

Detección y Reconocimiento Facial para Evitar Suplantación de Identidad

Guido Anthony Chipana Calderon
Universidad Nacional de Ingeniería, Lima, Perú
guido.chipana.c@uni.pe

Abstract—Este proyecto se centrara en el desarrollo de un sistema de detección y reconocimiento facial para prevenir la suplantación de identidad en diferentes áreas donde se requiera identificarlas, utilizando técnicas de aprendizaje automático.

La implementación estará basada en redes neuronales convolucionales (CNN) y haciendo uso de herramientas de procesamiento de imágenes como OpenCV, y Keras para construir la CNN. El objetivo principal es crear un sistema de autenticación seguro, eficiente y robusto que pueda identificar la identidad de una persona a través de sus características faciales. Y qué metodología seguiremos para realizar con éxito este proyecto?

La metodología que propongo incluye la recolección de datos, el procesamiento de estos datos, la construcción y entrenamiento de la CNN.

Con esto espero que se mejore la precisión en la autenticación facial con respecto a otros modelos ya existentes, así minimizando el riesgo de suplantación de identidad.

Index Terms—OpenCV, Keras, CNN

I. INTRODUCCIÓN

El reconocimiento facial en los últimos años ha tomado gran relevancia en las áreas de seguridad, autenticación, etc. Esto debido a la capacidad de las máquinas para detectar y reconocer rostros humanos de manera eficiente y precisa, sin embargo aún persisten problemas en la precisión y la capacidad de adaptación a diferentes condiciones como la luz, ángulo, expresiones faciales, etc.

Sabemos que existen otros métodos de autenticación como son las contraseñas y tarjetas de identificación, pero el reconocimiento facial no solo mejora la seguridad para evitar suplantación de identidad, sino que también agiliza el proceso de verificación sin necesidad de acordarse de una contraseña o tener a la mano una tarjeta de identificación.

A. Objetivos

- Desarrollar un modelo de red neuronal convolucional (CNN) capaz de detectar y reconocer rostros humanos en diversas condiciones.
- Evaluar qué tan efectivo es el modelo.
- Comparar el rendimiento de mi modelo con otros modelos ya existentes.

Espero que el proyecto y modelo creado sea utilizado en la Universidad Nacional de Ingeniería del Perú, para agilizar y evitar suplantación de identidad en prácticas calificadas o exámenes.

II. FUNDAMENTOS

A. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto diseñada para realizar tareas de procesamiento de imágenes y visión por computadora. En el proyecto se utilizará para preprocessar las imágenes faciales.

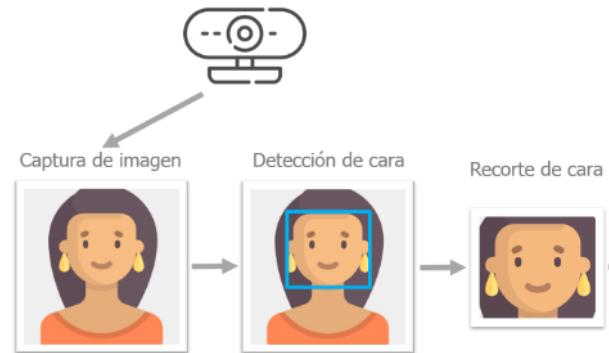


Fig. 1. Ejemplo de procesamiento de imágenes

B. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales o CNN son un tipo de red neuronal que se utiliza principalmente para tareas de clasificación de imágenes y reconocimiento de objetos, están conformadas por capas: Input Layer, Convolutional Layer, Activation Layer, Pooling Layer, Fully Connected Layer, Output Layer.

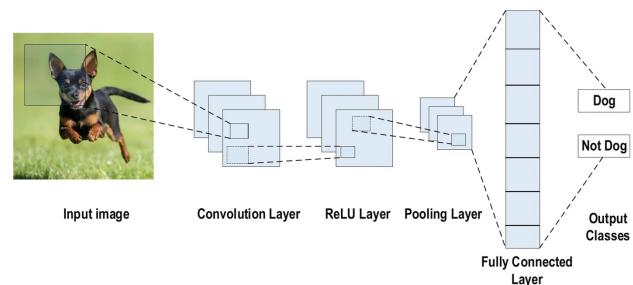


Fig. 2. Ejemplo de red neuronal convolucional

C. Keras

Keras es una biblioteca de código abierto para construir y entrenar modelos de aprendizaje profundo, esta librería nos proporcionará una interfaz para trabajar con redes neuronales.

D. Representación de Imágenes

Existen diferentes modelos y formatos de representación de imágenes, durante este proyecto se verán y usaran 2 tipos de representación de imágenes, los cuales son:

- 1) **Imagenes RGB:** Representan colores utilizando los canales Red (R), Green (G) y Blue (B), donde cada pixel de cada canal tiene valores de 0-255 que indican la intensidad de luz.



Fig. 3. Ejemplo de una imagen representada en RGB

- 2) **Imagenes Grises o Escala de Grises:** Esta representación solo tiene un canal donde cada pixel tiene un valor de 0-255 que indican la intensidad de luz, donde el mínimo representa el negro y el máximo es blanco.



Fig. 4. Ejemplo de una imagen representada en escala de grises

III. ARQUITECTURA DE LAS REDES NEURONALES CONVOLUCIONALES (CNN)

En esta sección veremos a mayor profundidad cada capa de las CNN.

A. Input Layer

Esta capa recibe la imagen, para este proyecto la imagen que se recibirá estará en escala de grises por lo que tendrá solo 1 canal y se le asignará una neurona a cada pixel de la imagen, donde cada pixel tiene un valor de entre 0-255. Por ejemplo en el caso en el que la imagen sea de tamaño $n \times n$, la Input Layer tendrá n^2 neuronas.

B. Convolutional Layer

En esta capa se realiza el proceso de convolución donde la imagen que se ingresa es pasada por varias matrices pequeñas llamadas kernel (por ejemplo kernel de 3×3 o 5×5), cada casilla del kernel tiene un peso la cual va cambiando o ajustando, esto sucede durante el entrenamiento del modelo. Este kernel se desplaza por la imagen "pixel por pixel", multiplicando los valores de los pixeles en la región local de la imagen por los valores del kernel y luego sumando estos productos para obtener un único valor, este proceso lo realizará para todos los kernels, reduciendo el tamaño de la imagen. El resultado de este proceso es un feature map (mapa de características) que refleja la presencia de ciertos patrones en la imagen, como bordes u otros.

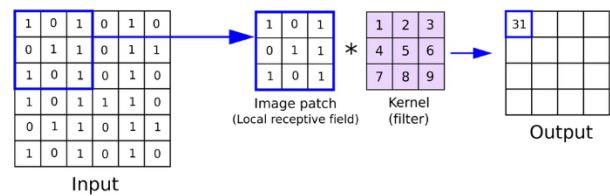


Fig. 5. Ejemplo de convolución en la Convolutional Layer

Otros parámetros importantes dentro de la Convolutional Layer son el stride (paso) y el padding (relleno):

- **stride:** El stride (paso) determinará como se moverá el kernel en la imagen, por ejemplo si el stride es 1, el kernel se moverá un pixel por cada paso y si el stride es 2, el kernel se moverá 2 pixeles por cada paso. Esto hará que la imagen se reduzca de diferente forma dependiendo del stride (paso).

El tamaño de las imágenes de salida de la Convolutional Layer o sea el feature map es:

$$((n - m)/s + 1) \times ((n - m)/s + 1)$$

Donde:

- n es la dimensión de la imagen de entrada.
- m es la dimensión del kernel.
- s es el stride.

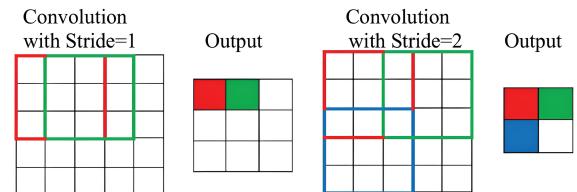


Fig. 6. Ejemplo con stride=1 y stride=2

- **padding:** El padding es el proceso de agregación de capas adicionales de valores a las filas y columnas externas de la matriz de entrada. Existen 2 tipos principales de padding:

- **Valid:** No agregará ninguna capa adicional de cero, o sea sin padding.
- **Same:** Se agregarán capas adicionales de ceros a las filas y columnas externas de la matriz de entrada, para que el feature map mantenga el mismo tamaño de la matriz de entrada.

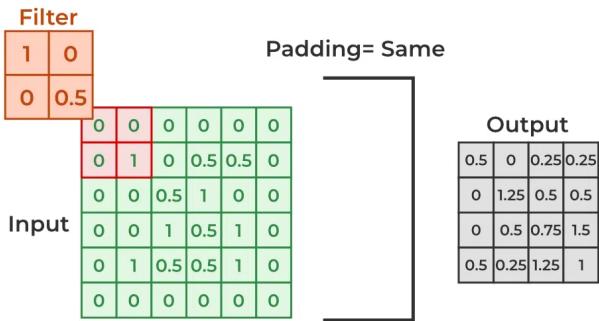


Fig. 7. Ejemplo de padding de tipo same

C. Activation Layer

En esta capa se aplicaran a los valores de entradas Activation Functions (funciones de activacion) con el proposito de introducir la no linealidad en la CNN, esto para que la CNN aprenda relaciones mas complejas.

Estos son los tipos de Activation Functions mas utilizados en la CNN:

- **Sigmoid:** Esta funcion tiene como salida valores entre 0 y 1, se representa de la siguiente manera:

$$f(x)_{\text{sigm}} = \frac{1}{1 + e^{-x}}$$

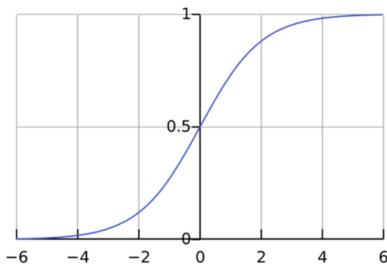


Fig. 8. Sigmoid Function

- **Tanh:** Similar a la sigmoid function con la diferencia de que la salida esta entre -1 y 1, se representa de la siguiente manera:

$$f(x)_{\tanh} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

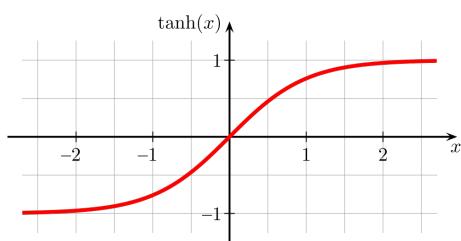


Fig. 9. Hyperbolic Tangent Function

- **ReLU:** Esta es la funcion mas utilizada en las CNN, la salida de esta funcion seran de solamente numero positivos, veamos 2 casos.

En el caso de que el valor de entrada sea negativo, la salida sera 0.

En el caso de que el valor de entrada sea positivo, la salida sera igual a la entrada, osea no cambiara.

Esta funcion se representara de la siguiente manera:

$$f(x)_{\text{ReLU}} = \max(0, x)$$

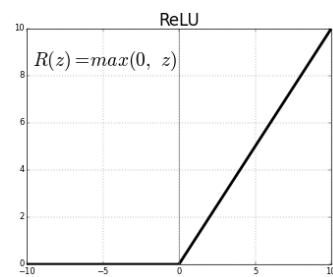


Fig. 10. Function ReLU

- **SoftMax:** Esta funcion convierte los valores de entrada en probabilidades, donde la suma de todas las probabilidades es 1, esta funcion se utilizara en la ultima capa de la Fully Connected Layer, se escribe de la siguiente manera:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

Donde:

- z_i es el valor de entrada para la cual estamos calculando la probabilidad.
- e^{z_j} el exponente de z_i asegurara que los valores resultantes sean positivos.
- $\sum_{j=1}^n e^{z_j}$ esta suma normaliza cada valor para que el resultado este entre 0 y 1, y que la suma total sea 1.

D. Pooling Layer

En esta capa, la entrada sera la salida de la Convolutional Layer, que viene siendo el feature map de cada kernel.

La idea general de esta capa es reducir el tamaño de los feature maps, de tal manera que resuman las caracteristicas que estan en los feature maps, asi reduciendo la cantidad de parametros y calculos en la CNN, haciendo el modelo mas robusto a las variaciones en la posicion de las caracteristicas en la imagen de entrada. Y como hace esto las Pooling Layers?

Esto se hace pasando una matriz llamada pool (comunmente de tamaño 2×2) por los feature maps, el pool no repetira celdas por las que ya paso antes, este nuevo feature map lo llamaremos Pooled Feature Map, entre las formas de pooling existen 2 principales:

- **Max Pooling:** Se tomara el maximo valor dentro del pool.
- **Average Pooling:** Se tomara el promedio de los valores dentro del pool.

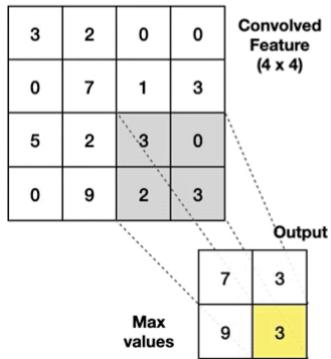


Fig. 11. Ejemplo de Max Pooling

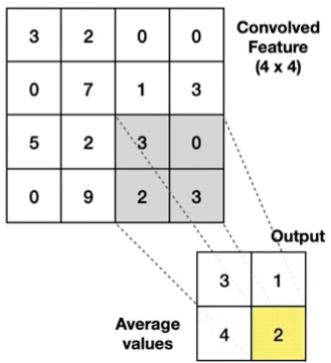


Fig. 12. Ejemplo de Average Pooling

E. Flattening

Este el ultimo proceso para pasar a las Fully Connected Layer, como el Pooled Feature Maps de la ultima Pooling Layer son matrices, necesitamos pasarlo a un vector para que sea compatible con la entrada de la Fully Connected Layer.

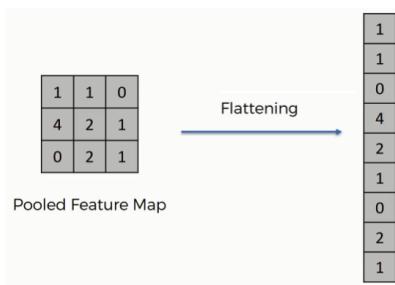


Fig. 13. Ejemplo de Flattening a un Pooled Feature Map

F. Fully Connected Layer

En esta capa se tomara las caracteristicas de la imagen, las procesara para hacer una prediccion final, ahora veamos como hace esto.

Dentro hay varias capas de neuronas llamadas Hidden Layers (Capas Ocultas), cada neurona esta conectada a todas las neuronas de la capa anterior, cada conexion con otra neurona tendra un peso y cada neurona tendra un sesgo, el valor de la siguiente neurona sera la suma ponderada de las neuronas

y sus pesos, mas en detalla vendra siendo el producto de los valores de la neuronas anteriores por el peso de la conexion, agregandole mas el sesgo de la neurona, su representacion matematica seria de la siguiente forma:

$$y_i = \sum_{j=1}^n w_{ij} \cdot x_j + b_i$$

Donde:

- y_i es el valor de la neurona i de la capa actual.
- w_{ij} es el peso de la conexion entre la neurona j de la capa anterior y la neurona i de la capa actual.
- x_j es el valor de la neurona j en la capa anterior.
- b_i vendra siendo el sesgo de la neurona i de la capa actual.

Luego de calcular los valores de las neuronas de la siguiente Hidden Layer, estos valores se pasaran a traves de una Activation Function (Funcion de Activacion), como hemos visto anteriormente en la Activation Layer, la Activation Function introducira no linealidad, permitiendo a la CNN aprender representaciones complejas de los datos.

La ultima capa de la Fully Connected Layer sera donde se produzca la prediccion final de la CNN, en esta capa habra tantas neuronas como clases de imagenes que queremos clasificar o reconocer, donde a cada valor de la neurona se pasara una Activation Function en este caso sera SoftMax, la cual convertira estos valores en valores de probabilidades, donde la suma total de estos valores de probabilidad sera de 1, lo que nos facilitara el proceso de interpretacion de cada valor como la probabilidad de pertenecer a cada clase.

Durante el entrenamiento, estos pesos y sesgos de la Fully Connected Layer iran cambiado/ajustandose mediante el Gradient Descent (Descenso de Gradiente) y Backpropagation (retropropagacion) optimizando o minimizando la Loss Function (Funcion de Perdida), esta Loss Function calculara el error del entrenamiento, este error representara la diferencia entre el resultado real y el previsto por la CNN.

Todo esto con el fin de que nuestro modelo pueda predecir exitosamente que "cosa" es la imagen que se esta mostrando.

IV. LOSS FUNCTION

A. Mean Squared Error

La Mean Squared Error (Error Cuadratico Medio) es una forma de calcular el error del entramiento en cada epoch (epoca), esta se escribe de la siguiente manera:

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Donde:

- n es el numero de muestras en el conjunto de datos.
- y_i es la etiqueta verdadera de la muestra i .
- \hat{y}_i es la salida o valor predicha por el modelo,

B. Mean Absolute Error

La Mean Absolute Error (Error Absoluto Medio) es otra forma de calcular el error de entrenamiento, esta se escribirá de esta forma:

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- n es el numero de muestras en el conjunto de datos.
- y_i es la etiqueta verdadera de la muestra i .
- \hat{y}_i es la salida o valor predicha por el modelo,

C. Binary Cross Entropy

Es una Loss Function utilizada para problemas de clasificación binaria, donde solo hay 2 opciones de clases, esta función se escribe de la siguiente manera:

$$-\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Donde:

- n es el numero de muestras en el conjunto de datos.
- y_i es la etiqueta verdadera de la muestra i (0 o 1).
- \hat{y}_i es la salida o valor predicha por el modelo,

D. Categorical Cross Entropy

Es una Loss Function utilizada principalmente en problemas de clasificación multiclas, donde a contrario a Binary Cross Entropy se tiene mas de dos categorías posibles, esta función se escribe de la siguiente manera:

$$-\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(p_{ij})$$

Donde:

- n es el numero de muestras en el conjunto de datos.
- m es el numero de clases.
- y_{ij} es un valor binario que indica si la muestra i pertenece a la clase j (1 si es cierto o 0 si no).
- p_{ij} es la probabilidad que da el modelo de que la muestra i pertenece a la clase j .

V. LEARNING RATE

El learning rate (tasa de aprendizaje) es un hiperparámetro en el entrenamiento de redes neuronales. Esta determina el tamaño de los pasos que el algoritmo de descenso de gradiente da al actualizar los pesos y sesgos del modelo

VI. BATCHES

Un batch (lote) es un subconjunto de los datos de entrenamiento. Normalmente utilizar todo el conjunto de datos conlleva un alto costo computacional, lo que lleva a un proceso lento tal que retrasa la convergencia. Por esta razón los modelos suelen utilizar mini-batches, que son subconjuntos pequeños del conjunto de datos, para equilibrar la eficiencia computacional.

VII. GRADIENT DESCENT

El Gradient Descent (Descenso de Gradiente) es un algoritmo iterativo que minimiza el error de entrenamiento o Loss Function, este algoritmo actualizará los parámetros de la CNN como los pesos y sesgos, esto para cada epoch de entrenamiento.

Para esto calcularemos la gradiente de la Loss Function aplicando una derivada de primer orden con respecto a un parámetro de la CNN (por ejemplo el peso y el sesgo).

Estos pesos y sesgos se actualizan en dirección inversa del gradiente, el proceso de actualización de parámetros se dará en la Backpropagation de la CNN, en la que la gradiente en cada neurona se propaga hacia atrás a todas las neuronas de la Hidden Layer anterior, este algoritmo iterativo se representa de la siguiente manera para los pesos:

$$w_{ij}^t = w_{ij}^{t-1} - \Delta w_{ij}^t$$

$$\Delta w_{ij}^t = \eta \times \frac{\partial E}{\partial w_{ij}}$$

Donde:

- w_{ij} es el peso de la conexión entre la neurona j de la capa anterior y la neurona i de la capa actual.
- w_{ij}^t es el peso final en la epoch de entrenamiento actual t .
- w_{ij}^{t-1} es el peso en la epoch de entrenamiento anterior ($t - 1$).
- η es la learning rate (tasa de aprendizaje) que se define como el tamaño del paso de la actualización del parámetro en este caso el peso.
- E es la Loss Function.

En el caso de los sesgos es similar:

$$b_i^t = b_i^{t-1} - \Delta b_i^t$$

$$\Delta b_i^t = \eta \times \frac{\partial E}{\partial b_i}$$

Donde:

- b_i es el sesgo de la neurona i de la capa actual.
- b_i^t es el sesgo final en la epoch de entrenamiento actual t .
- b_i^{t-1} es el sesgo en la epoch de entrenamiento anterior ($t - 1$).
- η es la tasa de aprendizaje que se define como el tamaño del paso de la actualización del parámetro en este caso el sesgo.
- E es la Loss Function.

VIII. BACKPROPAGATION

La Backpropagation (Retropropagación) es un algoritmo que se utiliza para entrenar una CNN, actualiza los pesos y sesgos para minimizar la Loss Function para mejorar el rendimiento de nuestra CNN.

Lo podemos separar en 3 pasos, el forward pass (pase hacia adelante), el loss calculation (cálculo de pérdida o error) y el backward pass (pase hacia atrás).

A. Forward Pass

Durante este paso los datos de entrada, en este caso imagenes, pasaran atraves de la CNN para determinar la salida, como ya vimos esta imagen de entrada pasara por las Convolutional Layers, Activation Layers, Pooling Layers y Fully Connected Layers.

B. Loss Calculation

Despues de obtener la salida de la CNN, se obtiene una Loss Function la cual calculara el error de entranamiento de esa epoch, podemos utilizar la Loss Function que mas nos convengan, vemos que en el caso de la clasificacion de imagenes de multiclase, viene bien utilizar la Categorical Cross Entropy.

C. Backward Pass

Para hacer el Backward Pass, calcularemos el gradiente de la Loss Function con respecto a cada peso en la CNN, veremos que comenzaremos de adelante hacia atras y aplicaremos la regla de la cadena.

- **Gradiente con ResPECTO a la SalIDA:** Primero se calculara el gradiente de la loss function con respecto a la salida de la capa final (\hat{y}):

$$\frac{\partial E}{\partial \hat{y}}$$

- **Gradiente a traves de la activation function:** Luego este gradiente se propagara a traves de la activation function, usaremos la regla de la cadena a traves de la activation function.

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x}$$

Donde x es el valor de entrada de la activation function, que es el valor que llega a la neurona antes de aplicarle la activation function.

- **Gradiente con respecto a los kernels en convolutional layer:** Continuaremos aplicando la regla de la cadena para encontrar la gradiente con respecto a los kernels.

$$\frac{\partial E}{\partial K} = \frac{\partial E}{\partial \hat{y}} * \text{rot180}(I)$$

Donde:

- **K** es el kernel.
- **I** es la entrada a la convolutional layer, es el valor que recibe la convolutional layer.
- **rot180(I)** indica que la entrada I se rota 180 grados antes de hacer la convolucion inversa. Esto sera necesario para alinear el calculo del gradiente de la misma forma en que se aplico la convolucion durante la propagacion hacia adelante.

Al aplicar este gradiente se ajustaran los valores de cada kernel en la convolutional layer, lo que va a permitir que los kernels se adapten para identificar mejor las caracteristicas de la imagen que se ingresa.

Estos tres procesos se realizara varias veces en bucle hasta tener bien ajustado los parametros de la CNN para identificar imagenes.

IX. METODOLOGIA

En esta seccion se describirá el proceso para desarrollar nuestro modelo con CNN.

A. Dataset

Para este proyecto se uso el dataset de VGGFace2, esta tiene 540 personas con un total de 198k imagenes, el dataset se obtuvo de la plataforma de Kaggle. Para probar el modelo me agregue a mi mismo en el dataset, para esto se uso un codigo en python en la que me toma 400 fotos y lo agregar al dataset en una carpeta con mi nombre. Al final se uso una version reducida de VGGFace2 por el alto costo de recursos en el proceso de desarrollo del modelo. Esta version reducida la cual la llamamos valVGGFace2 contiene 3720 imagenes entre 12 personas, en la cual una de estas personas soy yo.

B. Preprocesamiento de Datos

Preprocesaremos los datos del dataset valVGGFace2 para que el modelo reciba una estructura uniforme de los datos, eso se hace con el fin de mejorar la calidad y eficiencia del entrenamiento del modelo, seguiremos los siguientes pasos:

- **Detección de rostros:** Usaremos el algoritmo Haar Cascade, específicamente "haarcascade_frontalface_default.xml" para detectar, localizar y recortar los rostros en la imágenes, para centrarnos solamente en los rostros de las personas esto para que el modelo enfoque su aprendizaje únicamente en las características faciales de las personas.
- **Conversion a escala de grises:** Transformaremos las imágenes a escala de grises para reducir la cantidad de canales, pasaremos de 3 canales (RGB) a 1 canal en escala de grises. esto para minimizar la carga computacional.
- **Redimensionamiento:** Redimensionaremos todas las imágenes detectadas a un tamaño fijo, para nuestro proyecto, las imágenes se redimensionaron a 224x244 pixeles, esto para que el modelo reciba entradas con dimensiones homogéneas.
- **Normalización:** Normalizaremos cada imagen del dataset, pasaremos los valores de los pixeles de las imágenes a un rango de [0, 1], esto lo haremos dividiendo cada valor del pixel por 255, esto para que los datos que reciban el modelo estén en una escala uniforme y no muy dispersos. Acelerando la convergencia durante el entrenamiento del modelo.

C. Desarrollo del modelo con CNN

Se desarrollara el modelo usando CNN, el modelo seguirá la siguiente estructura:

1) Input Layer y primera Convolution Layer

```
modelo = Sequential([
    # Primera capa convolucional
    Conv2D(32, (3, 3), activation='relu',
           input_shape=(224, 224, 1)),
    BatchNormalization(),
    • Numero de Kernels: 32
```

- Tamaño del kernel: 3x3
 - Activation Fuction: ReLU
 - Input Shape: 224x224x1
 - Se aplico Batch Normalization
- 2) **Primera Maxpooling Layer**
`# Primera capa maxpooling
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.25),
• Tamaño del pool: 2x2
• Se aplico Dropout con una probabilidad del 25%`
- 3) **Segunda Convolution Layer**
`# Segunda capa convolucional
Conv2D(64, (3, 3),
activation='relu'),
BatchNormalization(),
• Numero de Kernels: 64
• Tamaño del kernel: 3x3
• Activation Fuction: ReLU
• Se aplico Batch Normalization`
- 4) **Segunda Maxpooling Layer**
`# Segunda capa maxpooling
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.25),
• Tamaño del pool: 2x2
• Se aplico Dropout con una probabilidad del 25%`
- 5) **Tercera Convolution Layer**
`# Tercera capa convolucional
Conv2D(128, (3, 3),
activation='relu'),
BatchNormalization(),
• Numero de Kernels: 128
• Tamaño del kernel: 3x3
• Activation Fuction: ReLU
• Se aplico Batch Normalization`
- 6) **Tercera Maxpooling Layer**
`# Tercera capa maxpooling
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.25),
• Tamaño del pool: 2x2
• Se aplico Dropout con una probabilidad del 25%`
- 7) **Cuarta Convolution Layer**
`# Cuarta capa convolucional
Conv2D(256, (3, 3),
activation='relu'),
BatchNormalization(),
• Numero de Kernels: 256
• Tamaño del kernel: 3x3
• Activation Fuction: ReLU
• Se aplico Batch Normalization`
- 8) **Cuarta Maxpooling Layer**
`# Cuarta capa maxpooling
MaxPooling2D(pool_size=(2, 2)),
Dropout(0.25),
• Tamaño del pool: 2x2
• Se aplico Dropout con una probabilidad del 25%`
- 9) **Flattening**
`# Aplanado (Flatten)
Flatten()`
- 10) **Primera Fully Connected Layer**
`# Primera capa completamente
conectada (256 unidades)
Dense(256, activation='relu'),
BatchNormalization(axis=1),
Dropout(0.5),
• Numero de neuronas: 256
• Activation Fuction: ReLU
• Se aplico Batch Normalization
• Se aplico Dropout con una probabilidad del 50%`
- 11) **Segunda Fully Connected Layer**
`# Segunda capa completamente
conectada (200 unidades)
Dense(200, activation='relu'),
BatchNormalization(axis=1),
Dropout(0.5),
• Numero de neuronas: 256
• Activation Fuction: ReLU
• Se aplico Batch Normalization
• Se aplico Dropout con una probabilidad del 50%`
- 12) **Tercera Fully Connected Layer**
`# Tercera capa completamente
conectada (150 unidades)
Dense(150, activation='relu'),
BatchNormalization(axis=1),
Dropout(0.5),
• Numero de neuronas: 256
• Activation Fuction: ReLU
• Se aplico Batch Normalization
• Se aplico Dropout con una probabilidad del 50%`
- 13) **Cuarta Fully Connected Layer**
`# Cuarta capa completamente conectada
(100 unidades)
Dense(100, activation='relu'),
BatchNormalization(axis=1),
Dropout(0.5),
• Numero de neuronas: 256
• Activation Fuction: ReLU
• Se aplico Batch Normalization
• Se aplico Dropout con una probabilidad del 50%`
- 14) **Output Layer**
`# Capa de salida (Softmax para
clasificación)
Dense(num_classes,
activation='softmax')
]
• Numero de neuronas: Numero de clases, para este
proyecto son 12 clases
• Activation Fuction: Softmax (para clasificacion multi-categoría)`

D. Entrenamiento del modelo

Para el entrenamiento, compilaremos el modelo de la siguiente forma:

```

modelo.compile(
optimizer='adam',
loss=Categorical_Cross_Entropy
_with_Penalization,
metrics=['accuracy'])
• Optimizador: Adam
• Loss Function: Categorical Cross Entropy with Penalization
• Metricas: Accuracy

```

Utilizaremos una variante de Categorical Cross Entropy, ya que esta penaliza cuando se ingresa una imagen que no pertenece a ninguna clase, esta nueva variante la llamaremos Categorical_Cross_Entropy_with_Penalization la cual se definira de la siguiente manera:

```

def Categorical_Cross_Entropy
_with_Penalization(y_true, y_pred):
# Entropia cruzada estandar
perdida_base = K.categorical_crossentropy
(y_true, y_pred)

# Calculo de la entropia de las
predicciones
entropia_pred = -K.sum(y_pred *
K.log(y_pred + K.epsilon()), axis=-1)

# Factor de peso
fac_peso = 0.1

# Penalizacion adicional
penalizacion = fac_peso * entropia_pred

# Combinar perdida base y penalizacion
return perdida_base + penalizacion

```

- perdida_base: Es la perdida de la entropia cruzada estandar
- entropia_pred: Calcula la entropia de las predicciones, una entropia alta indica que el modelo no esta seguro de sus predicciones
- fac_peso: Es el factor de peso que escala la influencia de la penalizacion sobre la perdida base. Un valor mas alto da mayor importancia a la penalizacion, mientras que un valor mas bajo la reduce. En este caso, el factor es 0.1.
- penalizacion: Termino adicional que castiga al modelo cuando sus predicciones son inciertas.

La Loss Function Categorical Cross Entropy with Penalization no solo medira el error estandar como lo hace Categorical Cross Entropy, sino que tambien penalizara las predicciones inciertas, esto nos ayudara cuando una imagen no pertenece a ninguna clase.

Ahora continuaremos entrenando el modelo con el dataset y estructura ya antes definida.

```

history = modelo.fit(
X_train, Y_train,
validation_data=(X_val, Y_val),
epochs=100,
batch_size=32,
verbose=1,

```

```

 callbacks=[early_stopping]
)
• Numero de epochs: 100
• Tamaño del batch: 32
• callbacks: Llamara a early_stopping al final de cada epoch, esta sera nuestro criterio de parada.

```

La funcion early_stopping se definira de la siguiente manera:

```

early_stopping = EarlyStopping(
monitor='val_loss',
patience=10,
restore_best_weights=True
)

```

- monitor: Especifica que metrica observara, en este caso 'val_loss'.
- patience: Vera si despues de 10 epoch consecutivas no mejora la metrica definida en monitor antes de pasar por todas las epoch, si es asi detendra el entrenamiento.
- restore_best_weights: El modelo restaurara automaticamente los pesos que lograron la mejor metrica monitoreada, en este caso el mejor 'val_loss'.

X. RESULTADOS

En esta seccion los resultados obtenidos durante todo el proceso del proyecto.

A. Resultados del entrenamiento del nuestro modelo en CNN

El modelo se entreno con un maximo de 100 epochs, el resultado es el siguiente:

```

Epoch 1/100
93/93 ----- 31s 123ms/step
- accuracy: 0.1167 - loss: 3.6137 -
val_accuracy: 0.0995 - val_loss: 10.3431
Epoch 2/100
93/93 ----- 6s 62ms/step -
accuracy: 0.2319 - loss: 2.7358 -
val_accuracy: 0.0995 - val_loss: 13.9541
Epoch 3/100
93/93 ----- 6s 65ms/step -
accuracy: 0.3537 - loss: 2.1904 -
val_accuracy: 0.0995 - val_loss: 11.3104
Epoch 4/100
93/93 ----- 10s 62ms/step
- accuracy: 0.3947 - loss: 1.9785 -
val_accuracy: 0.1169 - val_loss: 8.8595
Epoch 5/100
93/93 ----- 6s 67ms/step -
accuracy: 0.4689 - loss: 1.7284 -
val_accuracy: 0.1935 - val_loss: 3.0669

```

Continua...

```

Epoch 35/100
93/93 ----- 6s 64ms/step -
accuracy: 0.8821 - loss: 0.4403 -
val_accuracy: 0.4180 - val_loss: 2.1743
Epoch 36/100

```

```
93/93 ----- 11s 68ms/step
- accuracy: 0.8694 - loss: 0.4735 -
val_accuracy: 0.4341 - val_loss: 2.7962
```

Al final el entrenamiento se corta en la epoch 46 ya que hace 10 epochs la metrica val_loss no mejora, esto se hace con el fin de evitar el sobreajuste o sobreentrenamiento. El proceso de entrenamiento de nuestro modelo duro 4 minutos y 56.07 segundos

B. Evaluacion del modelo

Los resultados del nuestro modelo en el conjunto de validacion son los siguientes:

- Accuracy: 0.8642
- Loss: 0.4905

C. Pruebas del modelo en tiempo real

Para este proyecto tambien queremos identificar a una persona que no esta dentro del dataset de entreamiento como personas desconocida, para esto definimos una cota en la que si la probabilidad que nos retorna el modelo es mayor a esta entonces es una personas conocida, y en el caso que se menor a la cota esta sera una personas desconocida. La cota que se define en un programa de pruebas es del 0.87.

1) *Pruebas con personas conocidas:* Probaremos conmigo mismo si me identifica de forma correcta.

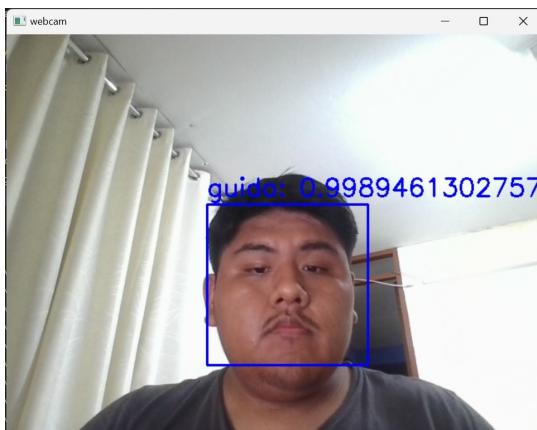


Fig. 14. Prueba con guido da 0.99 de probabilidad

2) *Pruebas con personas desconocidas:* Probaremos con una persona que no pertenece a ninguna clase.

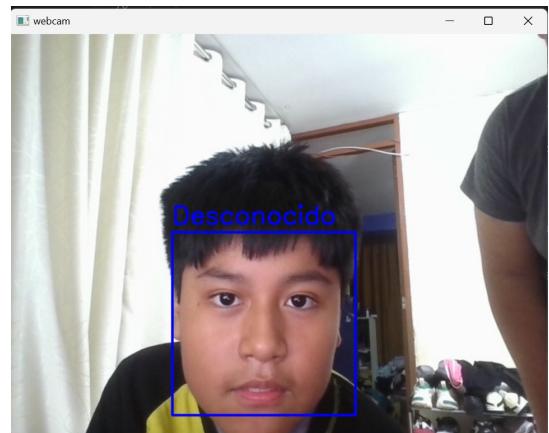


Fig. 15. Prueba con persona desconocida

3) *Pruebas con mas de 2 personas:* Probaremos con mas de 2 personas:

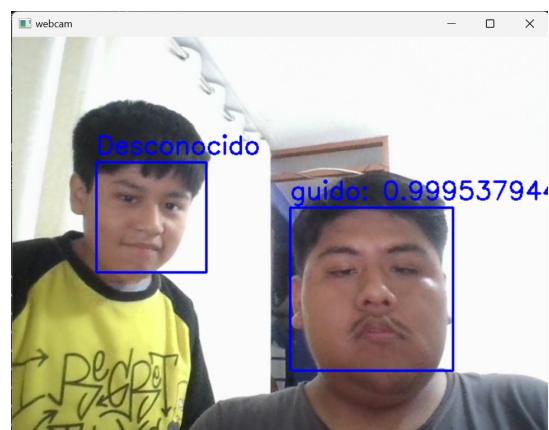


Fig. 16. Prueba con 2 personas

Vemos que si identifica de forma correcta a las personas conocidas y desconocidas.

4) *Problemas:* Hay momentos en el que no identifica de forma correcta a la persona que esta adelante, por ejemplo:

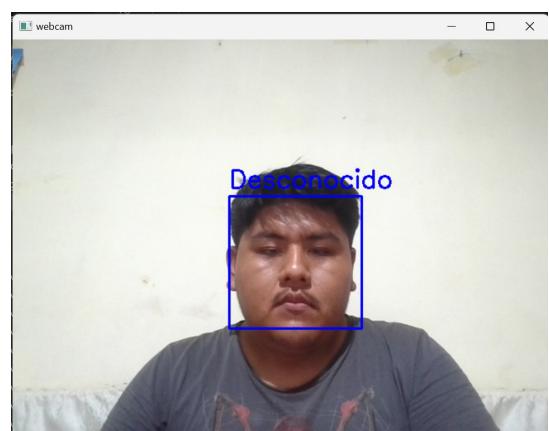


Fig. 17. Identifica a guido como desconocido



Fig. 18. Identifica a guido como otra persona

Vamos a ver que los problemas suceden cuando nos alejamos de la camara, normalmente el modelo funciona bien si nos acercamos a la camara.

REFERENCES

- [1] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data* 8, 53 (2021). [Link](#)
- [2] Zhang, Z. (2016). Derivation of Backpropagation in Convolutional Neural Network (CNN). University of Tennessee, Knoxville, TN. [Link](#)
- [3] Deepgram. (2024, 16 de junio). Loss Function. Recuperado de [Link](#)
- [4] GeeksforGeeks. (2024, 8 de septiembre). Backpropagation in Convolutional Neural Networks. Recuperado de [Link](#)
- [5] Purwono, Purwono and Ma'arif, Alfian and Rahmani, Wahyu and Imam, Haris and Fathurrahman, Haris Imam Karim and Frisky, Aufaclav and Haq, Qazi Mazhar Ul. (2023). Understanding of Convolutional Neural Network (CNN): A Review. [Link](#)
- [6] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. *Insights Imaging* 9, 611–629 (2018). [Link](#)

XI. DISCUSIONES

- Durante las pruebas en tiempo real se vio que aveces a una persona que no pertenece a ninguna clase o sea un desconocido, la probabilidad que se obtiene del modelo supera la cota de mayor a 0.87, por lo que llega a fallar.
- Durante el preprocesamiento del dataset, se vio que utilizar todo el dataset VGGFace2 consume mucha memoria ram, los 12GB de ram proporcionado por Google Colab no eran suficientes, por lo que para preprocesar un dataset con muchas clases e imagenes conlleva tener una maquina con mucha memoria ram.

XII. CONCLUSIONES

- Se logro desarrollar un modelo propio de CNN, con un Accuracy del 86.42% y un Loss del 49.05%.
- No se logro el objetivo de comparar con otros modelos ya existentes.
- Usar el dataset VGGFace2 completo conlleva un alto consumo de memoria ram, por lo que se redujo el tamaño a 3720 imagenes entre 12 personas.
- El modelo presento un rendimiento regular en pruebas en tiempo real, ya que aveces identificaba a personas desconocidas como conocidas, no es 100% preciso.
- El modelo deja de funcionar correctamente cada vez que nos alejamos mas de la camara.
- Este modelo podria ser aplicado en aplicaciones practicas como en sistemas de autenticacion facial, monitoreo en tiempo real y analisis de datos biometricos, etc.

XIII. TRABAJOS FUTUROS

- Mejorar el rendimiento del modelo creado.
- Utilizar mas datos para crear un modelo mas robusto.
- Mejorar la cota mediante pruebas en el modelo para evitar mas errores.
- Comparar el modelo con otros modelos ya existentes.