

# Détecteur de masque mobile

Projet de développement IOT pour le master Big Data

Auteurs: Syphax Aouadene, Walid Djafri, Amassine Nacerddine, Mathias Tiberghien

Dernière révision: 22/06/2022

## Sommaire

<b>1 Introduction</b>	<b>2</b>
1.1 Documents annexes	2
1.2 Objectif	2
1.3 Problématique	2
1.3.1 Modèle de détection	2
1.3.2 Serveur d'image	2
1.3.3 Client final	3
<b>2 Architecture</b>	<b>3</b>
2.1 Matériel	3
2.1.1 Serveur d'image	3
2.1.2 Client final	4
2.2 Connectique	4
2.3 Diagramme de flux	4
<b>3 Système réalisé</b>	<b>5</b>
3.1 Modèle de détection	5
3.1.1 Diagramme de flux	5
3.1.2 Code	6
3.2 Serveur d'image	7
3.3 Client final	8
<b>4 Résultats obtenus</b>	<b>11</b>
<b>5 Difficultés rencontrées</b>	<b>12</b>
<b>6 Conclusion</b>	<b>12</b>
6.1 Améliorations futures	12

# 1 Introduction

Ce projet a été réalisé dans le cadre du cours “Objets connectés et données massives” pour la deuxième année du Master Big Data à l'université Paris 8. Il s'agit de réaliser une solution utilisant une plateforme mobile telle qu'Arduino ou Raspberry et de la combiner avec un algorithme de machine learning.

## 1.1 Documents annexes

Ce document fait parti d'un ensemble de documents qui comprend les éléments suivants:

- IOT\_Mask\_Detection\_Doc\_SA\_WD\_AN\_MT.pdf : le présent document
- IOT\_Mask\_Detection\_Video\_SA\_WD\_AN\_MT.mp4: une vidéo de test du projet
- IOT\_Mask\_Detection\_Code\_SA\_WD\_AN\_MT: un dossier contenant:
  - mask\_detection\_training.py : le code d'entraînement du modèle détection
  - mask\_detection\_client\_app : le dossier contenant le code du client final

## 1.2 Objectif

Nous avons choisi de créer une solution portable de détection de masque: une caméra branchée sur une plateforme mobile permet de capturer et transmettre un flux vidéo. Un ordinateur client utilise ce flux et un modèle de détection de masque que nous avons développé afin d'afficher si un humain porte un masque ou non.

## 1.3 Problématique

La solution proposée est divisée en 3 parties:

- Un modèle de détection
- Un serveur d'image
- Un client final exploitant le modèle de détection et le flux d'image

### 1.3.1 Modèle de détection

La problématique est la détection d'humain associée à une classification binaire (l'humain porte un masque ou non)

### 1.3.2 Serveur d'image

Le système doit pouvoir capturer le flux d'image vidéo d'un périphérique de capture vidéo et le transmettre sur le réseau Wifi.

### 1.3.3 Client final

Le client final doit pouvoir charger le modèle informatique de détection d'image et se connecter au flux vidéo fourni par le serveur en Wifi. Le modèle de détection est alimenté par le flux d'image de la caméra mobile et fournit un flux image indiquant si un humain est reconnu et porte un masque.

## 2 Architecture

### 2.1 Matériel

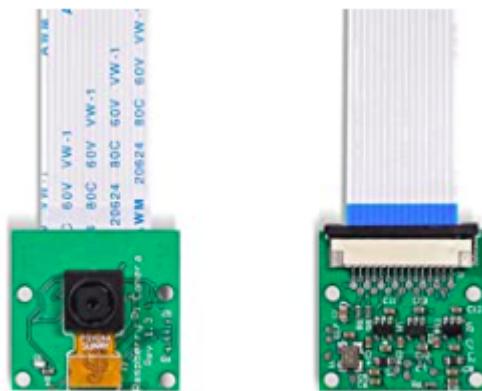
#### 2.1.1 Serveur d'image

Nous avons utilisé une Raspberry pi 3 hébergeant le système d'exploitation Raspberry pi OS.

La caméra est une caméra Raspberry Pi B01 de la marque TTLDA de résolution vidéo 1080p.



Raspberry PI 3



Caméra Raspberry Pi B01

### 2.1.2 Client final

Un pc portable équipé en Wifi et hébergeant le système d'exploitation Ubuntu.

## 2.2 Connectique

La caméra est connectée à la Raspberry PI via son port dédié et transmet le flux vidéo à l'aide de sa carte Wifi intégrée. Le client récupère le flux via sa carte Wifi intégrée.

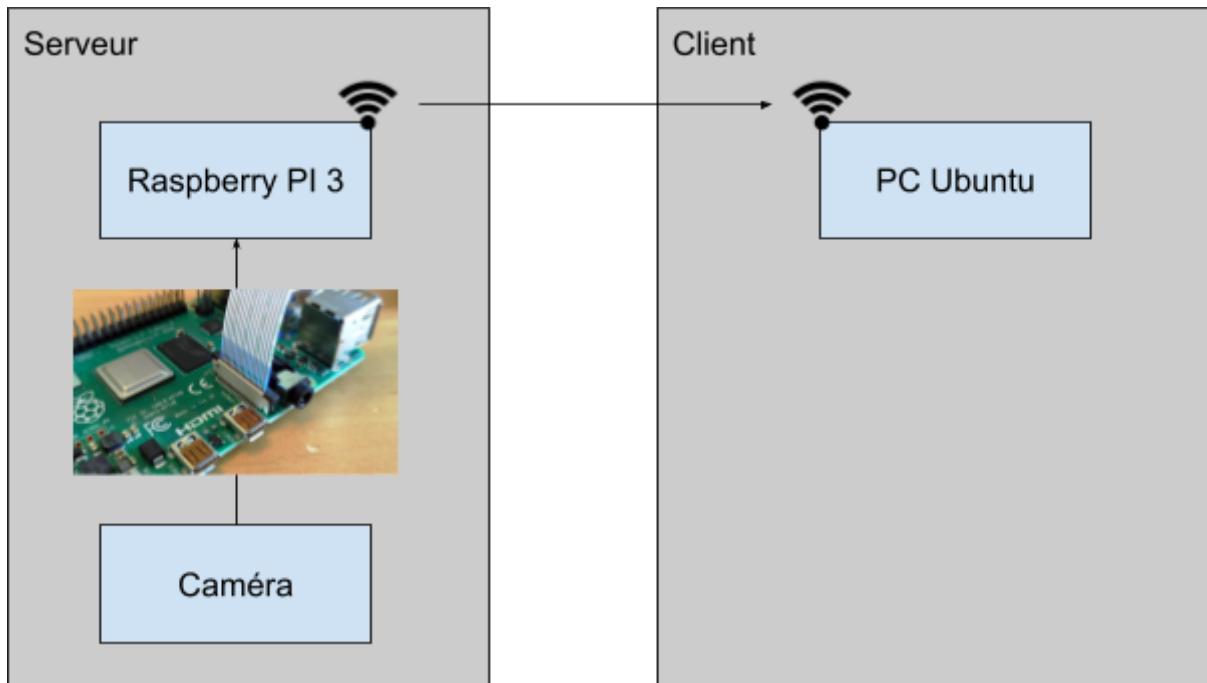
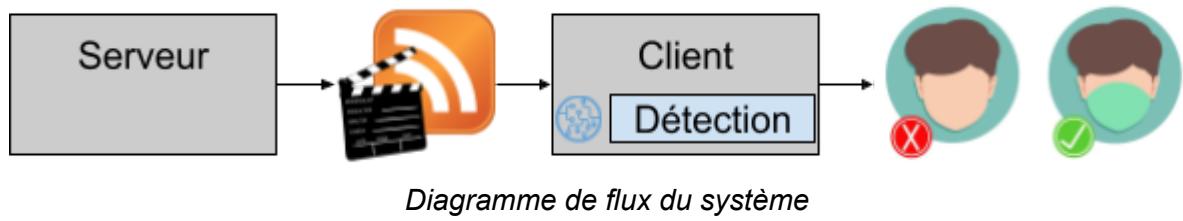


Schéma de connection entre les différents composants du système

## 2.3 Diagramme de flux

Le serveur transmet le flux vidéo au client qui utilise son module de détection intégré pour produire le flux vidéo enrichi avec l'information de port de masque ou non.



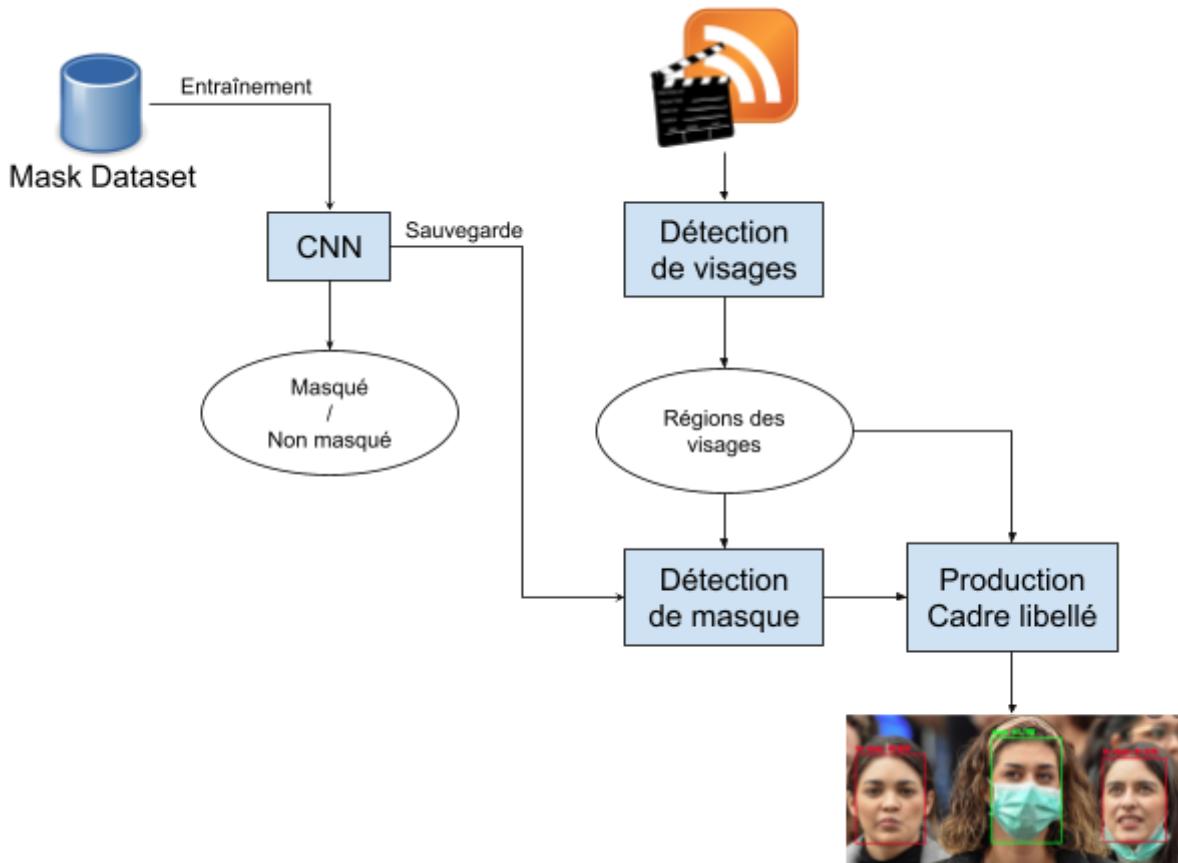
## 3 Système réalisé

### 3.1 Modèle de détection

Le modèle de détection s'appuie sur deux modèles:

- Un modèle de détection des visages issu de la librairie OpenCV
- Un classifieur binaire (visage masqué, visage non masqué) basé sur un CNN maison entraîné avec un jeu de données dédié issu du web

#### 3.1.1 Diagramme de flux



### 3.1.2 Code

L'intégralité du code ci-dessous est contenue dans le dossier annexe nommé IOT\_Mask\_Detection\_Code\_SA\_WD\_AN\_MT.

Le code ci-dessous, utilisant la librairie keras de tensorflow a servi à entraîner notre détecteur de masque.

```
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras import layers
import tensorflow as tf

import os # operating system

import cv2 # open cv
import matplotlib.pyplot as plt

batch_size = 64
img_height = 200
img_width = 200
data_dir = "mini_dataset"
# Dataset d'entraînement
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    class_names=['mini_face_dataset', 'mini_masked_face_dataset'],
    validation_split=0.2,
    subset="training",
    seed=42,
    image_size=(img_height, img_width),
    batch_size=batch_size,
)

# Dataset de validation
val_data = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    class_names=['mini_face_dataset', 'mini_masked_face_dataset'],
    validation_split=0.2,
    subset="validation",
    seed=42,
    image_size=(img_height, img_width),
    batch_size=batch_size,
)

num_classes = 2

# CNN Maison de classification binaire
model = tf.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(64, 4, activation='relu'),
    layers.MaxPooling2D(),
```

```

        layers.Conv2D(32,4, activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(16,4, activation='relu'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(64,activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    )

model.compile(optimizer='adam',
              loss=tf.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'],)

logdir="logs"

# Visualisation des métriques de validation
tensorboard_callback =
keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
write_images=logdir,

embeddings_data=train_data)
# Entraînement du modèle
model.fit(
    train_data,
    validation_data=val_data,
    epochs=12,
    callbacks=[tensorboard_callback]
)

from keras.models import model_from_json

# Sauvegarde du modèle en json
model_json = model.to_json()
with open("mask_detector.json", "w") as json_file:
    json_file.write(model_json)
# Sauvegarde des paramètres entraînable du modèles
model.save_weights("mask_detector.h5")
print("Saved model to disk")

```

## 3.2 Serveur d'image

Le serveur d'image est un simple script exécuté au lancement du système sur la Raspberry PI afin d'exposer le flux vidéo au moyen du protocole rtsp (real time stream protocol) à une adresse et un port prédéterminé.

Le code ci-dessous a été copié dans le fichier '/etc/rc.local' de la Raspberry PI afin de configurer l'IP statiquement et de lancer le flux vidéo en rtsp au démarrage de celle-ci.

```
ifconfig eth0 192.168.151.104 netmask 255.255.255.0 up
```

```
sudo raspivid -o - -t 0 -n -w 600 -h 400 -fps 30 -b 1000000 | cvlc -vvv stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

La commande raspivid permet de lancer la caméra sur la sortie par défaut et la commande cvlc

### 3.3 Client final

Le client final est une application web exécutée sur le PC client lisant le flux vidéo issu de la Raspberry et utilisant le modèle de détection sauvegardé et le modèle de détection de visage fourni par la librairie OpenCV.

Le code ci-dessous permet d'afficher le flux vidéo de la caméra augmenté d'un cadre précisant si l'utilisateur porte ou non un masque en s'appuyant sur la librairie Flask.

```
from flask import Flask, render_template, Response
import cv2
from tensorflow.keras.models import model_from_json
import numpy as np

app = Flask(__name__)
# Caméra Arduino
camera = cv2.VideoCapture('rtsp://192.168.151.104:8554/')
# Chargement du modèle de détection des masques
json_file = open('./static/models/mask_detector.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# Lecture des poids du modèles
loaded_model.load_weights('./static/models/mask_detector.h5')

# Chargement du modèle de détection des visages
face_cascade =
cv2.CascadeClassifier('./static/models/face_detector.xml')

print("the mask detector model is loaded successfully !")
target_names = ["with_mask", "no_mask"]

def gen_frames():
    while True:
        # Lecture de la trame
        success, img = camera.read() # read the camera frame
        if not success:
```

```

        break
    else:
        # La détection des visages requiert une image en nuance de
        gris
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

        # Détection des régions des visages
        faces = face_cascade.detectMultiScale(gray, 1.1, 4)

        # Pour chaque visage, on prédit le port ou non d'un masque
        for (x, y, w, h) in faces:
            roi_image = img[y:y + h, x:x + w]
            # L'image est redimensionnée pour correspondre au modèle
            de prédiction
            color_img = cv2.resize(roi_image, (200, 200))
            color_tensor = np.expand_dims(color_img, axis=0)
            result = loaded_model.predict(color_tensor, verbose=0)
            target_index = np.argmax(result, axis=-1)[0]
            text = target_names[target_index]
            frame_color = ((0, 255, 0)) if target_index == 0 else (0,
0, 255))
            cv2.rectangle(img, (x, y), (x + w, y + h), frame_color,
2)
            cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_SIMPLEX,
1, frame_color, 2)
            ret, buffer = cv2.imencode('.jpg', img)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n')

```

```

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video')
def video():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

if __name__ == '__main__':
    app.run()

```

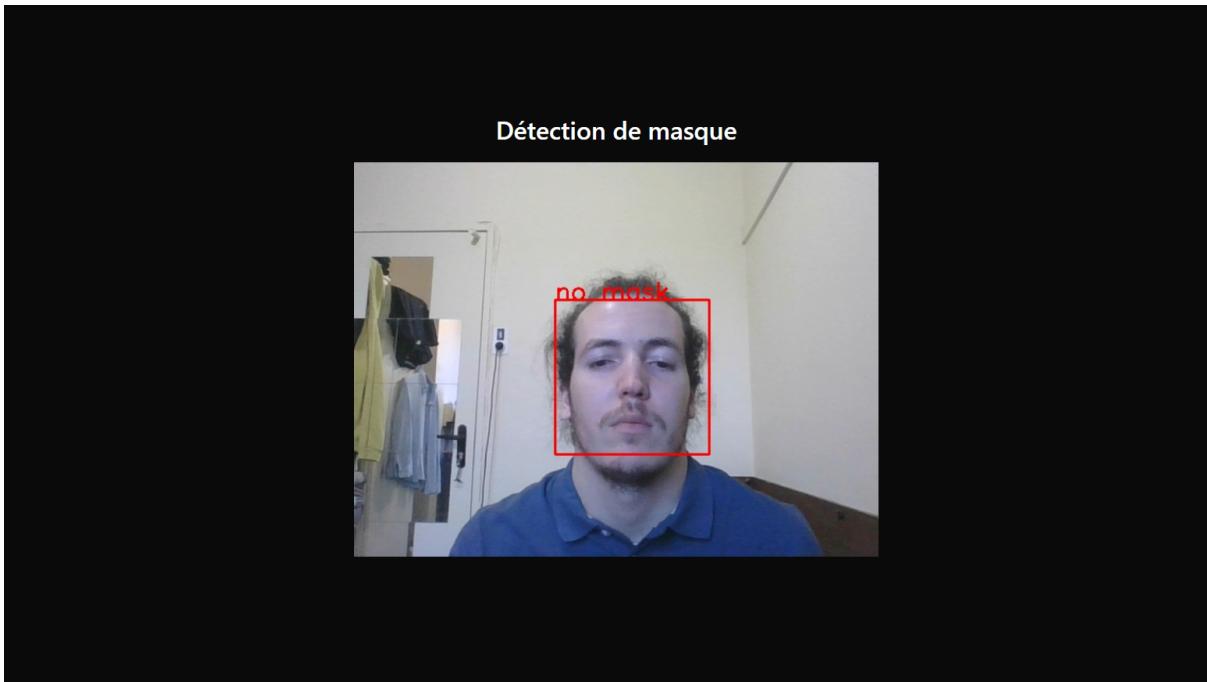
Le code ci dessous représente la page “index.html” affichant le flux vidéo généré avec le code ci-dessus:

```
{% extends "base.html" %}  
{% block container %}  
    <div class="title">  
        Détection de masque  
    </div>  
      
{% endblock %}
```

Ci-dessous le code de la page “base.html” qui définit le style et le layout de la page:

```
<!DOCTYPE html>  
<html style="overflow-y:auto">  
  
<head>  
    <meta charset="utf-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <title>Détection de masque</title>  
    <link rel="stylesheet"  
    href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.2/css/bulma.min.c  
ss" />  
    <link rel="stylesheet"  
    href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font  
-awesome.min.css">  
</head>  
  
<body>  
    <section class="hero is-fullheight is-black">  
        <div class="hero-head has-text-centered">  
  
        </div>  
        <div class="hero-body has-text-centered">  
            <div class="container is-text-centered">  
                {% block container %}  
                {% endblock %}  
            </div>  
        </div>  
    </section>  
</body>  
  
</html>
```

La capture d'écran ci-dessous montre l'application en cours de fonctionnement:



## 4 Résultats obtenus

Voici ci-dessous quelques captures d'écran obtenues avec notre projet.



*Captures d'écran issues du client final*

## 5 Difficultés rencontrées

La difficulté principale a été de contrôler les paramètres de diffusion vidéo afin d'obtenir la meilleure qualité possible: en effet avec des images de trop grande taille ou un nombre de trames par secondes (fps) élevé ou encore un débit binaire trop important (bitrate), la communication se trouvait saturée ce qui impactait le temps de latence et la qualité du flux vidéo. Heureusement, raspivid permet de contrôler ces paramètres de façon à adapter l'encodage vidéo en fonction de notre bande passante.

## 6 Conclusion

Ce projet nous a permis d'expérimenter:

- L'échange de flux vidéo entre deux systèmes par la communication Wifi
- L'utilisation de CNN pour résoudre un problème de classification binaire
- L'utilisation de la Raspberry PI
- L'application d'algorithmes de Deep Learning dans un contexte IoT

### 6.1 Améliorations futures

Il est nécessaire d'améliorer notre modèle de détection qui est très sensible à la lumière.