Informe de Trabajo Práctico 3

Aprendizaje Profundo con aplicación a Visión Artificial

Laura Zúñiga Osorio

10 de noviembre de 2025

1. Keras: CNN para clasificación de MNIST

En este ejercicio entrenamos una red neuronal convolucional (CNN) con el fin de clasificar la base de datos MNIST, utilizando `tensorflow`. En primer lugar, cargamos los datos junto con sus etiquetas y aplicamos el siguiente preprocesamiento:

- Separamos entre conjuntos de entrenamiento y evaluación, con 60000 y 10000 pares de imagen-anotación, respectivamente.
- Convertir las imágenes a `float32` y las dividimos entre 255 para normalizarlas al rango $[0, 1]$.
- Agregamos una dimensión a los vectores de las imágenes para especificar el número de canales igual a 1.
- Convertir las clases a vectores del tipo `one_hot`.

En la figura 1 se muestran algunas imágenes de entrenamiento con sus respectivas etiquetas.

Figura 1: Ejemplos de entrenamiento de la base MNIST con sus respectivas etiquetas.

El modelo propuesto para la CNN se definió utilizando los módulos de `keras.Sequential()`. Podemos obtener el detalle de la arquitectura a través del comando `model.summary()`, como se muestra en la figura 2.

El entrenamiento se realizó por 10 épocas, seteando batches de 32 elementos, definiendo como criterio de pérdida a `CategoricalCrossentropy` y usando el optimizador Adam con tasa de aprendizaje 10^{-3}. La evolución de estas métricas se muestra a través de las curvas de Loss y Accuracy como función de las épocas en la figura 3.

Los scores sobre el conjunto de test se calcularon utilizando la función `model.evaluate()`, obteniendo una loss de 0,0805 y una accuracy de 0,9768.

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_19 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten_13 (Flatten)	(None, 6272)	0
dense_13 (Dense)	(None, 10)	62,730

Figura 2: Arquitectura de CNN (keras) utilizada para clasificar base de datos MNIST.

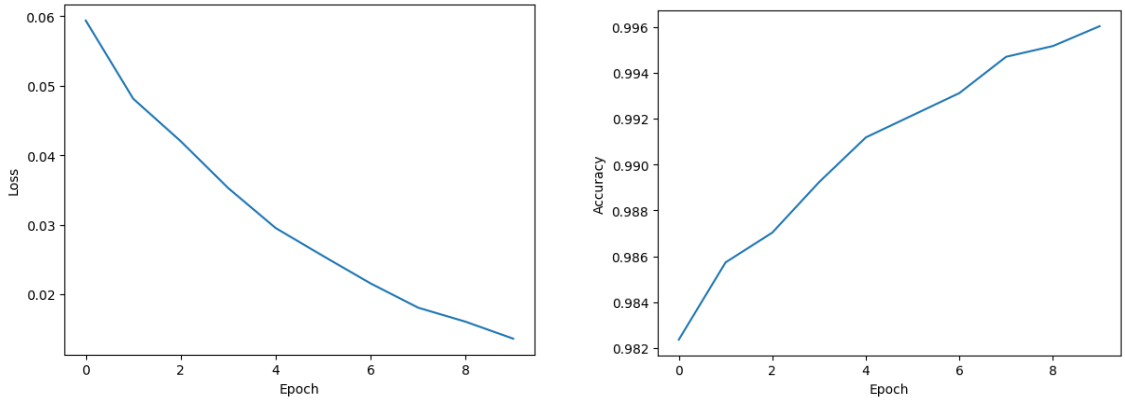

Figura 3: Evolución de métricas en el entrenamiento de CNN (keras) para clasificación de MNIST.

2. Pytorch: CNN para clasificación de MNIST

Este ejercicio tiene el mismo objetivo que el anterior, esta vez utilizando los módulos de `Pytorch`. Los datos se procesan automáticamente usando la librería `torchvision.datasets`, por lo que solo resta definir la arquitectura. Esta se define como una clase heredada de `nn.Module`, donde instanciamos las capas con sus dimensiones y definimos la interacción entre ellas en el `forward`. El siguiente cuadro incluye el código de construcción de la red "Net".

```
class Net(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = torch.nn.Conv2d(
        in_channels=1,
        out_channels=32,
        kernel_size=3,
        stride=1)
    self.conv2 = torch.nn.Conv2d(
        in_channels=32,
        out_channels=16,
        kernel_size=3,
        stride=1)
    self.linear = torch.nn.Linear(
        in_features=6*6*16,
        out_features=10)
  def forward(self, input):
```

```
    x = self.conv1(input)
    x = torch.nn.functional.relu(x)
    x = torch.nn.functional.max_pool2d(
        x,
        kernel_size=2,
        stride=2)
    x = self.conv2(x)
    x = torch.nn.functional.relu(x)
    x = torch.nn.functional.max_pool2d(
        x,
        kernel_size=2,
        stride=2)
    x = torch.flatten(x, start_dim=1)
    x = self.linear(x)
    x = torch.nn.functional.log_softmax(x, dim=0)
    return x
```

Entrenamos durante 50 épocas, seteando batches de 64 elementos, utilizando el criterio `CrossEntropyLoss` y el optimizador Adam con tasa de aprendizaje 10^{-3}. La evolución de la loss y la accuracy se muestran en la figura 4.

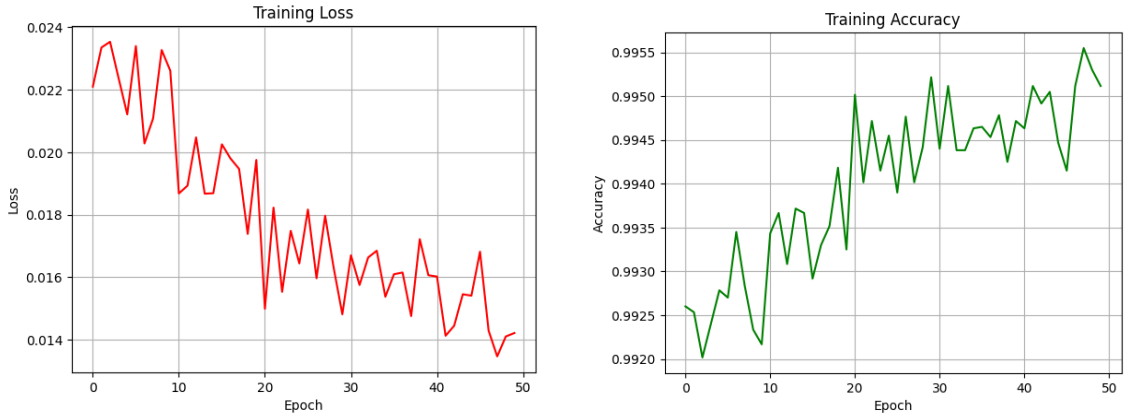

Figura 4: Evolución de métricas de aprendizaje de CNN (torch) utilizada para clasificación de MNIST.

La evaluación sobre el conjunto de test arroja una accuracy global de 0,9847, y podemos visualizar de forma más completa los resultados por clase mediante la matriz de confusión de la figura 5. Esta se obtuvo utilizando la librería `Sklearn`.

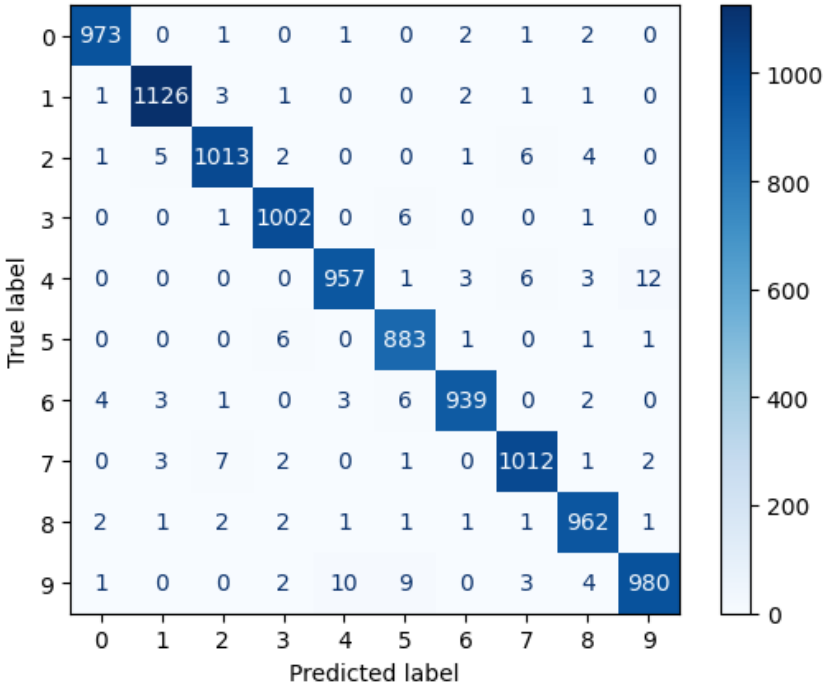

Figura 5: Matriz de confusión sobre el conjunto de test de MNIST para entrenamiento de CNN (torch).

3. Segmentación con U-Net

En esta ocasión implementamos una red del tipo U-Net para llevar a cabo una tarea de segmentación semántica sobre la base de datos Oxford Pets. Esta contiene imágenes de 37 clases (razas) y un total de 7349 imágenes, cada una acompañada de su máscara de segmentación a nivel de píxel. Solo fue posible cargar 500 datos de entrenamiento debido a las limitaciones de recursos computacionales.

Para el preprocesamiento se aplicaron los siguientes pasos:

- Redimensionar las imágenes y máscaras a 224×224 píxeles.
- Normalizar las imágenes al rango $[0, 1]$ y convertirlas a `float32`.
- Convertir las máscaras a formato binario por clase y aplicar `one_hot` cuando fue necesario.

Un ejemplo de las imágenes (consultado en internet) se muestra en la figura 6.

La arquitectura implementada corresponde a un U-Net clásico, esta se ilustra en el diagrama de la figura 7.

Como criterio de pérdida se utilizó `Sparse_Categorical_CrossEntropy` El optimizador utilizado fue Adam con tasa de aprendizaje 10^{-3}, batch size 32 y entrenamiento por 10 épocas.

La accuracy media alcanzada fue de 0,45 y la loss de 0,95.

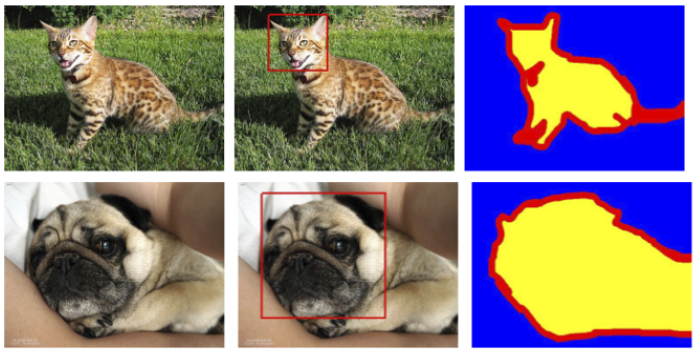

Figura 6: Ejemplo de entrenamiento de base de datos Oxford Pets.

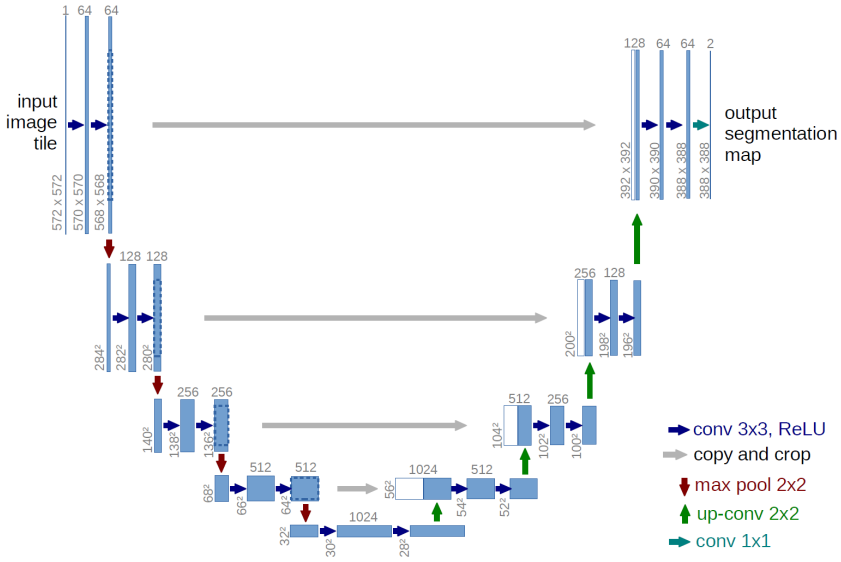

Figura 7: Diagrama de la arquitectura U-Net utilizada para segmentación semántica sobre la base de datos de Oxford Pets.