Trabajo Final: Aprendizaje Profundo para Análisis de Imágenes Biomédicas

Camilo Mariño

1 de abril de 2022

1. Introducción

1.1. Dataset

Para este trabajo se decidió utilizar el dataset *OrganSMNIST* [1], que cuenta con imágenes de varios órganos distintos con sus respectivas etiquetas. En particular, contiene 13940 imágenes de train, 2452 de validación y 8829 de test. En el Cuadro 1 se puede observar las cantidades de imágenes que se tiene de cada órgano.

Organo	**#Train**	**#Val**	**#Test**
Liver	3464	491	2078
Pancreas	2004	280	1343
Spleen	1556	213	968
Bladder	1148	188	811
Kidney-left	1132	140	704
Kidney-right	1119	159	693
Lung-right	803	275	439
Lung-left	741	261	397
Heart	721	246	510
Femur-left	637	104	439
Femur-right	615	95	447

Cuadro 1: Cantidad de imágenes de cada órgano en cada conjunto de datos.

Para mayor familiaridad con el dataset se visualizaron ejemplos de algunas de las imágenes que posee. En la Figura 1 se pueden observar las mismas.

1.2. Tarea a resolver

La tarea a resolver en este trabajo fue implementar modelos de Deep Learning, capaces de dada una imagen de un órgano, clasificarla en uno de los 11 órganos presentes en el Cuadro 1.

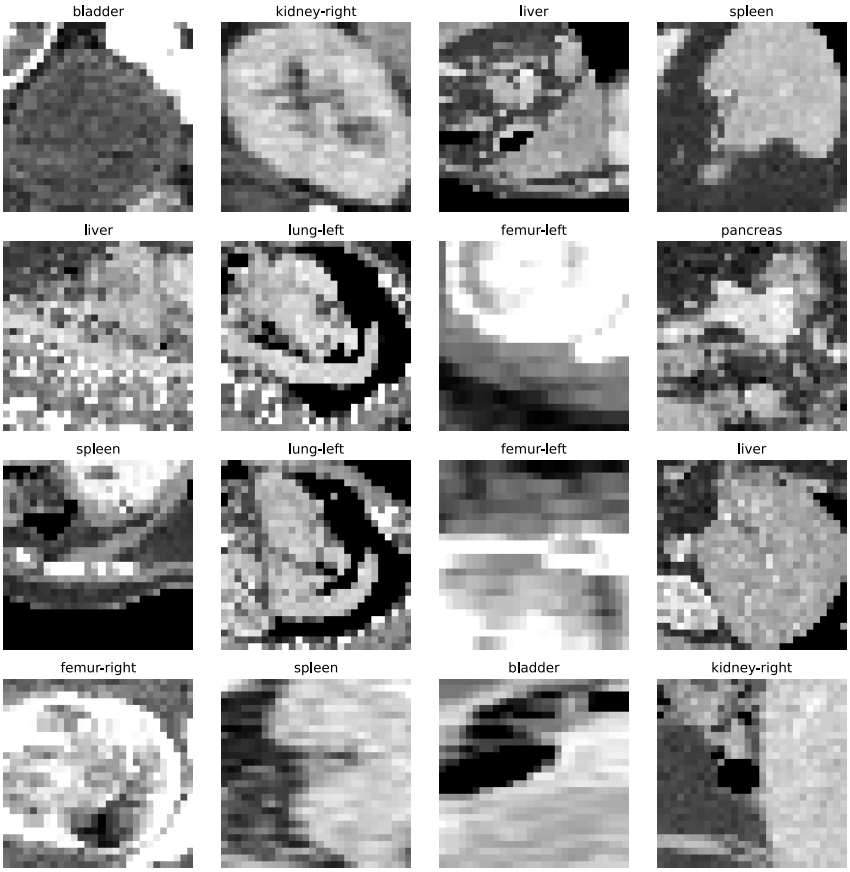

Figura 1: Ejemplos de imágenes del dataset OrganSMNIST.

1.3. Métricas de desempeño

Al tratarse de un problema de clasificación multi-clase, se utilizaron las métricas usuales, que son *Accuracy*, *Weighted-Precision*, *Weighted-Recall*, *Weighted-F1-score*. Que sean métricas *Weighted* representa que se calcula la métrica para cada clase y luego se la pesa según la porción que representa esa clase en el conjunto total de datos.

La definición precisa de cada una de las métricas es la siguiente:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Weighted - Precision = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \frac{TP_l}{TP_l+FP_l}$
- $Weighted - Recall = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \frac{TP_l}{TP_l+FN_l}$

- $Weighted - F1 - Score = \frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \frac{TP_l}{TP_l + \frac{1}{2}(FP_l + FN_l)}$

Donde:

- TP: Verdaderos positivos
- TN: Verdaderos negativos
- FP: Falsos positivos
- FN: Falsos negativos

El subíndice l, en TP_l, TN_l, FP_l, FN_l, representa la métrica calculada solo para la clase l. Mientras que L es el conjunto de todas las clases.

1.4. Modelo utilizado

La etapa de clasificación consistió en utilizar la misma arquitectura que la de la Tarea 3 del curso[1]. Esta arquitectura se puede observar en la Figura 2, cuenta exactamente con las siguientes características:

- Capa convolucional (6 feature maps de salida) + ReLU
- Max pooling 2 x 2
- Capa convolucional (16 feature maps de salida) + ReLU
- Max pooling 2 x 2
- Capa totalmente conectada (120 neuronas) + ReLU
- Capa totalmente conectada (84 neuronas) + ReLU
- Capa de salida (11 neuronas)

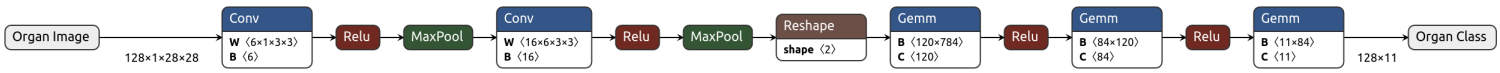

Figura 2: Esquema de la arquitectura base.

Este modelo cuenta con un total del 106,239 parámetros entrenables.

La única diferencia respecto al modelo de la tarea 3 es que se modificó a 11 la cantidad de neuronas de salida, ya que esta es la cantidad de clases que tiene el dataset *OrganSMNIST*.

Lo que sigue de este trabajo consistió en aplicarle mejoras a la arquitectura anteriormente mencionada, evaluarla y reportar sus resultados.

[1] `https://github.com/camilomarino/DLBioIm/blob/main/trabajos_practicos/soluciones/Practica_3_Camilo.ipynb`

2. Experimentos

En esta sección se irán mostrando los resultados al ir introduciendo distintas mejoras a la arquitectura base, descrita en la Sección 1.4.

Para que la comparación entre modelos sea justa, todas las redes fueron inicializadas aleatoriamente con los mismos pesos, de forma de evitar que la inicialización azarosa implicara algún beneficio para alguna de las mejoras.

Además, cada mejora incluye todas las anteriores, es decir, la última mejora incluye desde la primera hasta ella misma.

Las mejoras introducidas fueron:

- *Early stopping* (detención temprana).
- Aumentar cantidad de parámetros del modelo.
- *Data augmentation* (aumentado de datos).
- *Batch Normalization* (normalización por lotes).
- *Ensembles* (combinar modelos) con Transfer Learning (transferencia de aprendizaje).

Todos los experimentos compartieron una serie de hiper-parámetros y decisiones de diseño comunes. Estos se muestran en el Cuadro 2.

Optimizador	Adam
Learning rate	1×10^{-3}
Batch size	128
Loss function	Cross Entropy Loss

Cuadro 2: Hiper-parámetros y decisiones de diseño comunes para todos los modelos implementados.

A continuación, se irán mostrando los resultados desde la arquitectura base hasta llegar a combinar modelos mediante *ensembles*.

2.1. Arquitectura base

Se evaluó la arquitectura de la Figura 2 sin modificaciones. Se entrenó el modelo por 5 épocas.

La curva de loss obtenida y la evolución de la accuracy sobre train y validación durante el entrenamiento se observa en la Figura 3. Se aprecia que el algoritmo "aprende", incluso se tiene mejor accuracy sobre el conjunto de validación que sobre train.

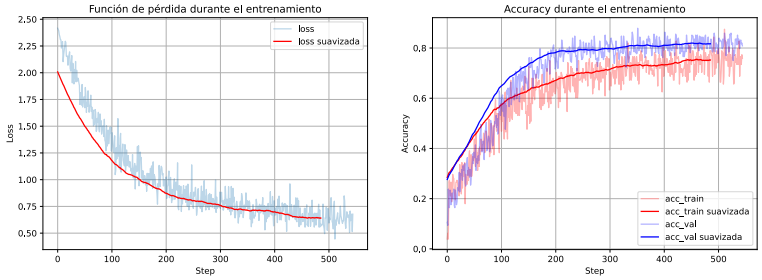

Figura 3: Curva de evolución de la loss y acc durante el entrenamiento.

En la Figura 4 se observa la matriz de confusión para la arquitectura base sobre train y validación. Se aprecia que en ambos conjuntos el desempeño es adecuado, la mayor cantidad de valores se encuentran sobre la diagonal de la matriz (lo cual indica una correcta predicción).

Uno de los patrones que aparece en la matriz de confusión es que la mayor parte de elementos que están fuera de la diagonal se encuentran en las últimas dos filas, es decir, para las imágenes que pertenecen a *Pancreas* y *Spleen* el modelo predice clases que en varias ocasiones son incorrectas. Sin embargo, notar que estos son dos de los tres órganos de los que más imágenes tiene el dataset, así que es de esperar que las clases que más elementos tengan, también sean las que más elementos mal clasificados posean.

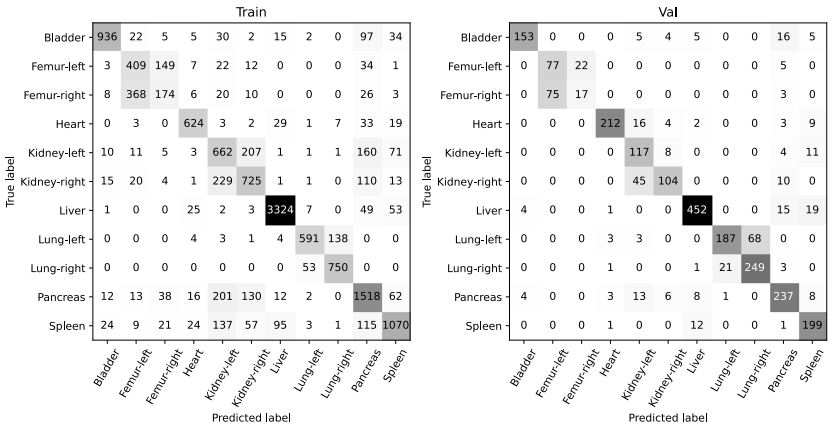

Figura 4: Matriz de confusión para la arquitectura base.

Por último, se reportan los resultados numéricos para las métricas en el Cuadro 3.

	Accuracy	F1-score	Precision	Recall
Train	0.774	0.774	0.779	0.774
Val	0.817	0.814	0.826	0.817

Cuadro 3: Métricas de desempeño sobre train y validación para la arquitectura base.

2.2. Mejora 1: Early Stopping

Motivación

En la etapa anterior se entrenó un modelo por una cantidad fija de épocas. La elección de que fuera por 5 épocas fue completamente arbitraria. El método de *Early Stopping* (detención temprana en español) establece una forma automática por cuantas épocas extender el entrenamiento.

Este método consiste fijar una métrica de interés e ir monitorizando su valor sobre el conjunto de validación a lo largo del entrenamiento. Es de esperar, que a lo largo del entrenamiento de una red neuronal, las métricas relevantes del problema comiencen a mejorar en los conjuntos de train y validación a medida que avanzan las épocas, llegando a determinado punto a partir del cual se comienza a producir *over fitting*, comenzando a empeorar la métrica en validación (pero continuando la mejora sobre train), lo cual no es deseado. *Early Stopping* consiste en frenar el entrenamiento luego de que la métrica de interés comienza a empeorar. El proceso anterior se encuentra ilustrado en la Figura 5.

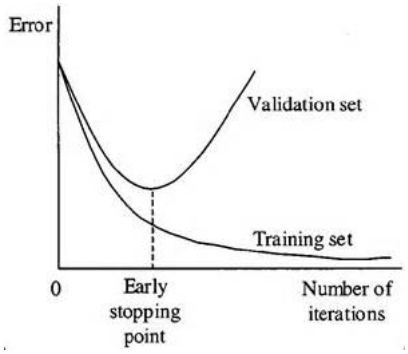

Figura 5: Ilustración de cuando se detiene el entrenamiento con Early Stopping.

Modificación

Se implementó el algoritmo siguiendo los pasos que se encuentran en la Figura 6. La métrica objetivo fue la ***accuracy* de validación**. Se utilizó paciencia de 5 épocas.

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Figura 6: Algoritmo *early stopping* extraído del libro *Deep Learning* [2] en su sección 7.8.

Resultados

En la Figura 7 se ven las curvas de loss y la evolución de la accuracy sobre train y validación durante el entrenamiento. Además, en la Figura 8 se muestra la matriz de confusión.

A simple vista se aprecia que el entrenamiento fue más extenso y que accuracy de validación alcanza valores más elevados que sin incluir *early stopping*. Además, en la matriz de confusión se observa que los valores fuera de la diagonal en las últimas dos filas disminuyeron.

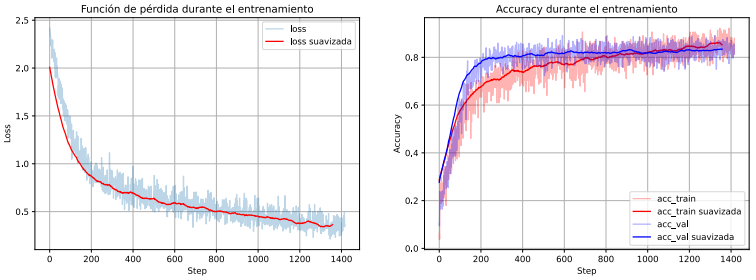

Figura 7: Curva de evolución de la loss y acc durante el entrenamiento.

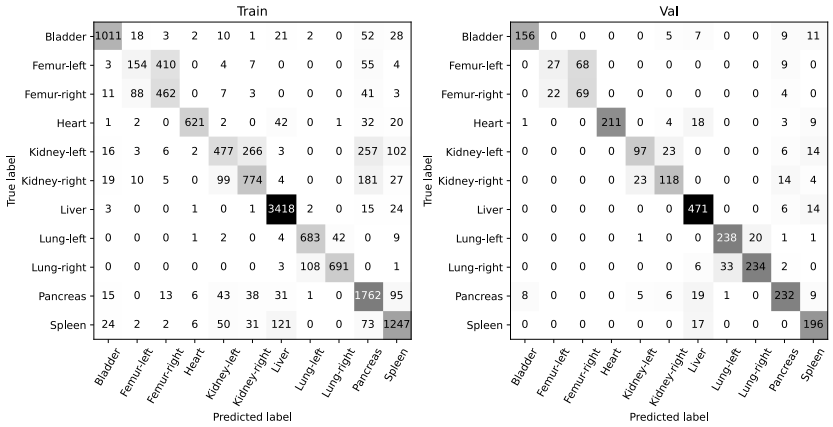

Figura 8: Matriz de confusión para arquitectura base.

Numéricamente, los resultados que se tienen son los del Cuadro 4. Se logra un incremento de 1,9 % en la *accuracy* sobre el conjunto de validación al incluir esta mejora.

	Accuracy	**F1-score**	**Precision**	**Recall**
Train	0.811	0.802	0.810	0.811
Val	0.836	0.832	0.839	0.836

Cuadro 4: Métricas de desempeño sobre train y validación para la arquitectura con *early stopping*.

2.3. Mejora 2: Aumentar cantidad de parámetros

Motivación

Cuando se implementa una arquitectura de red neuronal es difícil saber a simple vista si la cantidad de canales, neuronas, tamaño de filtros, *strides*, etc. son los indicados para el problema en cuestión. Variar estos hiper-parámetros conllevan en modificar la función paramétrica que representa la red neuronal, por tanto, también el rendimiento en clasificación.

Se debe tener en cuenta que los modelos con muchos parámetros son propensos al *over fitting*, ajustándose exclusivamente al conjunto de *train*. Por otro lado, si se tiene un modelo con pocos parámetros, este es propenso a que se produzca *under fitting*, es decir, que el modelo no sea capaz de aprender debido a su poca capacidad de representar a la función de clasificación.

Para encontrar el punto justo sin entrar en *over fitting* ni *under fitting* se deben probar distintas combinaciones de hiper-parámetros.

Modificación

En el caso particular de este trabajo, mediante prueba y error, se encontró el mejor rendimiento al cuadriplicar la cantidad de canales (en capas covolucionales) y neuronas (en capas completamente conectadas), y al modificar el *kernel size* a 5 y el *stride* a 2.

En resumen, la nueva arquitectura es la siguiente:

- Capa convolucional (24 feature maps de salida) + ReLU
- Max pooling 2 x 2
- Capa convolucional (64 feature maps de salida) + ReLU
- Max pooling 2 x 2
- Capa totalmente conectada (480 neuronas) + ReLU
- Capa totalmente conectada (240 neuronas) + ReLU
- Capa de salida (11 neuronas)

En la Figura 9 se puede ver una representación mediante bloques de esta nueva arquitectura.

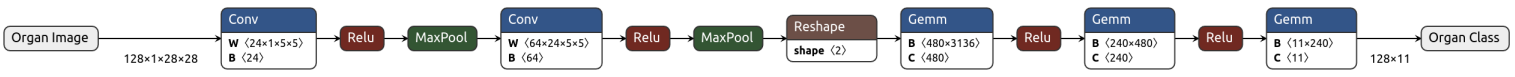

Figura 9: Esquema de la arquitectura con más parámetros.

Este nuevo modelo cuenta con una gran cantidad de parámetros entrenables más que el anterior, en total $1,662,939$.

Resultados

En la Figura 10 se ven las curvas de loss y la evolución de la accuracy sobre train y validación durante el entrenamiento. Además, en la Figura 11 se muestra la matriz de confusión.

Las curvas de evolución de la *accuracy* son un poco diferente a las de experimentos anteriores, en este caso el desempeño es sustancialmente superior en train que en validación. Esto podría hacer pensar que se esté dando *over fitting*, pero no es del todo cierto, ya que este incremento en la cantidad de parámetros logro mejores resultados sobre validación respecto a la mejora previa. De todas formas, se buscó disminuir este GAP en la mejora posterior mediante una técnica de regularización.

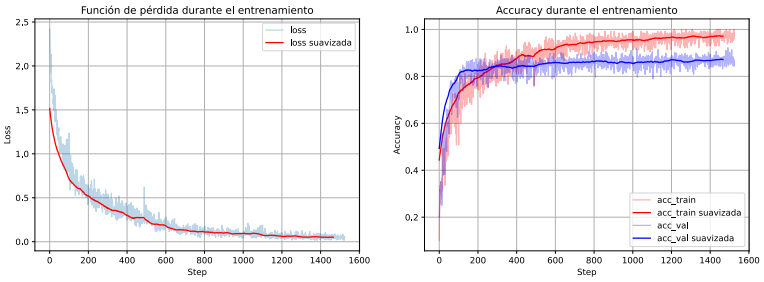

Figura 10: Curva de evolución de la loss y acc durante el entrenamiento.

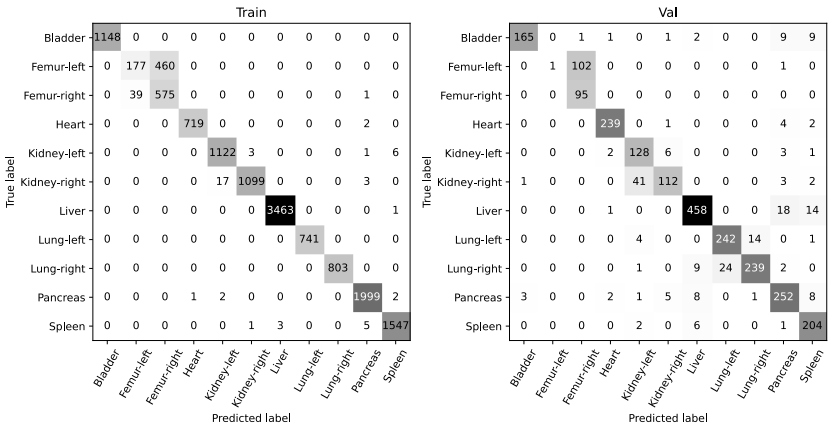

Figura 11: Matriz de confusión para arquitectura con más parámetros.

Los resultados numéricos son los del Cuadro 5. Se mejoró la *accuracy* sobre el conjunto de validación en 3,5 %.

	Accuracy	F1-score	Precision	Recall
Train	0.961	0.956	0.969	0.961
Val	0.871	0.858	0.897	0.871

Cuadro 5: Métricas de desempeño sobre train y validación para la arquitectura con mayor cantidad de parámetros.

2.4. Mejora 3: Data Augmentation

Motivación

Las redes neuronales actuales poseen una cantidad enorme de parámetros y las que utilice en las secciones anteriores no son la excepción. La forma de que un algoritmo tenga muchos parámetros y no se produzca *over fitting* es utilizar gran cantidad de datos de entrenamiento, pero muchas veces esto no es posible debido a la escaces de los mismos.

Una forma de lidiar con esto es generar nuevos datos de forma sintética a partir de perturbaciones de los datos originales.

Modificación

Se propusieron 3 variantes de aumentado de datos, extraídas todas de la biblioteca *torchvision*. Estas fueron:

- Random Rotation: Rotar cada imagen con probabilidad 0,5 un ángulo aleatorio no mayor a 30 grados.
- Random erasing [3]: Consiste en borrar un rectángulo aleatorio de la imagen (ver Figura 12). Esta transformación genera que la red neuronal aprenda extraer información de todas las regiones de las imágenes, ya que se lo fuerza a clasificar no conociendo la totalidad de la imagen.
- Transformación perspectiva aleatoria: Aplica con probabilidad 0,5 una transformación perspectiva aleatoria.

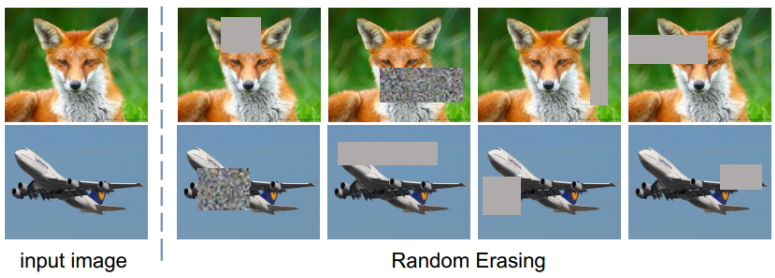

Figura 12: Ejemplo de *Random Eransing*. Imagen extraída de [3].

Resultados

Se presentan las mismas figuras que para los modelos anteriores, en la Figura 13 se ven las curvas de loss y la evolución de la accuracy sobre train y validación durante el entrenamiento. Mientras que en la Figura 14 se muestran las matrices de confusión.

En las Figura de la evolución de la *accuracy* se aprecia que se empeora el desempeño en train, pero sin embargo se mejora los valores sobre validación. Es decir, la generación sintética de datos regulariza el modelo y elimina la brecha de desempeño entre los datos de entrenamiento y validación.

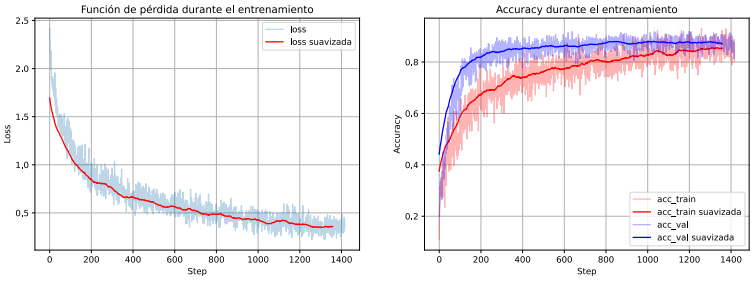

Figura 13: Curva de evolución de la loss y acc durante el entrenamiento.

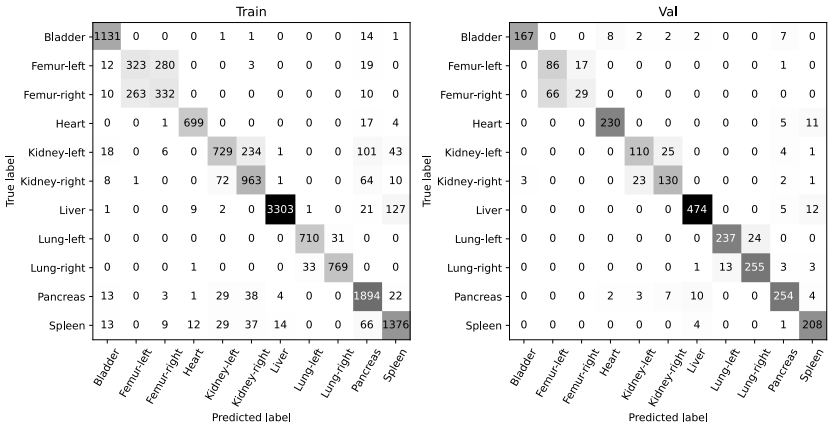

Figura 14: Matriz de confusión para arquitectura con *data augmentation*.

A continuación, en el Cuadro 6 se muestran los resultados de las distintas métricas. En este se verifica que empeoró el rendimiento en train, pero se mejoró 1,8 % la *accuracy* en validación.

	Accuracy	F1-score	Precision	Recall
Train	0.877	0.876	0.878	0.877
Val	0.889	0.887	0.892	0.889

Cuadro 6: Métricas de desempeño sobre train y validación para la arquitectura que incluye data augmentation.

2.5. Mejora 4: Batch Normalization

Motivación

Batch Normalization [4] es un método utilizado para hacer que las redes neuronales se entrenen más rápido y sean más estables, normalizando las entradas a las capas (usualmente previo a la función de activación), escalando los valores con su media y varianza.

Además, Batch Normalization tiene un efecto regularizador, ya que las medias y varianzas son calculadas en cada batch, lo cual introduce ruido al entrenamiento.

Modificación

Se aplico batch normalization luego de cada capa convolucional 2d y lineal, excepto en la última. Además, se elimina el termino de bias a las capas previas a batch normalization ya que es redundante.

Las características de la nueva arquitectura son:

- Capa convolucional (24 feature maps de salida) + Batch Normalization 2d + ReLU
- Max pooling 2 x 2
- Capa convolucional (64 feature maps de salida) + Batch Normalization 2d + ReLU
- Max pooling 2 x 2
- Capa totalmente conectada (480 neuronas) + Batch Normalization 1d + ReLU
- Capa totalmente conectada (240 neuronas) + Batch Normalization 1d + ReLU
- Capa de salida (11 neuronas)

Batch normalization agrega más parámetros entrenebales, totalizando esta nueva arquitectura $1,664,555$.

Resultados

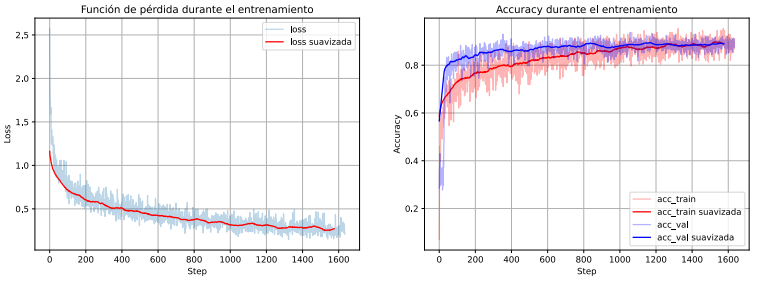

Figura 15: Curva de evolución de la loss y acc durante el entrenamiento.

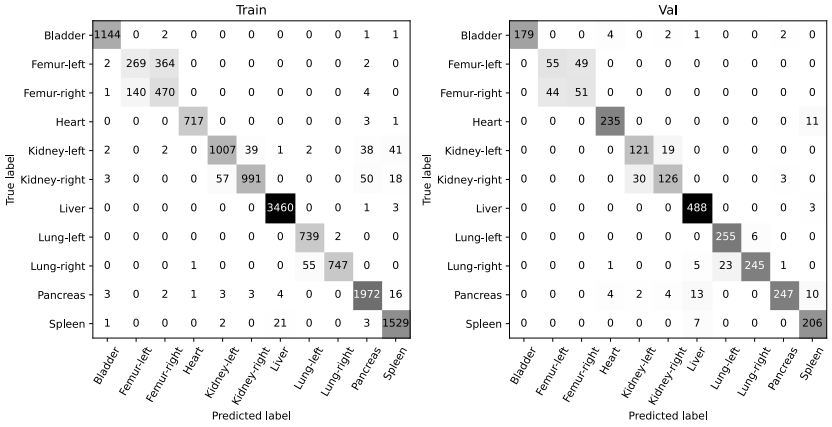

Figura 16: Matriz de confusión para arquitectura con *batch normalization*.

	Accuracy	F1-score	Precision	Recall
Train	0.936	0.934	0.938	0.936
Val	0.900	0.900	0.902	0.900

Cuadro 7: Métricas de desempeño sobre train y validación para arquitectura con *batch normalization*.

2.6. Mejora 5: Ensembles con Transfer Learning

Motivación

En los modelos de Machine Learning existe el concepto de varianza de un modelo, que representa el error frente a pequeñas modificaciones en los conjuntos

de entrenamiento y en las formas de entrenarlo.

Una manera usual de reducir esta varianza es entrenar varios modelos en simultáneo, ya sea el mismo con pequeñas modificaciones o varios distintos, y luego combinar sus resultados de alguna forma y así reducir la varianza de las predicciones.

Por otra parte, un modo de mejorar desempeños es utilizar conjuntos de datos en dominios similares, e intentar trasladar lo aprendido al dominio de interés. Una práctica habitual es entrenar arquitecturas de redes neuronales por una gran cantidad de épocas para tareas de clasificación sobre grandes conjuntos de datos etiquetados; y luego modificar la última capa de la red, adaptándola a la cantidad de clases del problema de interés y entrenar por unas pocas épocas sobre el conjunto de datos propio. El procedimiento anterior es conocido como *Transfer Learning*.

Modificación

Se entrenaron tres variantes de redes neuronales de la clásica arquitectura ResNet [5]. La particularidad de esta arquitectura es que cuenta con conexiones residuales, como las de la Figura 17, que fueron las encargadas, en el momento que fueron publicadas, de darles un salto de rendimiento respecto a las arquitecturas competidoras.

Las variantes utilizadas fueron ResNet-50, ResNet-101 y ResNet-152. Todas estas fueron entrenadas con los hipe-parámetros del Cuadro 2, con *data augmentation* y *early stopping*. Además, se aplicó transfer learning, ya que se partió del modelo pre-entrenado en ImageNet [6] presente en la biblioteca *torchvision*.

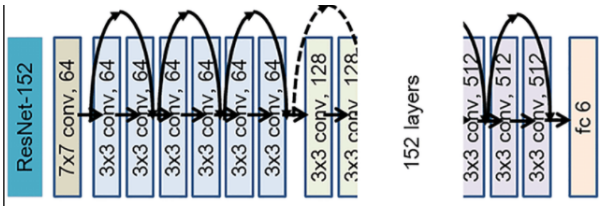

Figura 17: Arquitectura Resnet-152.

Se combinaron estas tres variantes de ResNet junto con la arquitectura propia, de esta forma se tienen 4 modelos independientes de clasificación sobre el dataset *OrganSMNIST* con el objetivo de reducir la varianza. Elmodo de combinarlos fue promediar las salidas de cada red neuronal, que no es más que promediar las probabilidades que asigna cada red de pertenecer a cada una de las clases.

Resultados

Las tres arquitecturas de ResNet obtuvieron las accuracys del Cuadro 8. Se obtienen valores en torno al $90\,\% - 91\,\%$, cercanos a los obtenidos con el modelo

implementado previamente.

	Accuracy Val
ResNet-50	0.912
ResNet-101	0.897
Resnet-152	0.892

Cuadro 8: Accuracy sobre el conjunto de validación para tres variantes de ResNets.

Al proceder a combinar los 4 modelos, la matriz de confusión que se tiene es la de la Figura 18. La misma es la matriz de confusión de este trabajo que posee la que mayor cantidad de elementos sobre la diagonal, para el *Pancreas* y *Spleen* hay muy pocas clasificaciones erróneas comparadas con las del modelo base.

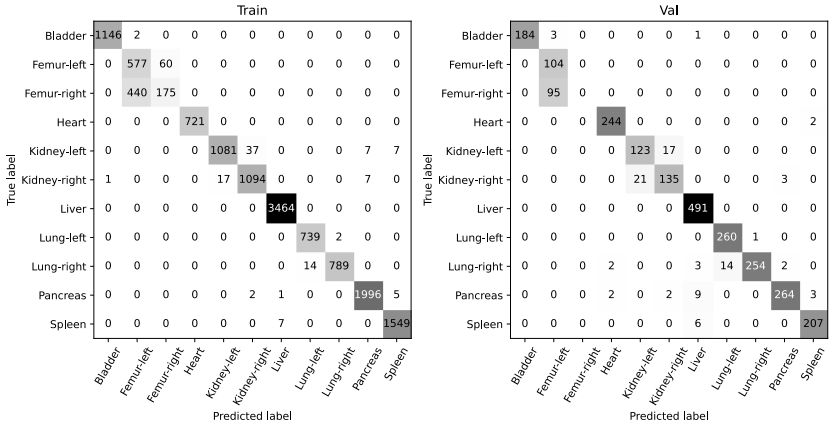

Figura 18: Matriz de confusión para arquitectura compuesta por *ensembles*.

Los resultados numéricos del *ensemble* son los del Cuadro 9, en el cual se aprecia que se mejora la accuracy hasta 91,8 % sobre el conjunto de validación, siendo este un incremento de 1,5 % respecto a la mejora anterior. Esto demuestra que combinar modelos no "tan buenos" genera un modelo "mejor".

	Accuracy	**F1-score**	**Precision**	**Recall**
Train	0.956	0.952	0.961	0.956
Val	0.924	0.911	0.905	0.924

Cuadro 9: Métricas de desempeño sobre train y validación para *ensembles*.

3. Comparación

Hasta el momento se han ido mostrando los resultados al ir introduciendo mejoras una a una. Para una comparación más sencilla en el Cuadro 10 se presenta los resultados para todas las variantes, incluyendo los resultados sobre test que hasta ahora no habían sido mostrados ni utilizados.

Para los conjuntos de validación y test ocurrió que se van incrementaron todas las métricas al introducir cada una de las mejoras. Sin embargo, no ocurrió lo mismo para el conjunto de train, en particular al introducir *data augmentation* empeoraron las métricas, por lo que esta mejora actúa como un regularizador.

	Train				**Val**				**Test**			
	acc	**pre**	**recall**	**f1**	**acc**	**pre**	**recall**	**f1**	**acc**	**pre**	**recall**	**f1**
Base	0.774	0.779	0.774	0.774	0.817	0.826	0.817	0.814	0.680	0.687	0.680	0.679
Early Stopping	0.811	0.810	0.811	0.802	0.836	0.839	0.836	0.832	0.685	0.678	0.685	0.671
Big	0.961	0.969	0.961	0.956	0.871	0.897	0.871	0.858	0.731	0.728	0.731	0.724
Data Aug.	0.877	0.878	0.877	0.876	0.889	0.892	0.889	0.887	0.738	0.738	0.738	0.736
Batch Norm.	0.936	0.938	0.936	0.934	0.900	0.902	0.900	0.900	0.765	0.764	0.765	0.758
Ensemble	0.956	0.961	0.956	0.952	0.924	0.905	0.924	0.911	0.811	0.810	0.811	0.808

Cuadro 10: Comparación de todas arquitecturas con variantes, ordenadas de arriba a abajo desde cual se aplico en primer lugar.

Para finalizar con el análisis comparativo, en la Figura 19 se muestra la evolución de la *accuracy* al introducir cada una de las mejoras en los 3 conjuntos de datos. Se observa que existe un salto de desempeño entre los conjuntos de train-validación respecto al de test, lo cual es normal que suceda debido a que estos dos primeros conjuntos fueron los usados para el entrenamiento, es decir, es inevitable que se tenga un poco de *over fitting*. En particular, el conjunto de train fue usado para el entrenamiento de las redes, mientras que el de validación se utilizó para análisis exploratorio de hiperparametros y que mejoras introducir, además de estar presente en el algoritmo de *early stopping*.

Por último mencionar, que el resultado inicial sobre test si no se hubieran incluido mejoras era una accuracy de 68,0 %, alcanzando un gran incremento hasta 81,3 % luego de todas las mejoras (aumento neto de 13,3 %).

Además, mencionar que en la página `https://medmnist.com/` se presenta un benchmark sobre este dataset en el que se menciona que el mejor modelo es *AutoKeras* [7] con un rendimiento idéntico de 81,3 %, con lo que se puede afirmar que se alcanzaron rendimientos del estado del arte. Es importante decir que la comparación no es del todo justa, ya que en el benchmark no se implementan ensembles de modelos, lo cual previsiblemente aumentaría la accuracy.

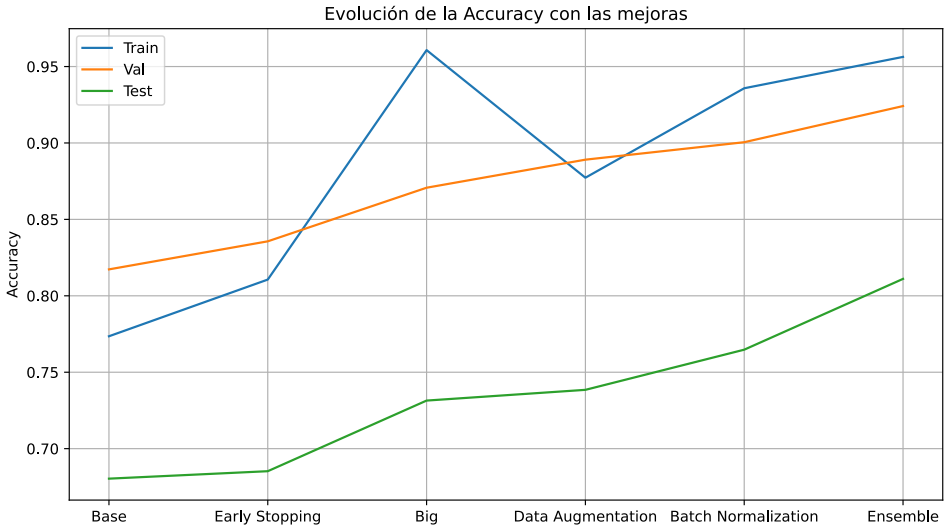

Figura 19: Evolución de la accuracy al ir introduciendo mejoras.

4. Implementación

La implementación se realizó basándose fuertemente en el *Jupyter Notebook* de la tarea 3. El código implementado de este trabajo y sus respectivos resultados se puede ver en `https://github.com/camilomarino/DLBioIm/blob/main/trabajo_final/TrabajoFinal_Camilo.ipynb`

.

El entrenamiento de las redes neuronales se realizó en equipos del Cluster-UY [8], que cuenta con GPUs Tesla P100.

Referencias

[1] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification, 2021.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[7] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[8] Sergio Nesmachnow and Santiago Iturriaga. Cluster-uy: collaborative scientific high performance computing in uruguay. In *International Conference on Supercomputing in Mexico*, pages 188–202. Springer, 2019.