

Computer Vision aplicado a diagnóstico de tumores

[Configuración del ambiente 1](#_Toc1973306558)

[Figura 1 2](#_Toc1747147975)

[El dataset 2](#_Toc69278588)

[Figura 2 3](#_Toc324488730)

[Cargando las imágenes y sus clases 3](#_Toc1217818026)

[Figura 3 4](#_Toc715577844)

[Figura 4 5](#_Toc930744585)

[Figura 5 6](#_Toc1752567181)

[Data augmentation 7](#_Toc627078960)

[Figura 6 8](#_Toc992347567)

[Figura 7 10](#_Toc1382489453)

[Figura 8 12](#_Toc1202197781)

[El entrenamiento 12](#_Toc798753988)

[Figura 9 13](#_Toc1001083929)

[Figura 10 14](#_Toc1091392572)

[Figura 11 14](#_Toc1907309625)

[Figura 12 16](#_Toc1989595361)

[La historia del entrenamiento 16](#_Toc549959300)

[Figura 13 17](#_Toc41672087)

[La evaluación 17](#_Toc1036364027)

[Figura 14 18](#_Toc1367518623)

## Configuración del ambiente

He realizado un ambiente fácilmente reproducible y que me permite utilizar mi gpu con Docker, vscode y su extensión devcontainers. La configuración es el json que se muestra en la Figura 1.

### Figura 1

*Devcontainer con gpu*



*Contenido del archivo Devcontainer.json para la definición del ambiente de desarrollo en un container.*

## El dataset

Para efectos prácticos, he utilizado un jupyter notebook para desarrollar la actividad, ya que guarda los outputs para luego mostrarlos fácilmente. He automatizado la descarga del dataset con el script mostado en la Figura 2.

### Figura 2

*Downloading the dataset*



*Celda de jupyter notebook descargando el dataset.*

## Cargando las imágenes y sus clases

Primero he instalado e importado las librerías necesarias para el ejercicio.

A continuación, creé un objeto Dataset para cargar y mapear las imágenes con sus respectivas clases, ya que estas se encuentran separadas en carpetas como benignos o malignos, como se muestra en la Figura 3.

### Figura 3

*SkinCancerDataset*



*Definición de la clase del Dataset.*

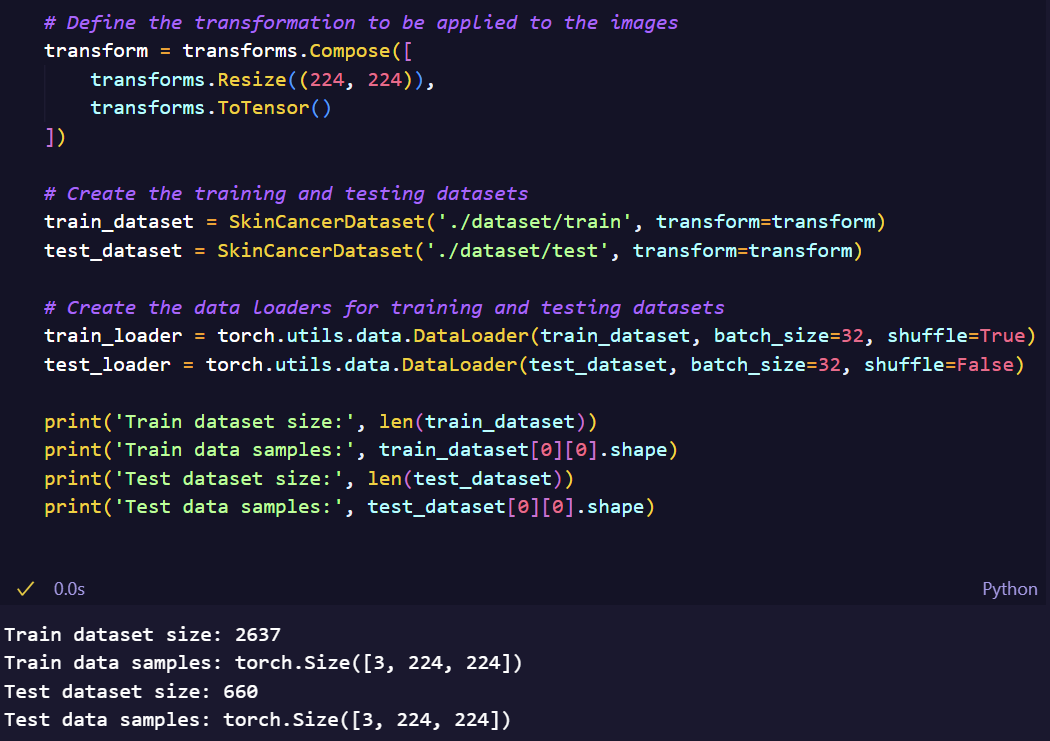
Definí la transformación al tensor con previa escalada de los pixeles.

Como el dataset viene previamente dividido en train y test, he utilizado esta metodología para dividirlos. Más adelante divido nuevamente el dataset de entrenamiento en train y validation, para efectos de entramiento.

También he defino el DataLoader.

### Figura 4

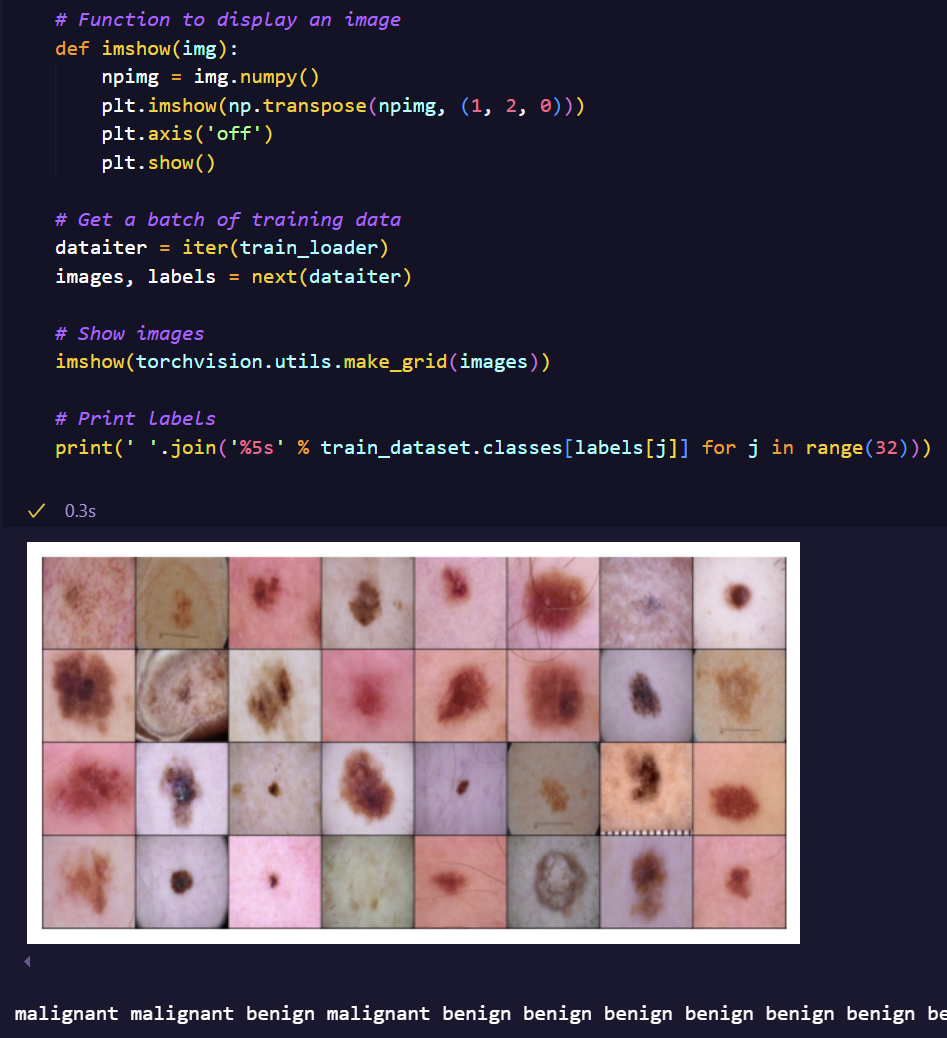
*Carga del dataset*

*Celda de jupyter notebook cargando el dataset en tensores.*

Visualizamos las imágenes y sus labels en la Figura 5.

### Figura 5

*Visualizando*



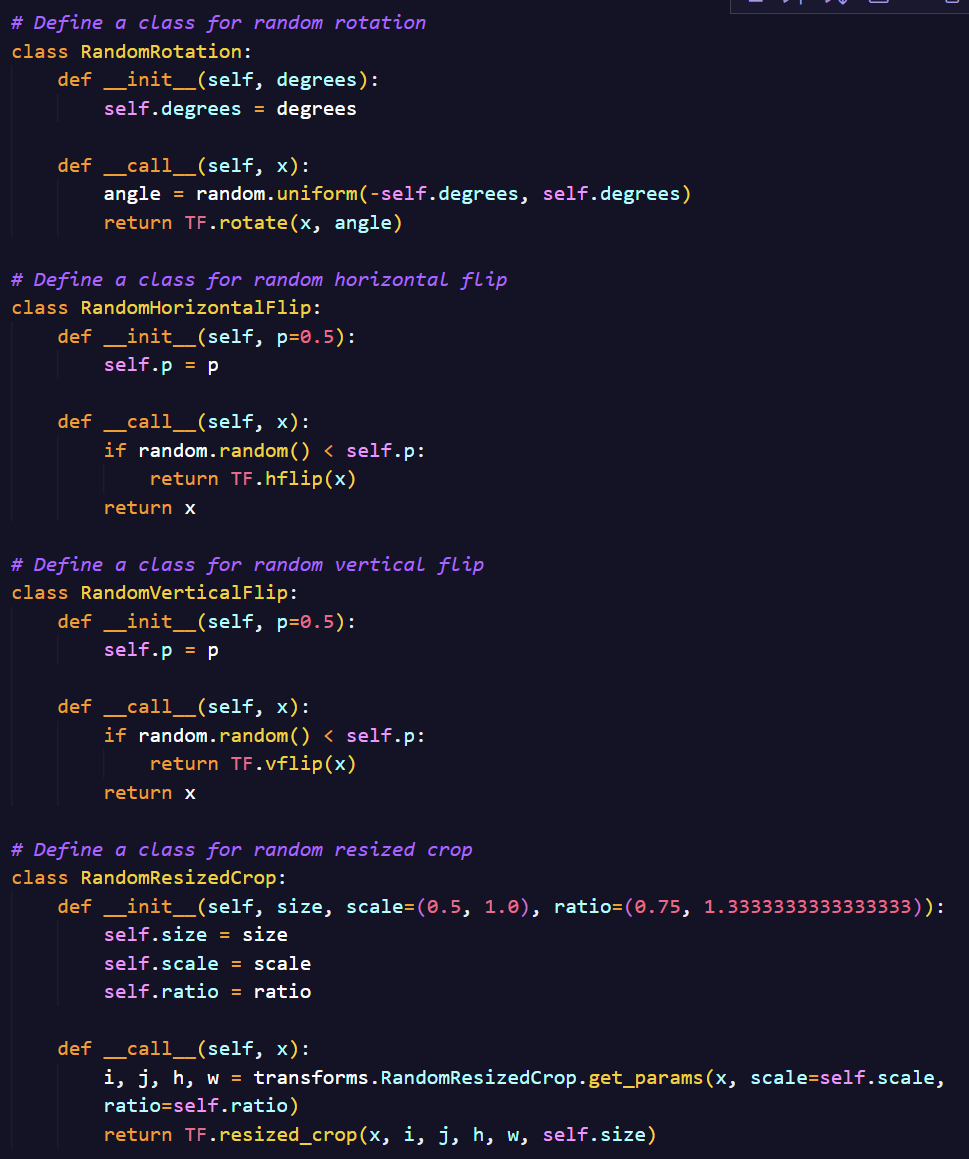
*Celda de jupyter notebook mostrando algunos ejemplos del dataset.*

## Data augmentation

Paso siguiente, defino las transformaciones para hacer data augmentation como se muestra en la Figura 6. Esto solo lo aplico al dataset de entrenamiento.

### Figura 6

*Transformaciones*



*Clases de python que definen transformaciones.*

Las apliqué con un factor de aumento de 4, y creé un nuevo Dataset con la nueva data aumentada.

### Figura 7

*Data augmentation*

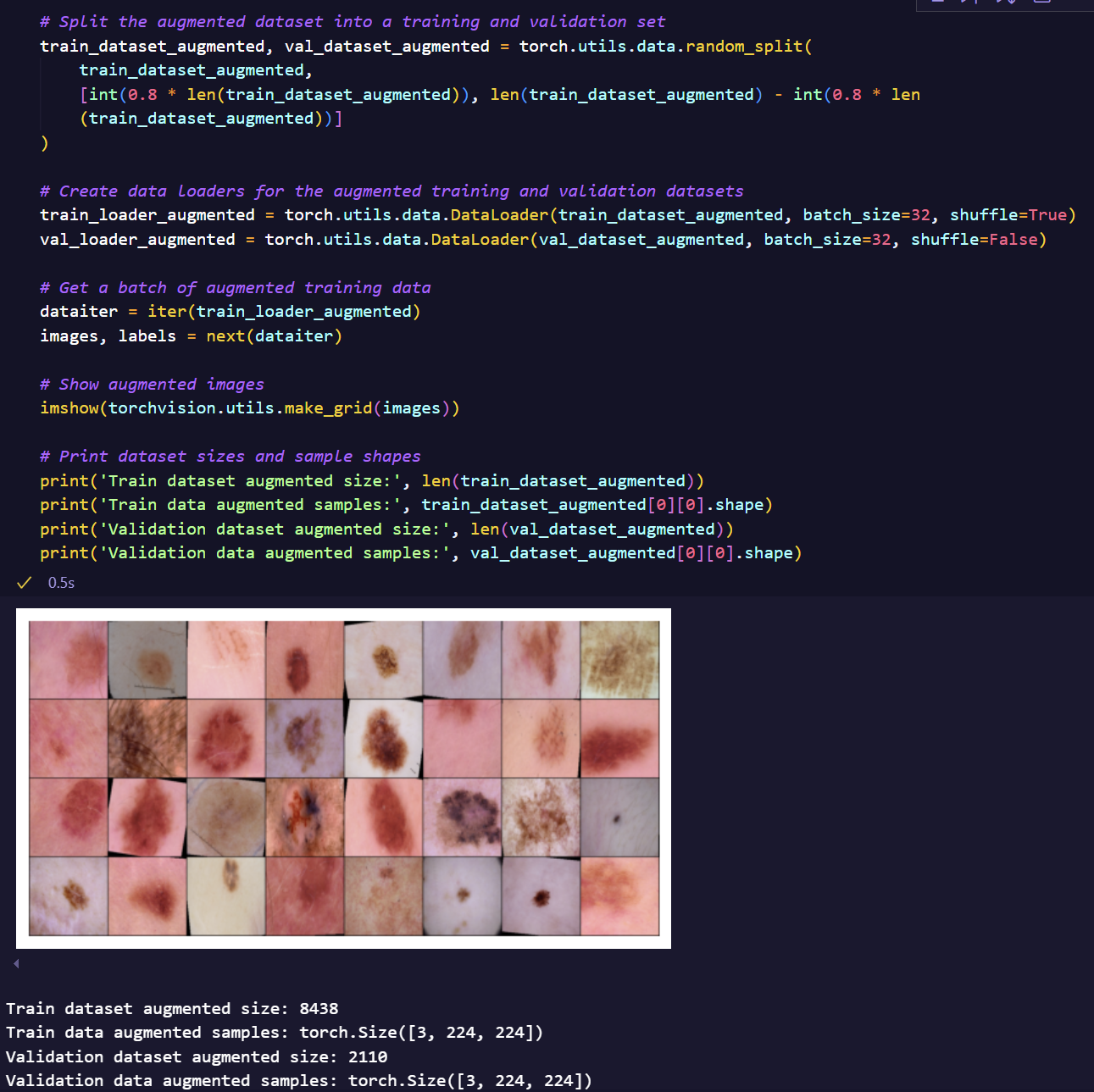


*Celda de jupyter notebook aplicando data augmentation.*

Dividí el dataset en train y validation. Visualizamos las nuevas imágenes que se muestran en la Figura 8.

### Figura 8

*Visualizando aumentada*

*Celda de jupyter notebook visualizando las nuevas imágenes generadas.*

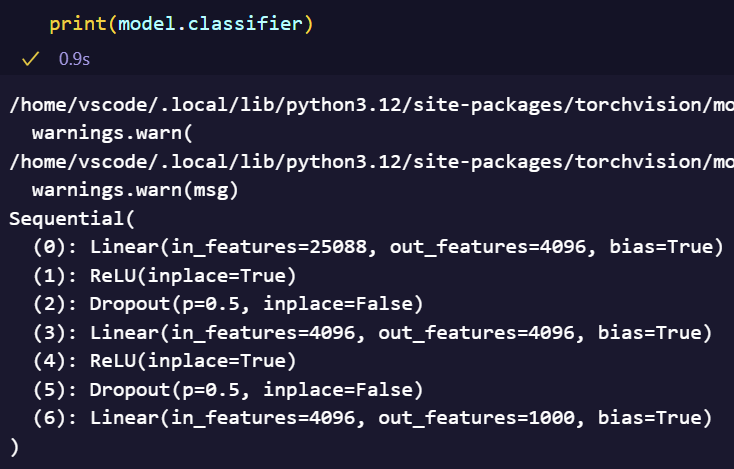
## El entrenamiento

Paso siguiente, descargo e instancio el modelo vgg16 pre entrenado con la librería torchvision y su modelo models.vgg16(pretrained=True).

Luego visualizé las capas del clasificador que se aprecian en la Figura 9.

### Figura 9

*Capas del clasificador*

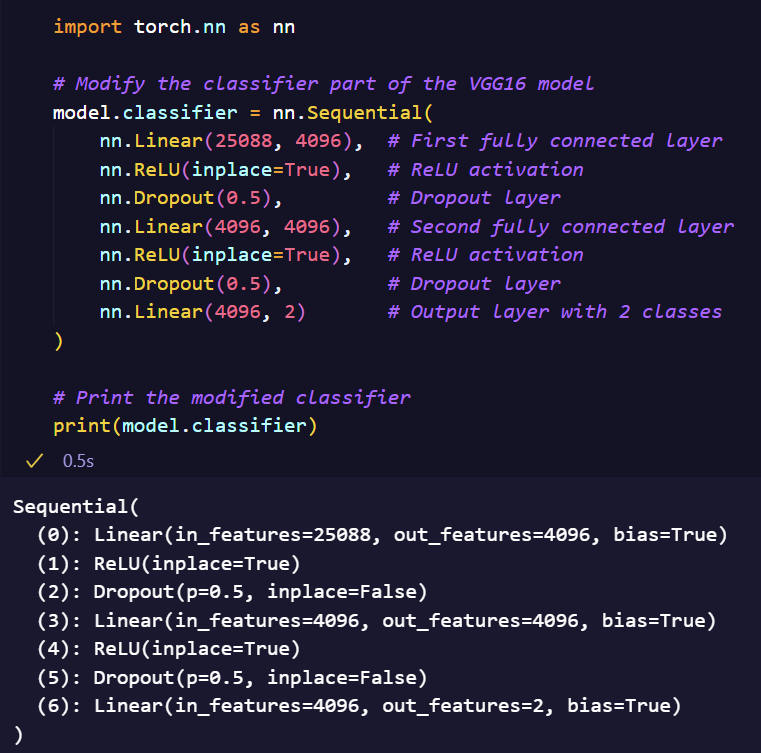


*Celda de jupyter notebook mostrando las capas del clasificador.*

Reemplazo la cabeza por una nueva mostrada en la Figura 10.

### Figura 10

*Nueva cabeza*



*Celda de jupyter notebook definiendo la nueva cabeza.*

Preparo las variables para entrenar el modelo. También creo las variables train\_losses y train\_accuracies para visualizarlas luego.

Ahora, defino el entrenamiento mostrado en la Figura 11.

### Figura 11

*El entrenamiento*

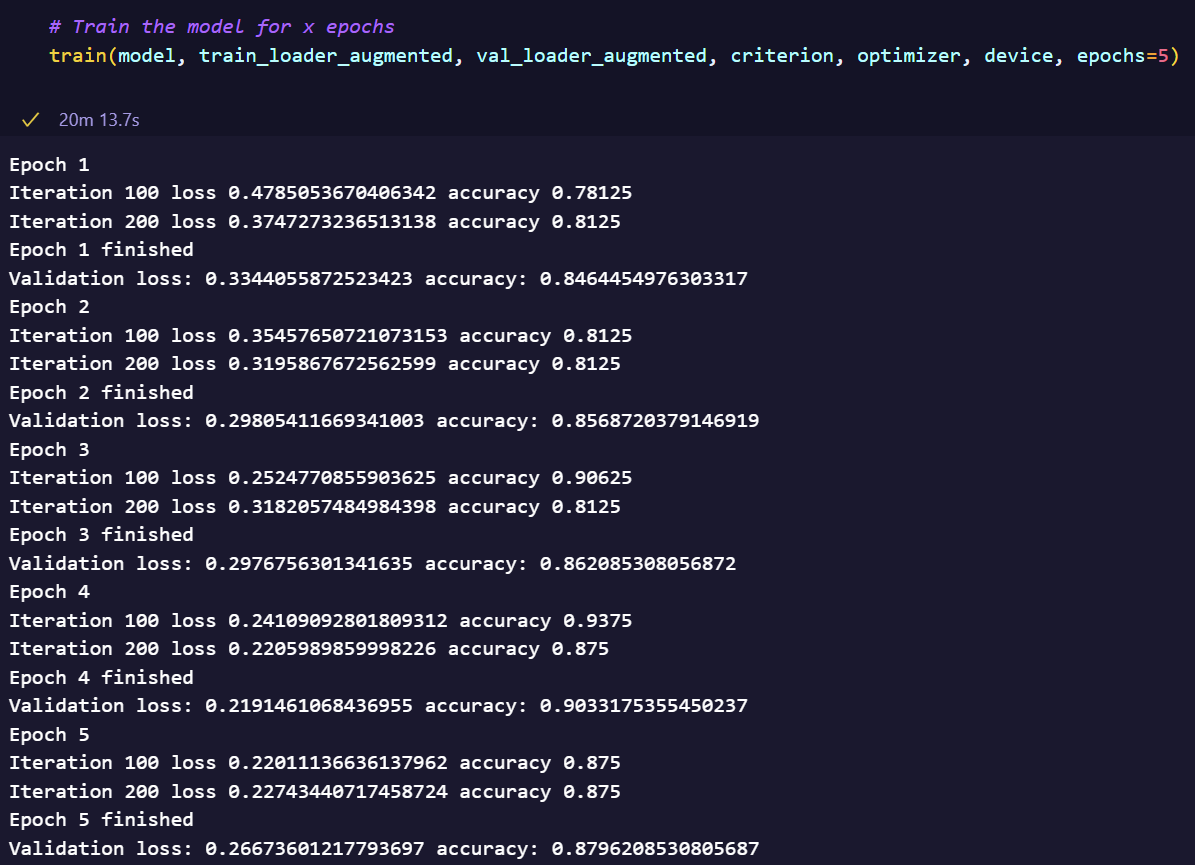


*Definición del entrenamiento con Pytorch*

Lo he corrido durante 5 epochs, dando los resultados de accuracy y lost para los datos de entrenamiento y validación mostrados en la Figura 12.

### Figura 12

*Entrenando*

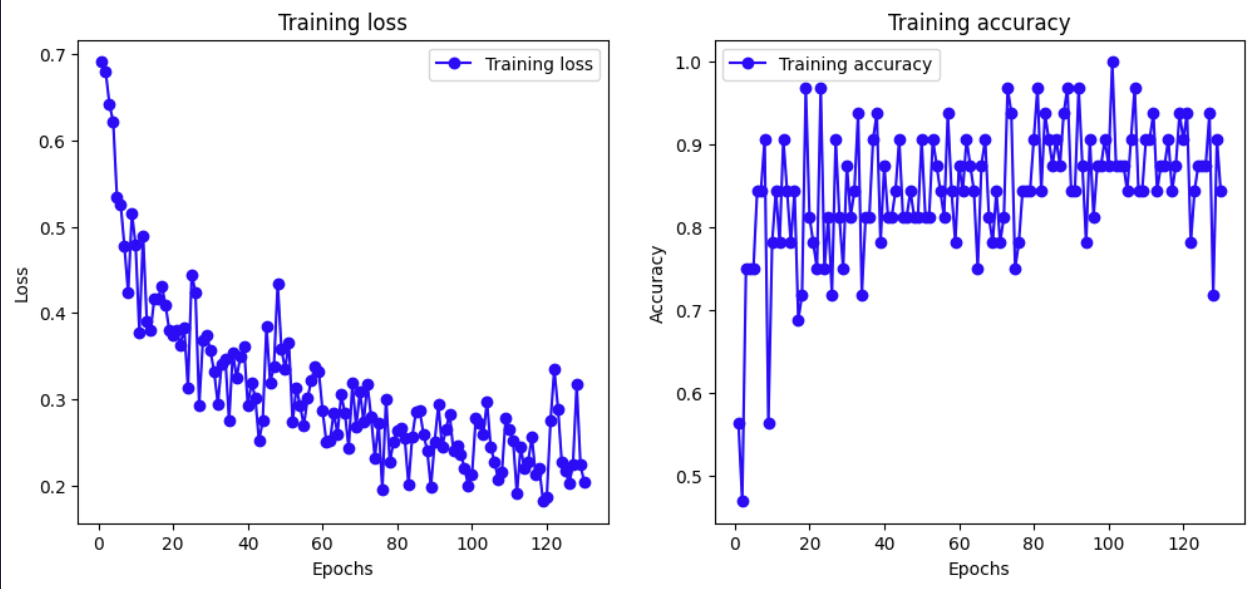
*Celda de jupyter notebook con el resultado del entrenamiento.*

## La historia del entrenamiento

La Figura 13 muestra un plot de los losses y accuracies de la fase de entrenamiento.

### Figura 13

*Historia del entrenamiento*

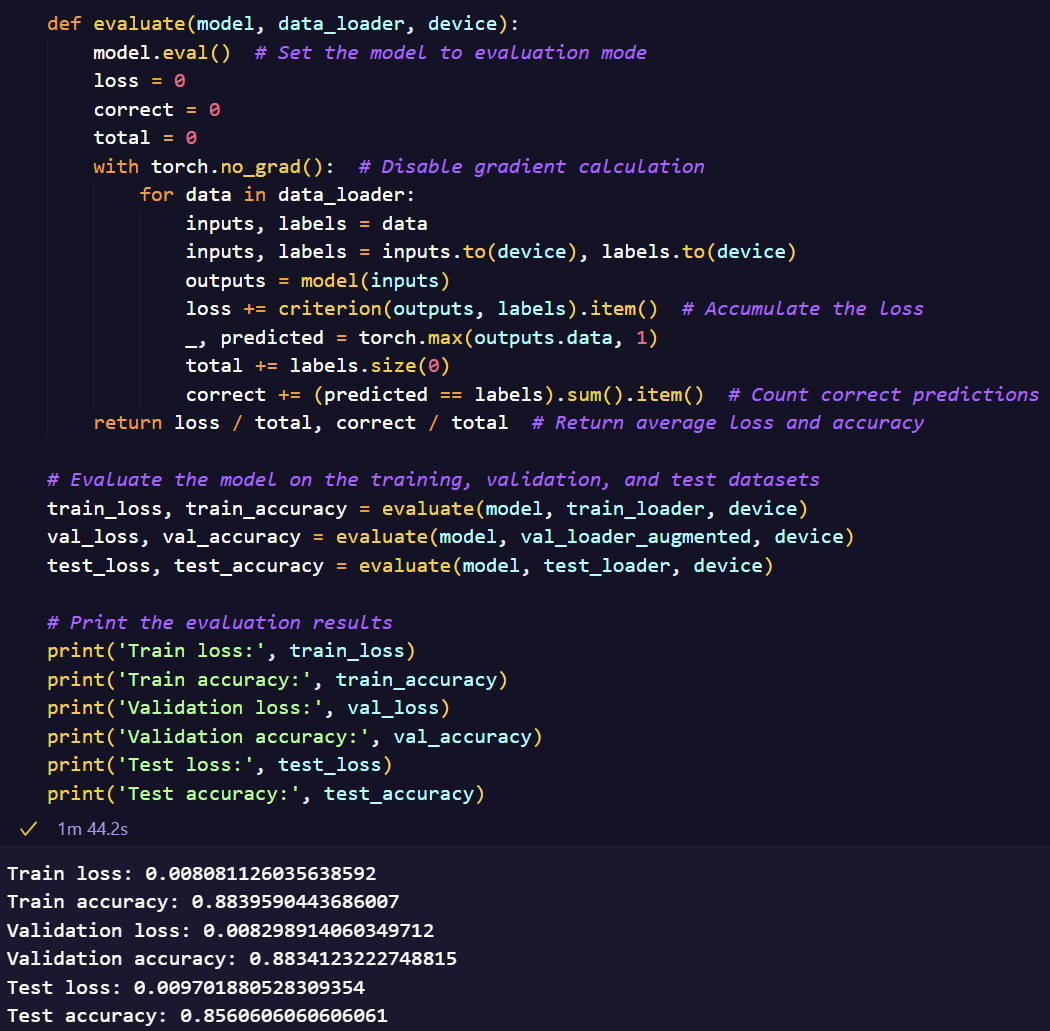
*Grafico del losses y accuracies del entrenamiento.*

## La evaluación

Finalmente, evalúo el modelo con los 3 datasets (train, evaluation y test) con el código de la Figura 14.

### Figura 14

*Evaluación*

*Celda de jupyter notebook con el resultado del loss y accuracy de los datasets de entrenamiento, validación y test.*

El resultado con el dataset de prueba es 85% de accuraccy, mientras que el de train y validation dan un 88%. Esto puede sugerir un pequeño sobreajuste, el cual lo dejaré para un futuro trabajo dada la demostración del caso.