8-2023

Deep Learning For Resource Constraint Devices

Sheikh Rufsan Reza

DEEP LEARNING FOR RESOURCE CONSTRAINT DEVICES

A Dissertation

by

SHEIKH RUFSAN REZA

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2023

Major Subject: Electrical Engineering

DEEP LEARNING FOR RESOURCE CONSTRAINT DEVICES

A Dissertation

by

SHEIKH RUFSAN REZA

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Xishuang Dong Committee Chair	John Fuller Committee Member
Xiangfang Li Committee Member	Lijun Qian Committee Member
Lin Li Committee Member	Annamalai Annamalai Interim ECE Department Head
Pamela Obiomon Dean, Roy G. Perry College of Engineering	Tyrone Tanner Dean, Graduate Studies

August 2023

Major Subject: Electrical Engineering

ABSTRACT

Deep Learning for Resource Constraint Devices

(August 2023)

Sheikh Rufsan Reza, PhD EE., Prairie View A & M University;

Chair of Advisory Committee: Dr. Xishuang Dong

The amount of Internet-of-things (IoT) devices is rapidly expanding. This also triggered the necessity of smart IoT devices which are capable of conducting any task by itself. Deep learning techniques are also booming due to the increased computing power and refined algorithms. The advantage of deep learning is that it can be tuned into any application without the manual feature extraction process. Now, the combination of deep learning with smart IoT devices/edge devices can result in any application that can be used in machine vision, vision inspection, autonomous vehicle, and many more. These applications can be automated which requires human operation. Now, combining deep learning and edge device together and running the application can be a difficult task. The main reason is that deep learning requires large computation power and edge devices does not have that capability. This study focused on this problem. Ie used techniques to encrypt and compress data which is essential for the edge devices. In addition, we developed novel methods to protect user privacy for data collection and learning on edge devices. Also, we conducted a study to evaluate different edge devices for different application purposes with certain compression technique of the models. Lastly, we conducted a real life experiment of collecting data, creating different models and evaluating it on different edge devices.

index terms - IoT, computer vision, deep learning, machine learning, quantization, autoencoder, mobilenet v1, mobilenet v2, inception v3, face mask detection

DEDICATION

To the GOD ALMIGHTY, my saviour and redeemer.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my advisor, Dr. Xishuang Dong, for his utmost support, tutelage, guidance and patience in the course of my research. His experience, knowledge, advice, encouragement, and inspiration have had a great impact on me.

I would also like to thank Dr. Xiangfang Li, Dr. John Fuller, Dr. Lijun Qian, Dr. Lin Li, who are members of my advisory committee for their support during the entire program. I also like to thank Dr. Richard Wilkins, the graduate coordinator, Dr. Annamalai Annamalai, the ECE department head, and Dr. Pamela Obiomon, the Dean of the College of Engineering.

This research work is supported by the United States Office of the Under Secretary of Defense for Research and Engineering (OUSD(RE)) under agreement number FA8750- 15-2-0119. I am also grateful to Electrical and Computer Engineering Department and Prairie View AM University.

Lastly, I am deeply grateful to my family for supporting me during this journey.

NOMENCLATURE

AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
API	Application Program Interface
FN	False Negative
FP	False Positive
TN	True Negative
TP	True Positive
MLP	Multi-Layer Perceptron
DL	Deep Learning
DNN	Deep Neural Network
KNN	K Nearest Neighbor
LSTM	Long Short Term Memory
OpenCV	Open Source Computer Vision Library
WLAN	Wireless Local Area Network
IoT	Internet of Things
ReLU	Rectified Linear Unit
TPU	Tensor Processing Unit
WLAN	Wireless Local Area Network
SSD	Singel Shot Multibox Detection
URL	Uniform Resource Locator

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

[1]

The expansion of Internet of things (IoT) devices has resulted in producing a huge amount data. IoT refers to devices which can establish an internet connection and can interact with one other [4]. The internet has revolutionized the entire communication system. In almost every aspect of life it has become essential. Some popular mentions are education [5], workplace, industry, healthcare [6] and many more. The popularity and desideratum of the internet has paved the widespread use of IoT devices. These devices can be found from industrial equipment to home appliances. For example, any IoT industrial equipment is more preferred than a regular equipment. The IoT device can be controlled and observed remotely where the regular device does not have that capability. Home appliances are also becoming IoT compatible for the similar reason. It is expected that the number of IoT devices will increase significantly with time. These devices are producing a lot data. These data are mostly in text format and some other formats are images and videos. It is crucial and can be used to learn predictive behavior. It makes any product more user friendly, easy and fast to use. Artificial Intelligence, namely machine learning is playing a huge role in procurement of this data. Also there has been a surge to make the IoT devices smarter with incorporation with artificial intelligence. One fundamental function of AI is that it can generate insights that can shape any application to user specification. This is making applications efficient, user-friendly and cost-effective. Moreover it is expanding in many sectors such as education, healthcare, city, home, energy, banking,

[1] This thesis follows the style of IEEE.

industry and many more. In details in the banking sector AI is helping to detect fraud and credit scoring [7]. Smart city is being designed with efficient energy usage [8]. AI is helping to get insight of disease, selecting more accurate treatment and medicine for individual patients. AI is also making the manufacturing industry more accurate and smart. Process automation and robotics is where AI is helping to improve the productivity significantly. The trend is now to make the IoT devices more smart and faster using AI. However in order to make the devices smarter, they must learn the predictive behavior from data. So, high computational resource is essential to procure this huge amount of data. Even after creating the AI model, it is not easy to deploy those on IoT devices because in most cases the devices does not have that sufficient resources to execute. Now the model on edge devices must be lighter to support the configuration of the device. Some trending advanced research that are helping AI models to be lighter are pruning, quantization, compression, knowledge transfer and many more.

1.1 Deep Learning

Deep learning is the most advanced version of machine learning. It is considered as a sub-field of machine learning [9]. Now machine learning is the idea to automate a system or predict the future. Machine learning is also a subsection of AI. AI is the science which focuses on building machines or systems that do task that requires human intelligence. In simple words AI is the study of building a machine that can learn a task by itself. Then it can provide solution based on its learning. So, it can learn better if the amount of data is huge. It also provides a complete solution where the learning is not required to tuned manually. Fig. 1.1 shows the relationship between fields. One main difference is that deep learning can perform better on large datasets. Also, machine learning requires manual feature extraction whereas deep learning does not. The aim is to achieve high level semantic information from huge

complex data. This method is very convenient as it is mostly automated. It makes easier to extract information from huge dataset.

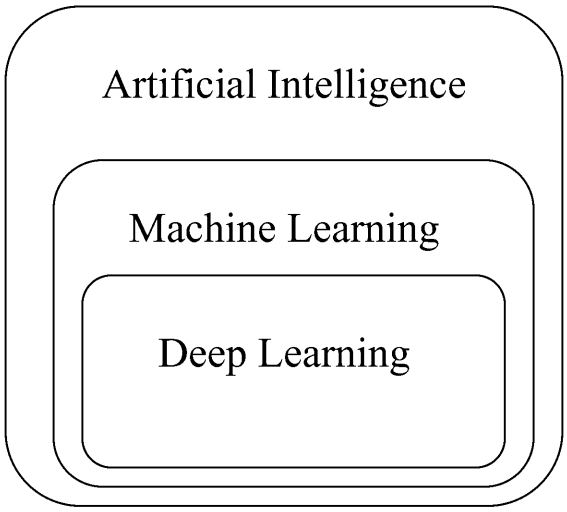


Fig. 1.1. Relationship between artificial intelligence, machine learning and deep learning. Machine learning is a subfield of artificial intelligence. Deep learning is a subfield of machine learning

1.1.1 Deep Learning Basics

Deep learning learns the pattern or function of a task with huge amounts of data which comes in different forms such as images, videos and texts. It uses the computational method to learn a function or model from the data. This method is different from the traditional programming. Traditional programming requires data and methods to get the desired output. Machine learning uses data and output to create the method as shown in Fig. 1.2. Predictive analysis such as classification, clustering and forecast are some of the popular models. Deep learning introduces multi-layer concepts to learn from its data. These layers are the building blocks of a neural network. The function of this layer is to take an input and then learn patterns

by using non-linear functions. Finally, it passes the output to the next layer. The series of layers are constructed using this technique. Usually, regular models have two to three layers but deep learning models can have as many as but not limited to 150 layers. These deep layers can solve very complex problem and require large data. It is capable of directly extracting features without human interaction, making it more efficient than machine learning. Moreover it requires a high graphics processing unit (GPU) to train the model.

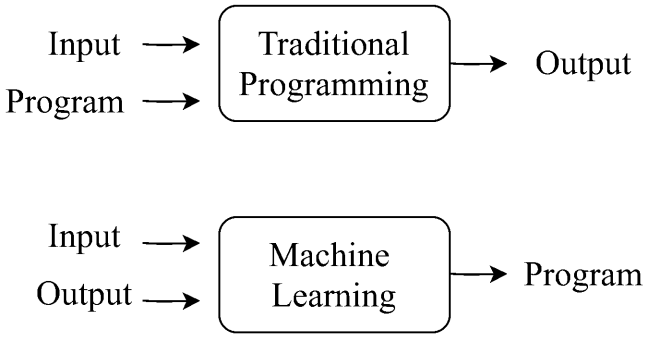


Fig. 1.2. Difference between traditional programming and machine learning. Machine learning uses data and output to create a model whereas traditional programming uses data and model to create output.

1.1.2 Workflow

Deep learning follows a standard workflow. Consider a problem where the objective is to identify two objects such as cats and dogs, then data in the form of images of the two classes is required. Even distribution of both classes ensures a more stronger and accurate model. Data collection is an integral part and can be obtained in many ways. The vast use of IoT devices has made it easier to produce huge quantities of data. Data preprocessing is required for clean and enhanced data. Cleaning or formatting is performed at this stage to eliminate noise. The size of

individual images needs to be same before feeding them into the model. Afterwards, the model is trained and different hyperparameters are tuned to get the maximum desired results. Backpropagation is used in a feed forward the network in order to learn. After training, a separate test set which is not used during training is used to examine the performance of the model. It is illustrated in Fig. 1.3.

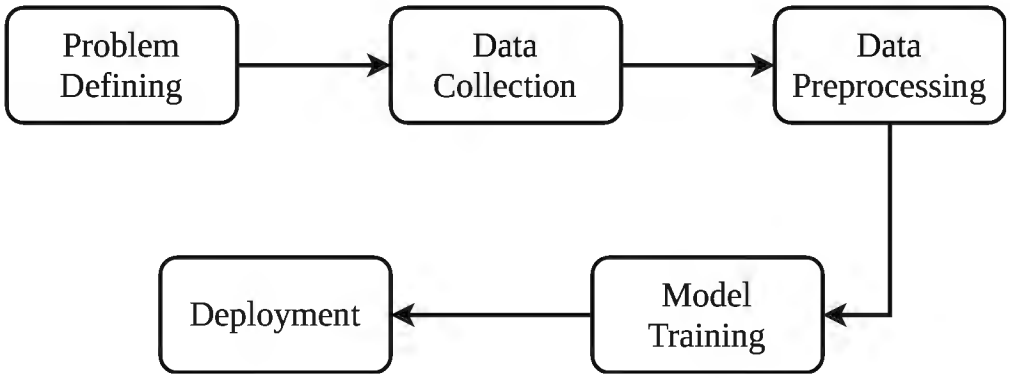


Fig. 1.3. The flow of building machine learning models.The first step is to figure out what sort of problem to solve. Then collecting data and preprocessing. The next step is to train the model and finally testing the model on a separate data.

1.2 Computer Vision

Computer vision is another subset of artificial intelligence which is associated with the study of high-level understanding of digital images or videos. Some of these applications are class recognition, object detection, autonomous driving, object tracking, semantic representation and many more. It goes in the category of supervised learning. The model learns how to predict each class from its labeled dataset. It is mostly used for predictive task. Another category is the unsupervised learning. It does not require labeled dataset and is used to learn pattern or structure.

1.2.1 Convolutional Neural Network

In this work we focus on image classification task. It is the idea of extracting high-level information about different classes from an image. Convolutional neural network [10] is the most common approach for this sort of learning. The most common building blocks are: convolution layer, pooling layer and fully connected layer. Feature maps are used in the convolution layer to extract relevant information of an image. The convolution operation is mainly the matrix multiplication of the image with the feature vector. Then pooling layer is used the reduce the size of the vector. The fully connected layer makes the matrix flatten and prepare for the final prediction. It is illustrated in Fig. 1.4. Moreover, if we consider X as data and K as the filter and i,j are the dimensions, then convolution operation can be expressed by S as shown in equation 1.1.

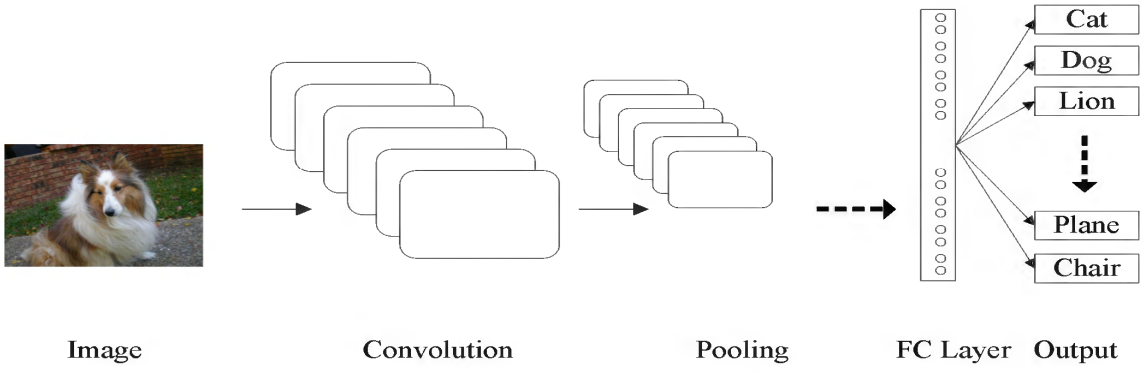


Fig. 1.4. The architecture of convolutional neural network. The main blocks are the convolution layer and pooling layer. A series of this is followed by a fully connected layer which makes the model ready for final prediction.

$$S(i,j) = (K * X)(i,j) = \sum_m \sum_n X(i-m, j-n)K(m,n) \tag{1.1}$$

1.2.2 Existing Model and Dataset

MNIST [11] is the first dataset having hand-written grey color images for classification. It has a total of ten classes representing zero to nine numbers. Then Cifar10 is another popular image dataset having ten classes of color images containing 10 different objects. The most popular image dataset is ImageNet which consists of 1.2 million images and over 1000 classes. The most commonly used pre-trained models such as AlexNet, VGG-16, ResNet, Inception, MobileNet are all trained on this dataset. Using a pre-trained model is a popular technique as this is working on top of a popular trained model which can perform certain tasks. The idea is simple, it implements the learning from one task into another. This is extremely useful in real life problems as it requires very less data. Also, the time becomes very less. For example, a pre-trained model that can classify between cats and dogs can be used to generalize different breeds of cats and dogs. This saves time and makes the training process much faster. All the pre-trained models have their own uniqueness. For example, AlexNet and VGG-16 were created to see depthness effects the performance of a DL model. The model architecture of MobileNet was created for a resource constraint embedded device. ResNet introduced residual learning which addresses the vanishing gradient problem by adding shortcuts to a model while adding more layers.

1.3 Machine Learning on Edge Devices

The main objective of machine learning on edge devices is to make smarter IoT devices. In recent times we do not only want smart IoT to do its application, but also to make smart decisions. For example, in most workplaces, CC tv cameras are used to monitor the security. A smart IoT camera can actually detect an outsider or visitor. Every company has an employee database. An object detection classifier trained on employee databases can detect each employee entering the facility. Usually these

applications are run on a workstation. If this model can be executed on the smart IoT device, then it works as a real time alarming system. It will notify the authorities if someone from outside enters the facility. Moreover this research is associated with both hardware and the model. The most common approach is making the model lighter for edge devices.

1.3.1 Edge Devices

Machine learning on edge is becoming popular as more and more different types of embedded devices are being developed. Some of these devices have small lightweight GPU with the ability to process data at a higher speed. For example, NVIDIA has four Jetson embedded devices:TX1, TX2, Nano and Xavier specially for AI computing. These are the most popular models used for edge deployment. NVIDIA Jetson TX2 can be mounted on top of a drone which can track an object in real time. All the Jetson models have their own specific design and memory which can be used for different case studies. Also, Google introduced Coral Edge TPU which has specialized ASIC for AI purposes. Intel has introduced Field programmable Gate Array (FPGA) which aims to replace GPUs for faster execution.

1.3.2 Existing Research

Traditional training of machine learning model is created using high computing resources such as GPU. For example, an image classifier can classify two objects such as cats and dogs. After training it can be used for deployment. A new image or a set of images not used during training is passed through the classifier and the classifier yields whether it is cat or dog. This part is called testing. Normally in most cases even testing is not possible on edge devices or it can perform at a slow speed. In order to execute these in real time, the model must be made lightweight. Different techniques such as pruning, quantization, knowledge transfer, and knowledge distillation are

some of the popular methods. The common tradeoff is between speed and accuracy. If a model is made lighter, it losses its accuracy. Now the main focus of this is to make the model lighter without losing much accuracy. Moreover, MobileNet models from Google are designed to make it most suitable for embedded devices.

1.4 Problem Statement and Contribution

In recent time, the demand to execute deep learning (DL) models on edge devices is booming as it preserves privacy and safety from any sort of adversarial attack during data transmission. However, deep learning is not usually compatible with resource-constrained devices. The requirement of high computation power makes it difficult to execute them on edge devices. In addition, DL models are too complex to be deployed on edge devices directly. Therefore, how to make the DL models faster and lighter is the core problem to this application. For example, how to make the model lighter while preserving most of its effectiveness is a key problem. Moreover, how these lighter models perform on different edge devices is another key problem. Finally, how to apply this technique to build real applications, such as face mask detection in public using IoT and DL, is an interesting problem. Main contributions are summarized below:

- Quantize three DL models to three different levels of compression and execute those models in various edge devices.
- Compare their performance for all cases which can serve as a benchmark to identify model, compression level and edge devices for different applications.
- Apply model compression techniques to build face mask detector on edge devices.
- Investigate privacy preserving techniques for deep learning models on edge devices.

1.5 Outline of the Dissertation

The rest of the study is outlines as follows. Chapter 2 presents a novel approach on privacy preserving edge computing framework for image classification. Inference performance of CNNs on various edge devices are compared in Chapter 3 and a benchmark to identify model, compression level and edge devices for different applications is provided. Chapter 4 presents an important application of edge intelligent DL for face mask detection on IoT. Chapter 5 concludes the study and presents future works.

CHAPTER 2

EFFICIENT PRIVACY PRESERVING EDGE COMPUTING FRAMEWORK FOR IMAGE CLASSIFICATION

2.1 Introduction

Emerging technologies such as the Internet of Things (IoT) and 5G networks will add a huge number of devices and new services, and as a result, a huge amount of data will be generated in real time. One of the important data types is image data, since many applications involve images and videos such as in video surveillance. In order to take advantage of the "big image data", data analytics must be performed to extract knowledge from the data. One way to handle the data would be uploading all data from edge devices to the cloud or remote data centers for processing and knowledge extraction [12]. However, as highlighted in Fig. 2.1, there are several factors that may render this practice infeasible: 1) the sheer volume of the images may overwhelm the uplink with limited bandwidth; 2) the uplink may not be always available especially when wireless communications is used due to weather (for example., for mmWave), distance, or jamming; 3) proprietary images may need encryption that introduce additional delay and 4) the end users may have concerns about the security and privacy of their images, thus they may not agree to upload raw images that may contain private information. Furthermore, uploading is subject to eavesdropping, interceptions, or other unauthorized access.

In order to address these challenges, a novel efficient privacy preserving framework for image classification in edge computing systems is proposed in this study.

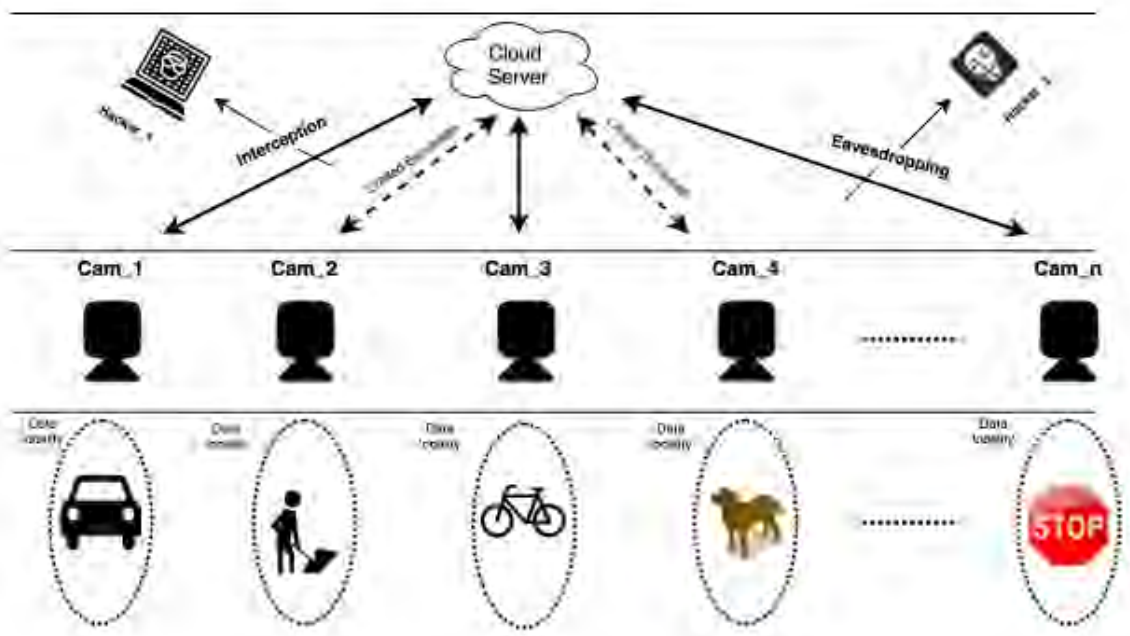


Fig. 2.1. Challenges incurred when uploading all data from edge devices to the cloud.

Specifically, the large raw data was processed locally at the edge by a pre-trained autoencoder. Then, instead of uploading the raw image, only compressed latent vector that contained critical features learned from the raw image were uploaded through the access point or hub to the cloud for further processing. The framework is highlighted in Fig. 2.2. It demonstrated that the learning performance of extracting knowledge at the cloud had very little degradation when the compression ratio was not large (for example, below 16 in the test cases). Furthermore, the raw images could be reconstructed with very small error at the cloud using the pre-trained decoder if needed.

Compared to traditional source coding like zip, using autoencoder has the following advantages: 1) instead of only reducing the redundancy in the raw data as in source coding or traditional data compression, autoencoder will extract critical features in the raw data and encode the features in a compact form, the latent vector. In

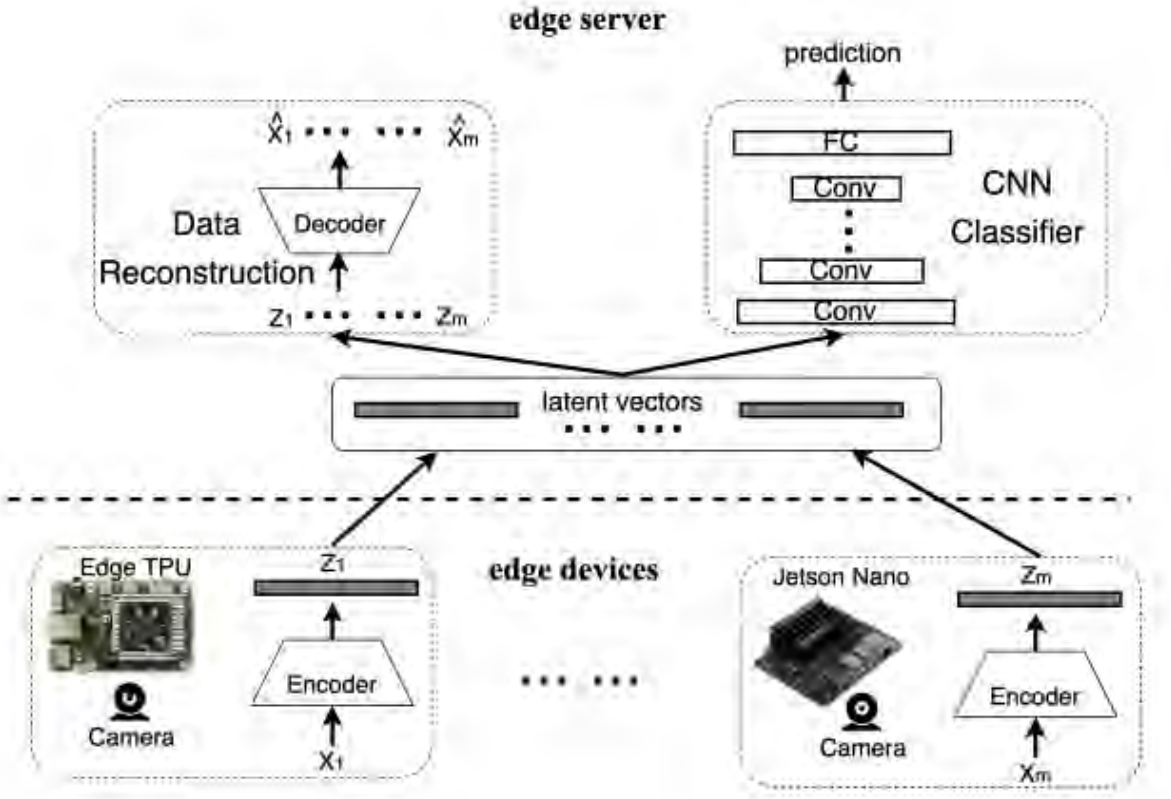


Fig. 2.2. The proposed efficient privacy preserving framework for image classification in edge computing systems. Here x_i is the raw image, z_i is the compressed latent vector, and x_i is the reconstructed image.

other words, the encoderperforms initial learning at the edge devices; 2) in addition to compressing the data, autoencoder also "encrypts" the data by transforming the raw data into latent vectors, which enhance the security of data. For example, a zipped file can easily be unzipped by an adversary if not encrypted. On the contrary, an adversary could not reconstruct the raw data from the latent vector without knowing the structure (that is, number of layers, number of nodes in each layer) and all the weights of the pre-trained autoencoder (the decoder part to be exact). It is shown in [13] that autoencoder provides a similar level of security to normal encryption - assuming that the decoder is not shared; (3) even if the edge device is captured by an adversary, it is very difficult for the adversary to deduce the decoder part from the encoder part on the edge device. The proposed framework has some similar characteristics such as taking advantage of large diverse data from many edge devices and data locality at each device as in federated learning [14]-[15]. However, compared to federated learning, the proposed framework had the following advantages: 1) in federated learning, the server and the end users (edge devices) train the same model. As a result, the complexity of the model is constrained by the computing capability and storage of the edge device. On the contrary, in the proposed framework, the training of the classifier was done at the cloud server only, thus it could be very deep and complex if needed, and it was not subject to the constraints of the edge devices. 2) in federated learning, the edge device must rely on the server to update the gradients and train the model. In the proposed framework, the training of the autoencoder can be done independently at each edge device without any server involvement, 3) in federated learning, the privacy of the end users' data is protected by applying differential privacy schemes [16] or through secure aggregation [17], thus introduce additional cost due to encryption or secret sharing. In the proposed framework, the privacy of the end users' data is protected by transmitting latent vectors without

additional cost of encryption.

2.2 Proposed Framework

The proposed efficient privacy preserving framework for image classification in edge computing systems is shown in Fig. 2.2. It has two levels: the edge devices and the edge server. It is assumed that the nodes of the edge devices contain sensors such as cameras and embedded computing devices such as Google edge TPU [18] or NVIDIA Jetson Nano [19]. The edge server is assumed to have strong computational capacity and large storage. The edge segment of the framework mainly contains the various edge devices of interest and the pre-trained encoder. The server mainly contains the hub, the pre-trained classifier and the pre-trained decoder. We only considered supervised learning in this study and it was assumed that the training dataset was labeled. The data from each of the edge devices were passed to the corresponding encoder attached to it. Unique pre-trained encoder is used for each of the edge devices in order to take advantage of the data locality at each device. The function of the pre-trained encoder, which is in the inference mode, is to extract the most important and critical features in the data. The encoder also ensures dimension reduction of the input data by a pre-determined amount. The extracted critical features (latent vectors, or feature maps when the data are images) are then transmitted to the hub at the server. The two major functions at the server are the classification task and the data reconstruction task (recover a copy the original image from the latent vectors). In other words, the latent vectors are input to the pre-trained classifier for prediction and they are also input to the corresponding decoder at the server for the reconstruction of the images.

Table 2.1. Information on the CIFAR10 and ImageNet (IMGNETA and IMGNETB) Datasets.

Dataset	Image size	# of images	Training Testing Ration	# of classes	Comments
Cifar10	$32 * 32 * 3$	60,000	5:1	10	
ImageNet-A	$256 * 256 * 3$	13,000	7:3	10	similar images
ImageNet-B	$256 * 256 * 3$	13,000	7:3	10	similar images

2.2.1 Training Stage

The dataset collected at each edge device is used to train an autoencoder for the corresponding device. This is done to take advantage of the data locality at each devices. Autoencoders are generative models where an artificial neural network is trained to reconstruct its own input in an unsupervised way. Fig. 2.3 illustrates all the components of an autoencoder and the training process. It is made up by two main blocks which are the encoder and the decoder [20], [21]. The encoder compresses the input X into a low dimensional representation of pre-determined size, called the latent vector denoted by Z that contains the most important features in the data. When the input data are images, Z will be the corresponding feature maps. The decoder then tries to reconstruct the original data from the latent vector Z. The reconstructed data obtained at the decoder output is denoted by X$\hat{}$. It should be noted that an autoencoder is a lossy network as the original image will not be fully recovered. However, it is expected that the critical features will remain in the recovered image.

The autoencoder achieves the proper training of the encoder and decoder by minimizing the differences between original input and the reconstructed input. This is achieved by the use of the mean square error loss function or any other loss function. After the training of the autoencoder, the encoder part of the autoencoder is then extracted, deployed in the inference mode on the edge device, and then used

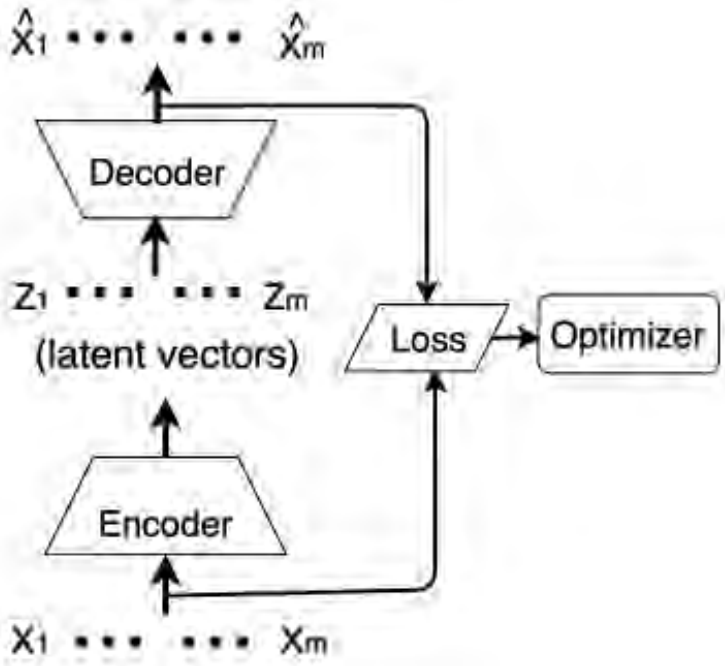


Fig. 2.3. The training for the proposed autoencoder at edge device.

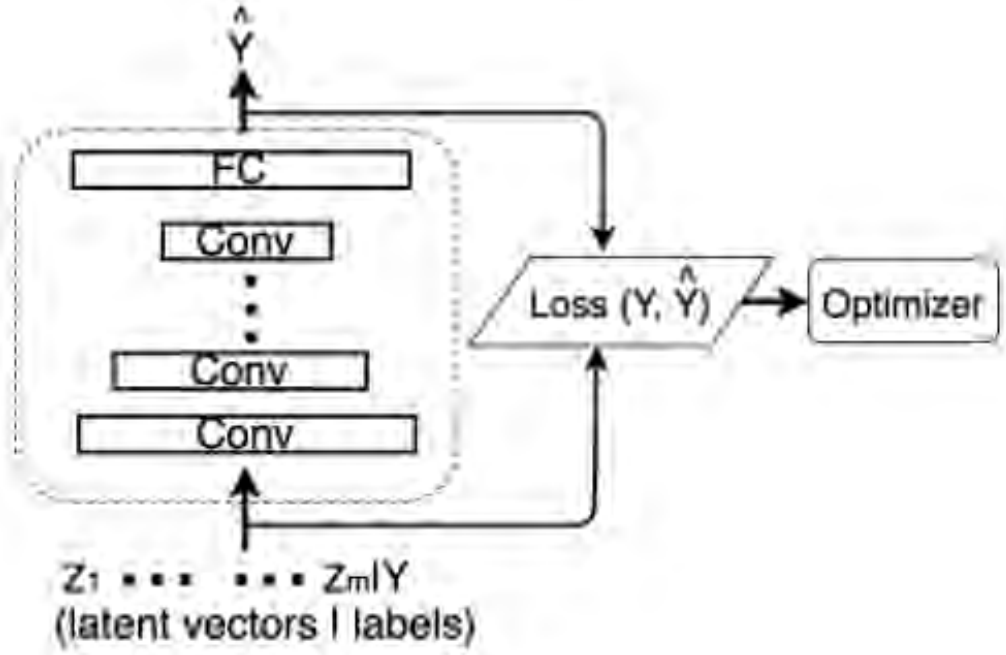


Fig. 2.4. The training for the proposed CNN classifier at the server.

to generate the latent vector Z. Hence, the dataset is transformed from [X, Y] to [Z, Y] where Y are the labels. The latent vectors and the corresponding labels are then aggregated at the hub and they are used for training a classifier on the cloud in a supervised manner as shown in Fig. 2.4. The type of classifier on the cloud is determined by the nature or type of supervised task to be done. The most common type of classifier used for image dataset is the convolutional neural network (CNN) and it uses the cross entropy loss function.

2.2.2 Inference Stage

In this stage,the pre-trained encoder, decoder and classifier are deployed in the inference mode. The data X from the edge device is fed to the corresponding pre-trained encoder attached to it. The encoder then transforms the data X to a latent vector Z which represents the most critical feature in X. The latent vector Z, which is smaller than X by a pre-determined ratio, is then transmitted to the cloud. At the cloud server, the latent vector Z is then fed into the pre-trained classifier and predict a label $\hat{Y}$. In situations where the original data is needed on the cloud, say I want to see a copy of the original image, the latent vector Z is also fed into the input of the corresponding decoder and the estimated data is obtained.

2.3 Experiments

2.3.1 Dataset Description

The result in this work was generated using three different datasets summarized in Table 2.1. These datasets were from the Canadian Institute For Advanced Research dataset (CIFAR10) [22] and the ILSVRC (ImageNet) 2012 datasets [23]. 1) Canadian Institute For Advanced Research (CIFAR10)is a dataset contains 60,000 color images, and is a subset of about 80 million labeled but tiny images. The dataset is further divided into 50,000 training samples and 10,000 testing samples. It has about 10

classes which are mutually exclusive and there is no semantic overlaps between images from different classes. 2) ILSVRC (ImageNet) 2012: The original ILSVRC 2012 dataset contain about 1.2 million color images of different sizes across about 1,000 classes. The 1,000 classes are either internal or leaf nodes but do not overlap. Two subsets of the ILSVRC 2012 dataset termed IMGNET-A and IMGNET-B were used in this work. Each subset contained about 13,000 images each resized to a dimension 256×256, spanning 10 classes. The images in each subset was further divided into training samples and testing samples with ratio 7:3. The difference between the two subsets lay in the type of nodes they contained. The IMGNET-A subset contains images from 10 different leaf nodes (diverse images) while IMGNET-B contained 10 child nodes from a single leaf node (similar images).

2.3.2 Deep Learning Model Design and Training Strategy

The autoencoder for the edge devices and the classifier at the edge server were chosen such that the autoencoder was optimized for feature extraction and the classifier was optimized for image classification. The autoencoder design and training strategywas affected by the type of images and the compression ratio. For instance, the model architecture for CIFAR10 dataset for compression ratios four and eight were different. This also applied for compression ratio four for the IMGNET-A and CIFAR10 datasets. Hence, different models were developed across several edge devices, compression ratios and datasets. Fig. 2.5 shows the model architecture for an autoencoder designed for CIFAR10 dataset for compression ratio of four. The model contained a mix of convolutional (same padding), max pooling, and upsampling layers. The relu function was used as the activation function for all layers except the last layer where the sigmoid function was used. The models were trained from scratch using glorot-uniform method as initializer, mean square error as the cost function and rmsprop optimization algorithm as the optimizer. After the convergence of the

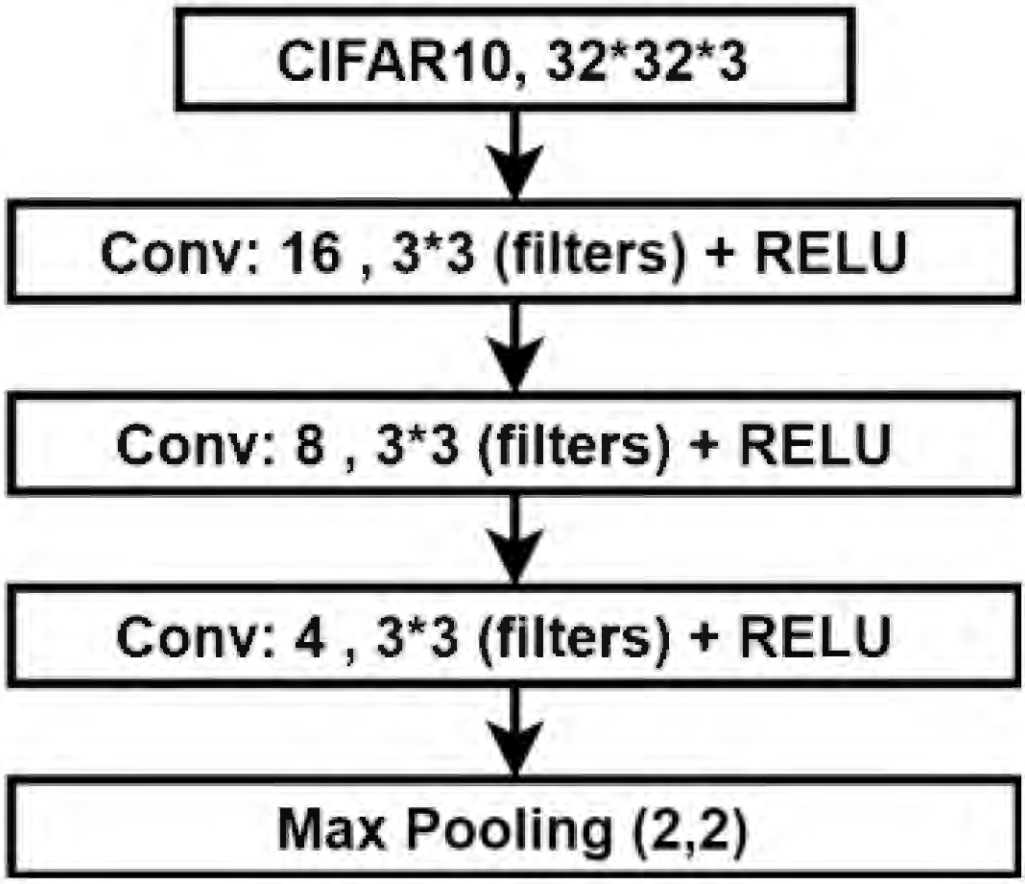


Fig. 2.5. Details of an encoder model for compression size of 4 using CIFAR10 dataset.

autoencoder model during training process, the encoder part of the autoencoder was then used in the inference mode to compress all the images.

2.3.3 Training Stage

1. Classifier Design: the convolutional neural network (CNN) models were used in this work. CNNs are well suited for image processing applications and other grid-like data [21]. They are more computationally efficient than the dense deep neural network thus reducing the memory usage. Using the filters, CNNs find and extract meaningful features from the images and preserve spatial relations. Three different CNN classifiers, denoted Model-A, Model-B and Model-C as listed in Table 2.2, were used in this work.

    - Model-A and Model-B: Model-A and Model-B are considered to be vanilla models because they were trained from scratch. Model-A and Model-B were specifically designed for the original input image and feature maps of CIFAR10 dataset and ImageNet dataset, respectively. The detailed CNN architecture of Model-A and Model-B are shown in Tables 2.3 and 2.4, respectively. The models contained a mix of convolutional, max pooling, and fully connected layers and relu and softmax activation functions. Furthermore, the models also contained some dropout layer in order to prevent over-fitting. The differences between Model-A and Model-B lay in the number of the different layers used and the use of padding in the convolutional layer of Model-A.

      The models were trained from scratch with the aim of minimizing the difference between the labels (ground truth) and the predicted labels. This was achieved by the use of glorot-uniform method as initializer, categorical cross entropy as the loss function and adams optimization algorithm as the

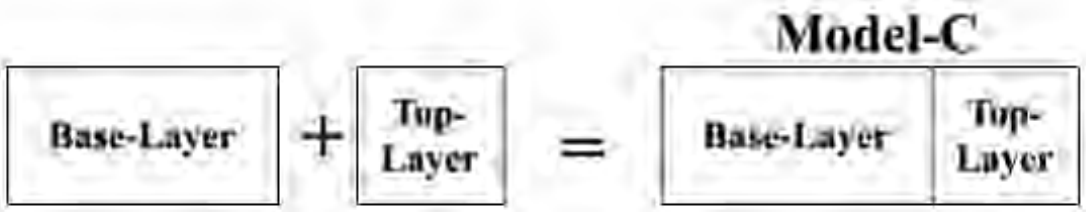


Fig. 2.6. The Transfer Learning Model Block (Model-C)

Table 2.2. The Deep Learning Models and the Dataset used in Training the Models

		Cifar10	ImageNet-A	ImageNet-B
Vanilla Model	Model-A	x	-	-
	Model-B	-	x	x
Transfer Model	Model-C	-	x	x

optimizer. Data augmentation was also used during the training process to mitigate overfitting due to the small quantity of the datasets. It should be stated that each classifier was trained with their respective original image and the feature maps (compressed images).

- Model-C: Model-C is a transfer learning based model designed specifically for the ImageNet dataset in this work. The CIFAR10 version was not presented in this work because of poor performance when used with the feature maps (compressed image) of CIFAR10 which can be attributed to its small dimension.

  The block diagram of the model is shown in Fig 2.6. Model-C is divided into two parts: the base layer and the top layer. The base layer is a pre-trained layer of other standard deep learning models without the fully connected layer that has been trained with data similar to the Imagenet

Table 2.3. The architecture of the vanilla model for CIFAR10 dataset (Model-A)

Vanilla Model For CIFAR10 Dataset
Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1,Padding
Activation Layer (Relu)
Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1, Padding
Activation Layer (Relu)
Max Pooling, Pool Size = 2*2,Stride = 1*1, Padding
Dropout(0.25)
Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
Activation Layer (Relu)
Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
Activation Layer (Relu)
Max Pooling,Pool Size = 2*2,Stride = 1*1, Padding
Dropout(0.25)
Flatten
Dense(512)
Activation(Relu)
Dropout(0.5)
Dense(10)
Activation(Softmax)

data and achieved better performance. By using this pre-trained models, the excellent feature extracting property of the standard model was being leveraged to achieve better performance. Furthermore, it also complemented data augmentation in training decent model in situations where datasets are limited. VGG16, VGG19, InceptionV3, InceptionResnetV2 and Resnet50 pretrained models [24] were used as base models for Model-C. The details of the top layer used for this work is shown in Table 2.5. It should be noted that the first dense layer of the top layer in Model 5 is smaller than that of the other models. This is because Model 5 was overfitting if the number of neurons in the first layer was 256, the same number used in the other models. Hence, the size of the dense layer was reduced to reduce overfitting and achieve good performance. A two-stage training method was used for the transfer learning model to minimize the error between the ground truth labels and the predicted labels. This approach was different from the training approach used for Model-A and Model-B which were trained from scratch. In the first stage, the base layer was fixed while the fully connected top layer was trained using the Adam optimizer after being initialized using the glorotuniform method. This was done in order to initialize the weight of the top layer close to the weight of the base layer. Thereafter, the whole model was retrained using the stochastic gradient descent (SGD) with momentum optimizer in order to tune the whole weight of the model appropriately. SGD with momentum was used because it is less aggressive than the adam optimizer as the use of an aggressive optimizer in the second step might cause the information in the base layer to be significantly eroded or lost. The binary categorical entropy was used as the cost function in the entire training process.

Table 2.4. The architecture of the vanilla model for Imagenet dataset (Model-B)

Vanilla Model For ImageNet Dataset
Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1,Padding
Activation Layer (Relu)
Conv2D, Filter Size=3*3, No of Filters=32, Stride=1*1, Padding
Activation Layer (Relu)
Max Pooling, Pool Size = 2*2,Stride = 1*1, Padding
Dropout(0.25)
Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
Activation Layer (Relu)
Conv2D, Filter Size = 3*3, No of Filters = 64,Stride = 1*1, Padding
Activation Layer (Relu)
Max Pooling,Pool Size = 2*2,Stride = 1*1, Padding
Dropout(0.25)
Flatten
Dense(512)
Activation(Relu)
Dropout(0.5)
Dense(10)
Activation(Softmax)

2.3.4 Experimental setup

This work sought to propose a new approach to design and implement deep learning models for distributed systems without compromising on data privacy and security. It achieved this by extracting the most important/critical machine features intelligible yet human unintelligible features from the dataset. These features were then transmitted across the communication network from the edge devices to the edge server where they are aggregated and used to train a classifier. The experimental methods, performance metrics and tools used in validating the proposed frame-

work is explained in this section. The experimental method: involved a two-stage methodology was used to validate the proposed framework and this method was the same irrespective of the type of dataset or model used. In the first stage, the original training set of the original input dataset (uncompressed images) was used to train the classifier. Thereafter, the test set was then used to obtain the needed performance metric in order to set the baseline. In second the stage, the training set of the feature maps (compressed images of the dataset used in the first stage), is used to train the same classifier model. The feature map, which is smaller than the original image by a pre-determined factor, is obtained by passing the original dataset through the encoder of the autoencoder. Thereafter, the performance metric of the classifier is obtained using the test set of the feature maps and the performance compared to the baseline. For the performance metric, the effectiveness of the framework was assessed using a simple classification task. The test accuracy of the model obtained after the training process was used as the primary performance metric. Furthermore, the effect of the proposed method on the training and testing time, and the number of model parameters were also investigated. Regarding software and hardware, The design, training and testing of the deep learning models (autoencoders and cnn classifiers) were implemented using Keras deep learning framework on TensorFlow backend, running on a NVIDIA Tesla P100-PCIE-16GB GPU.

2.4 Results and analysis

The results of the experimental work are presented in this section. The performance of the proposed framework was compared with the baseline using the performance metrics stated in Section III-D2 above. The baseline performance was represented by compression ratio 1 and was synonymous with using the uncompressed image to test our various models. Furthermore, it should be noted that the vanilla model for the CIFAR10 and ImageNet datasets were different as stated in Section

Table 2.5. The Architecture of the Transfer Learning Model for Imagenet datasets (Model-C))

	Model 1	**Model 2**	**Model 3**	**Model 4**	**Model 5**
Base Layer	VGG16	VGG19	InceptionV3	InceptionResNetV4	ResNet50
Top Layer	Dense(256)	Dense(256)	Dense(256)	Dense(256)	Dense(50)
	Activation(Relu)	Activation(Relu)	Activation(Relu)	Activation(Relu)	Activation(Relu)
	Dense(10)	Dense(10)	Dense(10)	Dense(10)	Dense(10)
	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)	Activation(Softmax)

III-D.

2.4.1 Effect on Test Accuracy

Fig. 2.7 shows the testing accuracy of vanilla CNN Classifiers (Model-A and Model-B) when trained and tested with compressed and uncompressed CIFAR10 and Imagenet datasets. The testing accuracy for the compression ratio 1 (uncompressed images), representing the baseline, was highest across all the cases, as expected. This was because all features in the raw images could be used for classification. Furthermore, the testing accuracy for IMGNET-A is higher than that of IMGNET-B. The differences in performance can be attributed to the very close similarity in the images in IMGNET-B, as classifying such images is a much more difficult classification task as compared to classifying images in IMGNET-A.

A general degradation in the testing accuracy is observed in Fig. 2.7 as the compression ratio was increased, although the rate of reduction varied across the models used for the three datasets. The rate of degradation of the testing accuracy of the model tested for CIFAR10 dataset is the highest for all the compression ratios. This was because of the small dimension of the CIFAR10 images (32*32), implyinh that the amount of features needed to perform a classification task was even smaller when compressed. Furthermore, the rate of degradation of the testing accuracy for IMGNET-A dataset was very modest across all the compression ratios. However, similar performance was not observed in IMGNET-B, particularly for compression

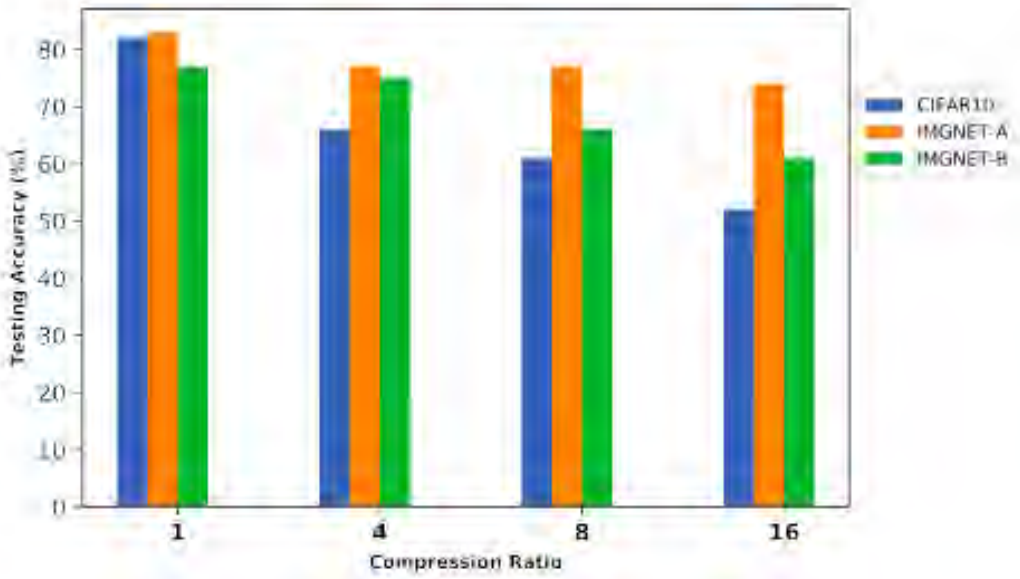


Fig. 2.7. Comparison of the testing accuracy of the vanilla models for the original dataset (compression ratio =1) and compressed dataset (latent variables) with compression ratio = 4, 8, 16.

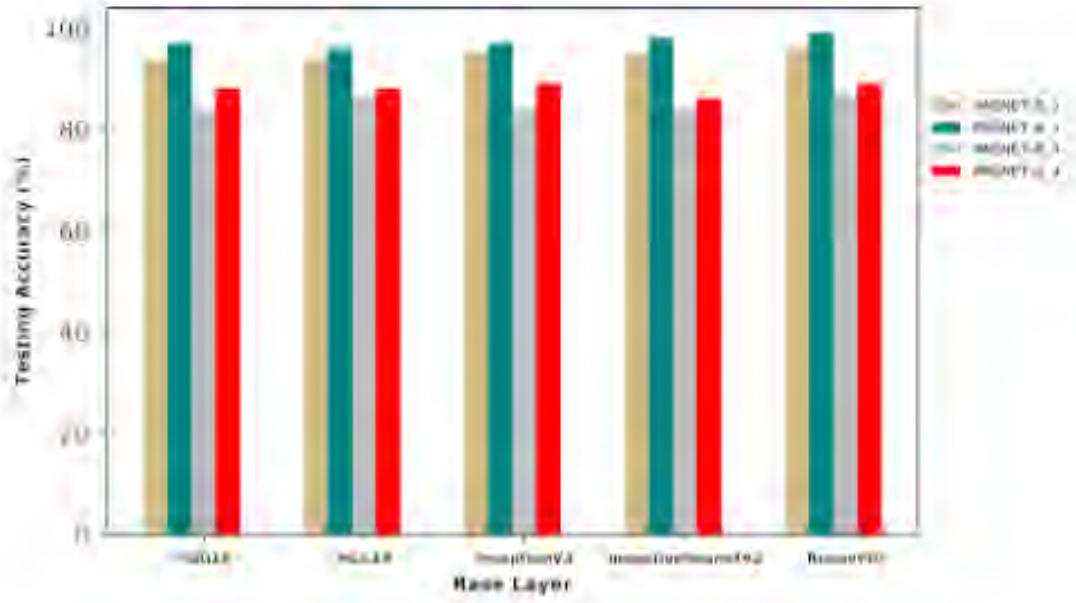


Fig. 2.8. Testing accuracy of the transfer learning based model (Model-C) using different base models for the ImageNet dataset with compression ratio = 4.

ratios 8 and 16 despite having the same image dimension (256*256). The bigger degree of degradation observed in the case of IMGNET-B for compression ratios eight and sixteen was due to the complexity of the classification task. This was because of the similarities in the images that made up the various classes in IMGNET-B, unlike IMGNET-A where the the images that made up the classes were very different.Hence, the complexity of its classification task means it requires a lot more features than that of IMGNET-A. Furthermore, bigger compression ratio also means the model has less amount of features to make a classification decision. The testing accuracy of transfer learning based model (Model-C) for the ImageNet dataset compressed by a factor of four, using different base models, is shown in Fig. 2.8. The transfer learning model was not designed and trained (see Fig. 2.8) Testing accuracy of the transfer learning based model (Model-C) using different base models for the ImageNet dataset with compression ratio = 4. using the CIFAR10 datasets as its performance was poor with the compressed images. The poor performance can be attributed to the very deep nature of the transfer learning based model leaving inadequate number of features available at the beginning of the fully connected layer (top layer) where classification took place. The same reason also explains why the transfer learning model was only designed and tested with ImageNet dataset with compression ratio one and four only. The testing accuracy of the transfer learning model across different base models for IMGNET-A and IMGNET-B datasets at compression ratio one (baseline) and four is higher than the corresponding performance of the vanilla model (Model-A and Model-B). This performance can be attributed to the powerful feature extraction property of the different base layer used. However, the rate of degradation in the testing accuracy for compression ratio four was higher than what was observed for the vanilla models. This can also be attributed to the very deep nature of the transfer learning models as a small amount of information/features left was not distinct enough to make accurate

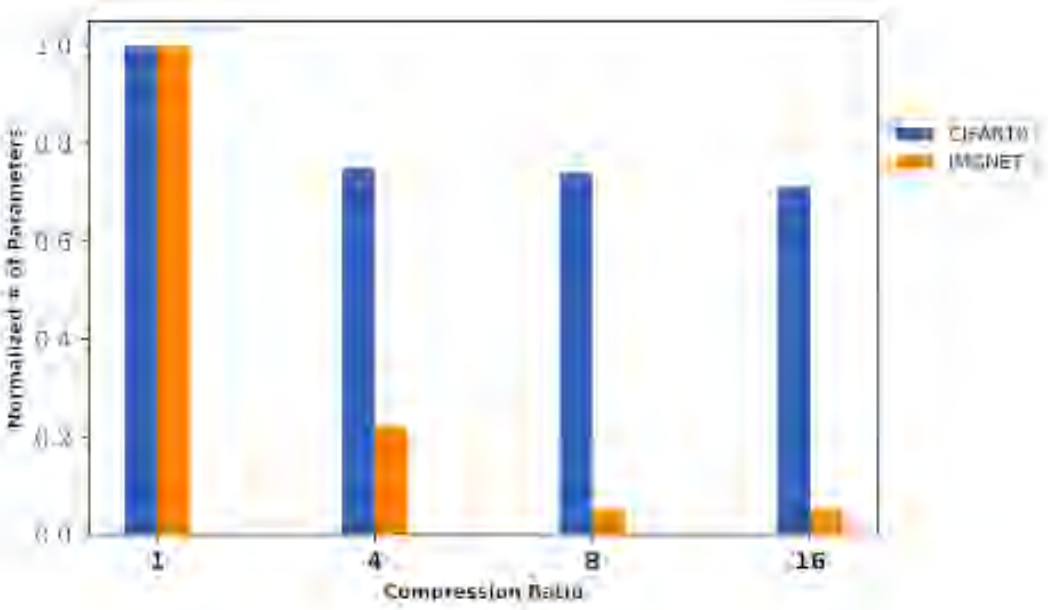


Fig. 2.9. Comparison of the normalized number of vanilla model parameters vs. data compression ratio

classification.

2.4.2 Effect on Number of Parameters

The number of parameters in a convolutional neural network is determined by many factors such as the filter size, the number of filters, the size of the input data, and the number and type of hidden layers, etc. Hence, reduction in the number of parameters can be achieved by reducing the size of the input data. The relationship between the normalized number of parameters in the vanilla model for the CIFAR10 and ImageNet datasets versus the compression ratio is shown in Fig. 2.9. It can be observed that for the same compression ratio, the rate of reduction in the normalized number of parameters of the vanilla model for the ImageNet dataset was bigger than that for the CIFAR10 dataset.

The bigger rate of reduction can be attributed to the size of the image, and in

turn, the number of features, which impacted the number of parameters in the fully connected layers of the model. It was also observed that the rate of reduction in the normalized number of parameters for each model appeared to be flat after a certain compression ratio which varied from model to model. The flatness was due to the reduction in the significance of the fully connected layer to impact on the number of parameters, as the number of features reduces below a certain point.

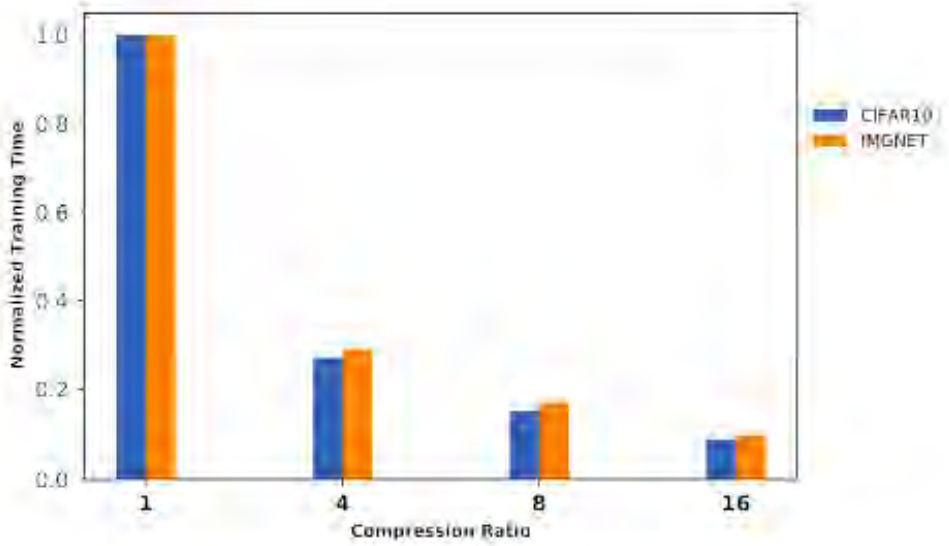
2.4.3 Effect on Testing and Training Time

Fig. 2.10 shows the comparison of the normalized testing time and training time of the vanilla models for various compression ratios for CIFAR10 and ImageNet datasets.

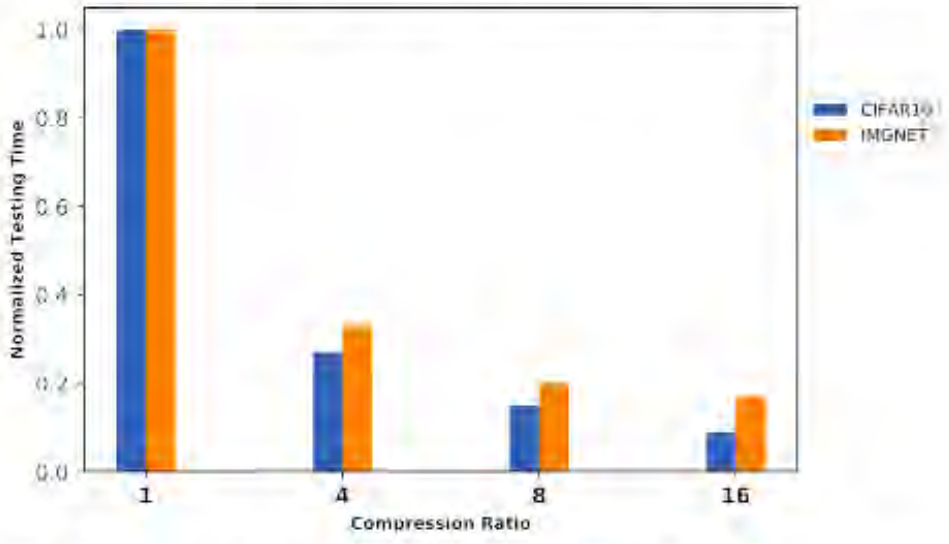
Fig. 2.10 show the normalized amount of time required for testing and training, respectively, of the vanilla models for various compression ratios for CIFAR10 and ImageNet datasets. A reduction in the amount of training and testing time was observed across the compression ratios and the models. The reduction can be attributed to the decreasing size of the input data and the smaller number of parameters to learn, when compression ratio increased.

2.5 Discussion and Related Works

There are several methods proposed in the literature to address the privacy and security concerns associated with data used for training deep learning models. Examples of popular approaches include homomorphic encryption [25], differential privacy [12], [26] and secure multiparty computation [27]. Despite the successes of these methods, some issues remain such as performance degradation, non-trivial overhead or limited application [28]–[29]. The use of collaborative deep learning method, such as federated learning, has been introduced in recent years to solve the problem of data privacy. Federated learning is a type of machine learning where the goal is to train a high quality centralized model while the data remains distributed over a



(a) Comparison of the normalized testing time



(b) Comparison of the normalized training time

Fig. 2.10. Comparison of the normalized testing time and training time of the vanilla models for various compression ratios for CIFAR10 and ImageNet datasets.

large number of clients [14]. It involves the sharing of model parameters and model gradients through a parameter server without sharing their local data. Federated learning is based on an iterative model averaging and it is robust to unbalanced data and non-i.i.d. data distribution. Federated learning has been applied to mobile keyboard prediction, vocabulary word learning and google keyboard query suggestions improvement [30]– [31]. Federated learning may be viewed as an extension of the idea discussed in [32] that stochastic gradient descent can be implemented in parallel and asynchronously. Federated learning may suffer from non-trivial communication cost. To deal with the high communications cost, an efficient multi-objective evolutionary algorithm, based on a scalable network connectivity encoding method, was proposed in [33]. To help reduce the uplink communication bottleneck, the use of structured and sketched updates was introduced in [34]. Federated learning may also suffer from security/privacy issues due to the need to communicate the model parameters to the central server. One recent study showed potential security/privacy issues due to the possibility of reconstructing original data from the shared gradient [35]. Secure aggregation, a type of secure multi-party computation algorithm for federated learning was introduced in [36]. This helps guarantee the privacy of data used in generating gradients shared by each model and improve communication efficiency. Furthermore, it was observed that federated learning performs poorly when the data distributed across the training center is strictly noni.i.d. of a single class. This statistical challenge was resolved by creating and using a small subset of data which is globally shared between all the edge devices [30] or adopting a multitask learning approach [37]. Autoencoder has been applied to address data privacy concerns in several recent works [13], [38], [39]. In [38], a convolutional autoencoder that perturbs an input face image to impart privacy to a subject is proposed. It is shown the method can protect gender privacy of face images. A proof-of concept study was performed in [13] to use

an autoencoder for preserving video privacy, especially when non-healthcare professionals were involved. A modified sparse denoising autoencoder has been applied in [39] to reduce the sparsity and denoise the data. Then a three-class classification is performed on the reconstructed data from the autoencoder and it is shown that the classifier can classify the original black class data as the transformed gray class data. Although autoencoder has been used to address data privacy concerns, this work is the first in the use of autoencoder for addressing privacy concerns, communication cost, and deep learning efficiency associated with mobile edge computing systems with large number of edge devices. This was achieved by using the autoencoder to extract human unintelligible but machine intelligible features from the data. The features or latent vectors were then used to train the classifier. Furthermore, the proposed approach comes with the added advantage of reducing the dimensionality of data needed to be transmitted, thus reducing the communication cost and the number of model parameters, as well as training and inference time. This approach did not suffer from leaking gradient problem associated with federated learning [35].

2.6 Conclusions

A novel edge computing framework for designing and implementation of privacy preserving image classification models was proposed in this work. The proposed framework provides 1) flexibility of training autoencoder at each edge device individually, thus protecting data privacy of end users because raw data is not transmitted at any time; 2) after the training of autoencoder was complete, raw data was "compressed" and "encrypted" by the encoder before transmitting to the edge server, and this will reduce the communications cost and further protect the data privacy and security; 3) the autoencoder provided features to the classifier at the server, thus allowing the classifier to be trained on the features with less and controlled dimensions; 4) the decoupling of the training of the autoencoder at the edge devices and the

training of the classifier at the edge server relaxed the frequent communications requirement between edge devices and edge server. Experiments have been carried out using CIFAR10 and ImageNet datasets, and detailed analysis of the tradeoff between classifier accuracy, dimensionality of data, compression ratio and different choice of classifiers have been given to provide benchmark and insights on the proposed scheme. For future work, comparison with federated learning in terms of classifier performance versus the communications cost and model complexity will be carried out for image classification tasks. This helped quantify the pros and cons of the proposed approach. Furthermore, the use of other types of autoencoder to extract latent variables and use of knowledge distillation to help mitigate the reduction in the model accuracy were explored.

CHAPTER 3

INFERENCE PERFORMANCE COMPARISON OF CONVOLUTIONAL NEURAL NETWORKS ON EDGE DEVICES

3.1 Introduction

The Internet of Things (IoT) is responsible for generating a large amount of data using different devices with an unprecedented speed. Any artificial intelligent application can be created to make it more user friendly and fast. This dataset was used to train an algorithm for predictive analysis. Usually this is computationally expensive and data center/cloud is used to perform this computation. Now it is necessary to process the data on local device as it protects user privacy and makes it more secure. It also reduces communication cost by saving bandwidth by not sending the data into the cloud. This section will cover how different popular and widely used pre-trained convolutional neural networks performs on three popular IoT devices. These IoT devices are specially designed for machine learning purposes. The experiment had a common test dataset which was used for each case. Three pre-trained models such as MobileNetV1 & V2 and Inception V3 were used for image classification task on NVIDIA Jetson TX2, Jetson Nano, and Google Coral Edge TPU USB Accelerator. Moreover, the model complexity has been reduced using quantization. This made the model more lighter and faster and to some extent, more suitable for low computational edge devices. The results will serve as a benchmark for practitioners of real-time and local learning for this type of task.

Pervasive interconnected smart devices interacting with one another have become

very popular as the development of low-cost sensor, wireless communication technologies and new internet techniques [40] have become rapid. It is formally known as Internet of Things and these devices are booming for their functionality and low-cost. There are already a huge number of IoT devices producing a huge amount of data which needs to be processed. In this circumstances, deep learning methods can be used to create models which can process data and a commonly followed method of this is shown in Fig. 3.1. IoT sensors and devices are responsible for creating a huge number of data and machine learning algorithms can be used to process to make future predictions. Typically, these sorts of method requires high computational power to train and evaluate the model.

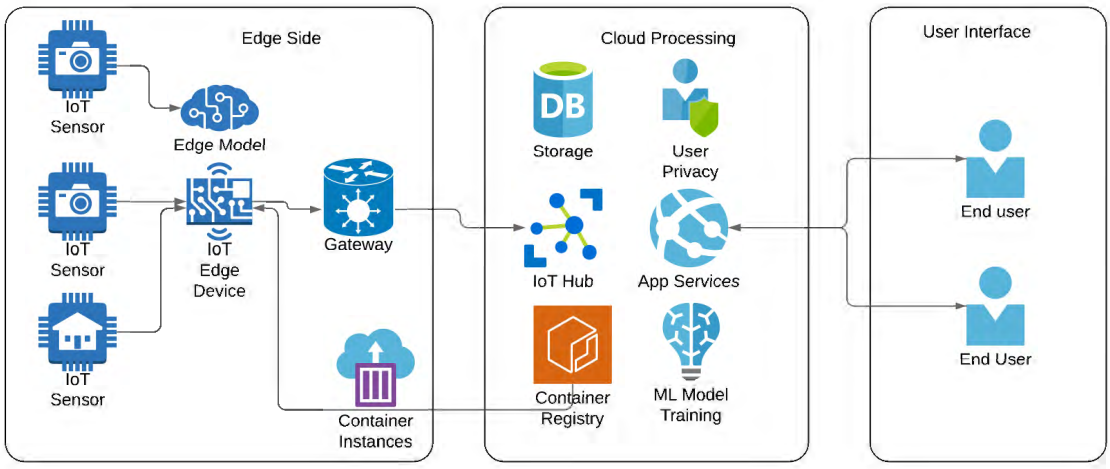


Fig. 3.1. Processing IoT data in cloud

Recently, the proliferation of IoT and the advancement of artificial intelligence increased the incentive for an intelligent edge device that can operate on real-time and locally. Although centralized procurement of big data is a formal method, it is sometimes not feasible to transmit the data due to privacy, bandwidth or adversarial

attack. Sometimes the delay in the transmission may break the tolerance level of the time-sensitive applications [41]. One reason can be limited wireless bandwidth availability due to exceedingly amount of edge devices. Moreover. transmitting private and confidential data is a big concern when the transmission method is not secure [42]. This triggers mobile edge computing where the aim is to perform the computation closer to the data [41]. Highly sophisticated network and client nodes is provided by the present wireless edge networks. It is also equipped with powerful sensors, larger computation and a good amount of storage resources. Considering these aspects, there is a huge opportunity to deploy mobile edge devices as learning engines. These devices can use on-device collected data or local data from the edge nodes in forms if audio, video or text and procure local learning models (edge model, see Fig. 3.1) without sending the original data through the cloud to the central processing zone. This approach significantly reduces the bandwidth cost, reduces latency and protects data privacy more so than the conventional learning solutions.

There are several examples where it is essential to process the data locally. A surveillance camera of an office can be an example. These cameras are the main equipment to monitor the security of a certain area. Normally these cameras are deployed at high number and requires a good amount of human resource to monitor all of them. Machine learning models can leverage the human resource required for this job. These models can be trained to detect any adversarial attack or an unfamiliar personal. Usually the data needs to be transferred to a high computational station where it can be procured. For this real-time application, any latency or bandwidth unavailability may be impractical. Running the model locally is a solution for this situation. The camera connects to an IoT device capable of locally running deep learning models without the need to transfer the data to a central station. The model will raise alarm or alert the authority for any unfamiliar personal. This saves

a lot of time, bandwidth, human resource and computational cost.

However, limited computational power and energy resources on edge devices are the biggest challenge for practical accomplishment. The enhancement of complexity of the DNNs is also another concern. The model is becoming more complex as the DNN models are providing solution to more complex problems. Some recent studies can also be found covering inference performance of popular machine learning models on IoT devices. It has been realized that Jetson Nano can run MobileNet V1 model at 64 FPS for images with resolution 300×300 on TensorFlow framework, and also can process larger image size of (960×544) at 5 FPS on a ResNet-18 SSD backbone [43].

This section covers a comprehensive comparison of inference performance of three convolutional neural networks (CNN) models, namely, MobileNet V1 [1], MobileNet V2 [44], and Inception V3 [45] by running image classification tasks on the same dataset. It used three edge devices specially designed for machine learning applications, NVIDIA Jetson TX2, NVIDIA Jetson Nano, and Google Edge TPU based on quantization. These machines did not have high computational power and the models needed to be lighter or compressed before execution. Extensive research was conducted on these techniques such as pruning [46], quantization [47, 48], binarized neural network [49, 50], and tensor decomposition [51] to reduce the model complexity and faster execution. The aim was to benchmark the advantages and disadvantages of deep learning models when running those models considering their performance and speed. This provides a guidance of practical choices of compressed DNNs over edge devices to achieve certain specifications of applications specially for delay-sensitive applications.

3.2 Deep Learning Models and Tools

3.2.1 Convolutional neural network models

Convolutional neural network (CNN) [10] is a category of multilayer neural network that was been chosen for the image classification task for this section. It was formed with three main components: convolution layer, pooling layer as well as non-linear activation layer and finally the fully connected layer. In general the convolutional layer extract features by parsing different size filters across the image. Pooling layer is for making the computation faster by decreasing the size of the input. It also lowers the risk of overfitting by selecting certain features. The fully connected layer was the final layer where the main prediction was made by getting the probability of the image belonging to a certain class by sending it through an softmax activation function. In image classification top-1 and top-5 accuracy is defined in many cases where top-1 refers to the highest probability score of the class which is same as the output class. On the other hand top-5 accuracy means that the actual output belongs to the either of the top 5 classes with the highest probability. CNN is not only limited to image classification task [52], but can be used for object detection [53], object tracking [54] and other applications also.

This section utilized three pre-trained models: MobileNet V1 [1], MobileNet V2 [44] and Inception V3 [45] for performance comparison. These models are popular for image classification task. Moreover, quantization method was used to tune these models into more lighter versions. It made the models compatible for edge devices without losing too much performance. MobileNet [1] is a popular model which is suitable for mobile and embedded devices for its state-of-the art architecture. The architecture ensures a faster execution of the model and also can be executed with very limited computational power. It breaks down the standard convolution into

into parts knows as depthwise convolution and pointwise convolution. Also width and resolution multipliers are used to tune the width of the image for version 1. This significantly reduces the computation and makes it faster. Equation (3.1) shows regular convolution operation (RCO) and Equation 3.2 shows depthwise followed by pointwise convolution operation (PCO) [1].

$$Cost_{RCO} = D_K * D_F * M * N * D_K * D_F \tag{3.1}$$

$$Cost_{PCO} = D_K * D_F * M * D_K * D_F + M * N * D_F * D_F \tag{3.2}$$

where D_K and D_K are the height and width of filter while D_F and D_F are the height and width of input feature map size. Moreover, M is the number of input channel and N is the number of output channel. The comparison between these two convolution operations results in $1/N + 1/D_K^2$, which means PCO has less computation than the RCO [1].

MobileNet V2 is the more advanced version which can perform better than the previous version. One key difference is that version 2 has a new layer which is called the projected layer. It is used more like an expansion of the layer.Residual connections are introduced in this version to address the overfitting problem of the model. The architecture used convolution layer with 32 filters. 19 residual layers were followed after this layer and ReLU6 is utilized as the activation function. Kernel size was kept standard at 3×3. The entire model was trained using 16 GPU and a batch size pof 96 was chosen. MobileNet V2 performed faster than MobileNet V2 to some extent [44].

Inception V3 [45] uses approaches such as factorizing convolution, auxiliary classifier, grid size reduction to build a more robust and lighter model. The model architecture reflected more accurate performance than the MobileNet models. For example, a large 5×5 filter is replaced by two 3×3 filters. This reduces the number

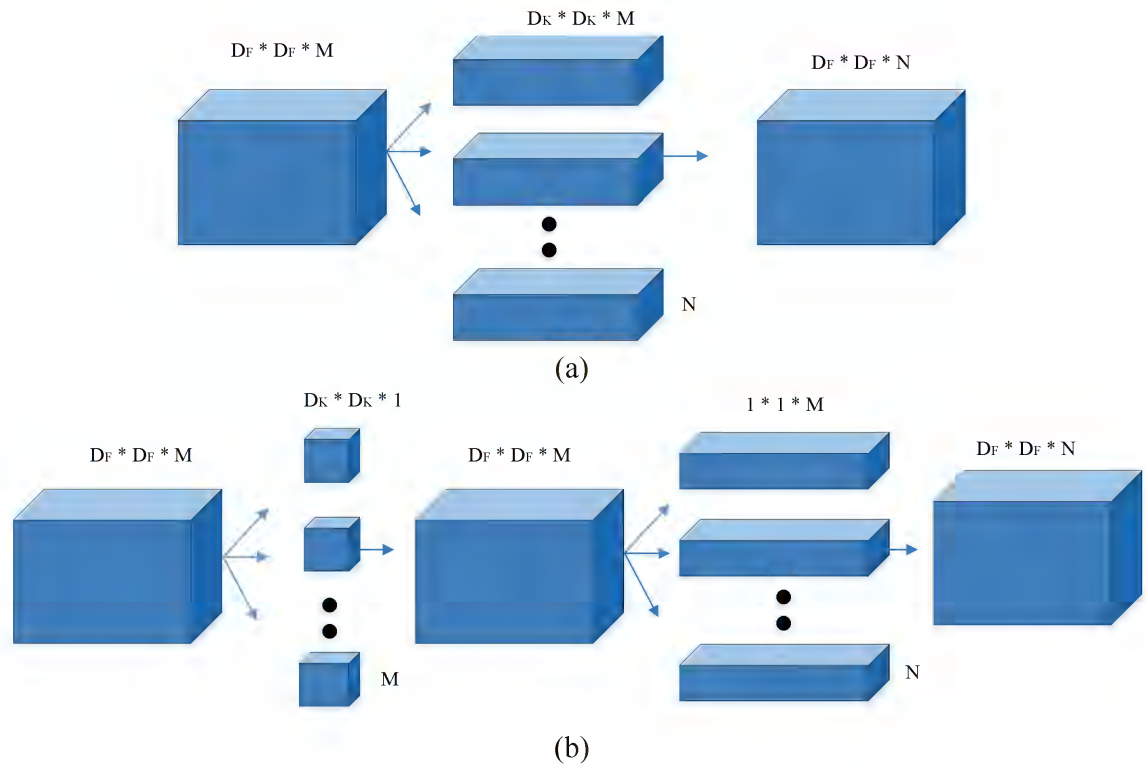


Fig. 3.2. (a): Regular convolution operation (RCO), (b): Point-wise convolution operation (PCO) [1]

of parameter from 25 to 18. Moreover, a 3×3 filter was replaced by 3×1 and 1×3 filter. This also reduced the parameter number from 9 to 6.

3.2.2 Model compression

CNN models require high computational resources for execution. In contrast edge devices have very limited computational and memory resources. Simplifying the CNN models into more lighter model is essential before deploying in edge devices [55]. The advantage of making the model lighter is that it makes the model faster but it also reduces the performance [56]. The weights in the DNN require huge memory for procurement and the aim is to reduce that requirement. Pruning [46], quantization [48], and data compression [57] are some of the popular model compression techniques. The main concept of pruning is to find the less important neurons [46]. Then the weight value can be either made zero or ignored in the network. Quantization refers to the method where the size of the floating point operation is reduced in order to make it faster and less resourceful [48]. It is extremely popular for speeding up the process as lesser computation is required between memory and network. The memory consumption can also be reduced without losing much performance. For example, binarized neural network is a version of quantization where the activation and weights are represented in binary, making it easily compatible for memory constrained devices [49]. Data compression is also an example of model compression. Parameters that were tuned can be stored in compressed form and can be decompressed during the execution [57].This also adds an extra layer of security in the data as it becomes encrypted during the compression technique.

3.2.3 Software tools

This research is based on evaluating different device and software libraries. TensorFlow was the main software used for this experiment. TensorRT was another

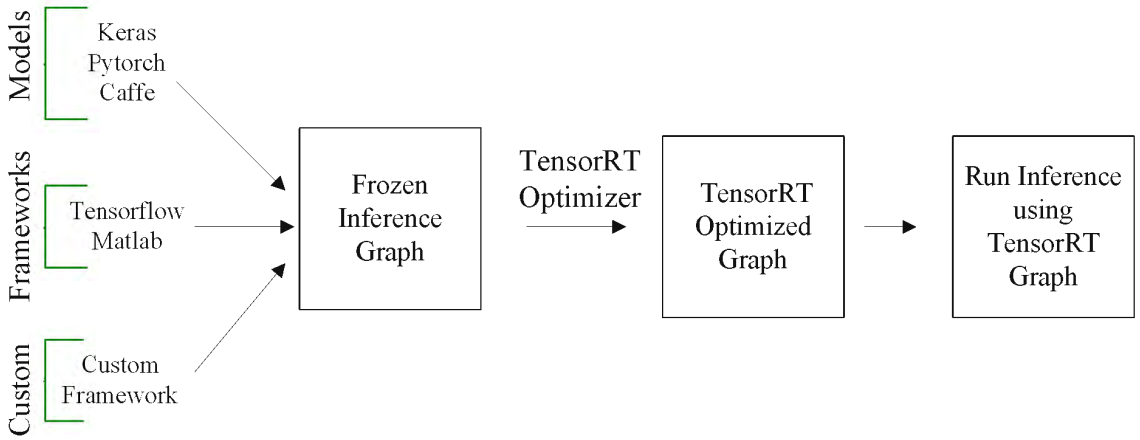


Fig. 3.3. TensorRT workflow [2]

software kit that was used which has high performance deep learning interface based on CUDA [2]. Maximum throughput can be achieved with lower latency by this tool. Another advantage of TensorRT is that it can be made compatible with other frameworks such as TensorFlow or MATLAB and supports APIs such as PyTorch, Keras and Caffe [58]. This entire information can be viewed in Fig. 3.3. It is compatible to provide the reduced bit point operation for computer vision, natural language processing and so forth [2, 59]. In this experiment we used three types of floating point operation: 32-bit, 16-bit and 8-bit operation. The 32-bit point operation was created directly from Keras applications. Keras is popular deep learning API which runs on top of TensorFlow [60]. Moreover, it gives user the flexibility to design any model by its simplicity and ease of use. TensorRT was used to convert the 32-bit operation into 16-bit operation. Lastly the 8-bit operation was directly derived from TensorFlow Lite model [3].

3.3 Edge Devices

Fig. 3.4 shows the devices used for this experiment. Two NVIDIA Jetson devices were selected for this part. It gives a good platform for machine learning tasks.

Specially these have a lightweight GPU installed making deep learning models run fluently. NVIDIA Jetson TX2 and Nano were selected where both of these have a 4-core ARM A57 core. TX2 performs at 2 GHz. Also both of these have 256-core and 129-core Pascal respectively.

Fig. 3.4. Left: Jetson TX2, Right (Top): Jetson Nano, Right (Bottom): Google Coral TPU USB Accelerator

Google Coral Edge TPU USB Accelerator is the third device used for the experiment. It is specially designed for deep learning task and cannot operate regular computer programs. Multiple applications such as machine vision, robotics, medical, retail and many more [61] can be deployed using this interface. It uses Tensor Processing Units (TPUs) which is a custom-developed application-specific integrated circuits (ASICs) designed for machine learning task by Google Inc. Google has this

technology in both cloud and edge devices. The edge devices are called Coral Development Board and USB Accelerator. This experiment used USB Accelerator and referred as Edge TPU as shown in Fig. 3.4. It is based on TensorFlow Lite and the hardware was designed in such way that machine learning models could be executed at faster speeds than other devices. This research utilized 8-bit floating point operation and the workflow as illustrated in Fig. 3.5. The difference between Dev Board and the USB Accelerator is that the Board has its own CPU and the other does not. The USB Accelerator needs a support machine to operate. A special type of training method named quantize-aware training is needed for the creation of 8-bit pre-trained model. The processing is designed such way that hundreds of thousands of operation of matrix multiplication can be performed where a conventional GPU can only do tens of thousands. It does the multiplication and addition instantly without sending those into the memory making them faster [3]. The comparison can be viewed in Table 3.1.

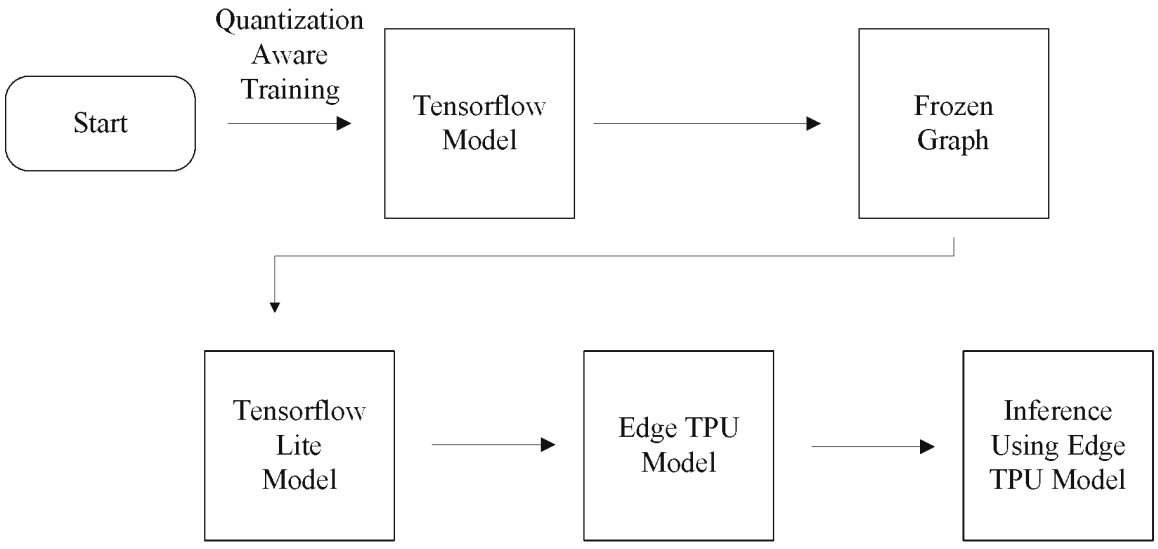


Fig. 3.5. TPU USB accelerator workflow [3]

Table 3.1. Edge Device Comparison

	Jetson TX2	Jetson Nano	Google Coral TPU USB Accelerator
Memory	8 GB	4 GB	NA
Storage	32 GB	16 GB eMMC	NA
Processor	Quad-Core ARM Cortex-A57 MPCore	Quad-core ARM Cortex-A57 MPCore	NA
AI Accelerator	256 Cuda Core (Pascal)	128 Cuda Cores (Maxwell)	Edge TPU

3.4 Experiment

3.4.1 Dataset

ImageNet is the most popular dataset for computer vision tasks. It is consists of 1.2 million images with 1,000 classes [62]. All the popular pre-trained models are also created using this dataset. We employed ImageNet 2012 validation dataset which had 50,000 images of 1,000 classes. The goal was to test the performance using pre-trained models based on TensorFlow and use different edge devices for comparison.

3.4.2 Evaluation Metrics

We utilized different evaluation metrics for performance comparison. They are listed below.

- FP denotes floating point operation taken 32 bit and 16 bit for Jetson devices and 8 bit operation for Edge TPU.
- Accuracy refers to the ratio of number of correct predictions to the total number of testing samples.
- Memory denotes how much dynamic memory has been allocated by the python thread in mebibyte or MiB (1 MiB = 1024×1024 bytes).
- Load denotes the pre-trained model loading time in seconds.

- Time denotes the total time python script requires to execute the task.
- Average Inference (Avg Inf) is the average inference time of a single image in seconds.
- FPS (inf) represents the frame per second which is how many images can this model run in one second. FPS (inf) shows the rate of processing image while only considering inference whereas the last column FPS shows the rate of processing image while taking model loading, image preprocessing and inference into consideration.

Frame per second (FPS) and Frame per second (inf) are given in equation (3.3) and (3.4).

$$FPS = \frac{N}{M + \sum_{n=1}^{50000}(P_n + I_n)}, \tag{3.3}$$

$$FPS(inf) = \frac{N}{\sum_{n=1}^{50000} I_n}. \tag{3.4}$$

Here N denotes the 50,000 validation images. M in the denominator is the model loading time, P is the time for preprocessing of images, and I is inference time. The model loading occurs only once but preprocessing and inference occurs for all 50,000 images.

Table 3.2. Performance Comparison on Various Edge Devices with MobileNet V1.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg Inf (ms)	FPS (inf)	FPS
TX2	32 bit	0.68364	1595.426	30.98	2582.42	40	25	19.36
TX2	16 bit	0.68374	2267.48	363.11	2033.77	20	50	24.58
Nano	32 bit	0.68362	1147.215	20.04	4591.97	70	14.29	10.89
Nano	16 bit	0.68372	2136.59	82.95	2151.96	20	50	23.23
TPU	8 bit	0.68008	108.516	3.06	1235.07	9.43	106.04	40.48

Table 3.3. Performance Comparison on Various Edge Devices with MobileNet V2.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg Inf (ms)	FPS (inf)	FPS
TX2	32 bit	0.68048	1818.398	53.95	2799.24	40	25	17.86
TX2	16 bit	0.68048	1914.27	187.56	2278.44	20	50	21.94
Nano	32 bit	0.68048	1546.164	28.93	5471.34	90	11.11	9.14
Nano	16 bit	0.68084	2102.309	78.96	2206.76	20	50	22.66
TPU	8 bit	0.69026	103.078	3.07	1315.2	11.28	88.65	38.02

Table 3.4. Performance Comparison on Various Edge Devices with Inception V3.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg Inf (ms)	FPS (inf)	FPS
TX2	32 bit	0.76276	1674.637	88.99	8860.52	150	6.67	5.64
TX2	16 bit	0.76284	3656.887	1945.94	4302.21	20	50	11.62
Nano	32 bit	0.76276	1044.277	47.38	17752.81	320	3.13	2.82
Nano	16 bit	0.76264	3213.441	469.4	4191.77	50	20	11.93
TPU	8 bit	0.7705	147.883	3.13	25463.41	490	2.04	1.96

3.4.3 Results and Analysis

Three models named: MobileNet V2, MobileNet V2 and Inception V3 were evaluated using three different floating point operations. The 32-bit, 16-bit and 8-bit operations were tested on three devices respectively on NVIDIA Jetson TX2, NVIDIA Jetson Nano and Google Coral Edge TPU. The Jetson devices were used for 32-bit and 16-bit operation whereas Edge TPU was used for 8-bit operation.

Table 3.2 reflects the result of MobileNet V1. the 16-bit took more model loading time but lesser execution time for the entire program than the 32-bit operation. The 32-bit floating point operation was obtained from Keras applications and 8-bit from TensorFlow Lite original hub. Both of these used their own platform to execute the model, making it efficient. However, the 16-bit operation was derived from the 32-bit pre-trained model and executed on newly developed platform. This made the

loading of the model in the 16-bit more time consuming than the other two. Only considering average inference time of a single image, the 16-bit was better or faster than the 32-bit floating point operation. The execution time of the entire task and memory consumption was more for the TensorRT interface.

MobileNet V2 showed the same trend of result as MobileNet V1 shown in Table 3.3 for different devices. The memory consumption was more for the TensorRT platform or 16-bit operation probably because the converted model required more memory for initial loading. It was also visible in the model loading time column. It was observed that Jetson TX2 in 32-bit operation is almost 12 times faster. Moreover, due to some hardware issue the test dataset was loaded in SD card for all TX2 experiments. The inference time was better in TensorRT platform in Jetson TX2. However, The Edge TPU outperformed all other by a good margin in for MobileNet models in accuracy, memory consumption, execution time and FPS.

Lastly Table 3.4 evaluated the performance of Inception V3 model on various devices. The network structure of Inception was much more complicated and robust than the MobileNet V1 and V2. It had more layers and larger input image size. Now, the saved model in h5 format of Inception V3 is 25.1 MB whereas MobileNet V2 was only 4.5 MB. This model had more weights and parameters than the other two giving higher accuracy but slower time in the testing environment. The Edge TPU was connected through the USB 2.0 ports. In a separate environment the same model was connected with a USB 3.00 port to another host computer with USB accelerator. The average inference time was about 43.6 ms, or 22.94 FPS, almost two times faster than NVIDIA Jetson TX2 and Nano.

3.5 Related Work

3.5.1 Model compression

Model compression is a popular research for accelerating the speed of the deep learning models by reducing model complexity. For example, the convolution architecture was redesigned in MobileNet which was able to reduce the parameter numbers by seve times by only losing one percent accuracy. If the number of parametner of MobileNet is compared to another model like VGG 166, then it is reduced by almost 35 times [1]. Moreover, the procurement of data is a few times faster because of lower computation. Jiaxiang et al. [47] presented that quantization was able to speed up the inference process by four to six times and drastically reduce the number of parameters 15 to 20 times. Han et al. [63] presented an efficient three-stage pipeline of a CNN containing pruning, quantization and huffman coding which can reduce the famous CNN model AlexNet's size from 240 MB to 6.9 MB.

3.5.2 Deep learning inference on edge devices

This section demonstrated that complex deep learning models can be accommodated using compression techniques on resource constraint devices. The results were consistent with the literature review. Recent studies show that NVIDIA Jetson Nano runs MobileNet V2 at 64 FPS where Google Coral Dev Board runs it at 130 FPS [43]. Also single 64-bit Intel Xeon Gold 6154 CPU at 3.00 GHz as the host machine with TPU USB Accelerator can complete inference ar 2.4 ms for MobileNet models. Also the dev board can take up to 53 ms and 51 ms to do inference for MobileNet V1 and V2 [64]. Taylor et al. [65] also presented a research work on edge devices where more than seven times accuracy improvement and one and a half times reduce time were observed for an adaptive deep learning model selection.

3.6 Conclusion

This section showed a comprehensive comparison of testing performance of three pre-trained CNN model on three edge devices. Specifically, MobileNet V1 and V2 and Inception V3 models were used on NVIDIA Jetson TX2, Jetson Nano, and Google Coral Edge TPU USB Accelerator for image classification task. Moreover. quantization technique was applied to observe how the model reacted to different bit-point operation. It was applied to reduce the model complexity thus making it lighter and easier for edge devices. Experimental results indicated faster inference time and more accurate results for Edge TPU than the NVIDIA Jetson devices. However, the USB Accelerator could not work independently and required a host to run the model. In addition NVIDIA Jetson TX2 performed better than Jetson Nano in most cases. It was expected as Jetson TX2 have more computational resource than the Nano. This experiment drew a line for model selection for two different tasks: speed and accuracy. MobileNet models are suitable for speed where Inception reflects more accurate performance. The work demonstrated that specific model and edge devices were suitable for specific applications. This work will serve as a benchmark for researchers or users to measure the DNN models, compression techniques and edge devices for different applications.

CHAPTER 4

ROBUST FACE MASK DETECTION USING DEEP LEARNING ON IOT DEVICES

4.1 Introduction

Covid-19 had catastrophic effect on this world and became a global pandemic [66]. The spread of this disease was difficult to control and one measure was by using face masks in public. The masks stopped the respiratory droplets from spreading which was the main carrier of the disease [67]. Face mask detection is a new concept in this era and this idea can be incorporated with IoT devices to track and warn people to wear masks. This section utilized this idea of detecting face mask on IoT devices using deep learning. Specifically, four convolutional neural networks, namely, MobileNet V2, Inception V3, VGG 16, and ResNet 50 were deployed for face mask detection and tested by IoT devices such as NVIDIA Jetson TX2 and Nano. Both devices had mobile GPU mounted on top of their system making computation efficient and faster. Performance comparisons were measured using different training dataset size. The experimental results showed that face detection was possible in real time on IoT devices.

The spread of viruses can be controlled by wearing a face mask can easily be understood in Fig. 4.1. It reflects the risk factor between an infected person and an uninfected person. A person infected with coronavirus has a high possibility of infecting another person if no one is wearing a mask. The possibility is slightly less if one of them is wearing a mask. However, it is lowest when both of them are wearing a mask [68] (the third row). As coronavirus spreads by the respiratory droplets, if it can be prevented from going out, then there is a high chance of controlling the disease.

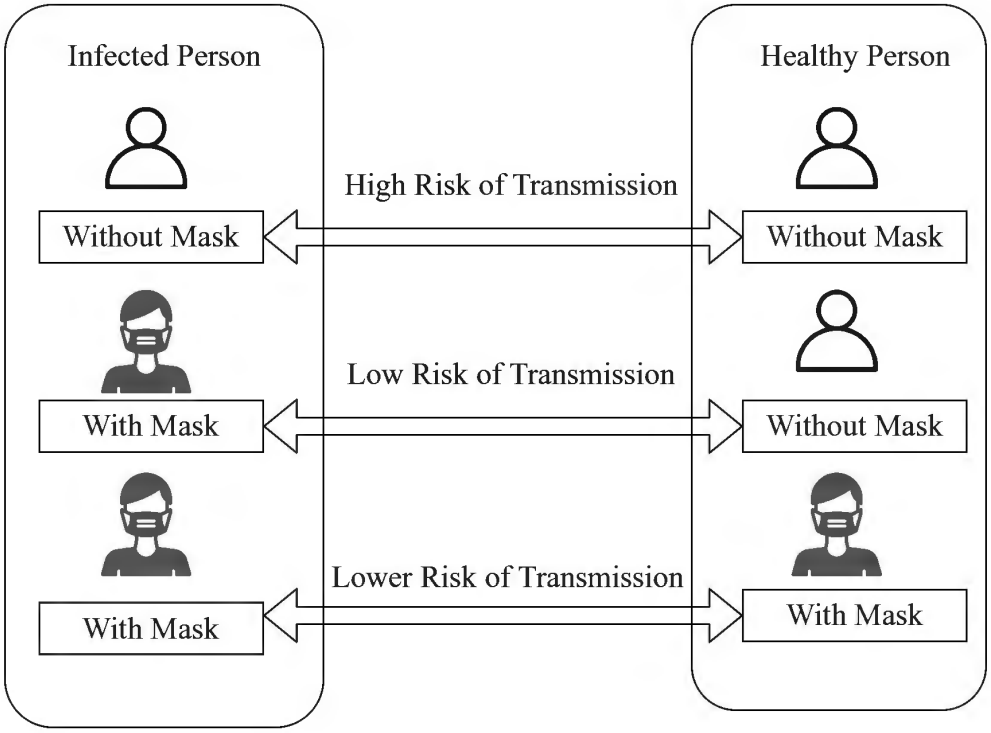


Fig. 4.1. Different risk of transmission between infected person (left column) and uninfected person (right column).

Also it is difficult reminding wearing a mask in public. Face mask detection [69] with deep learning can be used to monitor whether people are wearing masks. Moreover, deployment of this model ensures that IoT devices can be used for this application at real-time in almost most cases.

This section will demonstrate some research experiments of face mask detection using deep learning models on edge devices. This application can detect face mask in public in real-time. For example, some workplaces and businesses are mandating face mask wearing on their premises. A smart IoT camera capable of running DL models can detect and raise alarm if someone is not wearing a mask without human intervention. Mobile GPU such as NVIDIA Jetson devices can perform this tasks using a camera. It saves human resources and reduces cost making it beneficial and efficient for any company.

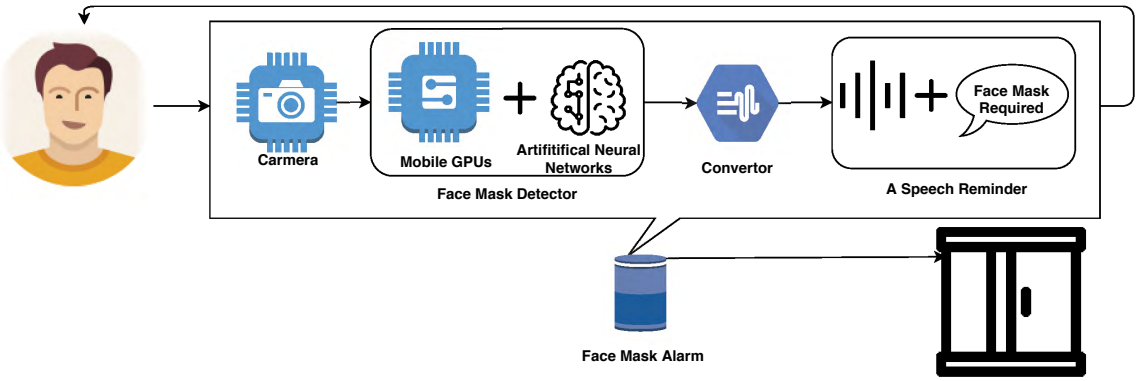


Fig. 4.2. The diagram of a face mask detection system. As an example, the proposed system is mounted on a door to remind people to wear face mask when entering a room. It consists of a camera, a mobile GPU, and an alarm. The image/video captured by the camera will be input to the mobile GPU and the pre-trained CNN will determine whether the person wears face mask. If not, an alarm such as "Face Mask Required" will sound.

Fig. 4.2 shows the diagram of a face mask detection system on controlling the entrance to a room based on the result of face mask detection. The proposed research was to build an algorithm based on deep learning that could run on IoT devices with mobile GPUs, which can be used implement the setup shown in the diagram. We implemented four convolutional neural networks (CNN), namely, MobileNet V2, Inception V3, VGG 16, and ResNet 50. Moreover, these models were verified by running inference on mobile GPUs including NVIDIA Jetson TX2 and NVIDIA Jetson Nano. Specially, we examined the model robustness by training small sizes of data, which is key to real world applications to emerging events such as COVID-19 outbreak, because usually we cannot collect big data for training due to few samples available and high annotation costs. Experimental results demonstrated that these models can achieve promising detection performance on IoT devices and robustly detect face mask even being trained on very small size of data.

4.2 Deep Learning Models

Face mask detection can be viewed as a binary image classification problem [69]. We deployed four popular CNN, namely, MobileNet V2, Inception V3, VGG 16, and ResNet 50 to check their performance. MobileNet is the most popular model for edge device for its lightweight. Smaller model size and less computational expensive were the main themes behind its design making it very suitable for resource constraint devices. One major design is that it breaks up the traditional computation expensive convolution operation into two parts. These are named depthwise and pointwise convolution [70]. The idea is to break up the complex matrix multiplication into two simple matrix multiplication. MobileNet V2 [71] is more advanced version of V1 which has an additional convolution layer and residual connection like ResNet. Width multiplier and resolution multiplier are fine tuned to control the resolution of the input.

Inception V3 [72] is another widely used image classification network for DL applications. The number of connections was reduced using factorized convolution while keeping a strong performance. It also has different version and version 3 is the most used. It is also computationally efficient in terms of older models. Smaller convolution, factorized convolution, auxiliary classifier, lower grid size, factorized convolution are the main techniques used to design this model.

VGG-16 [73] is a 16 layer CNN introduced to observe how a model deals with vanishing gradient problem as depth increases. The vanishing gradient refers to higher training errors observed as layer is increased. However, if a CNN gets deeper, it should be capable to accord with more complex learning. Also, a better performance was expected but the performance decreased rapidly. Performance dropped drastically as learning was saturated. VGG 16 had 13 convolutional layer and 3 fully connected layer. One disadvantage was that it was slow to train and weights were quite large

and 134M parameters needed to be trained.

ResNet model [74] addressed the vanishing gradient problem by applying residual learning into the network. Identity mapping, adds a layer which skips one or two layers in the model. The skipped connection tries to solve a residual function using H(x) = F(x) + x for a few stacked layer instead of conventional mapping. One reason is rather than mapping F(x), it is easier to get F(x) = 0. Moreover stride of two is used to make computation lesser. Also the number of trainable parameters was only 23M, which made it faster than VGG-16. This research used ResNet-50 for this application.

We select transfer learning approach as DL models for image classification tasks tend to perform better when they are trained from an existing popular trained model. Training from scratch does not yield good performance all the time. All the pre-trained models used to classify 1,000 classes from ImageNet [75]. It takes less time and effort to train a new model which can detect these 1,000 classes. It is very popular concept and widely used. The features learned from the previous task were transferred into the new classifier. All the pre-trained models used here are trained on ImageNet. The size of the input image is different. For example, MobileNet, ResNet and VGG-16 used image size $224 \times 224 \times 3$, whereas Inception V3 used a size of $299 \times 299 \times 3$.

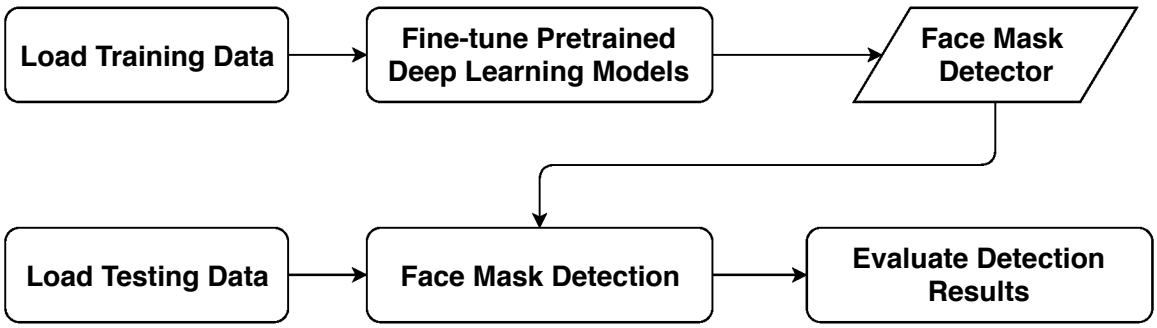


Fig. 4.3. The flow of building face mask detector.

Fig. 4.3 shows the overall plan of this research, which was particularly based on transfer learning. The training was been done on NVIDIA TITAN V GPU, and tested on NVIDIA Jetson devices. The pre-trained model was loaded first. Then the last layer is changed according to research which was a binary classifier. The training was done for separate models and for separate cases. In terms of testing, the model was saved in Keras format. In the edge devices, the model was loaded along with the testing dataset. The performance was evaluated with various evaluation metrics.

4.3 IoT Device with Mobile GPU

In recent times deploying DL models on edge device is getting a lot of attention. The urge is to make the devices smarter which can predict based on learning. Traditional DL models are computationally expensive and require GPUs to train and test. In contrast edge devices have only limited computation resources. Lightweight GPU, mounted on top of an edge device have the ability to run DL models. NVIDIA Jetson devices are types of devices which are widely used for local deployment of AI models. Moreover, edge devices are also getting some computational capacity and DL models are getting lighter which reflects a promising advancement in this area. NVIDIA Jetson TX2 and Nano were selected for this part of research as test devices. We ran the same model with same test data to observe which one performed better. There was a significant difference between these two devices, which is explained in details in table 4.1 and Fig. 4.4 shows the two models.

4.4 Experiment

4.4.1 Dataset

We used a public dataset for this part as this was a relatively new topic and limited dataset were available. This dataset had 1,916 images with face masks and 1,930 images without face masks. However, some data had some different characteristics.

Fig. 4.4. NVIDIA Jetson Nano (left) and NVIDIA Jetson TX2 (right)

Table 4.1. Comparison Between NVIDIA Jetson TX2 and NVIDIA Jetson Nano

	Jetson TX2	**Jetson Nano**
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU	Quad-core ARM Cortex-A57 MPCore processor
GPU	256-core Pascal @1300 MHz	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
Memory	8GB 128-bit LPDDR4	4 GB 64-bit LPDDR4
Storage	32GB eMMC 5.1	16 GB eMMC 5.1

Fig. 4.5. Example images from the datatset. The first row contains example images with face mask, where the first three images have one face with face mask occupying the most part of the image, while the latter three images having multiple faces with mask and people on the background. The second row contains example images without face mask under different background.

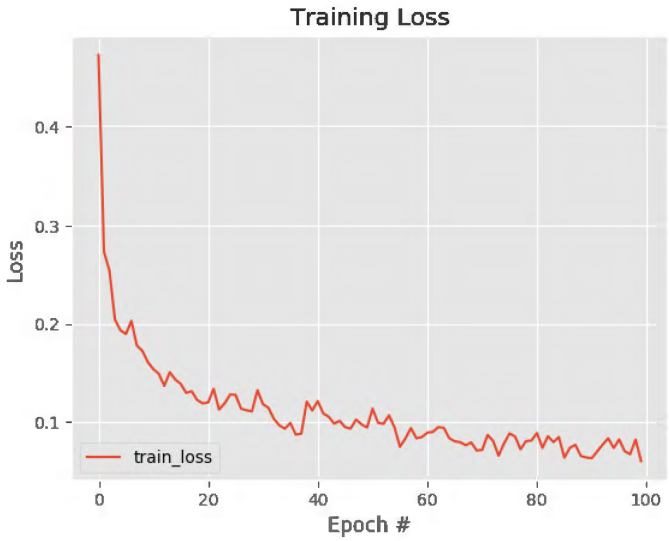


Fig. 4.6. Training Loss over 100 epochs for MobileNet V2

Table 4.2. Comparison of Accuracy and Training Loss of Four Models MobileNet V2, ResNet 50, Inception V3, and VGG 16.

Models	Training Accuracy	Training Loss	Testing Accuracy
MobileNet V2	0.9727	0.0781	0.9149
ResNet 50	0.9927	0.0212	0.9867
Inception V3	0.9666	0.0858	0.9894
VGG 16	0.9987	0.0027	0.9987

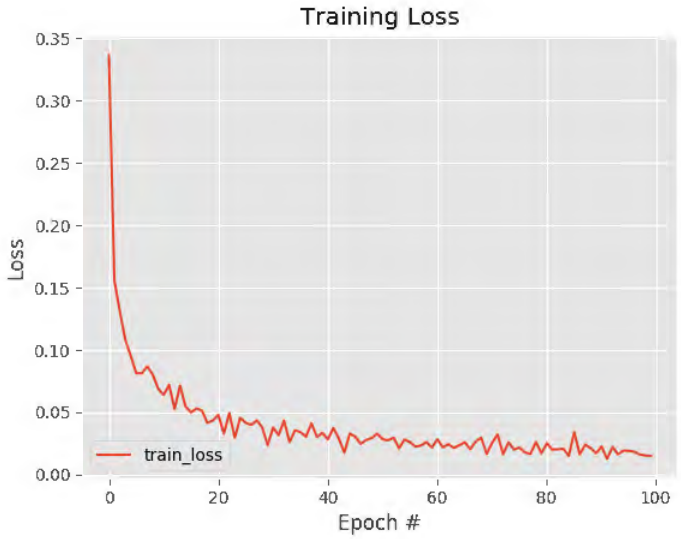


Fig. 4.7. Training Loss over 100 epochs for ResNet 50

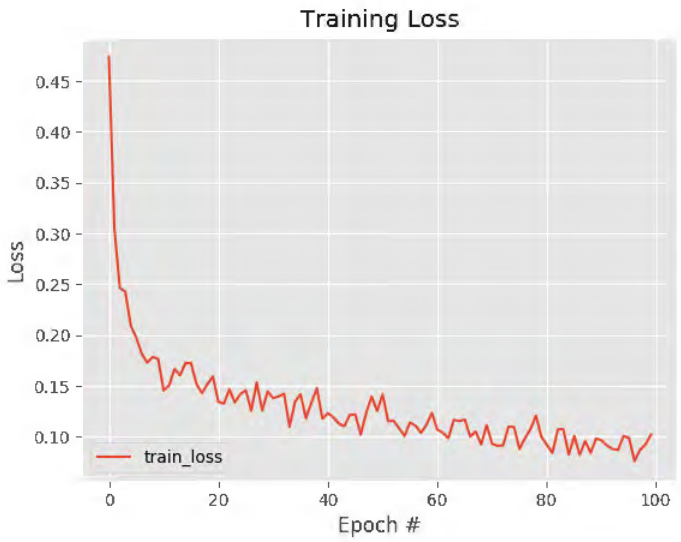


Fig. 4.8. Training Loss over 100 epochs for Inception V3

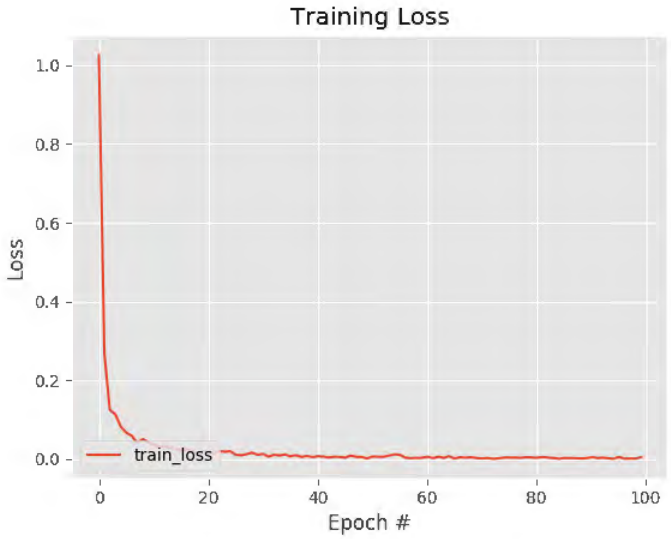


Fig. 4.9. Training Loss over 100 epochs for VGG 16

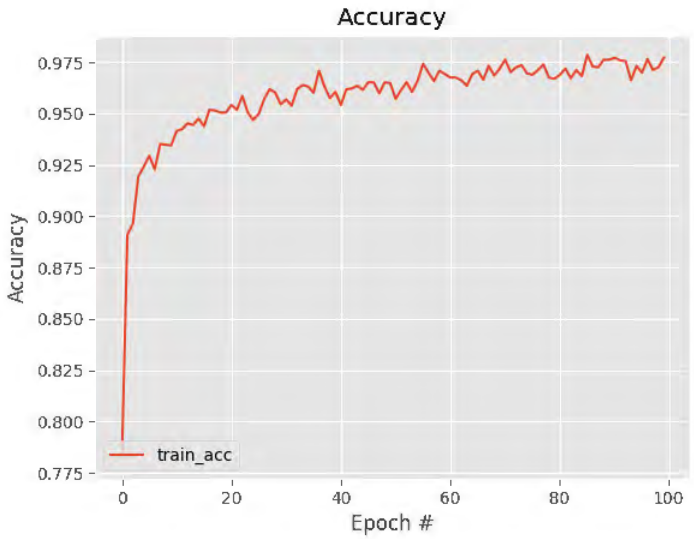


Fig. 4.10. Training Accuracy over 100 epochs for MobileNet V2

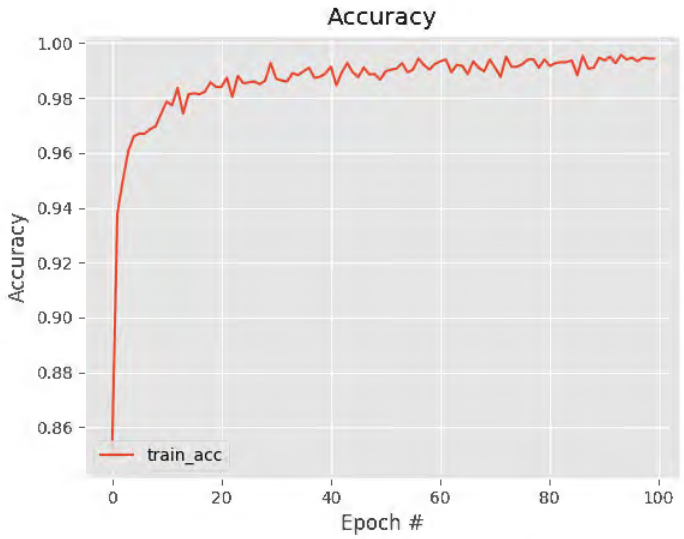


Fig. 4.11. Training Accuracy over 100 epochs for ResNet 50

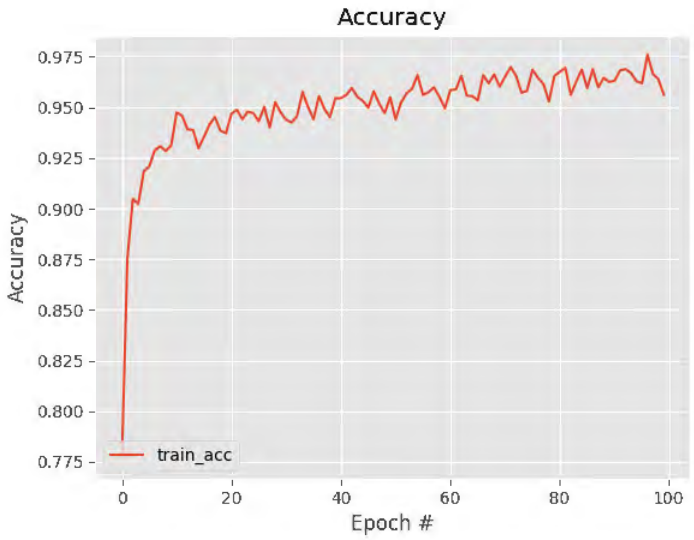


Fig. 4.12. Training Accuracy over 100 epochs for Inception V3

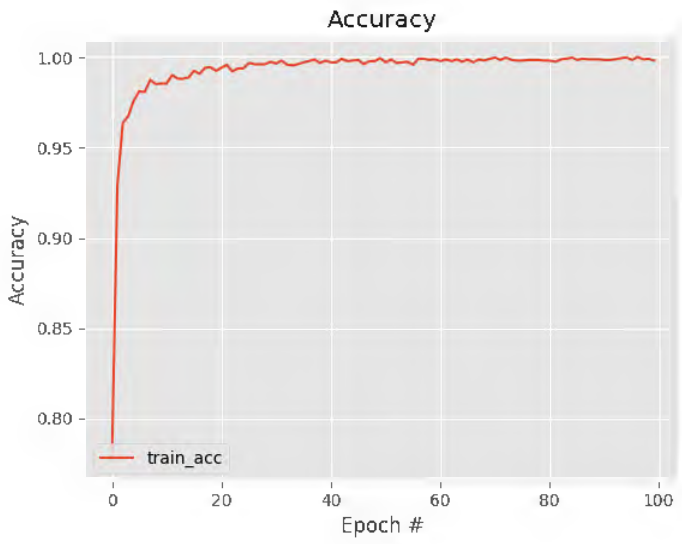


Fig. 4.13. Training Accuracy over 100 epochs for VGG 16

Table 4.3. Comparison of Precision, Recall, and Fscore

Models	Class	Precision	Recall	Fscore
MobileNet V2	With Mask	0.99	0.83	0.90
	Without Mask	0.86	0.99	0.92
ResNet 50	With Mask	1.00	0.96	0.98
	Without Mask	0.96	1.00	0.98
Inception V3	With Mask	1.00	0.98	0.99
	Without Mask	0.98	1.00	0.99
VGG 16	With Mask	1.00	1.00	1.00
	Without Mask	1.00	1.00	1.00

Eighty percent of the data was used for training and the rest was for testing. [1]

4.4.2 Experiment setup

The learning rate was 0.0001 with a batch size of 10. We kept the batch size lower so that it could be trained on limited memory. We kept a standard epoch size of 100. Cross entropy was selected as the loss function as it was a classification task.

4.4.3 Tools for Implementation

Tensorflow [76] which is an open source software was the main software for this section. One benefit of this software was the ability to operate at many processing units such as multicore CPUs, GPUs and Tensor Processing Unit (TPU) [77]. It supported a good number of applications and provided flexibility to design ML model. Moreover Keras [78] API was used, which runs on top of tensorflow. Keras application was used to directly import the pre-trained models. Then the classifier was trained on top of it.

4.4.4 Result Analysis

We implemented face detection with MobileNet V2, VGG 16, Inception V3 and ResNet 50, where different performance is shown in Tables 4.4, 4.2 and 4.3. The results show that VGG 16 outperformed others regarding accuracy, precision, recall, and fscore. The learning curve of this model also showed faster improvement than others. The intuition was that VGG has fixed kernels than others. As the problem was binary and the dataset showed a similar trend, this dense network was able to perform better than other. Table 4.2 reflects one scenario. VGG 16 was able to get the highest training and testing accuracy with lowest loss score. ResNet 50 and Inception V3 had only one percent difference in accuracy than VGG 16. Figure 4.10 and 4.6 shows the learning curves of MobileNet V2 in terms of training accuracy and

[1] https://github.com/chandrikadeb7/Face-Mask-Detection

Table 4.4. Performance comparison on face mask detection on Jetson TX2 and Jetson Nano. FPS refers to the number of images processed per second. The first column denotes the models. M denotes MobileNet V2, R denotes ResNet 50, I denotes Inception V3 and V denotes VGG 16.

	Training				Testing on TX2			Testing on Nano		
	Ratios of training data	Number of training images	Loss	Training accuracy	Testing accuracy	Inference time (ms)	FPS	Testing accuracy	Inference time (ms)	FPS
M	1%	30	0.107	0.9776	0.6957	26.97	37.27	0.6926	42.40	23.58
	5%	150	0.095	0.9733	0.7685	25.10	39.86	0.7659	42.00	23.80
	10%	300	0.114	0.9622	0.7842	25.10	39.87	0.7801	42.90	23.31
	20%	601	0.136	0.9490	0.7735	25.53	39.14	0.7699	43.03	23.24
R	1%	30	0.071	0.9666	0.5000	70.90	14.11	0.7433	195.05	5.13
	5%	150	0.050	0.9845	0.7976	71.00	14.09	0.8023	176.45	5.67
	10%	300	0.035	0.9900	0.7976	70.13	14.25	0.7962	173.05	5.78
	20%	601	0.027	0.9927	0.8651	71.00	14.08	0.8599	173.65	5.76
I	1%	30	0.167	0.9556	0.8275	89.27	11.21	0.8275	187.00	5.35
	5%	150	0.136	0.9578	0.8704	87.80	11.41	0.8713	188.53	5.31
	10%	300	0.136	0.9445	0.9066	92.33	10.85	0.9057	905.72	5.22
	20%	601	0.167	0.9379	0.8972	89.57	11.17	0.8977	192.77	5.19
V	1%	30	0.089	0.9667	0.6857	2138.77	0.47	0.6761	239.37	4.18
	5%	150	0.006	0.9978	0.9231	2139.60	0.47	0.9173	245.23	4.08
	10%	300	0.010	0.9956	0.9370	2136.00	0.47	0.9325	236.17	4.23
	20%	601	0.006	0.9994	0.9607	2139.57	0.47	0.9580	237.70	4.21

loss. VGG 16 was able to reduce the loss drastically within less than 20 epochs. The other three models were also able minimize loss function but VGG showed the most smoothest transition. The learning curve in terms of accuracy also showed that VGG 16 was able to get a good training accuracy in a short time. Also the other models indicated a good training accuracy. Table 4.3 gives further evaluation of the models. VGG 16 showed a perfect score in precision, recall and f-score followed by Inception V3 and ResNet 50. The precision to detect face with mask was done well by all the models.

In addition, we examined how the size of training samples effected the performance. For emerging tasks like face mask detection, it is difficult to obtain a huge dataset for training and testing. Therefore, lacking of data will be a big challenge for similar tasks. We applied different ratios of training data to examine the performance shown in Table 4.4. Experimental results indicated that MobileNet V2 was the fastest model and could process almost 40 images per second in Jetson TX2 while VGG 16 represented the highest accuracy. If we train the model while taking only one percent of the training data and keeping the testing data same, Inception performed better when considering the lowest amount of training data. It had a testing accuracy score of 0.8275. Although the training and testing accuracy difference showed the model was overfitting, it was able to recover as we kept increasing the data. Utilizing only five percent of the training data showed an accuracy of 0.76, 0.79, 0.87 and 0.92 for MobileNet, ResNet, Inception and VGG 16. This showed that the models were robust as they showed good performance on limited data. As we kept increasing the training data to 10 and 20 percent, the performance of all models started improving. VGG showed the maximum accuracy and MobileNet showed the lowest accuracy while considering 20 percent of the data. ResNet and Inception gave close performance in this case. The overfitting problem was addressed as the dataset samples kept increasing.

The table showed slight variation in some cases. For example, MobileNet and Inception performed slight better with 10 percent dataset than 20 percent. Our intuition could be the dataset features because the dataset have some complex images. We saw some exceptions in this Table. For example, ResNet 50 showed an accuracy of around 0.50 and 0.75 in Jetson TX2 and Nano. The difference was huge but the same model and same dataset was used for both cases. In Addition, VGG 16 was nine times faster in Nano than TX2. Hardware specification of individual device could be the reason behind this scenario. Overall VGG 16 showed the best result. One disadvantage was that it is extremely slow and required more high computing resources than others. It reflected the lowest speed in every case. In terms of speed MobileNet V2 showed the better performance as it was designed to perform faster and efficiently. One trade off of this model for being faster was accuracy. It was the reason it showed less accuracy in most cases than others. Jetson TX2 demonstrated faster results than the Nano. Although Nano was relatively close in terms of speed despite having half the memory than TX2.

4.5 Related Work

Face mask detection has been a popular topic due to the recent pandemic situation. This can be treated as either object detection or an image classification task. Object detection methods were used in [79] and [80]. One stage face mask detector was proposed by Mingjie Jiang *et al.* and pre-trained ResNet is used for transfer learning and feature extraction [79]. It detected the mask feature using a novel context attention. Medical face mask detection was created by Loey *et al.* [80] using object detection method. ResNet and YOLO V2 were combined in order to make the mask detector.

This task was also treated as an image classification problem in [81, 82, 83], where pre-trained face detector was used to extract the face and then classification model

was applied to check the mask. Lippert *et al.* presented a face mask detector using OpenCV pre-trained face detector and VGG16 was used to train the model. Madhura Inamdar et al. proposed a face mask detecting model using eight layers which perform the same task [81].

The methods mentioned above were created using DL models and in high performance computers with GPUs. However, this task is difficult if it is to be run in the cloud because bandwidth and security issues. Bandwidth can be unstable and transmission of private data is also a concern. The research showed a way to overcome this problem. The models were run on edge devices and could be used in real-time applications. This does not require any wireless or wired internet connection, and can be deployed almost anywhere. Moreover, we showed how it reacted to different models, devices and training sets.

4.6 Conclusion

This section showed the performance of four deep learning models, namely, MobileNet V2, Inception V3, VGG 16, and ResNet 50 for face mask detection. The testing devices were NVIDIA Jetson TX2 and Nano. The results indicated that MobileNet was the fastest in terms of speed, however the accuracy was low. The other models performed almost the same with good prediction results. The prediction was faster in TX2 than Nano. It was predicted as TX2 had twice the memory than Nano. However, Nano gave a closer performance compared to TX2. We also showed the robustness of the model by reducing the training sample. The intuition was that in real-time data was not always labeled. The future plan is to use semi-supervised learning to process raw data and then build the classifier. Also methods like pruning, quantization and knowledge transfer can be used to make the model lighter and efficient.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This research explored techniques designed for deployment of Deep Learning (DL) models on IoT devices that have limited computing resource and storage. Specifically,

- novel quantization methods have been studied that can be utilized to compress DL models;
- methods that allow DL models to perform efficiently on small amount of data were investigated;
- novel privacy preserving techniques for DL on edge devices were explored.

The effectiveness of the research was demonstrated through multiple experiments such as face mask detection using IoT devices. It gives an overall idea about how in the near future IoT devices will take advantage of DL. NVIDIA Jetson TX2 and Nano were used in the experiments which are popular mobile GPUs for edge devices.

5.2 Future Work

This research holds some promising advancement. The next step is to evaluate energy efficiency of the DL models in different platforms. Energy efficiency is also another major aspect of research for deploying models on edge devices. For example, any edge device that is operating on battery power would require high energy efficiency. In addition, semi-supervised learning will be considered as well when the amount of labeled data is limited, especially in IoT applications.

REFERENCES

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[2] "Deep learning sdk documentation." url= https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html.

[3] "Tensorflow models on the edge tpu." urlhttps://coral.ai/docs/edgetpu/models-intro/#compatibility-overview.

[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[5] L. McRae, K. Ellis, and M. Kent, "Internet of things (iot): education and technology," *Relatsh. between Educ. Technol. students with Disabil. Leanne, Res*, pp. 1–37, 2018.

[6] A. Ukil, S. Bandyoapdhyay, C. Puri, and A. Pal, "Iot healthcare analytics: The importance of anomaly detection," in *2016 IEEE 30th international conference on advanced information networking and applications (AINA)*, pp. 994–997, IEEE, 2016.

[7] J. Kingdon, "Ai fights money laundering," *IEEE Intelligent Systems*, vol. 19, no. 3, pp. 87–89, 2004.

[8] S. Khan, D. Paul, P. Momtahan, and M. Aloqaily, "Artificial intelligence framework for smart city microgrids: State of the art, challenges, and opportunities," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 283–288, IEEE, 2018.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[11] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[12] R. Ramesh, "Predictive analytics for banking user data using aws machine learning cloud service," pp. 210–215, 2017.

[13] D. J. e. a. D'Souza M, Van Munster CEP, "Autoencoder as a new method for maintaining data privacy while analyzing videos of patients with motor dysfunction: Proof-of-concept study.," 2017.

[14] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[15] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *CoRR*, vol. abs/1902.04885, 2019.

[16] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR*, vol. abs/1712.07557, 2017.

[17] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," *CoRR*, vol. abs/1611.04482, 2016.

[18] "Google edge tpu." `https://coral.ai/docs/edgetpu/faq/`.

[19] "Jetson nano developer kit." `https://developer.nvidia.com/embedded/jetsonnano-developer-kit`.

[20] M. Maggipinto, C. Masiero, A. Beghi, and G. A. Susto, "A convolutional autoencoder approach for feature extraction in virtual metrology," *Procedia Manufacturing*, vol. 17, pp. 126–133, 2018. 28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USAGlobal Integration of Intelligent Manufacturing and Smart Industry for Good of Humanity.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[24] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[25] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," 2014.

[26] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, oct 2016.

[27] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference*

on Computer and Communications Security, CCS '17, (New York, NY, USA), p. 619–631, Association for Computing Machinery, 2017.

[28] H. Bae, J. Jang, D. Jung, H. Jang, H. Ha, H. Lee, and S. Yoon, "Security and privacy issues in deep learning," 2018.

[29] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), p. 1310–1321, Association for Computing Machinery, 2015.

[30] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," 2018.

[31] F. S. Beaufays, M. Chen, R. Mathews, and T. Ouyang, "Federated learning of out-of-vocabulary words," 2019.

[32] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," 2016.

[33] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *CoRR*, vol. abs/1812.07478, 2018.

[34] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[35] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," 2019.

[36] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Con-*

ference on Computer and Communications Security, CCS '17, (New York, NY, USA), p. 1175–1191, Association for Computing Machinery, 2017.

[37] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *CoRR*, vol. abs/1705.10467, 2017.

[38] V. Mirjalili, S. Raschka, A. Namboodiri, and A. Ross, "Semi-adversarial networks: Convolutional autoencoders for imparting privacy to face images," 2017.

[39] R. Alguliyev, R. Aliguliyev, and F. Abdullayeva, "Privacy-preserving deep learning algorithm for big personal data analysis," *Journal of Industrial Information Integration*, vol. 15, 07 2019.

[40] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, pp. 2787–2805, Oct. 2010.

[41] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[42] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys and Tutorials*, vol. 21, pp. 2702–2733, thirdquarter 2019.

[43] "Jetson nano: Deep learning inference benchmarks." urlhttps://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks.

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[46] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[47] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

[48] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[49] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 15–24, 2017.

[50] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[51] C. Tai, T. Xiao, Y. Zhang, X. Wang, *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[52] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *arXiv preprint arXiv:1312.5402*, 2013.

[53] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *European conference on computer vision*, pp. 354–370, Springer, 2016.

[54] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *International conference on machine learning*, pp. 597–606, 2015.

[55] K. Yanai, R. Tanno, and K. Okamoto, "Efficient mobile implementation of a cnn-based object recognition system," in *Proceedings of the 24th ACM international conference on Multimedia*, pp. 362–366, 2016.

[56] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: For best speed/accuracy trade-off in neural architecture search," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[57] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders," in *Advances in neural information processing systems*, pp. 2791–2799, 2015.

[58] H. Vanholder, "Efficient inference with tensorrt," 2016.

[59] "Real-time natural language understanding with bert using tensorrt." url= https://devblogs.nvidia.com/nlu-with-tensorrt-bert/.

[60] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2019.

[61] "Internet of things." url= https://cloud.google.com/edge-tpu.

[62] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[63] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[64] "Edge tpu performance benchmarks." url= https://coral.ai/docs/edgetpu/benchmarks/.

[65] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, 2018.

[66] S.-C. Cheng, Y.-C. Chang, Y.-L. F. Chiang, Y.-C. Chien, M. Cheng, C.-H. Yang, C.-H. Huang, and Y.-N. Hsu, "First case of coronavirus disease 2019 (covid-19) pneumonia in taiwan," *Journal of the Formosan Medical Association*, 2020.

[67] T. Jefferson, C. B. Del Mar, L. Dooley, E. Ferroni, L. A. Al-Ansary, G. A. Bawazeer, M. L. Van Driel, S. Nair, M. A. Jones, S. Thorning, *et al.*, "Physical interventions to interrupt or reduce the spread of respiratory viruses," *Cochrane database of systematic reviews*, no. 7, 2011.

[68] A. N. Desai and P. Patel, "Stopping the spread of covid-19," *Jama*, vol. 323, no. 15, pp. 1516–1516, 2020.

[69] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic," *Measurement*, vol. 167, p. 108288, 2020.

[70] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[71] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[72] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[73] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[75] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[76] M. Abadi and A. A. and, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[77] M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pp. 1–1, 2016.

[78] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[79] M. Jiang, X. Fan, and H. Yan, "Retinamask: A face mask detector," *arXiv preprint arXiv:2005.03950*, 2020.

[80] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "Fighting against covid-19: A novel deep learning model based on yolo-v2 with resnet-50 for medical face mask detection," *Sustainable Cities and Society*, p. 102600, 2020.

[81] M. Inamdar and N. Mehendale, "Real-time face mask identification using facemasknet deep learning network," *Available at SSRN 3663305*, 2020.

[82] C. L. et al, "Face mask detector," 07 2020.

[83] G. J. Chowdary, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, "Face mask detection using transfer learning of inceptionv3," 2020.

CURRICULUM VITA

SHEIKH RUFSAN REZA

EDUCATION

PhD. Student in Electrical Engineering, Prairie View A&M University, Prairie View, Texas 77446, USA, Aug 2017 - till date

M.Sc. University of Massachusetts Lowell, USA, May 2016

B.Sc. Electrical and Electronics Engineering, North South University, Bangladesh May 2013

WORK EXPERIENCE

Graduate Research Assistant, ECE Department, Prairie View A&M University, Prairie View, Texas 77446, USA, October 2017 - till date

Graduate Research Assistant, University of Massachusetts Lowell, USA Aug 2015 - Dec 2015

PUBLICATIONS

1. Sheikh Rufsan Reza, Yuzhong Yan, Xishuang Dong and Lijun Qian, "Inference Performance Comparison of Convolutional Neural Networks on Edge Devices", *EAI Edge-IoT 2020.*

2. Sheikh Rufsan Reza, Xishuang Dong and Lijun Qian, "Robust Face Mask Detection using Deep Learning on IoT Devices", *2021 IEEE International Conference on Communications Workshops (ICC Workshops).*

3. Omobayode Fagbohungbe, Sheikh Rufsan Reza, Xishuang Dong and Lijun Qian, "Efficient Privacy Preserving Edge Computing Framework for Image Classification", *IEEE Transactions on Emerging Topics in Computational Intelligence. 2021.*