Efficient Device-Edge Inference for Disaster Classification

Nathaniel Tan Sze Yang
Department of Electrical and Electronic Engineering
Universiti Tunku Abdul Rahman
Kajang, Malaysia
nat980718@1utar.my

Mau-Luen Tham
Department of Electrical and Electronic Engineering
Universiti Tunku Abdul Rahman
Kajang, Malaysia
thamml@utar.edu.my

Sing Yee Chua
Department of Electrical and Electronic Engineering
Universiti Tunku Abdul Rahman
Kajang, Malaysia
sychua@utar.edu.my

Ying Loong Lee
Department of Electrical and Electronic Engineering
Universiti Tunku Abdul Rahman
Kajang, Malaysia
leeyingl@utar.edu.my

Yasunori Owada
Resilient ICT Research Center
National Institute of Information and Communications Technology (NICT)
Tokyo, Japan
yowada@nict.go.jp

Suvit Poomrittigul
Software Engineering and Information System
Pathumwan Institute of Technology
Bangkok, Thailand
suvit@pit.ac.th

***Abstract*—Image classification can learn useful insights from crisis incidents and is gaining popularity in the field of disaster management. This is fueled by the recent advances in computer vision and deep learning techniques, where accurate neural network models for disaster type classification can be accrued. However, these studies quite commonly neglect the prohibitive inference workload which may hamper its wide-spread deployment, especially for model execution on low-powered edge devices. In this paper, we propose a lightweight disaster classification model that recognizes four types of natural disaster plus one non-disaster class. To support real-time applications, the proposed model is optimized with OpenVINO, which is a neural network acceleration platform. Different from existing works which focus on benchmarking at training stage, our experimental results reveal the actual performance at inference stage. Specifically, the optimized version achieves up to 23.93 frames per second (FPS), which is more than doubled the speed achieved by the original model, while sacrificing only 0.935 % of classification accuracy.**

Keywords—natural disaster, deep learning, disaster image classification, OpenVINO, benchmarking

I. Introduction

Artificial intelligence (AI) algorithms are developed with the intention of making decisions in real life. Moving forward, convolutional neural network (CNN), which is an advanced version of AI, is able to learn more meaningful insights from images. The training process of these neural network models can be facilitated by open-source deep learning (DL) frameworks such as TensorFlow [1] and Keras [2]. The growing popularity of CNN have paved the way for new computer vision applications. One specific area would be disaster management [3], where video surveillance cameras and sensors can be leveraged to gain situational awareness.

A natural disaster is an incident caused by nature's threat. It can be defined as a natural phenomenon that causes the health impacts of mankind, loss of livelihoods and services, social and economic disruption, or properties and environmental damage [4]. Some examples are tornadoes, earthquakes, floods, and wildfires. Monitoring these disasters at large-scale coverage would require a plethora of Internet of things (IoT) devices [5], which often have long-range transmission range but low computational power.

Existing works for disaster classification quite commonly neglect the prohibitive inference workload which may hamper its wide-spread deployment, especially for model execution on low-powered edge devices. In this paper, we propose a lightweight disaster classification model that identifies four types of natural disasters and one non-disaster class. The optimized model facilitates edge computing, which is one of goals of the ASEAN IVO project titled "Context-Aware Disaster Mitigation using Mobile Edge Computing and Wireless Mesh Network".

The contributions in this study are threefold. First, we consolidate a dataset which consists of natural disaster and non-disaster images (natural sceneries). Second, we employ the transfer learning approach to output a disaster classification model before optimizing the model with OpenVINO. Third, we provide benchmark results for both training and inference stages, which sheds more insights into the actual implementation performance.

The rest of the paper is organized as follows. Section II discusses the related works. Section III describes the proposed solution. Section IV presents the experimental results and discussions. Section V concludes the article.

II. Related Work

DL, especially CNN has gained momentum in disaster monitoring. According to [6], majority works have focused on CNN instead of machine learning (ML) methods due to its superior performance. Such gain, however, are only possible under the availability of abundant labelled datasets.

Recognizing the importance of datasets, the authors in [7] consolidated a substantial amount of human detection and action detection dataset for disaster management application. The goal was to develop a DL-based drone surveillance framework. However, the framework did not consider disaster event classification. A similar work can be found in [8], where the authors utilized various CNN architectures including ResNet50, Inception V3 and AlexNet in identifying survivors in debris. This time, the annotated images were focused on earthquake-hit regions.

The work in [9] focused on classifying disaster events after collecting more than 7000 images consisting of cyclones, drought, earthquakes, floods, landslides, thunderstorms, snowstorms, and wildfires, from social media platforms. The burden of annotating data was relieved by adopting active learning, which automatically chooses and labels the data

from which it learns without human interaction. The authors in [10] further divided disaster-related images into four different categories, namely disaster type detection, informativeness, humanitarian and damage severity. By setting binary and multiclass classification labels on a consolidated dataset, benchmark results using several CNN architectures were provided.

Apart from the disaster-related images, text messages contain critical information such as infrastructural damages, casualties, and help requests. The usefulness of such social media data has motivated the authors in [11] to develop a multimodal fusion model, which combines both visual and textual features to classify relevant disaster images. However, all the above studies focused on accuracy measurement, where high-end graphics processing unit (GPU) such as NVIDIA Tesla P100 GPU was utilized. Deploying these trained models directly on resource-constrained edge devices remains a challenging task [12]. This is especially true for real-time disaster monitoring applications.

Different from the aforementioned works, the authors in [13] assessed the CNN performance in terms of accuracy and speed. Results showed that their proposed model was able to achieve 9 frames per second (FPS) on a low-powered embedded device, while maintaining reasonable accuracy. However, they did not explore the potential of neural network optimization on target devices at the inference stage. Such performance acceleration is made possible with an open-source CNN model inference engine called OpenVINO Toolkit [14]. The study in [15] benchmarked several pretrained CNN models under the OpenVINO settings. However, it remains unaddressed as in how much improvement can be brought to implementation by OpenVINO, as compared to the unoptimized version.

III. DL Model Deployment

To achieve a robust DL model, training and inference phases must be analyzed correctly. To this end, we propose the methodology shown in Fig. 1, where the three stages are necessary to evaluate the actual performance.

A. Model Training

A new natural disaster classification model is trained using the transfer learning approach. Without loss of generality, we select VGG16 to be the neural network architecture due to its high accuracy [6,10,11,13,14, 15]. The results are also expectably applicable to other architectures such as DarkNet-53 [16]. The collected dataset contains natural disaster and non-disaster images, which were downloaded from public sources: [17] and [18], respectively. The natural disaster data consists of cyclones, earthquakes, floods, and wildfires. On the other hand, the non-disaster images comprise of nature scenes such as coast, mountain, forest, open country, as well as man-made scenes like street, inside city, buildings, and highways. Table I summarizes the data distribution among training, testing, and validation.

The dataset split is 67.5 % for training, 25 % for testing, and 7.5 % for the validation split. The parameters to fine-tune the VGG16 model are shown in Table II. The batch size means the number of images from the dataset that are selected from the beginning and used to train the natural disaster classification model in each iteration throughout the training dataset. The number of steps is the number of iterations. After each step or iteration, the gradient of the natural disaster classification model will be updated. Once all images of the training dataset are gone through, one epoch is completed. The values of the minimum and maximum learning rates are used by the cyclical learning rate (CLR) technique to improve the accuracy of the model [19].

CLR requires the minimum and maximum boundary values before it can be used. A test on the learning rate range is executed whereby training starts at a lowest learning rate of $10\,e^{-10}$. After each batch update, the learning rate will increase exponentially until it reaches a rate of $10e^{1}$, and the current learning rate and loss will be logged simultaneously. The loss gives the idea of how the model performs in the training and validation datasets as shown in Fig. 2. The CLR technique makes the learning rate moves cyclically between the set boundaries as shown in Fig 3.

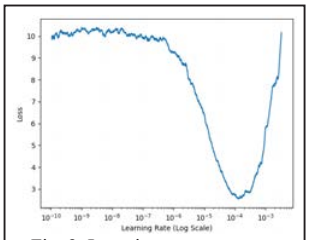

Fig. 2. Learning rate range test

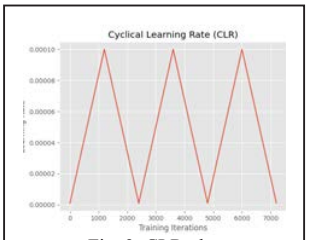

Fig. 3. CLR plot

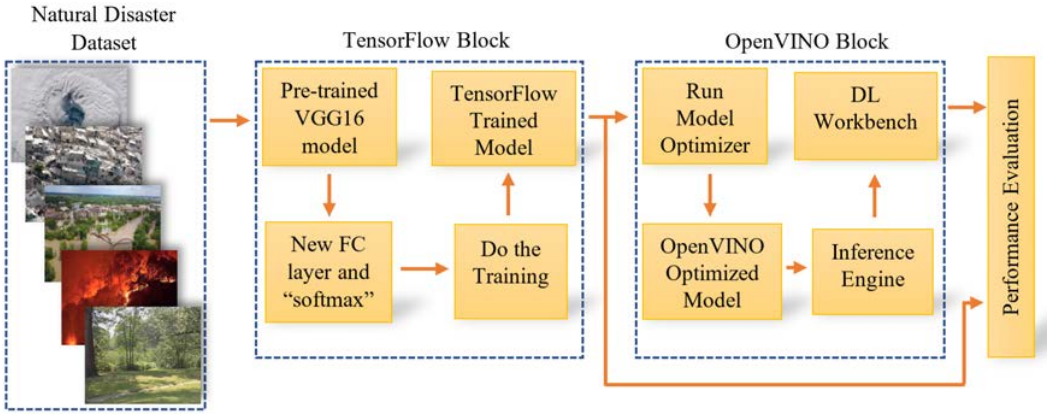

Fig. 1. Methodology for performance evaluation of the proposed model using OpenVINO Deep Learning Workbench.

TABLE I. DATA SPLIT FOR DISASTER CLASSIFICATION

Disaster Label	Train	Validation	Test	Total
Cyclone	599	78	251	928
Earthquake	923	86	341	1350
Flood	741	80	252	1073
Wildfire	724	68	285	1077
Non-Disaster	1821	223	652	2696

TABLE II. PARAMETERS AND ITS VALUES FOR FINE-TUNING THE VGG16 MODEL

Parameter	Value
Batch Size	32
Number of Steps	8
Epoch	48
Min Learning Rate	1e-6
Max Learning Rate	1e-4

Along with the immediate responses, the correctness of the result is also a very crucial parameter for such applications.

B. Model Optimization

To reach the goal of this study, we proposed the method shown in Fig. 1. There are five processes to get the output predictions for each image in the inference stage. Each stage will be discussed in the following sections.

1) *Obtaining the trained model.* A transfer learning approach is applied to the pre-trained VGG16 model to output a new natural disaster classification model. Fine-tunings and parameters are modified with the intention of minimizing execution time and increasing accuracy.

2) *Freezing the model.* The model is saved as a .pb file with the weights frozen during the training stage. TensorFlow version 2.0 is used to execute the training and freeze the model.

3) *Model conversion to a compatible format*. Conversion of the trained model into Intermediate Representation (IR) format needs to be done in order for it to be used in the OpenVINO environment [20]. OpenVINO's model optimizer tool is used to perform the conversion, with the following code and parameter:

```
python3 mo_tf.py --saved_model_dir <model-path>
--output_dir <output-dir> --input_shape
[1,224,224,3]
```

The parameter –input_shape [1,224,224,3] defines the input data properties of the model in the training as follows: Number of images [N] × Height [H] × Weight [W] × Channels [C]. Note that [N,H,W,C] is for a TensorFlow model. After a successful conversion of the model to IR format, a .xml (describes the network topology) and a .bin (contains the weights and biases binary data) file will be generated [21][22].

4) *Executing inferences*. In this stage, OpenVINO's Inference Engine tool is used to perform the inference. A custom Python script is executed to initiate plugins, load IR model, read the label, input data, infer and process the output. The alternative of using a python script is the Deep Learning Workbench (DL Workbench).

5) *Performance evaluation*. The original (TensorFlow) model and optimized (OpenVINO) model are evaluated based on the 25 % test dataset, for precise and accurate measurements. For the original model, it is evaluated using the *classification_report* function in TensorFlow while the optimized model uses DL Workbench. The precision of the TensorFlow's natural disaster classification model is floating-point (FP) 32. In the optimized model, the precision can be FP 32, FP 16, and integer (INT) 8. In theory, a higher precision gives a higher accuracy but requires higher computational power, and vice versa.

C. Model Inference

There are two inference modes: synchronous and asynchronous. The data were fed into the inference engine in a synchronous manner, allowing only one image to be processed per inference. Asynchronous inference, on the other hand, speeds up the process by inferencing one image while pre-processing the next image.

The script to run the inference of the TensorFlow model is in asynchronous mode. By default, the inference model in the Inference Engine of OpenVINO also uses asynchronous mode. However, there are certain disadvantages to this method, as acquiring the predictions comes after all of the flow is completed.

IV. EXPERIMENTAL RESULTS

The proposed model is evaluated using the dataset provided in [16] and [17]. Sample images from the dataset can be seen in Fig 1. The dimension of the images varies throughout the dataset. Image pre-processing based on the required input size of 224 × 224 pixels is done before the training or inferencing of the model.

Regarding the hardware used, there are two environments that are used to carry out the experiments. Their main specifications are described in the following items.

1) *Hardware on training phase*: The hardware used in this phase is an Intel NUC equipped with a 10th–generation i7-10710U 6-core Intel processor with 64 GB of memory and 1 TB of a solid-state drive as the storage. The operating system in the Intel NUC is the Ubuntu 18.04 LTS version. The software used for training is TensorFlow version 2.0.

2) *Hardware on the inference phase*: The hardware is similar to that of the training phase. The difference is that the inference for the optimized model is able to take advantage of the Intel integrated graphics, which is Intel UHD Graphics. The optimized model is executed in the OpenVINO environment, and the version is OpenVINO 2021.4.

A. Training Performance

Once the training of the model is completed, a performance test of precision, recall, f1-score, and accuracy is executed by the function *classification_report*. The 25 % test dataset is used for the performance evaluation and has a total number of 1781 images.

Table III shows the performance results of the TensorFlow model. It is noteworthy that *precision* gives us an idea of how well the model classifies a natural disaster when it output/classify a natural disaster; *recall* ratio gives us an idea of how well the model classifies a natural disaster, given an input of natural disaster scenario; *F1-score* is the harmonic mean between precision and recall; *support* is the number of occurrences (or images used) in each class to produce the results; and *accuracy* is the ratio of the correct predictions to all of the predictions.

The performance of TensorFlow's model has achieved an accuracy of 93 %. A few videos have been used as inputs to the model and it achieved an average of 7.70 FPS. Note that the same test dataset is used to evaluate the performance of the optimized model in the subsequent sub-section.

B. *Inferences Performance*

Once the trained model is converted successfully into OpenVINO IR format, the optimized model is used by the Inference Engine to do inferencing. The DL Workbench tool is used to obtain the inference performance of the optimized model. Figs. 4 - 7 shows the inference performance overview obtained from the DL Workbench.

TABLE III. RESULTS OF TENSORFLOW NATURAL DISASTER CLASSIFICATION MODEL

Disaster Class	Precision	Recall	F1-Score	Support
Cyclone	96%	98%	97%	251
Earthquake	94%	92%	93%	341
Flood	84%	90%	87%	252
Wildfire	93%	93%	93%	285
Non-Disaster	96%	93%	95%	652

Fig. 4. Performance overview of the optimized FP32 model on Intel CPU

Fig. 5. Performance overview of the optimized INT8 model on Intel CPU

Fig. 6. Performance overview of the optimized FP32 model on Intel GPU

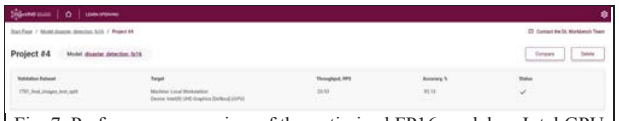

Fig. 7. Performance overview of the optimized FP16 model on Intel GPU

TABLE IV. RESULTS OF OPENVINO OPTIMIZED MODEL RUNNING ON CPU

Precision	FP 32	FP 16	INT 8
Throughput (FPS)	11.81	-	21.35
Accuracy (%)	92.19	-	92.30

TABLE V. RESULTS OF OPENVINO OPTIMIZED MODEL RUNNING ON INTEGRATED GPU

Precision	FP 32	FP 16	INT 8
Throughput (FPS)	9.15	23.93	-
Accuracy (%)	92.19	92.13	-

Table IV presents the performance results of the optimized model that runs on an Intel CPU. The INT 8 precision model has achieved an increase of 80.8 % FPS and 0.119 % accuracy, showing higher performance as compared to the FP 32 precision model. Hence, the best performance with Intel CPU is the INT 8 precision model. The FP 16 precision model is not available as it will upscale the model to the FP 32 to perform inference due to the limitation of DL workbench and the particular Intel CPU used in this study.

Table V shows the performance results of the optimized model that runs on an Intel integrated GPU. The FP 16 precision model has obtained 162 % higher FPS while sacrificing 0.0650 % accuracy, compared to the FP 32 precision model. The INT 8 precision model is not supported on the integrated GPU model [25]. Since the accuracy drop in the FP 16 precision model is very low, along with the considerable increase of throughput, the FP 16 precision model provides the best performance on Intel integrated GPU hardware.

Since TensorFlow's model runs on the CPU, to ensure reliable and accurate results, the comparison is done on the optimized FP 32 and INT 8 precision models of the optimized model that ran on the same CPU. The optimized FP 32 precision model achieves an increase of 53.4 % in throughput with a loss of 0.871 % in accuracy. Meanwhile, the optimized INT 8 precision model achieves an increase of 177 % in throughput at the cost of 0.753 % in accuracy.

On the other hand, if the program is implemented on an edge device, which is the Intel NUC in our study, and the optimized model is able to run on the Intel integrated GPU hardware. Only the optimized FP 32 and FP 16 precision model is able to take advantage of the GPU hardware. Since the optimized FP 16 precision model provides the best performance on GPU hardware, it achieves a substantial improvement of 211 % in throughput while only sacrificing 0.935 % accuracy as compared to the TensorFlow's model.

The comparisons show that the OpenVINO optimized models have a better performance enhancement over the TensorFlow's model in terms of frame rate inference while losing a negligible amount of accuracy.

DL Workbench is able to display the results of the performance summary of the model as shown in Figs. 8 - 11. This allows us to identify the throughput, latency, batch, and streams values of the model.

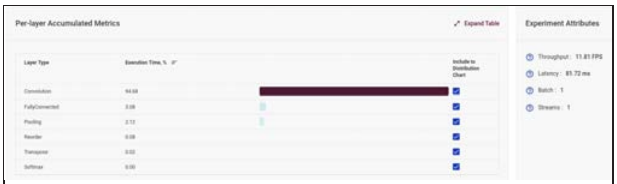

Fig. 8. Performance results of the optimized FP32 model on Intel CPU

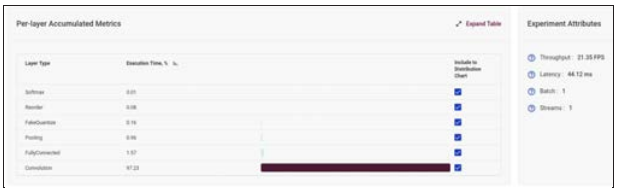

Fig. 9. Performance results of the optimized INT8 model on Intel CPU

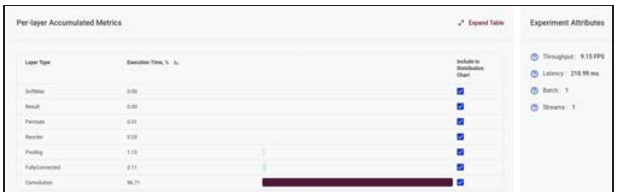

Fig. 10. Performance results of the optimized FP32 model on Intel GPU

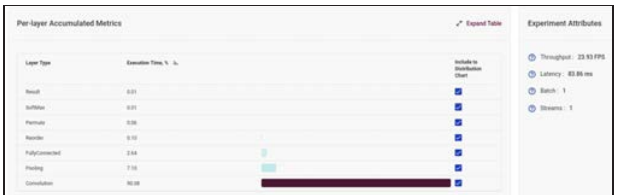

Fig. 11. Performance results of the optimized FP16 model on Intel GPU

In this part, we evaluate the performance in terms of throughput and latency. It is noteworthy that throughput is the number of images processed in a certain amount of time, which is one second, and latency is the amount of time used to perform an inference for a single image [23]. The INT8 model on the CPU and the FP16 model on the Intel GPU achieve a high throughput while only the INT8 model achieves the lowest latency of 44.12 milliseconds.

V. Conclusion and Future Lines

Natural disasters happen all around the world. Early detection of natural disasters for the people staying around the danger zone can enable safe evacuation of the people to a nearby shelter. The main issue that the current study aims to address is the unbalanced dataset and the need of powerful hardware for performing inference. To overcome the dataset limitation, we have consolidated a natural disaster dataset, and trained a new natural disaster classification model to classify natural disaster and non-disaster scenarios. We have addressed the need for powerful hardware by deploying the trained model into the OpenVINO platform. Lastly, we have evaluated the performance of the trained model and concluded that the model performs significantly better in the OpenVINO environment as compared to the TensorFlow environment. DL Workbench is a great tool for conversion of model, analysis of the converted model, as well as the performance measurement that can be done on it. As for future research works, the power consumption of the model running in different environments can be measured and it will be used as one of the performance metrics.

Acknowledgment

This work is the output of the ASEAN IVO (http://www.nict.go.jp/en/asean_ivo/index.html) project titled “Context-Aware Disaster Mitigation using Mobile Edge Computing and Wireless Mesh Network” and financially supported by NICT (http://www.nict.go.jp/en/index.html).

References

[1] "TensorFlow", [online] Available: https://www.tensorflow.org/overview.

[2] "Keras", [online] Available: https://keras.io/

[3] Lopez-Fuentes, L., van de Weijer, J., González-Hidalgo, M. *et al.* Review on computer vision techniques in emergency situations. *Multimed Tools Appl* **77,** 17069–17107 (2018). https://doi.org/10.1007/s11042-017-5276-7

[4] Chen, Y., Li, C., Chang, C. and Zheng, M., 2021. Identifying the influence of natural disasters on technological innovation. *Economic Analysis and Policy*, 70, pp.22-36.

[5] M. A. Al-Mashhadani, M. M. Hamdi and A. S. Mustafa, "Role and challenges of the use of UAV-aided WSN monitoring system in large-scale sectors," *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2021, pp. 1-5, doi: 10.1109/HORA52670.2021.9461292.

[6] R. R. Arinta and E. Andi W.R., "Natural Disaster Application on Big Data and Machine Learning: A Review," *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2019, pp. 249-254, doi: 10.1109/ICITISEE48480.2019.9003984.

[7] B. Mishra, D. Garg, P. Narang and V. Mishra, "Drone-surveillance for search and rescue in natural disaster", *Computer Communications*, 2020.

[8] Chaudhuri N, Bose I (2020) Exploring the role of deep neural networks for post-disaster decision support. Decis Support Syst 130:113234. https://doi.org/10.1016/j.dss.2019.113234

[9] L. Ahmed, K. Ahmad, N. Said, B. Qolomany, J. Qadir and A. Al-Fuqaha, "Active Learning Based Federated Learning for Waste and Natural Disaster Image Classification," in *IEEE Access*, vol. 8, pp. 208518-208531, 2020, doi: 10.1109/ACCESS.2020.3038676.

[10] F. Alam, F. Ofli, M. Imran, T. Alam and U. Qazi, "Deep Learning Benchmarks and Datasets for Social Media Image Classification for Disaster Response," *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2020, pp. 151-158, doi: 10.1109/ASONAM49781.2020.9381294.

[11] Zou, Z.; Gan, H.; Huang, Q.; Cai, T.; Cao, K. Disaster Image Classification by Fusing Multimodal Social Media Data. *ISPRS Int. J. Geo-Inf.* 2021, *10*, 636

[12] Z. Jiang, T. Chen, and M. Li, “Efficient Deep Learning Inference on Edge Devices”, in *Proceedings of ACM Conference on Systems and Machine Learning (SysML’18)*, 2018.

[13] C. Kyrkou and T. Theocharides, "Deep-learning-based aerial image classification for emergency response applications using unmanned aerial vehicles", *Proc. IEEE Conf. Comput. Vision Pattern Recognit. Workshops*, pp. 517-525, Jun. 2019.

[14] A. Rosebrock, Detecting Natural Disasters with Keras and Deep Learning:PyImagesearch, 2019, [online] Available: https://pyimagesearch.com/2019/11/11/detecting-natural-disasters-with-keras-and-deep-learning/.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv:1409.1556*, 2014.

[16] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018

[17] Gautam Kumar, Google Images, 2019 (accessed March 23, 2022). [Online]. Available: https://drive.google.com/file/d/1NvTyhUsrFbL91E10EPm38IjoCg6E2c6q/view

[18] Gautam Kumar, Google Images, 2019 (accessed March 23, 2022). [Online]. Available: https://drive.google.com/file/d/11KBgD_W2yOxhJnUMiyBkBzXDPXhVmvCt/view

[19] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 464-472, doi: 10.1109/WACV.2017.58.

[20] Docs.openvinotoolkit.ai. 2022. *Converting a Model to Intermediate Representation (IR)*. [online] Available at: <https://docs.openvino.ai/latest/openvino_docs_MO_DG_prepare_model_convert_model_Converting_Model.html> [Accessed 21 March 2022].

[21] Docs.openvinotoolkit.ai. 2022. *Model Optimizer Developer Guide*. [online] Available at: <https://docs.openvino.ai/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html> [Accessed 21 March 2022].

[22] Docs.openvinotoolkit.ai. 2022. *Converting a Model Using General Conversion Parameters*. [online] Available at: <https://docs.openvino.ai/2021.1/openvino_docs_MO_DG_prepare_model_convert_model_Converting_Model_General.html> [Accessed 24 March 2022].

[23] Docs.openvinotoolkit.ai. 2022. *DL Workbench Key Concepts*. [online] Available at: <https://docs.openvino.ai/2021.3/workbench_docs_Workbench_DG_Key_Concepts.html> [Accessed 28 March 2022].