CS 688 Final Project: Face Mask Detection

James Erickson, Jose Quirarte, Jack Wang

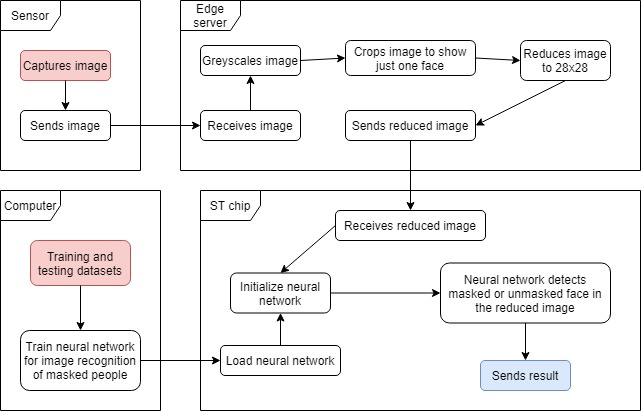
In this project, we set up an IoT mask detection system that uses a convolutional neural network loaded onto an ST microcontroller board to determine whether a person in a photo is wearing a mask. At the abstract level, the system could be divided into three components all connected via IoT: the sensor that takes a photo of a person, the edge server that crops the photo to just the person’s face, and the microcontroller board that predicts whether the person is wearing a mask. For our demonstration model, we used a smartphone as the sensor, a laptop as the edge server, and the ST board to run the IoT neural network. This project’s primary dependencies were uTensor and Create Face From Data. The uTensor library provided the neural network, which we train using images from the Real Time Medical Mask Detection dataset. We used Create Face From Data to extract faces from photos that could then be reduced and sent to the uTensor neural network. Our main challenges were working around the ST board’s limited computational resources, the initial unreliability of Create Face From Data, and the neural network sometimes returning false positives and false negatives. However, we successfully built a demonstration model able to detect masked or unmasked people with a good degree of reliability and accuracy.

1. **Introduction**

This project proposes an IoT system composed of a camera, an edge server, and a microcontroller board running a lightweight convolutional neural network (CNN) to detect whether a photographed person is wearing a facemask. We specifically want to use this CNN to perform object recognition on human faces to accurately predict whether a particular face is masked or unmasked. We will implement a demonstration setup using a STM32F401 Nucleo-64 board, a smartphone camera, and a laptop. Our neural network will be built using the uTensor library. We hope that this project can serve as a proof of concept for a cheap, automated mask detection system that can be deployed with minimal required expertise.

1. **Conceptual Design**

The general overview of our process is shown in the figure below.



Three major hardware components are required: an image sensor, an edge server, and an embedded board. Our setup uses a phone camera for the sensor, a laptop for the edge server, and an ST board for the embedded board. The image detection neural network will be trained on the same laptop and loaded into the ST board.

The process flow proceeds as follows:

1. The sensor captures an image of a person and sends it to the edge server.
2. The edge server performs the reduction algorithm on the image: greyscaling, cropping to show just the person’s face, and resizing to 28x28.
3. The edge server sends the reduced image to the embedded board.
4. The embedded board’s loaded neural network classifies the depicted face as masked or unmasked.
5. **Design and Implementation Details**

The project code is built on two other packages, uTensor and CreateFaceData. The uTensor library is based on the TensorFlow 2.1 neural network library. It provides an ultralightweight neural network that can be installed on ST microcontroller boards. The neural network itself is compiled by a Jupyter notebook, mnist\_conv.ipynb, that reads input images. The CreateFaceData package is based on the Python open\_cv library and the Caffe deep-learning framework. It performs extraction of faces from images.

Our uTensor implementation is directed by two scripts, main\_original.cpp and poll\_and\_send.py, and a Jupyter notebook, mask\_conv.ipynb. The mask\_conv.ipynb notebook is a modified version of mnist\_conv.ipynb that trains the neural network with reduced images, in our case of masked and unmasked people. As described in the high-level design, the reduction process performs grayscaling on an image, crops the image, and reduces the image to 28x28 pixels. The main\_original.cpp script runs the neural network to detect a mask. The poll\_and\_send.py script searches a user-specified directory for a new image, performs the reduction process on that image, and calls the main\_original.cpp script with the reduced image as input.

We sourced our training input data from the Real Time Medical Mask Detection dataset compiled by Arora et al. This dataset contains 5,521 images of both masked and unmasked people.

1. **Implementation Challenges**

The first challenge we encountered while implementing our design was getting uTensor’s default Helloworld to run correctly on the ST board. It specifically required TensorFlow 2.1.0 and Python 3.6.8, presumably because that was the current Python version at the time uTensor was coded. One group member had continual problems getting Python 3.6.8 environments to run on his Anaconda Powershell.

Another challenge was the resource limitations of the ST board, which among other things negatively impacted the neural network’s prediction accuracy. Because input images had to be in grayscale and they had to be reduced to 28x28, granularity was greatly decreased and the neural network had more difficulty accurately detecting masks in images with harsh lighting or when mask and skin color were similar. The ST board also could not accept multiple images in serial because this would cause issues with hex loading in the COM port. These problems are why we eventually decided that our ST board was only suitable for the proof-of-concept. To both provide reliable detection under real-world conditions and maintain minimal expense, we suggest the use of a slightly more powerful board that can accept RGB and higher-quality image input. ST likely provides such a board.

There were also problems getting the Create Face neural network to work reliably. An image could not have too much lighting or Create Face would return an error saying the image could not be detected. Faces had to be a certain distance away from the camera to be accurately detected and extracted. Eventually we did determine how best to take photos so that Create Face would consistently accept them. However, the unreliability of this library would present difficulties in real world conditions where maintaining consistently moderate lighting and distance cannot be guaranteed.

We also had a minor issue with the neural network returning false positives (incorrectly predicting masked) or false negatives (incorrectly predicting unmasked) when the lighting in a photo was too bright, the face was too distant or too close, or the face was angled too far away. Photos in which the mask and skin colors were similar in grayscale could also cause issues.

| Condition | Result |
| --- | --- |
| Very bright lighting | False negatives (sometimes) |
| Face is too distant or too close | Inconsistent predictions (sometimes false positives for unmasked, sometimes false negatives for masked)  Occasionally detection of multiple faces |
| Face angled 22.5 degrees away | False negatives when masked, correct predictions when unmasked |
| Face angled 45 degrees away | False negatives when masked, correct predictions when unmasked |
| Face angled almost 90 degrees away | False negatives when masked and false positives when unmasked  Occasionally, a face couldn’t be detected when masked at this angle |
| Mask and skin colors similar in grayscale | False negatives (almost always) |
| Darker skin color with lighter mask color | False negatives (almost always) |
| Lighter skin color with darker mask color | Correct predictions |

1. **Testing**

These demo steps assume that you are using a Windows 10 machine.

Demo link: <https://youtu.be/gGyDyFNz2hM>

Complete code repository: <https://github.com/erick016/facemask-detector>

Our code isolated for reading purposes (only what we wrote): <https://github.com/JackWang071/CS688-Facemask-Detection>

These are the steps to set up the run environment, with the correct dependencies, for the demo:

1. Use Windows PowerShell to git clone the uTensor Helloworld and Create-Face-Data-from-Images repos
2. Download the Real-Time-Medical-Mask-Detection dataset in Github
3. Extract Real-Time-Medical-Mask-Detection directly inside the root directory of uTensor Helloworld
4. Move poll\_and\_send.py to Create-Face-Data-from-Images root directory
5. Replace default uTensor main.cpp with custom main\_original.cpp file
6. Move mask\_conv.ipynb to root directory of uTensor Helloworld, this notebook takes a custom image set as input
7. Install Anaconda 3, November 2020 version
8. Open Anaconda Powershell in administrator mode
9. Create new Python 3.6.8 environment, run:   
   conda create -n myenv python=3.6.8 --no-default-packages  
   conda activate myenv
10. Use pip to install the following dependencies:

pip install mbed-cli

pip install utensor-cgen jupyterlab

pip install -Iv pyelftools==0.25

pip install -Iv protobuf==3.11.3

pip install -Iv opencv-python==4.5.1.48

pip install -Iv pyserial==3.5

pip install -Iv pillow==8.0.1

pip install -Iv tensorflow==2.1.0

1. Open mask\_conv.ipynb using Jupyter Notebook,   
   jupyter notebook mask\_conv.ipynb
2. Go to Kernel in the main menu, restart and run all cells to build the neural network using the MaskedFace-Net dataset images. This should generate a my\_model.cpp file inside the uTensor directory. Press Ctrl+C on the Anaconda Powershell to exit.
3. Before compiling, run this command:
   * pip install mbed-greentea
   * This will make sure the board changes the led from red to green when it finishes compiling
   * Unfortunately, the video doesn’t show this and was only caught after the recording. It did not affect the outcome but will explain why the led stays red after compiling. I tested it myself and it worked the same before and after installing mbed-greentea
4. To compile and install the neural network into the ST board, run this command:   
   mbed deploy (only necessary the first time)  
   mbed compile -m auto -t GCC\_ARM -f
5. Check the PATH environment variable, make sure that the folder containing the GCC\_ARM binary is there. Use the following command in case it is not:

mbed config -G GCC\_ARM\_PATH "C:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-update\bin"

These are the steps to actually run the demo.

1. Attach ST board
2. Use phone as sensor: connect phone to computer, select the file transfer option
3. Utilize WebDAV server app to get phone IP to mount phone as network drive
4. Map phone to network drive Z: http://[phone IP]:8080/
5. Open a second Anaconda or Windows PowerShell in regular user mode
6. Run poll\_and\_send.py, at the prompts enter the COM port and the path to your camera photo folder, e.g. Z:/DCIM/Camera
7. poll\_and\_send.py will search the indicated folder to grab a new image for mask detection
8. Must rerun for each new image
9. If execution is successful, the Anaconda Powershell should now display a hexadecimal representation of the image and the prediction label “masked” or “unmasked”
10. **Conclusion**

In conclusion, we successfully set up and tested a demonstration model of our mask detection concept. This demonstration model consisted of a smartphone for the sensor, a laptop for the edge server, and an ST board to run the neural network for mask detection. This neural network was built using the uTensor library.

We did have problems with the reliability of Create Face and with the computational limitations of the ST board, but we found solutions for how to best take photos that Create Face would accept and how to work around the ST board’s limitations. We also had minor issues with false positives and negatives. Future work on this concept could focus on further addressing these problems: using slightly more powerful boards that can run neural networks that accept RGB and higher-quality input or building more diverse datasets with greater variation in lighting, facial distance, and skin and mask colors, with which more robust neural networks can be trained.

**References**

Arora, R., Hemanth, J., Jain, R., Kataria, P., Madan, A., & Nagrath, P. (2020, June 10). Real-Time-Medical-Mask-Detection. Retrieved February 20, 2021, from https://github.com/TheSSJ2612/Real-Time-Medical-Mask-Detection/

Bartling, M., D., Pajak, D., & Tan, N. (2020, August 15). UTensor. Retrieved February 20, 2021, from https://github.com/uTensor/utensor-helloworld

Bhanot, Karan. (2019, March 4). Create-Face-Data-from-Images. Retrieved March 3, 2021, from https://github.com/kb22/Create-Face-Data-from-Images

STMicroelectronics. (2015). STM32F401RET6 Datasheet - Production Data. Retrieved from <https://www.st.com/content/ccc/resource/technical/document/datasheet/30/91/86/2d/db/94/4a/d6/DM00102166.pdf/files/DM00102166.pdf/jcr:content/translations/en.DM00102166.pdf>