Automated Access Management System with Decentralized Access Verification and Real Time Facemask Detection

Shubham Kapoor - 2019UCO1543
Mahima Singh - 2019UCO1517
Faisal Ahmed - 2019UCO1549
Suhail Malik - 2019UCO1567

ABSTRACT

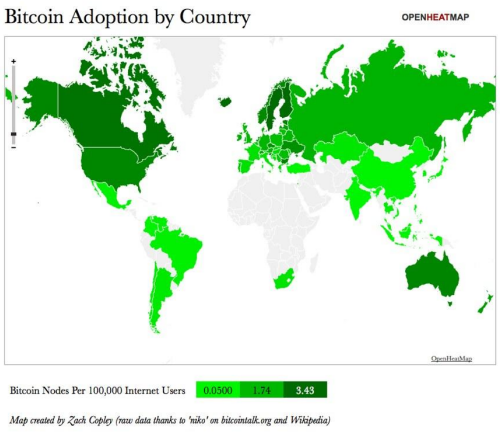
A decentralized access management system is a more reliable and secure way to manage access than traditional access control systems. This can be used in offices and both public and private properties. Our project has 2 web applications, one acts as an access management dashboard and the other is used when a user requests access to any "Room" via scanning of a QR code in front of the room, the QR code is generated using our own protocol and is linked to the RoomID of the room which is stored on the Ethereum blockchain. Along with this we have built a facemask detection model using CNN that is hosted on a flask server which, upon scanning of the QR code, it reads an image from our ESP32 camera web server and predicts if the person is wearing a facemask or not, once we have both bits of information, i.e, if the person is allowed in the room, and if they are wearing a facemask or not, the decision of weather to allow them into the room is then made and the actuator is triggered accordingly.

1. INTRODUCTION

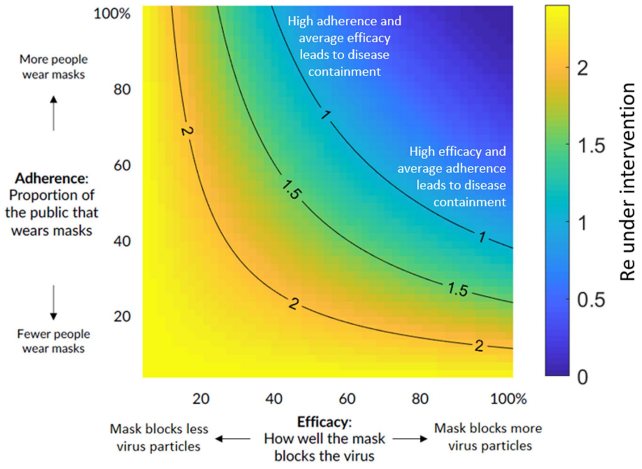
With the constant rise in security threats in both public, personal or corporate property, the requirement for a system that by itself implements all the concepts of Authenticity, Accountability, Integrity and securely either grants or denies access to a "room", both physical or digital is only going to increase.

Moreover, If we compare the rise in security threats with the amount of innovation done in the space to stop such threats, we can see a disproportionate growth of the amount of threats, this is true specially when it comes to cybersecurity and piracy. We intend to solve this very problem by making an IOT based system that automates this process.

We have used blockchain technology and deployed our smart contract on the Ethereum network to implement the required business logic and in a world where the adoption of the decentralized network to build secure and scalable products is rapidly increasing, we have developed a product that reduces fixed costs and removed the requirement to carry access cards or identification cards with you everywhere you go that requires a special authorization, making it both, more convenient and more secure.



[6] *Fig 1.1*



[7] *Fig 1.2*

Fig 1.1 shows us the adoption of bitcoin by displaying the number of bitcoin nodes per country. This shows us that the majority of the countries in the first world have accepted bitcoin and this trend is seen in most cases with the adoption of other technology as well. Ethereum has a very high direct correlation with bitcoin and this can be seen in the financial markets from the prices of these cryptocurrencies as well. Hence the trend for mass adoption of Ethereum as a platform for deploying contracts to serve as a database for decentralized applications is expected to increase, specially in the field of finance and security, we intend to use this to our advantage by assuming the adoption of cryptocurrencies and building on top of it.

The specific problem we aim to solve is that of granting or denying access of a physical location to or from a user, by creating a product that could be integrated with other smart doors or smart ecosystems present in both private or public properties for verifying whether an individual has permission to enter the premises or not, this could then be extended in use cases where digital access is managed and this would then eliminate the need of any hardware component.

According to *Fig 1.2* the more number of people that wear a facemask, the more the efficacy of each mask would be as it takes two people, one with the virus and the second person without the virus, for transmission of the virus from one to another, and we have hundreds of interactions in a day hence if a high percentage of people wear masks then the efficacy of each mask would increase and this would result in flattening the curve and resulting in the eventual end of the pandemic. Hence it's increasingly important to wear a facemask especially in public places.

While requesting permission to enter any physical location it's important in today's era to have a real time mask detection system that would grant access even to a user that has permission, only if they are wearing a face-mask, for solving this problem we will be using deep learning to implement our own CNN model that would receive an image which would be clicked by an ESP-32 camera upon the request of access to a room, this image would then be sent to a server on which our model would run and then predict. This would be used to determine if the user with permission should be granted access or not.

2. OBJECTIVE

We are going to develop a decentralized web application (DAPP) that can be used as a dashboard for the owner of a room to manage access to any room that they are the owner of by giving them certain privileges such as :-

1. Granting Permission to users
2. Revoking Permission from users
3. Transferring Ownership to users
4. Creating a room that they own
5. Viewing their Allowed Rooms
6. Viewing their Owned rooms

Each user can login to the dashboard directly from any Ethereum wallet that they trust and they will have also be given certain limited privileges such as the following :-

1. Creating a room that they own
2. Viewing their Allowed Rooms
3. Viewing their Owned Rooms

When a room is created, a QR code will be generated which has to be scanned to verify if the Ethereum wallet account, from which the QR was scanned, is allowed to enter that "Room" or not, depending on which the user would either be granted or denied entry.

When a QR scan happens, the ESP32 camera must click a picture of the user that has scanned the QR to request access and send that onto the server on which our CNN model would then check if the picture contains an individual who is wearing a facemask or not.

Then, both bits of information i.e., whether the user is Allowed in that Room and if they are wearing a Facemask, will be checked and if they BOTH are True, access will be granted.

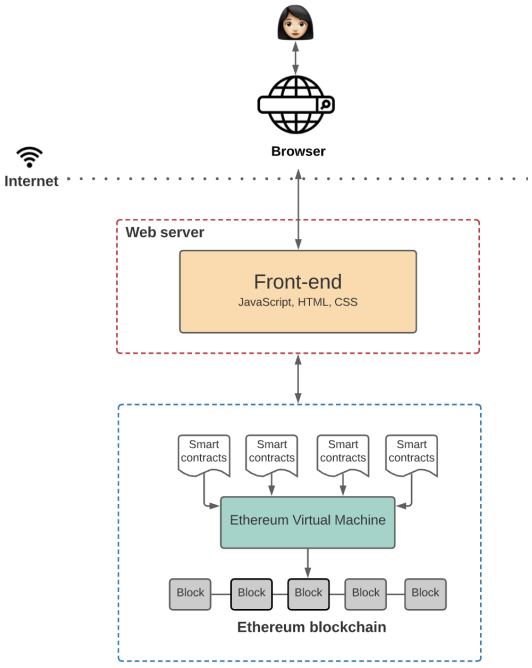
3. METHODOLOGY

Our work is divided into 7 components:

- 3.1- Web 3.0 Introduction using Ethereum
- 3.2- Dashboard to Manage Access
- 3.3- QR Code Generation Protocol
- 3.4- Access Verification
- 3.5- Facemask Detection
- 3.6- Hardware component
- 3.7- Gluing Everything Together

3.1. Web 3.0 Introduction using Ethereum

Our Decentralized web application is built on a **ReactJS** front-end and a **NodeJS back-end,** we have used the Infura API suite as a middleware to interact with our Ethereum smart contract hosted on the **Ropsten Test Network.** We've used **EthersJS** to interact with the Ethereum wallet from our web app.



[8] *Fig 3.1.1*

[1] *Fig 3.1.2*

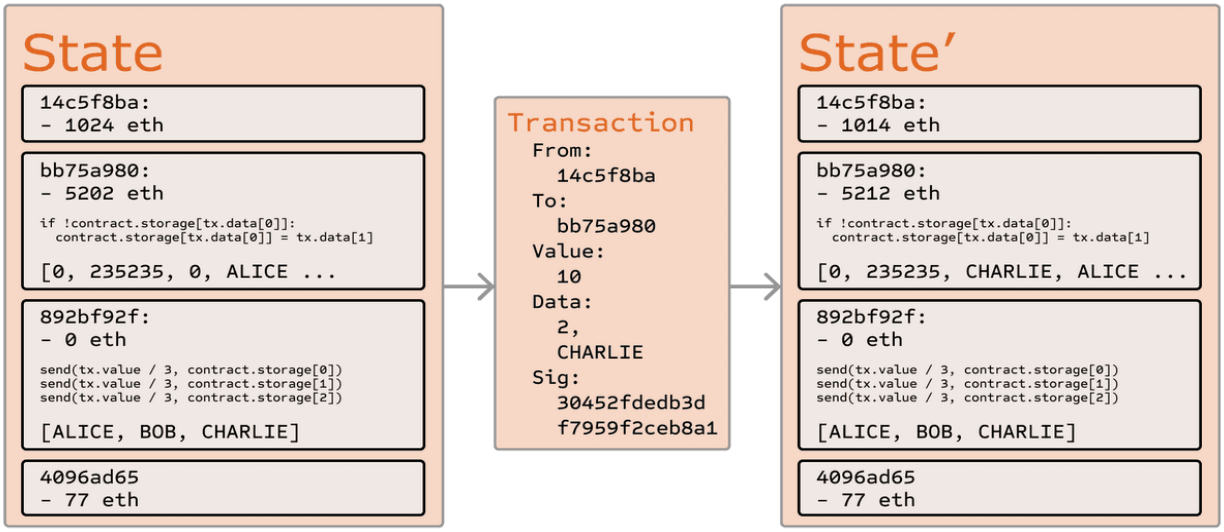


Fig 3.1.2 shows the state transition function of the Ethereum blockchain, each transaction has a "From", "To", "Value", "Data", "Sig" information and the state is then updated to the new state after the transaction on every node once the transaction is validated by the validators. This is done by Ethereum which uses a proof of stack consensus mechanism to validate a transaction.

Once the smart contract is tested and deployed, it can be interacted with using the Infura API suite directly from Web, Android or IOS applications, this is done because blockchain by default is a chain of nodes where each node represents a server, not a client, hence to read or write on the chain from any client side application or interact with any smart contract these third party APIs are used which make the process easier. This is shown in *Fig 3.1.1* how the user can interact with our smart contract directly from the front-end of our web application.

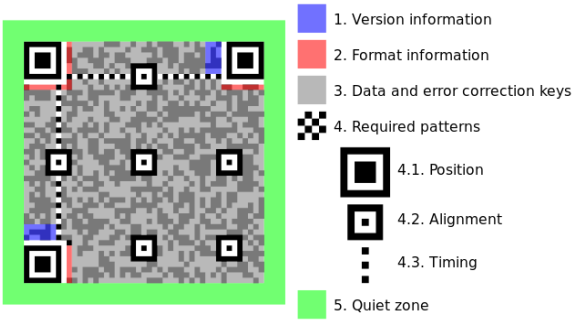
3.2. Dashboard to Manage Access

This component of our project was to create a web application which would act as an interface to show a user what all rooms are they Allowed to enter and what all rooms are they the owner of, along with that the users are given ability to create any new room of which they will be the owner of by default.

The users who own a particular room will be given additional privileges such as the ability to grant permission of that room to another user, they also have the ability to revoke permission of that room from a user who currently has permission. The owner can also transfer their ownership to some other user or some other Ethereum "Account" and then that user would be given ownership of that particular room. While performing a write operation the owner will have to pay a gas fee for the validation of the transaction.

3.3. QR Code Generation Protocol

When a "Room" is created by any user, the Ethereum "Account" of that "Room" would, by default be declared as the owner of that room, then a QR code will be generated which will have to be scanned by any other person who wants to enter that "Room" for verification of their permission to enter.



[9] *Fig 3.3.1*

Fig 3.3.1 Essentially shows how QR codes work, QR codes are essentially a 2d version of barcodes, this allows them not only to store more information but also have an up to 30% error tolerance/damage rate.

For the purpose of our Project, the QR code that's been generated is then used to verify permission to a room, which means it would have to be placed in front of a room which would then be scanned by a user who wants to enter that room.

The Hash of the QR code of a room is set to be the **URL of our deployed web application for Access Verification + "/" + the RoomID of the room that has been created**, hence upon an event of scanning the QR code from the QR scanner of their Ethereum wallet the user would be redirected and their permission to enter that room would be verified.

3.4. Access Verification

Upon scanning the QR code from the QR scanner of the user's Ethereum wallet, the user would be redirected to our web application instantly where a read operation would be performed on the Ethereum Blockchain where it will be checked if the Ethereum "Account" that was logged in when the QR code was scanned, is allowed in the "Room" with the RoomID that was derived from the QR code Hash value or not. Refer to *Fig 3.4.1* for the format of the QR code Hash, from here, the RoomID is extracted, the public key of the Ethereum "Account" is taken when the user connects their Ethereum wallet with our web application, this happens automatically, now we have enough information to check the existence of any permission of "Account" with public key X and a Room with ID of RoomID.

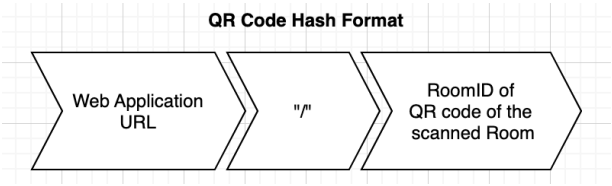
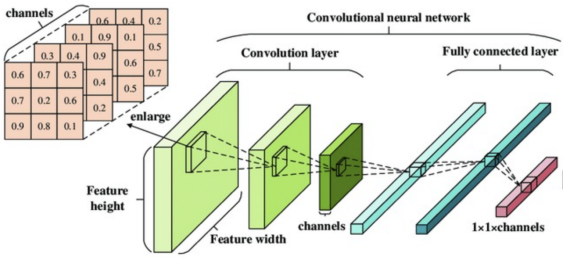


Fig 3.4.1

Once this check happens, the binary output is then sent to the **NodeJS** backend of the web application, when this happens, the data is then sent to a middleman server which is present to make the application more scalable and is our implementation of an MQTT broker, upon publishing the event, the data is then sent to our hardware component that's subscribed to that event and then that is used to trigger the actuator that would be linked to a smart door or a smart ecosystem to grant access if the facemask check also passes.

3.5. Facemask Detection

Facemask detection is achieved in this Project using CNN, once the data was collected, the next step was to label the data into “mask” and “no mask” labels and resize and format the data, the data was then divided into a train test split with 80% of the images in the training set, then, as seen in *Fig 3.5.1*, each image while evaluating first goes through multiple convolution layers, then the pixels are linearized to form a 1x1 channel and then the model is trained on the data using the backpropagation algorithm.



[10] *Fig 3.5.1*

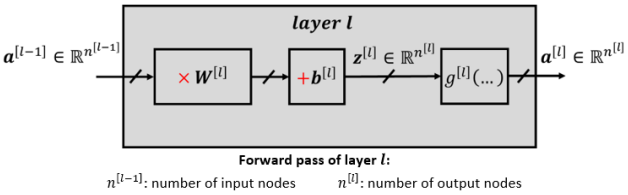
Once the data is preprocessed, it’s time to then train the model using backpropagation. *Fig 3.5.2* shows us the equations used for training the CNN. refer to *Fig 3.5.3* for the block diagram of a forward propagation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} = \left[\boldsymbol{W}^{[l]}\right]^{\mathrm{T}} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}} * g^{[l-1]\prime}\left(\mathbf{z}^{[l-1]}\right)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}} \cdot \left[\boldsymbol{a}^{[l-1]}\right]^{\mathrm{T}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}}$$

[11] *Fig 3.5.2*



[11] *Fig 3.5.3*

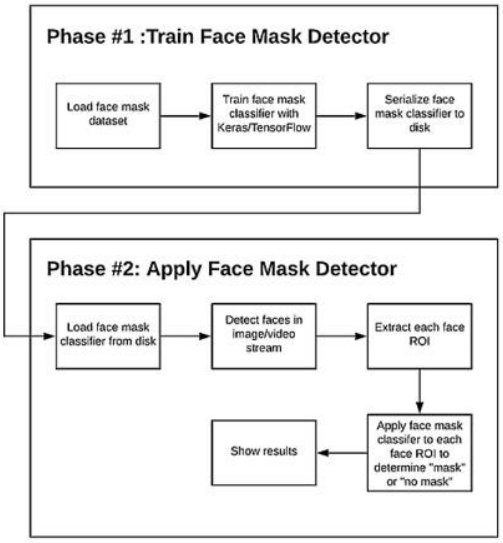


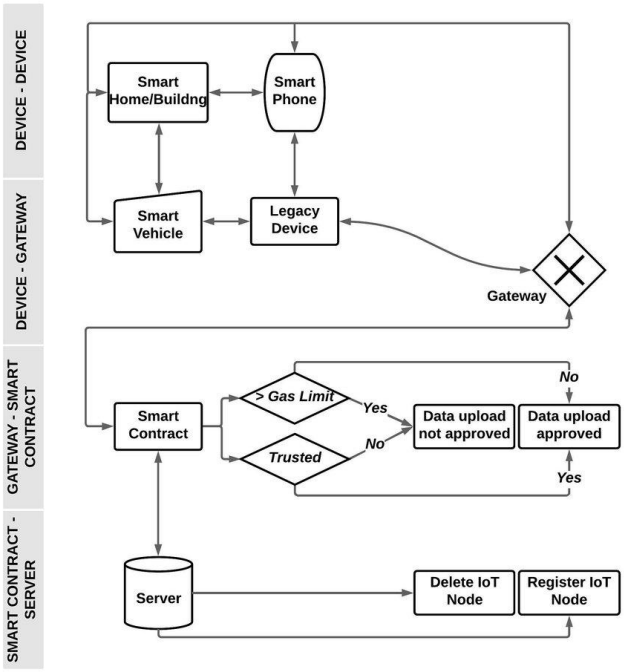
Fig 3.5.4

Once the model is trained, we deploy it onto a Flask server that runs the model on any image it reads from the request from the ESP32 camera web server stream, this request happens every time the middleman server receives an event when the QR code is scanned by a user who is requesting access. The model then predicts whether the user who has scanned the QR code is wearing a facemask or not, if a face isn’t detected then the model returns a “None”, if a face is detected then the model will return a “mask” and “no mask”. This data is then sent back to the ESP32 camera web server which then triggers the actuator accordingly.

3.6. Hardware Component

There are 2 main hardware components required for the IOT based implementation of this project, ESP32 camera and ESP8266 Node MCU, the ESP32 camera is used as a web server which responds to the request from the Flask server to read an image from the stream so that the model can predict if the user is wearing a facemask or not. The ESP8266 Node MCU is also used as a web server to serve requests from the middleman server, this is done so when the middleman server publishes an event on which the ESP8266 Node MCU is subscribed to, then the binary data is read and if the user's permission is verified then the actuator is triggered and the door is opened, if permission was not given to the user and access was denied then the event is ignored and the actuator is not triggered.

3.7. Gluing Everything Together



[12] *Fig 3.7.2*

Refer to *Fig 3.7.2* for the general data flow diagram when using Blockchain with IOT, the device interacts with a gateway server which then interacts with the smart contract.

Fig 3.7.1

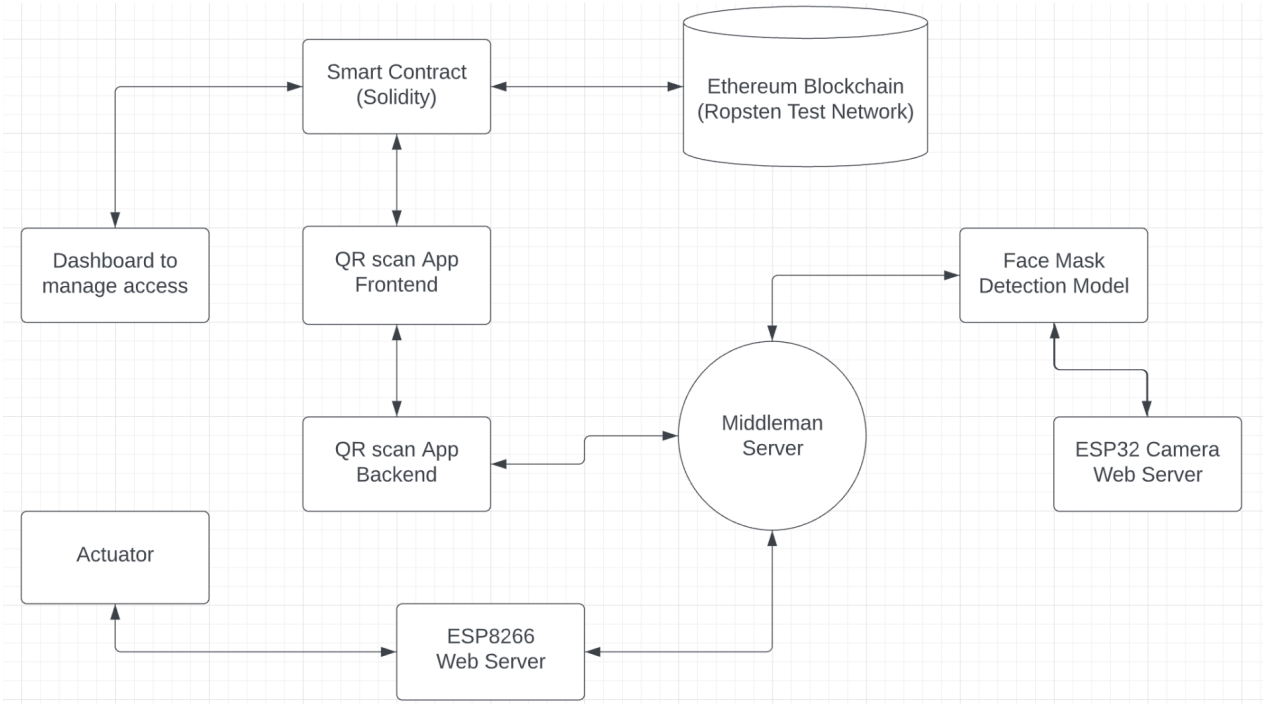


Fig 3.7.1 is the block diagram for our project and show's how every component of the application interacts with one another, the following steps happen when a user scans a QR code to verify their Access to a Room :-

1) The smart contract API is called from frontend of the web application which then performs a check if that "Account" has Access to the Room with ID as RoomID or not.

2) The result of smart contract read operation is then sent to the backend of the web application through web sockets..

3) The backend of the web application receives live access status through web sockets using the socket.io library.

4) The backend then sends the access status to the middleman server via an HTTP request

5) The middleman server then sends the status to the ESP8266 NodeMCU web server which is subscribed to that particular event.

6) The ESP8266 web server will then operate the actuator if access is granted to the user.

7) The middleman server then sends a request to the Flask server on which our model is run, it then reads the image from the livestream of the ESP32 camera web server.

8) A snapshot is taken by the ESP32 camera web server and that is sent to the Flask server..

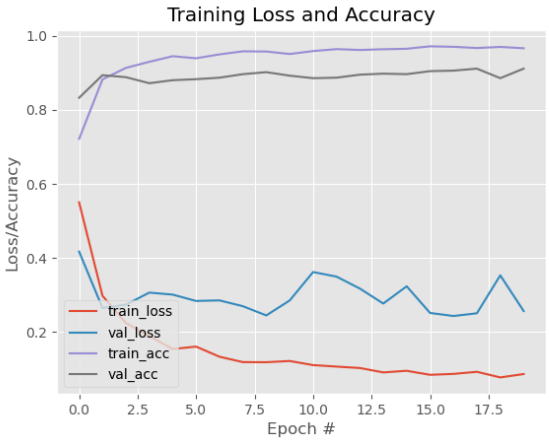
9) The model evaluates and classifies the image and returns the predicted value to the ESP32 camera web server.

10) Once the ESP32 camera web server receives the request, it triggers the actuator accordingly.

4. RESULT

Both the web applications were built and deployed, both the hardware components ESP32 camera web server and the ESP8266 web server were also built and deployed on the local network, once that was done we used Ngrok for port forwarding to expose the local network onto the global network so these servers could be requested from anywhere in the world, the smart contract was tested and deployed prior to all this, once everything was done the system was stress tested.

The facemask detection model was also tested on the test set and then again once deployment on the Flask server was done. The results of the test of the facemask detection model on the test data is shown below in *Fig 4.1*



As shown in *Fig 4.1,* 20 epochs were done and after each epoch the train_loss, val_loss, train_accuracy and val_accuracy were plotted and we observe that after the first 3 epochs, the improvement in the accuracy of the model was stagnated until the 18th epoch after which it increased up to 95%, the average accuracy of the model was 92% and on the images taken by the ESP32 camera web server the average accuracy was 90%.

	Precision	Recall	F1-Score	Support
With mask	0.98	0.83	0.90	384
Without mask	0.85	0.98	0.91	386
Accuracy			0.91	770
Macro avg	0.92	0.90	0.90	770
Weighted avg	0.92	0.91	0.90	770

Fig 4.2

Refer *Fig 4.2* for the evaluation metrics shown for the facemask detection model, the performance metrics that were used for the facemask detection model were Precision, Recall, F1 score and the Accuracy, both Macro average accuracy and the Weighted average accuracy, the average F1 score of the facemask detection model was 0.90 and the average accuracy of the model was 92% as it was shown in *Fig 4.1*, When the accuracy line is steady, it signifies that there is no need for further iteration to improve the model's accuracy.

5. FUTURE SCOPE

There is a lot of scope for improvements in this project that will have to be done before implementing it in industry, from using industry standard sensors and actuators for our hardware component to completely changing the blockchain from Ethereum onto a faster, more scalable blockchain.

Instead of using the Ethereum chain for access verification, in which a write operation takes on average 5-10 seconds and a high gas fees, we can use the Polygon layer which can handle up to 65,000 transactions per second or the Solana chain, which can also handle up to 70,000 transactions per second with minimum gas fees. This can be done to reduce the transaction validation time and reduce the gas fees and hence make the system more scalable, robust and lower the variable costs though this would increase the time to market as a proper infrastructure for working with blockchains like polygon and solana will have to be used which will take time as these are relatively newer blockchains and do not yet have the community and support like Ethereum does.

The middleman server can be scaled up to a full fledged MQTT broker which can interact between different hardware nodes as required in industry applications. This would make the project scalable to handle millions of different hardware components, the QOS models could also be implemented or we could use an MQTT broker service from any cloud service provider like AWS or Microsoft Azure.

A 5v Power source can be used and this would eliminate the need of the Arduino Uno since currently it's only use is to power the esp32cam and hence would reduce our costs, also custom hardware can be manufactured with just the required functions for our project as this would also reduce our cost per unit.

Edge computing can be used for getting the result of our Face Mask Detection model on an image using Arduino Nano BLE 33 Sense or Raspberry Pi Boards. This would eliminate the need of having a separate Flask server on which the model would run and this would make the process faster, though the fixed costs would increase which is a tradeoff that has to be considered before making this change.

The metamask Android and IOS app currently does not have all the features which the metamask chrome extension does, hence developing for mobile is currently difficult and that could be improved by building our own infrastructure to work with.

6. REFERENCES

[1] https://ethereum.org/en/whitepaper/

[2] https://www.ijert.org/face-mask-detection-Using-deep-learning-and-computer-vision

[3] https://www.researchgate.net/publication/323525875_Internet_of_Things_IoT_System_Architecture_and_Technologies_White_Paper

[4] https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3548431

[5] https://www.researchgate.net/publication/354653188_Decentralized_apps_of_the_future_implementing_Web3

[6] https://www.flickr.com/photos/zcopley/8085409641

[7] https://www.researchgate.net/figure/Impact-of-public-mask-wearing-under-the-full-range-of-mask-adherence-and-efficacy_fig1_341347043

[8] https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application

[9] https://en.wikipedia.org/wiki/QR_code

[10] https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[11] https://towardsdatascience.com/deriving-the-backpropagation-equations-from-scratch-part-1-343b300c585a

[12] https://www.researchgate.net/figure/Flow-diagram-of-the-IoT-Blockchain-network_fig4_325430814