www.ijaceeonline.com ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 9, September 2022, pp. 1 – 8

Inference Speed Optimization of Deep Neural Network on Heterogeneous Devices

Mohammad H. Ismail
Department of Technical Computer Engineering, Al-Hadba University College, Iraq
Email: mohammadhaqqi@hcu.edu.iq

Shefa A. Dawwd
Department of Computer Engineering, University of Mosul, Iraq
Email: shefa.dawwd@uomosul.edu.iq

Fakhrulddin H. Ali
Department of Computer Engineering, University of Mosul, Iraq
Email: fhazaa@uomosul.edu.iq

Abstract: *A set of models used in detection and recognition of sign language gestures are proposed to increase the speed of inference by the concept of optimization using the TensorRT inference accelerator tool. This is done by using different devices: Labtop, Jetson Xavier Nx, and Cloud Computing. When TF-TRT Integration optimization is applied, the speed of Inference increases about 3x-11x for static sign language recognition models and about 1x-3x for dynamic sign recognition models. While when using optimization of applied TensorRT (TensorRT C++ API), the speed of Inference increases about 14x-110x for static sign language recognition models and about 2x- 10x for dynamic sign recognition models.*

Keyword: *Deep Learning; Inference; Hetrogeneous; Optimization; TensorRT*

1. INTRODUCTION

Detecting hand gestures in real time is a challenging difficult problem. The presence or absence of a sign in addition to determining the hand gesture if it is static or dynamic should be done as quickly and accurately as possible [1]. Then recognizing hand gestures in real time quickly and accurately is done. All of these are in response to a sign recorded by a camera that captures a series of video frames in different lighting conditions[2]. This is done using machine learning methods to detect and to recognize gestures and translate them into text. The accuracy in detecting and recognizing the static or dynamic hand gesture is not only the problem, but the speed factor is very necessary in applications with time constraints. There is a possibility to use modern architectural systems in computers, in order to solve the problem of execution time, and improve the efficiency of the speed, by relying on multi-core CPUs and the GPU, and provides the ability to perform tasks in parallel with each other[3]. The objective of this research is to speed up the inference on GPU and CPU by using different devices for: a- Detection model. b- Static single and Multi-Models recognition. c- Dynamic single and Multi-Models recognition. Heterogeneous computing is a programming paradigm that distributes computational tasks across several processor types to complete one work quickly and accurately. Central processing units (CPUs) and graphics processing units (GPUs) are two common processor types that may be used by a heterogeneous application to do portions of the required calculations on the CPU and other parts on the GPU. To maximize power and performance, heterogeneous computing uses a variety of processor types. GPUs are computing systems with many compute cores that enable highly parallel computation and were initially designed for graphics processing. GPUs are made up of various computing units, each of which runs several threads in a lock-step method. For processing, heterogeneous computing uses both the CPU and the GPU. A GPU aggressively executes context switching to hide memory access latency, as opposed to a CPU, which employs a hierarchy of memory to speed up memory access. As a result, the GPU is best in parallel data applications, whereas the CPU specializes in general-purpose computing [4].

There are several single and multi-models used in the detection and recognition of static and dynamic gestures in Arabic sign language Table I. Heterogeneous devices are used for the purpose of evaluating the optimization of inference.

Cite this paper:

Mohammad H. Ismail, Shefa A. Dawwd, Fakhrulddin H. Ali, "Inference Speed Optimization of Deep Neural Network on Heterogeneous Devices", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 9, pp. 1-8, September 2022.

TABLE I MODELS FOR RECOGNITION & DETECTION OF ARABIC SIGN LANGUAGE.

MODEL		Reference
Static Sign Language Recognition	Single-model: DenseNet121(DEPTH)	[5]
	Single-model: VGG16(RGB)	
	Multi-model: DenseNet121(DEPTH)& VGG16(RGB)	
Dynamic Sign Language Recognition	Single-model: ResNet50-LSTM(RGB)	[6]
	Single-model: ResNet50-GRU(Depth)	
	Multi-model: ResNet50-BiLSTM-Normalization	
Detection Sign Language	Pose Estimation Distances and Angles Features + Bi-GRU	[7]

2. METHODOLOGY

2.1. Inference on Heterogeneous Device

In Deep Neural Network (DNN), an inference process is an act of feeding new input data into a trained DNN model to create a prediction in terms of values or labels. Several factors affect inference performance in computer vision, including the kind of DNN model, inference engine, feature extraction, and input image size. The measures used to compare and analyse inference performance are as follows[8]: 1. CPU usage: the rate at which the CPU is used during inference. Value is represented as a percentage and ranges from 0% to 100%. 2. GPU usage: the rate at which the GPU is used during inference it's ranges from 0% to 100%. TableII shows the specifications of Devices for training and inference used in this paper.

TABLE II THE SPECIFICATIONS OF STUDY USED DEVICES.

LABTOP	CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
	GPU	GeForce RTX 2060 1920 CUDA cores
JETSON XAVIER NX	CPU	6-core @1.2Ghz
	GPU	NVIDIA Volta™ 1.1Ghz 384-core
CLOUD COMPUTING (KAGGLE)	CPU	Intel(R) Xeon(R) CPU @ 2.00GHz
	GPU	Tesla P100 15.9GB 3584 CUDA cores

2.2 Model Performance Optimization(S)

The goal of this part is to improve model performance by optimization using the TensorRT [9] inference accelerator tool. For real-time assessment and on-board feasibility testing on edge devices, this is primarily for the purpose of increasing the frame per second (FPS) rate. A second benefit is to reduce the onboard memory footprint on the target device by using different optimization techniques. In DNN, an inference process is an act of feeding new input data into a trained DNN model to create a prediction in terms of values or labels. Several factors affect performance of inference in computer vision, including the kind of DNN model, inference engine, feature extraction, and input image size. NVIDIA presented the TensorRT inference accelerator to optimize the trained DNN model for reduced latency and greater throughput inference on NVIDIA devices in order to increase the speed of DNN inference on NVIDIA devices. Layers are combined, kernel selection is optimized, and normalization is used to achieve this optimization. PyTorch, Caffe, and TensorFlow are among the most common machine learning frameworks supported. TensorRT is based on NVIDIA's Compute Unified Device Architecture (CUDA), a parallel programming software platform that runs on top of their GPUs to conduct application computations more quickly and efficiently [10]. A model may be optimized for inference with three tools provided by TensorRT: TF-TRT (TensorFlow-TensorRT) integration, TensorRT C++ API and TensorRT Python API. There are parsers for Caffe, Open Neural Network Exchange (ONNX), and TensorFlow models included in the previous two tools. Models may be created programmatically using the APIs provided by C++ and Python. Nvidia's recommended technique for importing TensorFlow models into TensorRT is TF-TRT [11]. TensorRT C++ API and TF-TRT will be applied in this work. Each approach used on the trained model is briefly described here.

A. TF-TRT Integration: The TF-TRT integration is the most simplest. It may not provide all of TensorRT's optimization advantages, but it does provide a starting point within TensorFlow. Only the portions of TensorFlow model that are compatible with TensorRT optimizations will be optimized using TensorRT optimizations, while the parts that are incompatible will remain as TensorFlow operations. This conversion method additionally saves the model as a TensorFlow Saved Model, allowing to take advantage of TensorRT improvements while maintaining the TensorFlow model [12]. Figure (1) shows the TensorFlow TensorRT workflow.

B. TensorRT using TensorRT C++ API: In this part, it is demonstrated how to use the TensorRT C++ API to make efficient inferences on an existing TensorFlow model. TensorRT may be deployed as

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 9, September 2022, pp. 1 – 8

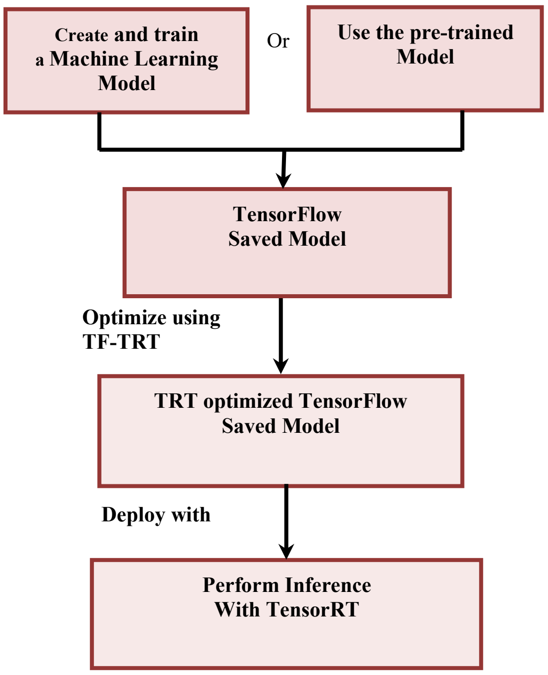
Create and train a Machine Learning Model
Or
Use the pre-trained Model
TensorFlow Saved Model
Optimize using TF-TRT
TRT optimized TensorFlow Saved Model
Deploy with
Perform Inference With TensorRT

Figure (1): TensorFlow TensorRT workflow[13].

shown in Figure (2) by first converting a trained model stored in one of the for mentioned frameworks to an ONNX format. The model would next be parsed by TensorRT's ONNX parser, which would be used to create the TensorRT engine. TensorRT [14] works by applying five forms of optimization strategies for enhancing the throughput of deep neural networks. First phase, it increases throughput by keeping model accuracy while quantizing models to float point

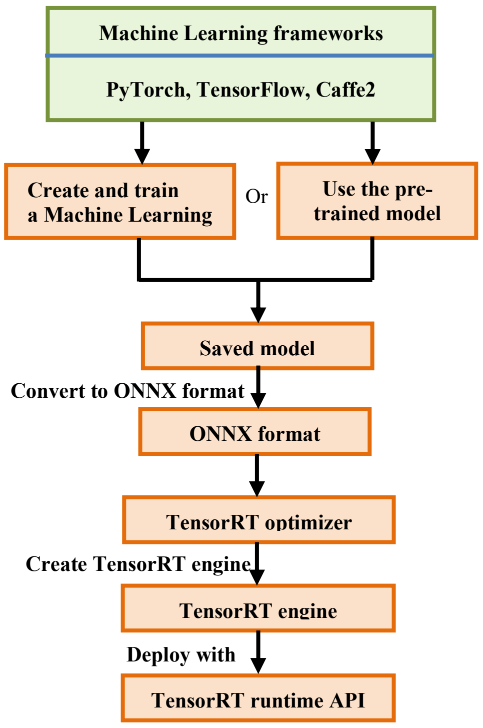

Figure (2): TensorRT workflow [15]

16-bit (FP16) or 8-bit integer (int8) data type precision. This approach considerably decreases the model size because it is changed from initially FP32 to FP16 version. A layer and tensor fusion approach is then used to further optimize the utilization of onboard GPU memory in the next phase. Kernel auto-tuning is the third phase in the process. In this most essential phase, the TensorRT engine narrows down the ideal network layers and batch sizes dependent on the target GPU. By distributing memory to tensors only during the time they are being used, it reduces the memory footprint and reuses memory. The next stage is to handle numerous input streams in parallel and then tune neural networks on a periodic basis using dynamically produced kernels. To get the most out of the Jetson Xavier NX, it is needed to set the power choices to maximum. The commands “sudo nvpmodel -m 0” and “sudo jetson_clocks” may be used to set the Jetson to maximum power use[16]. TensorRT optimizes the network after it has been trained, changing the structure in an indeterminable manner. Three alternative configurations were used in the optimization [17]:

INT8 – weights and activation have INT8 precision.
FP16 – weights and activation have FP16 precision.
FP32 – weights and activation have FP32 precision.

In this paper, we evaluated the suggested technique on an NVIDIA Jetson embedded device with heterogeneous accelerators. An octa-core CPU, a GPU, and two Deep Learning Accelerators DLAs are included on the board. For deep learning processes, the DLA is a fixed-function acceleration engine. DLA's goal is to accelerate convolutional neural networks entirely on hardware. Various layers are supported by DLA. The DLA is an accelerator that uses very little power, but its processing capacity is much lower than that of the GPU.

3. RESULTS AND DISCUSSIONS

Table III shows the Inference speed on GPU and CPU for Sign Detection Model by using different devices laptop, cloud, and Jetson. One can see that the Inference speed on GPU in the case of using the lap-

TABLE III INFERENCE ON GPU AND CPU FOR SIGN DETECTION MODEL BY USING DIFFERENT DEVICES.

DEVICES	INFERENCE ON CPU			INFERENCE ON GPU			Speed up
	FPS	GPU USAGE%	CPU USAGE%	FPS	GPU USAGE%	CPU USAGE%	
LAPTOP	43	0	20	42	19	18	0.98x
JETSON XAVIER NX	11	0	80	50	60	70	4.5x
CLOUD COMPUTING	35	0	100	36	4	90	1.03x

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 9, September 2022, pp. 1 – 8

top or the cloud is close to Inference speed on CPU, while in the case of using the Jetson, the Inference speed on GPU is up to 5x the speed on CPU. The reason for this belongs to the mediapipe that supports the GPU in the Jetson, while it is not possible in other devices, and it runs on the CPU only. It is also noticed that the Jetson is the fastest at Inference speed on GPU. And the laptop is the fastest at Inference speed on CPU.

The Tables IV&V show the Inference on GPU and CPU by using different devices as cloud, laptop, and Jetson for single and multi-models for static and dynamic sign language recognition respectively. In Table IV the case of static sign language recognition models, the Inference speed on GPU is slightly faster than the Inference speed on CPU. It is also noticed that the laptop is the fastest, in both Inference speed on GPU and on CPU. Depending on the specifications of the devices used: Both laptop and Cloud are 4.1x-5.2x faster than the Jetson at Inference speed on GPU. Also, both laptop and Cloud are 3.4x-7.8x faster than the Jetson at Inference speed on CPU.

In Table V the case of dynamic sign language recognition models, the Inference speed on GPU is faster than the speed on CPU, in Jetson it is about 4.7x-5.3x, in the laptop it is 6.3x-9.4x, while in the Cloud computing it is 22.6x-28.2x. It is also noticed that the cloud is the fastest, in the Inference speed on GPU, and the laptop is the fastest, in the Inference speed on CPU. This is because the GPU in the cloud has higher specifications, while in the laptop the CPU is the highest. The laptop is 4.4x-6.7x faster than both the cloud and the Jetson at Inference FPS on GPU. Both the laptop and the cloud are 2.5x-4.3x faster than it is in the Jetson at Inference speed on CPU.

TABLE IV INFERENCE ON GPU AND CPU FOR STATIC SIGN LANGUAGE RECOGNITION BY USING DIFFERENT DEVICES.

DEVICES	MODEL	INFERENCE ON CPU			INFERENCE ON GPU			Speed up
		FPS	GPU USAGE%	CPU USAGE%	FPS	GPU USAGE%	CPU USAGE%	
LAPTOP	Single-model: DenseNet121 (DEPTH)	21.3	0	36	26.5	98	19	1.2x
	Single-model: VGG16 (RGB)	21.7	0	62	34.1	98	19	1.6x
	Multi-model: DenseNet121 (DEPTH) & VGG16(RGB)	16.9	0	66	23.9	96	20	1.4x
JETSON XAVIER NX	Single-model: DenseNet121 (DEPTH)	4.3	0	80	5.2	23	70	1.2x
	Single-model: VGG16 (RGB)	2.8	0	70	6.5	47	70	2.3x
	Multi-model: DenseNet121 (DEPTH) & VGG16(RGB)	2.2	0	80	5.0	58	82	2.3x
CLOUD COMPUTING	Single-model: DenseNet121 (DEPTH)	14.6	0	79	21.3	13	62	1.5x
	Single-model: VGG16 (RGB)	10.9	0	96	28.3	6	62	2.6x
	Multi-model: DenseNet121 (DEPTH) & VGG16(RGB)	8.1	0	100	20.8	13	60	2.6x

TABLE V INFERENCE ON GPU AND CPU FOR DYNAMIC SIGN LANGUAGE RECOGNITION BY USING DIFFERENT DEVICES.

DEVICES	MODEL	INFERENCE ON GPU			INFERENCE ON CPU			Speed up
		FPS	GPU USAGE %	CPU USAGE %	FPS	GPU USAGE %	CPU USAGE %	
LAPTOP	Single-model: ResNet50-LSTM(RGB)	28.8	0	100	180	100	26	6.3x
	Single-model: ResNet50-GRU(Depth)	38.6	0	100	290	96	21	7.5x
	Multi-model: ResNet50-BiLSTM-Normalization	11.6	0	100	109.4	100	23	9.4x
JETSON XAVIER NX	Single-model: ResNet50-LSTM(RGB)	8.8	0	96	41.1	98	20	4.7x
	Single-model: ResNet50-GRU(Depth)	9	0	98	47.4	99	16	5.3x
	Multi-model: ResNet50-BiLSTM-Normalization	4.1	0	100	19.2	99	100	4.7x
CLOUD COMPUTING	Single-model: ResNet50-LSTM(RGB)	11.4	0	96	258	53	51	22.6x
	Single-model: ResNet50-GRU(Depth)	12.5	0	98	297	44	55	23.8x
	Multi-model: ResNet50-BiLSTM-Normalization	4.57	0	100	128.7	68	67	28.2x

The tool TF-TRT Integration was used to optimize inference firstly in the models: Single-model: DenseNet121 (DEPTH), Single-model: VGG16 (RGB), and Multi-model: DenseNet121 (DEPTH) & VGG16 (RGB) for static sign language recognition and secondly in the models: Single model: ResNet50-LSTM (RGB), Single-model: ResNet50- GRU (Depth), and Multi-model: ResNet50-BiLSTM-Normalization for dynamic sign language recognition. This optimization has been done when the executing of these previous

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 9, September 2022, pp. 1 – 8

models by using different devices cloud, laptop, and Jetson. The Figures (3&4) show Fps measured by using different devices cloud, laptop, and Jetson for single multi-models for static and dynamic sign language recognition respectively. In Figure (3) the case of static sign language recognition models when using the optimization, the fastest speed is the cloud. There is an increase in the speed of up to 2.6x-4.6x, in the DenseNet121 (DEPTH) model and the multi-model DenseNet121 (DEPTH) & VGG16 (RGB), while it reaches up to 6.9x-11.3x for the model VGG16 (RGB). This increase as a result of optimization is also shown in the Figure (5). In Figure (4) the case of dynamic sign language, the fastest speed is when using the cloud also. When using the optimization, there is an increase in the speed of up to 1.3x-2.5x. This increase as a result of optimization is also shown in the Figure (6). In the case without the use of optimization, in static recognition models are slow compared to dynamic recognition models, as in the first it deals with single image, while in the second with a series of frames, and after the optimization, the relative increase in the acceleration of static compared to dynamic. Figures (3&4) also confirm that the singe model form is faster than the multi-model.

The tool TensorRT using TensorRT C++ API was used firstly to optimize the inference in the models: Single-model: DenseNet121 (DEPTH), Single-model: VGG16 (RGB), and Multi-model: DenseNet121 (DEPTH) & VGG16 (RGB) for static sign language recognition and secondly in the models: Single-model: ResNet50-LSTM (RGB), Single-model: ResNet50-GRU (Depth), and Multi-model: ResNet50-BiLSTM-Normalization for dynamic sign language recognition. This optimization has been done when executing these models by using Jetson device. The

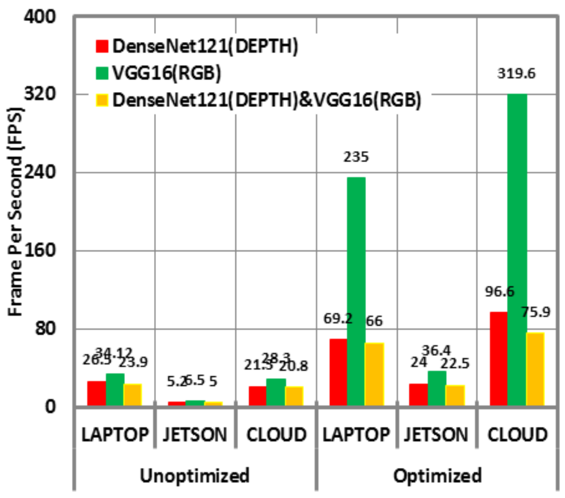

Figure (3): Fps measure for recognition of static sign language for single and multi-models using different devices.

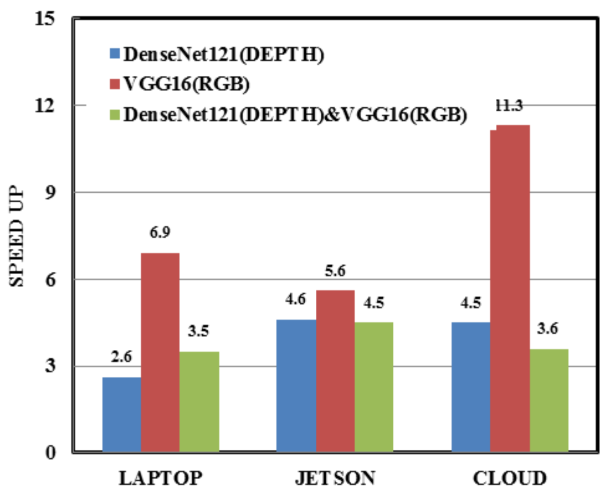

Figure (5): Speed up Inference for Recognition of Static Sign Language for Single and Multi-Models by Use Different Devices.

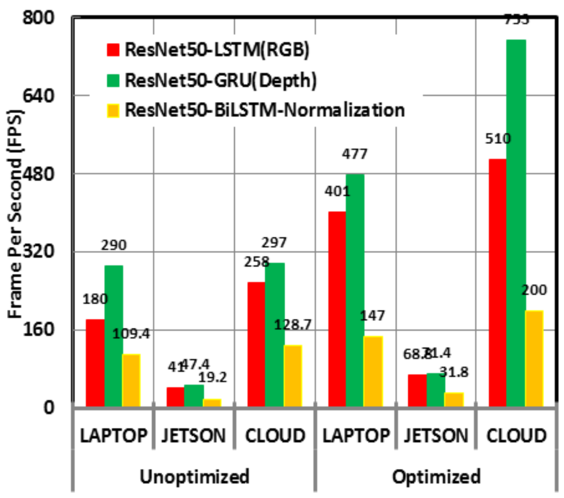

Figure (4): Fps measure for recognition of dynamic sign language for single and multi-models using different devices.

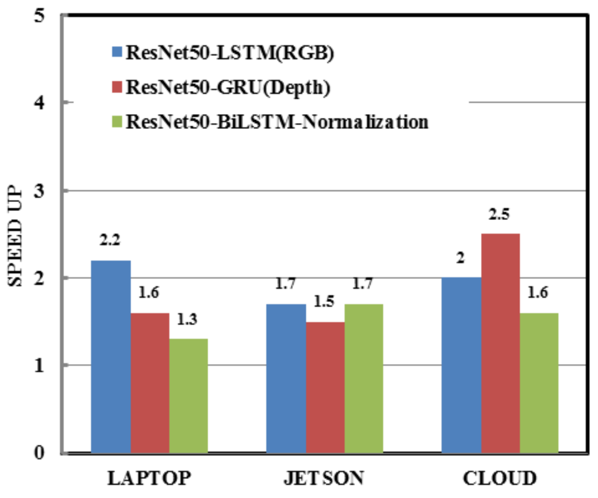

Figure (6): Speed up Inference for Recognition of Dynamic Sign Language for Single and Multi-Models by Use Different Devices.

Figures (7&8) show the Inference Fps on GPU measure by using JETSON device for Static and Dynamic Sign respectively. And the Figures (9&10) show the speed up **of** inference on GPU measure by using JETSON device for static and dynamic sign language recognition respectively.

From the comparison between the Inference speed in the case of static sign language and dynamic sign language, the Inference speed in dynamic sign language is much greater than in static one before the optimization process, ranging from 3.8x to 7.9x, while on the contrary after the optimization process, the Inference speed ratio in static sign language to dynamic sign language, is ranging from 0.8x to 1.5x. The figures also confirm that the single model form is faster than the multi-model. From comparing the increase in speed as a result of optimization according to the models, the single model DenseNet121 (DEPTH) changes from 24.2x to 110.2x and the single model VGG16 (RGB) from 18x to 64x, then the multi- model DenseNet121 (DEPTH) & VGG16 (RGB) from 13.6x to 36.4x this in the static sign recognition models. While the increase in speed was the result of optimization in the dynamic sign recognition models, as it was from 2.2x to 9x for the single model ResNet50-GRU (Depth) and the single model ResNet50-LSTM (RGB) 2.5x to 4.9x, and then from 2.7xto 10.4x by the multi- model ResNet50-BiLSTM-Normalization. From the comparison among Figures (5 &6) and Figures (9 & 10), it is clear that there is a high acceleration in the TensorRT C++ API method compared to the acceleration in the TF-TRT method for static and dynamic sign language recognition models using the Jetson device.

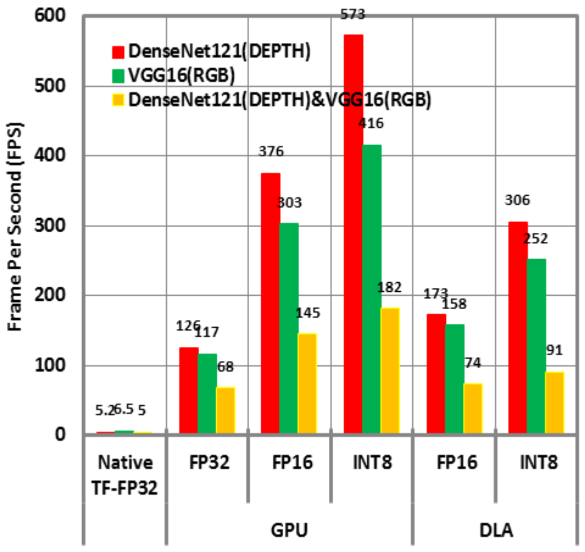

Figure (7): Inference FPS on GPU measure for static sign language recognition using Jetson device

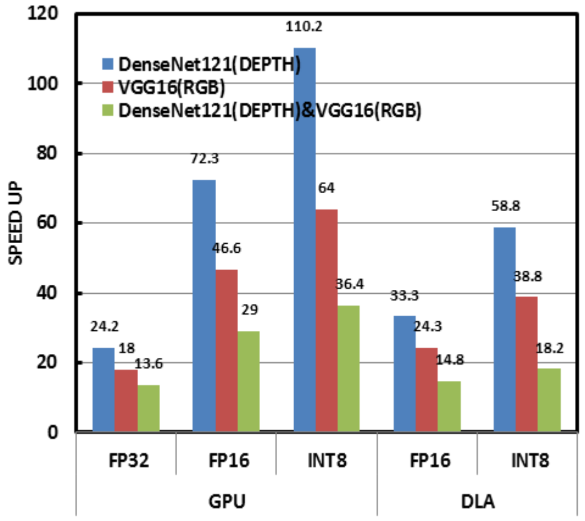

Figure (9): Speed up inference on GPU measure for static sign language recognition using Jetson device

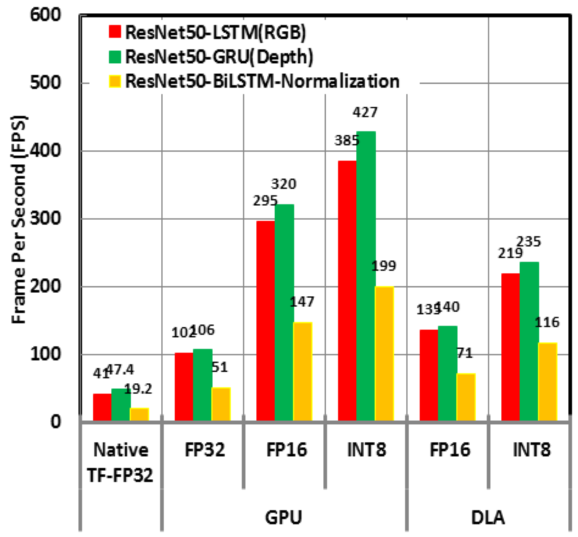

Figure (8): Inference FPS on GPU measure for dynamic sign language recognition using Jetson device

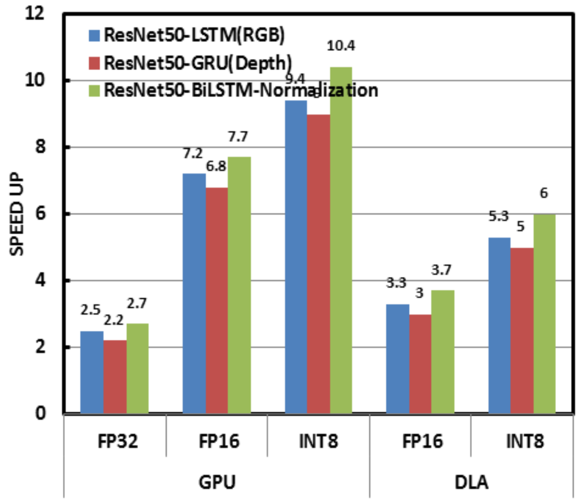

Figure (10): Speed up inference on GPU measure for dynamic sign language recognition using Jetson device

2. CONCLUSION

Through a review, and analysis of presented previous researches with regard to detecting and recognizing of sign language and by discussing and analyzing Tables I and II, which include a survey and comparison of the presented researches in this paper, the following is clarified: The deep learning-based classifier performed better than all the various classifiers in terms of recognition accuracy of sign language. There is no use of the multi-model fusion to recognize Arabic sign language. There is no evaluation of the different methods of fusion models to recognize Arabic sign language. In general, the average accuracy rate of 23 searches to recognize sign language by static hand gesture is 94%, and the average accuracy rate for 25 searches to recognize sign language by dynamic hand gesture is 86%, therefore it is necessary to develop a single model or multiple models to increase the performance and accuracy of the static and dynamic Arabic Sign Language recognition.

REFERENCES

[1] Li,Y., and P. Zhang. "Static hand gesture recognition based on hierarchical decision and classification of finger features." *Science Progress* 105, no. 1 (2022): 00368504221086362.

[2] Oudah, M., A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: a review of techniques." journal of Imaging 6.8, 2020.

[3] Lind, E., and Ä. P. Velasquez, A Performance Comparison between CPU and GPU in TensorFlow, 2019.

[4] Bishwajit, D., "Power Analysis and Prediction for Heterogeneous Computation", M. Sc. Thesis, Faculty of the Virginia Polytechnic Institute and State University, 2018.

[5] Ismail, M. H., S. A. Dawwd, and F. H. Ali, *"Static hand gesture recognition of Arabic sign language by using deep CNNs"*, Indonesian Journal of Electrical Engineering and Computer Science 24, no. 1, pp: 178-188. 2021.

[6] Ismail, M. H., S. A. Dawwd, and F. H. Ali, "*Dynamic hand gesture recognition of Arabic sign language by using deep CNNs*", Indonesian Journal of Electrical Engineering and Computer Science 25, no. 2, pp: 952-962. 2022.

[7] Ismail, M. H., S. A. Dawwd, and F. H. Ali, "*Arabic Sign Language Detection Using Deep Learning Based Pose Estimation",* 2nd International Conference on Engineering Technology and its Applications 2021, IEEE.

[8] Gondi, S., and V. Pratap, Performance and Efficiency Evaluation of ASR Inference on the Edge, *Sustainability* 13.22 , 2021.

[9] Ghadani,A., A. K. Abdullah, W. Mateen, and R. G. Ramaswamy. "*Tensor-based cuda optimization for ann inferencing using parallel acceleration on embedded gpu.*" In IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 291-302. Springer, Cham, 2020.

[10] Haas, B., "*Compressing MobileNet With Shunt Connections for NVIDIA Hardware*", M.Sc. Thesis, Computational Science and Engineering, Vienna University of Technology (TU Wien). 2021.

[11] Patel, B., and V. Sanchez, Inference with Nvidia T4 on Dell EMC PowerEdge R7425, Dell EMC Technical White Paper. 2019.

[12] hin, D. J., and J.J. Kim. "A Deep Learning Framework Performance Evaluation to Use YOLO in Nvidia Jetson Platform." Applied Sciences 12, no. 8, 2022.

[13] Ozgon, D., 3 Ways To Get Started With TensorRT 8 Using TensorFlow, https://becominghuman.ai/3-ways-to-get-started-with-tensorrt-8-using-tensorflow-8e419132ee85, 2021.

[14] Baller,S. P., A.l Jindal, M. Chadha, and M. Gerndt, "*Deep Edge Bench: Benchmarking Deep Neural Networks on Edge Devices.*" In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 20-30, 2021.

[15] Srivallapanondh, S. B., "*Dissecting the Performance of AI Applications Using NVIDIA GPUs at the Edge*", M.Sc. Thesis, School Of Science, Department Of Informatics And Telecommunication, National And Kapodistrian University Of Athens, 2021.

[16] Lukac M., How to deploy object detection on NVIDIA Jetson Nano, available at: https://www.ximilar.com/how-todeploy-object-detection-on-nvidia-jetson-nano/, 2021.

[17] Pettersson, L., "*Convolutional Neural Networks on FPGA and GPU on the Edge: A Comparison"*, UPTEC F 20028 Examensarbete 30 hp, Uppsala University, 2020.

Authors Biography

Mohammad Haqqi Ismail received the BSc and MSc degree in Computer Engineering in 2009 and 2017 from University of Mosul, IRAQ. He is work as assistant lecturer in Technical Computer Engineering, Al-Hadba University College, Mosul, IRAQ. Currently, He is PhD student at research stage in Computer Engineering Department University of Mosul, IRAQ. He researches interests include image processing, deep learning and parallel processing. He can be contacted at email: mohammadhaqqi@gmail.com.

Prof. Dr. Shefa A. Dawwd is a professor of computer engineering at the Computer Engineering Department-University of Mosul. He received the B.Sc. in Communication Engineering, the M.Sc. and the Ph.D. in Computer Engineering. He has authored about 40 international journal, conference papers and one-chapter book. His research focus is on the processing acceleration of 1D, 2D and 3D signals, real time applications, deep learning, Convolutional Neural Networks, and heterogeneous computing. He is a regular reviewer of IEE, Elsevier and other Scopus journals. He can be contacted at email: sheaf.dawwd@umosul.edu.iq.

Dr. Fakhrulddin H. Ali is assistant professor at the computer engineering Department-University of Mosul. He received B.Sc. in Electronic and Communication Engineering-Department of Electrical Engineering-University of Mosul. He received P.G. Diploma and M.Sc. from the same Department at 1977-1979. He graduated from university of Bradford-U.K. with a PhD degree at 1989. He has more than 30 scientific papers in journals and conferences. He supervised more than 25 postgraduate M.Sc. and PhD. Theses and dissertations. His field of interest is 3D computer graphics and real time systems. He can be contacted at email: fhazaa@uomosul.edu.iq..

Cite this paper:

Mohammad H. Ismail, Shefa A. Dawwd, Fakhrulddin H. Ali, "Inference Speed Optimization of Deep Neural Network on Heterogeneous Devices", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 9, pp. 1-8, September 2022.