Detección de enfermedades oculares mediante clasificación de imágenes.

Cristina Arroyo
Enzo Andreetto

June 15, 2024 — Universidad de Buenos Aires

Descripción del problema

Objetivo

El objetivo de este proyecto es el reconocimiento inteligente de enfermedades oculares utilizando el dataset de la base de datos ODIR(Ocular Disease Recognition). Se proponen técnicas de clasificación para identificar las enfermedades oculares en una de las ocho categorías: Normal (N), Diabetes (D), Glaucoma (G), Cataratas (C), Degeneración Macular Relacionada con la Edad (A), Hipertensión (H), Miopía Patológica (M) y Otras enfermedades/anomalías (O).

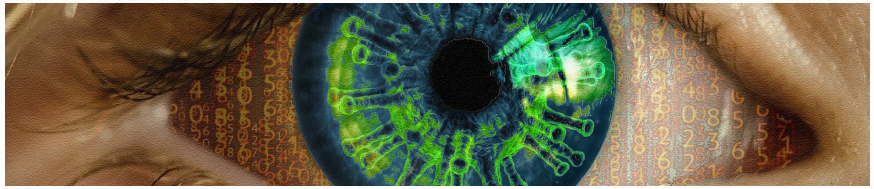
Problema

El problema que aborda este proyecto es la necesidad de una detección rápida y precisa de enfermedades oculares, lo que puede ser crucial para el diagnóstico temprano y el tratamiento efectivo.
Esta solución podría ser de gran ayuda en varios aspectos:

- - Diagnóstico temprano y tratamiento oportuno.
- - Reducción de errores humanos.
- - Escalabilidad y disponibilidad.
- - Utilización eficiente de recursos médicos.

Descripción del dataset

Ocular Disease Recognition



Imágenes de fondos de ojo

El dataset consiste en fotos de fondos de ojo tomadas de 5.000 pacientes reales y recopiladas de diferentes hospitales y centros médicos de China.
Utilizamos un conjunto de imágenes preprocesadas incluídas en el data set, de 512 x 512 pixels.
Constan de un total de 6.392 fotos, tanto de ojos izquierdos como derechos.

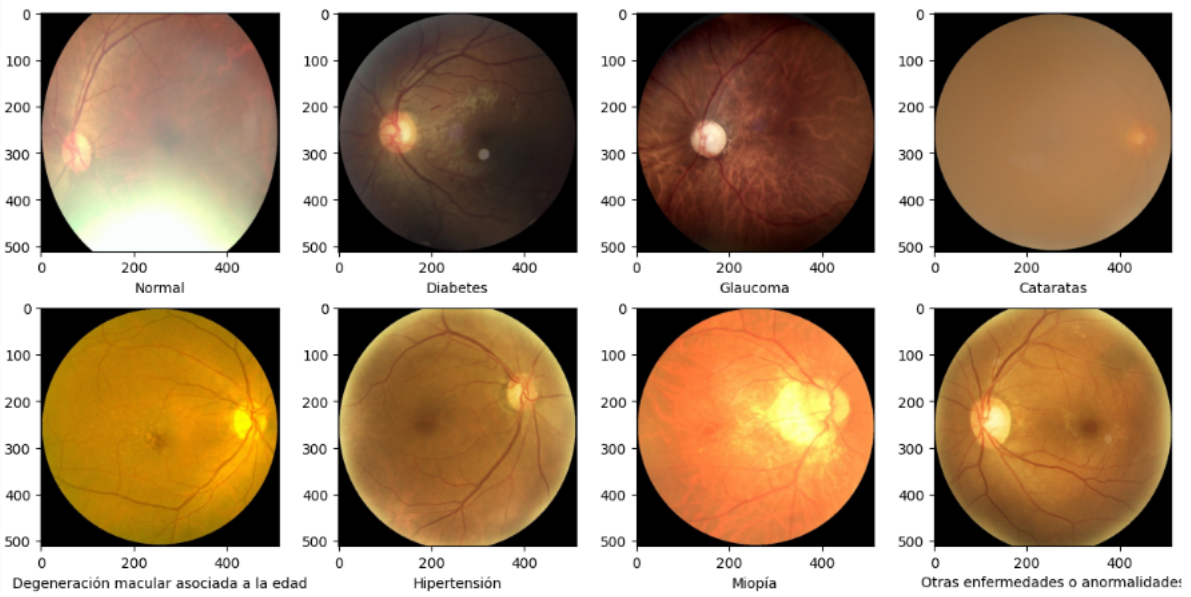
Datos tabulares

Además de las imágenes, el dataset consta de un .csv con datos tabulares. Entre otras features, cada fila asocia un archivo de imagen con su correspondiente target.

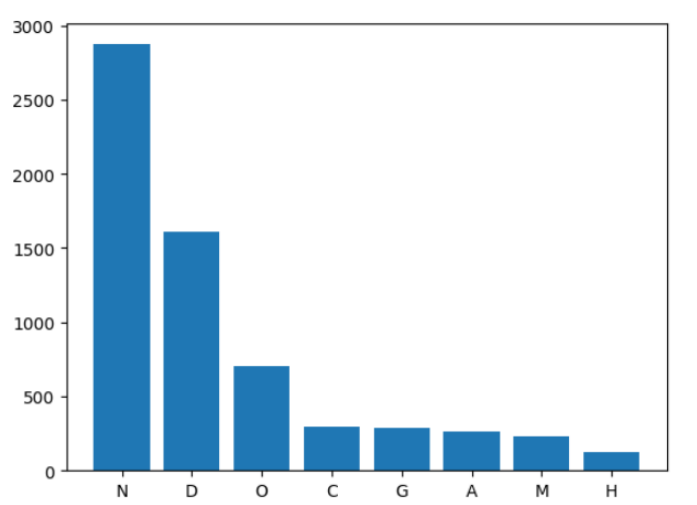
	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	labels	target	filename
0	0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0	0	0	1	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	0_right.jpg
1	1	57	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1_right.jpg

Hay 8 clases: Normal (N), Diabetes (D), Glaucoma (G), Cataratas (C), Degeneración macular asociada a la edad (D), Hipertensión (H), Miopía (M), Otras enfermedades o anormalidades (O).

Algunas imágenes y sus labels



Desbalance en las clases




```
N: 2873
D: 1608
O: 708
C: 293
G: 284
A: 266
M: 232
H: 128
```

ResNet

ResNet

Se tomó como premisa entrenar **ResNet18** y **ResNet50** usando transfer learning sobre ambos modelos preentrenados con el dataset de ImageNet, haciendo tanto **feature extraction** como **fine tuning**. Estos se tomaron como baseline para luego optimizar el rendimiento de uno de ellos.

- Split de datos 80%-20%
- Batch size de 32
- Cross entropy loss
- Adam optimizer con learning rate de 0.001
- 100 épocas con early stop
- Accuracy macro como métrica

ResNet

	ResNet18	ResNet50
Feature Extraction	• Épocas 9 • Val. loss 1.659 • Val. accuracy 0.160	• Épocas 39 • Val. loss 1.528 • Val. accuracy 0.175
Fine Tuning	• Épocas 35 • Val. loss 1.383 • Val. accuracy 0.270	• Épocas 13 • Val. loss 1.441 • Val. accuracy 0.232

ResNet50. Fine tuning, data augmentation y loss ponderada

Si bien ResNet18 con fine tuning dio mejor accuracy que ResNet50, optamos por mejorar el modelo que en principio puede ofrecer mejores métricas.

Además de fine tuning, se intentó

- Data augmentation: RandomHorizontalFlip, RandomVerticalFlip, RandomRotation, RandomResizedCrop
- Weighted loss
- Learning rate de 0.0001

ResNet50. Fine tuning, data augmentation y loss ponderada

Obteniendo los siguientes resultados

- Early stop en época 21
- Validation loss 1.778
- Validation accuracy 0.283

ResNet50. Fine tuning, data augmentation, loss ponderada y dropout por 130 épocas

Además de la configuración elegida en el modelo anterior, se agregó

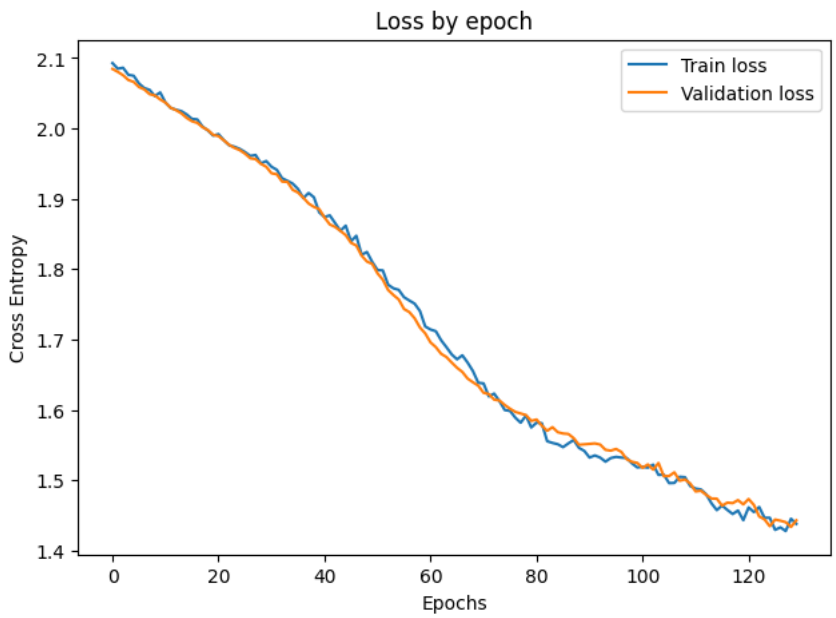
- Capa de dropout
- 130 épocas sin early stop
- Agregamos recall y precision como métricas
- El modelo quedó con 23.524.424 parámetros entrenables

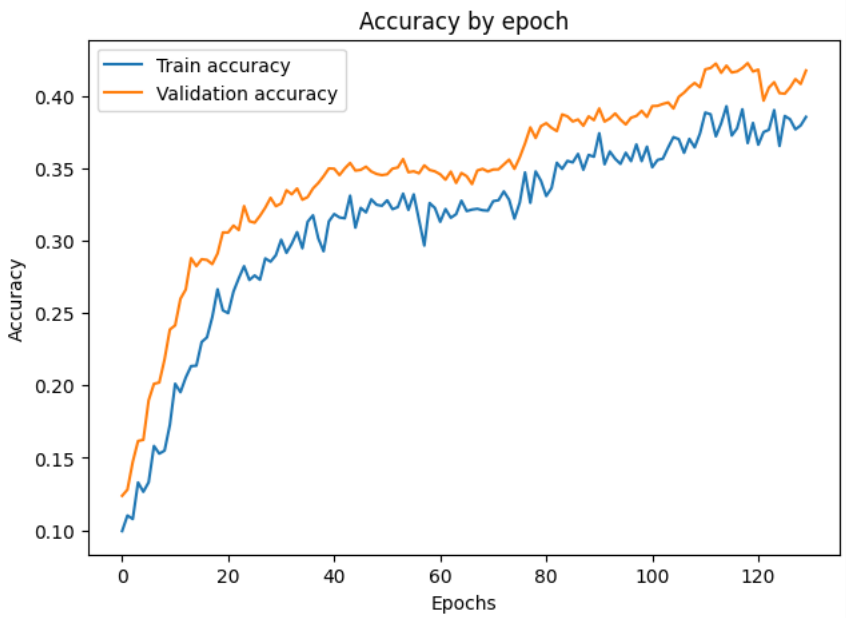
ResNet50. Fine tuning, data augmentation, loss ponderada y dropout por 130 épocas

Se obtuvieron las siguientes métricas

- Validation loss 1.444
- Validation accuracy 0.417
- Validation precision 0.367
- Validation recall 0.417

ResNet50. Fine tuning, data augmentation, loss ponderada y dropout por 130 épocas





ResNet18. Fine tuning, data augmentation, loss ponderada y dropout

Se buscó replicar la última configuración pero volviendo sobre ResNet18

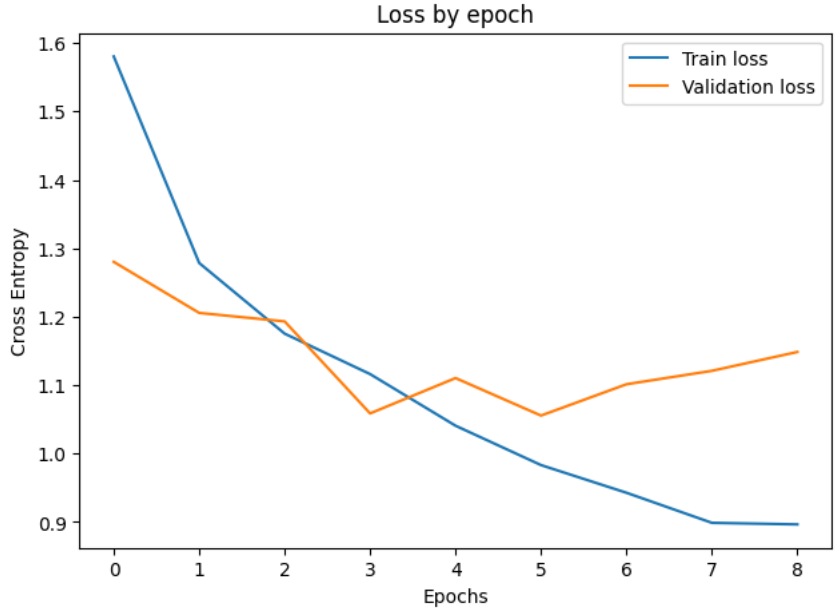
- Capa de dropout
- Early stop
- Accuracy, recall y precision como métricas
- El modelo quedó con 11.180.616 parámetros entrenables

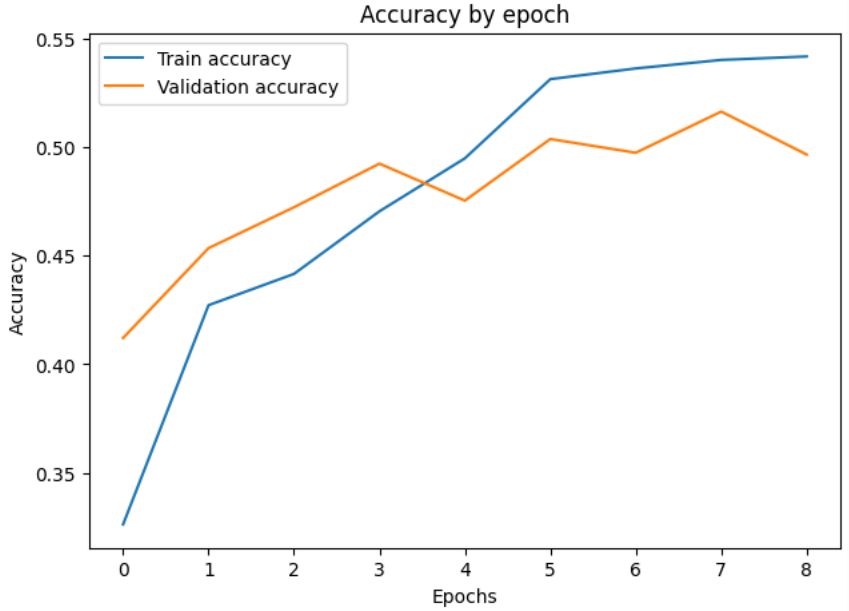
ResNet18. Fine tuning, data augmentation, loss ponderada y dropout

Se obtuvieron las siguientes métricas

- Early stop en época 9
- Validation loss 1.055
- Validation accuracy 0.503
- Validation precision 0.460
- Validation recall 0.503

ResNet18. Fine tuning, data augmentation, loss ponderada y dropout





VGGNet

VGGNet

Exploramos 2 estrategias de transfer learning: fine-tuning y feature extraction en las arquitecturas VGGNet16 y VGGNet19. Los parámetros de entrenamiento primarios son:

- Split de datos 80%-20%
- Batch size de 32
- Cross entropy loss
- Adam optimizer con learning rate de 0.001
- 100 épocas con early stop
- Accuracy, Precision y Recall como métricas

Usamos fine-tuning(FT) utilizando una fc con un clasificador lineal:

$$nn.Linear(in\_features = in\_features, out\_features = num\_classes)$$

VGGNet

Feature extraction la testeamos con dos enfoques distintos: El primero es utilizar una fc lineal similar a la que usamos con fine-tunning(FEL) y el segundo utiliza una fc con un clasificador de esta forma(FEC):

- $nn.Linear(in\_features = in\_features, out\_features = out1)$
- $nn.ReLU(True)$
- $nn.Dropout()$
- $nn.Linear(in\_features = out1, out\_features = out2)$
- $nn.ReLU(True)$
- $nn.Dropout()$
- $nn.Linear(in\_features = out2, out\_features = num\_classes)$

VGGNet16

	FTunning	FExtraction L	FExtaction C
Stop Epoch	16/100	100/100	41/100
Training Loss	1.5574	0.0000..	0.8137
Testing Loss	6.2296	0.0001	3.2547
Training Accuracy	0.1250	0.8694	0.4602
Testing Accuracy	0.1250	0.8685	0.4597
Training Precision	0.1409	0.9322	0.6595
Testing Precision	0.1409	0.9317	0.6589
Training Recall	0.1250	0.8694	0.4602
Testing Recall	0.1250	0.8685	0.4597

VGGNet19

	FTunning	FExtraction L	FExtaction C
Stop Epoch	14/100	100/100	41/100
Training Loss	1.5787	0.0000..	0.8281
Testing Loss	6.3146	0.0001	3.3125
Training Accuracy	0.1250	0.8664	0.4146
Testing Accuracy	0.1250	0.8655	0.4143
Training Precision	0.1145	0.9246	0.6413
Testing Precision	0.1145	0.9241	0.6410
Training Recall	0.1250	0.8664	0.4146
Testing Recall	0.1250	0.8655	0.4143

VGGNet

Se escogieron los tres mejores modelos comparando sus métricas:

- VGGNet16 con feature extraction y conjunto de capas fc.
- VGGNet19 con feature extraction y clasificador lineal
- VGGNet16 con feature extraction y clasificador lineal.

A cada una de estos se le aplicaron diferentes técnicas de optimización de acuerdo a los resultados obtenidos cada uno está entrenado con 100 epochs y early stop.

VGGNet16 con feature extraction y conjunto de capas fc

1. La primera variante fue agregar data augmentation con early stop de 100 épocas:
    - RandomCrop(224)
    - RandomHorizontalFlip()
    - RandomRotation(10)
    - ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)
2. La segunda variante además de data augmentation fue cambiar el learning rate de 0.001 a 0.0001.
3. La tercera variante fue disminuir el learning rate a 0.00001.

VGGNet16 con feature extraction y conjunto de capas fc variantes

	V1(lr=0.001)	V2(lr=0.0001)	V3(lr=0.00001)
Stop Epoch	43/100	28/100	48/100
Training Loss	0.7428	0.0415	0.1159
Testing Loss	2.9713	0.1662	0.4635
Training Accuracy	0.4579	0.7692	0.7744
Testing Accuracy	0.4575	0.7664	0.7729
Training Precision	0.6854	0.8740	0.9014
Testing Precision	0.6852	0.8728	0.8999
Training Recall	0.4579	0.7692	0.7744
Testing Recall	0.4575	0.7664	0.7729

VGGNet16 con feature extraction y conjunto de capas fc

A la última variante se le agregó la técnica de loss ponderada, tomando en cuenta los pesos como la inversa de la frecuencia:

Stop Epoch	58/100
Training Loss	0.0099
Testing Loss	0.0396
Training Accuracy	0.7937
Testing Accuracy	0.7923
Training Precision	0.9078
Testing Precision	0.9070
Training Recall	0.7937
Testing Recall	0.7923

VGGNet16 con feature extraction y conjunto de capas fc

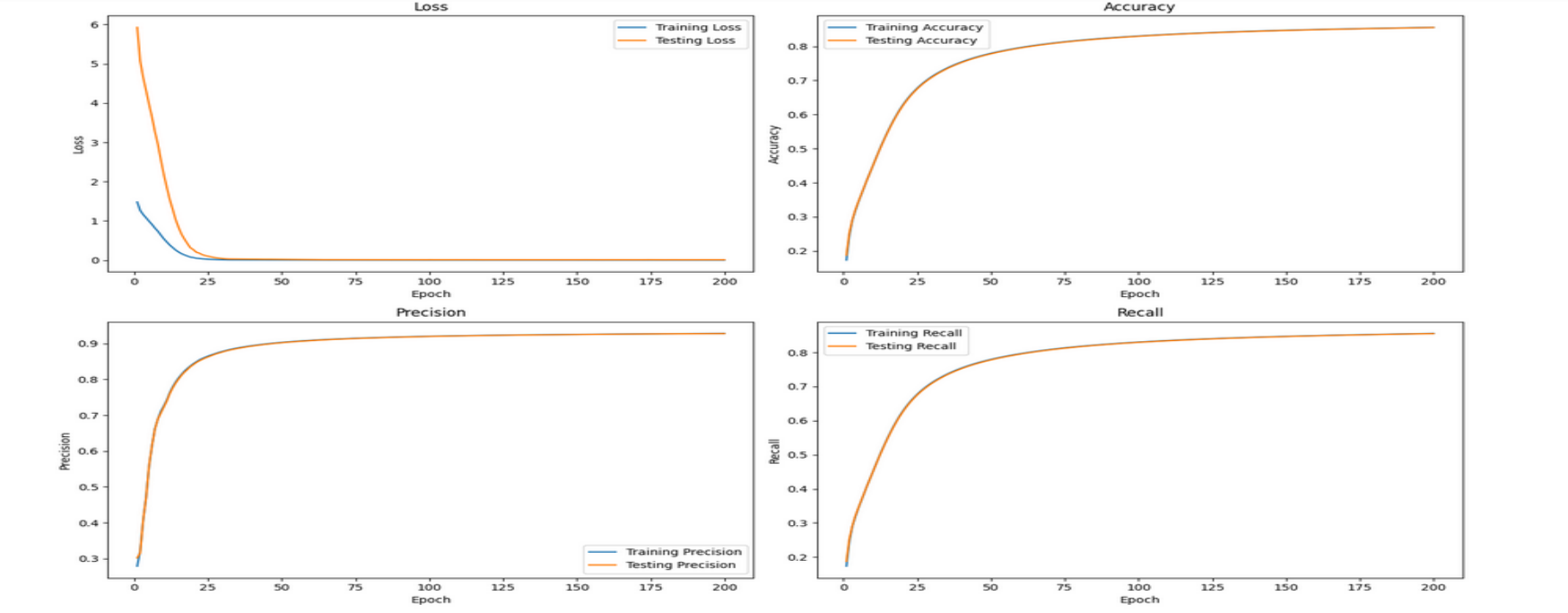
Debido a que se pudo observar que en conjunto la loss ponderada y la disminución de la learning rate mejoran las métricas se decidió agregar dos variantes más:

1. Learning rate variable inicial de 0.001, cada 30 épocas disminuye 0.1. 100 epochs sin early stop.
2. Learning rate variable inicial de 0.001, cada 30 épocas disminuye 0.1. 200 epochs sin early stop.

VGGNet16 con feature extraction y conjunto de capas fc entrenamientos finales

	100 epochs	200 epochs
Training Loss	0.0028	0.0027
Testing Loss	0.0110	0.0107
Training Accuracy	0.8294	0.8555
Testing Accuracy	0.8286	0.8550
Training Precision	0.9188	0.9280
Testing Precision	0.9182	0.9278
Training Recall	0.8294	0.8555
Testing Recall	0.8286	0.8550

VGGNet16 Conjunto de capas mejor modelo(200 épocas)



VGGNet16 con feature extraction y clasificador lineal

En este caso se hicieron dos variante más dado que se obtuvieron buenos resultados.

1. Data augmentation.
2. Disminución de la tasa de aprendizaje a 0.0001

VGGNet16 con feature extraction, data augmentation y clasificador lineal variantes

	V1(lr=0.001)	V2(lr=0.0001)
Stop Epoch	100/100	100/100
Training Loss	0.0000	0.0023
Testing Loss	0.0001	0.0091
Training Accuracy	0.8683	0.8457
Testing Accuracy	0.8673	0.8449
Training Precision	0.9373	0.9355
Testing Precision	0.9368	0.9350
Training Recall	0.8683	0.8457
Testing Recall	0.8673	0.8449

VGGNet16 con feature extraction y clasificador lineal con data augmentation

Se observa que en el caso del uso de un clasificador lineal, la disminución de la tasa de aprendizaje no representa una mejora significativa, incluso las métricas son menos óptimas al cabo de disminuye con el entrenamiento de 100 épocas. En los dos casos se llegó a las 100 épocas de entrenamiento lo que indica que después de las 100 épocas el modelo puede seguir aprendiendo por lo que la última variante planteada para este modelo es 200 epochs con early stop, con learning rate = 0.001, data augmentation y loss ponderada.

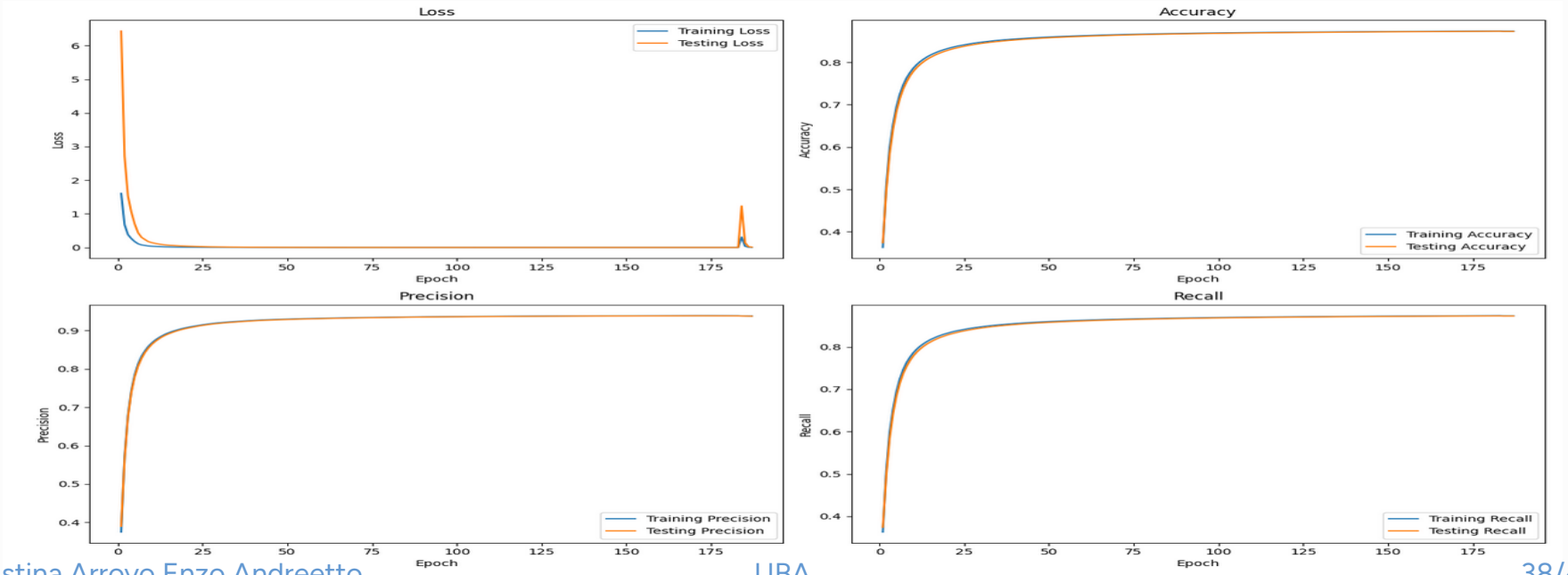
VGGNet19 con feature extraction y clasificador lineal con data augmentation

Dado el contexto de entrenamiento y dado que los entrenamientos son similares, se plantea un entrenamiento igual para el modelo correspondiente a VGGNet119 con 200 epochs con early stop, con learning rate = 0.001, data augmentation y loss ponderada a ser comparado con el modelo similar de VGGNet16

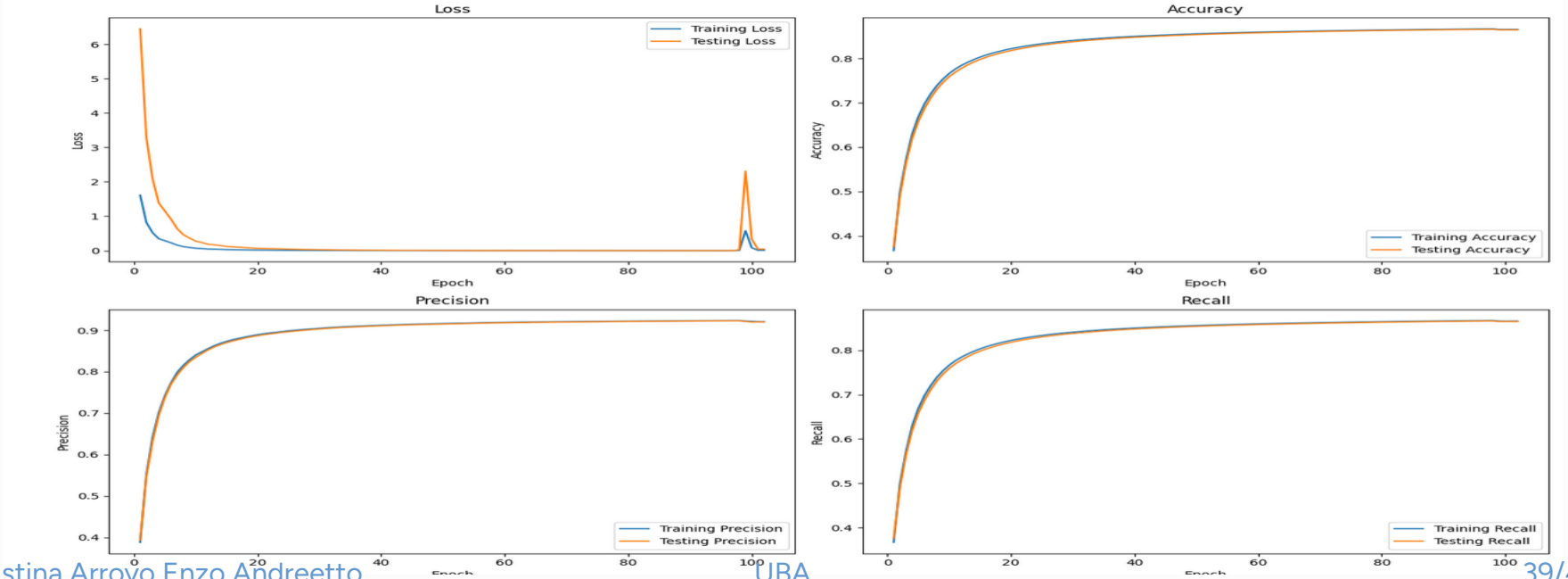
VGGNet16 y VGGNet19 con feature extraction y clasificador lineal con data augmentation, loss ponderada

	VGGNet16	VGGNet19
Stop Epoch	187/200	102/200
Training Loss	0.0024	0.0109
Testing Loss	0.0097	0.0435
Training Accuracy	0.8732	0.8656
Testing Accuracy	0.8727	0.8646
Training Precision	0.9381	0.9207
Testing Precision	0.9377	0.9199
Training Recall	0.8732	0.8656
Testing Recall	0.8727	0.8646

VGGNet16 clasificador lineal mejor modelo(200 épocas early stop)



VGGNet19 clasificador lineal mejor modelo(200 épocas early stop)



Conclusiones

ResNet

- Tanto en ResNet18 como en ResNet50, fine tuning funcionó mejor
- Al obtener accuracies bajas en pruebas iniciales, se pensó en apostar por ResNet50 que podría ofrecer mejores métricas, aunque a costa de requerir más poder de cómputo. Luego, éste demostró no ser necesariamente el caso
- Usar data augmentation, pesos para los distintos targets en la función de pérdida, reducir el learning rate y entrenar por un mayor número de épocas, mejoró los resultados
- No obstante las mejoras obtenidas, las métricas resultan aun insuficientes
- A pesar se ser un modelo menos complejo, ResNet18 mostró mejores resultados

VGGNet

- La técnica de transfer learning que mejor funciona en todos los casos de VGGnet es feature extraction.
- El modelo de VGGNet16 presenta mejores resultados en comparación con VGGNet19 en todos los entrenamientos con los mismos hiperparámetros.
- Las técnicas de data augmentation y loss ponderada no producen mejoras significativas para este caso, sin embargo, se tomaron para los entrenamientos posteriores.
- Al comparar un clasificador lineal con un clasificador basado en un conjunto de capas, el clasificador de capas necesita mucho más optimización de hiperparámetros sin llegar a obtener los mismos resultados que al utilizar un clasificador lineal con un entrenamiento más simple.

VGGNet

- Para optimizar el clasificador basado en capas se debe disminuir la tasa de aprendizaje y la mejor forma de hacerlo es utilizar una tasa de aprendizaje variable y aumentar el número de épocas.
- En el caso del clasificador lineal, no se recomienda para esta problema disminuir la tasa de aprendizaje ya que produce una disminución de las métricas.
- La mejor forma de optimizar el clasificador lineal es aumentar el número de épocas con una tasa de aprendizaje de 0.001.
- La arquitectura VGGNet presenta mejoras contundentes con respecto a la arquitectura ResNet en esta clasificación.

Gracias