2020

Detección de tapabocas basado en redes neuronales profundas

STEVEN MEDINA GONZALEZ – ALDAHIR ROJAS LANCHEROS

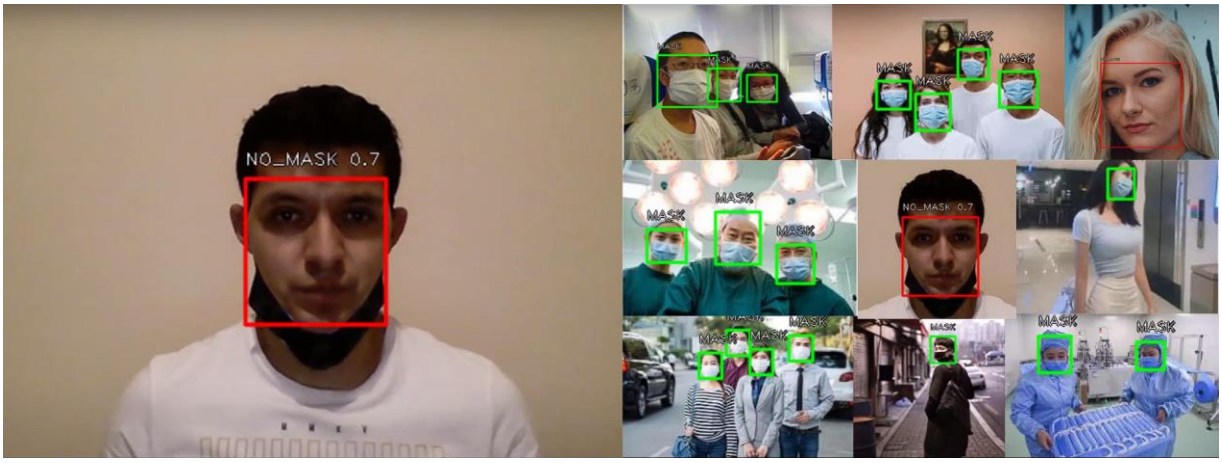
DICIEMBRE DE 2020

1 CONTENIDO

Contenido

2 PRESENTACIÓN

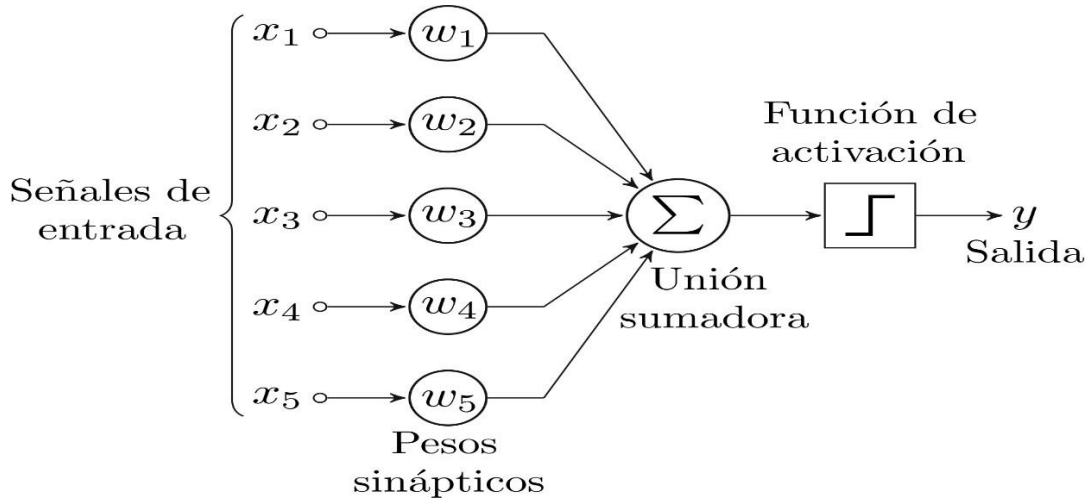
La presente monografía está orientada a la descripción la implementación de un sistema de redes neuronales profundas que permita la detección de imágenes en las cuales pueda identificar el uso de tapabocas en las personas de la foto.



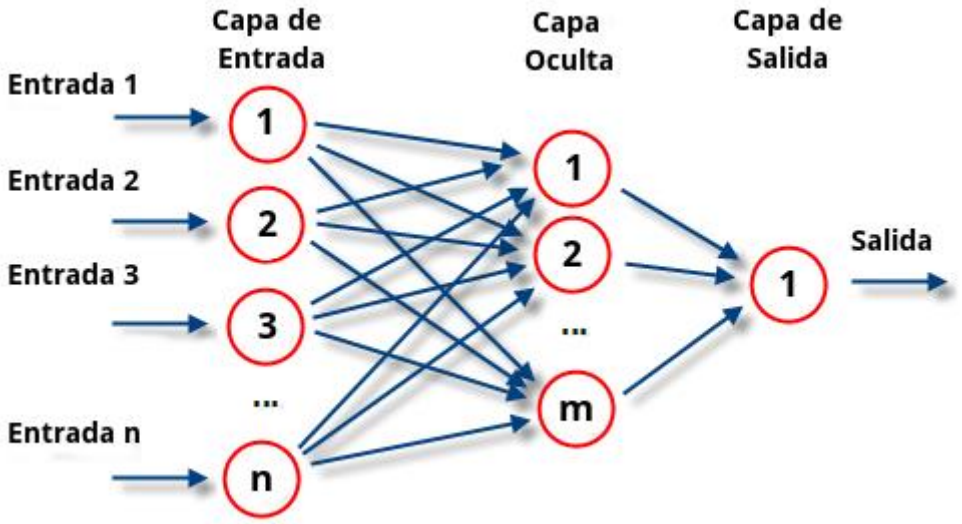
Para este proyecto se harán uso de varias librerías de Python como lo son keras, tensorflow, numpy, etc. Además de también hacer uso de la plataforma de Google Colab, para hacer el proceso de entrenamiento de una forma más rápida y precisa, añadiendo la opción de poder utilizar una tarjeta grafica Nvidia en el proceso, si el equipo tiene alguna implementada, haciendo este proceso aún más rápido, además de practico.

3 REDES NEURONALES

Las estructuras básicas de redes neuronales artificiales se basan en el Perceptrón, diseñado por Rosenblatt en 1959. Esta neurona artificial o perceptrón puede recibir varias entradas y, mediante su suma ponderada y una función de activación posterior, nos proporciona una salida que queremos que se aproxime a la salida deseada.
La precisión del perceptrón es limitada ya que solo permite clasificar problemas linealmente separables, como indicaron Minsky y Papert.



Posteriormente, como una mejora del perceptrón, surgió el perceptrón multicapa, diseñador en 1986 por Rumelhart y otros autores. Esta estructura representa una mejora ya que permite la clasificación de problemas que no son linealmente separables a diferencia del perceptrón simple. Se basa en añadir capas intermedias entre la capa de entrada y la capa de salida a la estructura, denominadas capas ocultas con diferentes tipos de funciones de activación.
De esta forma, se diseña una red de varias capas con nodos interconectados capaces de procesar la información que se les proporciona y devolviendo un resultado a la capa siguiente, hasta llegar a la última capa que devuelve un resultado basado en todo este proceso. El resultado que se obtiene en cada capa depende de las denominadas funciones de activación.
Normalmente las redes de neuronas cuentan con más de una neurona por capa y con varias capas como el caso del perceptrón multicapa antes mencionado. Estas neuronas pueden tener relación entre ellas o solo con las neuronas de capas próximas y anteriores, dependiendo del tipo de capa en el que nos encontremos. Cada una de estas relaciones, es decir, los pesos entre las neuronas y los bias son los parámetros que se entrenan en la red.

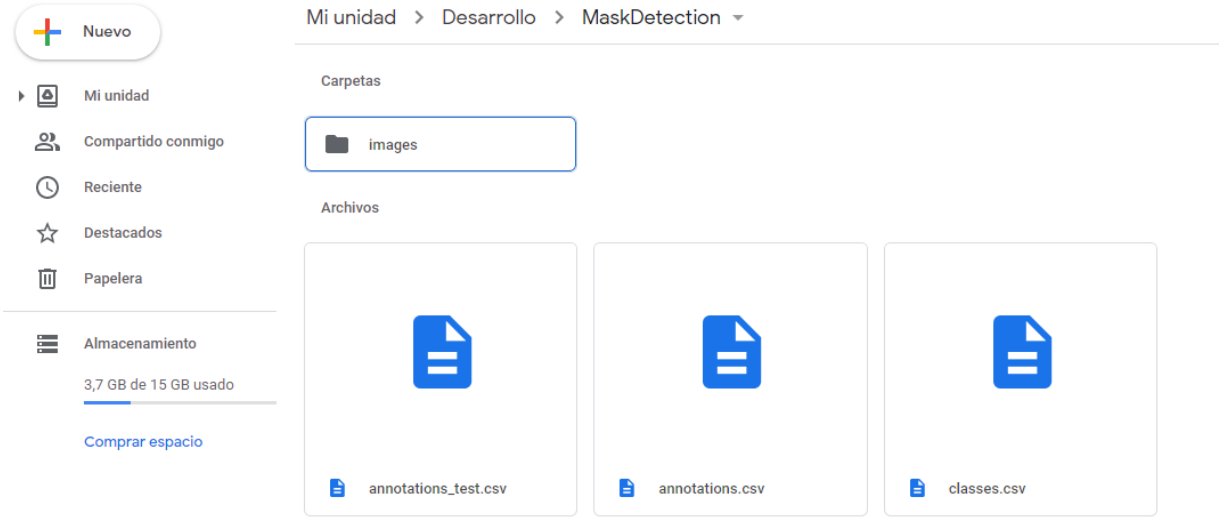


4 IMPLEMENTACIÓN

Para empezar la parte del entrenamiento, primero debemos subir unos archivos al drive, los cuales son la carpeta de imágenes el cual es un banco de imágenes ósea el dataset para realizar el entrenamiento y 3 archivos csv, todo esto se podrá encontrar en el siguiente repositorio.

https://github.com/DavidReveloLuna/MaskDetection

De manera que quede guardado en una carpeta de esta forma:



En nuestro caso tenemos 3 archivos csv, el primero es anotations y anotations_test donde están definidas todas las imágenes, cada una con sus respectivas coordenadas donde se localizan los objetos de interés, que esto por lo general se usa un software para hacer el etiquetado, pero ya el repositorio traía todo listo, y también están las clases que buscamos clasificar, que en este caso son 2, llevar mascara o no llevar mascara.

Adicionalmente en el git se encuentra un notebook con todo el código para la ejecución de la detección de objetos y un documento txt con todos los comandos para realizar el entrenamiento de la red neuronal, que veremos ahora.

Ya luego de tener la carpeta lista, procedemos a abrir Google Colab y a crear un cuaderno, en este punto empezamos a usar cada uno de los siguientes comandos:

```
!git clone  https://github.com/DavidReveloLuna/keras-retinanet
```

Lo usamos para hacer una copia del repositorio que tiene la estructura de la red neuronal.

```
cd keras-retinanet/
```

Procedemos a movernos a la carpeta que se creo con la copia del repositorio e instalamos las siguientes versiones de librerías:

```
!pip install keras==2.3.1
!pip install tensorflow==2.1
!pip install gast
!pip install tensorflow==2.1
```

Deben ser estas versiones ya que las últimas versiones dan problemas de compatibilidad.

Estas librerías son las necesarias para hacer el entrenamiento de la red neuronal, luego procedemos con el siguiente comando.

```
!pip install .
!python setup.py build_ext --inplace
```

Con este instalamos y configuramos los paquetes para el manejo de construcción e importación en el entorno.

```
from google.colab import drive
drive.mount('/content/drive')
```

Con este comando procedemos a crear una conexión entre nuestro drive y el notebook, para que al usar este siguiente comando procedamos a hacer una copia de esos archivos del dataset hacia la carpeta donde estamos operando la red neuronal.

```
!cp -r "/content/drive/My Drive/Desarrollo/MaskDetection/images" "/content/keras-retinanet"
!cp -r "/content/drive/My Drive/Desarrollo/MaskDetection/annotations.csv" "/content/keras-retinanet"
!cp -r "/content/drive/My Drive/Desarrollo/MaskDetection/annotations test.csv" "/content/keras-retinanet"
!cp -r "/content/drive/My Drive/Desarrollo/MaskDetection/classes.csv" "/content/keras-retinanet"
```

Es importante que coloquen la dirección exacta de su carpeta, pueden seguir el ejemplo nuestro donde teníamos la carpeta images y archivos csv contenido en la carpeta MaskDetection y luego dentro de otra llamada Desarrollo.

Seguidamente hacemos uso de este código para importar y organizar todo lo necesario para empezar el entrenamiento de forma correcta.

Ahora al usar el siguiente comando descargaremos el modelo pre entrenado que es la base para generar el nuevo modelo reentrenado con diferentes imágenes del dataset.

```
URL_MODEL = 'https://github.com/fizyr/keras-retinanet/releases/download/0.5.1/resnet50_coco_best_v2.1.0.h5'
urllib.request.urlretrieve(URL_MODEL, './snapshots/model.h5')
```

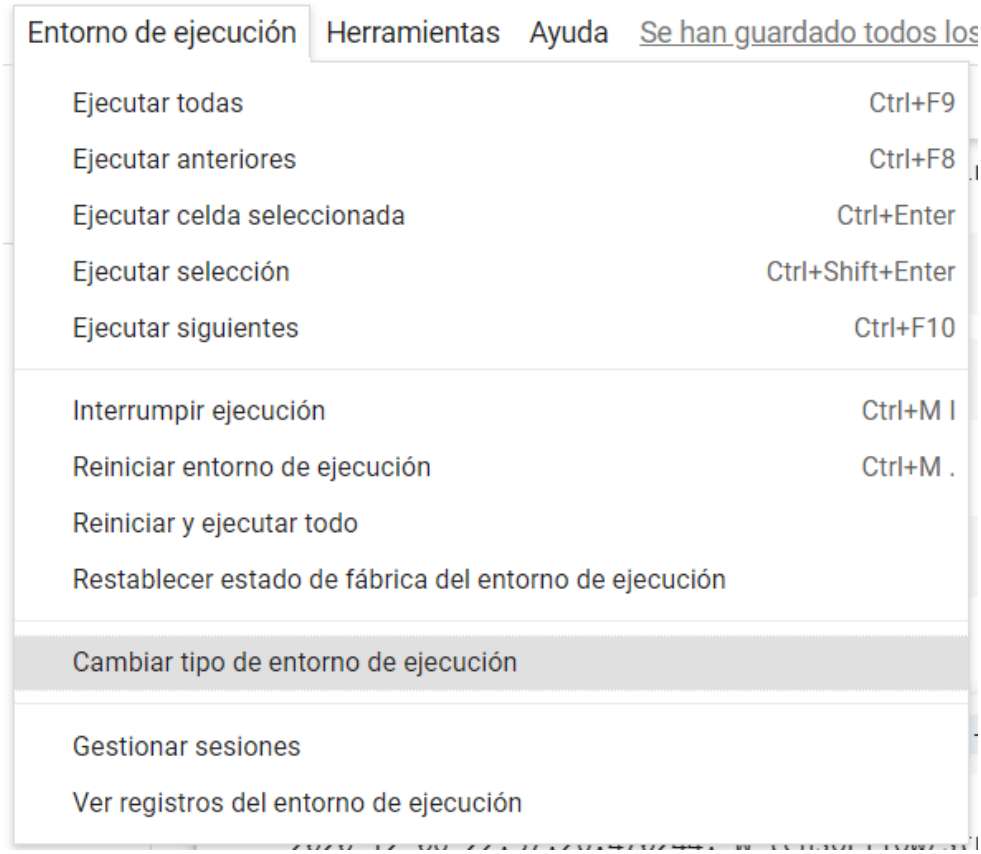
Después debemos dirigirnos a la carpeta bin para darle permisos de administrador al archivo train.py

```
cd /content/keras-retinanet/keras_retinanet/bin
!sudo chmod 777 train.py
```

Luego volvemos a la carpeta haciendo uso 2 veces del siguiente comando

```
cd ..
```

Finalmente comenzamos con el entrenamiento de la red neuronal con el modelo previamente descargado, con ejecutar el siguiente código empezara el proceso, pero antes de eso es importante escoger la opción en la barra de herramientas para activar e uso de la GPU del computador para hacer este proceso más rápido, de lo contrario se puede demorar unas horas de mas este proceso.



Se escoge la opción “Cambiar tipo de entorno de ejecución” y se escoge la opción GPU, esto solo se puede hacer si se tiene una HPU Nvidia, de lo contrario tendrá que esperar el proceso completo.

```
!keras_retinanet/bin/train.py --freeze-backbone --random-transform --
       weights snapshots/model.h5 --batch-size 8 --steps 88 --
              epochs 90 csv annotations.csv classes.csv
```

Con este comando elegimos el modelo que vamos a usar, ósea el model.h5 que descargamos y finalmente tenemos el batch size que es la cantidad de ejemplos de entrenamiento utilizados en una iteración, los steps donde se procesan una cantidad de ejemplos dependiendo del batch size y al final el epoch que es el ciclo completo de cada steps, así que se puede resumir en que es un tipo de for anidado.

```
from distutils.version import LooseVersion
import warnings
import tensorflow as tf

# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use TensorFlow version 1.0 or
newer.  You are using {}'.format(tf.__version__)
print('TensorFlow Version: {}'.format(tf.__version__))

# Check for a GPU
if not tf.test.gpu_device_name():
   warnings.warn('No GPU found. Please ensure you have installed TensorFlow correctly')
else:
   print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

Esta primera parte es básicamente para verificar si se tiene instalado el tensorflow y la versión del mismo, además de indicar si se reconoce o no el sistema de GPU.

```
import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
import urllib
import os
import csv
```

```
import cv2
import time
from PIL import Image

from keras.models import load_model
from keras_retinanet import models
from keras_retinanet.utils.image import read_image_bgr, preprocess_image
, resize_image
from keras_retinanet.utils.visualization import draw_box, draw_caption
from keras_retinanet.utils.colors import label_color
```

En la segunda parte importamos las librerías que se utilizaran a lo largo del código como lo son numpy, pandas, seaborn, pylab, matplotlib, csv os, cv2.

```
from keras.models import load_model
from keras_retinanet import models

model_path = os.path.join('snapshots', sorted(os.listdir('snapshots'),
reverse=True)[0])
print(model_path)

model = models.load_model(model_path, backbone_name='resnet50')
model = models.convert_model(model)

labels_to_names = pd.read_csv('classes.csv',
header=None).T.loc[0].to_dict()
```

En la tercera celda lo que hacemos es cargar el modelo ya entrenado, que en nuestro caso sería el de la época 99 y el archivo que contiene la información de las etiquetas de las clases que creamos.

```
test_df = pd.read_csv("annotations_test.csv")
test_df.head()
print(labels_to_names)
```

Se carga annotations_test.csv para hacer las pruebas en imágenes, con las clases que creamos que serían mask y no_mask.

```
import skimage.io as io

def predict(image):
  image = preprocess_image(image.copy())
  image, scale = resize_image(image)

  boxes, scores, labels = model.predict_on_batch(
   np.expand_dims(image, axis=0)
```

```
    )

    boxes /= scale

    return boxes, scores, labels

umbralScore = 0.4

def draw_detections(image, boxes, scores, labels):
    for box, score, label in zip(boxes[0], scores[0], labels[0]):
        if score < umbralScore:
            break

        #color = label_color(label)

        b = box.astype(int)
        if label == 0:
            color = [0,255,0]
        elif label == 1:
            color = [255,0,0]
        draw_box(draw, b, color=color)
        caption = "{} ".format(labels_to_names[label])
        draw_caption(draw, b, caption.upper())
        #draw_box(image, b, color=color)

        #caption = "{} {:.3f}".format(labels_to_names[label], score)
        #draw_caption(image, b, caption)
```

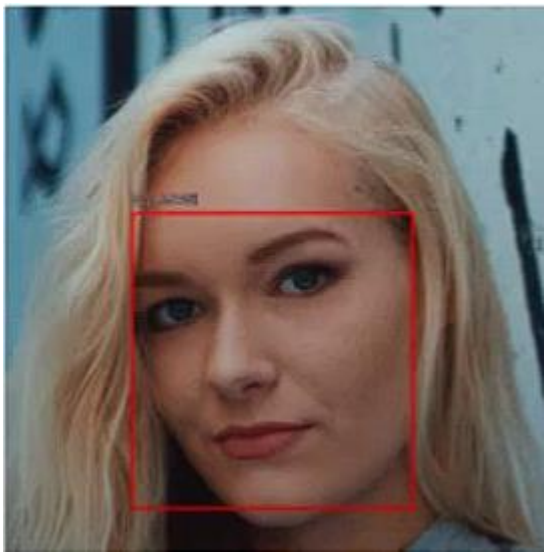
La función predicción recibe una imagen, la procesa, y hace un re escalado de la imagen. Y llamamos la función que hace la predicción con base en el modelo entrenado y nos devuelve las regiones de interés, los puntajes y las etiquetas.

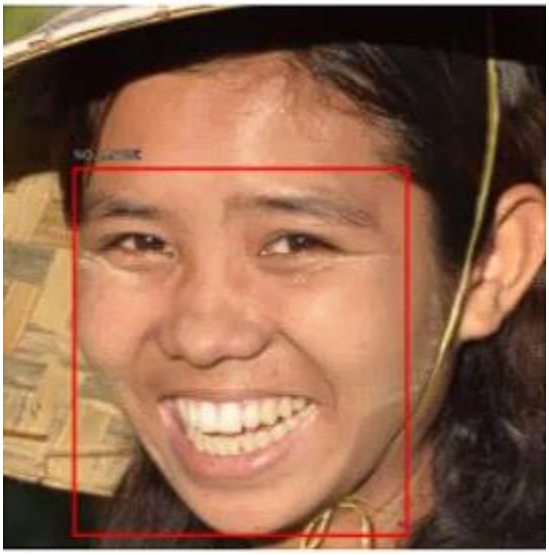
La siguiente función dibuja las clases, en caso de que la clase se la de no_mask se pintara un recuadro rojo, y en caso de que si se lleve mascara la clase se representara en un cuadro verde.

```
for index, row in test_df.iterrows():
    #print(row[0], index)
    image = io.imread(row[0])
    #image = io.imread('images/new_16.jpg')

    boxes, scores, labels = predict(image)

    draw = image.copy()
    draw_detections(draw, boxes, scores, labels)
```

```
plt.axis('off')
plt.imshow(draw)
plt.show()
```

Probamos las imágenes, donde vemos que dependiendo de si se lleva la máscara o no.

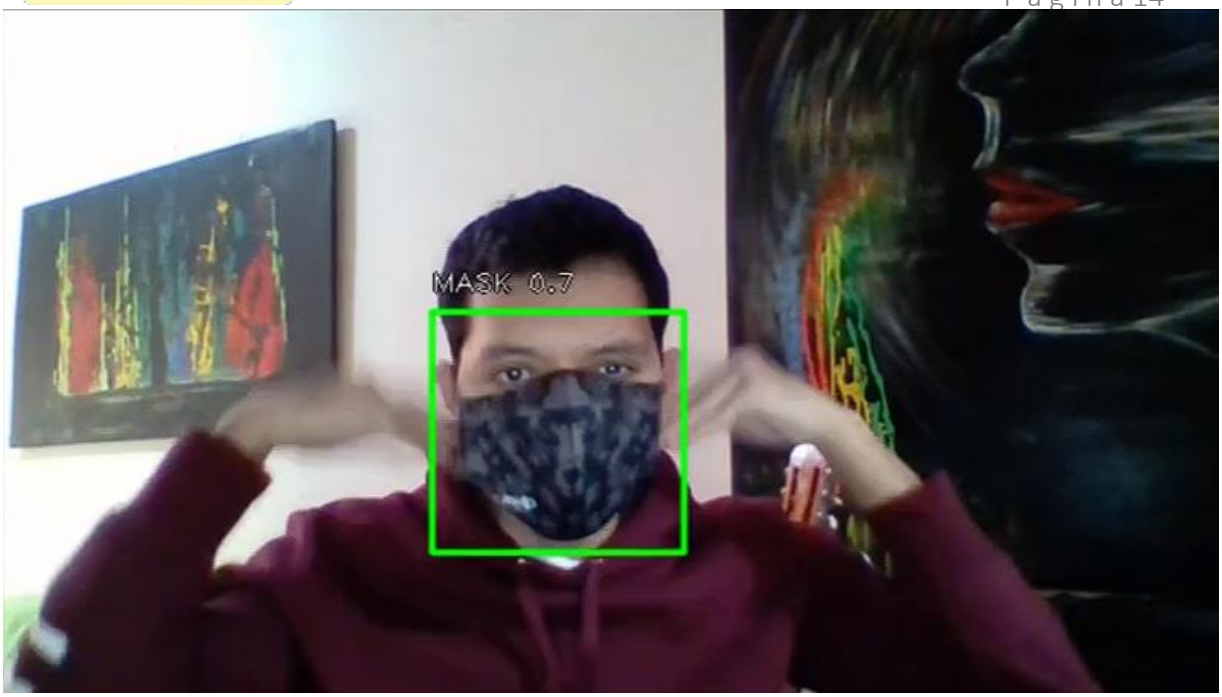




La siguiente sección del código es en caso de contar con una cámara web, se toma la captura de la cámara, se hace una lectura de los frames, se utiliza la función de predicción para cada frame, y luego se dibuja en la imagen los objetos que se dibujaron en cada clase y ya por último se muestra la imagen.

```
# get the reference to the webcam
camera = cv2.VideoCapture(0)
camera_height = 500

while(True):
   # read a new frame
   _, frame = camera.read()
```

```
    boxes, scores, labels = predict(frame)


    draw = frame.copy()


    for box, score, label in zip(boxes[0], scores[0], labels[0]):
        if score > 0.5:
            #print(box)
            b = box.astype(int)
            if label == 0:
                color = [0,255,0]
            elif label == 1:
                color = [0,0,255]

            draw_box(draw, b, color=color)
            caption = "{} {:.1f}".format(labels_to_names[label], score)
            draw_caption(draw, b, caption.upper())



    # show the frame
    cv2.imshow("Test out", draw)


    key = cv2.waitKey(1)

    # quit camera if 'q' key is pressed
    if key & 0xFF == ord("q"):
        break

camera.release()
cv2.destroyAllWindows()

camera.release()
```


MASK 0.7

5 CONCLUSIONES

Como conclusión el proyecto estuvo interesante, ya que viendo el contexto en que estamos actualmente, puede ser útil su implementación, por ejemplo, en centros comerciales para detectar si las personas llevan o no su tapabocas como medida sanitaria.

El propósito principal de una red neuronal es recibir un conjunto de entradas, realizar cálculos progresivamente complejos en ellas y dar salida para resolver problemas del mundo real como la clasificación. La parte más importante en esta implementación se encuentra en el entreno adecuado del modelo, ya que se requiere tener un balance para no sobreentrenar el modelo y evitar que genere fallos al conocer elementos que no son un tapabocas.

6 BIBLIOGRAFÍA

[1] https://github.com/DavidReveloLuna/MaskDetection

[2] https://es.wikipedia.org/wiki/Red_neuronal_artificial

[3] https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo