Dataset FANE:
https://www.kaggle.com/datasets/furcifer/fane-facial-expressions-and-emotion-dataset
Este dataset contiene imágenes recolectadas con Web scraping de las emociones:
enojo, miedo, feliz, triste, sorprendido.

Pre-procesamiento

1. Normalizar brillo

Esta parte del procesamiento la hicimos al darnos cuenta de que la iluminación era inconsistente a través del dataset. Es decir, existían imágenes muy brillantes e imágenes muy oscuras y era necesario acercarlas más.
Para esto creamos un método:

```python
##func normalizar brillo
def normaliza_brillo(img, target_mean=128):
    """Normaliza la luminosidad de la imagen a un valor medio objetivo"""
    # Convertir a YCrCb (Y es el canal de luminancia)
    ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    y, cr, cb = cv2.split(ycrcb)

    # Calcular el valor medio actual de Y
    current_mean = np.mean(y)

    # Ajustar el brillo
    y = np.clip(y * (target_mean / current_mean), 0, 255).astype(np.uint8)

    # Fusionar los canales y convertir de vuelta a BGR
    ycrcb = cv2.merge([y, cr, cb])
    return cv2.cvtColor(ycrcb, cv2.COLOR_YCrCb2BGR)
```

Primero, la imagen se convierte del formato BGR (el formato que usa OpenCV) al formato YCrCb. Este formato contiene las variables y=luminancia, Cr=crominancia roja, Cb=crominancia azul.
Una vez obtenemos la imagen en este formato, podemos obtener la media del arreglo de luminancia (guardado en la variable y) usando numpy.
Después, se ajusta el brillo haciendo operaciones sobre la matriz de luminancia multiplicando para que las imágenes más oscuras se vuelven más brillantes y al revés.
El método clip permite que el brillo no se pase de 255 y luego lo convertimos a un entero de 8 bits.

Por último, se vuelve a armar la imagen y no se transforma a otro formato para conservar los cambios.

2. Redimensionar (con re-sampling)

```python
#Redimensionar
img = cv2.resize(img, tamanio_objetivo, interpolation=cv2.INTER_CUBIC)
```

Al momento de estandarizar las imágenes al mismo tamaño encontramos dificultades, ya que ciertas imágenes tenían una resolución mayor o menor, y relaciones de aspecto diferentes. Para esto experimentamos con diferentes tamaños como la proporción áurea para las caras, pero nos decidimos por 224x224 ya que es divisible entre 32 para entrenar el modelo en tensorflow de manera eficiente.

Aplicamos re-sampling para todas las imágenes. El método de interpolación (como se maneja la información perdida o la información ganada) que elegimos fue Bicúbica, que suaviza las imágenes de menor resolución para no tener imágenes finales con píxeles visibles. Además, es un método que consideramos un punto medio ya que el método del vecino mas cercano nos resultó en imágenes feas y el método Lanczos4 es más lento.

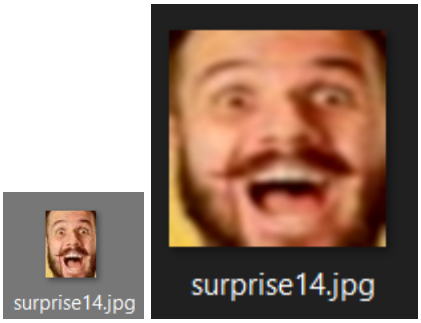
3. Aplicar blur a imágenes upscaled

```
#Borrosear
if(height<50 and width<50):
    img = cv2.GaussianBlur(img, (19, 19), sigmaX=0)
elif(height<75 and width<75):
    img = cv2.GaussianBlur(img, (17, 17), sigmaX=0)
elif(height<100 and width<100):
    img = cv2.GaussianBlur(img, (15, 15), sigmaX=0)
elif(height<125 and width<125):
    img = cv2.GaussianBlur(img, (13, 13), sigmaX=0)
elif(height<150 and width<150):
    img = cv2.GaussianBlur(img, (9, 9 ), sigmaX=0)
elif(height<224 and width<224):
    img = cv2.GaussianBlur(img, (5, 5), sigmaX=0)
```

Reconocimos que las imágenes que solían ser pequeñas (50x50, por ejemplo) y que fueron transformadas a el nuevo tamaño 224x224 perdían mucha información. Visiblemente se podían observar ciertos pixeles, así que para esto decidimos aprovechar estas imágenes que perdieron un poco de calidad en el producto final como una muestra de imagen desenfocada.

Lo que significa esto es que aprovechamos las imágenes con información inconsistente y aplicamos un pequeño desenfoque en la imagen para que estas se puedan usar para reconocer también emociones aunque la cámara desenfoque un poco.

Lo único que hace es aplicar un blur Gaussiano a las imágenes más pequeñas.



4. Normalizar contraste con CLAHE

```python
#Normalizar contraste
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
if gray.std() < 30:
    try:
        img_ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
        clahe = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(8, 8))
        img_ycrcb[:, :, 0] = clahe.apply(img_ycrcb[:, :, 0])
        img = cv2.cvtColor(img_ycrcb, cv2.COLOR_YCrCb2BGR)
    except Exception as e:
        print(f"Error aplicando CLAHE en {file}: {e}")
```

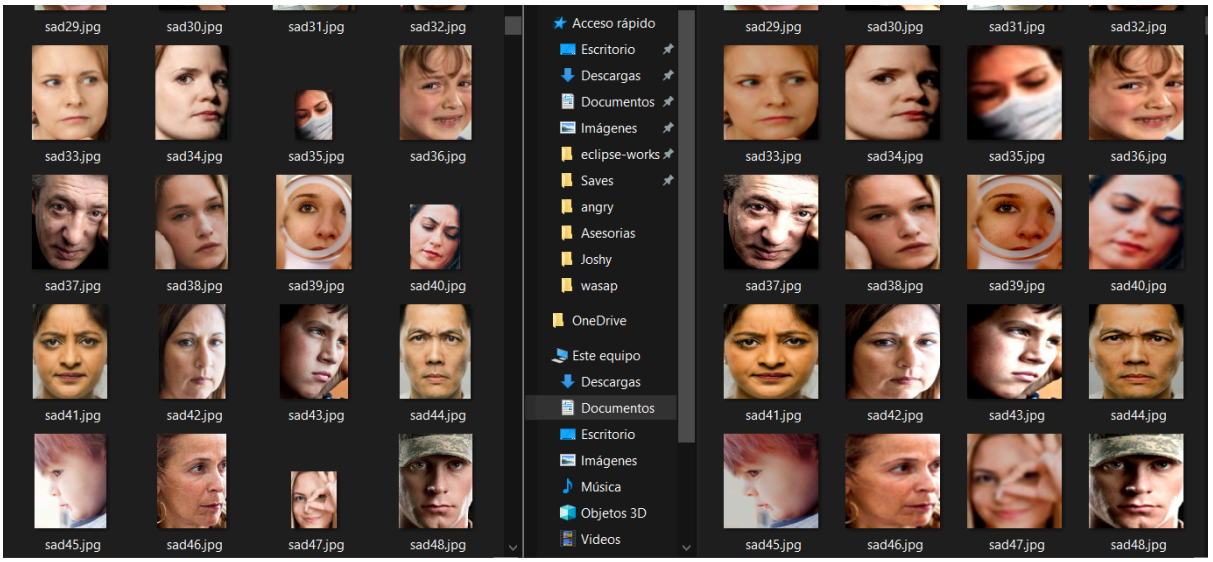
En este método se calcula la desviación estándar de los valores en los píxeles de la imagen, si se encuentra una imagen con menos de 30 de contraste se aplica CLAHE (Contrast Limited Adaptive Histogram Equalization). Que nos sirve para amplificar el contraste en imágenes que contienen poco detalle.

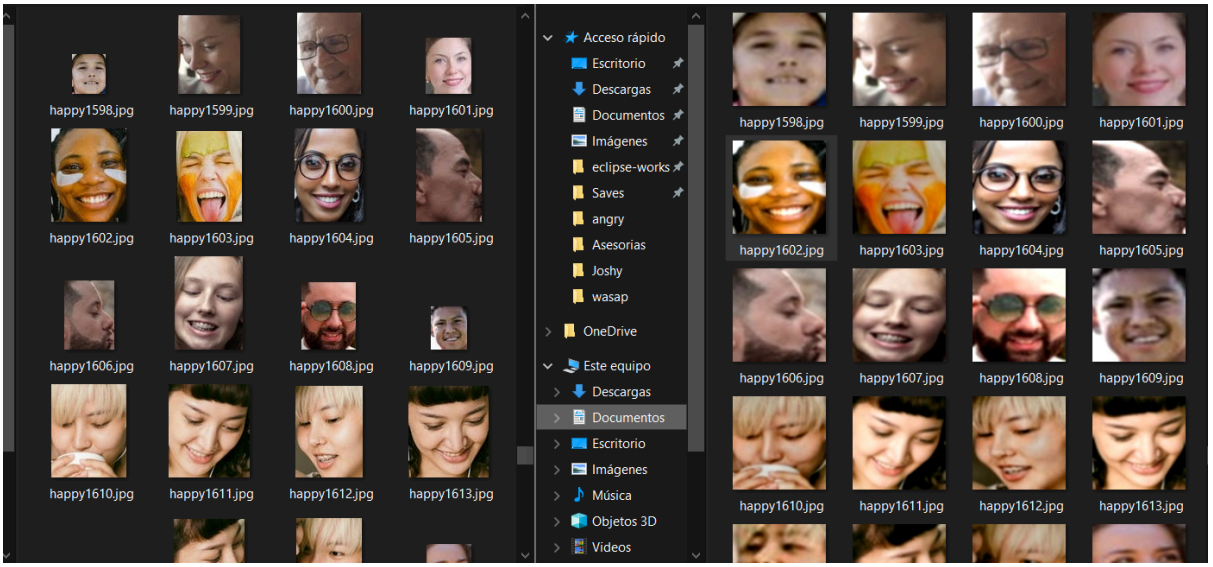
5. Aumentar contraste de manera lineal

```python
def aumentar_contraste(img, alpha):
    img_float = img.astype(np.float32) / 255.0

    # Aplicar transformación lineal en cada canal (B, G, R)
    img_contrastada = np.clip(alpha * (img_float - 0.5) + 0.5, 0, 1)

    img_contrastada = (img_contrastada * 255).astype(np.uint8)

    return img_contrastada
```

Por último, se aplica un aumento de contraste a cada pixel, multiplicamos cada pixel del arreglo de la imagen por alpha. Para esto es necesario restar 0.5 (la mitad del valor) para trabajar desde el contraste bajo y evitar pasarnos del objetivo de contraste. Se usa clip para que el valor no supere 1 (el máximo).

6. Resultados




happy1598.jpg
happy1599.jpg
happy1600.jpg
happy1601.jpg
happy1602.jpg
happy1603.jpg
happy1604.jpg
happy1605.jpg
happy1606.jpg
happy1607.jpg
happy1608.jpg
happy1609.jpg
happy1610.jpg
happy1611.jpg
happy1612.jpg
happy1613.jpg
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
eclipse-works
Saves
angry
Asesorias
Joshy
wasap
OneDrive
Este equipo
Descargas
Documentos
Escritorio
Imágenes
Música
Objetos 3D
Videos
happy1598.jpg
happy1599.jpg
happy1600.jpg
happy1601.jpg
happy1602.jpg
happy1603.jpg
happy1604.jpg
happy1605.jpg
happy1606.jpg
happy1607.jpg
happy1608.jpg
happy1609.jpg
happy1610.jpg
happy1611.jpg
happy1612.jpg
happy1613.jpg