Universidad de la República

Facultad de Ingeniería

Taller de Aprendizaje Automático

Proyecto 2 | Retinopatía Diabética

Autores:
Federico Bello
Gonzalo Chiarlone
Ernesto Rován

28 de septiembre de 2025

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Índice

1. Introducción y Objetivo **2**

2. Exploración de los datos **2**

3. Métrica **2**

4. Análisis inicial **3**

5. Experimentos **3**

5.1. *Loss* . . . 4

5.2. Preprocesado de imágenes . . . 4

5.3. *Data Augmentation* . . . 5

5.4. *Label Smoothing* . . . 5

5.5. *Defreeze* de la red . . . 6

5.6. Clasificación vs Regresión . . . 6

6. Otras Arquitecturas **6**

6.1. Xception . . . 6

6.2. DenseNet . . . 7

6.3. Transformers . . . 7

7. Modelo Final **7**

7.1. Arquitectura . . . 7

7.2. Estragia . . . 7

7.3. Augmentations y preprocesado . . . 8

7.4. Entrenamiento . . . 8

7.4.1. Optimizer y scheduler . . . 8

7.4.2. Loss . . . 8

7.5. Resutlados . . . 8

8. Conclusión **9**

1. Introducción y Objetivo

La retinopatía diabética (DR) es actualmente una de las mayores causas de ceguera, afectando en algún grado a mas de 90 millones de personas mundialmente, causando así un fuerte impacto, tanto a nivel social como a nivel económico [9]. La detección de esta enfermedad consiste en analizar una imagen del ojo en búsqueda de anomalías vasculares. Hasta este momento, la detección de DR fue un proceso manual, el cual conlleva una gran cantidad de tiempo, solo pudiéndose hacer por un profesional capacitado. Debido a esta alta demanda y la baja capacidad de satisfacerla se generan largas demoras en el sistema médico. Esto, es un grave problema debido a que el 90 % de casos de esta enfermedad pueden ser prevenidos mediante la temprana detección y tratamiento [8].

Viendo la problemática con un enfoque optimista, se esta frente al escenario ideal para poner en practica una técnica de aprendizaje automático. Se tiene una problemática real, la cual conlleva una gran cantidad de tiempo, comenzándose a desbordar el sistema médico y que podría ser solucionado mediante técnicas de visión por computadora, buscando que el algoritmo aprenda a detectar esas anomalías vasculares.

Naturalmente surge entonces definir el objetivo de este proyecto como la puesta en practica de las herramientas vistas en el curso para buscar una posible solución a este problema.

2. Exploración de los datos

Para el entrenamiento se cuenta con un gran conjunto de imágenes de retina de ojo, identificadas con un *id_number*, y con la etiqueta correspondiente a ojo izquierdo o derecho. Los valores a predecir son el nivel de presencia de retinopatía diabética, en un rango de clasificación del 0 al 4.

Una de las primeras consideraciones es el gran desbalance de clases, como puede observarse en la tabla a continuación. Más del 70 % de las imágenes pertenecen a ojos sanos, i.e.: nivel de retinopatía cero. Esto claramente puede provocar un gran sobre ajuste en el modelo, por lo que puede ser positivo utilizar *class weights* a la hora de entrenar. Otra cosa importante es que, de cada paciente, se cuenta tanto con imágenes del ojo derecho como del izquierdo, por lo que es importante dividir en train y validación por paciente, dejando ambos ojos en el mismo conjunto.

Clase	Conjunto de Train	Conjunto de Val.
0	0.734	0.750
1	0.069	0.070
2	0.152	0.142
3	0.026	0.019
4	0.020	0.019

Cuadro 1: Peso de cada clase

Para evaluar el desempeño de entrenamiento se extrae un 10 % del conjunto para validación (division hecha por paciente), y como se observa en la tabla, el desbalance de clases se traslada satisfactoriamente al conjunto de validación, manteniendo las relaciones.

3. Métrica

La métrica de desempeño utilizada en este proyecto será la *quadratic weighted kappa* (QWK), la cual mide el nivel de acuerdo entre dos clasificadores. La misma suele variar entre 0, cuando hay un nivel de acuerdo aleatorio entre los clasificadores, y 1, cuando los clasificadores están de acuerdo en todas las predicciones. Eventualmente, si el nivel de acuerdo es menor al esperado aleatoriamente, la métrica podría tomar valores negativos. Por lo tanto, es similar al *accuracy* multiclase, solo que tomando como base la decisión aleatoria.

Para calcular la métrica, se comienza realizando un histograma O de tamaño NxN ($N = 5$ en este caso), donde la celda $O_{i,j}$ corresponde a la cantidad de imágenes donde la *label* es i mientras que se la clasifico como j. Luego,

se arma una matriz de pesos de tamaño NxN, donde cada entrada marca la diferencia entre la etiqueta clasificada y la real, de la forma:

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

Notar como la diagonal sera siempre nula, mientras que se penaliza mas tener una mayor distancia de la etiqueta real. Luego, se crea un histograma de NxN con los valores de las salidas esperadas, asumiendo que no hay correlación entre los valores de las etiquetas, es decir, en la posición i, j debe aparecer el numero esperado de veces que la etiqueta real era i pero se clasifico como j. Es decir:

$$E_{i,j} = n \cdot P(Y = i) \cdot P(\hat{Y} = j) = n \cdot \frac{y_i}{n} \cdot \frac{\hat{y}_i}{n}$$

Donde Y son las etiquetas reales, n la cantidad de predicciones, $\hat{Y}$ las predichas y y_i.y $\hat{y}_i$ las cantidades respectivas. Por lo tanto, para calcular esto alcanza con calcular el producto externo entre las etiquetas reales y las etiquetas predichas, normalizando para que E y O tengan la misma suma.

Por ultimo, el valor de la métrica viene dado por:

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}$$

Notar entonces que el valor máximo de la métrica es 1, donde todas las predicciones se hacen iguales (ya que como se observo, los pesos en la diagonal son nulos). Se puede pensar también el numerador como una función de costo para el clasificador propuesto, mientras que el denominador es la función de costo de las clasificaciones esperadas. Por lo tanto, se obtiene un buen resultado cuando el valor de la función de costo del clasificador propuesto es considerablemente menor al del esperado.

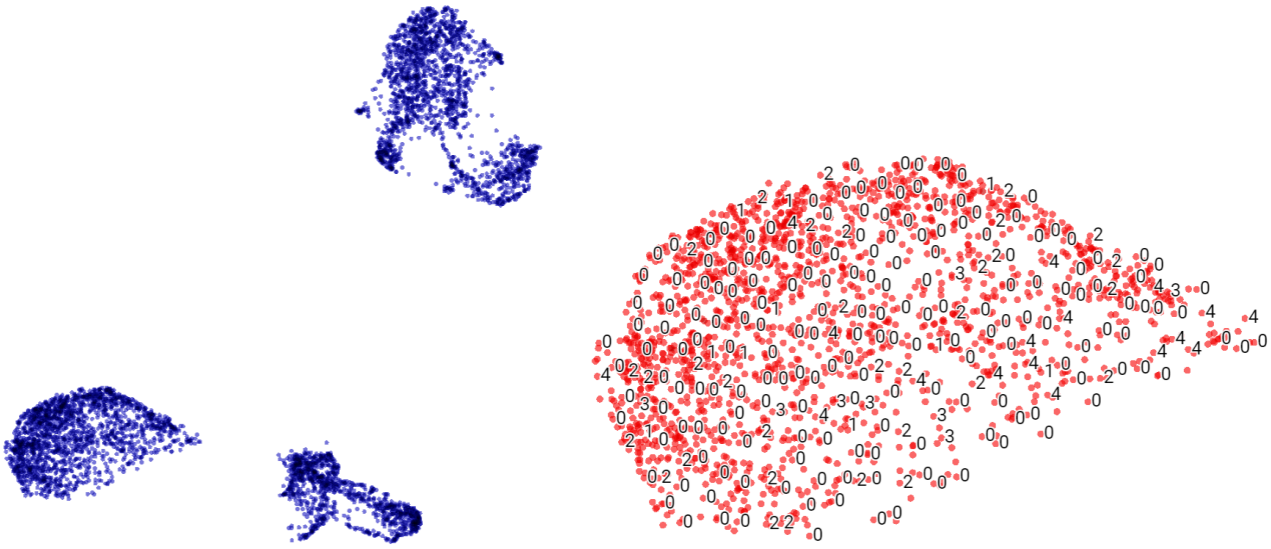
4. Análisis inicial

Luego de la exploración de datos, se abordó el problema mediante redes convolucionales, en principio utilizando la arquitectura *Resnet50*, aplicando *transfer learning* con el conjunto de datos *imagenet*. Debido a falta de recursos (GPU y datapoints), no se intentó entrenar ninguna arquitectura de cero.

El primer experimento realizado fue utilizar una *ResNet50* preentrenada para evaluar que tan bien generalizaba para este problema. Si bien más adelante se intentó entrenar solo un clasificador a partir de las características (*embeddings*) extraídas por esta red, primero se trató de realizar un enfoque que no requiriera entrenamiento. Para esto, se extrajeron los *embeddings* de todas las imágenes con la *ResNet50* preentrenada (vectores de 1024 dimensiones). Luego, se utilizó este espacio de *embeddings* como un clasificador, a través de un KNN. Es decir, dada una nueva imagen se calcula el embedding con la *ResNet* preentrenada y se clasifica segun la clasificacion de los K vecinos mas cercanos de los *embeddings* de la imagenes de entrenamiento. Observar que si bien *imagenet* es un dataset muy variado, y el *pretrained* de la *ResNet* logró el estado del arte en varios problemas de clasificación, es esperable que este modelo no pueda generalizar para un dataset tan especifico como este (imágenes médicas). En este caso, se obtuvieron resultados totalmente negativos (menos de 0.1 de QWK). Sin embargo, este resultado muestra algo muy importante: la representación espacial de que se obtiene con la *ResNet* no es totalmente acorde con la clasificación que se desea realizar. Por lo tanto, es altamente probable que sea necesario entrenar no solo un clasificador sobre el modelo, sino *finetunear* también los pesos de la propia ResNet. Esto mismo también se observó para otras arquitecturas de redes convolucionales.

5. Experimentos

Para entrenar varios modelos de redes convolucionales se buscaron optimizar tres cosas: la *data augmentation*, la *loss* utilizada y el modelo convolucional (*backbone*). Con el fin de poder corres varios experimentos, se utilizó una muestra del dataset de alrededor de 30 % del total de imágenes. Se realizaron diversos experimentos para poder analizar qué *data augmentation* y qué *loss* son las mas beneficiosas en este caso. A modo de igualar lo máximo



(a) *Clusters* **(b)** Etiquetas en un *cluster*

Figura 1: *Embeddings* generados por red preentrenada

posible las condiciones de los distintos experimentos, en todos los casos se utilizó como salida del modelo una capa densa de 1024 neuronas con un *dropout* de 0.2, seguido de una idéntica solo que de 512 neuronas y luego la capa de decisión de 5 neuronas. Todas las capas son precedidas de una capa *Layer Normalization*, se utiliza activación *gelu* en las capas intermedias y *softmax* en la de decisión. El optimizador usado fue *ADAM* con un *learning rate* de 1×10^{-3}.

5.1. *Loss*

Se siguieron las ideas presentadas en [2], con el objetivo de comparar las funciones de costo QWK y la entropía cruzada. En la tabla 3 se muestra el desempeño de cada *loss* en términos de distintas métricas de desempeño: F1, precisión categórica (CA por sus siglas en ingles) y QWK.

Loss	Conjunto	CA	F1	QWK
QWK	Entrenamiento	0.63	0.63	0.50
	Validación	0.64	0.63	0.47
Entropía Cruzada	Entrenamiento	0.77	0.76	0.49
	Validación	0.76	0.76	0.39

Cuadro 2: Desempeño de distintas funciones de costo

De la tabla se pueden extraer algunas observaciones interesantes. Primero, observar cómo utilizar la entropía cruzada obtiene mejores resultados en términos de precisión categórica y F1. Sin embargo, utilizar QWK de función de costo, como es esperable, obtiene mejores resultados en la métrica QWK. Sumado a esto, al utilizar entropía cruzada el modelo se comienza a sobreajustar rápidamente en términos de esta métrica, mientras que QWK no lo hace. Es posible que esto se deba a la diferencia entre ambas funciones de costo, es decir, mientras que QWK penaliza más por hacer una predicción mas lejana a la realidad, la entropía cruzada no lo hace. Es por estas razones, que para este problema parece mas adecuado utilizar como función de costo la propia QWK.

5.2. Preprocesado de imágenes

Se intentaron tecnicas de preprocesado de imágenes, imponiendo heruisticas y verificando si mejoraban o no el modelo. Primero, se experimentó pasar todas las imágenes a blanco y negro, partiendo de la base de que la

retinopatia no depende del color de la imagen. Sin embargo, esto no tuvo ningún efecto en el resultado. Por otro lado, se intentó *cropear* solamente el ojo en cada imagen, para no desperdiciar información en los bordes negros. Si bien este enfoque parecía prometedor, no se logro hacer un buen *crop* en todas las imágenes, ya que en algunos casos se descartaba parte del ojo, lo que resultaba en un descarte de información útil. Sin embargo, esta perdida de información eventualmente se podría compensar por mejorar la resolución del resto de la imagen, por lo que es una técnica que eventualmente se tendrá en cuenta. Por ultimo se probó dando vuelta todas las imágenes de un ojo (el izquierdo) para que el dataset fuera mas parecido entre si, pero esto tampoco tuvo efectos positivos o negativos sobre el modelo. Esto puede deberse a que el modelo ya es capaz de ignorar esta información. Debido a esto, se utilizaron las imágenes 'crudas', *resizeando* al tamaño adecuado para cada modelo.

5.3. *Data Augmentation*

Para agrandar de manera artificial el dataset y, a su vez, disminuir el sobreajuste, se probaron distintas *augmentations*. La idea principal era buscar cuales de estas son beneficiosas o perjudiciales para el modelo. Para esto el procedimiento fue simple: primero se eligieron algunas de las que se creyeron que podrían generar un impacto positivo, en este caso *random rotation, random brightness* y *random zoom*. Luego, se entrenaron modelos, donde en cada entrenamiento se agregó una *augmentation*, para luego ver cuales generaban un impacto positivo y cuales uno negativo. Los resultados de este experimento se resumen en la figura 2, donde se ve como tanto *random brightness* y *random rotation* generan este impacto positivo. Un comentario no menor es que al utilizar esta técnica se asume cierta independencia entre las distintas *augmentations*, por ejemplo, en este caso *random contrast* parece empeorar el modelo al aplicarlo junto con *random rotation*, la posibilidad de que la primera mejore el modelo si no se aplica junto con la segunda no se tiene en cuenta. A su vez, cabe destacar que la idea de utilizar esta técnica es de [4].

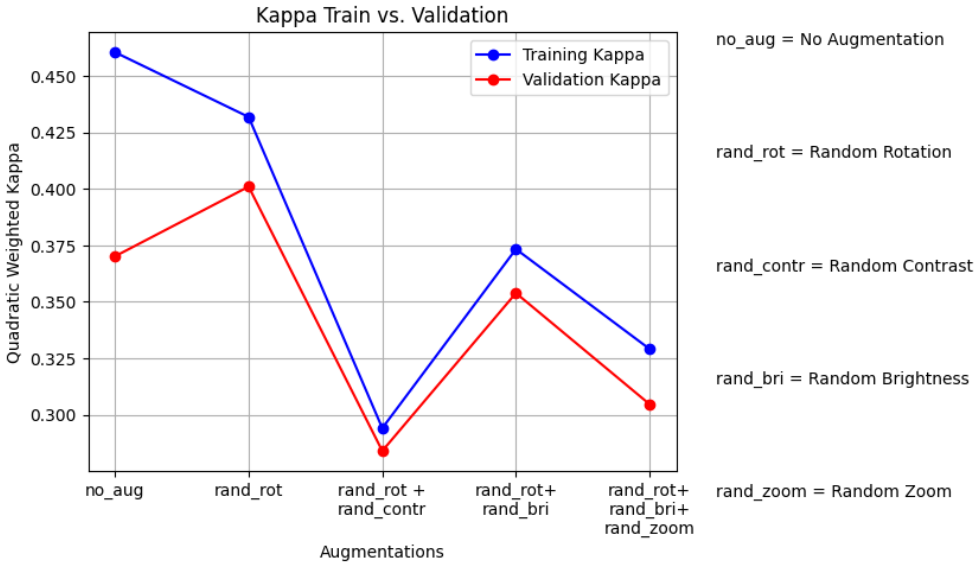


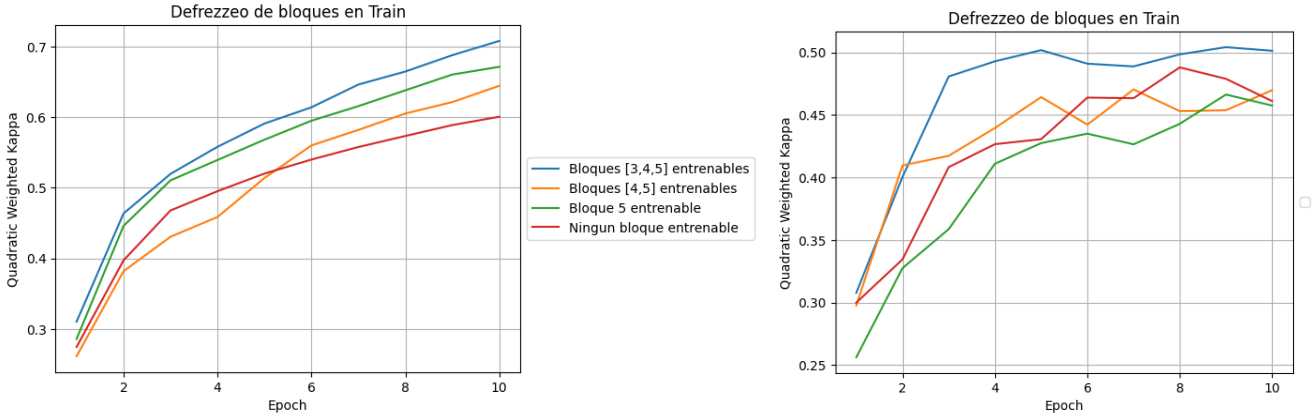
Figura 2: Efecto de distitnas *data augmentation* en el modelo

5.4. *Label Smoothing*

Se busco seguir las ideas presentadas en [5] y [6] , con el objetivo de regularizar el modelo aplicando *label smoothing*. Intuitivamente, la idea consiste en no 'exigirle tanto al modelo' sino que dejarle cierto margen respecto a sus predicciones. Es decir, si la salida esperada para una imagen es $[1, 0, 0, 0, 0]$, no parece razonable esperar del modelo que prediga esos valores exactos, sino que prediga valores cercanos. Por lo tanto, en lugar de utilizar esa *label*, se utiliza, por ejemplo, [0.9, 0.1, 0.1,. 0.1, 0.1]. De esta forma se lo penaliza menos por una respuesta cercana a la correcta. Los modelos anteriores no se encuentran sobre ajustados, por lo que no tiene sentido aplicar este método para los modelos anteriores. Esta técnica se evaluará en un modelo futuro.

5.5. *Defreeze* de la red

Otra alternativa, en lugar de entrenar solamente la *head* del modelo, es *finetunear* toda la red. Esto consiste en aprovechar la red preentrenada ajustándola al problema en cuestión, logrando una capacidad expresiva mayor para este problema. Se probaron distintas cantidades de bloques para ser *defrezeados*, esperando que a mayor cantiad de bloques mayor capacidad de expresión y mayor sobreajuste. Al realizar el experimento, efectivamente esto sucede, lo cual se ve resumido en la figura 3. Esta idea se explotará a lo largo de la búsqueda del modelo óptimo, haciendo uso de que al entrenar más capas de la red preentrenada la capacidad de expresión aumenta, buscando medidas para evitar el el sobreajuste. Un detalle es que, lógicamente, al entrenar mas capas el costo computacional aumenta, tanto en términos de tiempo de ejecución como de memoria, por lo que no es posible aumentar ilimitadamente la cantidad de bloques entrenables.



(a) En entrenamiento **(b)** En validación

Figura 3: Resultado de *defreezear* distinta cantidad de bloques

Si bien no es correcto extrapolar las conclusiones de estos experimentos para otros modelos, sí se tomaran como punto de partida, partiendo de la hipótesis de que estos resultados se deben principalmente al problema en cuestión y no a la arquitectura utilizada.

5.6. Clasificación vs Regresión

Un enfoque interesante al tratar el problema es que, si bien se trata de un problema de clasificación, también se puede entrenar como un problema de regresión. Esto se debe a que si bien la patología se separa en clases de gravedad, esta división es puramente arbitraria, por lo que es posible estimar un valor de gravedad para luego separar en clases. Para esto, se predice un valor numérico cambiando la ultima capa por una capa densa de un solo valor, con activación *Relu*. Luego de entrenar el modelo para predecir cada clase lo más cerca posible del valor real, se fijan umbrales para clasificar cada valor precedido por el modelo. Idealmente, la predicción de cada clase debería seguir una distribución normal, centrada en el valor de la clase, y con una varianza no mayor a 0.5. Notar que ese enfoque nos permite manejar mucho mejor las predicciones, ya que tenemos el control de los umbrales. A su vez, evita errores de clase múltiples (predecir una clase 0 como una clase 3), ya que estos se penalizan más fuerte que los errores simples.

6. Otras Arquitecturas

6.1. Xception

Luego de comenzar con los experimentos basados en la arquitectura *Resnet50*, se probó otra linea de experimentación y entrenamiento con otra arquitectura: *Xception*. En cuanto al paradigma de la arquitecura *Inception*, este consiste en utilizar bloques con filtros de diferentes tamaños que luego concatena para así extraer características a diferentes escalas.

Considerando lo visto en las secciones anteriores sobre *Data Augmentation*, las capas de *Head*, y lo conveniente de *defreezear* capas, se comenzó entrenando las últimas 10 capas para distintos *learning rates* con el optimizador *Adam*, y se procedió de igual manera con el optimizador SGD dado que en [1] ofrece mejores resultados para *Xception*. Puesto que no parece haber sobreajuste, otra idea menos arbitraria fue entrenar por entero el último bloque convolucional de la arquitectura, que está a la salida de la misma, antes de la capa *global average pooling layer* y la *head*. Este se denomina *"block14_sepconv3"*. Finalmente se prueba con una menor cantidad de capas de entrenamiento, dado que aumentar las mismas utilizando el bloque convolucional en cuestión mejoró los resultados en train pero no en validación. Entre todas las posibilidades exploradas, los mejores resultados fueron obtenidos a partir de una *Xception* con optimizador *Adam*, $lr = 1 \times 10^{-3}$ y 5 capas entrenadas. Con un promedio de *CohenKappa* en validación de 0.44, y picos de 0.48. No logró superar por mucho el mejor modelo basado en arquitectura *Resnet50*, más bien se alcanzaron niveles similares, y si bien el razonamiento anterior fue que aumentar la cantidad de capas entrenadas mejoraría el resultado, con esta arquitectura el rango de variación no fue muy considerable.

6.2. DenseNet

La siguiente arquitectura que se probó, y con la cual se obtuvieron mejores resultados fue *DenseNet*. Esta es una arquitectura de grandes bloques densos con múltiples capas de convolución, que se caracteriza por una estructura densamente conectada, donde cada capa está conectada directamente a todas las capas posteriores. Es un siguiente nivel en el incremento de la profundidad de las redes convolucionales, respecto a *LeNet*, *VGG* y *ResNet*.

Esta arquitectura a diferencia de la *ResNet50*,tiene la ventaja de poder variar el tamaño de entrada, por lo tanto, es posible mantener la proporción de las imágenes de entrenamiento. Ademas de esto, se probo también recortar los bordes negros, eliminando el 10 % de los mismos en todas las imágenes. De esta forma, se limita la información inútil que se le pasa al modelo, disminuyendo el costo computacional. Para esta arquitectura se realizaron pruebas en consideración de las observaciones realizadas anteriormente, y dado que aportó el mejor modelo sus resultados se exponen de forma detallada en la sección para del modelo final.

6.3. Transformers

Se intentó utilizar una arquitectura de *visaul transformers* [3], partiendo desde el modelo preentrenado CLIP [7], utilizando los pesos de la biblioteca *open source open-clip*. Se utilizo la arquitectura ViT-b32 preentrenada en Laion-2b. Si bien este modelo se enfoca en la relación texto-imagen, también se propone como un muy buen clasificador *zero-shot*. Sin embargo, al ser una tarea tan especifica, no se logro un buen resultado entrenando solo una head sobre el *encoder* de imágenes de CLIP. Por otro lado, *finetunear* todo el modelo requería de mucho entrenamiento, por lo que no pudo ser testeado durante el desarrollo del proyecto.

7. Modelo Final

7.1. Arquitectura

Como *backbone* principal se utilizo una *DenseNet121*, ya que fue la arquitectura que dio mejor resultado en comparacion con la *ResNet50* y la *Xception*.

7.2. Estragia

Para entrenar la red (tanto el *backbone* como la head) se entreno primero la head durante 10 epochs para que la prediccion no sea tan mala al principio. Luego, se entreno la arquitectura completa, desfreezeando los ultimos 3 bloques.

Para la *head* de clasificación, se utilizo una layer densa de 512, seguida de dropout 0.2, otra layer densa de 256, dropout 0.1 y una capa de clasificación final. En ambas capaz densas se utilizo activación *swish* y inicialización *g_normal*. La capa de clasificación final era una capa densa que bajaba a 5 dimensiones en el caso de clasificación, con activación *softmax*, y a 1 dimensión en el caso de regresión, con activación *relu*

7.3. Augmentations y preprocesado

Para el modelo final se utilizaron las *agumentations* de *RandomRotation* (rotación = 0.1) y *RandomBrightnes* (factor de atenuación = 0.4), ya que fueron las únicas que tuvieron un impacto positivo en los experimentos iniciales. A su vez, los valores para cada una se fueron probando partiendo desde 0.1 y aumentando el valor hasta disminuir la *performance* del modelo en validación. Para este modelo se eligió un tamaño de imagen de (224,291), manteniéndose de esta forma las proporciones de la imagen. También se mantuvo la estrategia de eliminar el 10 % de los bordes de la imagen, aunque esto no tuvo impacto positivo ni negativo.

7.4. Entrenamiento

7.4.1. Optimizer y scheduler

De manera genérica el entrenamiento consistió en el uso del optimizar *Adam*, buscando variar el *learning rate* a medida que se avanzo en el entrenamiento. Se comenzó con un valor inicial alto de 1×10^{-3}, tanto en el *backbone* como en la *head*. Una vez que el resultado comenzaba a oscilar, se comenzó a bajar el *learning rate*, primero en el *backbone* y luego en la *head*. El utilizar dos valores distintos para el optimizador no permitió hacer uso del *callback* *ReduceLROnPlateau*, por lo que este cambio de valores se realizo de forma manual. Ademas, se utilizo el *callback* de *checkpointing*, para ir guardar el mejor modelo entre todas las epocas, en este caso, la métrica de interés fue la QWK en validación. Con este ultimo punto en particular es necesario tener un poco de cuidado, ya que si por casualidad en una época el resultado de validación es excesivamente alto el modelo guardara este, cuando no necesariamente es el que mejor generaliza.

7.4.2. Loss

Como se mencionó de forma breve anteriormente, entrenar un modelo de regresión para este modelo tiene sentido a efectos intuitivos. Además, la función de costo MSE (*mean squared error*) tiene una gran correlación con la métrica QWK, ya que la misma penaliza estar mas lejos de la predicción correcta, en particular, de forma cuadrática al igual la métrica. El procedimiento consistió entonces en entrenar el modelo de clasificación. Una vez que se obtuvo un resultado satisfactorio con el mismo, se cambio el paradigma a uno de regresión, *finetuneando* todo el modelo y entrenando una *head* distinta con una sola neurona de salida. Para esto se tomo un *learning rate* bajo para la red (1×10^{-5}) y uno no tanto (1×10^{-3}) para la *head*.

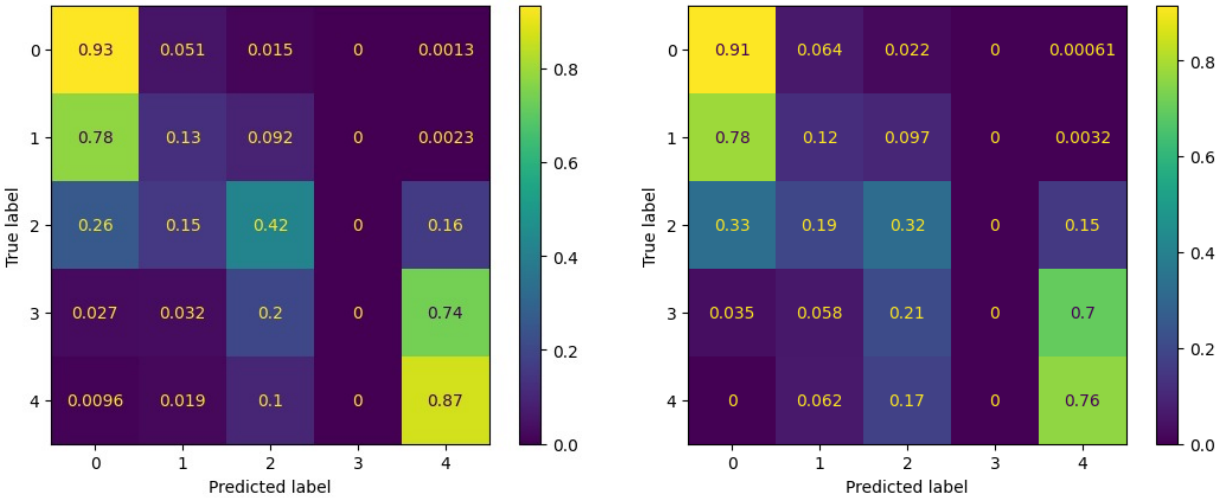
7.5. Resutlados

En la figura 4 se observan las matrices de confusión de ambos métodos implementados. De la misma se pueden extraer conclusiones interesantes. Por ejemplo, analizando la matrices del método de clasificación, se ve como el modelo aprende a no predecir la clase 3 y predice poco la clase 2. Haciendo esto se sobre ajusta a la métrica, ya que disminuye la probabilidad de errar a mas de una distancia de clase. Además, en ambos casos lo que mejor se predice son los extremos, ya que es determinar si tiene mucha retinopatía o no tiene nada.

A su vez, notar como en las matrices de decisión la matriz se asemeja mas a una matriz diagonal, ya que, como se menciono previamente, la regresión penaliza mas fuertemente el errarle por mas de una clase, pero además, al poder influenciar de forma manual en los *threshold* se evita que queden clases sin predecir.

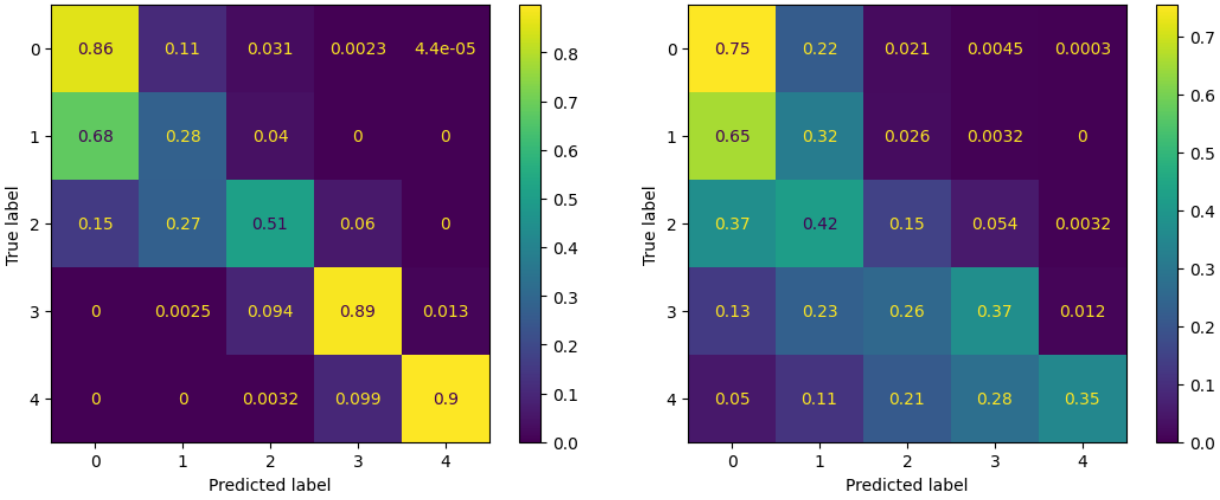
Modelo	Train	Val
Clasificación	0.77	0.71
Regresión	0.70	0.62

Cuadro 3: Resultados de los distintos modelo para la metrica QWK



(a) Clasificación en conjunto de entrenamiento

(b) Clasificación en conjunto de validación



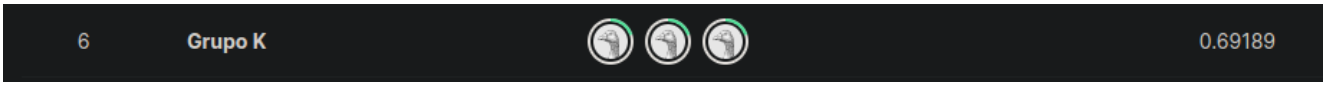
(c) Regresión en conjunto de entrenamiento

(d) Regresión en conjunto de validación

Figura 4: Matrices de confusión para ambos métodos en los distintos conjuntos

8. Conclusión

Se abordó el problema con distintos enfoques, tanto analizándolo como un problema de clasificación como uno de regresión. Se evaluó el mejor preprocesado de los datos, y para tres distintas arquitecturas se procuró el modelo con mejores resultados en validación, buscando los parámetros óptimos de entrenamiento para cada una de ellas. El mejor resultado fue arrojado por la arquitectura *DenseNet*, incluyendo los pesos correspondientes al desbalance de clases y entrenando la arquitectura entera. Si bien las capas preentrenadas arrojan buenos resultados en tiempo menor, dada la especificidad del problema en cuestión (detección de niveles de retinopatía diabética con imágenes oculares), la inversión en tiempo y costo computacional derivó en un desempeño altamente superior. La *submission* de la predicción del modelo en los datos de *test* alcanzó el siguiente *score* en *Kaggle*:



Resta probar cosas para mejorar este resultado. Lineas para seguir pudieron haber sido aprovechar la variabilidad del tamaño de entrada de algunas arquitecturas para trabajar con imágenes de mayor tamaño, manteniendo así la resolución de la misma. También se pudo haber recortado de forma exacta los bordes, con el objetivo de disminuir lo máximo posible los bordes negros. Otra alternativa hubiese sido, en lugar de usar *data augmentation*, buscar que el conjunto de datos fuera lo mas homogéneo posible, para así disminuir la varianza.

Referencias

[1] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. 8:1–8, April 2017.

[2] Jordi de la Torre, Domenec Puig, and Aida Valls. Weighted kappa loss function for multi-class classification of ordinal data in deep learning. *Pattern Recognition Letters*, 105:144–154, April 2018.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[4] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[5] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2019.

[6] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions, 2017.

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[8] Pedro Romero-Aroca, Juan Fernández-Ballart, Matias Almena-Garcia, Isabel Méndez-Marín, Merce Salvat-Serra, and Jose A. Buil-Calvo. Nonproliferative diabetic retinopathy and macular edema progression after phacoemulsification: Prospective study. *Journal of Cataract and Refractive Surgery*, 32(9):1438–1444, September 2006.

[9] Sanjay Sharma, Alejandro Oliver-Fernandez, Wei Liu, Patricia Buchholz, and John Walt. The impact of diabetic retinopathy on health-related quality of life. *Current Opinion in Ophthalmology*, 16(3):155–159, June 2005.