UNIVERSIDAD CATOLICA SAN PABLO - AREQUIPA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Topicos en Inteligencia Artificial

Informe

Laboratorio: Clasificación Binaria

Advisor: Ochoa Luna Jose Eduardo
Yari Ramos Yessenia Deysi
Student: Loayza Huarachi Angel Josue

AREQUIPA, NOVEMBER 2024

Contents

1 Introducción

El objetivo de este proyecto es desarrollar un modelo de red neuronal convolucional (CNN) para clasificar imágenes médicas y determinar si contienen o no melanoma. Para optimizar el rendimiento del modelo, exploraremos configuraciones arquitectónicas y estrategias de preprocesamiento que permitan una clasificación precisa y robusta. Link de Github: https://github.com/angel452/CNN_Melanoma

Metodología: Para lograr el objetivo, seguiremos una metodología estructurada que incluye las siguientes etapas:

- **Preprocesamiento y Carga de Datos:** Utilizamos un conjunto de datos de imágenes de melanoma, organizadas en carpetas de "Melanoma" y "NotMelanoma". Estas imágenes serán preprocesadas y cargadas mediante la biblioteca **ImageDataGenerator** de Keras.
- **Construcción de Modelos:** Se configuro 2 arquitecturas de CNN. Se probó con modelos variando el número de capas, funciones de activación e inicializaciones de parámetros. También comparamos la efectividad de redes convolucionales con y sin **dropout**.
- **Aumento de Datos:** Para mejorar la generalización del modelo, aplicamos técnicas de aumento de datos (data augmentation) a las imágenes, incluyendo rotaciones, desplazamientos y ajustes de brillo para ver su comportamiento con los datos de prueba.
- **Entrenamiento y Evaluación:** Entrenamos los modelos en imágenes en color y en blanco y negro, evaluando su rendimiento en el conjunto de validación y, finalmente, en el conjunto de prueba. Se utilizan gráficos de pérdida para visualizar y detectar posibles problemas de sobreajuste.
- **Evaluación Final:** Basándonos en el rendimiento de los modelos en las distintas configuraciones, seleccionamos el mejor modelo y evaluamos su precisión y pérdida en el conjunto de prueba.

2 Implementación

Primero, se realizó la descarga del DataSet; luego, la creación de generadores de imágenes; seguido, la preparación de los modelos y su respectivo entrenamiento para 2 tipos de redes neuronales CNN con diferente configuración; y por último, la evaluación final con los datos de prueba del mejor modelo.

Adicionalmente, para probar de más maneras dichos modelos, se realizo posteriormente un procesamiento de datos. Primero cambiar la escala de colores a solo 1 canal, y luego aumento de datos a este mismo conjunto de datos.

2.1 Generadores de imágenes

Para cargarlas imágenes se utilizaron generadores de imágenes mediante la clase ImageDataGenerator de Keras. Este enfoque permite la carga eficiente de las imágenes en lotes (batches) y facilita el aumento de datos (data augmentation) en el conjunto de entrenamiento. En este caso, se aplicaron transformaciones como escalado (normalización de píxeles entre 0 y 1), traslación (shear_range), zoom (zoom_range) y volteo horizontal (horizontal_flip).

Se crearon tres generadores específicos:

- Generador de Entrenamiento (train_generator): Imágenes de la carpeta train_sep. La resolución de las imágenes se ajustó a (128, 128) píxeles. Se dividió las imágenes en lotes de 32 para el entrenamiento, y también una clasificación binaria (class_mode='binary'), pues queremos diferenciar entre "Melanoma" y "NotMelanoma".

- Generador de Validación (validation_generator): Proviene de la carpeta "valid" y se escalan de la misma manera.

- Generador de Prueba (test_generator): Imágenes de la carpeta "test". Se especifica el parámetro shuffle como falso para preservar el orden de las imágenes y facilitar la interpretación de los resultados.

En la siguiente imagen 2.1 se puede ver unos cuantos ejemplos del DataSet obtenido con sus respectivas etiquetas:

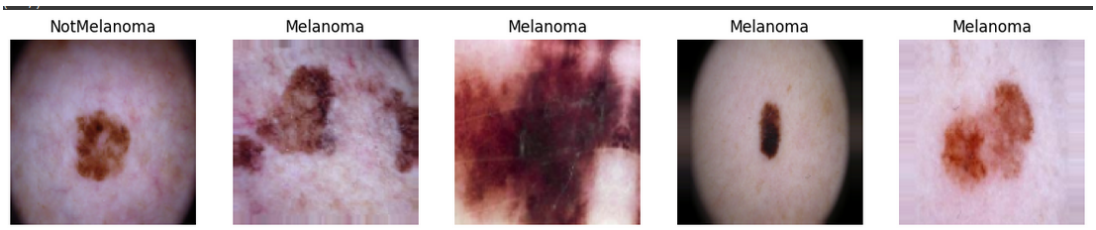

Figure 2.1: Ejemplos del datset recopilado

2.2 Preparación de los modelos

Se construyeron dos modelos de redes neuronales convolucionales (CNN). Ambos con diferentes configuraciones y técnicas de inicialización de parámetros:

- **Modelo 1 (modeloCNN):** Consta de dos capas convolucionales, cada una seguida por una capa de agrupación (MaxPooling). La función de activación utilizada en las capas convolucionales es ReLU, y las capas están inicializadas con el método he_normal. Para la capa densa final, se aplica una activación sigmoid para la clasificación binaria.

- **Modelo 2 (modeloCNN2):** Contiene cuatro capas convolucionales. Se introducen variaciones en las funciones de activación, como LeakyReLU en la primera capa y ReLU en el resto. Adicionalmente, incluye una capa de Dropout para reducir el riesgo de sobreajuste y la inicialización de parámetros se alterna entre he_normal y glorot_uniform.

Ambos modelos fueron compilados usando el optimizador Adam, que ajusta automáticamente la tasa de aprendizaje, y la función de pérdida binary_crossentropy, adecuada para la clasificación binaria entre "Melanoma" y "NotMelanoma".

2.3 Entrenamiento

Los modelos se entrenaron durante 10 épocas, con tiempos aproximados por época de 110 segundos para el Modelo 1 (modeloCNN) y 130 segundos para el Modelo 2 (modeloCNN2), siendo este último más costoso en tiempo debido a sus capas adicionales y la capa de Dropout como se puede ver en la Figura 2.2 y 2.3. Se utilizó TensorBoard como herramienta de seguimiento, almacenando los registros en la carpeta logs para monitorear en tiempo real el progreso de la precisión y pérdida en entrenamiento y validación.

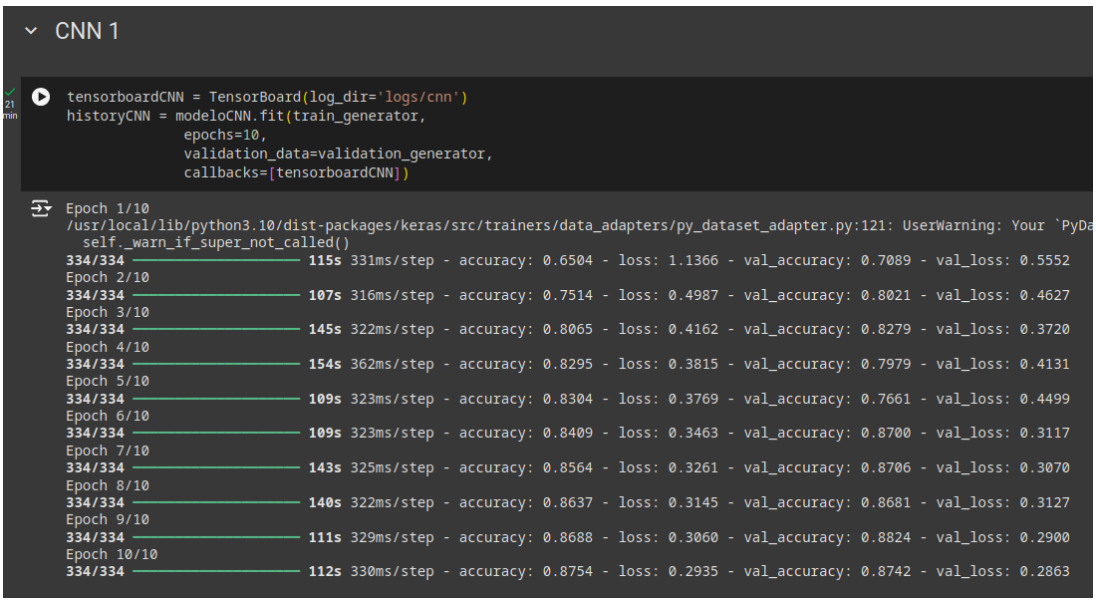

Figure 2.2: Tiempos de entrenamiento por época para el Modelo 1 (CNN más simple)

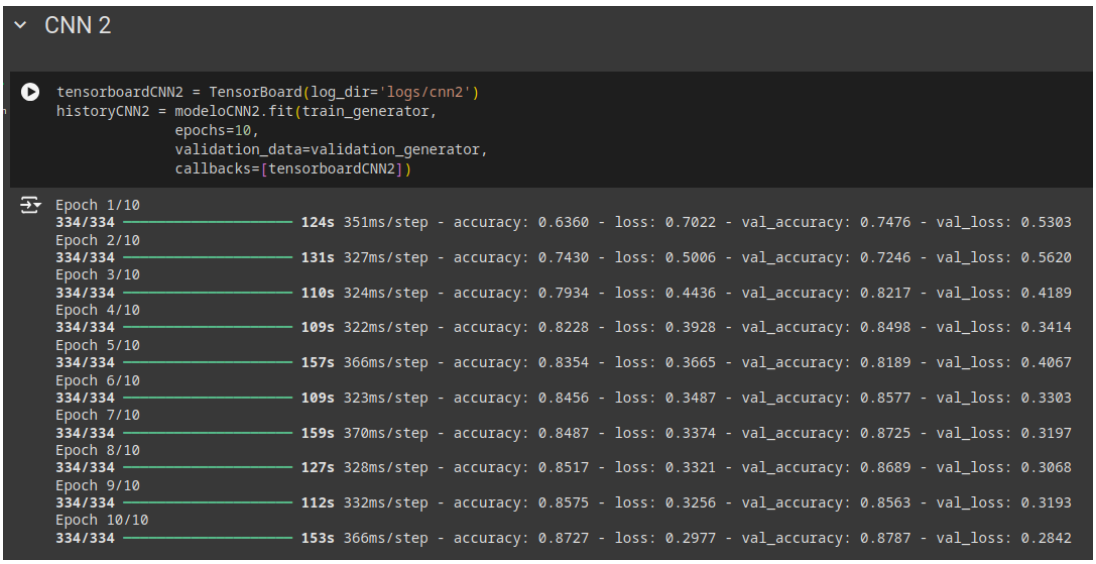

Figure 2.3: Tiempos de entrenamiento por época para el Modelo 2 (CNN más profunda)

3 Resultados

A continuación se presentan los gráficos obtenidos, mostrando la evolución de la pérdida y la precisión a lo largo de las épocas. El uso de TensorBoard resultó especialmente útil, ya que permite generar y visualizar estos gráficos de manera automática, facilitando el análisis detallado del rendimiento de los modelos durante el entrenamiento.

3.1 Modelo 1 (modeloCNN)

Perdida Según las figuras 3.1 y 3.2 podemos deducir:

- Entrenamiento: La pérdida durante la etapa de entrenamiento llega a obtener **0.2864**, lo que significa que el modelo está minimizando el error en sus predicciones.
- Validación: La pérdida en la etapa de validación fue muy similar, en **0.2863**, lo que también indica que el modelo se está comportando de manera consistente en ambos conjuntos de datos.
- Ambas líneas de pérdida muestran un comportamiento paralelo y en descenso, lo que sugiere que el modelo se está ajustando adecuadamente a los datos y no está sufriendo de sobreajuste en esta etapa.

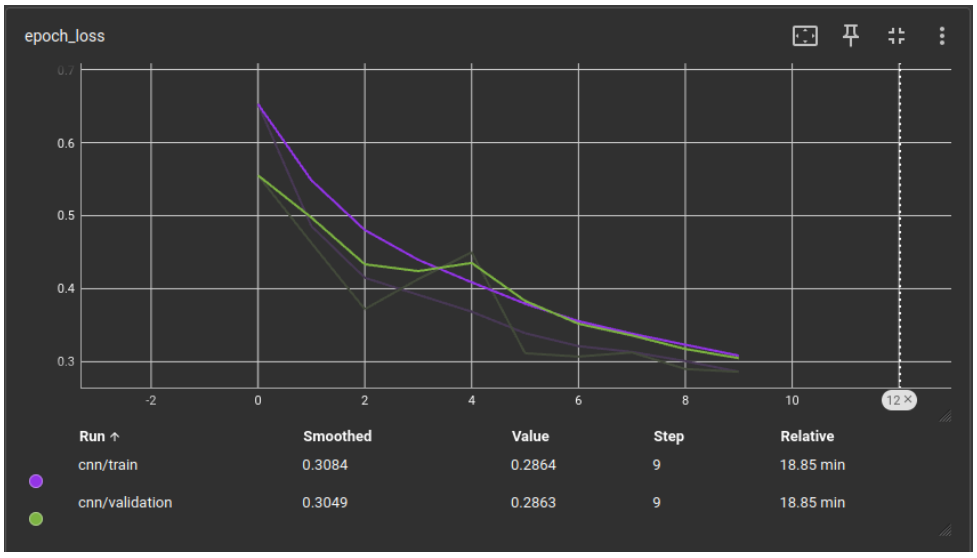

Figure 3.1: Función de perdida con Tensorboard para el modelo 1

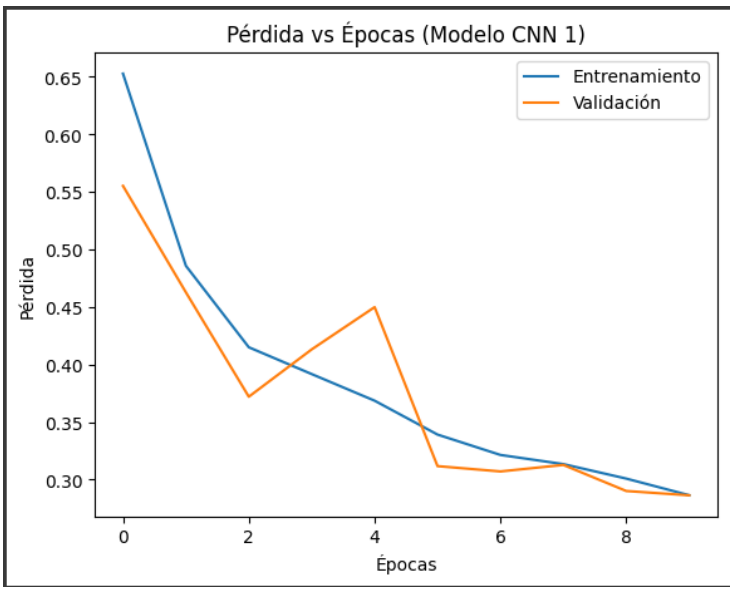

Figure 3.2: Función de perdida con MatplotLib para el modelo 1

Precisión Según la figura 3.3 podemos deducir:

- Entrenamiento: La precisión en la etapa de entrenamiento fue de aproximadamente **0.8776**, lo que indica que alrededor del 87.76% de las imágenes fueron clasificadas correctamente por el modelo durante esta fase.

- Validación: La precisión en la etapa de validación fue ligeramente menor, alcanzando **0.8742**. Esto sugiere que el modelo tiene un buen desempeño no solo en los datos de entrenamiento, sino también en datos que no ha visto anteriormente, lo cual es un indicativo de su capacidad de generalización.

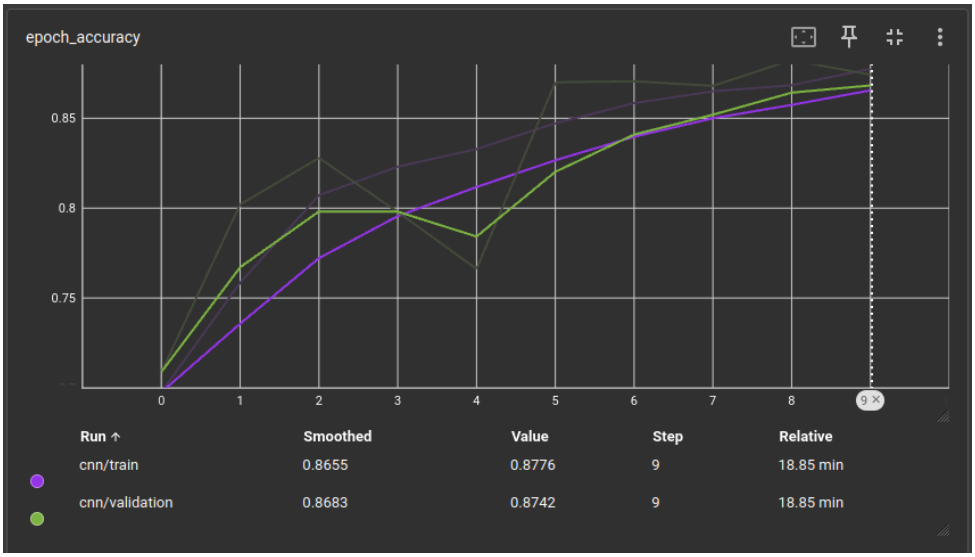

Figure 3.3: Nivel de precisión para el Modelo 1 (modeloCNN1)

3.2 Modelo 2 (modeloCNN2)

Perdida Según las figuras 3.4 y 3.5 podemos deducir:

- Entrenamiento: La pérdida durante la etapa de entrenamiento llega a obtener **0.2986**, lo que indica un error relativamente bajo en las predicciones del modelo durante el entrenamiento.
- Validación: La pérdida en la etapa de validación fue muy similar, en **0.2842**, sugiriendo que el modelo también se desempeña bien en datos no vistos, con un error de predicción similar al de la fase de entrenamiento.
- Al igual que en el Modelo 1, las líneas de pérdida presentan un comportamiento alineado, mostrando una tendencia a la baja en ambas fases, lo que indica que el modelo está aprendiendo adecuadamente y ajustándose a los datos.

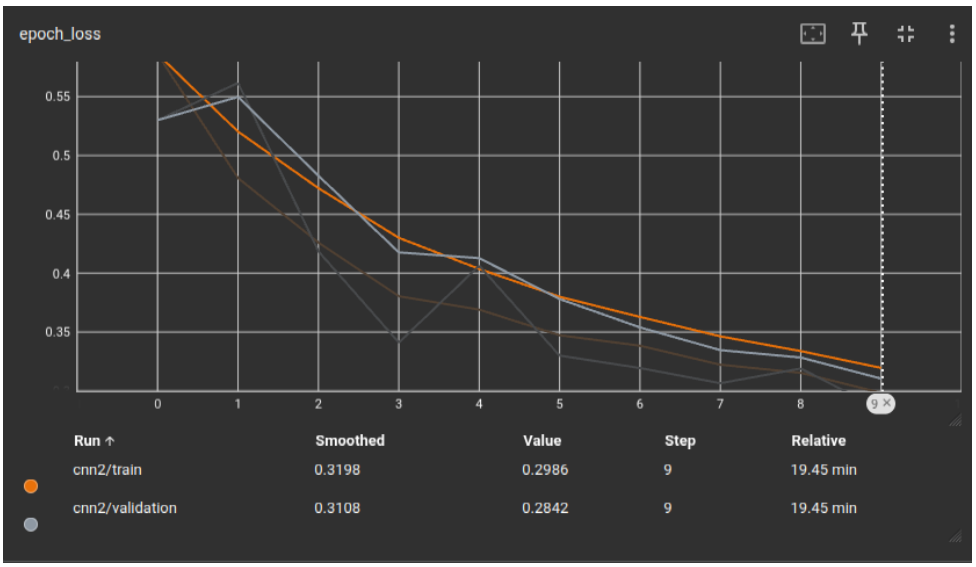

Figure 3.4: Función de perdida con Tensorboard para el modelo 2

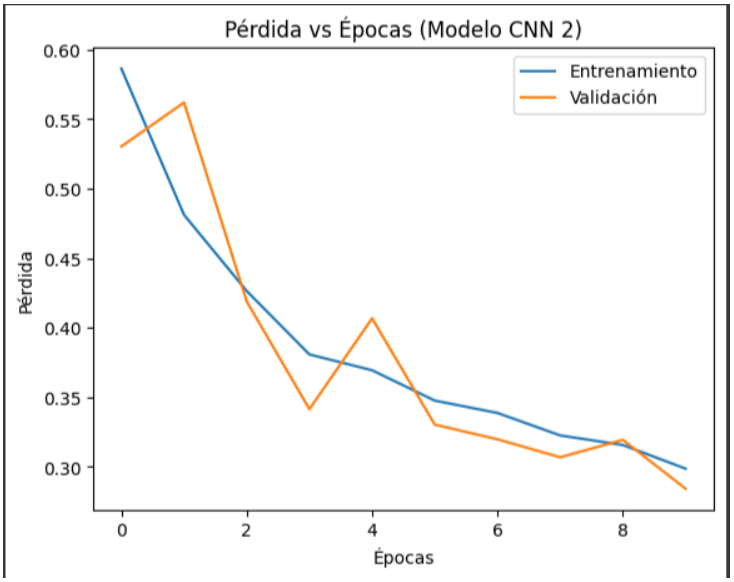

Figure 3.5: Función de perdida con MatplotLib para el modelo 2

Precisión Según la figura 3.6 podemos deducir:

- Entrenamiento: La precisión en la etapa de entrenamiento fue de aproximadamente **0.8719**, lo que indica que alrededor del 87.19% de las imágenes fueron clasificadas correctamente por el modelo durante esta fase.
- Validación: La precisión en la etapa de validación fue ligeramente menor, alcanzando **0.8787** que es ligeramente superior a la del conjunto de entrenamiento. Este

resultado sugiere que el modelo también está manejando bien los datos no vistos, lo cual es un indicador positivo de su capacidad de generalización.

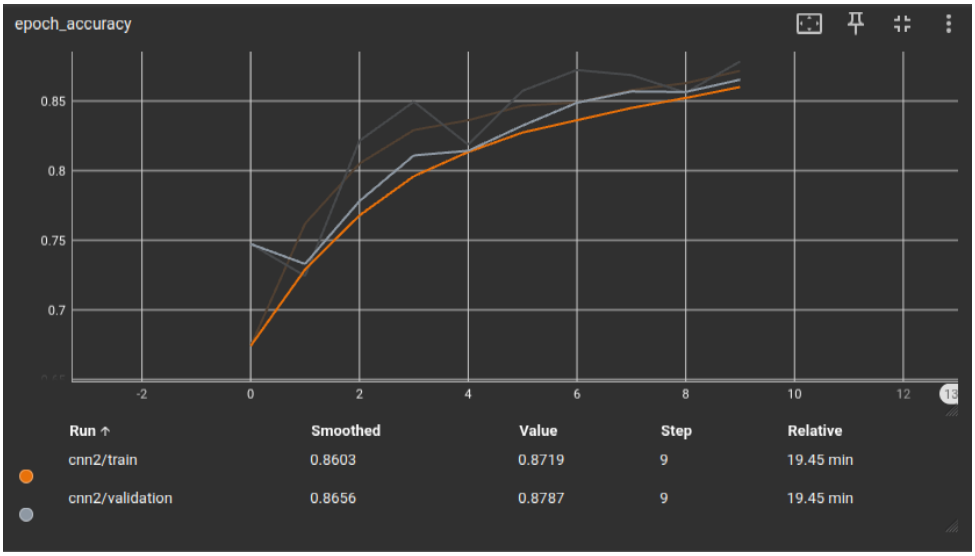

Figure 3.6: Nivel de precisión para el Modelo 2 (modeloCNN2)

3.3 Evaluación con el mejor modelo

Se eligió el Modelo 2 como el mejor debido a su rendimiento superior en las métricas de validación, donde mostró una mayor precisión y una menor pérdida en comparación con el Modelo 1. Esta elección se fundamentó en las gráficas de precisión y pérdida generales en las figuras 3.9 y 3.8, que indicaron que el Modelo 2 tenía un mejor ajuste a los datos. Posteriormente, se realizó la evaluación del modelo con el conjunto de prueba, obteniendo resultados prometedores: una precisión del **90.16%** y una pérdida de **0.2433**. Los resultados finales del conjunto de prueba reflejan una exactitud del **87.00%**, lo que valida la efectividad del modelo en la clasificación de imágenes como se puede ver en la figura 3.7.

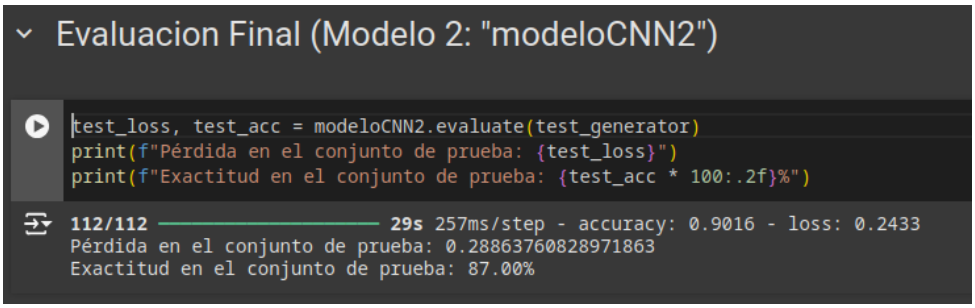

Figure 3.7: Resultados obtenidos tras evaluar los datos de prueba con el Modelo 2

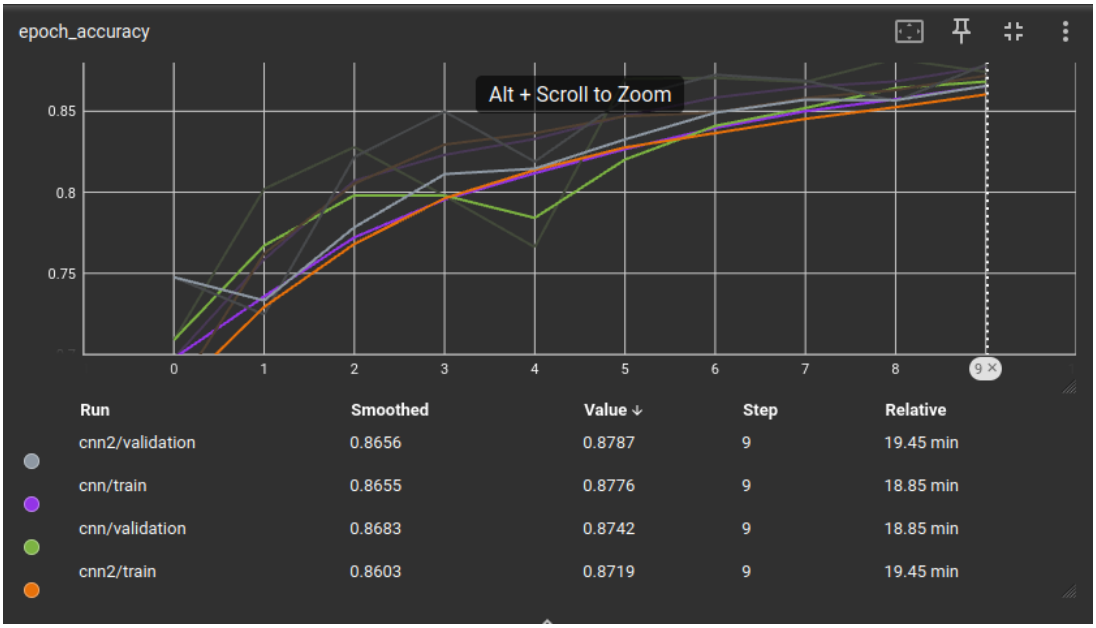

Figure 3.8: Gráfico de nivel de precisión general para ambos modelos

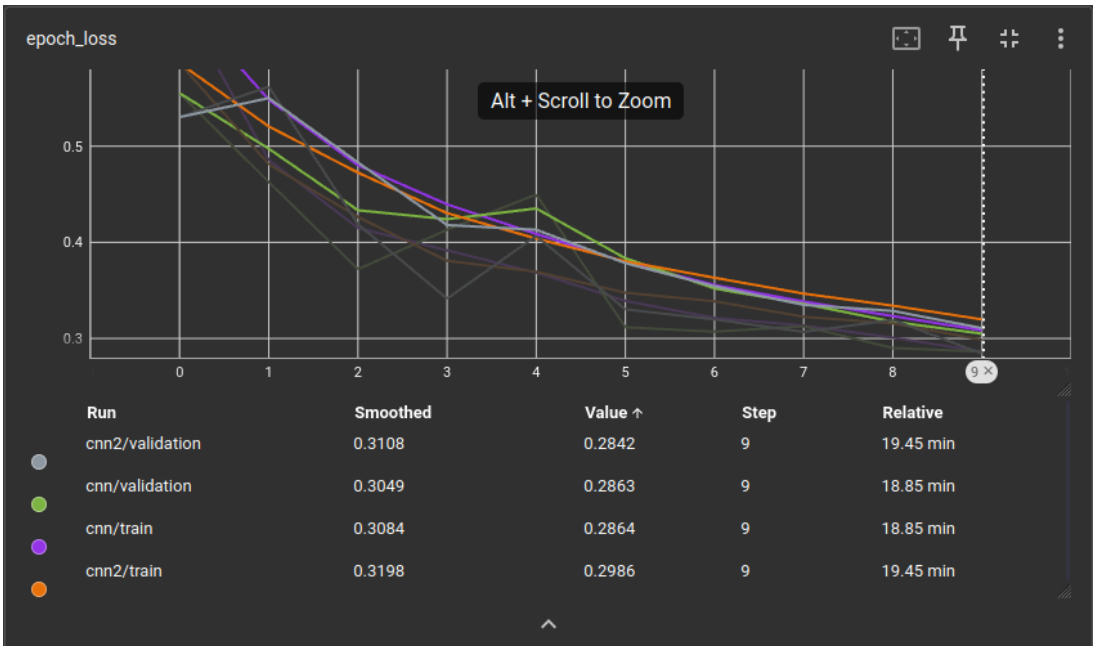

Figure 3.9: Gráfico de nivel de perdida general para ambos modelos

4 Plus: Cambio de escala de colores

Luego de esta evaluación, se implementó una transformación en el conjunto de datos mediante la conversión de imágenes a escala de grises, lo cual tenía como objetivo reducir

el tiempo de entrenamiento por época, permitiendo así incrementar el número de épocas. Para llevar a cabo esta modificación, se ajustó el parámetro **color_mode='grayscale'** en el generador de imágenes, y también se actualizó la configuración de los modelos, cambiando la dimensión de entrada a **input_shape=(128, 128, 1)**. En la figura 4.1 se puede ver un conjunto pequeño de muestras de como quedaron estas nuevas imágenes tras la transformación con sus respectivas etiquetas.

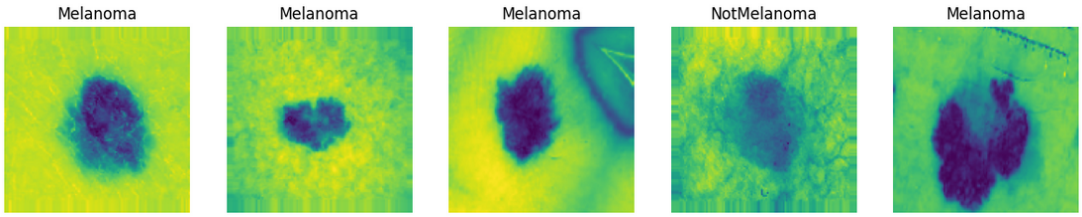

Figure 4.1: Nuevas imagenes tras la transformacion color_mode='greyscale'

Tras realizar estos cambios, se observó una disminución notable en el tiempo de entrenamiento, que se redujo a aproximadamente 70 segundos por época, como se muestra en la figura 4.2

```
tensorboardCNN2_bw = TensorBoard(log_dir='logs/cnn2_bw')
history_bw = modeloCNN2_bw.fit(train_generator_bw,
                               epochs=10,
                               validation_data=validation_generator_bw,
                               callbacks=[tensorboardCNN2_bw])

Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
334/334 ━━━━━━━━━━━━━━━━━━━━ 118s 327ms/step - accuracy: 0.5356 - loss: 0.6876 - val_accuracy: 0.7369 - val_loss: 0.5835
Epoch 2/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 105s 235ms/step - accuracy: 0.7722 - loss: 0.5091 - val_accuracy: 0.7973 - val_loss: 0.4483
Epoch 3/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 80s 231ms/step - accuracy: 0.7967 - loss: 0.4540 - val_accuracy: 0.8316 - val_loss: 0.3981
Epoch 4/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 81s 229ms/step - accuracy: 0.7996 - loss: 0.4520 - val_accuracy: 0.8203 - val_loss: 0.4022
Epoch 5/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 79s 234ms/step - accuracy: 0.8147 - loss: 0.4203 - val_accuracy: 0.8172 - val_loss: 0.4006
Epoch 6/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 77s 225ms/step - accuracy: 0.8207 - loss: 0.4069 - val_accuracy: 0.8377 - val_loss: 0.3826
Epoch 7/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 81s 226ms/step - accuracy: 0.8134 - loss: 0.4196 - val_accuracy: 0.8223 - val_loss: 0.3974
Epoch 8/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 77s 228ms/step - accuracy: 0.8312 - loss: 0.3904 - val_accuracy: 0.8304 - val_loss: 0.3978
Epoch 9/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 80s 238ms/step - accuracy: 0.8306 - loss: 0.3949 - val_accuracy: 0.8484 - val_loss: 0.3561
Epoch 10/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 78s 226ms/step - accuracy: 0.8355 - loss: 0.3765 - val_accuracy: 0.8366 - val_loss: 0.3590
```

Figure 4.2: Nuevos tiempos de entrenamiento por época con el nuevo DataSet

Al evaluar el modelo con el conjunto de prueba, se obtuvieron los siguientes resultados: una pérdida de **0.2905** y una exactitud del **88.40%**. La evaluación final en el conjunto de prueba mostró una pérdida de **0.3948** y una exactitud del **81.80%**, lo que sugiere que, aunque el tiempo de entrenamiento se redujo, el rendimiento del modelo también experimentó una disminución moderada en comparación con las versiones anteriores del

modelo. De igual manera, viendo las gráficas de precisión y perdida según las figuras 4.3 y 4.4, vemos que ambas líneas presentan un comportamiento alineado, indicando que el modelo sí intento aprender adecuadamente.

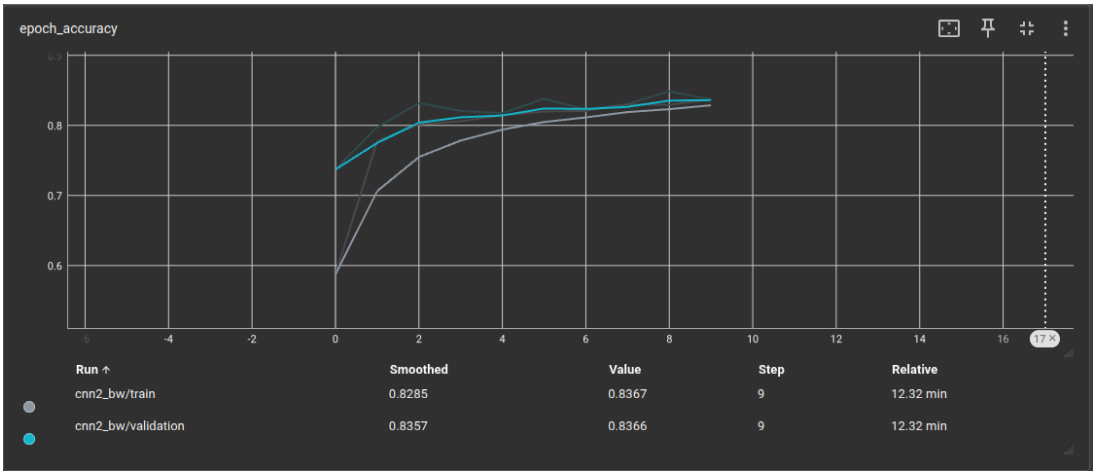

Figure 4.3: Gráfica de presicion con imágenes transformadas

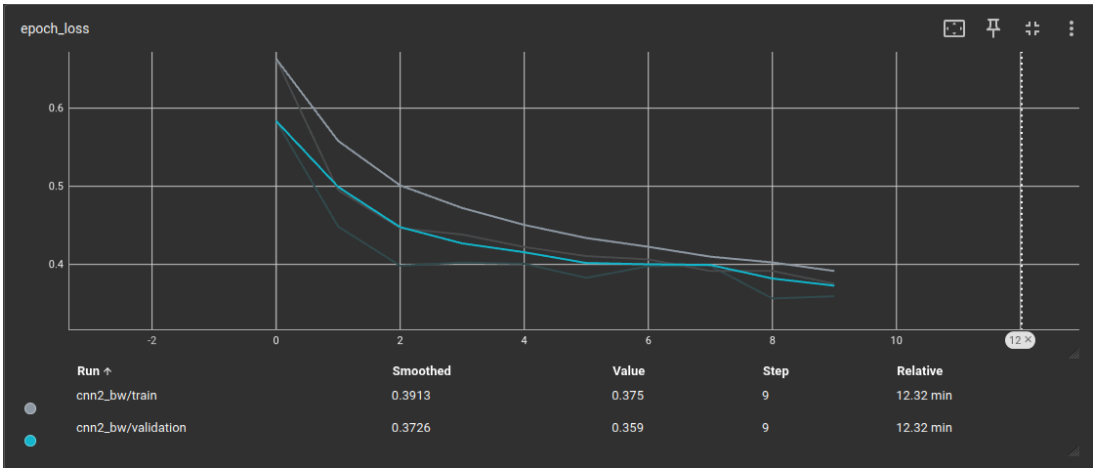

Figure 4.4: Gráfica de perdida con imágenes transformadas

5 Plus: Aumento de datos

Tras observar que la conversión a escala de grises no resultó en una mejora significativa en el rendimiento del modelo, se decidió implementar técnicas de aumento de datos con el propósito de compensar esta disminución en la eficacia. Se generaron nuevos generadores de imágenes aplicando diversas transformaciones, que incluyen:

- rotation_range=20: Rotación aleatoria de hasta 20 grados.

- width_shift_range=0.2: Desplazamiento horizontal aleatorio.
- height_shift_range=0.2: Desplazamiento vertical aleatorio.
- brightness_range=[0.8, 1.2]: Ajuste aleatorio del brillo dentro de un rango especificado.

Estas transformaciones se pueden observar en la figura 5.1.

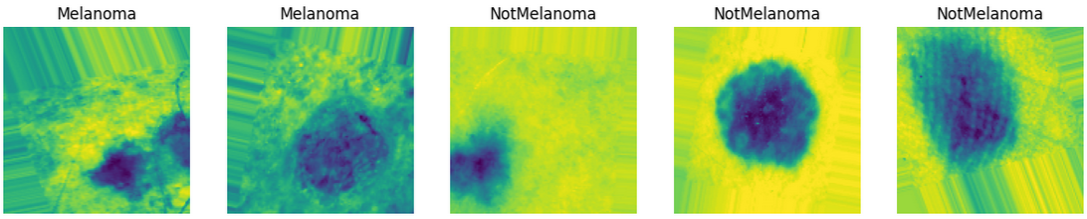

Figure 5.1: Nuevas imágenes tras Aumento de datos

Durante el entrenamiento bajo el mismo modelo, se notó un ligero aumento en los tiempos, que oscilaron entre 80 y 100 segundos por época, como se puede ver en la figura 5.2.

```
tensorboardCNN2_bw_aug = TensorBoard(log_dir='logs/cnn2_bw_aug')
history_bw_aug = modeloCNN2_bw.fit(train_generator_bw_aug,
                                   epochs=10,
                                   validation_data=validation_generator_bw_aug,
                                   callbacks=[tensorboardCNN2_bw_aug])
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDat
  self._warn_if_super_not_called()
334/334 ━━━━━━━━━━━━━━━━━━━━ 85s 250ms/step - accuracy: 0.7338 - loss: 0.5493 - val_accuracy: 0.7622 - val_loss: 0.5040
Epoch 2/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 83s 246ms/step - accuracy: 0.7623 - loss: 0.4928 - val_accuracy: 0.7793 - val_loss: 0.4701
Epoch 3/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 142s 248ms/step - accuracy: 0.7476 - loss: 0.5027 - val_accuracy: 0.7754 - val_loss: 0.4720
Epoch 4/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 82s 242ms/step - accuracy: 0.7649 - loss: 0.4854 - val_accuracy: 0.7695 - val_loss: 0.4817
Epoch 5/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 85s 252ms/step - accuracy: 0.7745 - loss: 0.4798 - val_accuracy: 0.7754 - val_loss: 0.4705
Epoch 6/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 83s 247ms/step - accuracy: 0.7656 - loss: 0.4896 - val_accuracy: 0.7656 - val_loss: 0.5031
Epoch 7/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 142s 246ms/step - accuracy: 0.7680 - loss: 0.4854 - val_accuracy: 0.7821 - val_loss: 0.4571
Epoch 8/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 85s 251ms/step - accuracy: 0.7841 - loss: 0.4637 - val_accuracy: 0.7723 - val_loss: 0.4817
Epoch 9/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 142s 250ms/step - accuracy: 0.7764 - loss: 0.4753 - val_accuracy: 0.7760 - val_loss: 0.4598
Epoch 10/10
334/334 ━━━━━━━━━━━━━━━━━━━━ 144s 256ms/step - accuracy: 0.7793 - loss: 0.4666 - val_accuracy: 0.7914 - val_loss: 0.4523
```

Figure 5.2: Tiempos de entrenamiento por época para datos con Data Augmentation

Al realizar la evaluación con el conjunto de prueba, los resultados fueron los siguientes: una pérdida de **0.4638** y una exactitud del **78.29%**. Estos resultados indican que, a pesar de las técnicas de aumento de datos implementadas, la efectividad del modelo se vio afectada, lo que sugiere que podría ser necesario explorar otras arquitecturas o configuraciones adicionales para mejorar el rendimiento en este conjunto de datos que fue

modificado respecto a sus colores. En las figuras 5.3 y 5.4 se ve el comportamiento del modelo según sus funciones de perdida y precisión, y al igual que los otros modelos, ambas líneas presentan un comportamiento alineado.

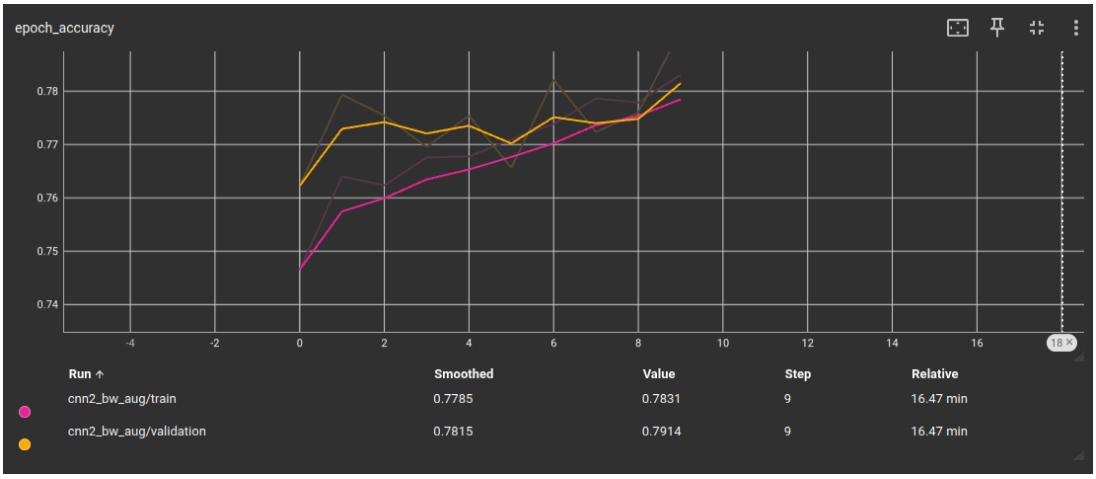

Figure 5.3: Nuevas imágenes tras Aumento de datos

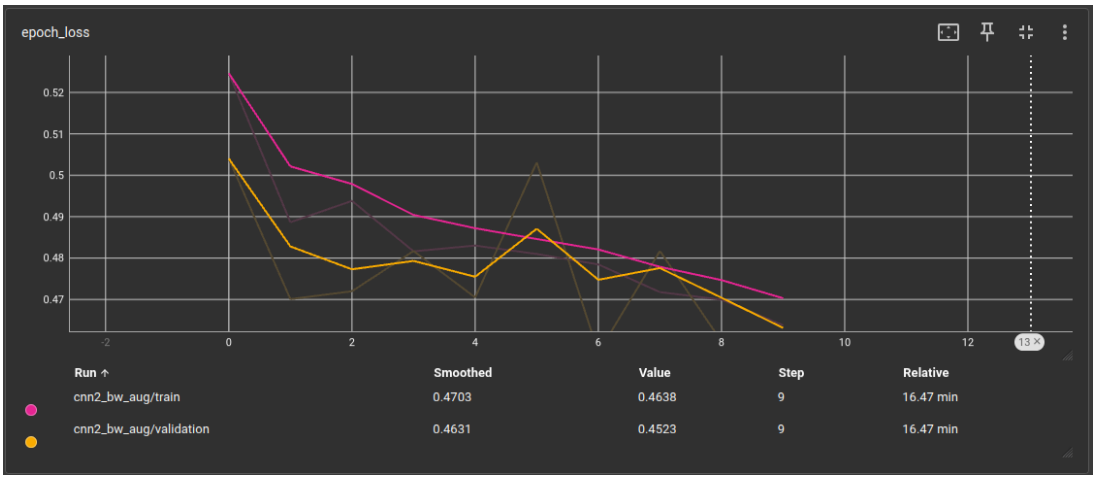

Figure 5.4: Nuevas imágenes tras Aumento de datos

6 Conclusiones

- Inicialmente, realizamos pruebas con imágenes de tamaño 224 x 224 píxeles. Aunque estas imágenes proporcionaban una mayor cantidad de información visual, el tiempo de procesamiento era de 140 segundos por época aproximadamente. Para bajar estos tiempos, se optó por reducir la resolución de las imágenes a 128 x 128 píxeles, lo cual disminuyó el tiempo de procesamiento a aproximadamente 100 segundos por época sin afectar significativamente la calidad de las predicciones.

- Consideramos reducir aún más la resolución de las imágenes, pero al visualizar ejemplos de imágenes con menor dimensionalidad, se notó una pérdida considerable de calidad. Por esta razón, se mantuvo el tamaño de 128 x 128 píxeles como el compromiso óptimo entre rendimiento y calidad de los datos visuales.

- Durante el proceso de entrenamiento y evaluación, se empleó TensorBoard, lo que facilitó el análisis de métricas clave, como precisión y pérdida, a lo largo de las épocas, permitiéndonos identificar patrones de rendimiento en los modelos de manera eficiente.

- El modelo 2 resultó ser el más efectivo para este conjunto de datos, con una arquitectura compuesta por cuatro capas convolucionales, una función de activación LeakyReLU en la primera capa, seguida de ReLU en las capas posteriores, y una capa de Dropout para mitigar el riesgo de sobreajuste. Los parámetros se inicializaron alternando entre he_normal y glorot_uniform. Este modelo mostró un balance adecuado entre precisión y generalización, como se observa en las imagenes generales de los modelos probados 7.1 y 7.2.

- La información de color parece haber sido fundamental en este estudio, ya que al reducir el conjunto de datos a escala de grises, la precisión del modelo disminuyó significativamente. Además, hubiera sido beneficioso probar con un mayor número de épocas en esta configuración para observar si un entrenamiento más extenso lograba compensar la pérdida de información de color.

- El aumento de datos aplicado al conjunto de imágenes transformadas a escala de grises no mostró mejoras notables en el rendimiento. Es posible que los resultados habrían sido mejores si el aumento de datos se hubiera aplicado directamente a las imágenes en color sin modificaciones previas, preservando así la información visual completa.

- Finalmente, este último experimento, que incluyó una transformación a escala de grises y aumento de datos, sugiere que el modelo podría haber sufrido de sobreajuste. A pesar de las transformaciones aplicadas, el rendimiento no mejoró sustancialmente en comparación con los otros modelos, lo que indica la importancia de elegir cuidadosamente tanto la preprocesamiento como las técnicas de aumento de datos en función de la naturaleza específica del conjunto de datos.

7 ANEXOS

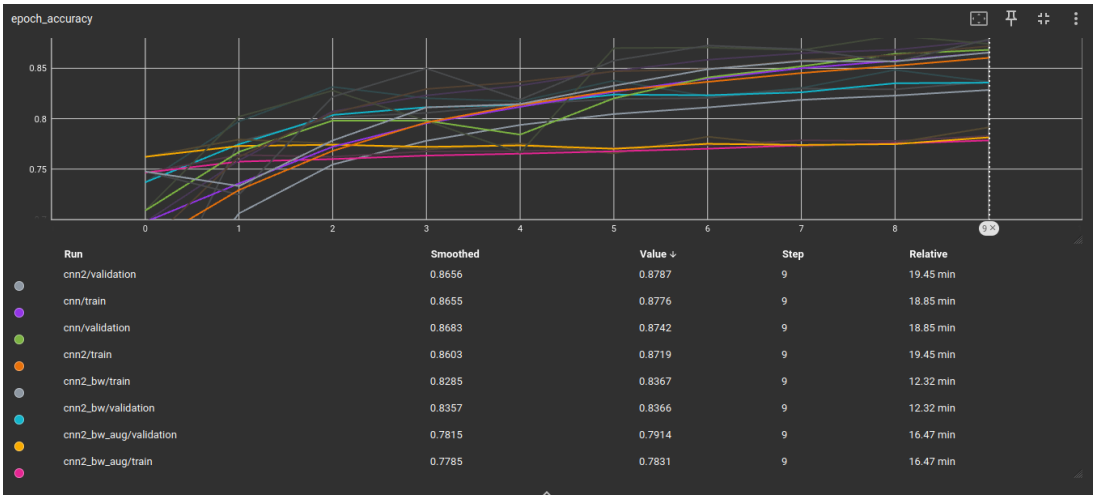

Figure 7.1: Gráfica general de los modelos respecto a su función de presicion

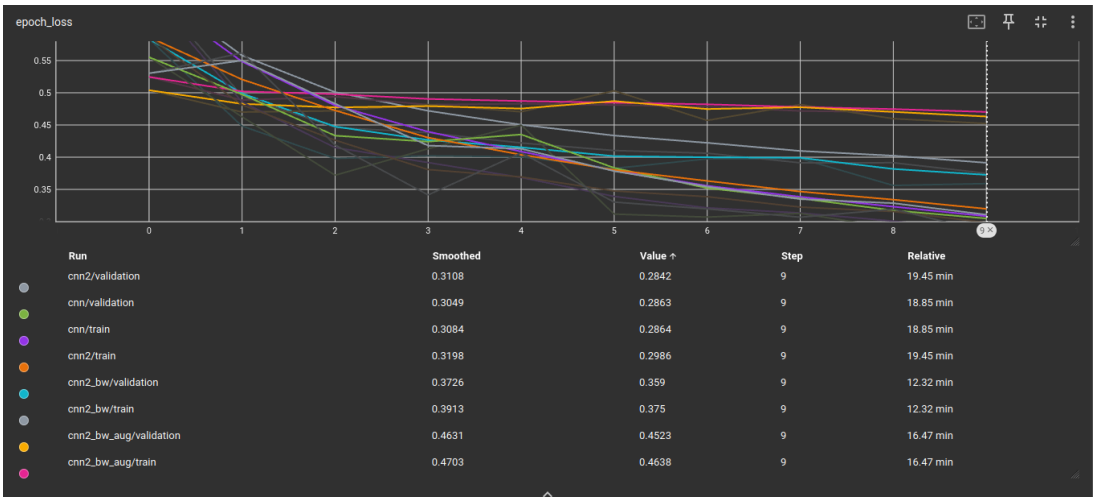

Figure 7.2: Gráfica general de los modelos respecto a su función de perdida