Classification of Compressed Domain Images Utilizing Open VINO Inference Engine

Tan, Kelvin Sim Zhen, Boezura Borhanuddin, Wong, Yee Wan, Ooi, Thomas Wei Min, Khor, Jeen Ghee

Abstract: ***This paper provides a platform to investigate and explore method of 'partial decoding of JPEG images' for image classification using Convolutional Neural Network (CNN). The inference is targeting to run on computer system with x86 CPU architecture. We aimed to improve the inference speed of classification by just using part of the compressed domain image information for prediction. We will extract and use the 'Discrete Cosine Transform' (DCT) coefficients from compressed domain images to train our models. The trained models are then converted into OpenVINO Intermediate Representation (IR) format for optimization. During inference stage, full decoding is not required as our model only need DCT coefficients which are presented in the process of image partial decoding. Our customized DCT model are able to achieve up to 90% validation and testing accuracy with great competence towards the conventional RGB model. We can also obtain up to 2x times inference speed boost while performing inference on CPU in compressed domain compared with spatial domain employing OpenVINO inference engine.***

Keywords: Discrete Cosine Transform (DCT), Convolutional Neural Network (CNN), Intermediate Representation (IR), Open Visual Inferencing and Neural Network Optimization (OpenVINO)

I. INTRODUCTION

With the advancement of recent computing technologies and artificial intelligence revolution, artificial neural networks [1] are becoming more crucial in our daily life. It represents a computational mathematical model which can be either simple or complex structure formed by group of artificial neurons aimed to solve real world problems [2][3]. It was inspired by the biological neural networks in human body and modelled by using certain activation functions [4]. From time to time, with the trend of computer hardware improvements, machine learning and deep learning have found their path to dive into the daily life of humankind.

Revised Manuscript Received on September 22, 2019.

Tan, Kelvin Sim Zhen, Department of Electronic and Electrical Engineering, University of Nottingham, Malaysia Campus Jalan Broga, 43500 Semenyih, Selangor D.E., Malaysia.

Boezura Borhanuddin, College of Graduate Studies, University Tenaga Nasional (UNITEN) 43000 Kajang, Selangor D.E., Malaysia.

Wong, Yee Wan, Department of Electronic and Electrical Engineering, University of Nottingham, Malaysia Campus Jalan Broga, 43500 Semenyih, Selangor D.E., Malaysia.

Ooi, Thomas Wei Min, Internet of Things Group Intel Technology Sdn. Bhd. Jalan Sultan Azlan Shah, Kawasan Perindustrian Bayan Lepas, 11900 Bayan Lepas, Pulau Penang, Malaysia.

Khor, Jeen Ghee, Department of Electronic and Electrical Engineering, University of Nottingham, Malaysia Campus Jalan Broga, 43500 Semenyih, Selangor D.E., Malaysia.

The beauty of deep learning with manoeuvring deep neural networks [5] is that the network itself is capable to perform feature extraction without human to do it manually.

Deep neural networks emerged over the years with more profound network architectures going deeper in terms of hidden layers. The most well-known deep neural networks in pattern recognition is the convolutional neural network (CNN) [6]. Conventionally, CNN [7] has substantially contributed to the image processing task due to its astounding ability to extract useful feature maps "Fig. 1" [8] and information for performing classification [9] and object detection [10].

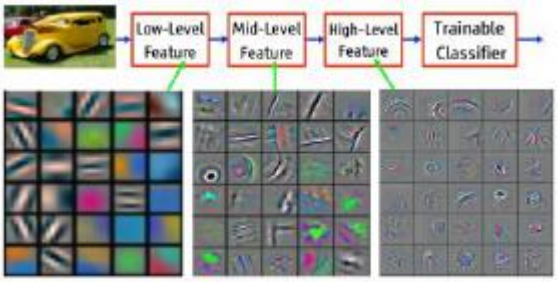

Fig. 1 Feature maps appearance depend on layer of CNN [11]

Many researchers commonly use deep learning to train deep CNN for performing multiple tasks to ease applications in both industry and academia. An example would be using a cascaded CNN to detect traffic signs for transportation purpose [12]. Beside vision recognition, CNN can be applied to audio signals too, such as speech recognition by using CNN to detect the emotional sentiments within conversations between human [13].

For image classification and object detection, these models are usually trained solely based on raw RGB images (24-bits per RGB pixel), which caused the model only accept raw RGB image during inference stage. Each pixel is required to be captured for both training and inference phase to be executed. This method will take substantially longer time to train dataset [14] [15] with larger image database or resolution. Most of the images on internet or in computer today are stored in a compressed format to save space, theseimages are in a 'compressed domain' form. To conduct inference on these images, full decoding or decompression is required for obtaining raw image data, which will cost us extra computational power and time.

To save computational power and improve efficiency, a new idea of training models such to perform inference on compressed domain image data has hit the research arena.

Recent studies and research on compressed domain analytics have becoming more popular [16]. The underlying motivation and mathematical fundamentals for compression algorithm is the Discrete Cosine Transformation (DCT) [17]. 2D DCT-II is used to convert image from spatial domain into frequency domain for better compression [18]. It reduces the spatial redundancy within raw image data to achieve higher amount of compression. Several processes such as quantization and entropy encoding comes after 2D forward DCT to pack the image data into its highest possible compression state.

In this paper, we presented some experimental setup that analyzed the performance difference for inferencing image on devices powered by Intel x86 CPU architecture under spatial domain (RGB) and compressed domain (DCT). The methodology presented here contains minor modifications from the conventional JPEG compression standards [19][20] with several simplifications. The main contributions of this paper covered the following ideas:

Accelerate inference speed by performing classification on compressed domain images using OpenVINO inference engine.

Optimize simple RGB and DCT classification models by converting them into Open VINO ***Intermediate Representation*** (IR) format to run optimally on Intelx86 CPU architecture.

Leverage compressed domain image data to perform inference on normal CPU instead of *Field-Programmable Gate Array* (FPGA) or *Graphic Processing Unit* (GPU).

The remaining of this paper will be structured into four main sections; whereby section II will cover the standard JPEG compression CODEC, some recent compressed domain neural networks and hardware acceleration implementations. Section III will establish the methodology and major pipelines of our work while section IV will provide discussion for analyzing the results and performances of our work.

II. RELATED WORK

A. Fundamentals of Image Compression

Digital media such as image and video on internet tend to grow tremendously over the years. With more and more data required to be stored rapidly, methods of compressing digital media acquire essential priority for efficient storage.

Image compressions pursue to encode the original raw image with lesser bits [21]. It is used to diminish image redundancy such to save or transfer image data in a more effective way. Popular image compression algorithms such as the wavelet and JPEG compression are found to be coherent [22]. We will focus on JPEG image here as the experiment conducted is based on JPEG image data.

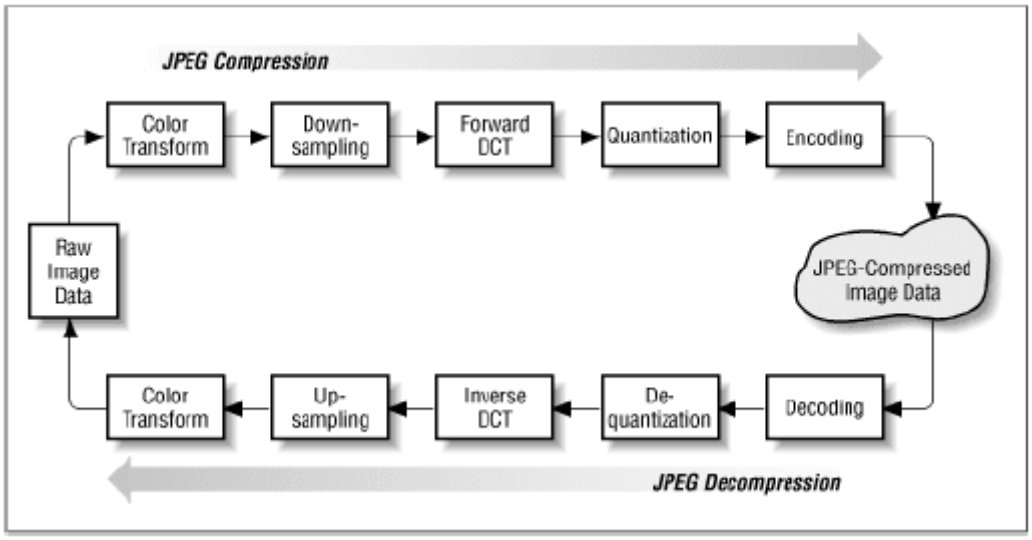

Fig. 2 Standard JPEG image compression and decompression [22]

The JPEG compression algorithm [19], as shown in "Fig. 2", is the most popular image compression standard known today. It started off with partitioning the image into 8x8 Minimum Coded Unit (MCU), extra padding is applied towards the image edge if the image size is not multiplies of 8. Next, color conversion from RGB domain into the YCbCr domain is applied throughout the image for every single pixel. The image will appear to be more compact in YCbCr color space representation. Down sampling sometimes is applied towards the Chrominance channels (CB and Cr) for a more compact format. "Fig. 3" below portrayed the individual channels of YCbCr after converted from RGB image. The image used is 'Lena' of size 512x512 [23].

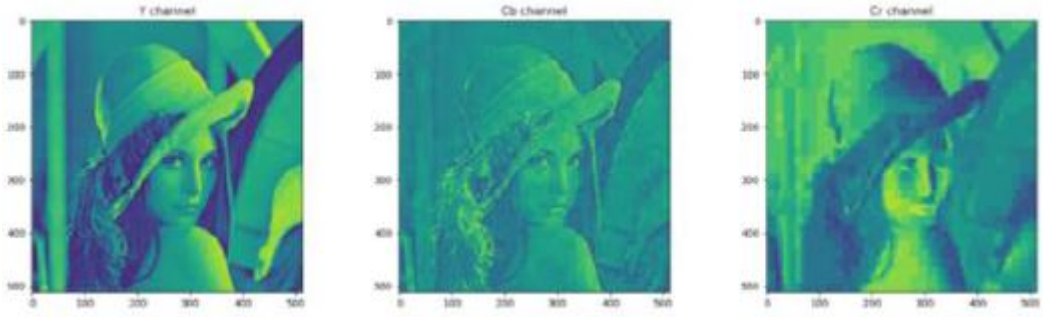

Fig. 3 Image in YCbCr domain for each channels

The upcoming phase includes the most vital process along the compression pipeline which converts the pixel information from spatial to frequency domain. This is done by applying 2D DCT-II [17] towards each of the 8x8 MCU block. 2D DCT-II treats the image as a spatial 2D signal. It takes the advantage of human vision drop-off threshold at higher frequencies by eliminating redundant spatial image information in each MCU blocks. The equation for 2D DCT-II (1) is featured as below with MCU block size of 8.

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos\frac{u\pi}{16}(2x+1)\cos\frac{v\pi}{16}(2y+1)\right] \quad (1)$$

$$\text{where } C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}}, for\ u\ or\ v = 0 \\ 1, otherwise \end{cases}$$

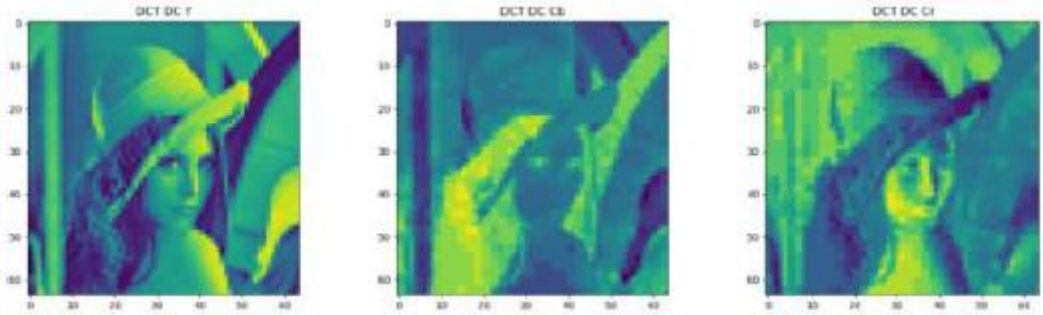

Fig. 4 DC coefficients of DCT image in YCbCr domain

After performing 2D DCT-II towards each of the MCU blocks, each block will result in 64 frequency domain data whereby the most top left data represent the DC coefficient while the remaining 63 data are the AC coefficients. By disregard all the 63 AC coefficients and acquire all the DC coefficient only, we will reduce the original Lena image size from 512x512 to 64x64 (h/8, w/8). "Fig. 4" shows that the DC components of each MCU blocks are sufficient to represent the image detail for the same Lena image.

The subsequent process will further remove higher frequency components within the MCU block by quantizing each of the 8x8 partitions. Quantization matrix of the same size as MCU blocks will be applied towards each MCU. The quantization matrix from quantization tables can be tuned accordingly to trade-off between image quality and compression rate. Standard quantization tables are usually used for simplification and standardization.

The final stage of image compression consists of zigzag encoding and Huffman encoding, which usually summarized as ***entropy encoding***. The zigzag encoding will convert 2D quantized image into 1D array following a special sequential pattern applied to each of the MCU blocks. Huffman encoding [24] further encode each of the MCU blocks into special bytes by referring to their frequency of presence within the image. Data which is often occurring within the image will be encoded with fewer bits and vice versa. It takes certain amount of time to compute the respective Huffman binary tree which is required to generate the relative Huffman Table so sometimes standardized Huffman Tables are used so save time in image compression.

Finally, encoded binary strings are converted and saved as bytes with *'.jpg'* file extension. It includes all the format of a standard JFIF file with special headers and markers [25]. The additional JFIF annotations are handled by specific library such as the ***'libjpeg'*** [26]. For decompression of JPEG images, it will be exactly the reverse method of compression pipeline following the sequence as shown in "Fig. 2".

The issues with encoding and decoding compressed images are time consuming and power inefficient. For the purpose of better storage and transfer rate, image compression is effective. But for usage in deep learning tasks, we hope to eliminate, or minimize the portion of image compression procedures, and try to use the information in between image encoding and decoding for classification or object detection.

B. Compressed Domain in Neural Networks

Over the past few years, popular deep neural networks such as VGG-Net [27], Squeeze-Net [28] and Res-Net [29] have celebrated huge implementation for its overwhelming achievement in image classification. Visual Geometry Group(VGG) kick-started by going deeper in convolution with using very small filter size configurations. The authors in [27] came out with several deep network designs with distinctive blocks of convolution depth to enhance the model performance. With up to millions of trainable parameters and large model size, the drawback is that the model will take a long time to train and only be able to use on higher end computer system. Besides, with deeper layers of convolutional blocks, the respective training and testing error will be higher. Deep residual learning established 'Residual Network' (Res-Net) [29] by computing a residual functions with reference towards the input layers. This novel method allows deep neural networks to go deeper up to 152 layers, which is 8x deeper than VGG-Nets while achieving a low error of 3.57% on the Image Net [30] test set. On the contrary, Res-Net is also a huge model with more parameters which takes time to train.

A so called 'Squeeze-Net' [28] architecture was modified from Alex-Net [31] with lesser parameters and smaller model size. It is easier to train and deploy towards end user. Mobile-Net [32] is also a similar small architecture which extends the model ability from image classification towards object detection. It is specially designed to fit in mobile devices for different applications.

Since much of the deep neural network architectures are trained based on 24-bit raw image data, could we possibly train these models with compressed or partially decoded image data? As more compressed domain images are presented in real world situation compared with spatial domain images, more researches are conducted based on compressed domain deep learning framework to accelerate the deep learning process.

Dan Fu and Gabriel Guimaraes [33] introduced a method of *'DCT truncation'* into the model training pipeline. By applying 2D DCT-II towards a spatial domain RGB image and modifying the image size such to mimic the compressed domain image data, the pre-processed image is feed into the network and train as usual. Similar approach was presented in [34] whereby models are trained based on DCT coefficients of MNIST [35] and CIFAR-10 datasets. This method was experimented on classification task by using very basic CNN on compressed domain images by applying DCT towards raw images yield a competitive result, whereby partial decoding from the compressed image (JPEG) is only needed during the inference stage. Both of the methods does not consider some of the JPEG CODEC procedures, in essence the Chroma down-sampling and quantization.

The author in [36] demonstrated using stacked DCT based sparse auto-encoders architecture for designing the model. The tradeoff point is taken between number of DCT coefficients selected, accuracy and training time. By performing experiment on the MNIST [35] datasets, the training time for DCT domain is 4 times faster than the spatial domain. Another paper [37] trained CNN (modified ResNet-50) straight from the block-wise DCT coefficients available during the image compression stage and achieve similar accuracy with a speed of 1.77x faster than the original model. Similar papers [38] [39] also exhibit different approaches for handling compressed domain images straight from CNN.

We came across lots of methods and models featuring deep learning in compressed domain, but mostly contains simplified or modified JPEG compression standard.

C. Implementation on Hardware Acceleration

Some of the recent works also focused on utilizing hardware accelerator such as *Field-Programmable Gate Array* (FPGA) and *Graphic Processing Unit* (GPU) to run machine learning algorithms. Zhao et. al [40] presented an FPGA architecture design based on CNN and Support Vector Machine (SVM) algorithms. The hardware design workflow is well suitable to run on the above two algorithms as well as other models. Similar work have been done in [41] to design CNN accelerator. Ardestaniet. al [42] showed by using hardware units on site computing with ISAAC and Newton architecture, it can perform better than digital accelerators. FPGA based accelerator is cheap when comparing with GPU, whereby FPGA is also power efficient and flexible in design.

As edge inferencing [43] is as important as model training, research has also fond to reduce the latency of inference time locally rather than connecting the edge devices towards the cloud server for inferencing. To enable model to run efficiently on the edge device such as an embedded device, model compression [44] is used to reduce model size and complexity. Vanhouckeet. al [45] demonstrated some methods to cut down computational power of neural networks inference on x86 architecture CPU.

FPGA is hard to design and implement, without specific knowledge, we were unable to get the most out of it. GPUs are expensive and power intensive equipment. It may provide us higher performance, but some edge devices will be unable to support GPU. Since FPGA and GPU are out of the choices, it left us with CPU whereas it exists in most of the computer system devices today!

III. METHODOLOGY

The objective of conducting this experiment is to make use of compressed domain techniques in image compression to improve the performance of image classification on CPU without the use of GPUs or FPGAs. Our method, namely 'Partial Decoding of JPEG Images' is to obtain nominal DCT coefficients from encoded image streams to perform classification. The whole pipeline will cover the following procedures:

- Import trained models (under both RGB and DCT domain).
- Read in images from specific directory or webcam to obtain encoded JPEG images stream.
- Partial decoding of JPEG compressed image streams to obtain 'Nominal DCT coefficients'.

- Standardization of 'Nominal DCT coefficients'.
- Feed the DCT coefficients into a trained DCT CNN model converted into OpenVINO IR format.
- Perform inference utilizing OpenVINO inference engine.

The difference between this method with the conventional one is that this method does not require JPEG image full decoding to obtain the spatial domain information (RGB) of image for inference. This is because the 'Nominal DCT coefficients' from JPEG image is sufficient to provide useful information for the neural network to recognize specific patterns or feature maps. Without the need of decompressingthe whole image, inference speed of classification is expected to increase when comparing with spatial domain (RGB) images.

Two different pipelines are shown in "Fig. 5" as below are being tested and explored in this paper.

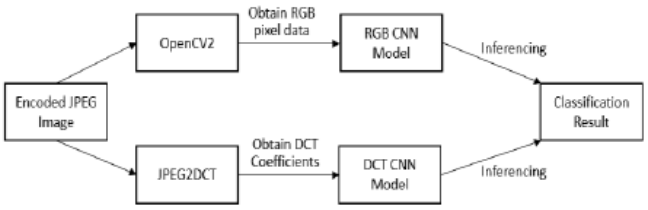

Fig. 5 RGB and DCT inferencing using different pipelines

In our experiment, we use 'OpenCV2' library to perform full decoding towards the JPEG compressed images and obtain the RGB pixel data for inference. While for the DCT path, we were using a module name ***'jpeg2dct'*** from the paper [37] to obtain our DCT coefficients. The RGB pixel and DCT coefficients are then feed towards their respective CNN model for inference and the consecutive classification results are computed.

Initially, the dataset is collected and pre-processed to train the aforementioned two models on a computer system and deployed as OpenVINO Intermediate Representation (IR) format. The models will be exported from the computer to serve as a more optimized model for inference later.

A. CNN Model Architecture for different domains

Two simple CNN models that will be used for classification in our experiment is established in Table I and Table II as below. These models are adapted and modified from the paper [34]. Simple CNN models are used as the datasets in the experiment later does not require thick convolution layers. The input layer of the RGB model is constructed from a simple CNN layer with default size of 224x224 while the DCT model contained customized input channels to fit the DCT coefficients of different down sampling ratio.

Table. I RGB CNN Model

Layer Name	Kernel Type [f, s]	Output Size
Conv_2D_1	[3x3, 1x1]	224x224, 16
MaxPooling_2D_1	[3x3, 2x2]	112x112, 16
Dropout_1	p = 0.25	112x112, 16
Conv_2D_2	[3x3, 1x1]	112x112, 32
MaxPooling_2D_2	[3x3, 2x2]	56x56, 32
Dropout_2	p = 0.25	56x56, 32
Conv_2D_3	[3x3, 1x1]	56x56, 64
MaxPooling_2D_3	[3x3, 2x2]	28x28, 64
Dropout_3	p = 0.25	28x28, 64
Conv_2D_4	[3x3, 1x1]	28x28, 128
MaxPooling_2D_4	[3x3, 2x2]	14x14, 128
Dropout_4	P=0.25	14x14, 128
Flatten	-	1, 25088
Dropout_5	p = 0.50	-
Softmax	-	1, class_number

Table. II DCT CNN Model

Layer Name	Kernel Type [f, s]	Output Size	Inherited From
DCT_Cb (Input)	-	14x14, 64	-
DCT_Cr (Input)	-	14x14, 64	-
DCT_Y (Input)	-	28x28, 64	-
UpSampling2D_cb	[2x2, 2x2]	28x28, 64	DCT_Cb
UpSampling2D_cr	[2x2, 2x2]	28x28, 64	DCT_Cr
Concat2D	-	28x28, 192	DCT_Y, UpSampling2D_cb UpSampling2D_cr

Conv2D_1a	[3x3, 1x1]	28x28, 32	Concat2D
Conv2D_1b	[3x3, 1x1]	28x28, 32	Conv2D_1a
MaxPooling2D_1	[2x2, 2x2]	14x14, 32	Conv2D_1b
Dropout_1	P=0.35	14x14, 32	MaxPooling2D_1
Conv2D_2a	[3x3, 1x1]	14x14, 64	Dropout_1
Conv2D_2b	[3x3, 1x1]	14x14, 64	Conv2D_2a
MaxPooling2D_2	[2x2, 2x2]	7x7, 64	Conv2D_2b
Dropout_2	P=0.5	7x7, 64	MaxPooling2D_2
Flatten	-	1, 3136	Dropout_2
Softmax	-	1, class_number	Flatten

The difference between the two models above as shown in Table I and Table II is that the RGB model contains ***four*** CNN blocks (***1x*** *Convolution, 1x Max-Pooling, 1x Dropout*) while the DCT model only have ***two*** CNN blocks (***2x*** *Convolution, 1x Max-Pooling, 1x Dropout*). The individual mentioned blocks for RGB and DCT models are extracted and shown in "Fig. 6" below for better clarity. The character *'f'* stands for convolution filter sliding window size, while *'s'* represents the stride.

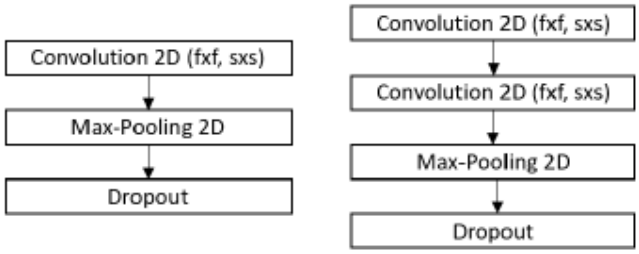

Fig. 6 RGB CNN block (left); DCT CNN block (right)

We design the DCT model to contain *two* convolution layers in each block, whereby the RGB model only contain *one* convolution layer in each block. The reason is because DCT domain image contain 64 coefficients of three channels resulting in 192 image data-like array in depth after concatenating. With only one layer of convolution will not be enough to acquire essential feature maps. Therefore two convolution layers were used in DCT model. The design of number of layers for DCT model is also by experimenting number of neurons and the respective inference speed.

Number of parameters, training time and accuracy of the aforementioned models will be evaluated in the subsequent section. Trained models will be initially saved as Tensor flow standard proto-buff (.pb) format and later exported into OpenVINO IR format, with the following settings listed as below:

Table. III Model Conversion Settings

Conversion Setup	DCT Model	RGB Model
Scaling	*512.0*	*255.0*
Batch Size	1	1
Data Type	FP32	FP32
Reverse Input Channels	True	True

From Table III above, only the scaling setup is different while the other settings remain the same. Explanation for different scale value towards both models will be covered in the subsequent section. Batch size is set to 1 such that the model only takes in single image at a time for inferencing. Data type of FP32 stands for 32-bits floating point precision standard. While parsing 'True' value towards the reverse input channels, the input will be altered from '*height x width x channels*' (***H x W x C***) into '*channels x width x height*' (***C x W x H***).

B. Dataset Preparation

To assess the performance of our methods, three datasets are used in our experiment, i.e. Malaria [46], Pneumonia [47] and Natural Images [48]. These three datasets were obtained from official Kaggle website. Malaria is a hazardous symptom transmitted by *Anopheles* mosquito. It usually parasites in the red blood cells in the human body and replicate in a very short time span, causing the cell to explode. Pneumonia is the human lung infection caused by virus or bacteria. Therefore it is equally important to perform classification task on these datasets for detection such an early cure can be carried out.

Malaria and pneumonia dataset consisted of 2 classes while Natural Images dataset consisted of 8 classes (airplane, car, cat, dog, flower, fruits, bike, and person). These dataset were split into training, validation and testing by a ratio of 7:2:1. Number of images per class after splitting are shown in Table IV below in brief.

Table. IV Number of Images per Class after Splitting

Dataset	Training Images	Validation Images	Testing Images
Malaria	2800	800	400
Pneumonia	1050	300	150
Natural Images	490	140	70

After the dataset is downloaded, a series of pre-processing algorithm is applied towards all the images. Image augmentation is not carried out for simplicity. Our main concern is able to perform inference on compressed domain images, more intuitively the standard JPEG image. Hence, we only conduct a few initial steps following the partial decoding, without applying pruning.

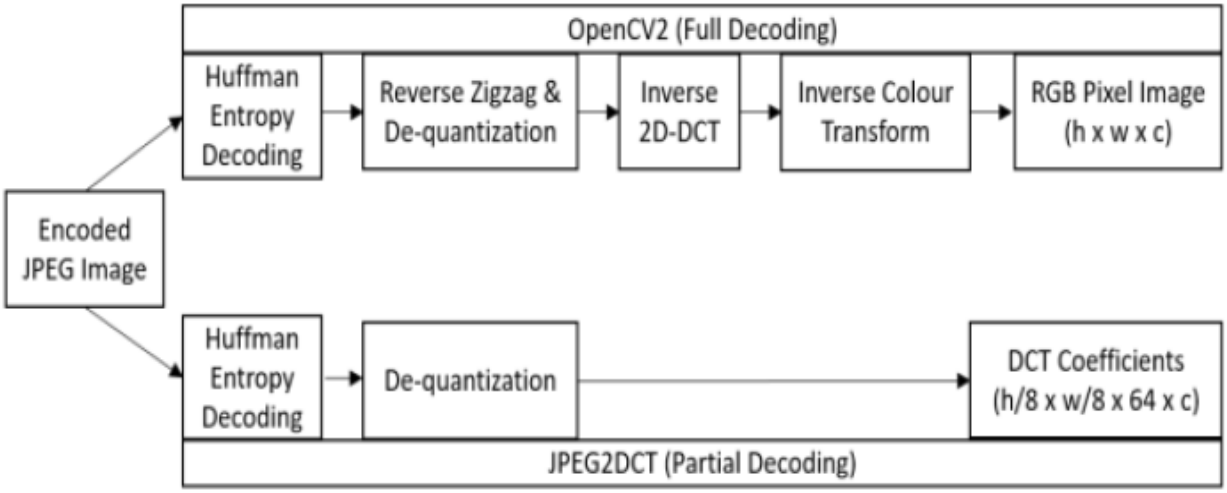

Fig. 7 Comparison between partial and full decoding of JPEG image

From "Fig. 7" as illustrated above, we can clearly differentiate between full and partial decoding using different Python module to obtain the final image data required for training or inferencing. For an RGB original image shape of 224x224x3, the three DCT coefficients (include one Luminance channel and two Chrominance channels) obtained will be [28x28x64], [14x14x64], [14x14x64] respectively. The compressed image is subjected to a down sampling ratio of 4:2:0.

After we obtain the above three channels of DCT coefficients, the data range is in between -1024 to +1024. Standardization is applied towards the dataset by dividing the DCT image array by 512. This narrows the dataset range into -2 to +2. Normalization is found to produce worse result compared with standardization as the accuracy is lower. Besides, with wider range of sample available, details of the coefficients can be captured by the network more easily to create related feature maps.

For RGB domain data, the original range of 0 to +255 is normalized by dividing 255 to position the range in between 0 to +1. After pre-processing of DCT and RGB domain data, they are saved into Python Numpy array under different directory.

IV. EXPERIMENTAL RESULT AND DISCUSSION

A. Experimental setup

We have a system running on a CPU of Intel Core i7-8750H, 6 cores with frequency of 2.2GHz to 4.1GHz. The RAM is 64GB with 500GB of Solid-State Drive. The experiment is built on top of Ubuntu 16.04LTS with OpenVINO 2019 R2 version. The computing language of the whole experiment is based on Python script. Therefore theresults in this experiment shall not be compared with other experiments with different computing language domain as discrepancies may present. Keras (version 2.2.4-tf) of Tensor flow framework (version 1.13.1) is used for creating our CNN models and evaluate its performance.

On any system or computer powered by Intel CPU whereby OpenVINO inference engine can support, the IR models can be deployed onto that system with ease. Encoded images stream in either real time provided by M-JPEG webcam or pre-saved images can be loaded from storage for inference. In our experiment, we setup a directory to contain testing JPEG images and do prediction based on them.

B. Evaluation on inference time

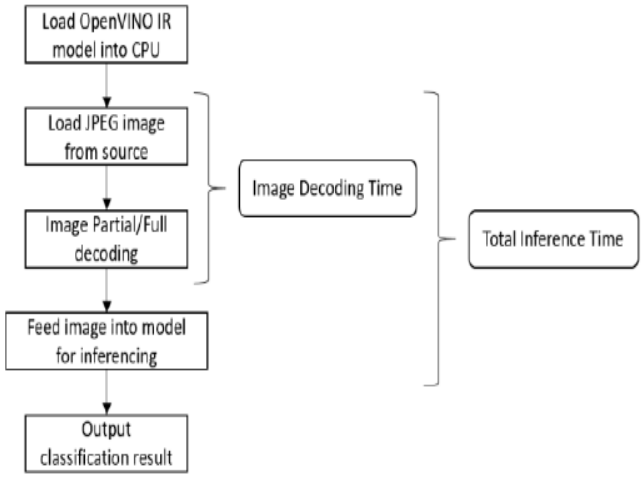

Fig. 8 Inference time track

The inferencing experiment is done with looping through all testing images saved in inference directory. Inference process includes loading the OpenVINO IR model, loading image to perform partial or full decoding with accordance towards the domain and finally perform prediction (refer to Fig. 5). The inference time is calculated based on loading image, decoding image and finally feeding the image into the model for inference. Procedures where time is tracked are shown in "Fig. 8" below for clarity.

To show the practical inference process, Open CV Python library is used to display the inferred image on two separate frames comparing RGB and DCT domain inferencing. Respective inference result, image decoding FPS and total inferencing FPS will be shown on the screen. For demo purpose, each image is delayed for around 0.1 second. The actual test for benchmarking the inference FPS later will not contain any delay.

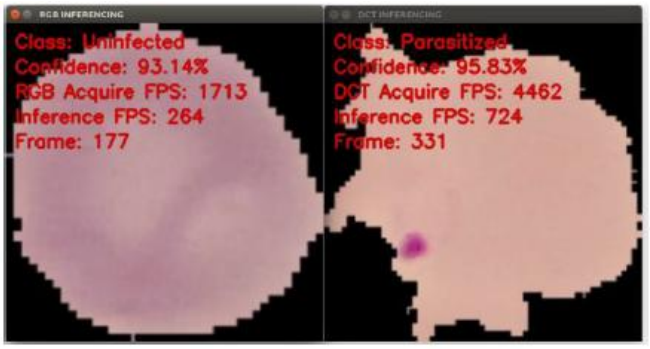

Fig. 9 Malaria inferencing (left – RGB, right – DCT)

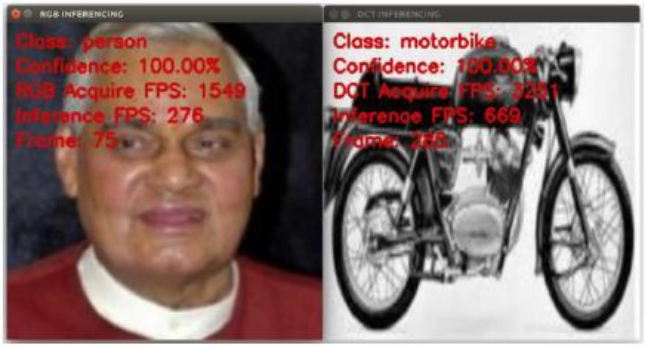

Fig. 10 Natural images inferencing (*left – RGB, right – DCT*)

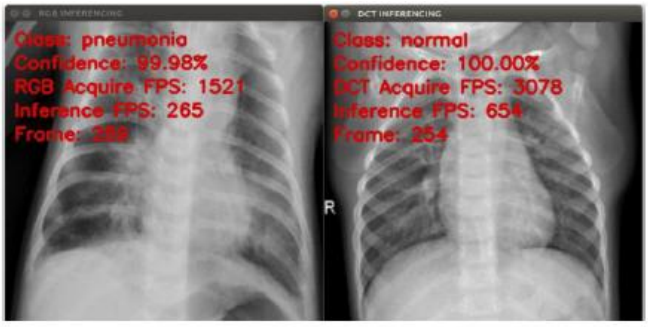

Fig. 11 Pneumonia inferencing (*left – RGB, right – DCT*)

The inference process is run on OpenVINO inference engine and some performance comparison is done between full decoding (OpenCV2) and partial decoding (jpeg2dct). The performance of frame rate (Frame Per Second, FPS) as recorded below are compiled based on an average taken out of 5 runs of the whole inference process. The inference time tracked for single image is from loading the image until feeding the image into the executable OpenVINO IR model. Each image will be iterated 1000 times to get the average frame rate. All images have a standard size of 224x224x3.

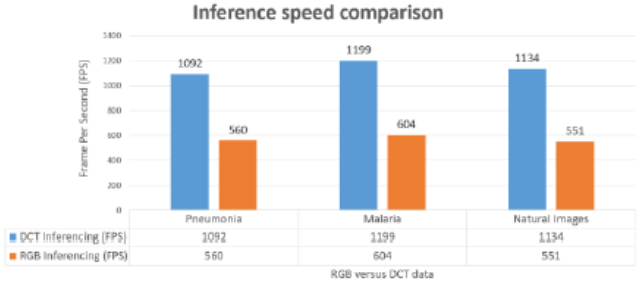

Fig. 12 Inference speed comparison between RGB and DCT domain

Table V. Model Conversion Settings

Dataset	Inference Speed Improvement
Pneumonia	1.95x
Malaria	1.98x
Natural Images	2.06x

From Table V above, we can clearly see that the inference speed (FPS) for partial decoding is higher than full decoding. The inference frame rate of compressed domain images is about ***two times*** greater than the spatial domain images. In other words, this means that more computational power is saved in return for 100% faster in inference speed. The inference speed (FPS) difference between "Fig. 12" and the figures shown above (Fig. 9, Fig. 10, Fig. 11) is the use of slight delay during the display for us to visualize the detected inference result.

C. CNN model properties and performances

In this section, we will discuss about the performances of the CNN models and assess their difference based on varying input domain of dataset – spatial domain (RGB) and compressed domain (DCT). Table VI below summarized the performance of the two models based on different datasets of different domain.

Table VI. CNN Model Performance

Model	RGB CNN Model	DCT CNN Model
Pneumonia		
Validation Accuracy (%)	95.57	**97.47**
Testing Accuracy (%)	**95.25**	93.67
Trainable Parameters (mil)	0.15	**0.13**
Malaria		
Validation Accuracy (%)	**94.56**	94.38
Testing Accuracy (%)	**93.00**	90.25
Trainable Parameters (mil)	0.15	**0.13**
Natural Images		
Validation Accuracy (%)	90.45	**91.07**
Testing Accuracy (%)	**90.00**	89.82
Trainable Parameters (mil)	0.3	**0.15**

From Table VI above, the testing accuracy difference between spatial and compressed domain is within 5%. Compressed domain models exhibit similar or higher validation and testing accuracy compared with spatial domain. The minor difference between model trainable parameters of RGB and DCT CNN models shall not affect the inference speed and model accuracy as the effect is minimal.

V.CONCLUSION AND FURTHER WORK

Along the experiment presented above, we can clearly visualize the application pipeline along the whole IoT industry from capturing input of image streams towards obtaining the classification result. With our method of training the neural networks in the compressed domain and inferencing done on partially decompressed image, we achieve averagely ***two times faster in inference speed*** when comparing with the RGB data retrieved from compressed domain images.

The benefits of performing classification on compressed domain images include saving time and computational power yet provide a faster inference speed towards real time application. It also improve the CPU inference speed by solving hardware deficiency using software tweaks. The only drawback of having this task done on compressed domain (DCT) is sometimes lower accuracy is obtained compared with spatial domain (RGB). The accuracy of the model is actually depends on the chosen model and dataset variations. With the continuous advancement of technology and design of new neural network architectures, we believe that the accuracy of such task is not a critical issue from time to time. The accuracy can be improved by feeding the network with more dataset and conduct image augmentation for acquiring a more robust network.

In a nutshell, analytics and network models of compressed domain are such an essential contribution towards a smarter computational intelligence for our digital world tomorrow.

ACKNOWLEDGMENT

This work is supported by Internet of Things Group (IOTG) – Intel Labs for Researchers, under Intel Malaysia. Profound guidance is provided by my PhD supervisor Ir. Dr. JG Khor and Dr Wong Yee Wan from the University of Nottingham Malaysia Campus, and my manager from Intel – Dr Thomas Ooi Wei Min.

REFERENCES

1. E. Parveen Kumar and E. Pooja Sharma, “Artificial Neural Networks-A Study,” *Int. J. Emerg. Eng. Res. Technol.*, vol. 2, no. 2, pp. 143–148, 2014.
2. E. Y. Li, “Artificial neural networks and their business applications,” vol. 27, pp. 303–313, 1994.
3. K. Kumar, G. Sundar, and M. Thakur, “Advanced Applications of Neural Networks and Artificial Intelligence: A Review,” no. May, 2012.
4. C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning,” pp. 1–20.
5. N. Kriegeskorte, “Deep neural networks: a new framework for modelling biological vision and brain information processing,” 2015.
6. S. Albawi and T. A. Mohammed, “Understanding of a Convolutional Neural Network,” *2017 Int. Conf. Eng. Technol.*, pp. 1–6, 2017.
7. K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” no. November, 2015.
8. M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” pp. 818–833, 2014.
9. E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, “Fully convolutional neural networks for remote sensing image classification,” *Int. Geosci. Remote Sens. Symp.*, vol. 2016-Novem, pp. 5071–5074, 2016.
10. M. Lokanath, K. S. Kumar, and E. S. Keerthi, “Accurate object classification and detection by faster-RCNN,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 263, no. 5, 2017.
11. S. Pöcheim, “Convolutional Neural Networks,” 2017. [Online]. Available: https://wiki.tum.de/display/lfdv/Convolutional+Neural+Networks. [Accessed: 11-Aug-2019].
12. D. Zang, J. Zhang, D. Zhang, M. Bao, J. Cheng, and K. Tang, “Traffic sign detection based on cascaded convolutional neural networks,” *2016 IEEE/ACIS 17th Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel/Distributed Comput. SNPD 2016*, pp. 201–206, 2016.
13. P. Tzirakis, J. Zhang, and B. W. Schuller, “End-to-end speech emotion recognition using deep neural networks,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2018-April, pp. 5089–5093, 2018.
14. M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes Challenge: A Retrospective,” *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, 2014.
15. T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014.
16. R. V. Babu, M. Tom, and P. Wadekar, “A survey on compressed domain video analysis techniques,” *Multimed. Tools Appl.*, vol. 75, no. 2, pp. 1043–1078, 2016.
17. N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete Cosine Transform,” *IEEE Trans. Comput.*, vol. C–23, no. 1, pp. 90–93, 1974.
18. X. Ji, C. Zhang, J. Wang, and S. H. Boey, “Fast 2-D 8×8 discrete cosine transform algorithm for image coding,” *Sci. China, Ser. F Inf. Sci.*, vol. 52, no. 2, pp. 215–225, 2009.
19. G. K. Wallace, “Wallace.JPEG,” *JPEG Still Pict. Compression Stand.*, pp. 1–17, 1991.
20. N. Kashyap, “JPEG Image Code Format,” pp. 1–21.
21. C. Science and S. Engineering, “Image Compression Technique under JPEG by Wavelets Transformation,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 4, no. 6, pp. 808–818, 2014.
22. S. Rawat and A. K. Verma, “Survey paper on image compression techniques,” *Int. Res. J. Eng. Technol.*, vol. 4, no. 3, pp. 1–6, 2017.
23. P. Meerwald, “Lena Image.”.
24. S. L. Bawa and D. A. V College, “Compression Using Huffman Coding Mamta Sharma,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 10, no. 5, p. 133, 2010.
25. E. Hamilton, “JPEG File Interchange Format,” *Interchange*, vol. 81, pp. 467–490, 2004.
26. SourceForge, “libjpeg.” [Online]. Available: http://libjpeg.sourceforge.net/. [Accessed: 12-Aug-2019].
27. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” pp. 1–14, 2014.
28. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” pp. 1–13, 2016.
29. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2015.
30. J. D. J. Deng, W. D. W. Dong, R. Socher, L.-J. L. L.-J. Li, K. L. K. Li, and L. F.-F. L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 2–9, 2009.
31. B. Monien, R. Preis, and S. Schamberger, “ImageNet Classification with Deep Convolutional Neural,” *Handb. Approx. Algorithms Metaheuristics*, pp. 60-1-60–16, 2007.
32. A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.
33. D. Fu and G. Guimaraes, “Using Compression to Speed Up Image Classification in Artificial Neural Networks Background: The Discrete Cosine Transform,” pp. 1–10, 2016.
34. M. Ulicny and R. Dahyot, “On using CNN with DCT based Image Data,” *Proc. 19th Irish Mach. Vis. Image Process. Conf.*, pp. 44–51, 2017.
35. F. Chen, N. Chen, H. Mao, and H. Hu, “Assessing four Neural Networks on Handwritten Digit Recognition Dataset (MNIST),” no. June, pp. 1–4, 2018.
36. X. C. Q. X. X. Zou, Xiaoyi; Xu, “HIGH SPEED DEEP NETWORKS BASED ON DISCRETE COSINE TRANSFORMATION,” *Int. Conf. Image Process.*, pp. 5921–5925, 2014.

37. L. Gueguen, A. Sergeev, R. Liu, and J. Yosinski, "Faster Neural Networks Straight from JPEG," *Neural Inf. Process. Syst.*, vol. 2, no. Nips, pp. 5–8, 2018.
38. Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "CNNpack: Packing Convolutional Neural Networks in the Frequency Domain," *Nips*, no. Nips, pp. 421–434, 2016.
39. B. Li, H. Luo, H. Zhang, S. Tan, and Z. Ji, "A multi-branch convolutional neural network for detecting double JPEG compression," pp. 1–16, 2017.
40. R. Zhao, W. Luk, X. Niu, H. Shi, and H. Wang, "Hardware Acceleration for Machine Learning," *Proc. IEEE Comput. Soc. AnnuSymp. VLSI, ISVLSI*, vol. 2017-July, pp. 645–650, 2017.
41. Z. .Zheng and T. . Zhang, "Hardware Accelerator Design for Machine Learning," *Sch. Enviromental Sci.*, 2012.
42. A. S. Ardestani, "Design and Optimization of Hardware Accelerators for Deep Learning," no. May, pp. 1–10, 2017.
43. C. J. Wu *et al.*, "Machine learning at facebook: Understanding inference at the edge," *Proc. - 25th IEEE Int. Symp. High Perform. Comput. Archit. HPCA 2019*, pp. 331–344, 2019.
44. Q. Zhang *et al.*, "Efficient deep learning inference based on model compression," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 1776–1783, 2018.
45. V. Vanhoucke, A. Senior, and M. Mao, "Improving the speed of neural networks on CPUs," *Proc. Deep Learn. …*, pp. 1–8, 2011.
46. Arunava, "Malaria Cell Images Dataset," 2018. [Online]. Available: https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria. [Accessed: 30-Apr-2019].
47. P. Mooney, "Chest X-Ray Images (Pneumonia)," 2018. [Online]. Available: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/metadata. [Accessed: 30-Apr-2019].
48. P. Roy, "Natural Images Dataset," 2018. [Online]. Available: https://www.kaggle.com/prasunroy/natural-images. [Accessed: 30-Apr-2019].