High level description

Overview

This project is designed to help with workplace/public safety in response to COVID-19.

We aim to ease the issue of non-compliance/forgetfulness with mask wearing before entering buildings by prompting users with on screen prompts based on our face mask detection model.

We have automated the process of temperature measurement; no human intervention is required. If the user has a temperature greater than 38°C the user is alerted of their elevated temperature and directed to not enter the building.

Part of the project aim was to allow future development/expansion of our model by providing scripts to create artificial dataset images (face masks overlaid on the user's choice of images) and making it easy for users to train a model using our algorithm, this process is described in detail within the dataset creation and model training documentation.

Detector.py

Detector.py uses multiprocessing and is setup as follows:

- Main process which handles the creation and output of the other processes.
- Face mask detection process (facemask_worker).
- Thermal data process (thermal_grabber_worker).

Face mask detection process:

The facemask detection process is passed the current webcam frame through a shared dictionary. The main method used within this process is detect_and_predict_mask takes in 4 parameters *frame (current webcam frame)*, *faceNet (face detection model)*, *maskNet (mask detection model)* and *confidence_arg*. The webcam frame is passed through the faceNet to obtain any faces and their coordinates, further the confidence_arg is used to filter out weak face detections. maskNet is then used to predict whether the faces in the detected area are wearing a mask or not. This method then stores the face location coordinates and the mask predictions in the shared dictionary for handling in the main process. Predictions on are made using multiple threads to avoid video lag.

Thermal data process:

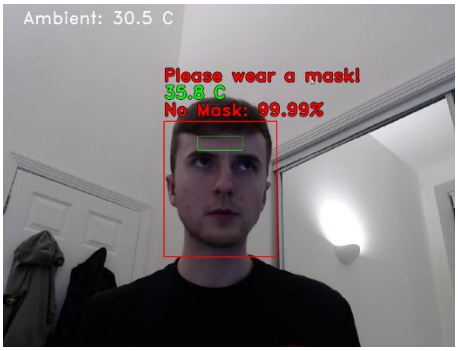
The thermal data process starts a subprocess of our thermal data grabber program which writes its debug and frame information to its *stdout* (standard output). We then parse the data from the *stdout* and store the relevant thermal data within the shared dictionary for handling in the main process.

Main process:

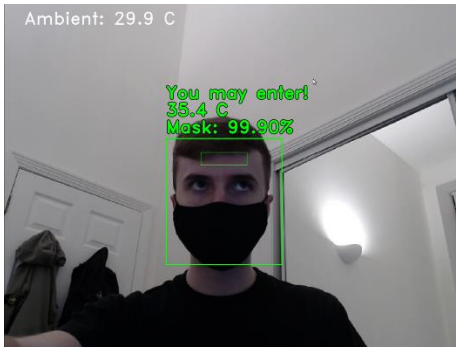
The main communication between the processes is through a shared dictionary contains the current webcam frame, the returned face mask detections and their confidence values along with the current thermal frame. Using the returned coordinates of the detected faces we take an average of the forehead temperature and use this value for detecting if the user is likely to have a fever or not (>38°C).

This allows us to process the data inside of the main loop where we draw the available information to a window using OpenCV.

Screenshots from the detector running with thermal mode:



No mask



Mask

Fever warning

Learning_algo.py

Learning_algo.py can be used to either train a pre-existing model or train a new model on a set of images for detecting masks on a human face.

The script carries out various tasks like loading image data, pre-processing, data augmentation, building a new fully connected head, loading the MobileNetV2 classifier (this model will be fine-tuned with pre-trained ImageNet weights) binarizing class labels, segmenting our dataset and printing the classification report.

We load in all the images using the load_img() method and preprocess it (converting it into an array and passing it through the preprocess_input() method). The label for the image is the name of the directory containing the image (e.g., mask and no_mask). The labels are then binarized using a LabelBinarizer, followed by partitioning the data into training and testing sets with a split of 80% to 20% respectively.

The ImageDataGenerator is used for data augmentation, where random rotation, zoom, shear, shift and flip are applied to images. This helps us to increase the size of the dataset and introduce variability in the dataset, without collecting new data. The base model is the MobileNetV2 classifier loaded with pre-trained ImageNet weights, a head for the model is the constructed various kinds of layers (dense, dropout, flatten etc.). The model uses an Adam optimizer with a learning rate decay schedule and binary cross-entropy, learning rate decay schedule is determined by the global variables for initial learning rate, number of epochs and batch size. The model is then fitted on the training dataset followed by an evaluation using the testing dataset. A classification report is then printed, and the model is saved in a h5 format.

Thermal grabber (thermal_grabber.cpp)

The thermal grabber part of the project is written in C++ and implements the low-level C Lepton SDK. Using the SDK we can set the appropriate thermal camera settings such as setting our gain mode to capture data within 0-150°C, running periodic Flat-Field Correction (FFC) normalisation which resets the image to reduce grain and. The main function of the thermal grabber is sending the received frames from the Lepton thermal camera to its *stdout* and resyncing the camera when the connection loses sync.

Other aspects of the project

Project installation – Described within **Installation guide**

Dataset creation, model training, and basic artificial face mask overlay – Described within **Dataset creation & model training**

Advanced artificial face mask overlay – Described within **Advanced artificial facemask overlay**

Thermal calibrator – Described within **Thermal camera calibration guide**