Coronavirus Prevention System

Emanuel Cicortas
Department of Engineering and Computer Science
University of Central Florida
Orlando, Florida
t3sla@knights.ucf.edu

Dat Nguyen
Department of Engineering and Computer Science
University of Central Florida
Orlando, Florida
dat.npt@knights.ucf.edu

Ha Tran
Department of Engineering and Computer Science
University of Central Florida
Orlando, Florida
htran49@knights.ucf.edu

Phuong Anh Vu
Department of Engineering and Computer Science
University of Central Florida
Orlando, Florida
vpa.991997@knights.ucf.edu

***Abstract* — During the ongoing COVID-19 (Coronavirus) pandemic, it was determined that a simple and effective defense mechanism against the virus is wearing a face covering to prevent its spread between persons. Providing masks and collecting statistics on how many people effectively use them was a major hurdle in the battle against the disease. This project seeks to provide a solution by leveraging Artificial Intelligence to scan the area for both people wearing and not wearing masks, providing a mask to those not wearing one, and logging events for statistics tracking purposes.**

Keywords—COVID-19, machine learning, computer vision, mask detection, facial detection, deep learning

I. Introduction

The Coronavirus Prevention project is centered on safety and personal well-being of the general public by providing masks to those not wearing one, and providing a statistics logging server to keep track of how many people wear masks/don't wear masks. An alert system additionally prompts the user that a mask is required to proceed, and to take a mask from the system. The Coronavirus Prevention system is a concept that stems from the popularity of Pixar's Wall-E. The friendly looking robot can store items within its center compartment and scan its surrounding environment with the need of superficially complex internal or external design. The reason we chose Wall-E as the inspiration for our design is because Wall-E is cute. In the current time of chaos, having something cute sitting on the front door would be pleasant to look at, and encourage people to directly look at the camera giving us a better shot of their face.

Our design of the Coronavirus Prevention system will simplify the design of Wall-E even further. Since our primary purpose is to hand out masks, then a single scan when the person first enters the building should be more than enough. Thus, we will make the robot stationary. This will help cut down on cost even further and save us much more time as we do not have to implement a navigation system. The system will also provide a button to the user. By pressing the button, the user will be able to manually open the center compartment and take a mask without having to wait for the image processing unit to process or in the case that there is a software failure. The visual processing capability will still be there. We hope to retain the cuteness of the machine even after removing some of the preference design's features.

II. Motivation & Goal

A. Motivation

SARS-CoV-2 of 2019, or simply COVID-19, is a strand of coronavirus that causes the COVID-19 pandemic that has been spreading quickly across the entire world since November of 2019. As the virus was determined early on to be both extremely contagious and virulent, prevention mechanisms were explored by many agencies. The CDC recommends social distancing at 6-feet apart as the easiest way to prevent the spread, though this is not always achievable in high traffic or dense areas. Another way to prevent the spread, according to the U.S. Center for Disease Control (CDC), is to wear a simple face covering, or mask [1]. The CDC mentions that wearing a mask is effective at protecting yourself and others around you, and that at any time in public a mask should be worn to prevent COVID-19 spread. Even though these studies are widespread and some show that 83% of U.S. adults believe a face mask is effective in preventing spread [2], some surveys place the number of people not wearing face masks when in contact with others at a staggering 49% [2].

Another major thing we saw in our research of the COVID-19 pandemic was that many of the statistics available are conducted as surveys. People are skewed and biased when self-reporting, and surveys tend to suffer as a result of this. By providing real-world statistics based on how many people are wearing masks or not wearing masks, we can paint a much more accurate picture of how our response to the pandemic is going. These statistics may be very helpful, for example, to health experts and scientists trying to map the efficacy of mask wearing to real infection rates in the area. Using survey data for this example would create a skewed picture, and our system trumps comes out on top in usefulness.

Based on the research speaking that most people do believe face masks are effective, though a large number do not wear them all the time when in contact with non-household members, along with the research behind automatic dispenser systems not being readily available, we felt we had to design a solution to the problem. At the same time one of the most popular trends in Computer Science and Engineering taking off is Artificial Intelligence. By leveraging a computer to determine who is wearing a mask and who is not, we can make accurate decisions in very little time (in our testing over 80% accuracy at 10 frames per second).

B. Goal

Based on our previously conducted research and motivation, our main goal is to use a computer to make determinations on

who is/isn't wearing a mask, and subsequently automatically distribute a mask to those who need one to prevent the spread of COVID-19. A statistics tracking database will also run concurrently to provide scientists and health experts with data on health and behaviors of the general public. These experts may benefit greatly from identifying trends of certain areas and determine how well we are responding to the pandemic in general. Additionally, to make our machine accessible to as many people as possible, we must keep our costs low.

III. Component Requirements

A project will not go anywhere without having a specific set of component requirements of both hardware and software for the developers to achieve. The requirements ensure that the final product meets or surpasses the outlined measurements and performance. These requirements not only outline the functionality of the Coronavirus Prevention system, but they also lay the foundation for the design and implementation of the machine. These are the standards in which to the final product is to be built upon.

The purpose of the Hardware/Software requirements section below is to outline the specific, verifiable, and measurable outputs the product must achieve. At the bare minimum the finalized product must meet these requirements specified below to meet the intent of the project. These requirements will be verified as per the verification section in Testing And Evaluation.

A. Hardware Requirements

This hardware requirements section details what will need to be met in terms of hardware sign for the project. They are focused on and may include physical requirements or constraints but are not limited to these two. Figure 1 shows the image of our complete system.

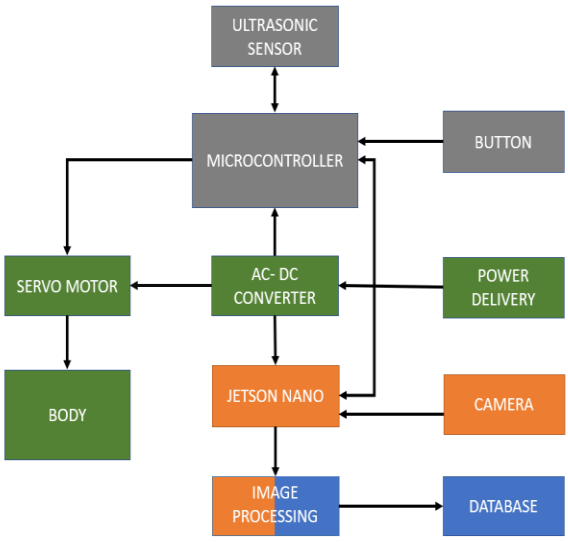


Figure 1: Coronavirus Prevention System Block Diagram

1) Housing

The first thing that someone will see when the look at the machine is likely to the outer body. In order to get someone to use the machine, the body must look non-intimidating, but rather, it must be as friendly looking, and as inviting as possible. However, since we are on a budget, we must carefully consider the build material, as well as the design of the housing itself. If we use high-end materials and design the housing to look sophisticated, the building cost would be off the roof. Thus, we opted for something a simple and cute looking design that can be done using cheap materials. This is where our design inspiration, Pixar's Wall-E, comes in. Pixar was able to design Wall-E to be so simplistic, yet at the same time, adorable. We want to capitalize on this simplicity for our robot to cut down the housing cost as much as possible.

Among all the materials that we can use like wood, aluminum, or molded hard plastic, we have decided to use 1/8" thick acrylic sheets as the body. Although acrylic does not have to elegant look of wood, nor is it as malleable as aluminum or hard plastic, it is much easier for beginners to work with than the alternatives. All we need are a drill and a saw. We will cut each panel separately, sand down the edges, and stick them together using hot glue.

The housing body must be large enough to accommodate all the hardware, as well as some masks. The hardware inside the body will be the Jetson Nano, PCB, and motor. Other components like the ultrasonic sensor and the camera module will be hanged on the outside. From a rough estimation, we found that a dimension of 5in x 5in x 5in would be sufficient to house all the components. However, to make it easier to mount the components and manage cables, we have decided to double all of the dimension to make it more spacious inside. There will be a door at the front of the robot that is controlled by a motor to make it easy for the user to reach in a fetch a mask. The initial design of the door is modelled after the old garage door that is lifted with a string-pulling motor. However, we have found that it would be easier for the motor to rotate on the vertical axis and open the door to the side than to open the door upward, so we have adjusted the design and mounted the door's hinge to the side.

2) Motors

The motor in the coronavirus prevention system is just a cherry on top of the sweet pie. It is not required to get the system to be functional, however, having a motor adds to the convenience of the machine as a whole. The motor's purpose is to open and close the front door of the robot's body. Because people are weary of the pandemic, they will not feel comfortable having to push the door in to fetch a mask. This is why we want to make the door to be automatic such that when our machine learning model have determined that the person is not wearing a mask, the microcontroller will turn on the motor to open the door for the person to reach in a fetch a mask.

Initially, we had wanted to use a cheap DC brushless motor to pull the string that is connected to the door. However, after the initial testing, we have discovered that the DC motor is not at all consistent. Given the same activation voltage, same PWM settings, and same active period, the number of rotations in each active period is slightly different from each other. This makes it so that sometimes the door is not shut all the way, or sometimes, the door is only half-opened. The reason behind the inconsistency is because the rotation is generated by charging

the magnets. Sometime, the magnets are not charged enough, causing the motor to not spin enough. Some other times, the magnets get charged too quickly and spin the motor too fast; even after the current is cut off from the motor, the angular momentum continues to spin the motor for a bit longer.

This is where the servo motor comes in. Servomotor is a specialized class of DC motors that is capable of enabling precise control of the angular velocity and position of the shaft. Servo motor's limitation over a typical DC motor is that it cannot rotate infinitely. Most servo motors have a 180-degree rotational axis. However, 180-degree is enough for our application. Having a limited spinning range decreases the risk of having the motor spinning too much. A servo motor box is just a gear box. Multiple gears are cascaded together to make it so that the rotatory torque is high. With high torque, the shaft would still be in position even when the motor is off. This makes it ideal for door opening and closing. We have decided to use the Tower Pro SG90 servo motor for its small size, relatively high torque, and a cheap price.

3) Sensors and Buttons

For this project, there will be 2 sensors and 1 button that will be implemented. Of those, only 1 sensor is required to accomplish the task determined initially. The rest are superfluous add-ons that aim to improve the quality-of-life for the users. When take out the one of the sensors and the button, the machine should still be functioning correctly. However, the codes will be more complex, and the wait time will potentially be longer. The required sensor for this project is the image sensor (photo camera module). Without out it, the tasks of facial detection and mask recognition would not be realizable. The quality-of-life enhancements to the machine are the proximity sensor and the override button.

For the proximity sensor, we have decided to go with the HC-SR04 ultrasonic sensor. We did not go with the awesome range of LiDAR or Infrared but chose a relatively short-range ultrasonic sensor because that is all we need. We do not need long distance tracking, but rather, we need high short-range accuracy. While the accuracy of LiDAR does not lack behind that of the ultrasonic sensor, depending on the LiDAR's wavelength, it could be harmful to the health of the users. We do not want to have the risk of having a LiDAR sensor functioning out of specification and cause harm to its users. The LiDAR sensors that are available on the consumer's markets simply do not go through the strict quality control procedures as those made for large corporations. Also, cost is also a factor that needs to be taken into consideration. Since we are on a budget, we cannot afford a high-quality LiDAR sensor. As for those cheaply made Chinese products, we just do not know if they function the way they say they do or if they would just work for one day, and the next they, the frequency increases drastically. We have no way of measuring light waves currently, so the product needs to be from a reputable manufacturer that we can trust. Unlike LiDAR, ultrasonic sensors are available from many US-based reputable manufactures. They are also much cheaper in comparison. We did not choose the infrared sensor because we do not know the lighting conditions of the place where the machine is going to be. If the room has too many infrared signals from sources like sunlight, lightbulbs or computers, the infrared sensor will experience constant interference and the result will not be at all accurate.

At first glance, the ultrasonic sensor may not be necessary in the system. However, it is a way that we can save on energy consumption of the system. If there are no proximity sensors, the Jetson Nano-controlled camera will have to continuously be taking pictures, and the Jetson Nano will have to continuously analyze the pictures to detect if there is anyone in front of the machine. Compared to the microcontroller and the proximity sensor, the Jetson Nano and camera use much more power. To conserve power, while the proximity sensor has not detected anyone, the Jetson Nano will be in sleep mode. Once there is someone in front of the machine, the ultrasonic sensor will send an interrupt signal to the microcontroller. The microcontroller will send a message to the Jetson Nano via UART, subsequently causing an interrupt signal to occur in the Jetson Nano, waking it from sleep.

As stated previously, since our goal is facial recognition, we will need to be able to capture high resolution images of the face of a person. To achieve that goal, for our photo camera, we have gone with the IMX219 camera module, with the resolution of 3280 x 2464. To put that into perspective, the resolution of a 4k camera is 3840 x 2160. The horizontal length of our picture is a bit shorter than 4k, but in exchange, our image is taller. The reason that we have gone with a development camera module rather than a full-blown camera is the cost. A full-blown camera is around 20x more expensive than a comparable camera module. The difference in cost is stemmed from the inclusion of an onboard image preprocessor. For our application, we do not need the image to be sharpened before being processed by the AI, so a camera module alone is more than enough.

Lastly, let us talk about the use of a push button in the system. The button is just the typical single push, single throw button. This is added as a failsafe option in case someone or something is blocking the ultrasonic sensor, preventing it from detecting an approaching object. This also helps in case the robot is mounted in high places and the person is too short to be detected by the ultrasonic sensor. Whatever the case may be, all the user has to do is to push the button to override the ultrasonic sensor detection. After pressing the button, the Jetson Nano will be turned on to do the AI processing sequence like usual. The reason we use the button to override the ultrasonic sensor and not the jetson nano is because the IMX219 camera module has a much wider field of view than the ultrasonic sensor, so the chance of it not being able to capture an image that does not contain a person is low.

4) Microcontroller

The microcontroller that we are using for this project is the MSP430G2553. The MSP430G2553 ports a 16-bit processor with 16KB of flash memory, 2 16-bit timers with 24 GPIO lines, 3 of which are PWM capable, and it supports basic communication protocols like UART, SPI, and I2C. Rather than using a G2ET development kit for the MSP, we minimized the footprint by putting the MSP430G2553 microcontroller chip on the same PCB as the power delivery system. We do not need the fancy features provided by the development board like easy computer connection with the micro-USB port, nor do we need the 8-segments LCD screen. We do not even need a reset button

for our current project. We only need direct access to the pins for the peripheral control and to set up the buttons. Also, since the interior of the robot is relatively small, putting another board into the body would take up too much space. We believe doing so is the right choice as the size of the microcontroller itself is only about $1/100^{th}$ of the size of the entire board.

As for the port and pin connectivity to and from the MSP430G2553 microcontroller, here is how it works. DVCC is the power source of the chip, so it is connected to the 3.3V rail of the power supply. Obviously, another terminal is needed to complete the circuit; so, we then connect the DVSS to the ground terminal of the power supply. Because the reset pin is active low, we must, at all time, bridge the reset pin with DVCC via a 47k resistor to hold it high. Without hooking it up the voltage source, the reset pin will float between high and low and constantly reset the microcontroller. This phenomenon is similar to a boot loop many people have experienced with the personal computer. The push button will be configured to bit 3 of port 1 of the microcontroller. As stated in the motor section, the motor will be hooked up to the motor driver, then the motor driver will be linked with the microcontroller via P2.0, P2.1, and P2.3. Finally, the buzzer will be linked to P2.4 and P2.5.

According to Texas Instruments' datasheet, the MSP430G2553 consumes no more than 1.4mW even under heavy load. MSP430G2553 takes in 3.3V as the operating voltage. Under heavy load, the input current is measured to be roughly 400uA, which is within expectation. What more, under low-power mode, the power consumption dropped drastically. With low-power mode 3, the amount of current it needs dropped to roughly 2uA, making the required power decreased to 6.6uW. Since the MSP430G2553 does not need that much power to operate, it can be powered very easily. Since our 3.3V rail of the power delivery can deliver up to 5W of power, it should be more than enough to power not just one, but multiple MSP430G2553 at once.

5) Image Processor

The image processor used in this design is ultimately chosen to be the NVIDIA Jetson Nano. The decision behind choosing a graphics/image processor development board in this case was simple due to both the specifications of the Nano being far above its competitors in the same price range and abundant support from NVIDIA for deploying AI applications. Among the abundant support for the Jetson platform is a face mask detection flow that has already been fleshed out officially by NVIDIA as an example design and the methodology can be easily found online.

The main reason for choosing the Jetson Nano is that it has the strongest GPU among the all the processors in the similar price range – a 128 CUDA core NVIDIA Maxwell-series. Essentially, since we have an artificial neural network at the heart of our processing that consists of many simple multiplication and addition operations needed to be done in parallel, a CPU would not be very efficient at doing this since it is both inherently sequential and has a low number of cores. What a neural network needs for good performance are many "dumb" cores since we do not care about concepts such as cache or a complex ISA provided by CPUs since each node in the neural network is doing just simple arithmetic operations. Something that needs to be realized as well is that we need to actually be able to sufficiently utilize the GPU available. In terms of support, the Nano has a massive platform provided by NVIDIA for CUDA core drivers, making sure we get the maximum performance possible out of the GPU, so this is not a problem since it is built into their deployment SDK.

Power consumption was not a major reason for choosing the Jetson Nano, however we must still take it into account when designing the hardware. The Jetson Nano requires a steady 5v/2a DC power source in order to run at its peak 10W power usage for optimal performance. This is not taking into consideration the carrier board or attached peripherals, which may end up taking more power. NVIDIA's recommendation is a 5v/4a DC source to provide a stable voltage rail with no droops below 4.75V to prevent brownouts. This is a hard requirement due to the fact that we need to be able to run the Nano as hard as possible since we cannot afford any performance losses in our AI algorithm.

There are three different ways to power the board. The micro-USB route has been seen to be very unreliable in testing and will not be used. The 2.1mm barrel jack connector can supply up to 5v/4a and is a possible choice, however the simplest way to power the nano is through the 5v VDC headers (pins 2, 4) shown in Figure 38. They may take up to 2.5A per pin, however we will not require the full 5A supply since the Nano itself takes up 2A, and the only peripheral connected is the Camera module which should in no way take up more than an entire amp of power. Therefore, a loose requirement is 5V/3A connected to pins 2, 4 on the header.

6) Power Delivery

For power delivery for this project, we are going to use an external power brick (AC-DC converter) which converts from 120VAC to 12VDC, then, our PCB will implement 2 rails with voltage regulators that turn the input 12VDC to 5VDC and 3.3VDC respectively in order to run motor driver, MSP430G2553, and the Jetson Nano.

Now, let us walk through the entire circuit to see how it operates. First, the input signal is generally in the range of 120V(rms), 60Hz AC signal that comes out from United States' wall outlets. The input signal is connected directly to the primary side of step-down the transformer (integrated into the power brick). The step-down transformer that we used in this circuit that converts from 120VAC to 12VDC. To regulate the output voltages, we initially used the LM7805 and NCV4274CDT33RKG linear regulators. We later found out from our professor that the LM78XX parts are extremely inefficient in voltage conversion. The efficiency rating of the 78XX parts is only around 50%. If we have all the energy in the world, such low efficiency would not be a problem. However, the world's electrical energy is limited, so we do not have the leisure to waste energy. The current design's efficiency is 80%+, which is a huge up lift from the 50% efficiency. The 12V output from the DC power brick will be taken in by the LM25085A regulator to be converted to 5V. The 3.3V rail runs parallel to the 5V rail, so we will run the 12V output voltage from the power brick into the TPS563249 regulator to convert it into 3.3V. The use of each of the output voltage is the same as mentioned before. The only difference between the old set up

and the new one is that the new one is more stable and more efficient. The efficiency number and the design of the new circuit is taken directly from the TI's WEBENCH power designer tool.

7) PCB Design

The obvious step immediately after designing the multi-rail power delivery system is to design the PCB. There are many free and paid options that can be used to design a PCB. Due to our inexperience and lack of expertise, we will be using Autodesk Eagle to design our PCB. We chose Eagle because it is the software that we have some experience with, and it is also that the software that is widely used by our professors at the University of Central Florida.

After designing comes the manufacturing process. There are many PCB manufactures that we can use, from those that are based in the US to those that are based overseas. To save on cost, the obvious option is to go with a Chinese manufacture. Initially, we chose PCBWay to manufacture and assemble our PCB. However, after a 2-months delay from Lunar New Year, the board came with manufacturing defects and many included components were dead-on-arrival. We got the board to be remanufactured, this time by the more reputable JLCPCB. After the board arrived, the sent the board, alongside the components to The Quality Manufacturing Services at Lake Mary (QMS), a manufacture with existing partnership with UCF, to be assembled. The board is a simple 2-layer PCB with size of 100mm x 90mm in size and with 31 unique components. Since the component count is not high, QMS agreed to assemble the components for us for free. An image of our PCB presented in Figure 2.

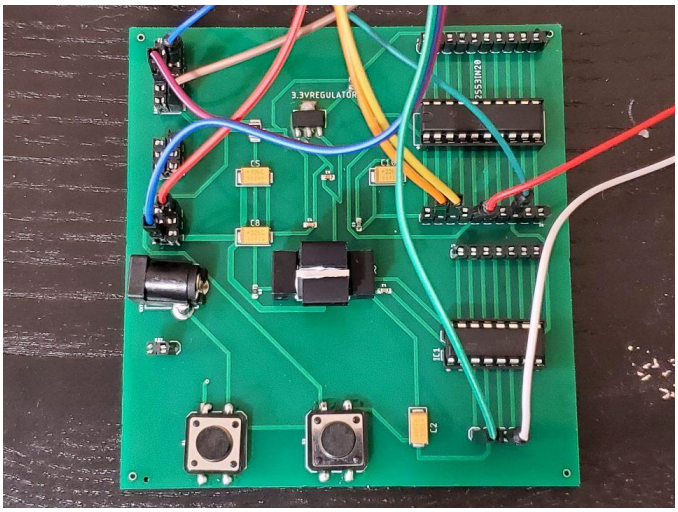


Figure 2: PCB

B. Software Requirements

The below subsections detail our design of the software components of the robot. For obvious reasons, our project cannot be solely hardware-based. We need to have a seamless combination between the hardware and software to make the user's experience as stress-free as possible. Aside from listing the software components, we will also go over the purpose of the part in the grand scheme of functionality of the system.

1) Machine Learning Model Training and Pruning

The training of our machine learning model proved to be the most complex part out of designing the system. In order to create an effective model, we decided to follow NVIDIA's Transfer Learning Toolkit process due to its popularity in the AI community, and proven performance.

Training a machine learning model is a complex process. Thankfully NVIDIA has standardized their transfer learning process. The sub-sections below detail the process we went through to train our model.

"Transfer Learning" is the method of machine learning which uses the knowledge in an already trained model and applies it onto a different, but related problem. The idea of transfer learning is reusing the model developed for a certain task as the initial model for a related task which does not have sufficient data. Instead of starting the process by collecting data, the users can begin with the model that "solves" a related task, which is extremely beneficial in saving a huge amount of training time due to the fact that it uses a previously trained and specialized model.

Transfer learning is useful when we do not have enough data for a new domain and there is already a pre-existing data pool that can be reused for the new problem. The NVIDIA Transfer Learning Tool Kit provides pre-trained models that can be re-trained to in order to achieve any arbitrary model. Also, TLT does not require high-level understanding in deep learning, so it is easy to use that make our own custom model for our project. The flow we will follow is shown in Figure 3.

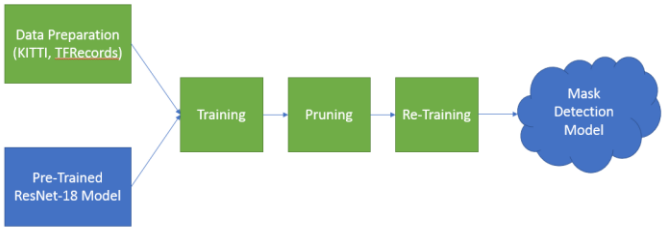


Figure 3: Transfer Learning Toolkit (TLT) Flow

To quickly get up to speed on training our model, we first had to configure our environment to perform the actual training. Since a GPU is more or less required for training, we used a laptop with an NVIDIA GTX 1060. To configure the Linux environment, we used Windows 10's Windows Subsystem for Linux 2 (WSL 2). This second iteration of WSL allows for use of the GPU inside the Linux subsystem. Inside this Linux subsystem we ran the nvidia:tlt-streamanalytics docker container for access to a standardized TLT environment.

The model training process is as follows: First, we download the dataset of faces with mask and without mask and convert them into KITTI format using the data2kitti.py script given, and then into TFRecords format for the underlying TensorFlow API that TLT runs on top of. The TFRecords conversion was done using the tlt-dataset-convert executable built into the environment. We also download the pretrained model from NGC (NVIDIA GPU Cloud). Specifically, we will use the DetectNet_v2 with the ResNet-18 for accuracy and

efficiency. After that, the quantized TLT deep learning model is deployed with DeepStream SDK to identify faces with mask or no mask. By using NVIDIA TLT and NVIDIA DeepStream SDK, it saves a significant amount of time in training as well as allows us to achieve high accuracy performance in real-time.

The main step in this software flow is the actual TLT training done inside of the Jupyter Notebook provided by NVIDIA, inside of the TLT docker container, inside of WSL 2. Once our labeled images are converted to TFRecords format and we have the pre-trained DetectNet_v2 model from NGC, we can now continue to training.

The first step in training is re-configuring the detectnet_v2_train_resnet18_kitti.txt config file in the specs dir which controls many parameters for the training such as paths to files, validation split, and many others that have been pre-configured by NVIDIA. The ones that must be reconfigured are the image width, height, fold number (for validation split of images), and other training parameters such as number of epochs, batch size, etc. along with the paths to the training TFRecords.

We set our parameters to 960x544 (the original DetectNet_v2 model resolution), 120 epochs, and a batch size of 12. Depending on the amount of both system RAM and GPU DRAM is available we must lower the batch size parameter. The batch size defines how many samples we work through during the current epoch before we update the model parameters. A higher batch size implies better training for the model at the cost of taking more time and more memory since we are performing operations on a larger subset of data. In training, we found that a batch size of 12 was the most optimal setting – the highest we could go with acceptable training speed without running into memory starvation.

After the parameters are set, we begin the training by using the tlt-train executable. Our final accuracy for the unpruned model ends up at 82% and 84% for mask or no mask inference, respectively. This is close to NVIDIA's calculated numbers of ~85% for each, though less likely due to a different split of training data, less epochs, smaller batch size, etc. Had we run training for more epochs on better hardware with larger batch size this accuracy would likely increase, although non-deterministically since training varies on every run.

The next stage in the TLT flow is the pruning stage, followed by re-training. From a high-level pruning is known as "Data compression" technique that takes nodes in the network that have a very small impact on the overall decision of the output and strips them away from the model. This method helps the model to achieve better solution, reduces the complexity of the model as well as improving the accuracy by decreasing overfitting the model. Also, this greatly reduces inference time and is extremely valuable to do in our case since we will be running on relatively weak hardware (Jetson Nano), and can use whatever performance increases we can get within reason. Theoretically, this technique will only reduce the size of decision trees without increasing errors. However, in this specific case, removing nodes from our neural network, we lose the accuracy since some may have a ripple effect through the network. An acceptable loss in pruning for our application is an arbitrary 5% loss.

To actually prune the model, we run the tlt-prune with a "-pth #" option setting that acts as a threshold value and is a floating-point number between 0 and 1. A larger number closer to 1 prune more of the network while a smaller number near 0 leaves more intact. It's obvious that the higher the value is, the more accuracy we will lose since more of the network will be stripped away, so it ends up being a balancing act between performance gain and accuracy loss.

Additionally, due to the fact that we will be losing accuracy upon doing this it is necessary that we re-train the network once pruning finishes, and that we re-evaluate the network to see if we pruned too much. If our accuracy suffers too much (more than the arbitrary 5% stated before), we will have to go back and prune less, and re-train once again. Thus, next we prune the model with a starting value of 0.4 and re-evaluate. The re-evaluated model accuracy is shown in the figure below and is satisfactory at ~83% for both mask and no mask. Later on, we will deploy our model to the Jetson Nano, and if inference performance (frames per second) is satisfactory, we are done. However, if either accuracy or inference performance is lacking too much, we may re-visit the pruning and modify the threshold value to modify our tradeoff in this area.

2) Deploying Machine Learning Model

NVIDIA further provides a framework for deploying custom models that utilize the TLT flow called Deepstream SDK. We chose to use NVIDIA's deployment platform due to having good community support, and it allows us to follow their example designs. Since we used TLT, deployment is as straightforward as modifying the example designs.

We chose to follow the Deepstream Python example applications for deploying our model due to familiarity with Python in UCF's Artificial Intelligence course. We first had to modify the Primary GPU Inference Engine (PGIE) configuration to point to our custom model, and set all the relevant parameters to our model (image height, width, confidence threshold).

Once the configuration was set, we modified the example Python application. The GStreamer pipeline is first generated by the NVIDIA provided code. Next, we wait inside of a loop forever until we receive serial data from the MCU in "sleep mode". If the serial data received is an "S" ASCII character, we have our start condition and begin playing the pipeline. A set number of frames are generated by the camera (128), and at the end of the 128 frames we record the metadata found by the AI – we check here for people seen wearing masks or no masks. Statistics logging then occurs with the database, and we inform the MCU if we have seen someone without a mask. The loop then restarts. An image of the mask recognition demo is presented in Figure 4 below.

Figure 4: Real Time Mask Test

The actual internals of the Deepstream SDK uses a GStreamer pipeline setup to transfer data between elements' sources and sinks. These elements may perform video or audio decoding, image transformation, inference, and many other functions. To actually "run" the AI we need to first set up a pipeline, which is quick to pick up since NVIDIA provides many examples. To get any valuable data out of the AI, a probe is inserted at the end of the onscreen display element. Inside of this probe we extract valuable metadata from the buffer object including frame number, number of each element seen in the image (masks/no-masks), and interact with our database. The deployment flow is shown in Figure 5.

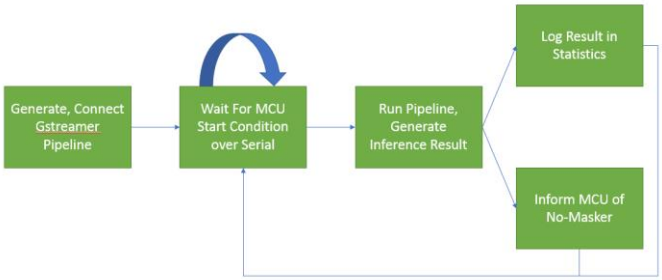


Figure 5: Jetson Nano SW Flow

3) Database Communication

Our database system is based on the LAMP (Linux, Apache, MySQL, Python) stack. Linux, MySQL, Python are such powerful platform that contributes essential capabilities to the stack as well as providing the sufficient requirements for database system for our mask detection task.

Linux (The operating system): Serve as the first layer for this stack model. Linux is free and open-source operating system, which provide flexibilities and configuration options to run our server.

Apache (The web server): Since we are planning to host the database locally, we do not use the Apache to deliver the website on the internet.

MySQL (The database): MySQL is known as open-source relational database management system for storing information. This will be the second layer for our model, that will store the data, and deliver the information as requested.

Python (The programming language): We use Python as our primary scripting language to create dynamic database server.

This database system will have three main layers as mentioned above, Linux sets the foundation for the stack model, following by MySQL and Python. Python communicates with MySQL for fetching or storing data referenced in the code. We will manage database directly using SQL commands instead of using a web server. This stack model is efficient enough to handle the changing of data depends on date, time and user IDs.

The database will automatically generate a unique number each time a person comes through the mask detection system with a unique identifier to ensure the data recorded is unique. After a primary key is created (null values will be rejected), the new data associated with the key guarantees two persons will never have same value and allows us to parse data within the table quickly.

After 128 frames captured the person coming toward the camera within distance, every person in the video will be labeled as "Mask" or "No Mask", which will be recorded in the metadata file. This data file is imported to MySQL database with "PersonID" will be the primary key to distinguish with the preexisting data. "PersonID" is "AUTO_INCREMENT", which allows us to generate unique number automatically when a new data record is added into the table, following by "Mask", will be stored in Boolean type: Mask: 1 and No-mask: 0. To store date and time information, we use SQL datetime type in order to define specific time a person has experience mask detection system.

IV. Testing And Evaluation

A. Hardware Testing

With some exceptions, the majority of the hardware testing is accomplished by plugging in the PCB and measure the voltage and current at each of the lead to ensure the output is as expected. The testing is presented in the Table 1.

Table 1 Electrical Components Testing

Component	Test	Outcome Determination
DC Power Brick (12V rail)	Use the multimeter to check the output voltage and the maximum current output	The output voltage should constantly be at 12VDC, and the maximum current should be no less than 5A if the power brick is functioning correctly
5V rail	Use the multimeter to check the output voltage and the maximum current output	The output voltage should constantly be at 5VDC, and the maximum current should be no less than 6A if the power brick is functioning correctly
3.3V rail	Use the multimeter to check the output voltage and the maximum current output	The output voltage should constantly be at 3.3VDC, and the maximum current should be no less than 2A if the power brick is functioning correctly
Push button	Use the multimeter to measure the resistance of the switch as the button is pressed	If the resistance drops from high to close to 0, then the button is working correctly.
Motor	Connect the terminals of the motor with a voltage source and a PWM source	If the motor turns on and rotates, then it works
MSP430 pins	Set the pins to high and low. Use multimeter to measure the voltage at each pin with respect to ground	If the high voltage is around 3.5V and the low voltage is around 0V, then the pins of the MSP430 works
MSP430 timer	Configure PWM for a PWM-enable pin on the MSP430. Use the oscilloscope to measure period and the active percentage within the period	If the period and the active percentage matches what was set in the software, then the internal timer works perfectly
MSP430 UART	Configure UART. Set up a serial communication with a computer. Send data using UCA0TXD	If the computer is connected to the MSP430 and the correct data is printed onto the serial console, then UART works
MSP430 Interrupt	Configure timer interrupt every second. Toggle an LED whenever there is an interrupt	If the LED toggles regularly, the Interrupt routine works
Proximity sensor	Configure UART on MSP430. Place an object in front of the sensor. Print out sensor's status to console continuously	If the sensor's readings are close to the actual distance of the object to the sensor, then the sensor works
Buzzer	Run a PWM signal through the buzzer	If the buzzer makes sound and the sound changes as the PWM signal changes, then the buzzer works
Image camera	Connect the camera to the Jetson Nano	If there is an image, then the camera works
Jetson Nano	Connect to a monitor	If there is an output image, then the Jetson Nano is working

B. *Software Testing*

The testing for the Jetson Nano is shown in the tables below. The MSP430 software is collaterally tested in the hardware section.

Table 2 Camera Testing

Test Name	Test camera feed
Purpose	Verify Jetson can utilize Camera image Verify Camera functionality
Test Materials	Jetson Nano Jetson Nano power supply Camera Module Keyboard, Mouse, HDMI Monitor
Prerequisites	Power supply, keyboard, mouse, HDMI monitor and camera plugged into Nano Jetson JetPack installed on Nano
Procedure	Pull the NVIDIA "jetson-inference" Docker container Follow instructions from NVIDIA's "Running the Docker container". Run application, follow the script and use command line *./video-viewer /dev/video0*
Expected results	This test should return the video stream in real time using Jetson Nano

Table 3 Mask Detection AI Testing

Test Name	Mask Detection
Purpose	Verify the AI model qualitatively, independent of the Jetson before deployment.
Test Materials	face-mask-detection.ipynb Jupyter Notebook (face-mask-detection github) Trained and Pruned Face Mask Detection AI model (resnet18_detector_pruned.tlt)
Prerequisites	Windows 10 PC with WSL2 installed along with Ubuntu 20.04 Docker installed and started resnet18_detector_pruned.tlt trained and pruned AI model Environment set up for face-mask-detection docker container Jupyter Notebook Testing images consisting of people with mask, no mask in the image
Procedure	1. Modify the detectnet_v2_inference_kitti_tlt.txt file to point to the resnet18_detector_pruned.tlt model. 1. Drop testing images inside DATA_DOWNLOAD_DIR in the Docker container. 2. Run step 8 of the Jupyter Notebook with the pruned AI model. 3. Visually confirm the output images.
Expected results	The output images are expected to have been labeled with the persons wearing masks/no-masks in the image correctly. Red bounding boxes signify a person with no mask, and green signifies a person with a mask.

Table 4 Jetson Nano Deployment Testing

Test Name	Jetson Nano Deployed Model
Purpose	Qualitatively determine that the Jetson can run our AI model using DeepStream in real-time from the Camera.
Test Materials	Jetson Nano Jetson Nano power supply Camera Module Keyboard, Mouse, HDMI Monitor
Prerequisites	resnet18_detector_pruned.etlt pruned and deployed mask detector model Jetson Nano Camera connected, verified Jetson Nano DeepStream SDK set up
Procedure	Launch the Deepstream SDK program pointing to the .etlt model. The video should be viewed on the screen with the AI results present.
Expected results	The program should output the video stream for mask detection with green box outlining if a person wearing a mask, and a person without a mask, the bounding box is red. This bounding box will appear around human's face that been detected and will follow the face until the person move out of the camera's view.

Table 5 Jetson Nano GPIO Interface Testing

Test Name	Jetson Nano GPIO
Purpose	Quantitatively determine that we are able to read from the GPIOs and output to the GPIOs used for the communication between the Jetson and the MCU.
Test Materials	Jetson Nano Jetson Nano power supply Camera Module Keyboard, Mouse, HDMI Monitor Multimeter
Prerequisites	Python GPIO script
Procedure	1. Set the Tx pin to "out", Rx pin to "in" using the GPIO script. 2. Read the Tx pin with multimeter set to voltage mode with positive connected to Tx GPIO pin, negative connected to the GND on the GPIO. 3. Use Python GPIO script to set Tx pin HIGH. 4. Read voltage from Tx pin. Verify it is at 3.3v. 5. Read Rx pin using GPIO script. Verify it is at logic level LOW. 6. Connect Nano 3.3v DC to Rx GPIO pin. 7. Read Rx pin using GPIO script. Verify it is at logic level HIGH.
Expected results	The program verifies functionality for the Nano's GPIO pins for Rx and Tx.

Table 6 Jetson Nano Full System Test

Test Name	Jetson Nano Full System Test
Purpose	Determine that our entire system works as expected on the Nano's end.
Test Materials	Entire system connected
Prerequisites	Complete assembly, connection from MCU to Jetson Nano GPIO pins
Procedure	User steps in front of distance sensor to activate the system with no mask. The system should initiate and trigger the Nano to begin inference.
Expected results	Upon no mask being seen, the mask case should open along with the buzzer ringing.

V. Conclusion

In conclusion, Coronavirus Prevention System is AI based, face mask detection, that uses streaming data from the camera, combined with machine learning techniques to detect a person without a mask. The team was able to meet all specifications that are required for this project. Throughout testing and integration, we noticed that sometimes the door is open slower than usual, but this would be an easy fix.

Overall, the team learned many valuable skills that not only for individual growth but also the idea that everyone's contribution can lead to something that is socially beneficial, during the current pandemic COVID-19. We believe this project would be a great way to demonstrate our skills and knowledge that we have learned from coursework at UCF including hardware design and software design.

Acknowledgment

We would like to give thanks to Dr. Samuel Richie for overseeing our project from the very beginning to the end. We received many valuable advices from him, ranging from choosing our development platforms to manufacturing our PCB. We would like to thank the Quality Manufacturing Services at Lake Mary for helping us assembling the PCB components free-of-charge.

References

[1] "Cloth Face Cover Guidance." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/cloth-face-cover-guidance.html

[2] Key, J. (2021, January 21). Half of U.S. adults don't wear masks when in close contact with NON-HOUSEHOLD MEMBERS. Retrieved from https://dornsife.usc.edu/news/stories/3388/understanding-coronavirus-in-america-mask-use-among-us-adults/

[3] "About COVID-19." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, www.cdc.gov/coronavirus/2019-ncov/cdcresponse/about-COVID-19.html.

[4] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[5] Analog Devices, Inc. "Understanding How a Voltage Regulator Works." *Understanding How a Voltage Regulator Works | Analog Devices*, 2009, www.analog.com/en/technical-articles/how-voltage-regulator-works.html.

[6] Bhargava, Hansa D. "Coronavirus History: How Did Coronavirus Start?" *WebMD*, WebMD, 15 Apr. 2020, www.webmd.com/lung/coronavirus-history

[7] Geitgey, Adam. "Build a Hardware-Based Face Recognition System for $150 with the Nvidia Jetson Nano and Python." *Medium*, Medium, 25 June 2019, medium.com/@ageitgey/build-a-hardware-based-face-recognition-system-for-150-with-the-nvidia-jetson-nano-and-python-a25cb8c891fd/

[8] Han, Song, et al. "Learning Both Weights and Connections for Efficient Neural Networks." *ArXiv.org*, 30 Oct. 2015, arxiv.org/abs/1506.02626.

[9] "JetPack SDK." *NVIDIA Developer*, 21 Oct. 2020, developer.nvidia.com/embedded/jetpack

[10] "Jetson Nano Developer Kit." *NVIDIA Developer*, 9 Oct. 2020, developer.nvidia.com/embedded/jetson-nano-developer-kit

[11] Kulkarni, Amey, et al. "Implementing a Real-Time, AI-Based, Face Mask Detector Application for COVID-19." *NVIDIA Developer Blog*, 13 Oct. 2020, developer.nvidia.com/blog/implementing-a-real-time-ai-based-face-mask-detector-application-for-covid-19/

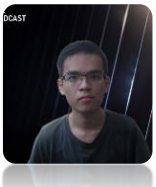
[12] "NVIDIA DeepStream SDK." *NVIDIA Developer*, 13 Nov. 2020, developer.nvidia.com/deepstream-sdk

[13] "NVIDIA Transfer Learning Toolkit." *NVIDIA Developer*, 13 Nov. 2020, developer.nvidia.com/transfer-learning-toolkit

[14] Pachev, Sasha. "Understanding MySQL Internals." *O'Reilly Online Learning*, O'Reilly Media, Inc., www.oreilly.com/library/view/understanding-mysql-internals/0596009577/ch01.html

TEAM MEMBERS

Emanuel Cicortas is a Computer Engineering student whose interest is in VLSI. He will be working on the Jetson Nano side to help design the mask detection software, train the AI model, and deploy to the Jetson. He has been interning for Lockheed Martin in FPGA design for 2 years and will begin work as an ASIC Verification Engineer for NVIDIA after graduation.



Dat Nguyen is a Computer Engineering student who specializes in microcontrollers. For this project, he will be in charge of programming the microcontroller as well as testing the peripherals. After graduation, he will continue his academic endeavor at UCF.

Ha Tran is the only Electrical Engineering student in our group, and his primary interest is in Power Systems. Ha has been working as an intern at Jacobs for 1.5 years, and he has experience in the renovation of electrical engineering design services for a variety of projects encompassing building power, power distribution, lighting, controls, etc. In this project, he specializes in signal processing, so he is in charge of designing and testing the power delivery of our system. Upon graduating, he will transition from an intern to full time employee at Jacobs as an Electrical Designer.

Phuong Anh Vu is a Computer Engineering student who specializes in software design. In this project, she will be in charge of building database server and helps with software design including facial recognition and machine learning algorithms. She plans to pursue working career in computer engineering profession in short-term future and gain a Master in Machine Learning in long-term future