Implementation of Video-dehazing based on AOD-Net using Docker on Jetson Nano

Challa Gopala Krishna Reddy
Electronics and Communication Department
National Institute of Technology Warangal
Warangal, India
cg21ecb0b09@student.nitw.ac.in

Shrikar Kaveti
Electronics and Communication Department
National Institute of Technology Warangal
Warangal, India
ks21ecb0b21@student.nitw.ac.in

Dr. Ravi Kumar Jatoth
Electronics and Communication Department
National Institute of Technology Warangal
Warangal, India
ravikumar@nitw.ac.in

Abstract—**This paper deals with the implementation of a machine learning model used for dehazing video inside a Docker container. Docker containers provide the user a standardized environment, promoting portability and making it easier to deploy the model across various computing platforms. The paper discusses in detail about the decision behind selecting the model, process of containerizing the dehazing model, including required dependencies and configurations. The advantages of usage of Docker for video dehazing are discussed, stressing upon the enhanced portability, streamlined deployment, and potential for cloud-based applications. The performance of the model is judged based on parameters like PSNR, SSIM and average time taken for processing each frame. The paper concludes by outlining the effectiveness of Docker in facilitating the practical use of machine learning models for video dehazing tasks.**

Keywords—docker, Jetson Nano, Video Dehazing, TensorFlow, PSNR, SSIM

Abbreviations—PSNR (Peak Signal to Noise Ratio), SSIM (Structural Similarity Index Measure), GPU (Graphics Processing Unit), DCP (Dark Channel Priors), CNN (Convolution Neural Network), AOD-Net (All-in-One Dehazing Network), CPU (Central Processing Unit), ML (Machine Learning), ReLU (Rectified Linear Unit), MSE (Mean Square Error), RMS (Root Mean Square Error)

I. Introduction

Haze and Fog are weather conditions which are highly prevalent in many regions across the world during winters which lead to degradation of human vision as well as visual quality in images and videos [1]. Due to reduced visibility, a lot of details present in the scene are disturbed [26]. This reduced visibility is a major threat for a lot of applications which depend on these important details such as surveillance [2], object detection [3], autonomous driving [4] and outdoor cinematography [5]. Dehazing techniques are used in order to recover some clarity from the scene by removing the haze/fog present to enhance visibility of objects. Single Image dehazing deals with improving the clarity of individual frames whereas Video-dehazing deals with improving the visual quality of entire video sequence. In most of the cases, the requirement is to implement the models in real-world applications on different computational platforms. But these models require specific environment and dependencies in order to work properly, which makes the deployment of these models a complicated an cumbersome process. In addition to this, it is also important that the model being executed uses minimal resources so that it is easy to implement on low-end devices. Therefore, there is a need for a solution which is lightweight and is easily executed but also easily deployable across different platforms.

In this paper, we explore the implementation of model inside a Docker container. Docker provides a lightweight, portable, and scalable platform making it easier for deploying applications in devices in varying computing infrastructures [6] . This type of implementation has several advantages. Firstly, it provides an isolated environment, which eliminates all compatibility issues. Secondly, Docker is highly efficient in utilizing resources by sharing the underlying operating system across containers. In addition to these, this style of deployment also has another advantage of being easily scalable for supporting real-time applications and processing large datasets [7].

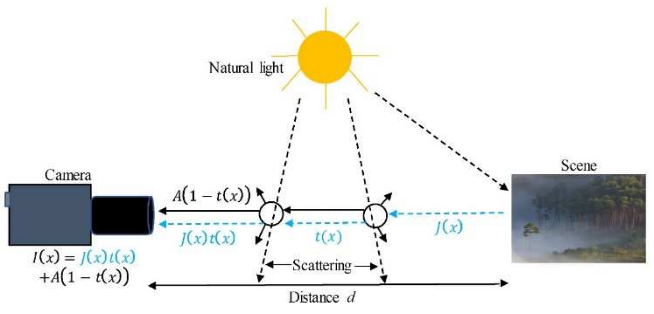


Fig. 1. Atmospheric Light Scattering Model [27]

Finally, we show the performance of the model by making use of metrics such as PSNR and SSIM where videos being used are of resolution 480x650. We also show the implementation of the model on Jetson Nano board by making use of docker and provide statistics like GPU consumption and time taken to process each frame inside the Docker container showing that this can be implemented easily on low-end computational devices.

II. Related Work

The research that has happened in the field of image dehazing can be divided into 2 sections which are prior based methods and deep learning based [8]. Prior models are mainly based on mathematical models like DCP (Dark Channel Priors) [9][23]. The biggest problem with these is that they cannot be helpful in wide range of scenarios [10][24][25]. Deep learning models are preferred to these as they generally

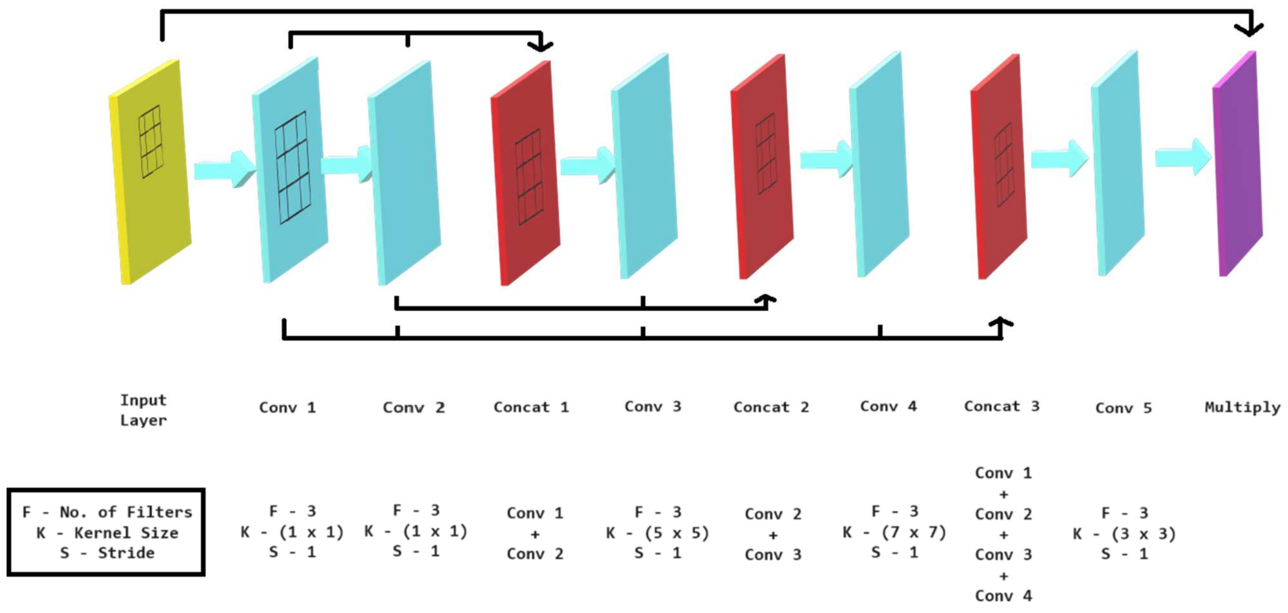


Fig. 2. Network Design

perform better than these traditional methods [11][12]. Hence, multiple studies [13][14] use CNN based models to dehaze images.

A. *Atmospheric Light Scattering Model*

Any hazy image can be described easily by making use of atmospheric scattering model as follows [15]:

$$I\ (x) = J\ (x)\ t(x) + A(1 - t(x)) \tag{1}$$

Where $I(x)$ is observed hazy image, $J(x)$ is scene radiance, A is global atmospheric light and, $t(x)$ is transmission matrix is defined as

$$t(x) = e^{-\beta d(x)} \tag{2}$$

where, beta is scattering coefficient of atmosphere and $d(x)$ denotes distance between camera and object

AOD-Net (All-in-One Dehazing Network**)** [14] makes use of CNN's in order to give the final dehazed image. In other models, the transmission map and atmospheric light are calculated separately and then clean image is reconstructed based on equation (1) [13]. The problem with this process is that it error that is present in each step of the estimation will keep increasing as both estimation errors add up leading to increased error.

However, AOD-Net uses a different approach which helps us in getting the desired dehazed image by estimating only one parameter.

Equation (1) can be rearranged as follows:

$$J(x) = \frac{1}{t(x)}\, I(x) - A\, \frac{1}{t(x)} + A \tag{3}$$

Instead of calculating two different parameters, AOD-Net unifies them into one single parameter $K(x)$. The modified version of equation (1) can be written as follows:

$$\begin{gathered} J(x) = K(x)\, I(x) - K(x) + b, where \\ K(x) = \frac{\frac{1}{t(x)}\,(I(x) - A) + (A - b)}{I(x) - 1} \end{gathered} \tag{4}$$

B. *Docker*

Docker is an platform which is used for developing, shipping and running applications. It packages the application with everything the application needs such as code, libraries etc into a container [16]. These containers are independent and can run on any computer, making it easy to share and deploy software quickly.

Unlike virtual environment which virtualize hardware, docker virtualizes software which makes them much lighter and faster. Docker containers use a Linux kernel mechanism in order to allocate resources. When a container is being created, the user can allow the allocation of resources, such as network configuration, CPU, and memory [6].

While Docker can be used for different sets of applications, the use of Docker for ML based applications is gaining interest for various reasons. Firstly, docker containers provide a high level of isolation, ensuring that each application is running in its own environment without interfering with other processes or dependencies [7]. This is a very important feature for ML models as each model has its own requirements for libraries, dependencies and runtime environments.

Secondly, the docker containers are easily portable across different platforms and environments, making it easy to deploy ML application. This reduces the risk of compatibility issues and simplifies the deployment process [16].

Using Docker, one can easily reproduce the same results as the dependencies do not change over time. This solves a major problem for ML developers as they can easily share Docker images containing their models, datasets and code, and this can be used by others in order to replicate their experiments exactly. Lastly, Docker allows user to make use of GPUs for computationally intensive ML tasks by specifying the GPU resources within the container.

III. Network Design

The Model used for demonstration is AOD-Net which consists of two functions (Shown in Fig. 2): Initial function has five convolution layers for estimation of K factor, then in final function multiplication operation is performed on each pixel to recover the De-hazed image from the Haze image.

The K factor is used for calculation of the depth and Beta (Haze Density) of the Image. As shown in the Fig. 2. In the Initial Function, each convolution layer has kernel with different size for identifying multi-scale features. Convolution layers in the model are concatenated for capturing characteristics of the Image at different scales. "concat1" layer is the concatenation of "conv1" layer and "conv2" layer; "concat2" layer is the concatenation of "conv2" layer and "conv3" layer, "concat3" layer concatenation of "conv1" layer, "conv2" layer, "conv3" layer and "conv4" layer. for testing the performance improvement caused by concatenation layer, a model without concatenation is tested which produces Average PSNR (15.889) and SSIM (0.51909) and Increasing number of concatenation also degrades the PSNR values and increases the computational cost for running the model.

Layer Name	Layer	No. of Filters	Kernal Size	Stride
Conv 1	Convolution 2D	3	1 x 1	1
Conv 2	Convolution 2D	3	1 x 1	1
Conv 3	Convolution 2D	3	5 x 5	1
Conv 4	Convolution 2D	3	7 x 7	1
Conv 5	Convolution 2D	3	3 x 3	1

Table I. CNN Hyperparameters

A. *Necessity of K factor Module*

Most Machine learning models used for generating the dehazed image require very high computational power. These model use Up and Down sampling or Image denoising to generate the dehazed image. because of the requirement for high computation it is difficult for the model to run in real time and can be run on portable devices (Ex: Auto-Pilot Vehicles) [17]. The K Factor is calculated using the image atmospheric light and depth of the Image as given the Atmospheric Scattering Model which makes it easy to model and computationally light weight.

B. *Necessity of Light Weight Model*

Dehazing Model can be used in Automated Vehicles for better detection of path and objects in adverse weather conditions which require real time computation which is possible by using the light weight model [18] but the light weight model cannot be more accurate or cannot produce better results than the computationally heavy models [19].

Computationally heavy model can be used for surveillance or other uses which doesn't require real time computation and model can take a large amount of time for post processing.

Hyperparameters	Value
Stride	1
Padding	same
Activation	ReLU
Neural Network Bias	True
Kernal Initializer	Random (Normal Distribution)
Kernal Regularizes	L2 (Ridge Regression)
Batch Size	1
Epochs	25
Callback	Reduce Learning Rate on Plateau
Optimizer	Adams
Loss Function	MSE
Model Loss	0.030715337023139
Model Accuracy	89.92100358009338 %

Table II. Network Hyperparameters

IV. Evaluation

The model that was used for evaluation was first trained on NYU2 Depth Dataset. This Dataset consists of indoor images. The images present in this dataset have beta values belonging to the range [0.4, 1.6]. The model was trained on Multiple Datasets (NYU2, HazeRD, REVIDE) images. Out of these, 550 were used in training dataset and 50 images were used in validation dataset. The activation function used for training was ReLU. The loss function used while training our model was MSE (Mean Square Error) whereas the optimizer used while training the model was Adam's Optimizer. The model took around 10 training epochs to converge.

Even though the model was trained on indoor images, outdoor images were used in order to test the model. The images used to test the model were taken from HazeRD dataset which consists of ground truth image and hazy images of different beta values giving a good understanding about the models output for different environments. After testing the model on various outdoor images, we tested the model by giving it a few synthetic video samples and a few real-life video samples.

	50m	100m	200m	500m	1000m
Weather Condition	Dense	Thick	Thick	Moderate	Light
Scattering Coefficient (Beta)	78.2	39.1	19.6	7.82	3.91
PSNR	12.8007	17.5181	21.1731	21.4665	20.909
SNR	7.5038	12.2212	15.8762	16.1692	15.612

Table III. Visual range and corresponding scattering coefficient, PSNR, SNR and Weather Condition.

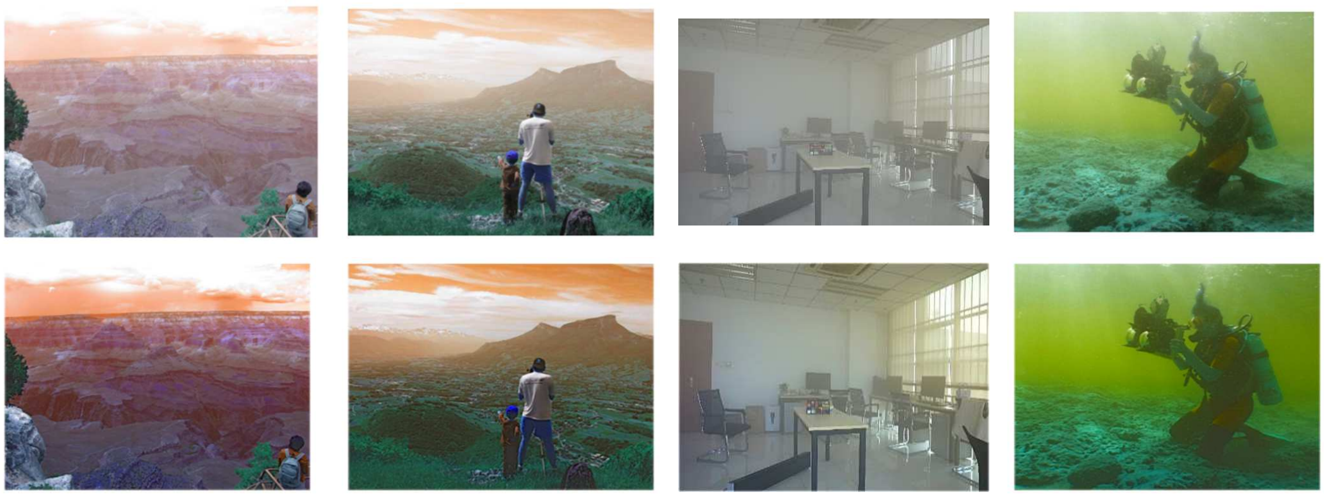


Fig. 3. Dehazing of Synthetically Haze generated Images (A, B) Input Haze Images (Top Row), Output Dehazed Image (Below Row), Dehazing of Real Haze generated Images (C, D) Input Images (Top Row), Output Dehazed Image (Below Row)

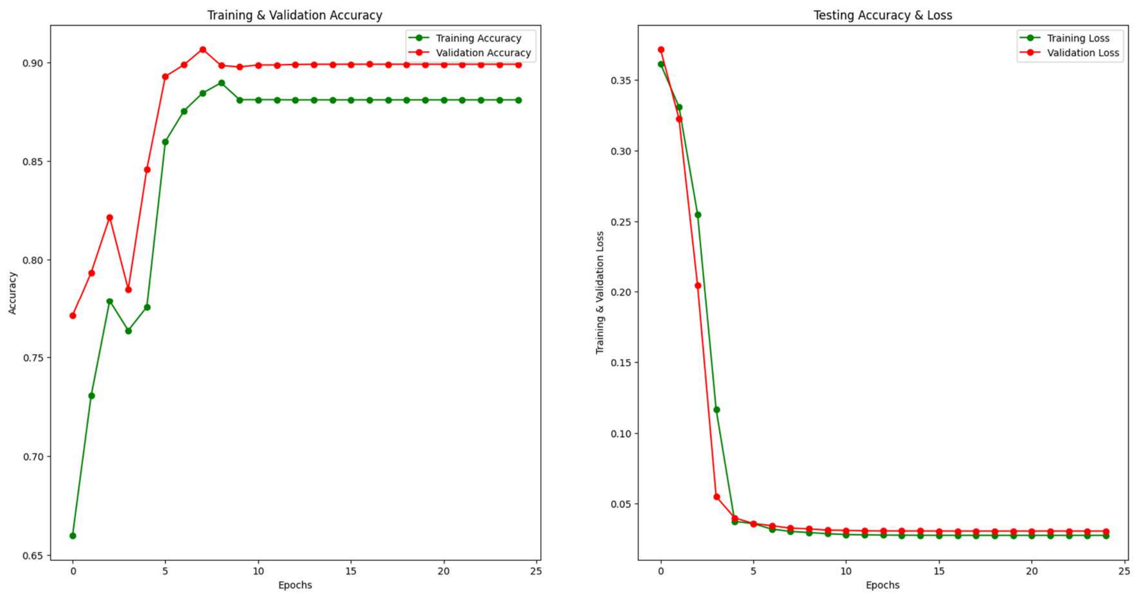


Fig. 4. Model Training Curve

We trained the model on NYU2 Dataset [20] (Synthetic Dataset) which are Indoor Images and tested model on the Real Haze Images from HazeRD and REVIDE Dataset [21]. HazeRD [22] Dataset is a Input Dataset the Images are captured indoor with haze generated using fog generators. AOD Net was evaluated on metrics PSNR, SSIM and MSE. AOD-Net has SSIM metric on average 0.51909 which performs similar to other models tested on HazeRD Dataset. PSNR on the Real Images is on average 15.881 which is lesser than the PSNR Average for Synthetic Dataset. AOD Net has MSE 0.2835 on average which is similar to other models tested on HazeRD Dataset. We tested on model on a real haze video taken in Delhi early morning in December shown in Fig. (6 & 7). The Outputs can be accessed for the following GitHub Repository (https://github.com/shrikarkaveti/Video_Dehazing).

AOD-Net	Average	Max	Min
PSNR	15.8819	21.839	13.5945
SSIM	0.51909	0.89	0.43
RMS	0.2835	0.3621	0.1401

Table IV. AOD-Net Statistics (PSNR, SSIM, RMS)

V. Hardware Implementation

After training the model using dataset and testing it on different conditions, we implemented the model on hardware platform. The Hardware platform used in this case was NVIDIA's Jetson Nano. Jetson Nano has a GPU memory of 4GB 64-bit LPDDR4 which uses NVIDIA's Maxwell architecture. The CPU being used in Jetson Nano is an Quad-core ARM Cortex-A57 MP Core processor. Using Swap File, the memory is expandable up to 6GB where both CPU and GPU are used. Firstly, we took a board and flashed a memory card with Jetpack 4.6.1. This version of the Jetpack comes with Ubuntu 18.04 and has a python version of 3.6.9. After setting up the board using Jetpack, we installed all the necessary libraries. The TensorFlow version required for specific Jetpack version is found from Nvidia Documentation. The TensorFlow version that is supported for this version of Jetpack is 2.7.0.

Since the Python and TensorFlow version being used in the board are relatively old, the model had to be trained in a virtual environment with these specific dependencies so that it can be used properly for execution without any problems. After training the model in this environment, the model was used to make predictions in Jetson Nano.

VI. Docker implementation

The process of setting up the board and installing required dependencies is tedious task which consumes considerable amount of time and the setup process varies from one device to another. In order to solve this problem, we can use Docker to run the model. Docker is a platform that helps developers to package the code/application with all the required dependencies. The main advantage of using Docker is that we can create a portable environment which can be used in any system as it only uses the computational resources of the device.

Unlike Virtual machines which virtualise the hardware of the device, Docker virtualises the software of the device [6]. In order to run the program inside a docker container, a container/image with all the required dependencies has to be created. The required dependencies in our case were TensorFlow, OpenCV-python and other libraries that are in-built in python. In order to create a container, we can either use a pre-existing Docker Image or create a Docker Image with all required dependencies using a Docker File. In our case, we used a pre-existing Docker Image which consists of required TensorFlow version. This Docker Image is present in Docker hub and was created by NVIDIA. In order to use the image, we need to pull it from Docker Hub.

After pulling the image, we ran the image in interactive mode using the following command:

sudo docker run -it --name testcontainer --volume (source_path:destination_path) --rm --runtime nvidia --network host <req-image>

Here the tag --runtime nvidia is important as it enables the image to utilise the GPU present in the device. The tag --rm is used to remove the container after the execution has stopped. By using --volume command, we mount the required files to the docker container to a specified destination inside the docker container. After running this command, we enter inside the container. Inside the container, we can install libraries/tools that are required by us such as open-cv or jupyter notebook etc. After installing all the required libraries, we ran the python file containing the model and saved the predictions in a folder inside the container. These can be copied to any destination inside the host device using the following command:

sudo docker cp <containerId>:/file/path/within/container /host/path/target

One can create a new image based on the container which has all the required libraries as follows:

sudo docker commit <container-id> <image-name>

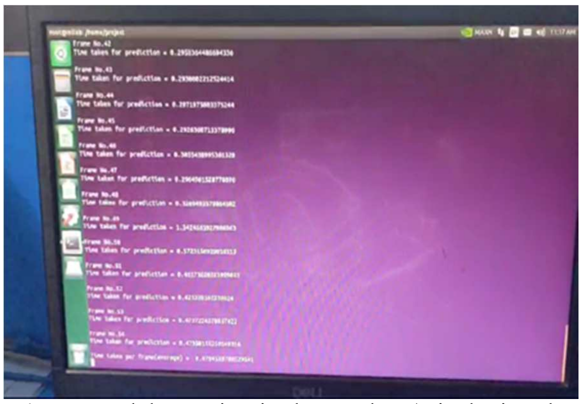
By creating a new image in this way, we get an image which can be executed directly without installing any other libraries while being sure that the image works perfectly. This image can be used in order to execute any code on any Jetson Nano device with the only requirement being that the device must have Docker present inside it. By having a readily executable image, the process of execution of model on any device becomes very simple giving this a wide range of use cases.

VII. Result

A video sequence consisting of 55 frames was given as input for the model which was being executed using docker on Jetson Nano board. It was observed that the time taken on an average to give the prediction for each frame is 0.48s. The Prediction time taken while running the model in docker is almost similar to prediction time for model running in device with a slight delay. This shows that docker is doesn't consume a lot of resources of the device on which it is running.

Metrics	AOD-Net	AOD-Net (Real Images)
PSNR	19.6954	15.8819
SSIM	0.8478	0.51909

Table V. Comparison of Performance metrics of AOD-Net on Synthetic Images and Real Images



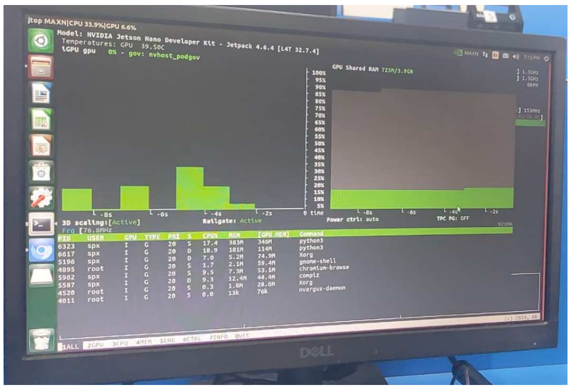
```
Frame No.42
Time taken for prediction = 0.2953[illegible]
Frame No.43
Time taken for prediction = 0.2930[illegible]
Frame No.44
Time taken for prediction = 0.2971[illegible]
Frame No.45
Time taken for prediction = 0.2926[illegible]
Frame No.46
Time taken for prediction = 0.3055[illegible]
Frame No.47
Time taken for prediction = 0.2964[illegible]
Frame No.48
Time taken for prediction = 0.3269[illegible]
Frame No.49
Time taken for prediction = 1.3424[illegible]
Frame No.50
Time taken for prediction = 0.3723[illegible]
Frame No.51
Time taken for prediction = 0.4017[illegible]
Frame No.52
Time taken for prediction = 0.4233[illegible]
Frame No.53
Time taken for prediction = 0.4737[illegible]
Frame No.54
Time taken for prediction = 0.4730[illegible]
Time taken per frame(average) = 0.4794[illegible]
```

Fig. 5. Model Running in the Docker (Displaying the Prediction time for Each Frame)

Before the execution started, the device allocated 350MB of GPU memory for docker.

Metrics	On-Device	Docker
Avg. Time	0.357 ms	0.48 ms

Table VI. Comparison of Average Prediction time for AOD-Net running on Device and Docker



```
jtop MAXN|CPU 33.9%|GPU 6.6%
Model: NVIDIA Jetson Nano Developer Kit - Jetpack 4.6.4 [L4T 32.7.4]
 Temperatures: GPU  39.50C
 1GPU gpu    0% - gov: nvhost_podgov
                                              GPU Shared RAM 723M/3.9GB
3D scaling:[Active]              Railgate: Active      Power ctrl: auto        TPC PG: OFF
 Frq [76.8MHz
PID   USER   GPU  TYPE  PRI  S  CPU%  MEM    [GPU MEM]  Command
6323  spx    I    G     20   S  17.4  383M   346M       python3
6617  spx    I    G     20   D  18.9  181M   114M       python3
5196  spx    I    G     20   D  7.0   5.2M   74.9M      Xorg
4895  root   I    G     20   S  1.7   2.1M   59.4M      gnome-shell
5962  spx    I    G     20   S  9.5   7.3M   53.1M      chromium-browse
5587  spx    I    G     20   D  9.3   12.4M  44.4M      compiz
4520  root   I    G     20   S  0.3   1.8M   28.6M      Xorg
4011  root   I    G     20   S  0.0   13k    76k        nvargus-daemon
1ALL  2GPU  3CPU  4MEM  5ENG  6CTRL  7INFO  Quit
```

Fig. 6. GPU Utilization and Memory Consumption while running in docker.

VIII. Conclusion

The paper has presented the successful implementation of video dehazing model based on AOD-Net utilizing Docker on Jetson Nano platform. The implementation was able to remove the haze from hazy images effectively while maintaining the computational efficiency even while running on a Docker container. The process of containerization using Docker simplifies the deployment process and ensures consistency across various platforms. The achieved results demonstrate the ease of using this approach in various practical scenarios on embedded systems.

IX. Reference

[1] Gao, K., Tu, H., Sun, L., Sze, N. N., Song, Z., and Shi, H., “Impacts of reduced visibility under hazy weather condition on collision risk and car-following behavior: Implications for traffic control and management”, International Journal of Sustainable Transportation, vol. 14, no. 8, pp. 635–642, 2020.

[2] Riaz, Samia & Anwar, Muhammad & Riaz, Irfan & Kim, Hyun-Woo & Nam, Yunyoung & Khan, Muhammad. (2022). Multiscale Image Dehazing and Restoration: An Application for Visual Surveillance. Computers, Materials and Continua.

[3] Yin, Yifan & Cheng, Xu & Shi, Fan & Zhao, Meng & Li, Guoyuan & Chen, Shengyong. (2022). An Enhanced Lightweight Convolutional Neural Network for Ship Detection in Maritime Surveillance System. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[4] Y. Meng and H. Wu, "Highway Visibility Detection Method Based on Surveillance Video," 2019 IEEE 4th International Conference on Image, Vision and Computing (ICIVC), Xiamen, China, 2019, pp. 197-202.

[5] Ryan Wen Liu, Weiqiao Yuan, Xinqiang Chen, Yuxu Lu, "An enhanced CNN-enabled learning method for promoting ship detection in maritime surveillance system",Ocean Engineering, Volume 235, 2021.

[6] Yadav, Anuj & Garg, M. & Mehra, Ritika. (2019). Docker Containers Versus Virtual Machine-Based Virtualization: Proceedings of IEMIS 2018, Volume 3.

[7] Abha Anand B., Darunya B. C., Anisha B. S., S. G. Raghavendra Prasad. A survey on docker container and its use cases, International Journal of Advance Research, Ideas and Innovations in Technology, www.IJARIIT.com.

[8] Jeong, Chi Yoon & Moon, KyeongDeok & Kim, Mooseop. (2023). An end-to-end deep learning approach for real-time single image dehazing. Journal of Real-Time Image Processing.

[9] He, K., Sun, J., Tang, X.: Single image haze removal using dark channel prior. IEEE Trans. Pattern Anal. Mach. Intell. 33(12), 2341–2353 (2011).

[10] Ren, W., Liu, S., Zhang, H., Pan, J., Cao, X., Yang, M.-H.: Single image dehazing via multi-scale convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer vision—ECCV 2016, pp. 154–169. Springer, Cham (2016).

[11] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016).

[12] Chen, W., Gong, X., Liu, X., Zhang, Q., Li, Y., Wang, Z.: Fasterseg: Searching for faster real-time semantic segmentation. In: International Conference on Learning Representations (2020).

[13] Dong, H., Pan, J., Xiang, L., Hu, Z., Zhang, X., Wang, F., Yang, M.-H.: Multi-scale boosted dehazing network with dense feature fusion. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition(CVPR) (2020).

[14] Li, B., Peng, X., Wang, Z., Xu, J., Feng, D.: Aod-net: All-in-one dehazing network. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 4780–4788 (2017).

[15] H. Koschmieder, “Theorie der horizontalen sichtweite,” Beitrage zur Physik der Freien Atmosphare, vol. 12, pp. 33–53, 1924.

[16] W. Wang, "Research on Using Docker Container Technology to Realize Rapid Deployment Environment on Virtual Machine," 2022 8th Annual International Conference on Network and Information Systems for Computers (ICNISC), Hangzhou, China, 2022, pp. 541-544.

[17] B. Cai, X. Xu, K. Jia, C. Qing and D. Tao, "DehazeNet: An End-to-End System for Single Image Haze Removal," in IEEE Transactions on Image Processing, 2016.

[18] H. Ullah et al., "Light-DehazeNet: A Novel Lightweight CNN Architecture for Single Image Dehazing," in IEEE Transactions on Image Processing, vol. 30, pp. 8968-8982, 2021.

[19] Q. Zhu, J. Mai and L. Shao, "A Fast Single Image Haze Removal Algorithm Using Color Attenuation Prior," in IEEE Transactions on Image Processing, vol. 24, no. 11, pp. 3522-3533, Nov. 2015.

[20] N. Silberman, D. Hoiem, P. Kohli and R. Fergus, "Indoor Segmentation and Support inference from rgbd images," in European Conference on Computer Vision, 2012.

[21] X. Zhang, "Learning to Restore Hazy Video: A New Real-World Dataset and A New Method," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 2021.

[22] Y. Zhang, L. Ding and G. Sharma, "HazeRD: An outdoor scene dataset and benchmark for single image dehazing," in IEEE International Conference on Image Processing (ICIP), Beijing, China, 2017.

[23] K. Tang, J. Yang, and J. Wang. Investigating haze-relevant features in a learning framework for image dehazing. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2995–3000, 2014.

[24] G. Sahu, A. Seal, D. Bhattacharjee, M. Nasipuri, P. Brida and O. Krejcar, "Trends and Prospects of Techniques for Haze Removal From Degraded Images: A Survey," in IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 6, no. 4, pp. 762-782, Aug. 2022.

[25] R. P. Kumar and N. M. Naik, "A Review on Efficient state-of-the-art Image Dehazing Techniques," 2022 International Conference on Innovations in Science and Technology for Sustainable Development (ICISTSD), Kollam, India, 2022, pp. 115-121.

[26] S. K. Nayar and S. G. Narasimhan, "Vision in bad weather," Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 820-827.

[27] Suo, H.; Guan, J.; Ma, M.; Huo, Y.; Cheng, Y.; Wei, N.; Zhang, L. Dynamic Dark Channel Prior Dehazing with Polarization. Appl. Sci. 2023, 13, 10475.