

PROJET DÉTECTEUR DE MASQUES - Eva & Patricia



Détecteur des masques : objectifs du projet

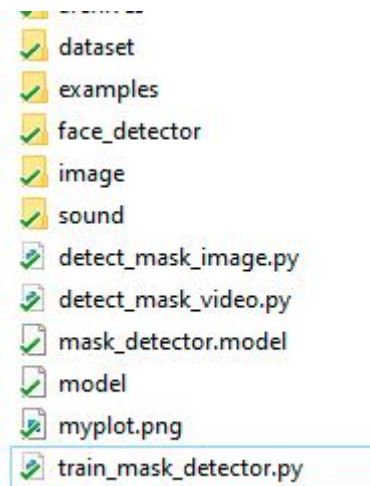
Vous allez construire un détecteur des masques en deux étapes:

- En premier temps sur les images, ton détecteur va détecter si la personne dans l'image porte un masque ou pas (Pensez à afficher un message pour dire si la personne porte un masque ou pas)
- Puis en temps réel pour détecter si la personne sur la webcam porte un masque ou non.

Vous allez entraîner votre modèle en utilisant des réseaux deep.
Optez par les bibliothèques Tensorflow, Keras, OpenCV, etc
(Pensez à utiliser des emojis pour dire est ce que c'est Ok ou pas, ensuite un message audio qui demande à la personne de porter un masque si c'est pas le cas.)

Avant de commencer

Structure du projet



- dataset
 - jeu d'entraînement avec masques (690 photos)
 - jeu d'entraînement sans masque (686 photos)
 - examples (images test pour le test des images)
 - images avec et sans masques (exemples 01.png...)
 - image (smiley pour détection vidéo)
 - smiley heureux
 - smiley triste
 - sound
 - bande d'enregistrement pour signifier de mettre un masque lors de la détection vidéo
- detect_mask_image.py fichier pour la reconnaissance d'image
- detect_mask_video..py fichier pour la reconnaissance vidéo
- my plot png (courbe training loss et accuracy)



Lancement du programme

démarrer CMD anaconda : cd “ emplacement du fichier”

```
(cd C:\Users\utilisateur\Google Drive\microsoft_ia\Google  
Drive\projets\ia\Reconnaissance de masques\FICHIERS\face-mask-detector)
```

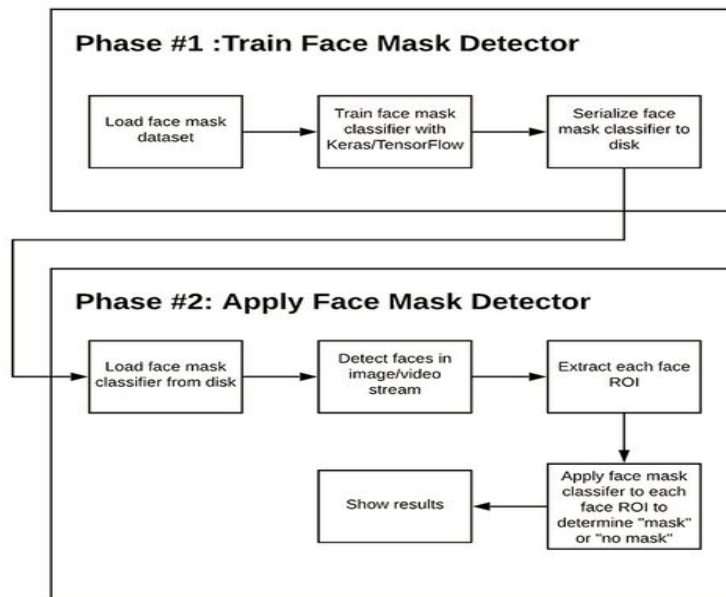
pour les images :

```
python detect_mask_image.py --image examples/example_02.png
```

pour la vidéo :

```
python detect_mask_video.py
```

Two-phase COVID-19 face mask detector





sommaire

- ensemble de données
- création du jeu de données de masque facial
- structure du projet
- implémentation de notre script de formation au détecteur de masque facial
- entraînement du détecteur de masque facial avec Keras et Tensorflow
- implémentation de notre détecteur de masque facial pour les images avec open cv
- détection du masque facial dans les images avec open cv
- implémentation de notre détecteur de masque facial dans les flux vidéo avec smiley et bande son
- détection des masques faciaux avec open cv en temps réel



DONNÉES

Cet ensemble de données se compose de **1376 images** appartenant à deux classes:

- with_mask : 690 images
- without_mask : 686 images

Pour créer cet ensemble de données : il est important de créer des repères faciaux. pour déduire l'emplacement des structures faciales(yeux, nez, bouche ..)

- d'abord image ne portant pas de masque facial
- ensuite , on applique la détection du visage pour calculer l'emplacement du visage
- une fois l'image trouvée dans l'image, on peut extraire la région d'intérêt (ROI):

Ensuite , on fait de même avec une image de masque.

Le masque est ensuite redimensionné et tourné en le plaçant sur le visage.

En répétant ce processus pour toutes nos images d'entrée , on crée notre jeu de données de masque facial artificiel

nb : ne pas réutiliser les images sans masques utilisées pour l'ensemble d'entraînement. Sinon le model sera fortement biaisé.



Nous allons passer en revue trois scripts Python dans ce tutoriel:

- `train_mask_detector.py` : Accepte notre jeu de données d'entrée et ajuste **MobileNetV2** dessus pour créer notre `mask_detector.model` . Une histoire de formation `plot.png` contenant des courbes de précision / perte est également produit
- `detect_mask_image.py` : Effectue la détection du masque facial dans les images statiques
- `detect_mask_video.py` : À l'aide de votre webcam, ce script applique la détection du masque facial à chaque image du flux

I - train_mask_detector.py

chargement des bibliothèques

Open up the `train_mask_detector.py` file in your directory structure, and insert the following code:

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. # import the necessary packages
2. from tensorflow.keras.preprocessing.image import ImageDataGenerator
3. from tensorflow.keras.applications import MobileNetV2
4. from tensorflow.keras.layers import AveragePooling2D
5. from tensorflow.keras.layers import Dropout
6. from tensorflow.keras.layers import Flatten
7. from tensorflow.keras.layers import Dense
8. from tensorflow.keras.layers import Input
9. from tensorflow.keras.models import Model
10. from tensorflow.keras.optimizers import Adam
11. from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
12. from tensorflow.keras.preprocessing.image import img_to_array
13. from tensorflow.keras.preprocessing.image import load_img
14. from tensorflow.keras.utils import to_categorical
15. from sklearn.preprocessing import LabelBinarizer
16. from sklearn.model_selection import train_test_split
17. from sklearn.metrics import classification_report
18. from imutils import paths
19. import matplotlib.pyplot as plt
20. import numpy as np
21. import argparse
22. import os
```

Arguments de ligne de commande

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
24. # construct the argument parser and parse the arguments
25. ap = argparse.ArgumentParser()
26. ap.add_argument("-d", "--dataset", required=True,
27.                 help="path to input dataset")
28. ap.add_argument("-p", "--plot", type=str, default="plot.png",
29.                 help="path to output loss/accuracy plot")
30. ap.add_argument("-m", "--model", type=str,
31.                 default="mask_detector.model",
32.                 help="path to output face mask detector model")
33. args = vars(ap.parse_args())
```

- --dataset : chemin vers le jeu de données
- --plot : chemin d'accès à l'historique d'entraînement en sortie
- --model : chemin d'accès au modèle de classification de masque sérialisé

Définition des hyperparamètres d'apprentissage

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
35. | # initialize the initial learning rate, number of epochs to train for,  
36. | # and batch size  
37. | INIT_LR = 1e-4  
38. | EPOCHS = 20  
39. | BS = 32
```

On initialise le taux d'apprentissage

but => valeur optimale dans notre modèle, descente du gradient pas à pas, si modèle trop petit ca peut prendre du temps, trouver le bon compromis.

1 epoch contient plusieurs itérations, pour savoir combien d'itération dans l'epoch, on divise le nombre d'image dans la bdd ds le batch.

exemple 256 images; batch = 32 soit 8 itérations; 1 epoch = 8 itérations, 20 epochs = 160 itérations

exemple 2 bs = 2 (chien, chats) et 4 (chien, chat, girafe, éléphant), plus le modèle est grand plus le modèle est précis

BS => Batch ici nous mettons 32 images dans notre modèle

pour 1 epoch on parcourt 128/32 images soit 4 itérations => 20 epochs 80 itérations

CHARGEMENT ET PRÉTRAITEMENT DE NOS DONNÉES

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
41. # grab the list of images in our dataset directory, then initialize
42. # the list of data (i.e., images) and class images
43. print("[INFO] loading images...")
44. imagePath = list(paths.list_images(args["dataset"]))
45. data = []
46. labels = []
47.
48. # loop over the image paths
49. for imagePath in imagePath:
50.     # extract the class label from the filename
51.     label = imagePath.split(os.path.sep)[-2]
52.
53.     # load the input image (224x224) and preprocess it
54.     image = load_img(imagePath, target_size=(224, 224))
55.     image = img_to_array(image)
56.     image = preprocess_input(image)
57.
58.     # update the data and labels lists, respectively
59.     data.append(image)
60.     labels.append(label)
61.
62. # convert the data and labels to NumPy arrays
63. data = np.array(data, dtype="float32")
64. labels = np.array(labels)
```

- Saisie des 'imagePaths' dans le jeu de données
- Initialisation des "data" et des "labels"
- On boucle sur "imagePaths"
- puis on charge et prétraite les images (redimensionnement à 224X224 pixels, conversion au format de matrice et la mise à l'échelle des intensités de pixels dna l'image d'entrée)

Encodage des labels, fractionnement des données en jeu de test (20%) et d'entraînement, générateur d'images d'entraînement pour l'augmentation des données

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
66. # perform one-hot encoding on the labels
67. lb = LabelBinarizer()
68. labels = lb.fit_transform(labels)
69. labels = to_categorical(labels)
70.
71. # partition the data into training and testing splits using 80% of
72. # the data for training and the remaining 20% for testing
73. (trainX, testX, trainY, testY) = train_test_split(data, labels,
74.     test_size=0.20, stratify=labels, random_state=42)
75.
76. # construct the training image generator for data augmentation
77. aug = ImageDataGenerator(
78.     rotation_range=20,
79.     zoom_range=0.15,
80.     width_shift_range=0.2,
81.     height_shift_range=0.2,
82.     shear_range=0.15,
83.     horizontal_flip=True,
84.     fill_mode="nearest")
```

Mise en place du modèle de référence

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
86. # load the MobileNetV2 network, ensuring the head FC layer sets are
87. # left off
88. baseModel = MobileNetV2(weights="imagenet", include_top=False,
89.     input_tensor=Input(shape=(224, 224, 3)))
90.
91. # construct the head of the model that will be placed on top of the
92. # the base model
93. headModel = baseModel.output
94. headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
95. headModel = Flatten(name="flatten")(headModel)
96. headModel = Dense(128, activation="relu")(headModel)
97. headModel = Dropout(0.5)(headModel)
98. headModel = Dense(2, activation="softmax")(headModel)
99.
100. # place the head FC model on top of the base model (this will become
101. # the actual model we will train)
102. model = Model(inputs=baseModel.input, outputs=headModel)
103.
104. # loop over all layers in the base model and freeze them so they will
105. # *not* be updated during the first training process
106. for layer in baseModel.layers:
107.     layer.trainable = False
```


Compilation et formation de notre réseau de détecteurs de masques faciaux avec l'optimiseur Adam et modèle de classification binaire

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
109. # compile our model
110. print("[INFO] compiling model...")
111. opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
112. model.compile(loss="binary_crossentropy", optimizer=opt,
113.               metrics=["accuracy"])
114.
115. # train the head of the network
116. print("[INFO] training head...")
117. H = model.fit(
118.     aug.flow(trainX, trainY, batch_size=BS),
119.     steps_per_epoch=len(trainX) // BS,
120.     validation_data=(testX, testY),
121.     validation_steps=len(testX) // BS,
122.     epochs=EPOCHS)
```

```

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])

```

Une fois la formation terminée, nous évaluerons le modèle résultant sur l'ensemble de test:

Notre dernière étape consiste à tracer nos courbes de précision et de perte:

Entraînement du détecteur de masque facial COVID-19 avec Keras / TensorFlow

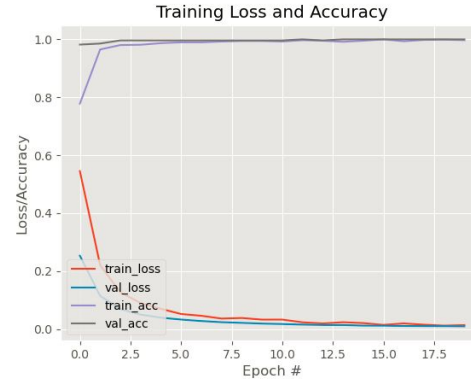
: python train_mask_detector.py -d dataset -m model -p myplot

```
[INFO] Evaluation du réseau ...  
rappel de précision prise en charge du score f1
```


with_mask	0.99	1.00	0.99 138
sans_masque	1.00	0.99	0.99 138

précision	0.99 276		
macro moyenne	0.99	0.99	0.99 276
moyenne pondérée	0.99	0.99	0.99 276

Précision de 99%.
Peu de signes de surajustement



II- detect_mask_image.py



Implémentation de notre détecteur de masque facial COVID-19 pour les images avec OpenCV

BUT :

Notre détecteur de masque facial étant formé

- **détection des visages dans l'image**
- **appliquer notre détecteur de masque facial pour classer le visage comme avec ou sans masque**



Import des bibliothèques

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
1. # import the necessary packages
2. from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
3. from tensorflow.keras.preprocessing.image import img_to_array
4. from tensorflow.keras.models import load_model
5. import numpy as np
6. import argparse
7. import cv2
8. import os
```

```
# construct the argument parser and parse the arguments
#construire l'analyseur d'arguments et analyser les arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
print(args)
```

Nos quatre arguments de ligne de commande incluent:

- --image
: Le chemin vers l'image d'entrée contenant les visages pour l'inférence
- --face
: Le chemin vers le répertoire du modèle du détecteur de visage
- --model
: Le chemin vers le modèle de détecteur de masque facial
- --confidence
: Un seuil de probabilité facultatif peut être défini pour remplacer 50% afin de filtrer les détections de visage faibles

Chargement des modèles de détecteur de visage et de classificateur de masque facial

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
24. # load our serialized face detector model from disk
25. print("[INFO] loading face detector model...")
26. prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
27. weightsPath = os.path.sep.join([args["face"],
28.     "res10_300x300_ssd_iter_140000.caffemodel"])
29. net = cv2.dnn.readNet(prototxtPath, weightsPath)
30.
31. # load the face mask detector model from disk
32. print("[INFO] loading face mask detector model...")
33. model = load_model(args["model"])
```

redimensionnement de l'image à 300X300 pixels, puis détection de visage pour localiser les masques dans l'image

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
35. # load the input image from disk, clone it, and grab the image spatial
36. # dimensions
37. image = cv2.imread(args["image"])
38. orig = image.copy()
39. (h, w) = image.shape[:2]
40.
41. # construct a blob from the image
42. blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
43.                               (104.0, 177.0, 123.0))
44.
45. # pass the blob through the network and obtain the face detections
46. print("[INFO] computing face detections...")
47. net.setInput(blob)
48. detections = net.forward()
```

détection de visage pour localiser où se trouvent tous les visages dans l'image.
puis nous assurerons qu'ils répondent aux seuil de confiance , la délimitation de la boîte d'affichage dans laquelle figure l'image du visage.

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
50. # loop over the detections
51. for i in range(0, detections.shape[2]):
52.     # extract the confidence (i.e., probability) associated with
53.     # the detection
54.     confidence = detections[0, 0, i, 2]
55.
56.     # filter out weak detections by ensuring the confidence is
57.     # greater than the minimum confidence
58.     if confidence > args["confidence"]:
59.         # compute the (x, y)-coordinates of the bounding box for
60.         # the object
61.         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
62.         (startX, startY, endX, endY) = box.astype("int")
63.
64.         # ensure the bounding boxes fall within the dimensions of
65.         # the frame
66.         (startX, startY) = (max(0, startX), max(0, startY))
67.         (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

détection de masque pour faire les prédictions avec ou sans masque.

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

```
69. |         # extract the face ROI, convert it from BGR to RGB channel
70. |         # ordering, resize it to 224x224, and preprocess it
71. |         face = image[startY:endY, startX:endX]
72. |         face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
73. |         face = cv2.resize(face, (224, 224))
74. |         face = img_to_array(face)
75. |         face = preprocess_input(face)
76. |         face = np.expand_dims(face, axis=0)
77. |
78. |         # pass the face through the model to determine if the face
79. |         # has a mask or not
80. |         (mask, withoutMask) = model.predict(face)[0]
```


on attribue une couleur qui sera
«vert» pour la présence de masque
«rouge» pour sans masque.
et le cadre de sélection autour du visage

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

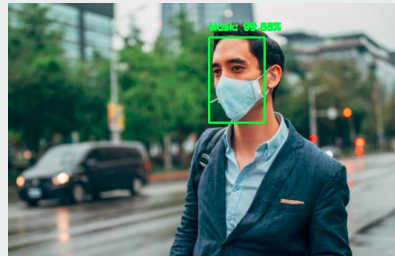
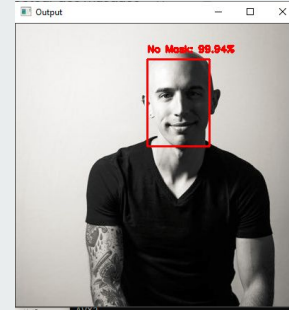
```
82. |         # determine the class label and color we'll use to draw
83. |         # the bounding box and text
84. |         label = "Mask" if mask > withoutMask else "No Mask"
85. |         color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
86. |
87. |         # include the probability in the label
88. |         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
89. |
90. |         # display the label and bounding box rectangle on the output
91. |         # frame
92. |         cv2.putText(image, label, (startX, startY - 10),
93. |                     cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
94. |         cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
95. |
96. |     # show the output image
97. |     cv2.imshow("Output", image)
98. |     cv2.waitKey(0)
```

TESTS IMAGES

lancer l'invite de commande cmd

```
python detect_mask_image.py --image examples/example_02.png
```

```
python detect_mask_image.py --image examples/example_01.png
```



III- detect_mask_video.py


but :

- **détecter si l'on porte un masque ou pas**
- **incorporer une bande son qui précise de mettre son masque si la personne ne porte pas de masque.**

Import des bibliothèques

```
# python detect_mask_video.py

# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import simpleaudio as sa
import numpy as np
import argparse
import imutils
import time
import cv2
import os
import vlc
```

création de la fonction detect_and_predict_mask avec trois paramètres

- frame : Une image de notre flux
- faceNet : Le modèle utilisé pour détecter où dans l'image les visages sont
- maskNet: Notre modèle de classificateur de masque facial COVID-19

```
def detect_and_predict_mask(frame, faceNet, maskNet):  
    # grab the dimensions of the frame and then construct a blob  
    # from it  
    (h, w) = frame.shape[:2]  
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),  
    (104.0, 177.0, 123.0))  
  
    # pass the blob through the network and obtain the face detections  
    faceNet.setInput(blob)  
    detections = faceNet.forward()  
  
    # initialize our list of faces, their corresponding locations,  
    # and the list of predictions from our face mask network  
    faces = []  
    locs = []  
    preds = []
```

À l'intérieur, nous
construisons un blob

qui détecte les visages
et initialise les listes,
dont deux sont définies
pour renvoyer la
fonction.

Ces listes incluent nos
faces (c.-à-d., ROI),
locs (les emplacements
du visage), et
preds (la liste des
prédictions de masque /
pas de masque).

Mise en place d'une boucle, pour filtrer les détections faibles

extraction des boîtes englobantes tout en veillant à ce que les dimensions de la boîte ne tombent pas en dehors des limites de l'image.

```
# loop over the detections
for i in range(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with
    # the detection
    confidence = detections[0, 0, i, 2]


    # filter out weak detections by ensuring the confidence is
    # greater than the minimum confidence
    if confidence > args["confidence"]:
        # compute the (x, y)-coordinates of the bounding box for
        # the object
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # ensure the bounding boxes fall within the dimensions of
        # the frame
        (startX, startY) = (max(0, startX), max(0, startY))
        (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

        # extract the face ROI, convert it from BGR to RGB channel
        # ordering, resize it to 224x224, and preprocess it
        face = frame[startY:endY, startX:endX]
        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
        face = cv2.resize(face, (224, 224))
        face = img_to_array(face)
        face = preprocess_input(face)


        # add the face and bounding boxes to their respective
        # lists
        faces.append(face)
        locs.append((startX, startY, endX, endY))
```

Nous nous assurons qu'au moins un visage est détecté



```
# only make a predictions if at least one face was detected
if len(faces) > 0:
    # for faster inference we'll make batch predictions on *all*
    # faces at the same time rather than one-by-one predictions
    # in the above `for` loop
    faces = np.array(faces, dtype="float32")
    preds = maskNet.predict(faces, batch_size=32)

# return a 2-tuple of the face locations and their corresponding
# locations
return (locs, preds)
```



```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
                default="face_detector",
                help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
                default="mask_detector.model",
                help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
                help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Nos arguments de ligne de commande incluent:

- --face : Le chemin vers le répertoire du détecteur de visage
- --model : Le chemin vers notre classificateur de masque facial qualifié
- --confidence : Le seuil de probabilité minimum pour filtrer les détections de visage faibles


```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

on initialise notre:

Détecteur de visage

Détecteur de masque facial
COVID-19

Flux vidéo de la webcam

On passe en boucle sur les
images du flux:

```

# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

```

Dans notre boucle sur les résultats de prédiction, on va créer un cadre de délimitation de visage et masquer / ne pas masquer la prédiction. On détermine le cadre et la couleur. Enfin, on affiche les résultats et on effectue le nettoyage:

Insertion smiley et bande son

```
#insertion smiley
smiley = "\heureux.jpg" if mask > withoutMask else '\smtriste.jpg'
path = 'image' + smiley
# Reading an image in default mode
image = cv2.imread(path)
# Window name in which image is displayed
window_name = 'Image'
# font
font = cv2.FONT_HERSHEY_SIMPLEX
# org
org = (50, 50)
# fontScale
fontScale = 1
# Blue color in BGR
color_smiley = (255, 0, 0)
# Line thickness of 2 px
thickness = 2
# Using cv2.putText() method
image = cv2.putText(image, '', org, font,
                    fontScale, color_smiley, thickness, cv2.LINE_AA)
# Audio sound
if mask < withoutMask:
    wave_obj = sa.WaveObject.from_wave_file("sound/msgtxt.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()
```

Enfin, nous affichons les résultats et effectuons le nettoyage:

```
# include the probability in the label
label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

# display the label and bounding box rectangle on the output
# frame
cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

# show the output frame
cv2.imshow("Frame", frame)

# Displaying the image
cv2.imshow(window_name, image)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop
if key == ord("q"):
    break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

conclusion



Nous avons appris à créer un détecteur de masque facial COVID-19 à l'aide d'OpenCV, Keras / TensorFlow et Deep Learning.

Pour créer notre détecteur de masque facial, nous avons formé un modèle à deux classes de personnes *portant des masques* et de personnes *ne portant pas de masques*.


Nous avons affiné MobileNetV2 sur notre jeu de données *masque / pas de masque* et obtenu un classificateur **précis à ~ 99%**.

Nous avons ensuite pris ce classificateur de masque facial et l'avons appliqué à la fois aux *images* et *aux flux vidéo en temps réel* en:

1. Détection des visages dans les images / vidéos
2. Extraire chaque visage individuel
3. Application de notre classificateur de masque facial

Notre détecteur de masque facial est précis, et puisque nous avons utilisé l'architecture MobileNetV2, il est également *efficace en termes de calcul*, ce qui facilite le déploiement du modèle sur des systèmes embarqués (

source



<https://translate.google.com/translate?hl=fr&sl=en&u=https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/&prev=search&nto=auo>