Clasificación de glaucoma utilizando Machine Learning

Ailyn Camacho R. *acamachor@mail.uniatlantico.edu.co*
Departamento de Física
Universidad del Atlántico

Abstract

En este estudio, se desarrolló una red neuronal convolucional para clasificar imágenes de glaucomas en las categorías Normal, Temprano y Avanzado. La arquitectura incluyó capas Conv2D con regularización L2, MaxPooling2D y Dropout para prevenir el sobreajuste. Durante el entrenamiento, se monitorizó el *accuracy* y la pérdida, graficando estos resultados (Figura 9). El modelo logró un 77.18 % de precisión en las predicciones

1. Introducción

El glaucoma es una enfermedad ocular que daña el nervio óptico, el cual transmite las imágenes al cerebro. Si el daño al nervio óptico es severo, puede provocar pérdida de la visión, por lo que la detección temprana del glaucoma es fundamental. Si se detecta el glaucoma a tiempo, el tratamiento puede ayudar a prevenir la pérdida de la visión. Uno de los métodos para diagnosticar y monitorear el glaucoma es la evaluación del nervio óptico, que consiste en fotografiar y observar el nervio óptico y la retina [1].

En este contexto, los avances en inteligencia artificial, particularmente en aprendizaje profundo (Deep Learning), emergen como herramientas valiosas para la detección y seguimiento de diversas enfermedades, entre ellas el glaucoma. Este progreso ha dado origen a las redes neuronales convolucionales profundas, las cuales están acelerando la integración de la inteligencia artificial con dispositivos de evaluación, tales como medidores de campo visual, imágenes de fondo de ojo y tomografía de coherencia óptica. El propósito subyacente es impulsar de manera más eficiente el desarrollo de técnicas clínicas destinadas al diagnóstico y pronóstico del glaucoma. Los algoritmos de aprendizaje profundo poseen la capacidad de analizar miles de imágenes con una precisión superior a la de los seres humanos, reduciendo así las posibilidades errores en el diagnóstico [2].

En este trabajo se utilizó un conjunto de datos público del repositorio de datos en línea "Harvard Dataverse", el cuál proporciona acceso a gran variedad de datos. El conjunto consiste en imágenes de la retina, incluidas imágenes de individuos sanos y pacientes con glaucoma [3]. Estas imágenes fueron empleadas para la implementación de una red neuronal convolucional (CNN) con el propósito de extraer características distintivas y clasificar ojos como sanos o afectados por glaucoma.

El conjunto de datos se organiza en tres carpetas, cada una destinada a imágenes de glaucoma en diferentes etapas: `advanced_glaucoma`, `early_glaucoma` y `normal_control`. Esta estructura es fundamental, ya que el modelo utilizado opera bajo aprendizaje supervisado, requiriendo que los datos estén etiquetados. Además, el archivo de datos proporciona una breve descripción del conjunto, indicando que las fotografías ya han sido preprocesadas.

2. Redes neuronales convolucionales (CNN)

Una red neuronal convolucional (CNN) es un tipo de red neuronal artificial que está especialmente diseñada para el procesamiento de imágenes. Las CNN se utilizan en una amplia gama de aplicaciones, que incluyen reconocimiento de imágenes, visión artificial y procesamiento de señales. Este tipo de algoritmo toma como entrada una imagen, para luego asignar un peso a los diferentes objetos o aspectos de la imagen y poder diferenciar uno del otro. Este tipo de red neuronal requiere un menor preprocesamiento en comparación con otros algoritmos de clasificación.

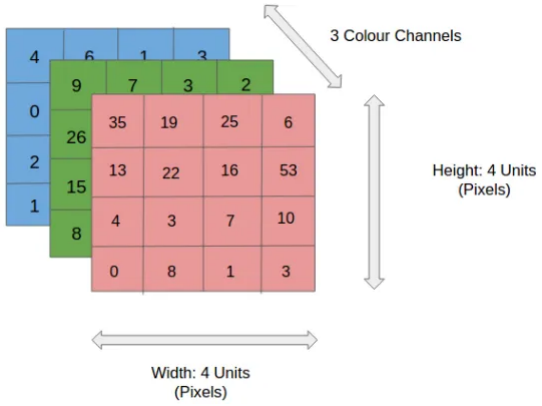

Figura 1: Imagen RGB que ha sido separada por sus tres planos de color: rojo, verde y azul. Esta es la entrada de una CNN [4].

En la Figura 1. se observa la representación de una imagen de 4 píxeles. Se puede apreciar la complejidad computacional que surge, especialmente cuando las imágenes alcanzan dimensiones elevadas. La función principal de una CNN es reducir la complejidad de estas imágenes a una forma más manejable para su procesamiento, sin perder características esenciales que son cruciales para realizar predicciones precisas.

El funcionamiento básico de una CNN se puede dividir en tres etapas:

- Convolución: en esta etapa la CNN aplica un filtro a una imagen. El filtro es una matriz de números que detecta características específicas en una imagen, como bordes, líneas o formas.
- Agrupación: aqui la CNN agrupa los resultados de la convolución para reducir la cantidad de datos que deben procesarse.
- Clasificación: finalmente la CNN conecta todas las neuronas de las capas anteriores en una sola capa. Esta capa se utiliza para tomar una decisión sobre la imagen.

Estas etapas se traducen en capas, por lo que la CNN va a tener 3 capas principales: la capa de convolución (Convolutional Layer), la capa de agrupación (Pooling Layer) y la capa de clasificación (Fully Connected Layer o FC layer), también conocidas como capas densas. Aunque es posible tener capas convolucionales seguidas por más capas convolucionales o capas de agrupación, la FC layer es la capa final [4][5].

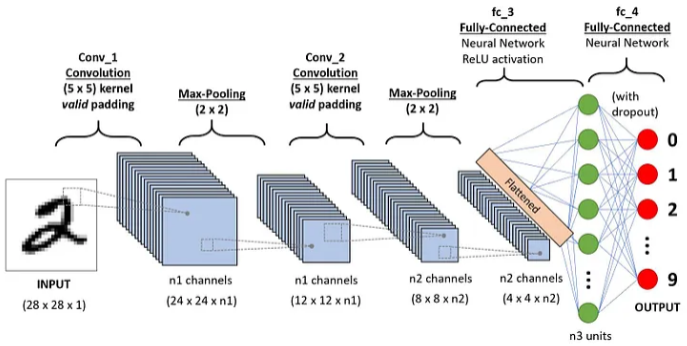

Figura 2: CNN para clasificar dígitos escritos a mano [4].

3. Metodología

1. Carga de los datos y procesamiento: Se cargaron las imágenes obtenidas de [3] y se procesaron para que tuvieran la forma adecuada para ingresar a la red neuronal. La división en conjuntos de prueba y entrenamiento también se realizó en esta etapa.

2. Creación del modelo: Se diseñó el modelo empleando múltiples capas de convolución y agrupación. Con el objetivo de mitigar el sobreajuste, se incorporaron capas Dropout.

3. Evaluación del modelo: La evaluación del modelo se llevó a cabo utilizando métricas como el *accuracy*. En esta fase, se ajustaron empíricamente parámetros clave, tales como el número de capas de la red, cantidad de filtros, número de *epochs*, y el tamaño del *batch*. El propósito fue encontrar combinaciones óptimas de estos parámetros que condujeran a un rendimiento mejorado en términos de *accuracy*.

4. Carga de los datos y procesamiento

Para poder utilizar las imágenes de los diferentes glaucomas estas se convirtieron previamente a matrices de píxeles utilizando la función `cv2.imread()`, que es una función de la biblioteca OpenCV en Python y se utiliza para leer imágenes desde archivos. Esta función toma como entrada la ruta de un archivo de imagen y devuelve una matriz NumPy que representa la imagen. Las imágenes se leen y representan como imágenes en color, donde cada píxel tiene tres valores (RGB). Las matrices de salida serán tridimensionales, representando la intensidad de luz en cada píxel para cada canal de color (rojo, verde, azul). A continuación se muestra una visualización de una imagen cargada:

Adicionalmente en el proceso de carga de los datos se asignaron las etiquetas a las imágenes, de la siguiente manera: Normal=0, Avanzado=1, Temprano=2. Los datos (matriz de cada imagen y etiqueta) se almacenaron inicialmente en una lista llamda `datos`, cuyos elementos correspondian a tuplas de la forma `[img_array, class_num]`, donde `img_array` es la matriz y `class_num` la etiqueta. Es importante notar que los datos en la lista estarán en el orden en que fueron creados (primero aparecerán los etiquetados con Normal=0, luego los Avanzado=1 y finalmente los correspondientes a Temprano=2), por lo que se utilizó la función `random.shuffle` para mezclar los elementos en la lista y reorganizar aleatoriamente los elementos.

Las variables x e y se crearon a partir del conjunto de datos previamente creado. La variable x contiene matrices de píxeles, mientras que la variable y almacena las etiquetas correspondientes (0, 1 y 2).

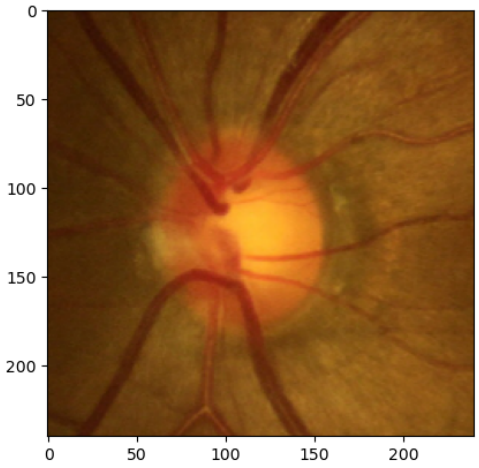

Figura 3: Glaucoma del conjunto de datos visualizado con Matplotlib.

```python
#Función para crear el dataset

datadir = 'dataset/glacucoma'  #Ruta de la carpeta con las imágenes
categorias = ['Normal', 'Avanzado', 'Temprano'] #Carpetas con las imágenes

datos = []

def crear_data():
    for categoria in categorias:
        path = os.path.join(datadir, categoria)
        class_num = categorias.index(categoria)  #En este caso los índices serán: Normal=0, Avanzado=1, Temprano=2
        for img in os.listdir(path):
            img_array = cv2.imread(os.path.join(path, img))
            img_array_rgb = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB) #Convertir las imágenes al formato RGB
            datos.append([img_array_rgb, class_num])
```

Figura 4: Función para crear el conjunto de datos a partir de las imágenes obtenidas en [3].

Por último se prepararon los datos de x como entradas válidas para una red neuronal. En primer lugar, las imágenes en x fueron redimensionadas a un tamaño uniforme de 50×50 píxeles mediante el uso de la función `cv2.resize()`. Posteriormente se creó un arreglo de NumPy a partir de x. Por último se modificó la forma del array utilizando `reshape()`. La forma resultante de x es (-1, 50, 50, 3), donde -1 indica que la dimensión se ajustará automáticamente para acomodar todos los elementos, 50 y 50 son las dimensiones de altura y ancho de las imágenes y 3 indica que cada píxel tiene tres valores (RGB) que representan la intensidad de luz en cada canal de color.

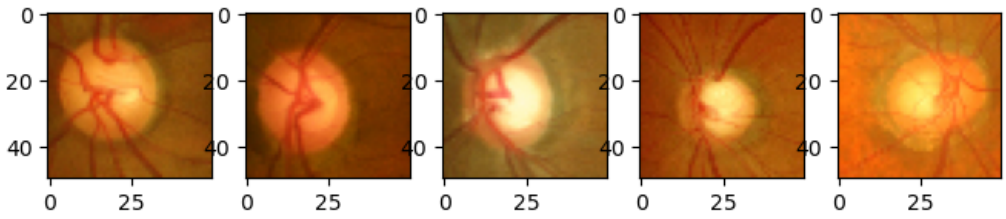

Figura 5: Algunas de las imágenes en x redimensionadas.

El conjunto de datos se dividió en conjuntos de entrenamiento y prueba utilizando una proporción de 60:40, respectivamente. Además se normalizaron los datos dividiendo cada valor de píxel por

255, esto para escalar los valores de píxeles en el rango $[0, 1]$, lo cual es una práctica común al trabajar con imágenes en tareas de aprendizaje automático.

5. Creación del modelo

La CNN utilizada constaba de varias capas de convolución y agrupación, seguidas de capas densas (ver Apéndice, Figura 10). Se agregaron además capas BatchNormalization, que normalizan las activaciones de la capa anterior. Ayuda a mejorar la estabilidad y velocidad de convergencia del modelo al normalizar la entrada de cada neurona. Las capas Dopout y la regularización aplicada a la capa Conv2D ayudan a prevenir el sobreajuste.

Como clasificador se utilizó la función Softmax, ya que es un problema de clasificación multiclase. Esta función utiliza una técnica de codificación One-Hot para calcular la pérdida de entropía cruzada y obtener el máximo. Por lo tanto, se utilizó de antemano la función `to_categorical` de keras para convertir las etiquetas a una matriz binaria. Se añadieron además capas Dropout para controlar la penalización de pesos y prevenir sobreajuste.

```
#Compilación del modelo

modelo.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = modelo.fit(x_train, y_train_oh, batch_size=65, epochs=40)
```

Figura 6: Compilación del modelo.

Para la compilación del modelo se utilizó la función de pérdida *Categrical crossentropy*, la cuál es comúnmente utilizada en problemas de clasificación multiclase cuando las etiquetas están codificadas de manera categórica (One-Hot encoding). Se utilizó además el optimizador Adam, el cuál es muy utilizado y adapta automáticamente la tasa de aprendizaje durante el entrenamiento. La presición (*accuracy*) se rastreó durante el entrenamiento.

Finalmente se hizo un programa sencillo para asignar las etiquetas a las predicciones, ya que la salida de la función Soxftmax devuelve la probabilidad de cada categoría, y no la categoría en si. Por último, el programa permite visualizar las n primeras predicciones (ver Apéndice, Figura 11). El resultado para las 5 primeras predicciones fue:

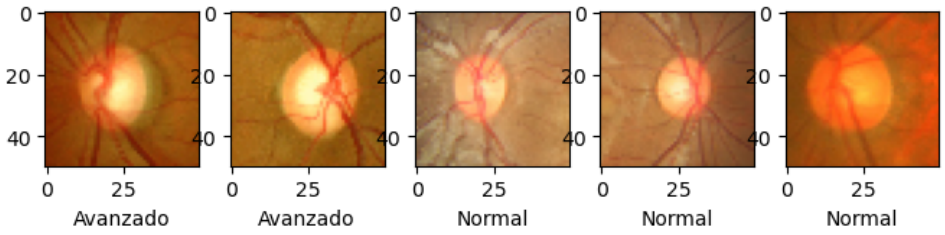

Figura 7: Resultados del programa.

6. Evaluación del modelo

La exactitud (*accuracy*) de un modelo es una métrica que mide la proporción de predicciones correctas realizadas por el modelo en relación con el total de predicciones. En este caso, el cálculo

de la exactitud del modelo planteado dió como resultado un valor de 0.7718, es decir que el 77.18 % de las predicciones realizadas por elmodelo son correctas.

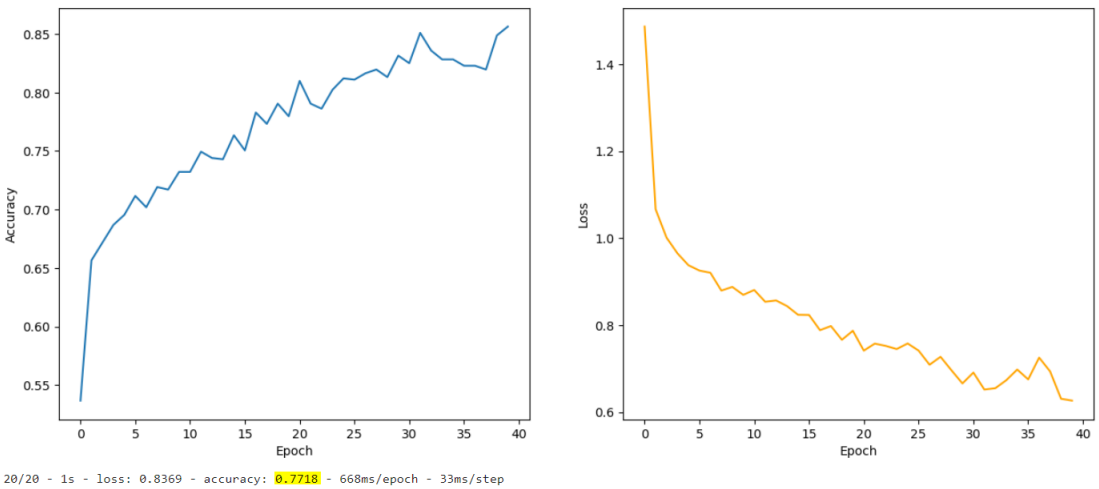

Figura 8: *Acuracy* y *loss* en función del número de *epochs*.

En la Figura 8. se muetra la as gráficas de los parámetros *accuracy* y loss en funcón del número de épocas durante el proceso de entrenamiento del modelo. El *accuracy* aumenta con las épocas mientras que la pérdida disminuye. Cabe resaltar que los datos utilizados presentan un imbalance de clases, como se puede ver en la Figura 9. La clase 0, que corresponde a "Normal", tiene mayor presencia en el conjunto de datos que las demás clases. Esto puede estar afectando la exactitud del modelo, por lo que si se quieren obtener resultados por encima de 70 % se debe tener en cuenta este factor.

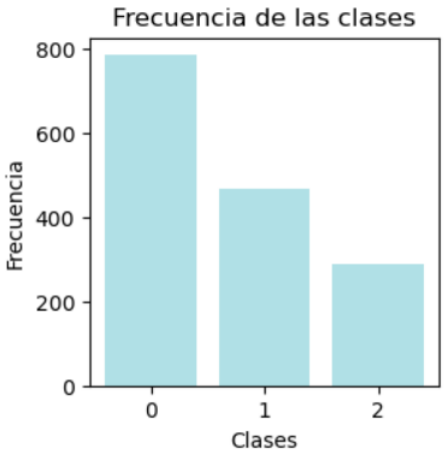

Figura 9: Frecuencia de las clases en el conjunto de datos.

7. Conclusión

En este trabajo se construyó una red neuronal convolucional simple para clasificar imágenes de glaucomas e identificar si correspomden a las categorías Normal, Temprano o Avanzado. Se utilizaron capas Conv2D (regularización L2 en los pesos de la capa), MaxPooling2D y capas Dropout

para prevenir el sobreajuste. Durante el proceso de entrenamiento se monitoreó el *accuracy* y la pérdida (*loss*) en cada una de las *epoch*, y se grafico este comportamiento. En la Figura 9 se observa el resultado. El resultado concuerda con el esperado: el accuracy aumenta mientras que la pérdida disminuye, tanto para el conjunto de prueba como para el conjunto de validación. Se obtuvo que el 77.18 % de las predicciones realizadas por el modelo son correctas. Finalmente, existe la oportunidad de mejorar aún más el rendimiento del modelo. Esto podría lograrse mediante ajustes finos en los parámetros del modelo o implementando estrategias más efectivas para gestionar el desbalance de clases.

Referencias

[1] "Glaucoma - Diagnosis and treatment." https://www.mayoclinic.org/diseases-conditions/glaucoma/diagnosis-treatment/drc-20372846, sep 30 2022.

[2] L. Zhang, L. Tang, M. Xia, and G. Cao, "The application of artificial intelligence in glaucoma diagnosis and prediction," *Frontiers in Cell and Developmental Biology*, vol. 11, may 4 2023.

[3] U. Kim, "Machine learn for glaucoma." https://doi.org/10.7910/DVN/1YRRAC, nov 15 2018. Harvard Dataverse, V1.

[4] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," *Towards Data Science*, nov 16 2022.

[5] "What are Convolutional Neural Networks?." https://www.ibm.com/topics/convolutional-neural-networks.

A. Apéndice

Figuras mencionadas en la Sección 5:

```
modelo = Sequential()
base_filtros = 32
lr = 89e-5

#Conv 1
modelo.add(Conv2D(base_filtros, (3, 3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(lr), input_shape=x.shape[1:]))
modelo.add(BatchNormalization())

#Conv 2
modelo.add(Conv2D(base_filtros, (3, 3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(lr)))
modelo.add(BatchNormalization())
modelo.add(MaxPooling2D((2,2)))
modelo.add(Dropout(0.2))

#Conv 3
modelo.add(Conv2D(2*base_filtros, (3, 3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(lr)))
modelo.add(BatchNormalization())
modelo.add(Dropout(0.3))

#Conv 4
modelo.add(Conv2D(2*base_filtros, (3, 3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(lr)))
modelo.add(BatchNormalization())
modelo.add(MaxPooling2D((2,2)))
modelo.add(Dropout(0.4))

#Conv 5
modelo.add(Conv2D(4*base_filtros, (3, 3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(lr)))
modelo.add(BatchNormalization())
modelo.add(Dropout(0.4))

#Conv 6
modelo.add(Conv2D(4*base_filtros, (3, 3), padding='same', activation='relu', kernel_regularizer=regularizers.l2(lr)))
modelo.add(BatchNormalization())
modelo.add(MaxPooling2D((2,2)))
modelo.add(Dropout(0.4))

#Clasificación - Flatten

modelo.add(Flatten())
#modelo.add(Dense(56, activation='relu'))
modelo.add(Dense(22, activation='relu'))
modelo.add(Dense(3, activation='softmax'))
```

Figura 10: Arquitectura del modelo.

```python
#Clasificación de las predicciones

def clasificacion(data):
    probs = []
    for i in range(data.shape[0]):
        valor = []
        for j in range(data.shape[1]):
            valor.append(data[i, j])
        valor_max = np.argmax(valor)
        if valor_max == 0:
            #probs.append(0)
            tipo = 'Normal'
            probs.append(tipo)
        if valor_max == 1:
            #probs.append(1)
            tipo = 'Avanzado'
            probs.append(tipo)
        if valor_max == 2:
            #probs.append(2)
            tipo = 'Temprano'
            probs.append(tipo)

    return probs

#Visualizar las n primeras predicciones

def vizualizar_prediccion(n):
    cl = clasificacion(y_pred)
    class_names = cl[:n]

    plt.figure(figsize=(8, 8))
    for i in range(n):
        plt.subplot(1, 5, i+1)
        plt.imshow(x_test[i])
        plt.xlabel(class_names[i])
        i = i + 1
    plt.show()
```

Figura 11: Programa para asignar las etiquetas predichas por el modelo y visualizar las imágenes correspondientes de `x_test`.