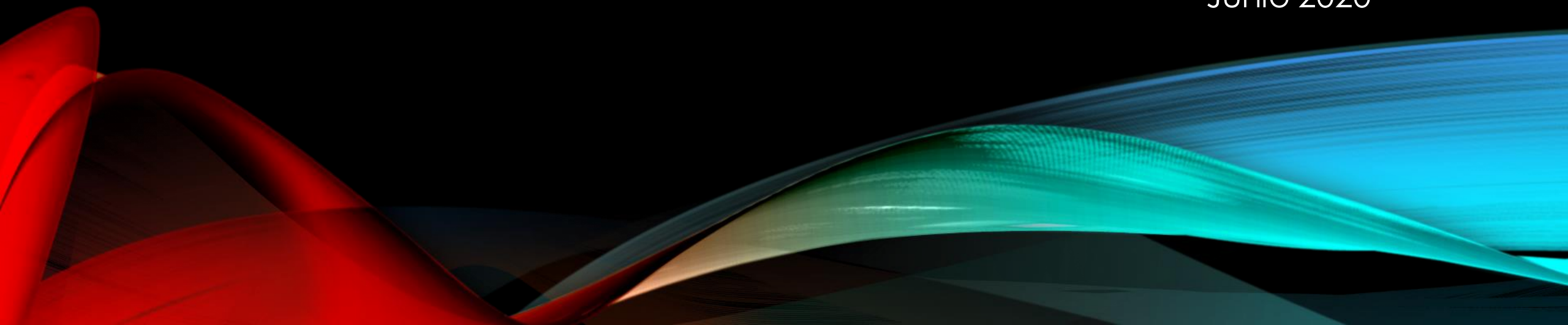
COVID-19: DETECTOR DE MASCARILLA

Henryk Zorrilla Mori
NEOLAND SCHOOL
Junio 2020



DESCRIPCIÓN DEL PROYECTO

- Después de haber superado esta Pandemia, ahora nos toca volver a la normalidad, tomando eso sí, unas ciertas medidas de seguridad para evitar un posible rebrote.
- Una de estas medidas de seguridad es salir a la calle con mascarilla. Para una persona es fácil reconocer al resto de personas solo con fijarse en los ojos, mirada, el peinado, etc.
- Por el contrario, no ocurre lo mismo con los sistemas de reconocimiento facial. Esta situación ha llevado a los algoritmos a intentar adaptarse a esta nueva etapa, ya que algunos son inefectivos cuando hay algo tapando parte del rostro, en este caso una mascarilla.
- El objetivo de este proyecto es crear una red neuronal (deep learning) para poder detectar a las personas cuando llevan mascarillas y cuando no.

OBJETIVO GENERAL

- Creación de un modelo de red neuronal convolucional (CNN), usando la biblioteca TensorFlow con Keras
- Usar OpenCV para detectar si una persona está usando mascarilla o no a través de una imagen.
- En caso de un rebrote, un buen modelo de deteccion de mascarilla podría ayudar a localizar el foco, localizando a las personas afectadas cuando no han estado usando la mascarilla como medida de seguridad.

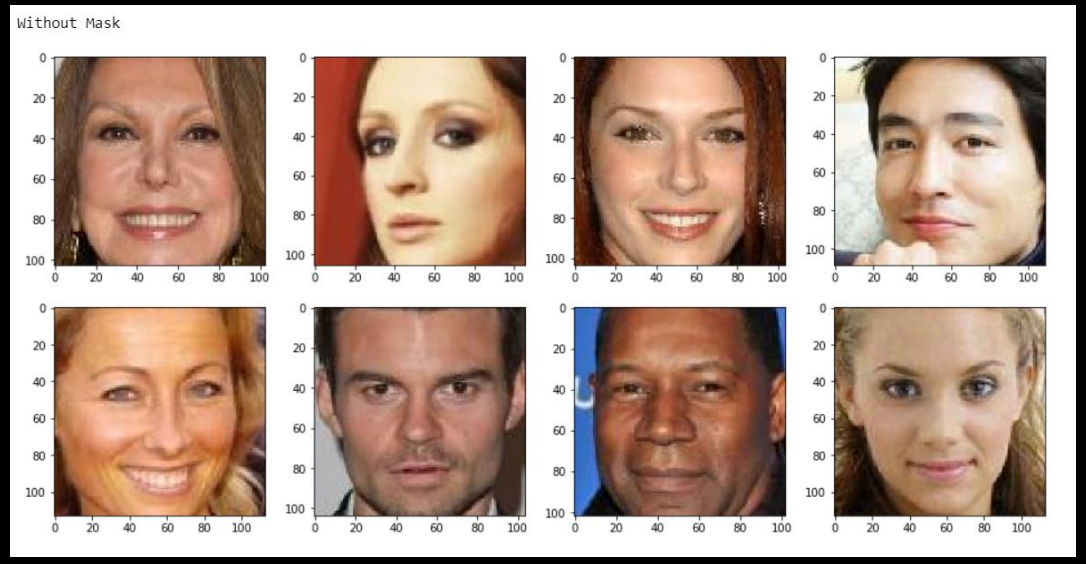
ETAPAS DEL PROYECTO

- 1. Recopilación de datos y creación del dataset
- 2. Construcción del modelo CNN secuencial con varias capas similar al modelo VGG-16
- 3. Pre-entranmiento del modelo
- 4. Entrenamiento del modelo
- 5. Predicción del modelo
- 6. Testear el modelo
- 7. Carga del OpenCV y uso del reconocimiento facial

1. RECOPILACION DE DATOS Y CREACIÓN DEL DATA SET

Clasificación de las imágenes en dos categorias (con mascarilla, sin mascarilla),

Personas con mascarilla



Personas sin mascarilla

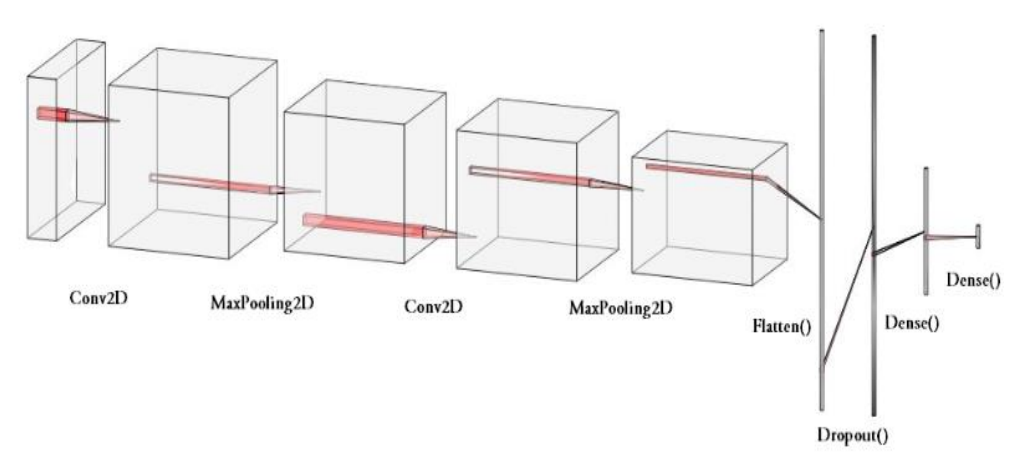
1. RECOPILACIÓN DE DATOS Y CREACIÓN DEL DATA SET

Agrupación de las imágenes en tres clases (train, test y validation), para el entrenamiento del modelo

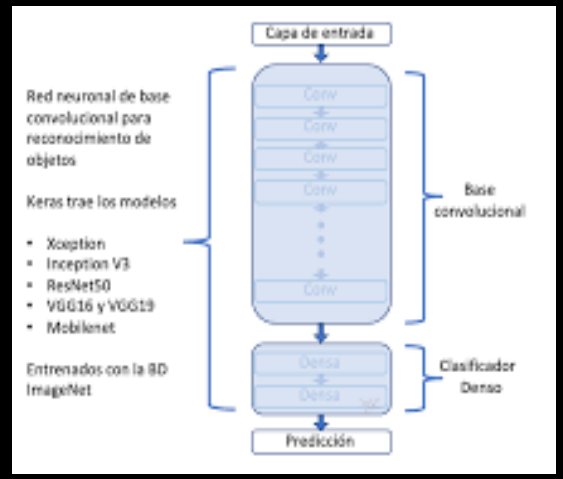
```
total training with_mask images : 4555
total training without_mask imagen images : 4536
total validation with_mask images : 808
total validation without_mask images : 875
total test with_mask images : 1077
total test without_mask images : 1001
```

TIPOS DE REDES NEURONALES CONVOLUCIONALES

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico



Una red neuronal convolucional VGG-16 se caracteriza por su simplicidad con 16 capas convolucionales usando solamente 3x3 capas convolucionales juntas una detrás de otra incrementando su profundidad. Reducción el tamaño con el max pooling. Dos capas conectadas con 4096 nodos seguidas por un clasificador softmax



2. CREACIÓN DEL MODELO

```python
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(240, 240, 3)),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',),

    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',),

    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',),

    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.ZeroPadding2D((1,1)),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.AveragePooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu',bias_initializer='zeros'),
    tf.keras.layers.Dense(1000, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Construcción del modelo CNN secuencial con varias capas como Conv2D, MaxPooling2D, Flatten, Dropout y Dense, similar al modelo VGG-16. En la última capa se utiliza la funcion de activación "sigmoid", para limitar la salida a un rango entre 0 y 1. Para obtener mejores resultados.

2. CREACIÓN DEL MODELO

```
model.compile(optimizer=keras.optimizers.SGD(lr=0.02),
              loss='binary_crossentropy',
              metrics = ['acc'])
```

```
from keras.callbacks import EarlyStopping
es = EarlyStopping(
    monitor='val_acc',
    mode='max',
    patience=6
)
```

Teniendo en cuenta la clasificaicon de dos clases (con y sin mascarilla), tiene por defecto el optimizador "adam", y le definimos un loss de "binary_crossentropy" ya que se trata de dos variables.

3. PRE-ENTRANMIENTO DEL MODELO

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator( rescale = 1.0/255.,zoom_range=0.2,rotation_range=180 )
validation_datagen  = ImageDataGenerator( rescale = 1.0/255.,zoom_range=0.2,rotation_range=180 )
test_datagen  = ImageDataGenerator( rescale = 1.0/255.,zoom_range=0.2,rotation_range=180 )


train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(240, 240))

validation_generator =  validation_datagen.flow_from_directory(validation_dir,
                                                               batch_size=20,
                                                               class_mode  = 'binary',
                                                               target_size = (240, 240))


test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=20,
                                                   class_mode  = 'binary',
                                                   target_size = (240, 240))
```

```
Found 9091 images belonging to 2 classes.
Found 1683 images belonging to 2 classes.
Found 2078 images belonging to 2 classes.
```

Crearemos el train_generator, test_generator y validation_generation para adaptarlo al entrenamiento del modelo.

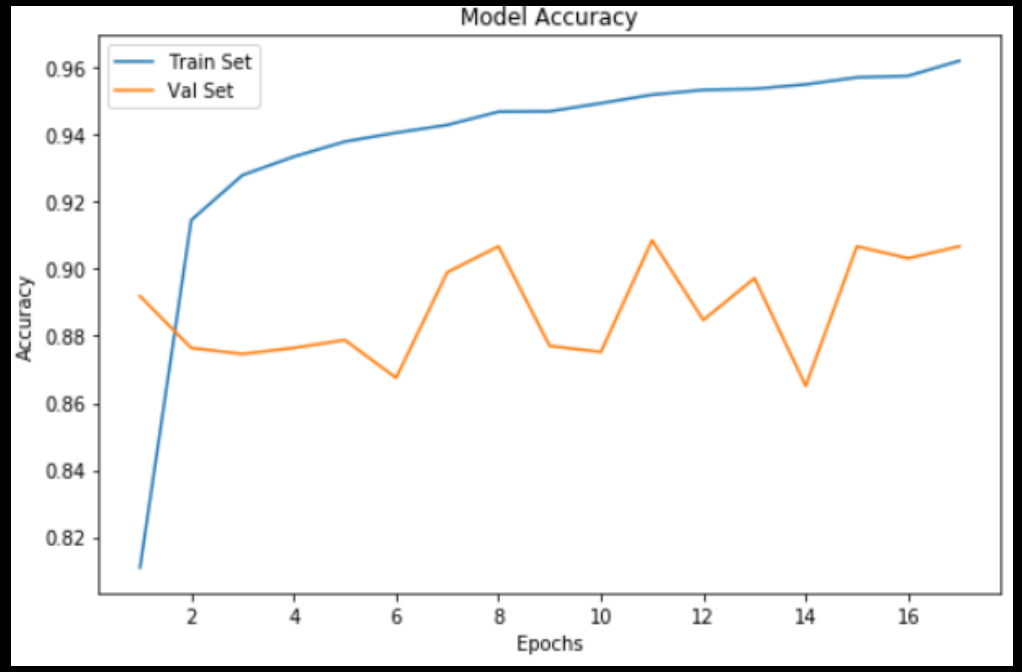
4. ENTRENAMIENTO DEL MODELO

```
import time
print(time.strftime("%H:%M:%S"))
history = model.fit(train_generator,epochs=10, validation_data=validation_generator, callbacks=[es])
print(time.strftime("%H:%M:%S"))
```

```
12:54:44
Epoch 1/10
455/455 [==============================] - 3294s 7s/step - loss: 0.6610 - acc: 0.6351 - val_loss: 0.4301 - val_acc: 0.8336
Epoch 2/10
455/455 [==============================] - 3277s 7s/step - loss: 0.4204 - acc: 0.8383 - val_loss: 0.3452 - val_acc: 0.8752
Epoch 3/10
455/455 [==============================] - 3482s 8s/step - loss: 0.3012 - acc: 0.8906 - val_loss: 0.3390 - val_acc: 0.8752
Epoch 4/10
455/455 [==============================] - 3568s 8s/step - loss: 0.2785 - acc: 0.8954 - val_loss: 0.3818 - val_acc: 0.8764
Epoch 5/10
455/455 [==============================] - 3273s 7s/step - loss: 0.2610 - acc: 0.9033 - val_loss: 0.4237 - val_acc: 0.8330
Epoch 6/10
455/455 [==============================] - 3271s 7s/step - loss: 0.2508 - acc: 0.9033 - val_loss: 0.3545 - val_acc: 0.8336
Epoch 7/10
455/455 [==============================] - 3267s 7s/step - loss: 0.2320 - acc: 0.9119 - val_loss: 0.3535 - val_acc: 0.8396
Epoch 8/10
455/455 [==============================] - 3762s 8s/step - loss: 0.2314 - acc: 0.9133 - val_loss: 0.3947 - val_acc: 0.8206
Epoch 9/10
455/455 [==============================] - 3274s 7s/step - loss: 0.2375 - acc: 0.9138 - val_loss: 0.4468 - val_acc: 0.8134
Epoch 10/10
455/455 [==============================] - 3270s 7s/step - loss: 0.2187 - acc: 0.9131 - val_loss: 0.2808 - val_acc: 0.8723
22:17:02
```

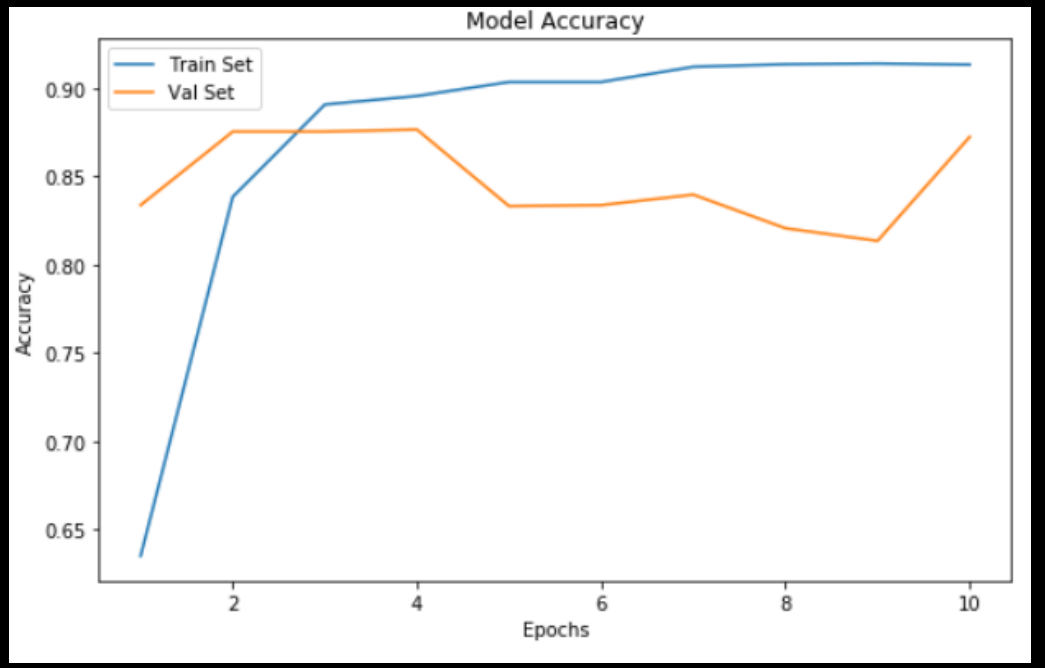
Entrenamiento del modelo con 10 épocas, era más que suficiente para conseguir buens resultados y evitar el sobreentramiento. Dada la complejidad de la red, tardó bastante en entrenar el modelo.

5. PREDICCIÓN DEL MODELO



Red neuronal con 3 capas.

Accuracy: 0.81



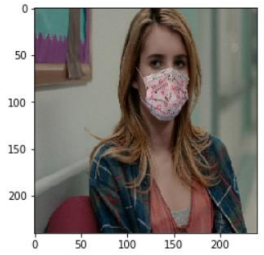
Red neuronal similar al VGG-16.

Accuracy: 0.84

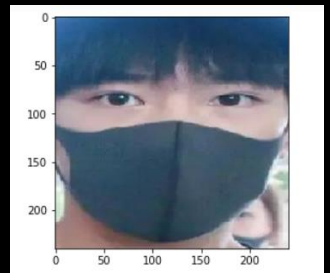
6. TESTEAR EL MODELO

```
test_generator =  test_datagen.flow_from_directory("mascarillas/test/prueba1",
                                                   batch_size=20,
                                                   class_mode  = 'binary',
                                                   target_size = (240, 240))

for i in range(20):

    x=test_generator.next()[0][0]

    image=np.expand_dims(x, axis=0)

    classes = model.predict(image)[0][0]

    plt.imshow(image[0])
    plt.show()
    if classes>0.5: print(" DONT HAVE A MASK:", classes)
    else: print(" HAVE A MASK:", classes)
```

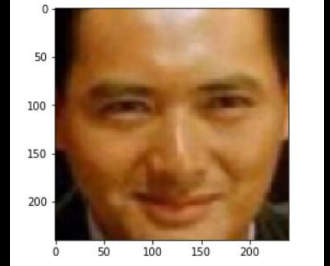
Found 10 images belonging to 2 classes.



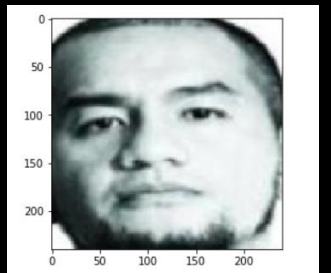
HAVE A MASK: 0.30612803



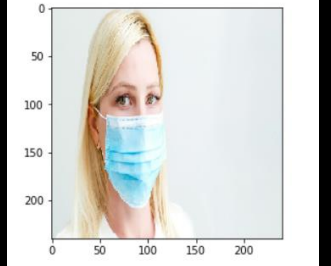
HAVE A MASK: 0.0148299



DONT HAVE A MASK: 0.6633662



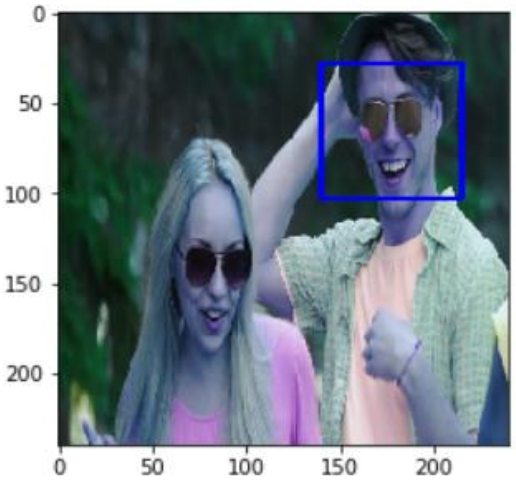
HAVE A MASK: 0.08134135



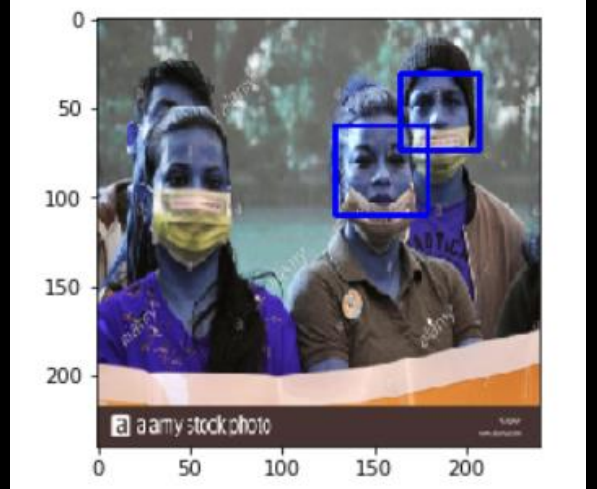
HAVE A MASK: 0.011582552

El modelo da buenos resultado, prediciendo la mayoría de las fotos.

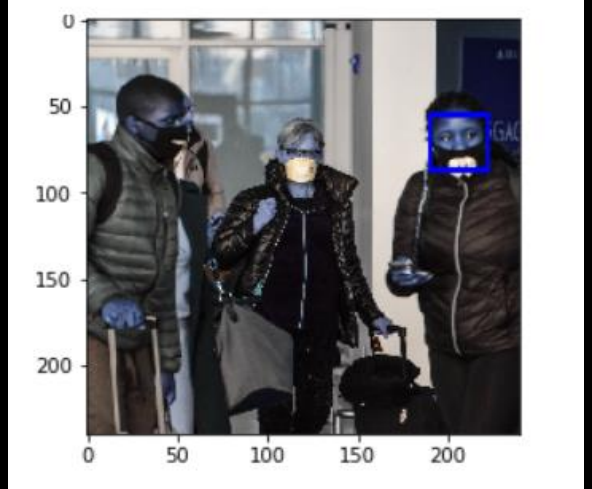
7. CARGAR DEL OPENCV Y USO DEL RECONOCMIENTO FACIAL



```
En esta imagen hay 1 personas
Tienen mascarila: 1 personas
No tienen mascarilla: 0 personas
```




```
En esta imagen hay 2 personas
Tienen mascarila: 2 personas
No tienen mascarilla: 0 personas
```



```
En esta imagen hay 1 personas
Tienen mascarila: 1 personas
No tienen mascarilla: 0 personas
```

PROBLEMA

- Algunas imágenes eran muy difíciles de clasificar, porque no se diferenciaba bien las mascarillas, haciendo que el modelo diese resultados poco fiables. Se solucionó haciendo un filtro manual para conseguir eliminar el ruido y obtener mejores resultados.
- El modelo entrenado al principio daba resultados pocos fiables, haciendo que busquemos un modelo mejor y recreando el modelo VGG-16 para obtener mejores resultados
- A la hora de procesar las imágenes una vez entrenado con el modelo VGG-16, el testeo daba que todas las imágenes tenian mascarilla. Ese problema se solucionó generando las imagenes con "I**magendatagenerator**", al igual que cuando creamos el train_generator, test_generator y validation_generation. Al testearlo de esa manera conseguimos que los resultados sean muy fiables y casi no había error en el testeo
- A la hora de pasarle el reconocimiento facial, no reconocía todas las caras de las personas en la imagen debido a las transformaciones que habia tenido la imagen anteriormente con lo que no se podiía predecir bien cuantas personas habían en la imagen y menos si tenian o no mascarilla. Los resultados que se obtenían eran contradictorios a lo que tenía que dar.
- La solución a este último problema sería encontrar un método donde el proceso de imágenes no afectase al reocnocmiento facial y al testeo del modelo.

CONCLUSIONES

- Cuanto mejor sean las imágenes menos ruido generará en el modelo
- Para conseguir alta fiabilidad se necesita entrenar un modelo complejo, así podrá distinguir cuando una persona usa mascarilla o no.
- Habría que encontrar un método para procesar bien las imágenes, para que cuando pasemos el reconocimiento facial pueda reconocer la cara de la persona perfectamente y pueda hacer bien la predicción.

¡¡GRACIAS!!

Henryk Zorrilla Mori

https://www.linkedin.com/in/henrykzm/