CLASIFICACIÓN DE ROSTROS CON ANTEOJOS USANDO REDES NEURONALES

Gustavo Cadena

Giancarla Mendez

DESCRIPCIÓN DEL PROBLEMA

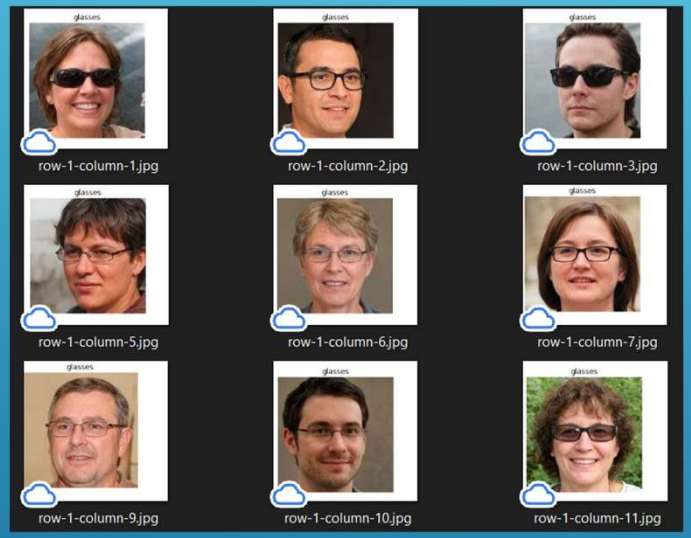
Los rostros de las personas tienen características diferentes (ojos, labios, nariz, mentón).

El uso de anteojos es muy común.

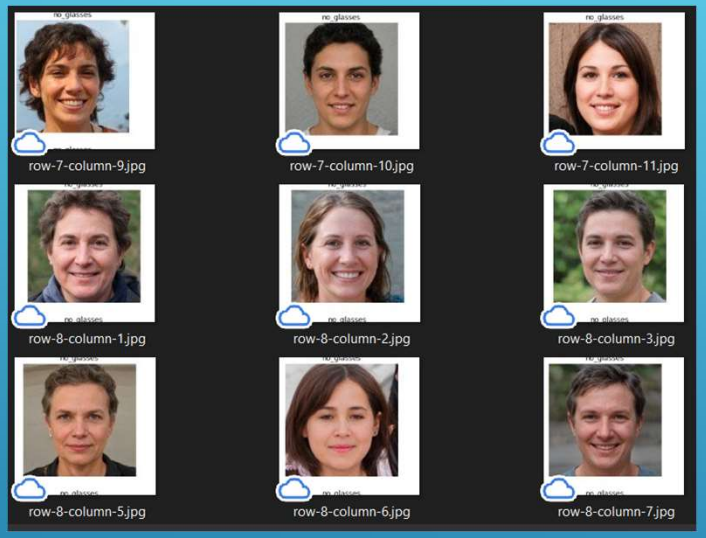
El reconocimiento del uso de anteojos mediante Deep Learning puede ser muy útil en entornos donde sea necesario o combinarlo con otros modelos de reconocimiento facial.

DATOS

Glasses



No Glasses

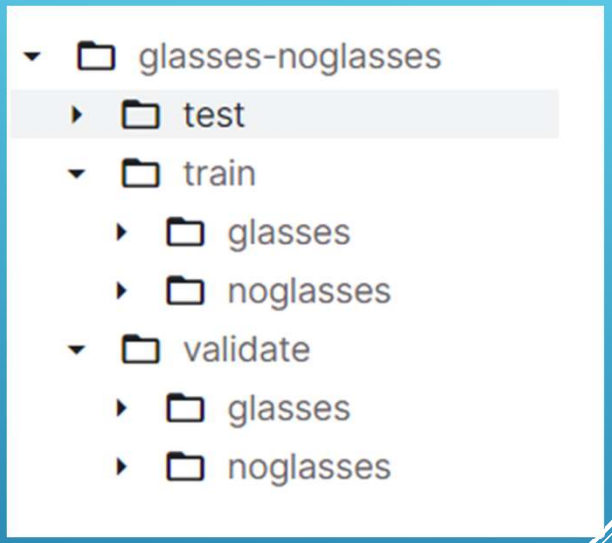


Mediante kaggle.com se extrajo un dataset consistente en:

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

144 en total
72 % train
28% validation

CARACTERISTICAS IMAGENES



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

DISEÑO INICIAL DEL MODELO

Red Neuronal Convolucional (CNN)

Algoritmo de clasificación de imágenes, usando un procesamiento relativamente pequeño

Parámetros iniciales

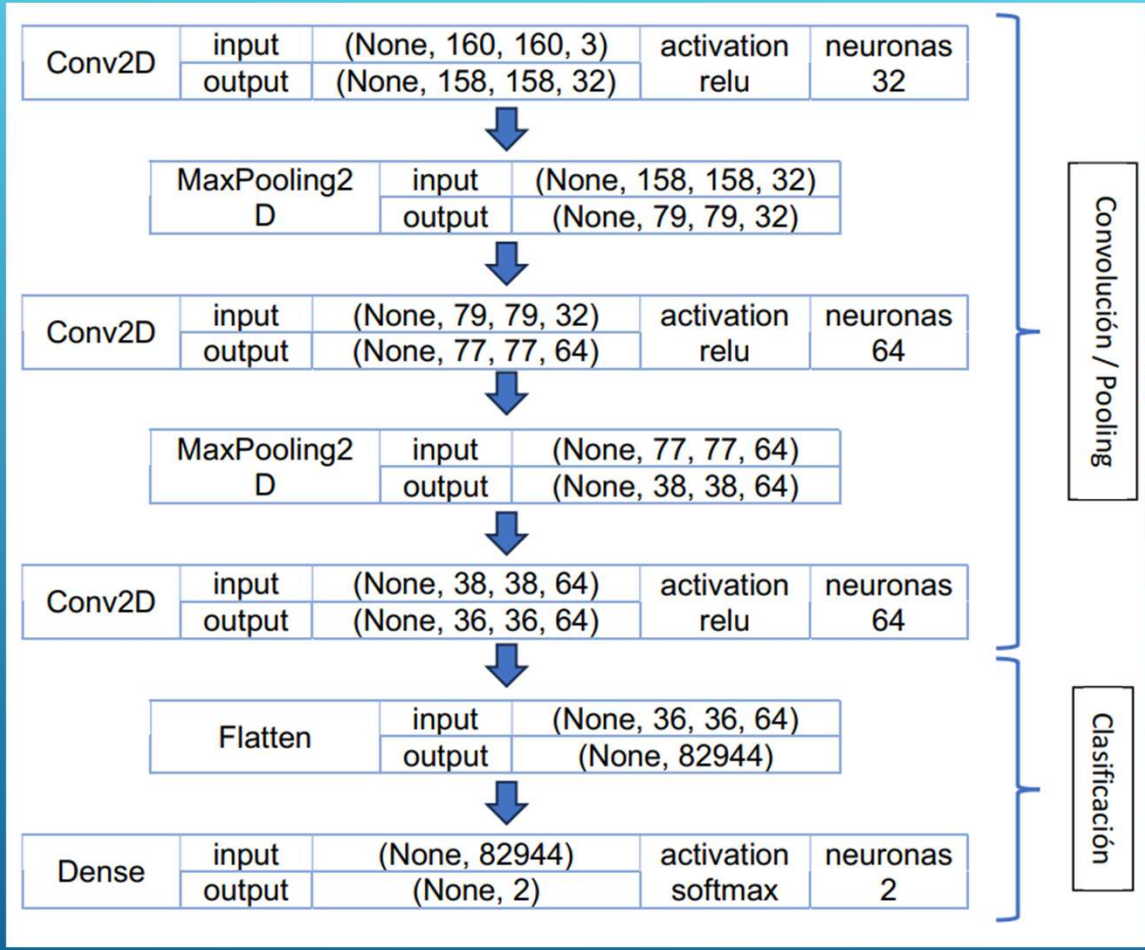
Input Size: 160 x 160
Canales: 3
Model: Sequential

Función de pérdida

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

Optimizador

"adam" porque minimiza la función de pérdida y aprendiendo las características de las imágenes para predicciones más precisas



IMPLEMENTACIÓN EN PYTHON CON KERAS

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
```

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'))
```

ENTRENAMIENTO Y AJUSTE DE PARÁMETROS

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)

```
#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'])
```

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

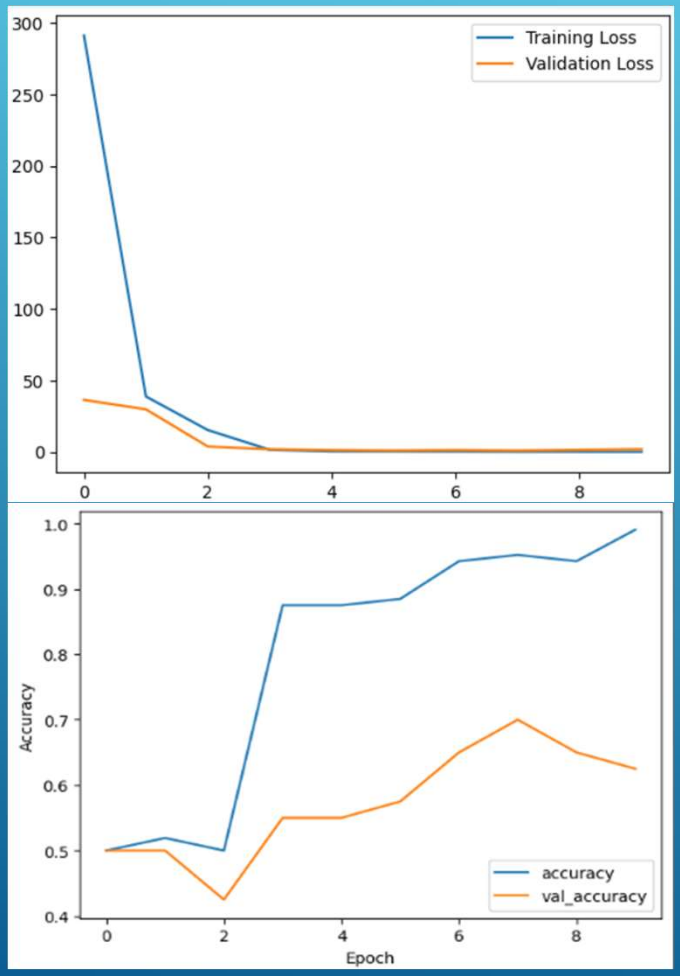
CNN con tres capas convolucionales y capas de max-pooling

```
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)
_________________________________________________________________
```

Uso de binary_crossentropy como función de pérdida y optimizador Adam.

```
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
```

ANÁLISIS DE LOS RESULTADOS



- Loss (pérdida)

  Disminuye significativamente por cada época, indicando que el modelo está mejorando

- Accuracy (Exactitud)

  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

  La pérdida de validación disminuye desde la primera época. La exactitud de validación varía, pero parece estabilizarse en un porcentaje alrededor del 62.5% en la última época.

- Tiempo

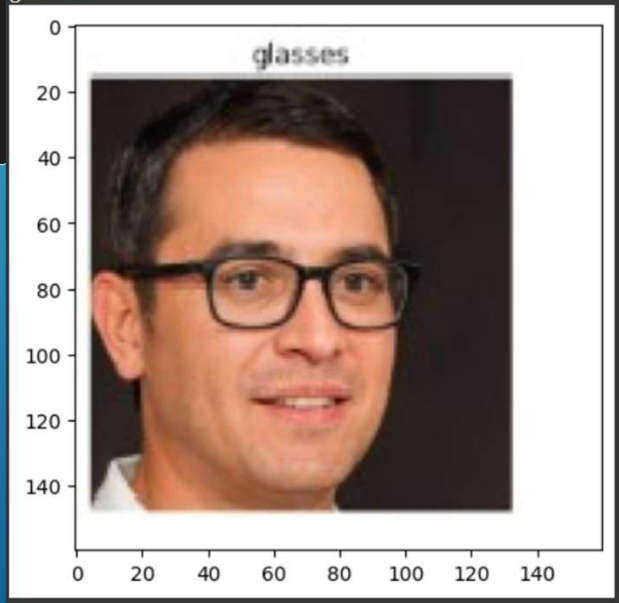
  generalmente oscilan entre 5 y 7 segundos por época

PRUEBAS

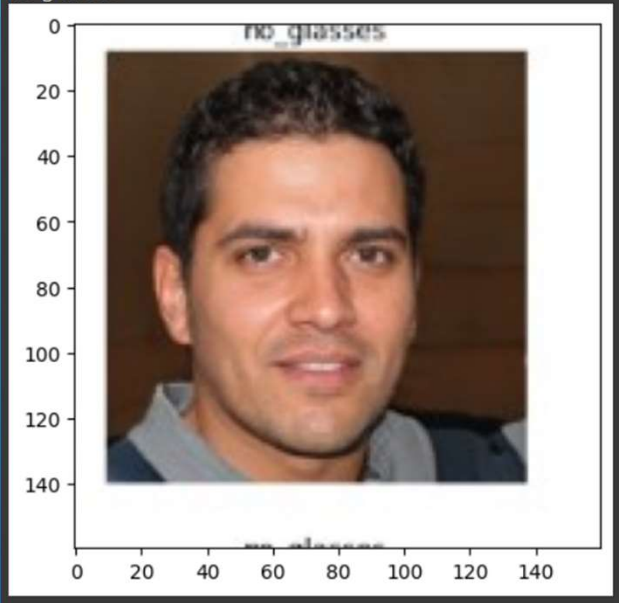
Cargamos imágenes del set de testeo sin clasificar comprobando que el modelo trabaja correctamente:

```
#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')
```

```
1/1 [==============================] - 0s 143ms/step
glasses
```




```
1/1 [==============================] - 0s 114ms/step
no-glasses
```



CONCLUSIONES

- El diseño e implementación del modelo de Red Neuronal Convolucional (CNN) para la clasificación de imágenes de personas con y sin anteojos ha arrojado resultados acordes con las expectativas establecidas
- 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.
- 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