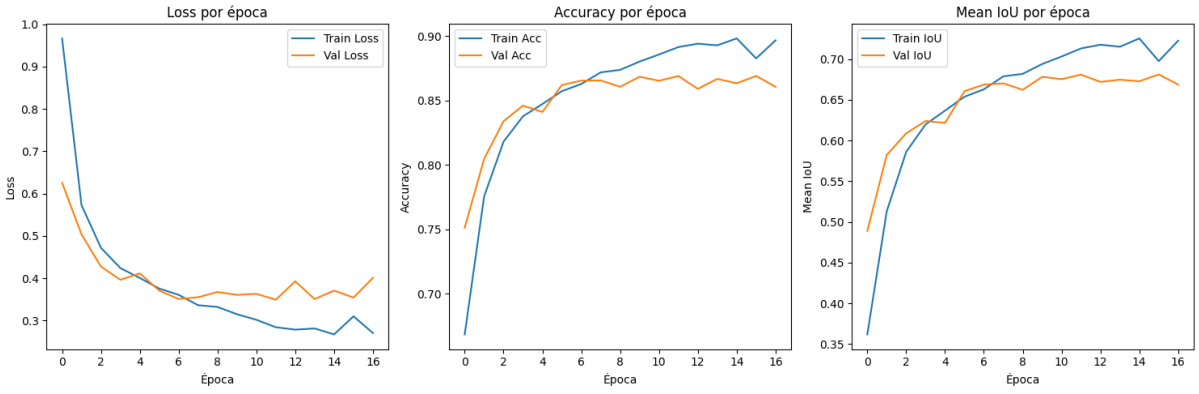

Figura 6: Curvas de loss, accuracy y IoU.

En la Fig. 7, 8, 9 se pueden ver ejemplos de las mascaras obtenidas con la red U-net implementada sobre el dataset de test. En los dos primeros ejemplos se puede observar que la segmentación obtenido coincide en gran medida con la predicha, teniendo buenos valores en las métricas accuracy y IoU. En el tercer ejemplo la segmentación obtenida no fue tan buena, se ve que la red confunde las sombras de la luz.

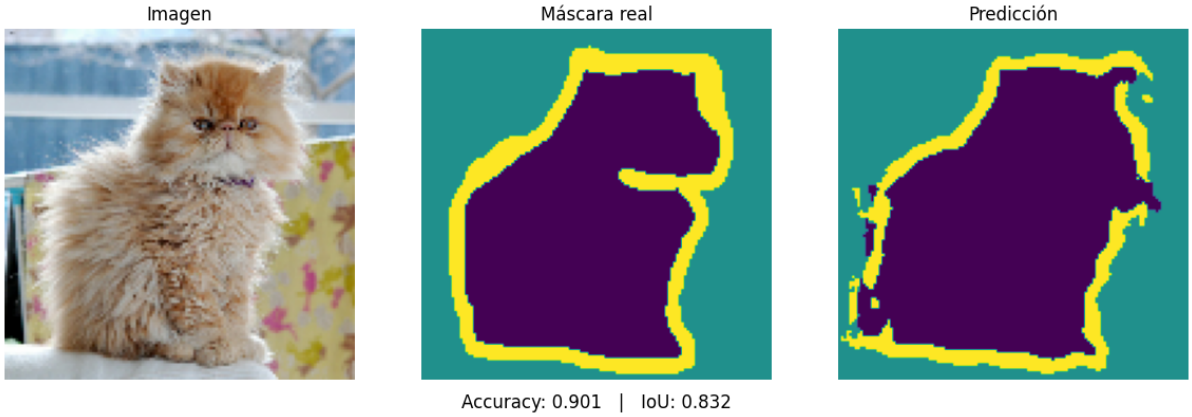

Figura 7: Ejemplo de segmentación 1.

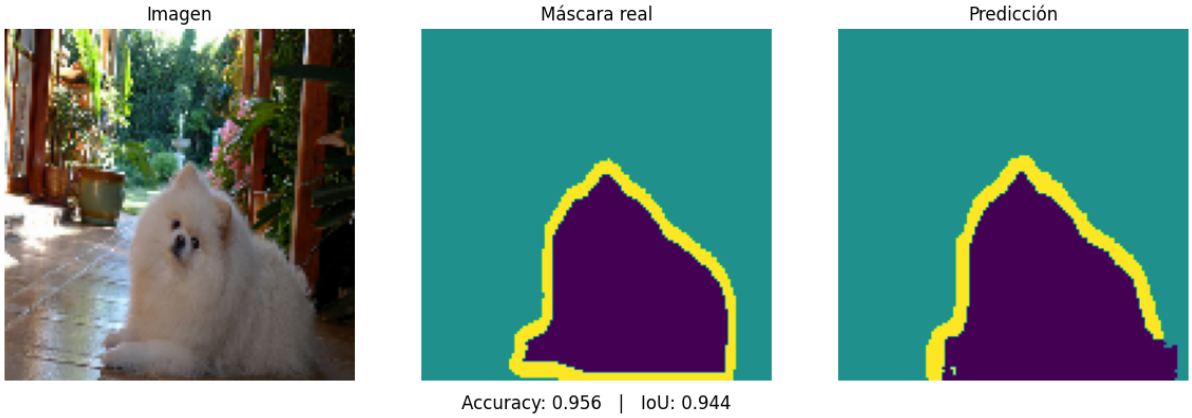

Figura 8: Ejemplo de segmentación 2.

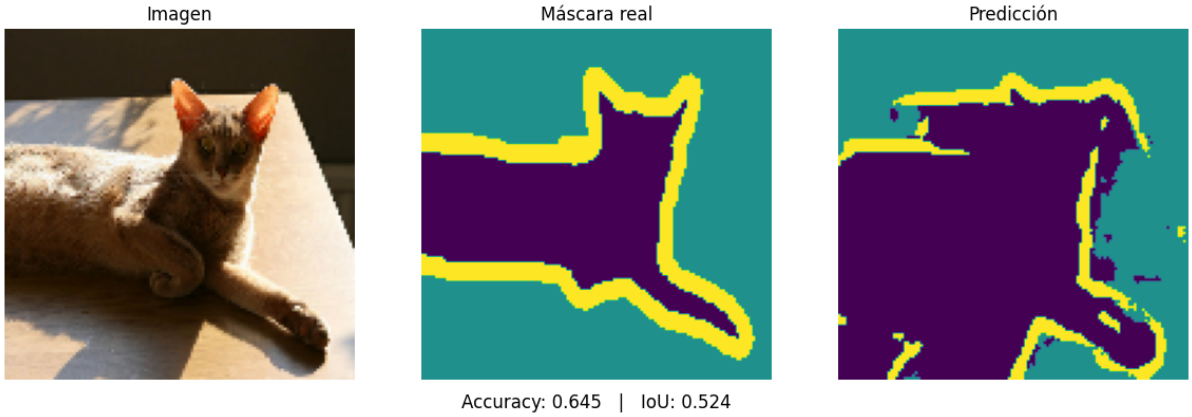

Figura 9: Ejemplo de segmentación 3.

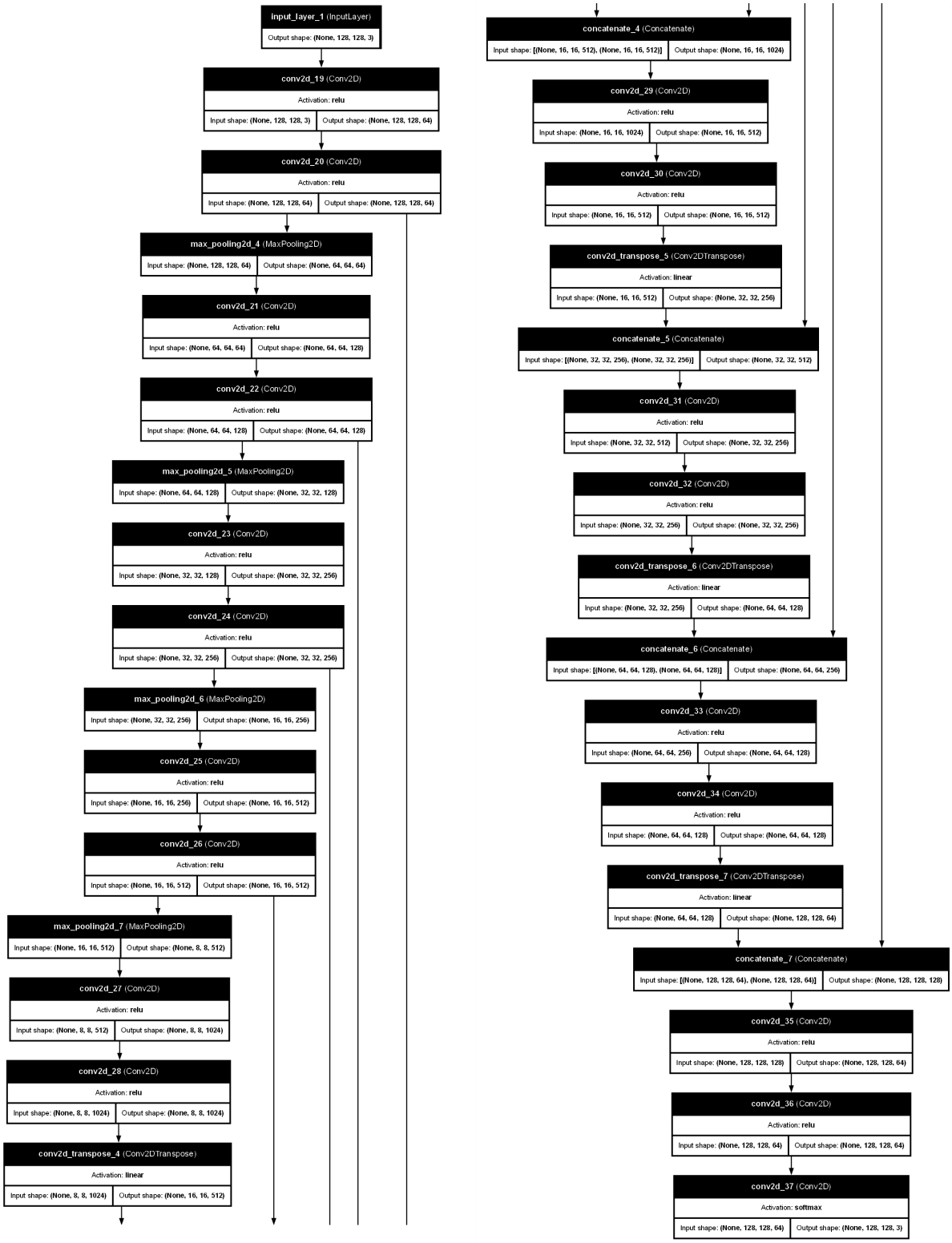

Figura 10: Arquitectura de la red U-net implementada. La imagen continua al costado.

2. Apéndice

Los códigos con los ejercicios resueltos se encuentran en el repositorio: `https://github.com/AnaMorresi/DeepLearning`.