Deployment of Facial Recognition Models at the Edge: A Feasibility Study

Xihao Zhou
School of Computing Science
University of Glasgow, UK
2427216Z@student.gla.ac.uk

Sye Loong Keoh
School of Computing Science
University of Glasgow, UK
SyeLoong.Keoh@glasgow.ac.uk

***Abstract*—Model training and inference in Artificial Intelligence (AI) applications are typically performed in the cloud. There is a paradigm shift in moving AI closer to the edge, allowing for IoT devices to perform AI function onboard without incurring network latency. With the exponential increase of edge devices and data generated, capabilities of cloud computing would eventually be limited by the bandwidth and latency of the network. To mitigate the potential risks posed by cloud computing, this paper discusses the feasibility of deploying inference onboard the device where data is being generated. A secure access management system using *MobileNet* facial recognition was implemented and the preliminary results showed that the deployment at the edge outperformed the cloud deployment in terms of overall response speed while maintaining the same recognition accuracy. Thus, management of the automated deployment of inference models at the edge is required.**

***Index Terms*—machine learning, edge computing, artificial intelligence, facial recognition, security access management**

I. Introduction

The number of connected Internet of Things (IoT) devices is anticipated to reach over 50 billion by the end of 2020 [3]. IoT devices range from basic sensor and actuator nodes that log and report data collected for cloud processing, to edge devices with the capability of processing and analysing the collected information. With the advances in modern computing technologies, Artificial Intelligence (AI) has been slowly integrated into these devices, from smart home appliances to autonomous vehicles. AI is one of the de-facto paradigm used to intelligently process raw data collected from those devices to produce prediction outputs without the need to program it explicitly. To assist the process of developing AI applications, one of the most commonly used methods is Machine Learning (ML) algorithms such as deep learning [13].

Traditionally, AI and ML have primarily been performed on servers and high-performance machines with powerful Graphics Processing Units (GPU) [8]. At the moment, cloud computing is the most common solution in providing the necessary computing power, having devices to send the generated data to a remote server for centralised processing. However, cloud computing does come with several flaws, such as dependency on a strong network connectivity and potential security issues to protect the confidentiality and integrity of sensitive data.

The volume of data produced by edge devices had skyrocketed in recent times, with more data generated in the past two years than in the entire human history before that. It is estimated that around 1.7 Megabytes of new data is created every second for every human on the planet. Transmitting a vast amount of raw data to be processed in the cloud can be a bottleneck due to the heavy reliance on the bandwidth of the network. It is extremely resource intensive to stream real time data to the cloud platforms to perform AI and ML functions, thus, network latency is almost inevitable.

This paper proposes a new approach to mitigate the aforementioned issues by moving the computation from the cloud closer to the network's edge, where the data is being generated. This motivates the emergence of a new computing paradigm, edge computing, which improves computation efficiency and reduces overall latency. Moreover, it provides greater control over data generated by the edge devices, offering better privacy and security over cloud computing. With the ability to run inference engine on edge devices, incremental machine learning can be further optimised such that newly acquired data can be sent to the cloud for training in batches, and the re-trained inference model is then securely deployed onto the edge devices. With edge intelligence, devices are granted the ability to take control of the communication management and their own data, thus improving quality and accuracy as well as reducing the possibility of delayed response due to network latency. One of the major advantages of edge intelligence is having the ability to make an instant decision at the source of where the data is created. Using a secure access management system as a use case, we developed a facial recognition system based on *Tensorflow Lite* and conducted extensive experiments on both the edge with a Tensor Processing Unit (TPU) and the cloud, to evaluate their performance in terms of recognition rate, inference speed and network latency.

This paper is organised as follows: Section II presents related work in edge intelligence. Section III describes the proposed management architecture for AI at the edge, while Section IV details the implementation of secure access management using facial recognition at the edge. Section V presents the performance evaluation, and finally we conclude the paper with future works in Section VI.

II. Related Work

One of the major applications of edge computing is the implementation of IoT sensors by the manufacturing industries to monitor and control manufacturing processes and systems. This facilitates the automation of manufacturing processes including mechanical systems. IoT sensors are installed to monitor dust, humidity and temperature within the plant via RFID. This system provides real-time data to identify how each and every part of electronics are assembled and stored and also to monitor the time taken to complete each process [10]. It has helped the company to establish 75% coverage of the plant via the RFID-based monitoring system, saving the company more than 5,000 man-hours per year. However, data collected from sensors are being transmitted and processed in the cloud.

Cloud computing is still the most popular and prominent method for AI applications. Large support for AI and ML platforms provides a flexible and open environment making it easier to collect and load datasets for AI systems directly from the cloud-based data warehouse and data storage services offered by the cloud platform providers. The major challenge of this method is the movement of a large amount of data from the point of generation to a cloud-based data warehouse for the development of AI systems [7]. It is usually more difficult to move this amount of data to a centralized data center due to bandwidth limitations, cost, and latency issues.

AI models can be trained in the cloud and later deployed onto devices to be executed closer to the data being produced [18]. Another approach is to train AI models on the edge device itself on the basis of the data flowing through the device. While both of these methods aid in reducing latency issues, they can give rise to issues concerning accuracy. The development and training of dataset to achieve highly accurate AI models integrated within edge devices can pose challenges due to the fact that they are built for low power and low cost, thus limiting the amount of data that can be analysed onboard.

Several hardware accelerators are available to enable the execution of AI inference engine on edge and mobile devices with limited computation resources. Google Coral [4], Intel Neural Compute Stick [6] and Nvidia Jetson [12] are hardware platform designed to enhance AI performance on the edge. These accelerators contain specialised chips built to perform AI inference, while attached to an edge or mobile device, this can effectively offload the computation needs for AI operations from the device's CPU onto the accelerator.

One prominent example of deploying AI on the edge is the use of biometrics as a form of identification to unlock smartphones. Most if not all smartphones in the market uses some type of biometrics identification such as fingerprint recognition or facial recognition as part of its unlocking mechanism. However, such models are able to run on smartphones as it was designed to recognise and identify a small number of identities, for most cases, the system is only able to store the identity of the phone owner only. Furthermore, smartphones nowadays have relatively strong computational power as compared to simple IoT devices, allowing it to run simple AI applications with ease.

It has been shown that object detection models can be deployed onboard Unmanned Aerial Vehicles (UAV) with an embedded edge device [13]. Several object detection models were used in the experiment, namely, YOLO [14], an advanced computer vision framework that is optimized for object detection and DroNet [9], an efficient real-time detector for UAVs. The models were trained to detect vehicles and pedestrians on the road, key performance indicators includes frames per second (FPS) and accuracy of the model. The embedded edge device used was Odriod XU4 which has a 2GHz Octo core CPU, 2GB of RAM and deployed on a Linux based operating system. Most of the DroNet models used were able to achieve a decent FPS of 4 - 9 and accuracy of over 80%, concluding that AI at the edge is feasible.

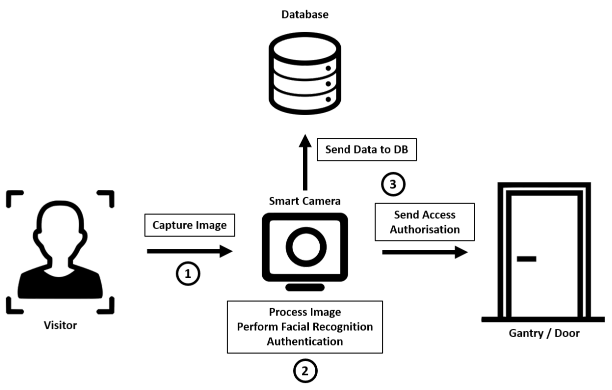

Fig. 1. Facial Recognition at the Edge.

III. Artificial Intelligence at the Edge

Facial recognition is one of the most popular methods for biometric identification as it is fast and relatively easy to implement. It has been deployed in many application areas such as a simple usage of unlocking mobile phones, secure visitor registration, access management and attendance taking in classrooms. In recent years, this technology has been utilised for security purposes like video surveillance in sensitive or restricted areas to detect unauthorised personnel and identification of wanted criminals in the public.

The core technology behind facial recognition systems is AI. There are multiple well established AI algorithms that can accurately detect and identify faces from an image or video. With AI, a model can be trained to recognise faces for identification and authentication purposes. In most scenarios, real time recognition is necessary for efficiency and security reasons. With that, having a facial recognition and identification system on edge devices is highly desirable.

A. Secure Access Management Use Case

Figure 1 shows an example deployment of facial recognition system at the edge to enable secure access management. In this simple deployment architecture, both the IoT camera and the smart door or gantry are connected together in a local network, while there is a cloud database that is used to store metadata of access requests, to facilitate re-training of the facial recognition model. The main edge device in the system is the IoT camera which will perform the key system operations such as data processing and facial recognition.

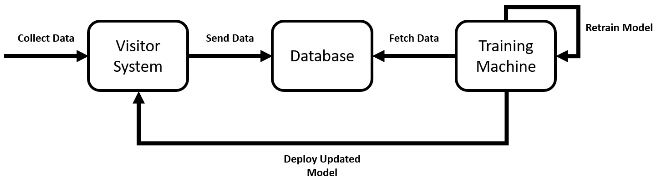

Fig. 2. Continuous Deployment of AI Inference Model on the Edge.

In this application use case, when a user requests to enter a building, photo(s) of the user is taken by the IoT camera and then processed to ensure that the photo is readable by the facial recognition model on the edge device. An authentication decision is made locally by the IoT camera using the inference with the deployed facial recognition model. Access is granted if the user can be recognised and authenticated, and subsequently triggers an action on the smart door or gantry.

B. Continuous Deployment of Inference Model

In the cloud deployment, the inference model is always updated. On the contrary, for edge deployment, the inference model at the edge devices must be managed to ensure that critical security updates and inference ability are updated periodically. The ability to constantly re-train and update the facial recognition model is hence an important consideration in order to build a robust system. Figure 2 shows the process flow the continuous deployment of AI model. When an user (not known to the system) enters the building for the first time, the metadata of the visit will be sent to the database, identifying that it is a new visitor. The machine used to train the facial recognition model will fetch the data of new visitors and use relevant data such as the photos taken and the name of the visitor to retrain the facial recognition model so that the person will be recognised by the system for future visits. After the training is done, the updated model will then be deployed to the IoT camera.

C. Development of Facial Recognition Model at the Edge

As there are many facial recognition models that can be used, it is important to first develop and evaluate facial recognition models in order to identify the most applicable model to be deployed on an edge device. The development of AI model is divided into several steps: *Data Acquisition*, *Data Preparation*, *Model Selection*, *Training*, and *Evaluation*.

1) Data Acquisition: Three datasets were selected to develop the facial recognition model, including two public datasets, *VGG_Face2* [1] and *CASIA-WebFace* [17], as well as a private dataset. Table I shows the metadata of both public datasets, the *VGG_Face2* dataset is one of the largest public datasets available online with a large amount of unique identities and a high number of samples per identity. *CASIA-WebFace* is significantly smaller than *VGG_Face2*, however, it contains more identities with a smaller number of samples per identity. The private dataset consisted of photos obtained from university students. This private dataset was made up

TABLE I
METADATA OF PUBLIC DATASETS

Dataset Name	Number of Identities	Number of Photos	Average Sample Size per Identity	Dataset Size (GB)
VGGFace2	8631	3,124,422	362	36
CASIA-WebFace	10,575	494,414	46	4.9

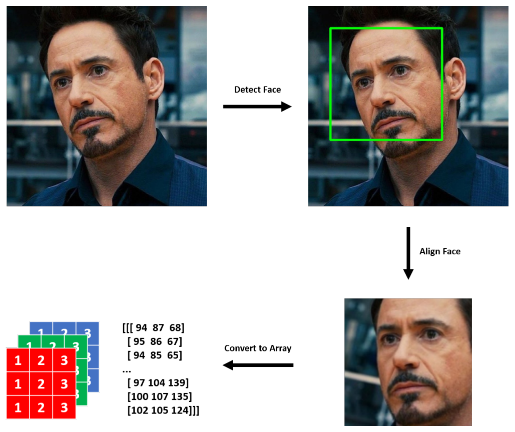

Fig. 3. Data Preparation Steps.

of photo shots with the consent of the subjects and personal photos provided by the subjects.

2) Data Preparation: Figure 3 shows the data preparation process for the development of facial recognition model. The identified faces in the dataset are extracted and then facial alignment is performed. This acts as a form of normalisation of the data, ensuring that they are uniformed. The images are then encoded into a three dimensional array object that can be used to train the AI model. All encode images are then merged into a list of array object and data splitting is performed to divide the data into two unique sets, one is used for model training and the other for model testing or validation.

TABLE II
BASE MODEL INFORMATION

Base Model Name	Number of Layers	Number of Parameters
SimpleNet	6	134,528
InceptionResNetV2	572	55,873,736
InceptionV3	159	23,851,784
Xception	126	22,910,480
MobileNet	88	4,253,864
MobileNetV2	88	3,538,984

3) Model Selection: Six base models were put to test in order to evaluate their accuracy and efficiency. Table II shows some basic information of the base models such as number of layers and parameters, amongst the six models, five are pre-existing model proven to be efficient in object recognition and image classification tasks. Of which, there are highly dense models namely, *InceptionResNetV2* [15], *InceptionV3* [16] and *Xception* [2]. *MobileNet* and *MobileNetV2* [5] are optimised models built to ensure smooth AI operations when deployed on the edge and mobile devices. Finally, a custom model called *SimpleNet*, which contains a simple convolutional and max pooling layer were used as part of the selection as illustrated in Table III.

TABLE III
BASE MODEL STRUCTURE OF SIMPLENET

Layer	Input Shape	Output Shape	Parameters
Conv2D_1	160 x 160 x 3	160 x 160 x 256	3328
MaxPooling2D_1	160 x 160 x 256	80 x 80 x 256	0
Dropout_1	80 x 80 x 256	80 x 80 x 256	0
Conv2D_2	80 x 80 x 256	80 x 80 x 128	131200
MaxPooling2D_2	80 x 80 x 128	40 x 40 x 128	0
Dropout_2	40 x 40 x 128	40 x 40 x 128	0

4) Model Training and Hyper Parameter Optimisation: During the model training phase, parameters such as loss and accuracy are used to monitor the performance of the model. The loss of a model indicates the number of errors made during the training, thus, the lower the loss the better the model. On the other hand, the accuracy during training determines the number of correct prediction the model made using the training dataset, percentage values are produced, and a higher accuracy is desirable. The loss and accuracy values are co-related to each other, a lower loss means a higher accuracy, and vice versa.

Hyper-parameters such as the *batch size* and *epoch* are used to fine-tune the model. It is important to tune the training epoch, which is the number of times the training dataset iterates through the training model. Running too little *epochs* will produce a model with poor performance, while running too many *epochs* might result in over-fitting when the model starts to learn from statistical noise within the dataset. Batch size is the number of samples being passed to the training model during each iteration of a training run. Having a too small or too large batch size will both affect the training effectiveness.

IV. USE CASE IMPLEMENTATION

A. Implementation Framework and Hardware

TensorFlow was used as the main framework to implement the facial recognition model due to its ability to be deployed on a variety of platforms, especially its support for edge devices and AI accelerators with TensorFlow Lite. Moreover, TensorFlow also has native integration with Keras API, which is very useful for building deep learning models. Other than TensorFlow, Scikit Learn was also used for the Triplet Loss models.

As for the edge device, the secure access management system was built using a Raspberry Pi B+ device, with a camera attached. *Coral Accelerator* which has TPU was chosen for its native support for TensorFlow and ease of model conversion and deployment. This setup was used for the facial recognition model evaluation and the deployment of the secure access management system.

B. Model Training

While performing the training of facial recognition models, the optimal batch sizes selected were 128, 64 and 16 for the *VGG_Face2*, *CASIA-WebFace* and the private set we collected respectively. Tensorboard [11] was used to monitor and visualise the number of epochs affecting the training results. It is observed that 50 *epochs* is the optimal number that will not cause overfitting and produced the best models.

Early stopping was another fail-safe mechanism that was put in place to avoid overtraining of the model and reduced overall training time. The TensorFlow Keras library provides a built in early stopping function that can be used, it uses the loss value to monitor the training progress. Whenever the loss value stops getting better for 10 consecutive *epochs*, the training will be stopped and the *epoch* with the best weights will be saved.

C. Model Conversion for Edge Deployment

Model conversion was needed so that they can be deployed onto the Raspberry Pi and the Coral Accelerator. Two conversions had to be performed, first the regular TensorFlow model was converted to a TensorFlow Lite model which can be used on the Raspberry Pi. This was executed using the Python TensorFlow Lite library. Next, the regular TensorFlow was converted to a quantised TensorFlow Lite model where quantisation is a process of converting 32-bit floating-point numbers to the nearest 8-bit fixed-point numbers. This was done so that the quantised TensorFlow Lite model can be converted to a Edge TPU model which the Coral Accelerator can read. The conversion to Edge TPU model was performed using the Edge TPU Compiler tool by Google.

After all the conversions, three new models were created for the deployment of facial recognition system on the cloud (*regular TensorFlow model*), Raspberry Pi (*TensorFlow Lite model*), and Raspberry Pi with TPU (*Edge TPU model*). All models were able to go through the conversion process other than models created using SimpleNet.

V. EVALUATION AND RESULTS

This section presents the evaluation of the base models with *Triplet Loss* on different platforms in terms of model accuracy, inference speed, and size of the model.

A. Model Accuracy and Inference Speed

Figure 4 illustrates the results of accuracy and inference speed of *regular Tensorflow*, *Tensorflow Lite* and *Edge TPU* when they are deployed on a laptop vs. Raspberry Pi. It is observed that most models were able to achieve an accuracy of over 80%.Accuracy of the models were maintained after converting to the TensorFlow Lite format. As for the Edge TPU, there was a slight decrease in accuracy due to the quantisation process of the models when converting to the Edge TPU format. With Triplet Loss, most models were not affected much with a slight decrease in accuracy of around 1 – 3%, with the exception of *Xception* where a 22% decrease in accuracy was observed. *InceptionV3, MobileNet*,

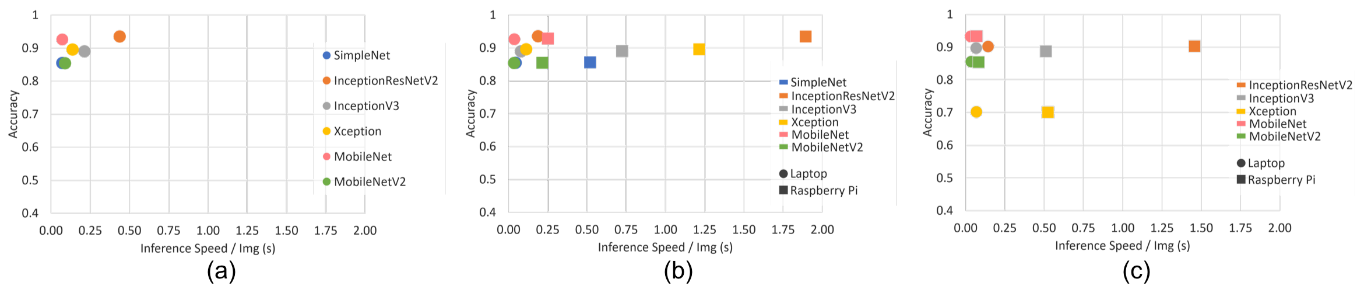

Fig. 4. Tensorflow with Triplet Loss: (a) Regular Tensorflow (b) Tensorflow Lite (c) Edge TPU

TABLE IV
SUMMARY OF EACH MODEL'S INFERENCE SPEED, ACCURACY AND SIZE WHEN DEPLOYED ON THE RASPBERRY PI

	TF Lite			**Edge TPU**		
	Speed (s)	**Accuracy**	**Size (MB)**	**Speed (s)**	**Accuracy**	**Size (MB)**
SimpleNet	00.486	0.73	69.20	-	-	-
InceptionResNetV2	01.807	0.91	207.00	01.433	0.72	54.3
InceptionV3	00.711	0.88	83.70	00.491	0.88	22
Xception	01.188	0.91	80.00	00.504	0.72	22.2
MobileNet	00.233	0.93	12.50	00.051	0.94	3.5
MobileNetV2	00.190	0.66	8.80	00.059	0.65	2.9

and *MobileNetV2* models are the most consistent in terms of accuracy for all three formats. There was no change in the accuracy when models were tested on the Raspberry Pi.

In terms of inference speed, it is evident that the *Triplet Loss* are slower than the *Softmax*, this is due fact that *Triplet Loss* models require one extra round of inference with the SVM models. Even so, the difference in speed were relatively minor for TensorFlow Lite and Edge TPU. Amongst the three formats, it was clear that the Edge TPU models are the fastest with an average of 32% improvement in speed when compared to Tensorflow Lite and 45% when compared to regular TensorFlow. At the same time, a noticeable distinction between the speed of regular models and edge optimised models like *MobileNet* and *MobileNetV2* can be seen. Since the edge optimised models were designed to operate efficiently on CPUs with lesser computing power, inference speed were relatively faster as well. As expected, inference speed of models when running on the Raspberry Pi is much slower than running on Laptop, this is particularly true for the TensorFlow Lite models. However, it can be observed that the decrease in speed was more apparent for models with more layers. Edge optimised models were not affected as much, especially for the Edge TPU, since most of the inference functions were offloaded to the Coral Accelerator.

B. Model Size

Table IV shows the sizes of the various models on *Tensorflow Lite* and *Edge TPU*. The initial size of the regular Tensorflow was relatively large, especially for the models not optimised for the edge devices. After conversion to TensorFlow Lite, the size decreased by an average of 67% across the board. The most significant decrease can be seen when the base models were converted to run on Edge TPU with an average of 74% reduction in size from the Tensorflow Lite. Edge optimised models, e.g., *MobileNet* and *MobileNetV2* can be reduced to less than 5 MBs, making them highly feasible for edge deployment.

C. Deployment for Secure Access Management

The evaluation results have revealed that *MobileNet* is the best model to be used as it outperformed others in terms of inference speed and accuracy on the Raspberry Pi with a Coral Edge Accelerator. It achieved 94% recognition accuracy and recorded the fastest inference speed at only 0.051 seconds per image when running on the Edge TPU. In terms of size, only *MobileNetV2* was slightly smaller, even though its speed was comparable with *MobileNet*, but its accuracy was the worst amongst all.

We have deployed a secure access management system using *MobileNet* on the cloud using the regular *Tensorflow*, as well as on a Raspberry Pi with Coral Accelerator using the *Edge TPU*. Further system evaluations were conducted to compare the performance of real-time facial recognition between cloud and the edge. In this case, the private dataset collected was used, in which ten test subjects were part of the training data, while another two subjects were unknown. Each participant was tasked to use the system ten times in a random order. This process was carried out for both the edge and cloud deployment.

1) System Accuracy: Table V shows the average results for the 120 tests performed on each deployment method. It is observed that there was a huge drop in the accuracy of the models when real time facial recognition was performed, from the initial 93% on the static dataset to around 64%. The decrease in the accuracy was most likely due to two main factors: quality of the photos taken by the Raspberry Pi camera

TABLE V
EVALUATION RESULTS FOR BOTH DEPLOYMENT METHODS

	Accuracy	**Inference Speed (s)**	**System Response Speed (s)**
Cloud Deployment	0.62	0.418	2.891
Edge Deployment	0.64	0.036	0.036

and the lighting conditions. Most of the images in the training dataset were taken using a mobile phone with a high resolution camera. There was no major difference in terms of accuracy on both cloud and edge.

2) Inference Speed: On the other hand, a significant difference in inference speed can be observed between cloud and edge. The deployment on the edge was around 70 times faster than deployment on the cloud. This is attributed to two main reasons: namely, boost in speed due to the Coral Accelerator and network latency. As can be seen in Table IV, the inference speed on Edge TPU is significantly faster than the regular Tensorflow by 55% since most of the AI operations were offloaded to the Coral Accelerator. Network latency seems to have affected the overall response speed for the cloud deployment. The actual inference speed of the facial recognition model on the cloud server was 0.418s which was already much slower than the inference speed on the RaspberryPi. With The actual response speed was 2.891s by the cloud, this means that approximately 2s were spent on network communication.

VI. CONCLUSIONS

This paper has shown that it is feasible to deploy AI applications and systems onto edge devices, to mitigate concerns of using cloud platform that mainly revolves around bandwidth limitations and network latency. Edge computing has emerged recently to tackle these issues, by moving processing of data to where it is being generated. Our work has taken a step further by having AI applications deployed solely on edge devices to create a truly intelligent edge.

After an extensive evaluation of different facial recognition systems and the ability to convert them for edge deployment, we have also shown that a security access management system using facial recognition as authentication, executed at the edge was able to function efficiently and make predictions accurately on edge platforms.

In order to ensure robustness of the systems with AI at the edge, the process of continuous deployment for facial recognition model must be automated. Currently, the process involves multiple elements that require human intervention, such as running the program to retrain the model with data of new visitors and deploying the updated model onto the edge device. Automation and managing continuous deployment of AI inference engine must be implemented, to ensure the security and integrity of the inference models deployed.

Finally, as there are six levels of edge intelligence as conceptualized by [19]. The proposed study has only managed to achieve level 3 where AI inference is performed on the edge, with the model training done in the cloud. In the future, it is expected that model training will be feasible on the edge, so that it can be truly independent from the cloud. This will be very challenging due to the limited computation and storage capabilities of edge devices, nonetheless there are already ongoing research to make this possible by having distributed and federated learning across multiple edge devices.

REFERENCES

[1] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 67–74. IEEE, 2018.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[3] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. 2011.

[4] Google. Google coral. https://coral.ai/.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[6] Intel. Intel neural compute stick. https://software.intel.com/en-us/neural-compute-stick.

[7] Fatemeh Jalali, Olivia J Smith, Timothy Lynar, and Frank Suits. Cognitive iot gateways: automatic task sharing and switching between cloud and edge/fog computing. In *Proceedings of the SIGCOMM Posters and Demos*, pages 121–123. 2017.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[9] Christos Kyrkou, George Plastiras, Theocharis Theocharides, Stylianos I Venieris, and Christos-Savvas Bouganis. Dronet: Efficient convolutional neural network detector for real-time uav applications. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 967–972. IEEE, 2018.

[10] Sastry KM Malladi, Thirumalai Muppur Ravi, Mohan Komalla Reddy, and Kamesh Raghavendra. Edge computing platform, 2019. US Patent 10,379,842.

[11] D Mané et al. Tensorboard: Tensorflow's visualization toolkit, 2015.

[12] Nvidia. Nvidia jetson. https://www.nvidia.com/en-sg/autonomous-machines/.

[13] George Plastiras, Maria Terzi, Christos Kyrkou, and Theocharis Theocharidcs. Edge intelligence: Challenges and opportunities of near-sensor machine learning applications. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–7. IEEE, 2018.

[14] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[15] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[17] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[18] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er, and Mérouane Debbah. Big data caching for networking: Moving from cloud to edge. *IEEE Communications Magazine*, 54(9):36–42, 2016.

[19] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.