International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 01, January-2022

Robust Multi-Sensor Facial Recognition in Real Time using Nvidia DeepStream

Dr. Saurabh Gupta
Scientist-G
National Informatics Centre
New Delhi, India

Shri Piyushank Gupta
Scientist-C
National Informatics Centre
Lucknow, Uttar Pradesh, India

Shri Anup Kumar
Scientist-C
National Informatics Centre
New Delhi, India

Mohd. Wasim
Technology Consultant
N.I.C.S.I
New Delhi

***Abstract*—Parallel processing of multi-sensor face recognition system had always been a challenge in terms of processing speed. Making best utilization of GPU time and memory requires lot of efforts and hyper-tuning of face recognition inference parameters. This paper presents the scaling up of the Face Recognition System [5] using Nvidia DeepStream SDK to make it more robust over multiple camera processing simultaneously and to generate insights in real time. An end to end solution that can work in an integrated fashion has been achieved which can handle multiple live feeds through a single instance and is contained in a single application. Ease of deployment and best optimization of models have made face recognition pipeline lightweight and has scaled the processing speed on a low-end device like Jetson. DeepStream SDK enabled the solution to be as configurable as possible so that a new instance can quickly adapt to the new GPU environment with minimum efforts. This paper explains the process adopted in achieving the solution mentioned and also tried to find out the cost of reduction in deployment of face recognition system, discussed in further sections.**

Keywords*— *Face Detection, Face Tracking, Face Recognition, FaceNet, FaceDetect model, Nvidia, DeepStream

I. INTRODUCTION

With advent of DNNs (Deep Neural Networks), it has not only been possible to increase accuracy of Inference but also to accelerate the Inference over GPU environment. This acceleration is very much needed to increase the cost effectiveness of computer vision applications deployment to solve real-life computer vision problems in real time. A face recognition system consists of two DNNs in pipeline; a face detector model followed by face embeddings extractor model. Face detector finds minimum bounding boxes that contain human faces from a video frame. This bounding box is supplied to face embeddings extractor which creates a vector for each face. For a robust face recognition system, both DNNs needs to be optimized and should work with good accuracy. Availability of pre-trained and pre-optimized models like FaceDetect from Nvidia has helped in detecting faces at greater speeds and greater accuracy. State of the art model like FaceNet [1] (for face embeddings extraction from detected faces) is quite heavy to load into memory and has to be optimized. Now, the major problem is how to consider benefit-cost ratio to deploy a face recognition system because the deep learning algorithms needs GPU to process data in real time. Since GPUs are costly, DNNs need to be optimized and pruned so that face recognition pipeline can process more sensor data in real time. Processing multiple sensor data in real time is another challenge which needs fast parallel processing and multithreaded approaches to build a robust face recognition system. Today Nvidia not only provides advanced GPUs but also developed SDK known as DeepStream SDK to build an end to end pipeline to implement and deploy a video analytics system. Using DeepStream, the solution developed is able to handle multiple sensors using a single GPU or multiple GPUs. This is achieved using hardware acceleration at different stages of pipeline and optimizing the models being loaded into the pipeline for the GPU being used for inference. The SDK also helped us to track faces in a video so that the same face detected in subsequent frames need not to be sent for further processing again.

Figure 1.1 shows the block sequence of activities that have been used to build facial recognition system. A facial recognition system first detects faces from video frames after little pre-processing like noise reduction to make pixel data clearer and doing video frames resizing to the required size. This follows the face detection stage which uses FaceDetect model which is a pre-built and pruned model. The model gave better results on RGB images and smaller faces. After detection of face, the face tracker tracks faces detected by previous stage in subsequent video frames. This is done to avoid processing same face again and again in the later stages of pipeline. The tracking stage outputs the cropped faces being tracked, assigns them a unique identifier and gives cropped faces to face recognition stage. Face recognition stage uses state of the art architecture FaceNet [1] based available model for recognizing faces against a pre-available watch list of faces. A watch list is nothing but a collection of face embeddings extracted from suspect images in advance and saved in a binary file format.

This file consists of pairs of label and face embedding. A label is the unique name assigned to a face embedding. The last stage is the classification and alert generation based on the matches that lie above a threshold percent match. This stage can be connected to either an on-screen display or a message broker like Kafka to send alerts back to a central processing server. The innovative components of the solution are:

- Usage of Nvidia DeepStream to scale a single application to handle at least 4-5 IP camera on a low-end edge computing device like Jetson NANO.
- Optimization of FaceNet model for extracting face embeddings from detected faces. This includes conversion of original FaceNet Keras model .h5 format to ONNX format (DeepStream supported model format) and optimization of model using various techniques discussed in section II (Methodologies).

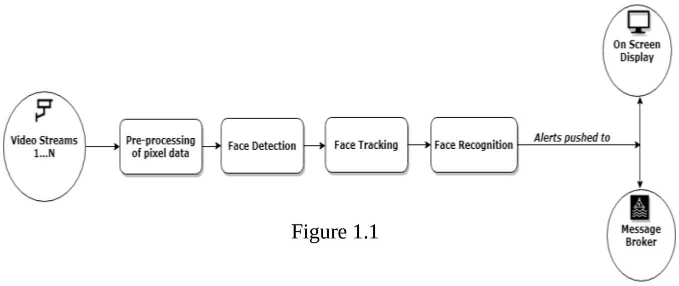


Figure 1.1

II. METHODOLOGIES USED

To handle multiple streams and increase speed of processing by processing frames in parallel for facial recognition system, following methods were followed:

A. Model Optimization and High-Speed Face Detection & Recognition

Initially, Dlib library has been taken and its HoG based model [7] is used for detecting faces but Dlib's HoG face descriptor model was having many drawbacks. The HoG based face detector does not make use of deep learning approach and can't be accelerated over GPU. Also, HoG face detector can only detect frontal faces with a good accuracy from good quality multimedia only. To leverage hardware acceleration for handling already installed IP cameras at premise which captures faces at wide variety of angles, it was decided to go with Dlib's CNN based model [7]. After this step Dlib's CNN was tested which was RESNET29 [2] that detects faces with great accuracy and is able to detect smaller faces and at odd angles. Dlib based face detector [2] was still heavy for optimization on a low-end device like Jetson NANO. The unpruned model was used for training purpose but for efficient and real time inferencing, models needed to be optimized. Hence, a pruned model was taken from Nvidia NGC for face detection which was highly optimized. The model is based on Nvidia DetectNet_v2 detector with ResNet18 as a feature extractor. The model from Nvidia is already optimized for production deployment but following optimizations techniques have been used to compress FaceNet model for fast inference with some level of compromised accuracy:

- Model conversion in DeepStream native format: The Keras .h5 format VGG based FaceNet model is taken (which is the original format of FaceNet) and converted to ONNX format which is supported natively by DeepStream [6].
- Model pruning: Pruning is the process of reducing the number of weights from the model. This is done using DeepStream by removing unnecessary connections in the FaceNet network followed by removal of unnecessary neurons from it. DeepStream does it automatically from ONNX formatted model.
- Model quantization: Quantization is the process of reducing range of model weights. This is done to reduce FP32 (floating point 32) precision of model weights to FP16 or INT8. This is done to increase the speed of tensor mathematical operations at the cost of some degradation in accuracy. In our case, FP16 has been chosen for quantizing model to be deployed on Jetson Nano.

DeepStream exposes configurable settings file to achieve all steps as discussed above. For each stage of the pipeline, there is a separate configuration file. Model optimization process results in the generation of an engine file which is native of TensorRT. DeepStream uses TensorRT for the purpose of model optimization for development of high performing machine learning inference.

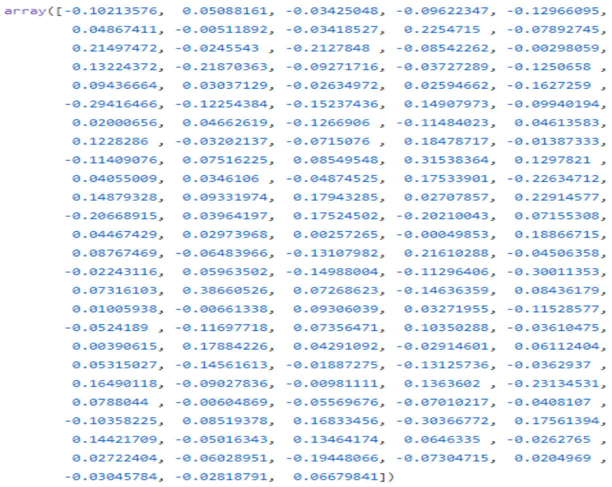
The experiment achieved speed of 10 fps for face detection and face recognition over Jetson Nano with FP16 (floating point precision 16) calibration with batch-size of 4. Batch-size must be equal to the number of sensors connected. This model accepts 736x416x3 dimension input tensors and outputs 46x26x4 bbox coordinate tensor and 46x26x1 class confidence tensor.

B. More Reduced Processing using Face Tracking Technique

Face tracking is being done to keep track of unique faces detected from FaceDetect model and pass them to subsequent face recognition step. Since the face recognition phase using FaceNet is much heavier (even after optimization) as compared to detector and due to complexities in human face embeddings extraction process and their 128-D vector representation, tracker is placed between detector and recognizer. Tracker tracks same faces in subsequent frames and assigns a unique id to each detected face so that only a single and unique cropped face against each person, seen in a batch is sent to recognizer for identification of faces. This ultimately improves throughput of entire DeepStream pipeline [3].

C. Face Recognition using Optimised FaceNet (as optimized in step B) and Machine Learning Based Classifier

Face recognition is the process of identifying faces from face vectors (also known as face embeddings) which is a two-step process. At first step, FaceNet, as optimized in step B, has been used to generate these vectors from live video frames that generates 128-D embeddings (figure 2.1) which uniquely defines a face. The optimized FaceNet has shown a considerable increase in speed in generating face embeddings over Jetson Nano. A sample 128-D embedding is as given:



```
array([-0.10213576,  0.05088161, -0.03425048, -0.09622347, -0.12966095,
        0.04867411, -0.00511892, -0.03418527,  0.2254715 , -0.07892745,
        0.21497472, -0.0245543 , -0.2127848 , -0.08542262, -0.00298059,
        0.13224372, -0.21870363, -0.09271716, -0.03727289, -0.1250658 ,
        0.09436664,  0.03037129, -0.02634972,  0.02594662, -0.1627259 ,
       -0.29416466, -0.12254384, -0.15237436,  0.14907973, -0.09940194,
        0.02000656,  0.04662619, -0.1266906 , -0.11484023,  0.04613583,
        0.1228286 , -0.03202137, -0.0715076 ,  0.18478717, -0.01387333,
       -0.11409076,  0.07516225,  0.08549548,  0.31538364,  0.1297821 ,
        0.04055009,  0.0346106 , -0.04874525,  0.17533901, -0.22634712,
        0.14879328,  0.09331974,  0.17943285,  0.02707857,  0.22914577,
       -0.20668915,  0.03964197,  0.17524502, -0.20210043,  0.07155308,
        0.04467429,  0.02973968,  0.00257265, -0.00049853,  0.18866715,
        0.08767469, -0.06483966, -0.13107982,  0.21610288, -0.04506358,
       -0.02243116,  0.05963502, -0.14988004, -0.11296406, -0.30011353,
        0.07316103,  0.38660526,  0.07268623, -0.14636359,  0.08436179,
        0.01005938, -0.00661338,  0.09306039,  0.03271955, -0.11528577,
       -0.0524189 , -0.11697718,  0.07356471,  0.10350288, -0.03610475,
        0.00390615,  0.17884226,  0.04291092, -0.02914601,  0.06112404,
        0.05315027, -0.14561613, -0.01887275, -0.13125736, -0.0362937 ,
        0.16490118, -0.09027836, -0.00981111,  0.1363602 , -0.23134531,
        0.0788044 , -0.00604869, -0.05569676, -0.07010217, -0.0408107 ,
       -0.10358225,  0.08519378,  0.16833456, -0.30366772,  0.17561394,
        0.14421709, -0.05016343,  0.13464174,  0.0646335 , -0.0262765 ,
        0.02722404, -0.06028951, -0.19448066, -0.07304715,  0.0204969 ,
       -0.03045784, -0.02818791,  0.06679841])
```

Figure 2.1

The second step of the recognition is to compare each unknown embedding with a list of already generated known embeddings. Already generated embeddings are stored in a file format where each face embedding has a unique assigned label. This file can be called as watch-list.

The combined FPS of face detection, tracking and face recognition is 10 from 224*224 video frames on attaching 4 IP cameras.

D. *DeepStream SDK*

With DeepStream SDK, AI has been applied to streaming video and simultaneously optimized video decode/encode, image scaling & conversion and edge-to-cloud connectivity for complete end-to-end performance optimization. DeepStream offers exceptional throughput for a wide variety of object detection, image classification and instance segmentation-based AI models. DeepStream is a readymade bundled toolkit which exposes configuration files for defining various pipeline [3] parameters like batch size, precision (INT8, FP16, FP32). DeepStream lets us optimize the model by creating a model engine execution plan which makes best use of available CUDA cores on the same GPU over which the inference has to be done. The creation of engine file is one-time process for a particular GPU against a particular batch size. DeepStream uses C++ written plug-in(s) which works in a pipeline [3] to achieve high speed video decoding (reading from file or camera). It helps in preparing batches, skipping frames, tracking camera, their frames and objects in these frames, and pushing results over a message broker to achieve an end to end face recognition system making the best use of available GPU resources. Using these configurable plug-in(s), a high-performance face recognition pipeline [3] has been built which can run as a single instance to handle multiple sensor data.

III. Detailed ARCHITECTURE AND improvements

All processing related to the implementation of multi-stream facial recognition system has been done on Nvidia Jetson Nano. The major platform and runtimes used to build DeepStream 5.1 based face recognition system includes;

- JetPack 4.5.1 – Nvidia JetPack SDK is the most comprehensive solution for building AI applications. It includes the latest OS images for Jetson products, along with libraries and APIs, samples, developer tools, and documentation.
- CUDA 10.2.89 – CUDA is a parallel programming interface (collection of APIs) that lets your code communicate with the Nvidia based GPU architecture.
- cuDNN 8.0.0+ - It is a collection of CUDA (Computed Unified Device Architecture) libraries for deep neural networks that are hardware accelerated)
- TensorRT 7.1.3 – TensorRT is core library which optimizes TensorFlow, PyTorch, Keras, and other compatible deep learning models for Nvidia DeepStream framework. It re-architects trained models for CUDA compatible Nvidia GPU and dGPU like A100, V100 as well as edge computing device like Jetson

A. *Application Architecture*

The high-performance pipeline as in figure 3.1 [3] starts with decoding process which accepts H.264 and H.265 format elementary streams which are the formats that support hardware acceleration. Today, most of the sensors support these formats (also known as elementary stream). An elementary stream doesn't have audio data along with it.

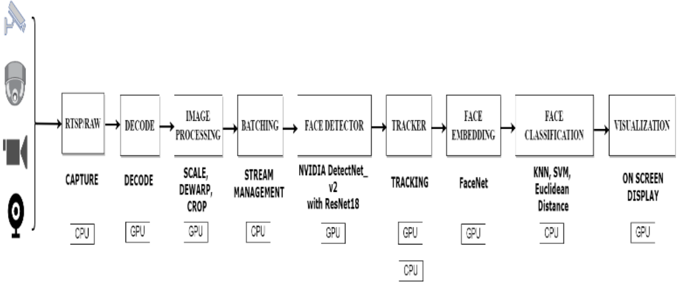


Figure 3.1

Image processing is also hardware accelerated using DeepStream plug-in written in C++.

Batching is an important step because GPU processes data in batches. Because of this reason, powerful GPU(s) are made with abundance of CUDA cores. Nvidia Jetson Nano has been used which is having 128 CUDA cores. For best performance and as recommended, batch size is kept equal to the number of IP camera(s) attached to the device. In our case batch-size of 4 has been used. DeepStream batching process (also known as multiplexing) takes care of which face is detected from which frame and which frame belongs to which sensor. This is required so that alerts can be generated properly.

After batching, frames are passed to CUDA cores for detection of faces in multiple streams simultaneously. Nvidia based FaceDetect model is loaded into memory and detection is done at high speeds. The model has already been optimised by creating engine file for the same GPU. DeepStream uses TensorRT SDK to do these optimizations.

Tracker has been placed after detector which is again a GPU accelerated stage so that only one cropped face for a detected human face is being sent to recognizer.

After tracker and before visualization, one more DNN, known as FaceNet [1] is deployed which is optimized like detector model using DeepStream SDK's engine file creation for this particular GPU.

Visualization is achieved using Nvidia NVOSD plug-in. OSD stands for on screen display which is GPU accelerated stage.

Only the batching (also known as multiplexing) is taken care of by CPU. Other stages are able to be accelerated over GPU. This makes pipeline very fast. Both models; detector and recogniser are optimised well in advance, before the inference pipeline has started. For detector, TLT (Nvidia transfer learning toolkit) format model is used. For recogniser, VGGFace2 pre-trained FaceNet model in ONNX format has been used.

B. *Accuracy Improvements*

Accuracy of a face recognition system depends on the overall accuracy of the detector model, FaceNet model and finally the matching algorithm that is used to match two face embeddings generated by FaceNet. Since Nvidia provided pre-trained model known as FaceDetect has been used, which is very accurate at detecting faces from video frames, accuracy of face detection is 96%. FaceDetect detects smaller faces (faces greater than 10% of the image area) at odd angles with high confidence.

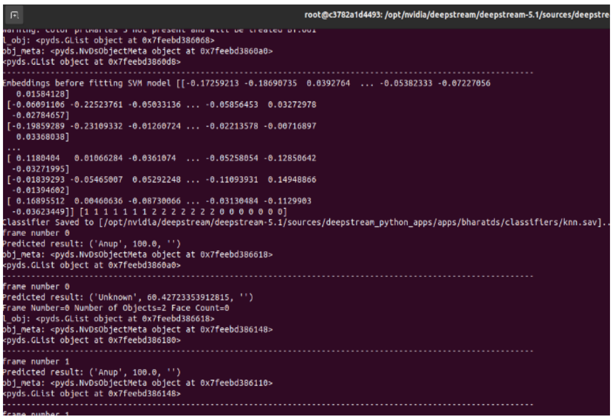
VGGFace2 pre-trained FaceNet model has been used which is able to touch accuracy of almost 100% on YALE, JAFFE, AT & T datasets. Accuracy of FaceNet [1] while creating face vectors (face embeddings) from video frames depends on many factors like percentage of person's face with respect to frame size, degree of sharpness, environmental conditions, occlusion level, angle of the face etc. Hence, there is a need to pre-process detected faces before passing it to FaceNet. Following methods to pre-process detected faces:

- Normalization: To improve the lighting condition of the cropped faces
- Face alignment: Aligning faces to straighten them horizontally with respect to eyes, nose, and other face landmarks.
- Face Resizing: Up-scaling face up to the required level so that better embeddings can be obtained.
- Gray Scale Conversion: Simply converting cropped faces to gray scale so as to increase accuracy because color information of detected faces is not required.

The last step is to match known face embeddings with unknown embeddings. Known embeddings were already placed in a watch-list and unknown embeddings are expected to come from DeepStream pipeline from live feeds. Following methods to do the comparison and corresponding accuracies have been used:

- Euclidean Distance: Euclidean distance did not work well and is basically used for face verification. Since, our work is related to face classification; this is not appropriate and reliable metric for identifying faces from live feeds. Hence, multiple images per person in watch-list are needed during model building phase for good accuracy. In those cases where multiple images per person are not available, image augmentation techniques to generate synthetic images from a single image like horizontal flip, scaling, change in lighting conditions etc. can be used. The next two approaches work well with multiple images per person.
- K-nearest neighbors (KNN): KNN is a simple, supervised learning classification algorithm which takes multiple images per person to train a KNN classifier which is then used to predict matches. The accuracy of prediction achieved (using 20 images per person for creating watch-list) was 65%.

- Support Vector Machines (SVM): Another supervised learning approach used was SVM (Figure 3.2) which works in a similar way to KNN. The accuracy of prediction (using 20 images per person for creating watch-list) was 69%.


root@c3782a1d4493: /opt/nvidia/deepstream/deepstream-5.1/sources/deepstre

```
Warning: Color primaries 5 not present and will be created BT.601
l_obj: <pyds.GList object at 0x7feebd386068>
obj_meta: <pyds.NvDsObjectMeta object at 0x7feebd3860a0>
<pyds.GList object at 0x7feebd3860d8>
-----------------------------------------------------------------------------------------------
Embeddings before fitting SVM model [[-0.17259213 -0.18690735  0.0392764  ... -0.05382333 -0.07227056
   0.01584128]
 [-0.06091106 -0.22523761 -0.05033136 ... -0.05856453  0.03272978
  -0.02784657]
 [-0.19859289 -0.23109332 -0.01260724 ... -0.02213578 -0.00716897
   0.03368038]
 ...
 [ 0.1180404   0.01066284 -0.0361074  ... -0.05258054 -0.12850642
  -0.03271995]
 [-0.01839293 -0.05465007  0.05292248 ... -0.11093931  0.14948866
  -0.01394602]
 [ 0.16895512  0.00460636 -0.08730066 ... -0.03130484 -0.1129903
  -0.03623449]] [1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 0 0 0]
Classifier Saved to [/opt/nvidia/deepstream/deepstream-5.1/sources/deepstream_python_apps/apps/bharatds/classifiers/knn.sav]..
Frame number 0
Predicted result: ('Anup', 100.0, '')
obj_meta: <pyds.NvDsObjectMeta object at 0x7feebd386618>
<pyds.GList object at 0x7feebd3860a0>
-----------------------------------------------------------------------------------------------
Frame number 0
Predicted result: ('Unknown', 60.42723353912815, '')
Frame Number=0 Number of Objects=2 Face Count=0
l_obj: <pyds.GList object at 0x7feebd386618>
obj_meta: <pyds.NvDsObjectMeta object at 0x7feebd386148>
<pyds.GList object at 0x7feebd386180>
-----------------------------------------------------------------------------------------------
Frame number 1
Predicted result: ('Anup', 100.0, '')
obj_meta: <pyds.NvDsObjectMeta object at 0x7feebd386110>
<pyds.GList object at 0x7feebd386148>
-----------------------------------------------------------------------------------------------
Frame number 1
```

Figure 3.2

ratio while considering implementation of a face recognition system from multiple sensors.

IV. RESULT SCREENS OF EXPERIMENT

Figure 4.1 demonstrates RTSP webcam stream through DeepStream face recognition pipeline on Jetson Nano:

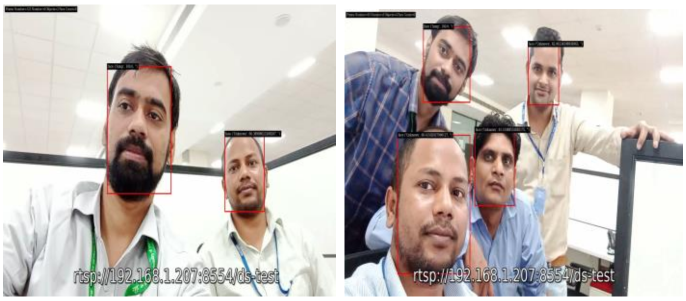

rtsp://192.168.1.207:8554/ds-test
rtsp://192.168.1.207:8554/ds-test

Figure 4.1

Figure 4.2 demonstrates live CCTV stream through DeepStream face recognition pipeline on Jetson Nano:

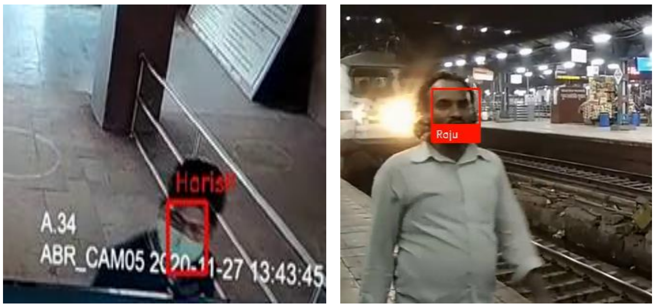


Figure 4.2

Jetson Nano set up (Figure 4.3):

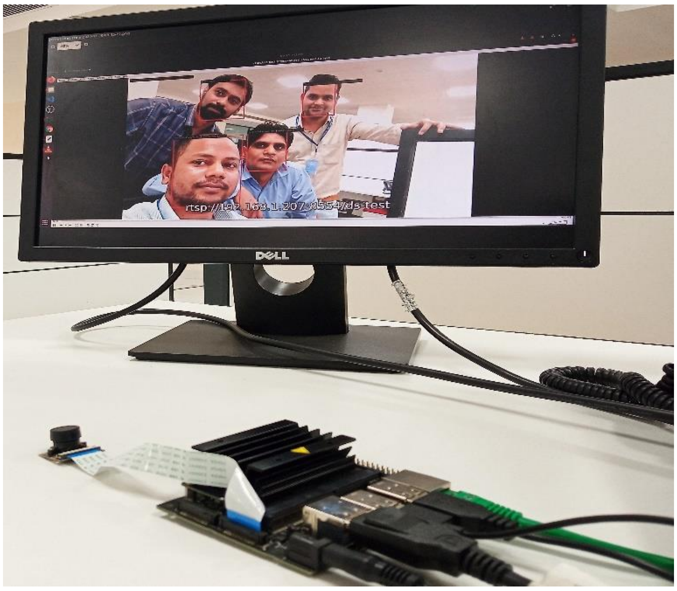


Figure 4.3

V. CONCLUSION

The use of DeepStream SDK has scaled up face recognition pipeline that can not only works in an integrated fashion to fulfil all the tasks like video decoding, batching, running deep neural networks and then collecting results [3]. All this has been done simultaneously from multiple live feeds in an integrated way. Only a single instance running on an edge device (having 128 CUDA cores) is now capable to handling at least 4 live streams in real time which was not possible without using hardware acceleration at various stages of pipeline. This shows that cost factor has also been brought down by factor of 4. Also, model optimization has been achieved to make inference faster over the GPU used for inference. This leads to a better benefit-cost

VI. FUTURE WORK

The following features which can make face recognition more robust and that is in our short-term plan are as follows:

- Addition and deletion of sensors in a live fashion. This feature lets the pipeline notified about a new camera attached or detached from the network. This will let the pipeline continuously running and need not to be re-configured and restarted.
- Use of optical flows in face recognition pipeline [4]. Nvidia GPUs, starting with the Nvidia GPU Turing generation and Jetson Xavier generation, contain a hardware accelerator for computing optical flow. Optical flow vectors [4] are useful in various use cases such as object detection and tracking, video frame rate up-conversion, depth estimation, stitching, and so on.

ACKNOWLEDGEMENTS

The work described herein has been inspired from DeepStream and TensorRT from NVIDIA [https://developer.nvidia.com/deepstream-sdk]. The work is an extended approach of sample python apps available at https://github.com/NVIDIA-AI-IOT/deepstream_python apps. The python bindings available for DeepStream was used which was an extension of Pybind11. Thanks to NVIDIA NGC [https://ngc.nvidia.com] for providing base container and base model for face detection to achieve the high-speed inference.

REFERENCES

[1] Florian Schroff ; Dmitry Kalenichenko ; James Philbin ; FaceNet: A Unified Embedding for Face Recognition and Clustering retrieved from arXiv:1503.03832v3 [cs.CV] 17th June, 2015

[2] Davis E. King; Dlib-ml: A Machine Learning Toolkit, Journal of Machine Learning Research 10 (2009) 1755-1758 Submitted 10/08; Revised 4/09; Published 7/09.

[3] MyungJoo Ham; Ji Joong Moon; Geunsik Lim; Wook Song; Jaeyun Jung; Hyoungjoo Ahn; Sangjung Woo; Youngchul Cho; Jinhyuck Park; Sewon Oh; Hong-Seok Kim; NNStreamer: Stream Processing Paradigm for Neural Networks, Toward Efficient Development and Execution of On-Device AI Applications, retrieved from arXiv:1901.049885v1 [cs.DC] 12th January, 2019.

[4] C. Hsieh, S. Lai and Y. Chen, "An Optical Flow-Based Approach to Robust Face Recognition Under Expression Variations," in IEEE Transactions on Image Processing, vol. 19, no. 1, pp. 233-240, Jan. 2010, doi: 10.1109/TIP.2009.2031233.

[5] Advances in Intelligent Systems and Computing Image Processing and Capsule Net, 2021, Volume 1200, ISBN: 978-3-030-51858-5.

[6] Keras to ONNX conversion available at https://github.com/riotu-lab/tf2trt_with_onnx.

[7] Dlib models available at https://github.com/davisking/dlib-models.