Selección de Modelos y Optimización de Hiper Parámetros para el dataset Retinamnist

Astrid Carolina Melo Guayacán 1202590
Jenifer Lizethe Leiva Martin 1202617

Universidad Militar Nueva Granada
Facultad de ingeniería
Deep Learning

Bogota D.C
25 de agosto de 2025

Marco de trabajo

MedMINST

Med MINST es una colección de datasets biomédicos diseñada para la investigación en aprendizaje automático profundo y, en particular, para el entrenamiento y evaluación de modelos de deep learning en imágenes médicas 2d y 3d. Contiene 18 subconjuntos de datos de varias condiciones médicas cuyas imágenes están normalizadas y reducidas a baja resolución (28×28 píxeles) para simplificar la experimentación y reducir el costo computacional.
Su propósito es meramente educacional, pues pretende establecer un benchmark estandarizado para algoritmos de clasificación en imágenes médicas y facilitar la investigación y la comparación entre modelos que no son específicos de uso clínico, es decir son más accesibles y menos complejos.

RetinaMNIST

La retinopatía diabética (DR) es la enfermedad ocular más común causada por la diabetes, y es la principal causa de pérdida de visión en adultos. El diagnóstico se hace mediante la inspección de imágenes del fondo de ojo en busca de lesiones como microaneurismas, hemorragias y exudados. El dataset RetinaMNIST es un compilado de grandes cantidades de imágenes de fondo de ojo con anotaciones de expertos, lo que facilita el desarrollo y validación de modelos automatizados de clasificación de DR.
Dentro de las características del conjunto de datos: hay imágenes de fondo de ojo, con resolución original de 1.736 × 1.824 píxeles.,Las imágenes están centradas y se redimensionan a 28 × 28 píxeles en formato RGB (3 canales), las etiquetas o clases de las imágenes contenidas en un vector de 5 posiciones corresponden a su clasificación clínica, que distingue cinco niveles de DR según la escala internacional ICDR, estos niveles son:

Gravedad de la condición médica		
Nivel	**Descripción**	**Observaciones de oftalmoscopia**
0	Sin retinopatía aparente	Sin anormalidades
1	NPDR leve	Microaneurismas
2	NPDR moderada	Microaneurismas y otros signos visibles
3	NPDR severa	Hemorragias, dilatación venosa, anomalías microvasculares, sin retinopatía proliferativa

4	PDR (retinopatía Proliferativa)	Neovascularización, hemorragia vítrea y complicaciones.

Clases presentes en el dataset RetinaMNIST

Metodología

Criterios de diseño

Primeramente hay que preparar los datos y hacer la configuración inicial para entrenar el modelo:

Instalación de dependencias.

1. Importar las librerías necesarias para trabajar en Python con datos, imágenes y modelos de aprendizaje automático.

```
import numpy as np
import random
import tensorflow as tf
import matplotlib.pyplot as plt
```

Importación de librerías y carga de datasets.

2. Importar módulos específicos de Keras, Tensor Flow, para construir y entrenar redes neuronales.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.utils import to_categorical
```

3. Instalar la librería Med MINST: para facilitar la carga de datos y división en train/val/test.

```
!pip install medmnist
```

4. Instalar Keras Tuner, librería de Python diseñada para encontrar la mejor configuración de hiper parámetros.

```
!pip install keras_tuner
```

5. Importar directamente el dataset RetinaMNIST desde la librería MedMNIST, para cargar imágenes de fondo de ojo ya pre procesadas y listas para usar en el modelo.

```
from medmnist import RetinaMNIST
```

6. Descargar y cargar los conjuntos de entrenamiento, validación y prueba de RetinaMNIST.

```
train_dataset = RetinaMNIST(split="train", download=True)
val_dataset   = RetinaMNIST(split="val", download=True)
test_dataset = RetinaMNIST(split="test", download=True)
```

7. Convertir las imágenes y etiquetas del conjunto de entrenamiento en arreglos de NumPy para trabajar con TensorFlow/Keras.

```
x_train = np.array([img for img, _ in train_dataset])
y_train = np.array([label for _, label in train_dataset])

x_val = np.array([img for img, _ in val_dataset])
y_val = np.array([label for _, label in val_dataset])

x_test = np.array([img for img, _ in test_dataset])
y_test = np.array([label for _, label in test_dataset])
```

Redimensionamiento de imágenes.

8. Dar forma a los datos de imágenes para que sean compatibles con modelos de TensorFlow/Keras que esperan un tensor 4D:

```
x_train = x_train.reshape( (x_train.shape[0], 28, 28, 3))
```

Donde:

- Posición 0: x_train.shape es el número de imágenes o tamaño del lote(batch size)
- Posición 1: es el ancho de cada imagen (width)
- Posición 2: es el alto de cada imagen (height)
- Posicion 3: son los canales de color (RGB)

Así, cada imagen queda en un formato (28, 28, 3) y el conjunto completo en un tensor (num_imágenes, 28, 28, 3).

Normalización

9. Normalizar las imágenes.

```
x_train = x_train.astype('float32') / 255.0
```

En este caso astype('float32') convierte los valores de píxel de enteros (uint8, rango 0-255) a números de punto flotante (float32). La división entre 255 escala los valores de píxel para que estén entre 0 y 1, en lugar de 0 a 255, lo cual ayuda a que la red neuronal converja más rápido durante el entrenamiento.

10. Mostrar el contenido del arreglo de entrenamiento y, que en este punto contiene las etiquetas de las imágenes de entrenamiento.

```
y_train
```

Codificación de etiquetas

11. Convertir etiquetas a one-hot encoding, que es el formato que Keras espera para clasificación multiclase.Convierte un array de etiquetas enteras (0, 1, 2, 3, 4) a vectores binarios de 5 posiciones.Cada posición representa una clase y solo tiene un valor 1, indicando la clase correcta, y 0 en las demás.

```
y_train = to_categorical(y_train, num_classes=5)
```

Características del modelo

Entre las principales características del modelo está que es secuencial y un perceptrón de tres capas: la primera una capa oculta de unidades que se definen por un tuner explicado más adelante, la segunda una capa oculta con 64 unidades y una capa de salida con un número de unidades equivalente a las clases, en este caso 5. En donde, como muestra la tabla resumen, se tiene una capa de entrada que al los datos tener más de una dimensión los aplana, dejando la entrada como un vector 28x28x3 elementos, lo que resulta en un modelo con capas totalmente interconectadas.

En cuanto a funciones de activación para cada capa, se asignaron la ReLu, la cual rectifica los valores de entrada, pues anula los negativos y mantiene los positivos, agrega la no linealidad al modelo y es sencilla a la vez que efectiva, la sigmoide para valores intermedios aunque genera un problema de gradientes que desaparecen, que reduce valores a un rango de 0,1 y la softmax para la capa de salida, que convierte valores en probabilidades dado que es la más común para obtener las probabilidades dadas por el modelo a cada clase.

Para la función de pérdida se utilizó la Categorical Crossentropy, la cual es la más común para problemas de clasificación multiclase con etiquetas one-hot-encoding, al calcular la diferencia entre las probabilidades predichas por el modelo, que se obtienen de la función softmax, y la verdadera distribución de clases.

El método utilizado para seleccionar los hiperparametros fue Hyperband Tuner, que a diferencia de solo probar combinaciones de hiperparametros al azar, dedica tiempo y recurso de entrenamiento a ellas. Hyperband genera combinaciones

aleatorias de hiper parámetros y en lugar de entrenar todos los modelos hasta el final, los entrena pocas épocas primero, luego selecciona solo los más prometedores y descarta los malos rápidamente (early stopping adaptativo), repite este proceso mediante iteraciones y al final entrena a máximo número de épocas (max_epochs) solo los mejores candidatos. Este modelo ahorra tiempo y cómputo y encuentra hiper parámetros óptimos más rápido, además posibilita probar muchos parámetros de manera eficiente. En últimas, al centrarse en los mejores candidatos, es más probable lograr una mayor precisión con menos pruebas.

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 2352)	0
dense (Dense)	(None, 256)	602,368
dense_1 (Dense)	(None, 64)	16,448
dense_2 (Dense)	(None, 5)	325

Total params: 1,857,425 (7.09 MB)
Trainable params: 619,141 (2.36 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,238,284 (4.72 MB)

Tabla resumen de las características del modelo

Resultados de la selección de hiper parámetros

Los tres hiperparametros que se seleccionaron para realizar el hyperparameter tuning fueron: las unidades, el optimizador y la tasa de aprendizaje, dada su influencia en el rendimiento, aprendizaje y complejidad del modelo. En donde, en primer lugar se separó la porción del dataset correspondiente a los datos de validación. Y se determinó el método a utilizar para el tuning, el cual fue el de hiperbanda, porque es rápido, y dinámicamente ajusta los recursos a los conjuntos de hiperparametros más prometedores, al dedicar más épocas y recursos computacionales para entrenar el modelo con los mismos.

```
tuner = kt.Hyperband(
    hypermodel=build_model4,
    objective='val_accuracy',
    max_epochs=30,
    directory="keras_tuner_dir",
    project_name="keras_tuner_demo4"
)
```

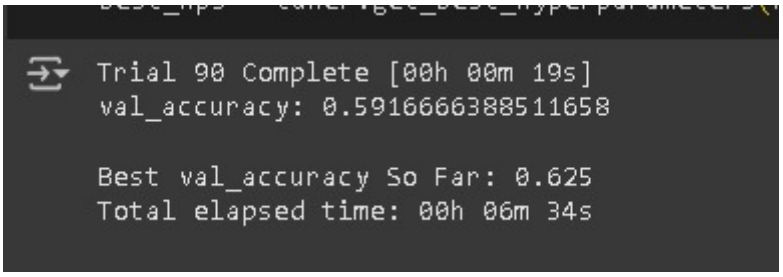
Objeto Hyperband de Keras Tuner

Luego, se plantearon 4 posibles modelos en los que se cambiaban las opciones del hiperparámetro de unidades, el batch size, el número de capas ocultas o se utilizaron técnicas de regularización, con el fin de seleccionar el que mejor overfitting tuviera en la etapa de entrenamiento.

En el primer modelo, se permitió un rango amplio de posibles unidades, lo que implicaba más tiempo evaluando posibilidades. Se establecieron 3 opciones de optimizadores populares y comúnmente utilizados en deep learning, y se dieron 3 valores de tasa de aprendizaje pequeños.

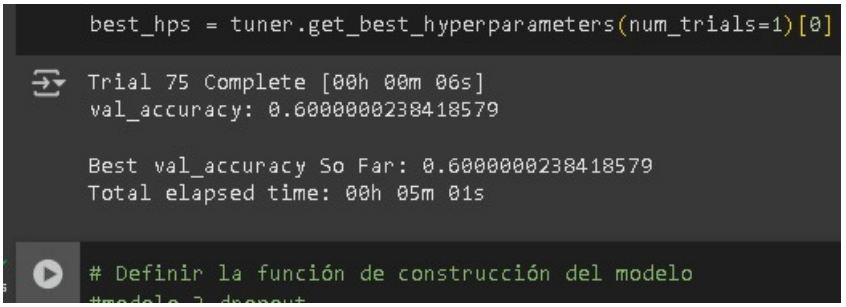
```
units=hp.Int('units', min_value=32, max_value=256, step=32)
optimizer = hp.Choice('optimizer', ['adam', 'sgd', 'rmsprop'])
learning_rate = hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])
```

Opciones de los tres hiperparametros seleccionados para el primer modelo



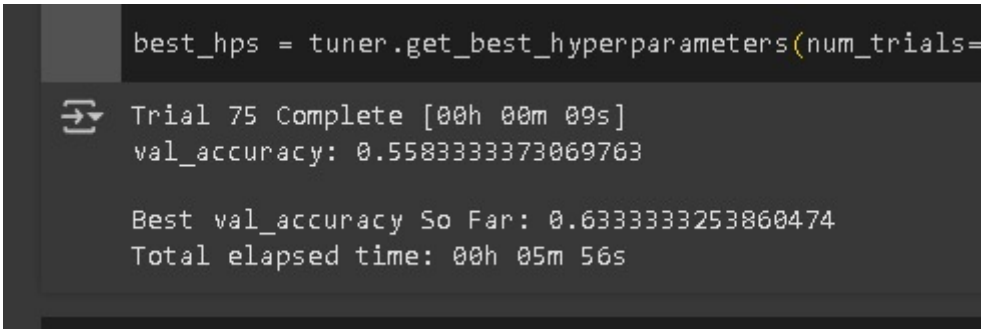
Resultados del valor de accuracy alcanzado por el modelo 1 con el conjunto más óptimo durante el proceso de evaluación de hiper parámetros y tiempo que tomó llevarlo a cabo

En el segundo, se aumentó el batch size de 32 a 42 y se cambió el mínimo valor de unidades a 96, para que hubieran menos combinaciones posibles de hiper parámetros, y las que quedarán correspondieran a unidades de modelos más complejos.



Resultados del valor de accuracy alcanzado por el modelo 2 con el conjunto más óptimo durante el proceso de evaluación de hiper parámetros y tiempo que tomó llevarlo a cabo

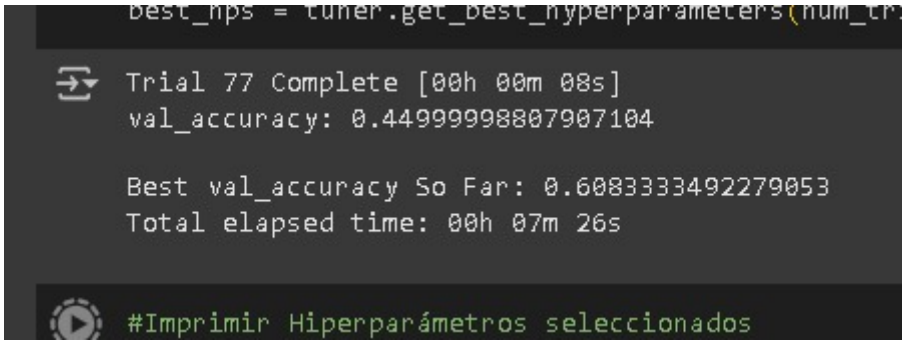
En el tercero y cuarto, a lo que se tenía en el segundo modelo se le agregaron técnicas de regularización. En el caso del tercero, el DropOut con un porcentaje para unidades de entrada en cero del 50%, y en el caso del cuarto, la técnica de Weight decay con un factor de penalización de 0.001. Además, de que en el cuarto se agregó otra capa oculta con función de activación sigmoide y un número de unidades de 64, buscando mejorar la generalización.



Resultados del valor de accuracy alcanzado por el modelo 3 con el conjunto más óptimo durante el proceso de evaluación de hiper parámetros y tiempo que tomó llevarlo a cabo

```
model4.add(layers.Dense(
    units=64,
    activation='sigmoid',
    kernel_regularizer=tf.keras.regularizers.l2(weight_decay)
))
```

Capa oculta agregada en el modelo 4



Resultados del valor de accuracy alcanzado por el modelo 4 con el conjunto más óptimo durante el proceso de evaluación de hiper parámetros y tiempo que tomó llevarlo a cabo

En donde, tras evaluarlos a todos, se llegó a que el modelo 4 definía el conjunto de hiperparámetros más óptimo, ya que si bien, en la etapa de evaluación de hiperparametros pareciera que el modelo 1 da mejores resultados, al aplicarlos al entrenamiento, se encontró que el modelo 4 era el que lograba en mayor medida

eliminar el overfitting, al dejar únicamente alrededor de 0.01 de diferencia entre el accuracy de validación y entrenamiento, a la vez que lograba valores de accuracy similares a los de los otros modelos.

Resultados del accuracy de entrenamiento y validación obtenidos al entrenar el modelo 4

Siendo este conjunto de hiperparametros el siguiente:

```
{'units': 256, 'optimizer': 'adam', 'learning_rate': 0.001,
'tuner/epochs': 10, 'tuner/initial_epoch': 4, 'tuner/bracket': 2,
'tuner/round': 1, 'tuner/trial_id': '0066'}
```

Resultado de lo obtenido en la selección de hiper parámetros por medio del tuner hyperband y con el modelo 4

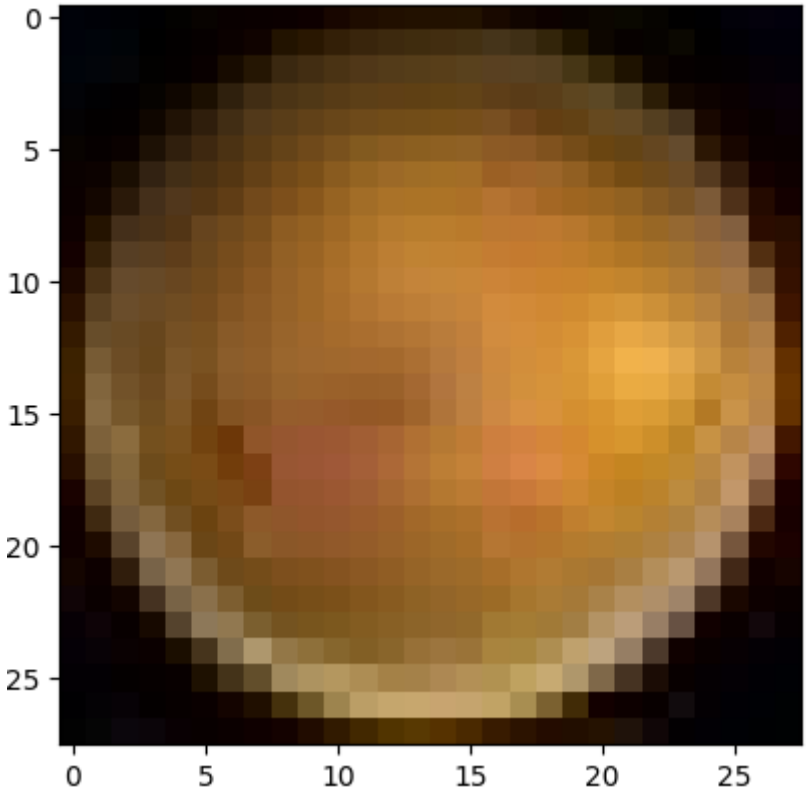
Optimizador: Adam
Unidades: 256
Tasa de aprendizaje: 0.001

Cabe resaltar que a los demás hiper parámetros se les asignó un valor fijo, como es el caso de la función de pérdida en Categorical Cross entropy y el número de épocas en 30.

Resultados de generalización

Al poner a prueba el modelo con la proporción de datos denominada test, se encontró que sus predicciones eran coherentes con la precisión que alcanzaba en entrenamiento y validación, siendo que lograba predecir bien la clase de las imágenes un número de veces similar al que daba predicciones erróneas.

En este sentido, a continuación se presentan algunas de las predicciones hechas con el modelo:




```
Etiqueta real: 3
```

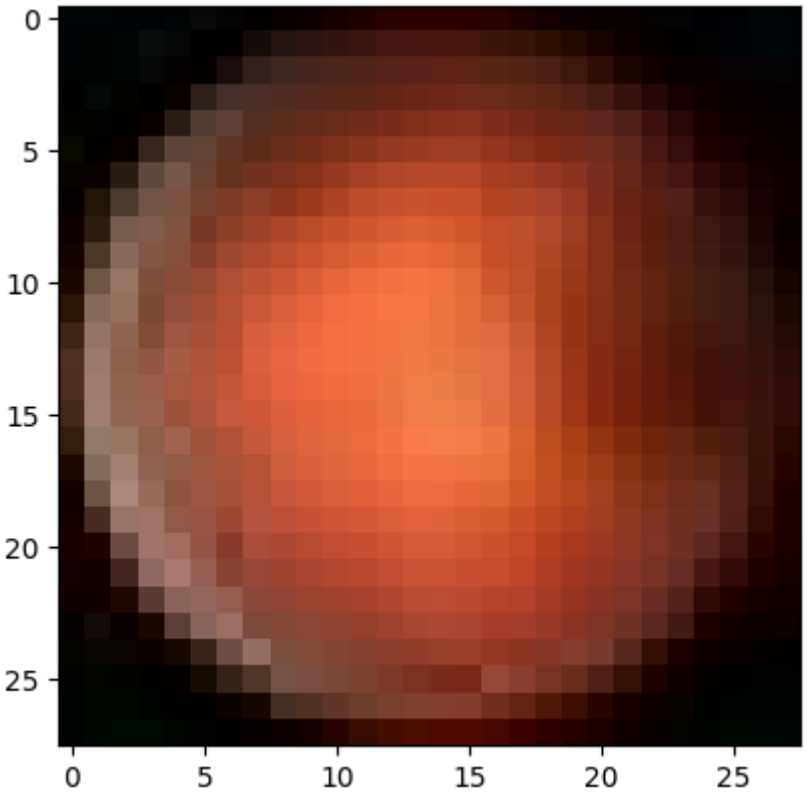
```
[ ]  model.predict(image)[0]

     1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
     array([0.1365473 , 0.28778505, 0.23342934, 0.3166142 , 0.0256241 ],
           dtype=float32)

[ ]  digit = np.argmax(model.predict(image)[0], axis=-1)
     print("Prediction: ", digit)

     1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
     Prediction:  3
```

Ejemplo 1, Predicción correcta de clase 3 del modelo para una imagen de clase 3




```
Etiqueta real: 0
```

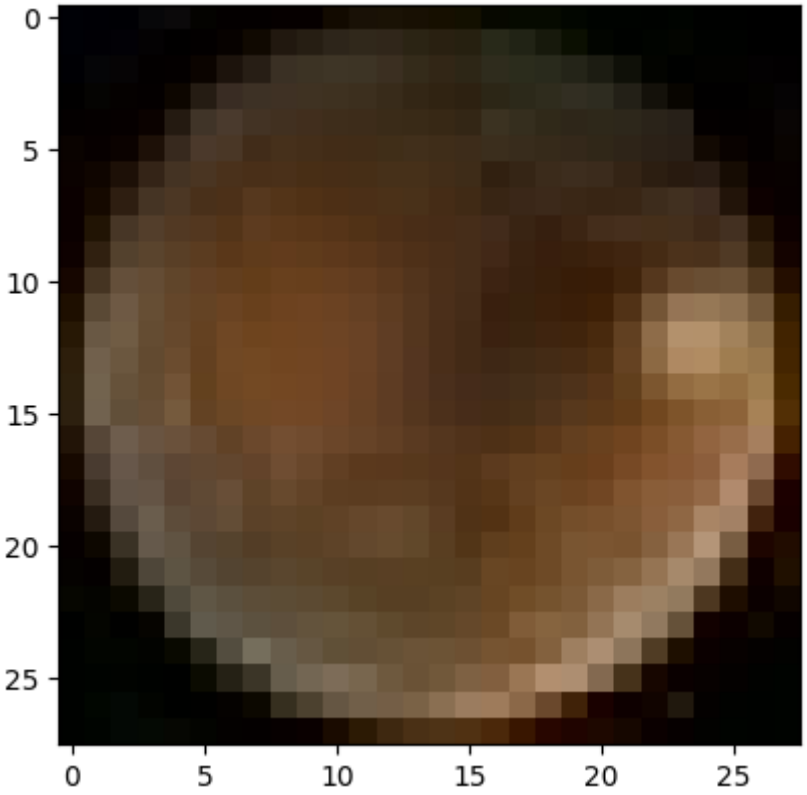
```
[57] model.predict(image)[0]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
array([0.8046655 , 0.0867056 , 0.06905208, 0.03056304, 0.0090137 ],
      dtype=float32)
```

```
digit = np.argmax(model.predict(image)[0], axis=-1)
print("Prediction: ", digit)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Prediction:  0
```

Ejemplo 2, Predicción correcta de clase 0 del modelo para una imagen de clase 0




```
Etiqueta real: 3
```

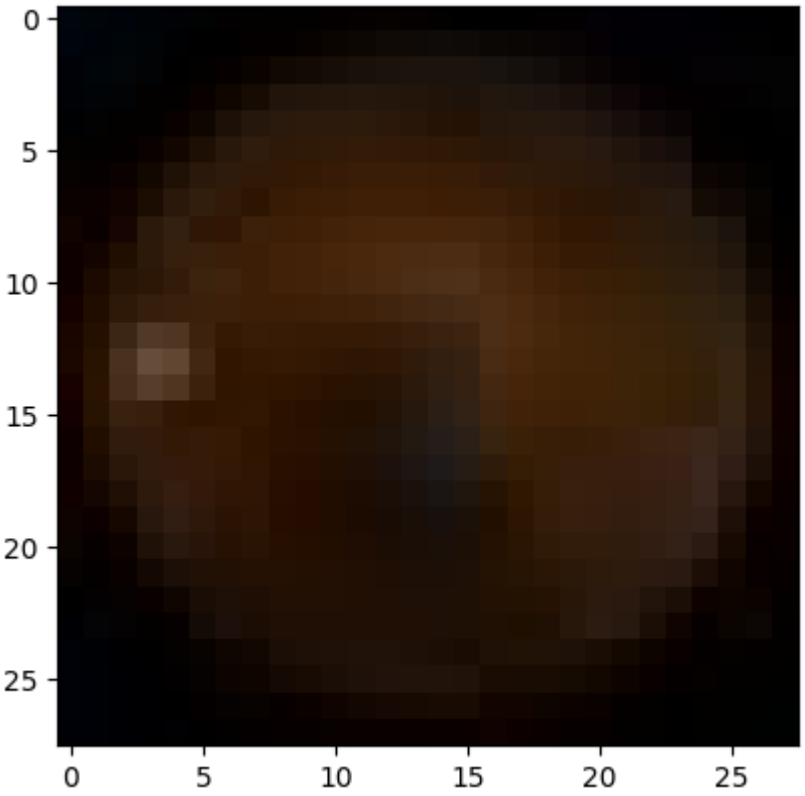
```
model.predict(image)[0]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
array([0.2612877 , 0.29510906, 0.22122438, 0.19708325, 0.02529565],
      dtype=float32)
```

```
digit = np.argmax(model.predict(image)[0], axis=-1)
print("Prediction: ", digit)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
Prediction:  1
```

Ejemplo 3, Predicción incorrecta de clase 3 del modelo para una imagen de clase 1




```
Etiqueta real: 2
```

```
model.predict(image)[0]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
array([0.15287521, 0.17753237, 0.23928744, 0.3194745 , 0.11083043],
      dtype=float32)
```

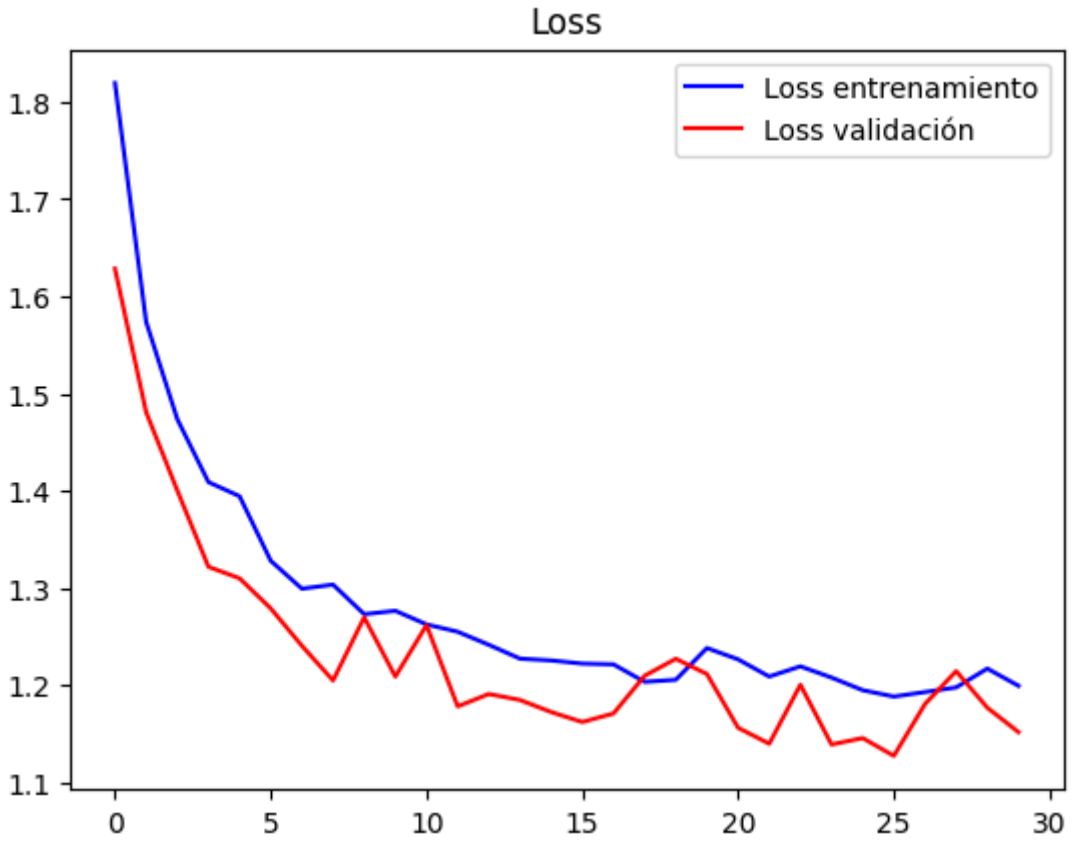
```
digit = np.argmax(model.predict(image)[0], axis=-1)
print("Prediction: ", digit)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Prediction:  3
```

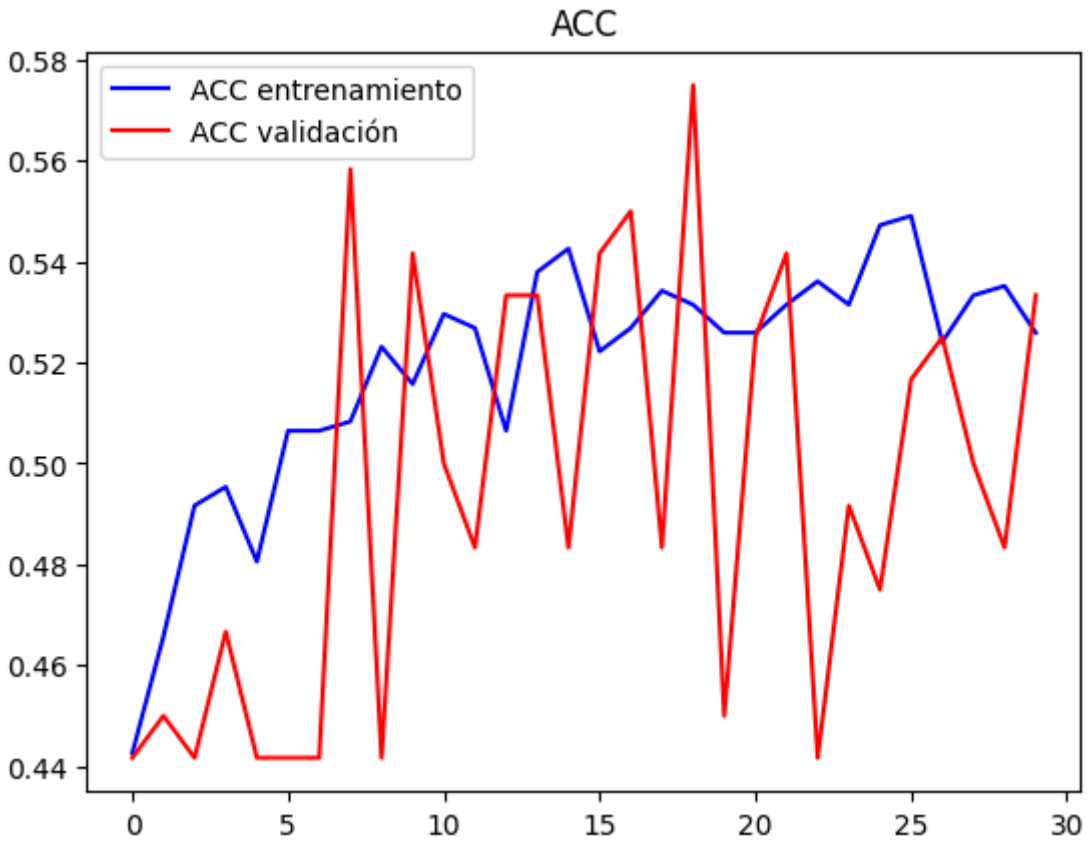
Del vector de probabilidades, se ve que si bien en algunos casos como el ejemplo 3, algunas clases tienen probabilidades muy similares, otras veces como en el ejemplo 2, el modelo logra asignar una probabilidad bastante alta a la clase correcta. Lo que implica que el modelo tiende a equivocarse en predicciones de imágenes cuyo color o características son similares entre clases, o como se ve en los ejemplos las más oscuras.

Evaluación de rendimiento

Para la evaluación de rendimiento del modelo, se toman como referencia las gráficas obtenidas para la evolución de la función de pérdida y el accuracy respecto al número de épocas.



Gráfica de Loss vs # épocas



Gráfica de Accuracy vs # épocas

De las que se observa que tanto para accuracy como para loss, los resultados con datos de entrenamiento y validación son similares, únicamente separadas por 0.01 o 0.03 aproximadamente, lo que indica que la técnica de regularización aplicada logró reducir el overfitting del modelo en gran medida.

No obstante, se ve que el valor de la función de pérdida supera la unidad, mientras que el accuracy se queda alrededor del 0.53, lo que implica que el rendimiento del modelo es deficiente, y en consecuencia, hay una alta probabilidad de que sus predicciones sean erróneas.

Esto se puede atribuir a que como la resolución de las imágenes del dataset es muy baja y, el color de las mismas en cada clase no presenta diferencias muy marcadas, sino que estas variaciones están más bien en zonas pequeñas blancas u oscuras, se hace difícil para el modelo poder identificar estos patrones pequeños dentro del ruido que presenta la imagen por la baja calidad.

Por ende, se analiza que un dataset con imágenes de mejor resolución, en el que los detalles sean más finos, visibles y sin ruido, podría resultar en un mejor desempeño del modelo.

Asimismo, dado que únicamente se manejan dos capas ocultas, se puede decir que el modelo que se propone es bastante sencillo para un problema complejo como lo es la identificación de patrones en imágenes. Por lo que tal vez, manejar mayor cantidad de capas hubiera dado un mejor desempeño en la generalización al permitirle al modelo aprender patrones más detallados.

Conclusiones

- Si bien no se obtuvo un modelo eficiente, si se lograron identificar y aplicar las principales características de un Perceptrón Multicapa, con un modelo secuencial fully connected con capa de entrada, ocultas y salida, el uso de funciones de activación que contribuyen a superar la linealidad, y funciones de pérdida basadas en un vector de probabilidades de las clases.

- Se comprendió al aplicar el método de hyperband que existen otras alternativas más eficientes a simplemente definir por medio de una constante los valores que el modelo no aprende durante el entrenamiento (hiper parámetros). Estas se conocen como hyperparameter tuning y ofrecen mayores ventajas al evaluar por medio de un conjunto de validación que hiper parámetros se ajustan mejor al modelo, y de este modo contribuir a un mejor aprendizaje y generalización por parte de este.

- Se encuentra que un dataset con imágenes de mayor resolución y mayor cantidad de muestras para las diferentes clases, así como la consideración de mayor cantidad de capas ocultas que permitan analizar al modelo relaciones más complejas, pueden llevar a obtener predicciones mucho más satisfactorias que las actuales.

- Se concluye del accuracy obtenido al probar los 4 modelos en entrenamiento, que las técnicas de regularización son efectivas para eliminar el overfitting, al reducir, por medio, de estrategias que involucran otros aspectos como limitar el peso de los parámetros o la anulación de ciertas unidades de entrada, la complejidad del modelo.

- El uso de la función de activación sigmoide para la capa de salida generó predicciones poco consistentes. Esto provocó que el modelo tendiera a predecir sólo ciertas clases de forma repetitiva, en lugar de distribuir adecuadamente la probabilidad entre las cinco categorías.

Anexo

Link al repositorio que contiene el código ipynb del proyecto:
https://github.com/Jenifer-Leiva/Trabajo-Practico-RetinaMNIST.git

Referencias

MedMNIST. (s. f.). https://medmnist.com/v2

Redirecting. (s. f.). https://www.google.com/url?q=https://www.sciencedirect.com/science/article/pii/S2666389922001040&sa=D&source=docs&ust=1756183328521342&usg=AOvVaw1IZHcDnZum7JsXsSVhRFCh

Halder, Arindam & Gharami, Sanghita & Sadhu, Priyangshu & Singh, Pawan & Woźniak, Marcin & Ijaz, Muhammad Fazal. (2024). Implementing vision transformer for classifying 2D biomedical images. Scientific Reports. 14. 10.1038/s41598-024-63094-9.

Wadekar, S. (2021, 15 enero). *Hyperparameter Tuning in Keras: TensorFlow 2: With Keras Tuner: RandomSearch, Hyperband. . ..* Medium. https://medium.com/swlh/hyperparameter-tuning-in-keras-tensorflow-2-with-keras-tuner-randomsearch-hyperband-3e212647778f