UNIVERSIDAD PRIVADA BOLIVIANA

DIPLOMADO EN MACHINE LEARNING Y CIENCIA DE DATOS

MÓDULO: Fundamentos de Deep Learning

TRABAJO FINAL

“Clasificación de rostros con anteojos usando redes neuronales”

DOCENTE: Mgr. Lesly Zerna

INTEGRANTES:

Gustavo Cadena

Giancarla Mendez

Fecha: 17-02-2024

1.SELECCIÓN DEL DOMINIO Y EL PROBLEMA

La tecnología ha evolucionado a pasos agigantados durante los últimos años, fruto de dicho avance día a día tenemos cada vez más y mejores sistemas de asistencia y apoyo a las tareas humanas, una de las áreas con mayores avances y de mayor impacto es el reconocimiento de imágenes, la sola idea que una máquina pueda reconocer patrones y catalogarlas es algo sorprendente.

Desde el reconocimiento de rostros, identificación mediante lectura del iris de la retina de los ojos, clasificación de animales, determinación de tipos de vehículos y otras tantas miles de tareas que podamos imaginar, el procesamiento y clasificación de imágenes es un área muy grande y extensa, existen miles de aplicaciones y funcionalidades cada una de las cuales merece un capítulo aparte.

Los rostros de las personas tienen características diferentes, ya sean ojos, labios, nariz, mentón u otras particularidades que los hacen únicos, sin embargo existe un elemento sumamente común, el uso de anteojos.

El presente estudio utilizará las redes neuronales para clasificar rostros con o sin anteojos (glasses), entrenaremos un modelo aplicando técnicas de Deep Learning, el resultado servirá para aplicar el modelo en entornos donde sea necesario reconocer el uso de anteojos o combinarlo con otros modelos de reconocimiento de rostros profundizando el avance en esta área

2. RECOPILACIÓN DE DATOS

La plataforma Kaggle, contiene una variedad de datasets a ser utilizados con fines de investigación, tras una búsqueda en dicho repositorio, encontramos un dataset con imágenes de personas con y sin anteojos, compuesto de varios archivos en las siguientes cantidades:

Set	Glasses	No Glasses	Total
Entrenamiento (train)	52	52	104
Validación (validate)	20	20	40
Prueba (test)	10	10	20

Totalizando 144 archivos para la fase de entrenamiento/ validación del modelo y 20 archivos sin clasificar para utilizarlos como prueba.

Respecto a porcentajes el 72% de los archivos serán destinados al entrenamiento y 28% para validación.

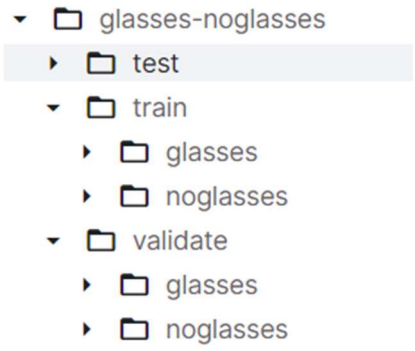
Respecto a sus características denotamos lo siguiente:

- Resolución: ≈160 x 155
- Tipo de archivo: JPG
- Clasificación: separados por directorios
- Modo: Color RGB

A continuación una captura de pantalla denotando las características descritas

Ejemplos de archivos con etiquetas Glasses/No Glasses

Propiedades de un archivo tipo



Clasificación por directorios

3. DISEÑO DEL MODELO

Para el procesamiento, entrenamiento y clasificación de imágenes en redes neuronales, el tipo más utilizado son las redes neuronales convolucionales (CNN), las cuales han supuesto una revolución en el sector del reconocimiento de imágenes, ya que, a diferencia de las redes neuronales convencionales y otros algoritmos de clasificación de imágenes, usan un procesamiento relativamente pequeño.

La red es entrenada a partir de imágenes utilizando solo píxeles y etiquetas como entradas. Se introducen el ancho y alto de las imágenes que deben estar previamente procesadas y el número de canales, 1 para blanco y negro y 3 si es en color RGB.

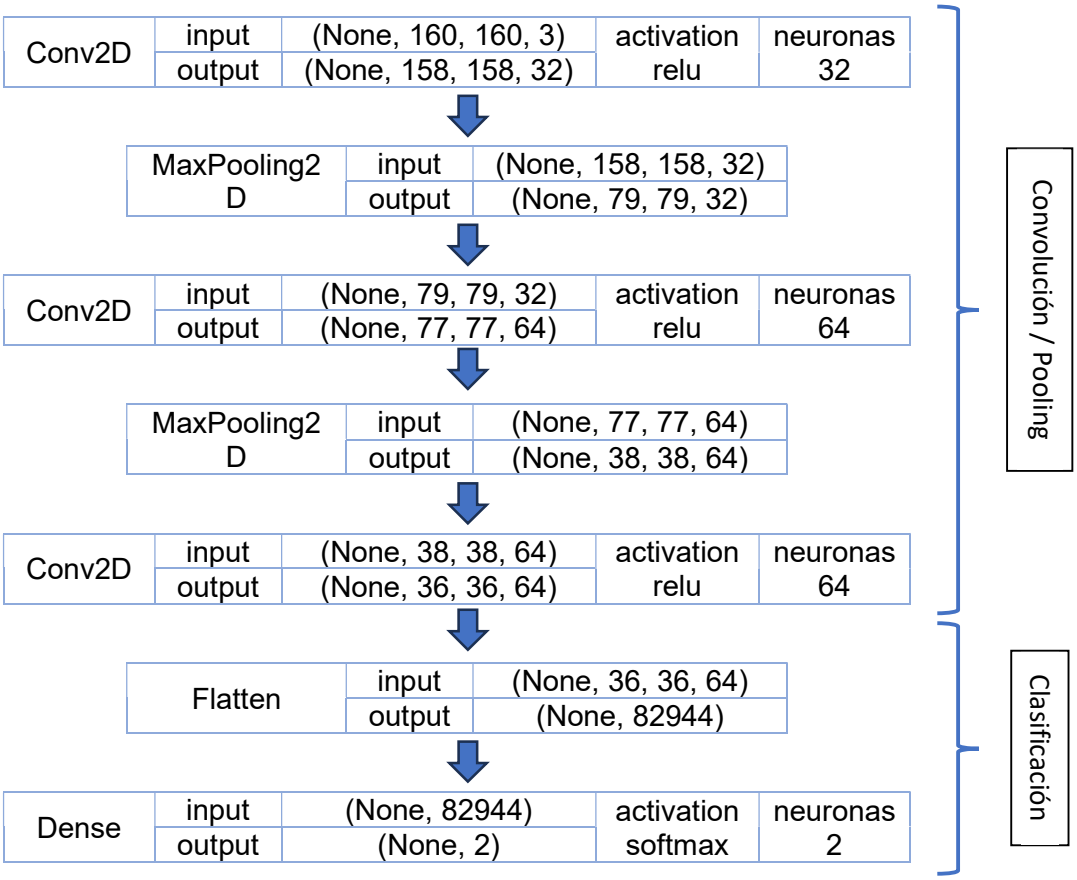
Internamente se agregan capas ocultas especializadas y con una jerarquía que mediante un proceso de retroalimentación imitan el cortex visual del cerebro humano identificando patrones utilizando operaciones matriciales matemáticas.

Para nuestro modelo utilizaremos el siguiente diseño y jerarquía:

Parámetros iniciales

Input Size: 160 x 160
Canales: 3
Model: Sequential

Diseño



Función de pérdida

"binary_crossentropy", dado que nuestra clasificación es binaria (glasses/no glasses).

Optimizador

"adam" porque combina beneficios de AdaGrad y RMSProp, optimizando los parámetros del modelo, minimizando la función de pérdida y aprendiendo las características de las imágenes para predicciones más precisas.

4. IMPLEMENTACIÓN DEL MODELO

Keras es una biblioteca de Redes Neuronales de código abierto escrita en Python. Además del soporte para las redes neuronales estándar, ofrece soporte para las Redes Neuronales Convolucionales y para las Redes Neuronales Recurrentes.

Librerías importadas necesarias

Para implementar nuestro modelo utilizaremos Keras dentro de Python:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
```

Cargado de imágenes utilizando ImageDataGenerator para el conjunto de entrenamiento y prueba

```
# cargamos los directorios
traindata = ImageDataGenerator()
train_images = traindata.flow_from_directory(
    directory='/content/drive/MyDrive/TF/glasses-noglasses/train',
    target_size=(160,160),
    class_mode='categorical'
)
testdata = ImageDataGenerator()
test_images = testdata.flow_from_directory(
    directory='/content/drive/MyDrive/TF/glasses-noglasses/validate',
    target_size=(160,160),
    class_mode='categorical'
)
```

se puede observar que están cargados los datos

```
Found 104 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
```

Preparación del modelo y sus capas de acuerdo al diseño

```
# preparamos el modelo
model = models.Sequential()
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(160, 160, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
# agregamos capas dense y flatten
model.add(layers.Flatten())
model.add(layers.Dense(2, activation='softmax'))
```

Funciones de pérdida, optimizador y entrenamiento

```
# compilamos y entrenamos el modelo
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['accuracy'])
history = model.fit(train_images , epochs=10, validation_data=(test_images))
```

5.Entrenamiento y Ajuste de Parámetro

Cargado de imágenes

El modelo se entrena utilizando el conjunto de datos de entrenamiento que se encuentra dentro de la siguiente carpeta:

Antes de realizar entrenamiento verificamos que las imágenes estén correctamente cargadas

```
# Definimos las clases
class_names = ['con lentes', 'sin lentes']
plt.figure(figsize=(10,10))

# Obtener un lote de imágenes y etiquetas
images, labels = next(train_images)

for i in range(8):  # Asumiendo que quieres visualizar 8 imágenes
    plt.subplot(2, 4, i + 1)  # Ajusta el subplot para 8 imágenes
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(images[i].astype('uint8'))  # Conversión para visualización correcta
    plt.xlabel(class_names[np.argmax(labels[i])])  # Asumiendo que las etiquetas están en
formato categórico

plt.show()
```

FIGURA Nº 4

Verificación de las imágenes

Una vez que se realizó la verificación de las imágenes que estén correctamente cargadas, procedemos a realizar el preparado del modelo

Preparación del modelo

Las capas de convolución se utilizan para extraer características importantes de las imágenes, y las capas de max pooling reducen la resolución espacial. La función de activación ReLU introduce no linealidades en el modelo. Este tipo de arquitectura es utilizada para tareas de clasificación de imágenes, adicionalmente usamos la capa ***Flatten() y la capa (Dense)*** con 2 neuronas de salida debido a que estamos abordando un problema de clasificación binaria, y usamos la ***función de activación softmax*** para normalizar las salidas de las dos neuronas de manera que sumen 1. Esto proporciona una interpretación de las salidas como probabilidades. por lo que la usamos en nuestro caso.

```
# agregamos capas dense y flatten
model.add(layers.Flatten())
model.add(layers.Dense(2, activation='softmax'))
```

Procedemos a mostrar los resultados de la preparación del modelo de la red neuronal

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 158, 158, 32)      896

 max_pooling2d (MaxPooling2  (None, 79, 79, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 77, 77, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 38, 38, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 36, 36, 64)        36928

 flatten (Flatten)           (None, 82944)             0

 dense (Dense)               (None, 2)                 165890

=================================================================
Total params: 222210 (868.01 KB)
Trainable params: 222210 (868.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

La arquitectura completa tiene un total de 222,210 parámetros entrenables.

Compilación y Configuración

La compilación del modelo se realiza especificando la función de pérdida ***(binary_crossentropy)***, el optimizador con tasa de aprendizaje adaptativa ***(adam)***, y se monitorea la métrica de precisión ***(accuracy)***. Esta configuración es esencial para optimizar la capacidad del modelo para la clasificación binaria.

```
#compilamos y entrenamos el modelo

from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001)

model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

La elección de la función de pérdida ***binary_crossentropy*** en la compilación del modelo la realizamos debido a la tarea de clasificación binaria que estamos abordando. Esta función de pérdida está especialmente diseñada para problemas de clasificación binaria, donde cada instancia pertenece a una de las dos clases posibles que en nuestro caso es (con lentes o sin lentes). La ***binary_crossentropy*** mide la discrepancia entre las distribuciones de probabilidad predicha y real, y es adecuada para problemas de clasificación binaria.

En las pruebas realizadas este fue el que mejor se adaptó a los datos que estamos usando

Entrenamiento del Modelo

El modelo es entrenado con el conjunto de datos de entrenamiento durante 10 épocas. La elección de 10 épocas se basa en la observación del comportamiento del modelo a medida que se ajusta a los datos. El conjunto de validación se utiliza para evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento, lo que proporciona información valiosa sobre su capacidad para generalizar

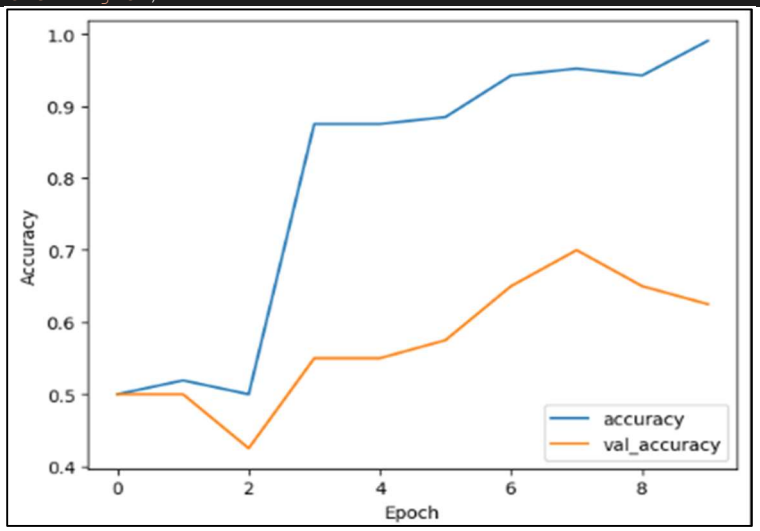
```
history = model.fit(train_images , epochs=10, validation_data=(test_images))
```

Se realiza el seguimiento del rendimiento del modelo a lo largo de las épocas mediante la recopilación del historial de entrenamiento.

```
Epoch 1/10
4/4 [==============================] - 7s 1s/step - loss: 290.9869 - accuracy: 0.5000 - val_loss: 36.3142 - val_accuracy: 0.5000
Epoch 2/10
4/4 [==============================] - 5s 1s/step - loss: 38.7215 - accuracy: 0.5192 - val_loss: 29.7208 - val_accuracy: 0.5000
Epoch 3/10
4/4 [==============================] - 7s 2s/step - loss: 15.2137 - accuracy: 0.5000 - val_loss: 3.7178 - val_accuracy: 0.4250
Epoch 4/10
4/4 [==============================] - 5s 1s/step - loss: 1.4298 - accuracy: 0.8750 - val_loss: 1.7855 - val_accuracy: 0.5500
Epoch 5/10
4/4 [==============================] - 6s 2s/step - loss: 0.3241 - accuracy: 0.8750 - val_loss: 1.1502 - val_accuracy: 0.5500
Epoch 6/10
4/4 [==============================] - 6s 2s/step - loss: 0.2934 - accuracy: 0.8846 - val_loss: 0.9062 - val_accuracy: 0.5750
Epoch 7/10
4/4 [==============================] - 5s 1s/step - loss: 0.2277 - accuracy: 0.9423 - val_loss: 1.1521 - val_accuracy: 0.6500
Epoch 8/10
4/4 [==============================] - 6s 2s/step - loss: 0.1238 - accuracy: 0.9519 - val_loss: 0.8856 - val_accuracy: 0.7000
Epoch 9/10
4/4 [==============================] - 6s 1s/step - loss: 0.1171 - accuracy: 0.9423 - val_loss: 1.3271 - val_accuracy: 0.6500
Epoch 10/10
4/4 [==============================] - 5s 1s/step - loss: 0.0552 - accuracy: 0.9904 - val_loss: 1.8470 - val_accuracy: 0.6250
```

La información de precisión se grafica para evaluar el desempeño del modelo durante el entrenamiento y la validación.

```
#Evaluamos el modelo y visualizamos la precisión a lo largo de las épocas

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
#plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```



6. ANÁLISIS DE LOS RESULTADOS

Loss (Pérdida):

En la primera época, la pérdida de entrenamiento es bastante alta (290.9869), pero disminuye significativamente en las siguientes épocas, indicando que el modelo está mejorando.

Accuracy (Exactitud):

La exactitud mide la proporción de predicciones correctas. Un valor de 0.5 en la exactitud podría indicar un modelo que está prediciendo aleatoriamente, pero en este caso, parece que mejora en las épocas siguientes.

A medida que avanzan las épocas, la exactitud de entrenamiento va mejorando hasta llegar al 99.04% en la última época.

Validation Loss y Validation Accuracy (Pérdida de Validación y Exactitud de Validación):

La pérdida de validación disminuye desde la primera época, indicando una mejora en la generalización del modelo.

La exactitud de validación varía, pero parece estabilizarse alrededor del 62.5% en la última época.

Tiempos de entrenamiento:

Los tiempos de entrenamiento por época varían, pero generalmente oscilan entre 5 y 7 segundos por época.

La precisión en el conjunto de entrenamiento ha variado a lo largo de las epochs y ha alcanzado un máximo del 99.04% en la última epoch. Esto indica un rendimiento muy sólido en el conjunto de entrenamiento.

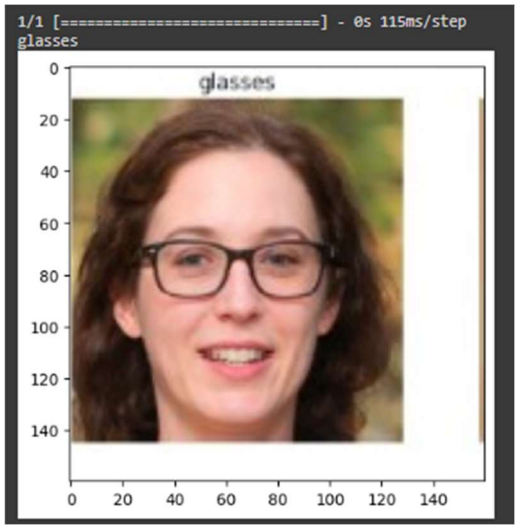
Cargamos una imagen para realizar pruebas

Para poder realizar estas pruebas primero guardamos el modelo posteriormente de la carpeta de test que tenemos en el directorio utilizamos una imagen y realizamos la prueba

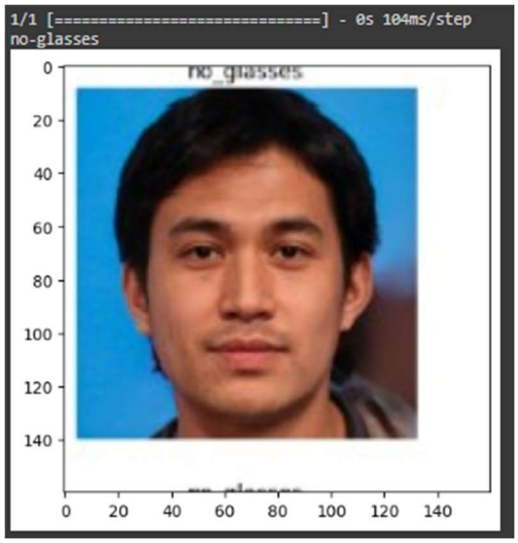
```
#Cargamos una imagen de prueba y la mostramos
from keras.preprocessing import image
img = image.load_img("/content/drive/MyDrive/TF/glasses-noglasses/test/row-1-column-
2.jpg",target_size=(160,160))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)
from keras.models import load_model

# Cargamos el modelo guardado
saved_model = load_model("pruebas")

# Realizamos la predicción sobre la imagen de prueba
output = saved_model.predict(img)
if output[0][0] > output[0][1]:
    print("glasses")
else:
    print('no-glasses')
```

Como podemos observar en las siguientes imagenes el modelo predice correctamente

Con lentes



Sin lentes



7. CONCLUSIONES

El diseño e implementación del modelo de Red Neuronal Convolucional (CNN) para la clasificación de imágenes de personas con y sin lentes ha arrojado resultados acordes con las expectativas establecidas. Al evaluar el modelo en el conjunto de imágenes de prueba, se observó un rendimiento coherente y preciso en la identificación de las clases objetivo.

Evaluación en Imágenes de Prueba

El modelo fue sometido a pruebas utilizando un conjunto de imágenes independientes no vistas durante el entrenamiento. Los resultados obtenidos al realizar predicciones sobre estas imágenes indican que el modelo ha logrado generalizar adecuadamente y reconocer patrones relevantes para distinguir entre personas con lentes y sin lentes.

Resultados Esperados

Los resultados obtenidos fueron consistentes con las expectativas y objetivos del proyecto. El modelo demostró su capacidad para generalizar a partir del aprendizaje adquirido durante el entrenamiento, clasificando de manera precisa a las personas en las dos categorías definidas. La elección de la arquitectura de CNN, junto con la función de pérdida binary_crossentropy y el optimizador Adam, ha demostrado ser efectiva para resolver este problema específico de clasificación binaria.

ANEXO 1 Pruebas realizadas

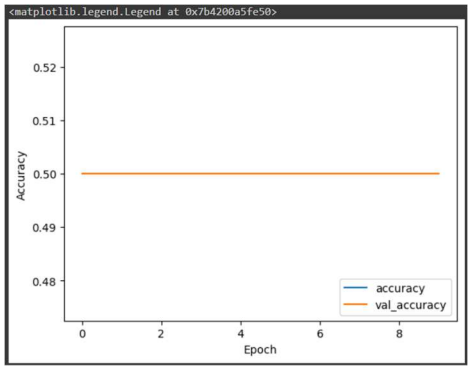
Prueba 1

Para poder tomar decisiones importantes sobre el tipo de redes neuronales a utilizar, la estructura de la red, las funciones de pérdida y los algoritmos de optimización, realizamos diferente pruebas que se pueden observar a continuación

al agregar la capa de activación “sigmoid” que es para datos binarios pudimos observar en la gráfica que

```
# agregamos capas dense y flatten
model.add(layers.Flatten())
#model.add(layers.Dense(2, activation='softmax'))
model.add(layers.Dense(1 activation='sigmoid'))
```

pudimos observar en la gráfica que no hay diferencia entre los datos



Prueba 2

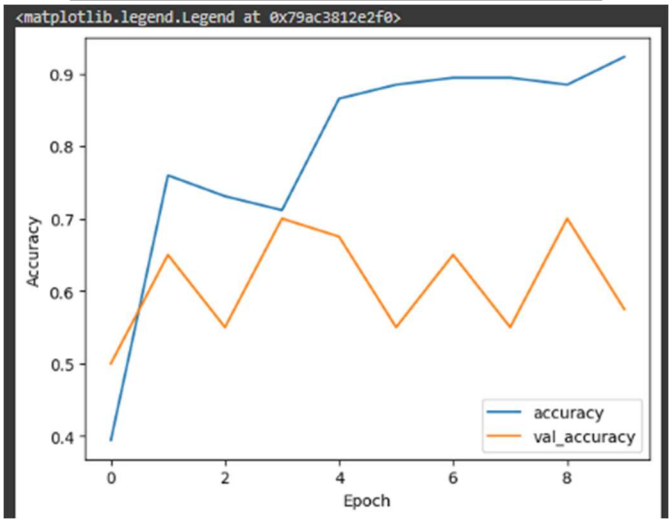
```
# Cargar imágenes de entrenamiento
traindata = ImageDataGenerator(rescale=1./255)  # También puedes agregar otras configuraciones de aumento de datos aquí
train_images = traindata.flow_from_directory(
    directory='/content/drive/MyDrive/TF/glasses-noglasses/train',
    target_size=(160,160),
    class_mode='binary',  # Cambiar a 'binary' para clasificación binaria
    batch_size=32  # Ajustar el tamaño del lote según sea necesario
)

# Cargar imágenes de prueba/validación
testdata = ImageDataGenerator(rescale=1./255)
test_images = testdata.flow_from_directory(
    directory='/content/drive/MyDrive/TF/glasses-noglasses/validate',
    target_size=(160,160),
    class_mode='binary',  # Cambiar a 'binary' para clasificación binaria
    batch_size=32  # Ajustar el tamaño del lote según sea necesario
)
```

Cambiar class_mode a 'binary' para que las etiquetas de clase sean 0 o 1, que es apropiado para una tarea de clasificación binaria. Además, se ha agregado la configuración rescale=1./255 en los generadores de datos para normalizar los valores de píxeles en el rango [0, 1].

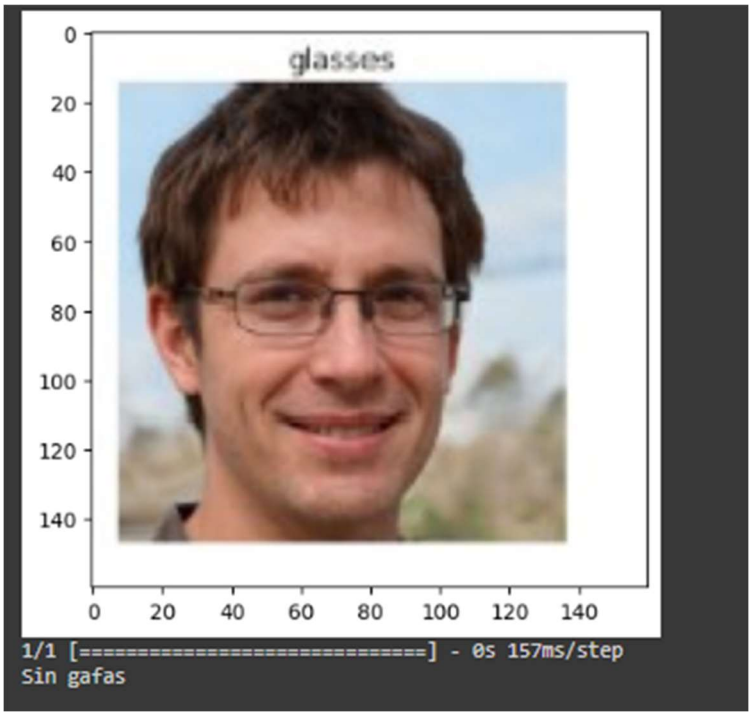
y utilizamos la función de activación 'sigmoid' y probamos agregando capas densas para la clasificación final

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 158, 158, 32)      896

 max_pooling2d (MaxPooling2  (None, 79, 79, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 77, 77, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 38, 38, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 36, 36, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 128)               1048704

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 1297857 (4.95 MB)
Trainable params: 1297857 (4.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
<matplotlib.legend.Legend at 0x79ac3812e2f0>
```



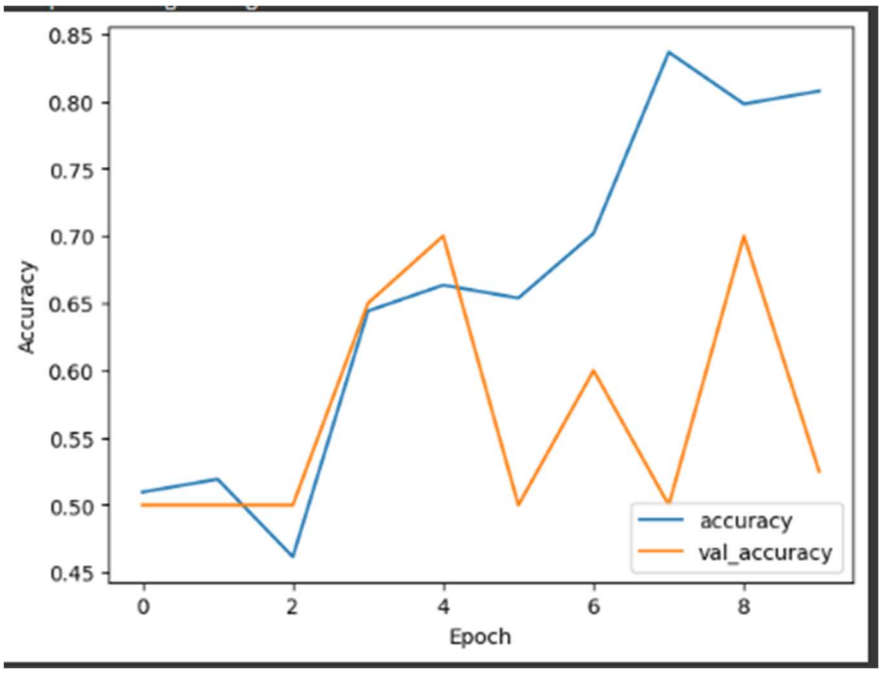
Si bien se ve que existe una mejora en los datos al momento de realizar las pruebas con las imágenes vemos que no reconoce



prueba 3

agregar capas de dropout antes de las capas densas para ayudar a prevenir el sobreajuste.

```
from tensorflow.keras import layers, models

model = models.Sequential()

model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(160, 160, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dropout(0.5))  # Agregar capa de dropout antes de la primera capa densa
model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dropout(0.5))  # Agregar capa de dropout antes de la segunda capa densa
model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))  # Para una clasificación binaria
```



La precisión en el conjunto de entrenamiento parece mejorar ligeramente a lo largo de las épocas, llegando a alrededor del 80% al final. Sin embargo, la precisión en el conjunto de validación es variable y oscila alrededor del 50-70%, lo cual podría indicar cierta dificultad para generalizar a datos no vistos.