BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 3
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0028

Hardware Response and Performance Analysis of Multicore Computing Systems for Deep Learning Algorithms

Lalit Kumar, Dushyant Kumar Singh

CSED, MNNIT Allahabad, Prayagraj, India
E-mails: lalitkmr170@gmail.com dushyant@mnnit.ac.in

Abstract: *With the advancement in technological world, the technologies like Artificial Intelligence* (*AI*)*, Machine Learning* (*ML*)*, and Deep Learning* (*DL*) *are gaining more popularity in many applications of computer vision like object classification, object detection, Human detection, etc., ML and DL approaches are highly compute-intensive and require advanced computational resources for implementation. Multicore CPUs and GPUs with a large number of dedicated processor cores are typically the more prevailing and effective solutions for the high computational need. In this manuscript, we have come up with an analysis of how these multicore hardware technologies respond to DL algorithms. A Convolutional Neural Network* (*CNN*) *model have been trained for three different classification problems using three different datasets. All these experimentations have been performed on three different computational resources, i.e., Raspberry Pi, Nvidia Jetson Nano Board, & desktop computer. Results are derived for performance analysis in terms of classification accuracy and hardware response for each hardware configuration.*

Keywords*: Embedded devices, Nvidia Jetson nano board, deep learning, image classification, hardware performance, Cuda core.*

1. Introduction

In the modern age, Machine Learning (ML), and Deep Learning (DL) algorithms have attracted significant attention of researchers. One of the reasons is the incremental improvements in DL algorithms where higher accuracy outcomes can purposely be saught for many problem areas like Data Analytics, Natural Language Processing, Image Processing, etc. These achievements have encouraged researchers to apply deep learning methods for various complex tasks like object detection, object classification, human detection, etc. [16, 20, 22].

Deep learning architectures like Convolutional Neural Network (CNN) perform well but have intense requirements for high-end computation hardware. The huge amount of data getting processed brings a high condition of computation need. The advanced computing system equipped with highly evolved multicore CPUs is

actually cooping this demand. On the other hand, GPUs are there which could help in accelerating the entire process of deep learning training and validation. It has often been demonstrated in the literature that the need for CPU and GPU cores depends on the layered architecture of the CNN. As the number of layers increases in CNN architecture, more number of hardware cores are then required, for faster parallel processing. A single-core CPU has restricted computational limits (e.g., computation units and memory) to handle large-scale neural networks. To overcome this issue, multicore architectures are now getting utilized. Modern parallelization of CPU and GPU in multicore systems has shown remarkable results in deep learning kind of applications [25-27].

Not only current day's desktop PCs or Laptops facilitate multicore CPU configuration, rather small embedded devices are now designed to contain such CPUs and GPU [10, 14]. The embedded devices have gained the tremendous capability to deal with AI algorithms using multicore architecture. The evolution in nanometer-scale fabrication has made it possible to encapsulate miniature size CPU & GPU on a very small size board. Raspberry Pi & Nvidia Jetson Nano are few to name [11]. They are made capable of even running complex training tasks of deep learning models. Intel Neural Stick is yet another example that embeds the dedicated processor cores for processing of neural network operations using TensorFlow and/or Caffe libraries. To analyze the hardware response of these devices on deep learning algorithms, the two of the above-mentioned embedded hardware are taken, and experiments have been carried out for response time-related performance [15]. The analysis is also done for desktop PC containing GPU, and relatively assessment is done by comparing the three cases [24].

1.1. Raspberry Pi 3 B+

Raspberry Pi is a lightweight, compact embedded device capable of working like a full-fledged computer system. Its embedded nature makes it suitable for real-time applications. Raspberry Pi is with their sufficient computer power can efficiently be used for various computer vision applications. Due to its low cost and small size, this is a good option to use it inplace of a Desktop PC for moderate tasks [9].

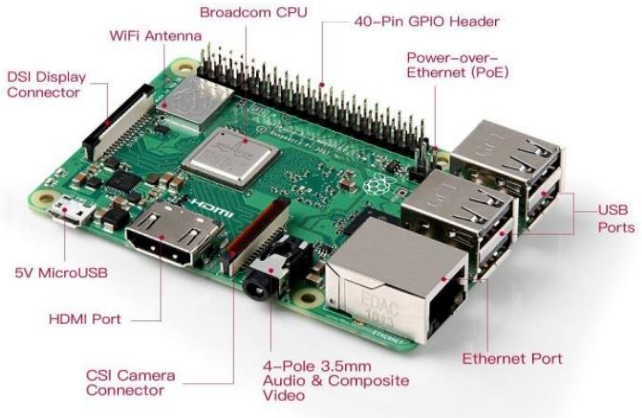

Fig. 1. Structure of Raspberry Pi 3 B+ [32]

Raspberry Pi, in its beginning was designed with a 700 MHz CPU in the year 2009. Later, it evolves as Pi 2, Pi 3, Pi 3 B & Pi 3 B+, where 3B+ is running at 1.2 GHz frequency with a 64-bit Broadcom CPU. Pi 3B+ possesses a quad core CPU which can run complex algorithms at a faster speed while exploiting the parallelizability in multiple cores. This support in hardware has motivated us to employ deep learning-based object classification algorithms on Raspberry Pi to analyze the hardware response [19].

1.2. Nvidia Jetson Nano Board

Nvidia in 2014 first developed an embedded board named Jetson TK1, which contains the quad core ARM Cortex A15 CPU. Later, it advances as TK2, Xavier, and Nano, where Nvidia Jetson Nano board has Quad core A57 CPU @1.53GHz and 921 MHz Maxwell generation GPU. The Quad core structure of the Nvidia Jetson Nano board helps to execute complex AI errands with great efficiency. It also has a quality of CPU & GPU based heterogeneous architecture, where CPU is used to speed up the OS, and CUDA-capable GPU can be rapidly planned to accelerate the tasks of ML. The overall hardware specification of Nvidia Jetson Nano motivated us to use it in various computer vision problems like object detection, image classification, and video surveillance [17, 18].

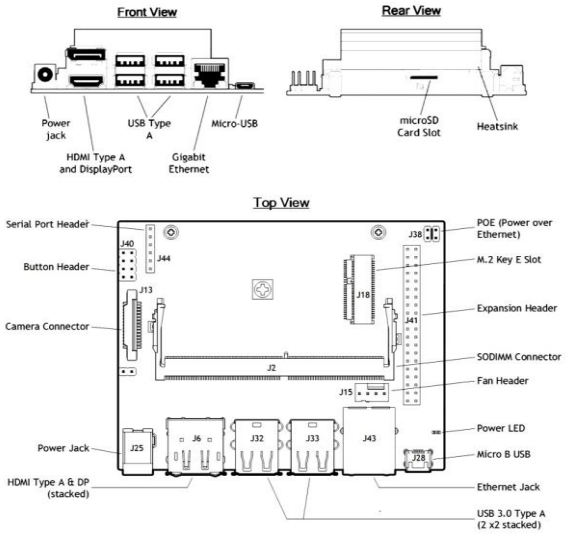

Fig. 2. View of Nvidia Jetson Nano Board [11]

1.3. Desktop PC

The PC has been taken as one competitive hardware for analyzing response as PC's has seen tremendous evolution in last few years. The advancement in desktop PC is done by increasing the clock speed, increasing CPU cores and/or adding up the GPU's. The computational power of desktop PC is effectively increasing with continuous evolution in CPU and GPU cores. The desktop PC used here is based on

a 64-Bit octa-core Processor @ 4.2 GHz CPU and 4 GB Nvidia GTX 1650 Ti running at a boost frequency of 1.3-1.5 GHz GPU. Here, the Nvidia GTX 1650 TI has 1,536 number of Cuda cores. This Quad core structure of CPU and multicore Cuda architecture of GPU allows execution of the complex deep learning model with faster speed.

This paper explores the performance of above mentioned embedded devices (like Raspberry Pi 3 B+, Nvidia Jetson Nano Board) and desktop PC over image classification tasks. A CNN model is used here to perform image classification. The response time of each device is evaluated by running CNN model on different training inputs. Due to a large number of CPU and GPU cores, the hardware response of a desktop PC seems much better than both embedded devices. Another side, the Nvidia Jetson's hardware response is quite good compared to the Raspberry Pi 3B+ because it has a large number of CUDA cores of Maxwell generation GPU, that may allow it to be used in place of a desktop PC where space and cost are constraints. In contrast, the Raspberry Pi 3 B+ can also be used in place of a desktop PC, only for small and medium level AI tasks [12, 13].

The remaining paper is organized as follows. Section 2 discusses related works. The proposed CNN architecture to perform image classification tasks is discussed in Section 3. Section 4 discusses the experimental results obtained using CNNs on different hardware. Finally, the last section concludes the manuscript work presented in this.

2. Related works

This section discusses the related work done on different computational hardware for numerous applications, which is as follows:

Zu [1] have developed a heterogenous multicore architecture model based on CPU and Multicore accelerator. This work proposes a parallel algorithm for Machine learning and execution model related to heterogeneous multicore architecture systems. It is mainly focused on performing testing and evaluation on S698P's emulator GRSIM. After testing S698P's emulator, it is found that the effectiveness of deep learning tasks is improved compared to traditional parallel processing. Gomatheeshwari and Selvakumar [2] propose a framework for optimizing the performance and energy of multicore architecture. A deep learning model for multicore architecture is designed for this system being proposed based on the asymmetric multicore processor. This framework is based on the workload characterization of CPU cores, followed by the core prediction module. The scheme being proposed is implemented in basically three stages: feature Extraction, optimization, and prediction of the core on which workload is distributed. This work reduces energy consumption up to 35% and achieves 97% accuracy in core prediction for the workload.

Mittal [3] has reviewed the performance of various NVIDIA Jetson platforms strategies for speeding up Neural networks on this platform. This article talks about the hardware and CNN level enhancement. It shows that Nvidia Jetson is speeding up analytical tasks for a large area of applications. Demir and Ertürk [4] have

proposed a combination action model consisting of SVM classification and a hierarchical approach to improve the SVM classification and reduce the load on SVM testing. For the image classification task, 2D wavelet decay is applied to each hyperspectral picture band, and low spatial recurrence parts of each level are utilized for hierarchical classification [7].

Sultana, Sufian and Dutta [5] have developed an advanced CNN framework for image recognition tasks. They reveal that the network being proposed is more extensive than the AlexNet, ZFNet, and VGGNet, which follow the architecture of the traditional CNN model like LeNet-5. It has been experienced that an efficient conventional CNN model is created after combining the initiation module and residual blocks with the GoogLeNet and ResNet for better precision. The model being proposed gets an efficient accuracy for various datasets. Salimi, Dewantara and Wibowo [6] presents a garbage identification and characterization framework to deal with a real-world scenario. To initially classify any objects on the floor, the Haar-Cascade technique is used here. At that point, the GrayLevel Co-occurrence Matrix (GLCM) and HistOgram Gradient (HOG) are combined to get efficient results. The pre-trained model of SVM is used here to categorize natural waste, non-natural waste, and non-squander highlights using image classification tasks and achieve up to 82.70% accuracy.

Ramcharan et al. [7] have presented a method to identify Soybean diseases. The proposed deep neural networks provide an avenue for the fast deployment of this technology on mobile devices. After performing a deep learning approach, they get 98% accuracy for brown leaf spot, 96% accuracy for red mite damage, 95% accuracy for green mite damage, 98% for brown cassava streak, and 96% accuracy for cassava mosaic disease. The best-qualified model accuracy has been 98% [29, 30].

The traditional approaches require a large computational setup for performing a real-time image classification task. As a result, they require a lot of time in computational training and testing. Recently, the researchers have followed a simplistic approach to the utilization of current design computational hardware. Researches going all around are following the traditional hardware and therefore limited in dealing with real-time and practical design & usage. Therefore, we have tried analyzing multiple hardware for such deep learning-based complex tasks and so used NVIDIA JETSON NANO, Raspberry Pi 3B+, and desktop PC for image classification tasks using the CNN model. The description of the proposed methodology is covered in the next section with the training results and performance comparison between embedded devices and Desktop PC.

3. Proposed methodology

As mentioned earlier, the methodology being proposed aims at developing CNN-based solutions for benchmark classification problems with respective datasets and then executing them on different computational hardware. First, we have discussed about CNN modeling for our set of classification problems.

3.1. CNN modeling

CNN architecture incorporates four essential layers, namely Convolution Layer, Pooling Layer, Fully Connected Layer, and Output Layer. The convolution layer has several filters that perform the convolution operation on the image dataset. Here 3×3 filters are used to extract the features, such as edges from the input images. The convolution layer captures low-level features like color, borders, gradient orientation, etc. The pooling layer converts the high-dimensional data space into low-dimensional data space. This conversion helps to reduce the computational power required for the processing. The fully connected layer consists of neurons that facilitate training the CNN model, and the output layer is responsible for generating the final result using the activation function. The layers of CNN provide a scalable approach for image classification and object recognition tasks in the various application domains. The overall architecture of the proposed model is shown in Fig. 3. The parameter level details of CNN layers are provided.

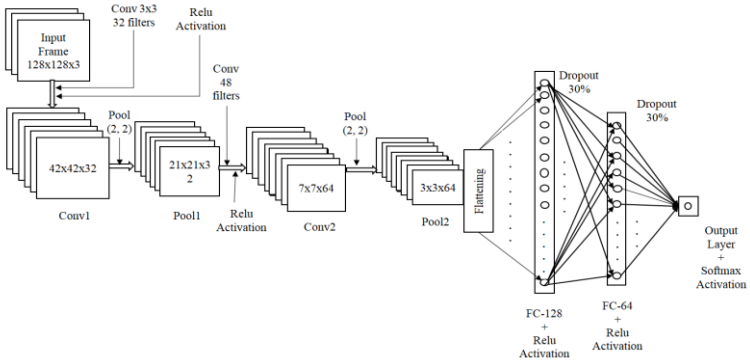

Fig. 3. Convolution Neural Network Architecture

Table 1. Parameter level details of the proposed CNN

Layer	Output shape	Param #
Convolution 2D_1	(None, 42, 42, 32)	896
Max Pooling_1	(None, 21, 21, 32)	0
Convolution 2D_2	(None, 7, 7, 64)	18496
Max Pooling_2	(None, 3, 3, 64)	0
Flatten	(None, 576)	0
Dense	(None, 128)	73856
Dense	(None, 64)	8256
Dense	(None, 5)	325
Total parameter		101,829

This model uses one input, two Convolutional, two pooling, and one output layer. An image of size 128×128×3 is input to the first convolutional layer containing 32 filters of size 3×3 with pooling operation of pool size 2×2. The evaluated consequences are again passed to the convolutional layer containing 64 filters of each size of 3×3, followed by a pooling layer of pool size 2×2. The flattening layer is then used to convert evaluated multidimensional vector to vector size spaces, which are further utilized by FC1 and FC2 layers. Here, 30% of dropout is used in both FC

layers to protect our training data from overfitting and underfitting. FC layers are solely responsible for training. Finally, output yields softmax activation and produces an outcome. Here, reLU activation is used in the convolution layer & tanh activation is used for FC layers. The total trainable parameters in the proposed network come to be 101,829.

The CNN architecture as discussed earlier is trained for three different classification problems. First of these is plant leaf classifications. It could be a reliable tool to understand plants for those who remain away from agriculture. It is hard to remember plants by viewing a large variety of flora. This is one popular problem found in the research literature, therefore we had considered this as our benchmark for the hardware response analysis task. The other classification problem covered here is the garbage classification, one of the hitting issues related to "Swach Bharat Abhiyan." Same way, trash classification is another benchmark classification problem considered here. To get to this objective, the CNN model is trained to classify the trash in real-time, which could help the municipalities to manage the household trash.

For the garbage management system, image classification helps determine the type of garbage. There are some basic classes like household waste, kitchen waste, city waste, etc., on which the CNN model is trained.

3.2. Hardware setup

The model being proposed is trained on a machine configured with a CPU Ryzen 7-4800H, 8 Gb RAM, and 4 GB NVIDIA GTX 1650Ti GPU (1024 CUDA cores). Other side, the trained model is tested on two embedded devices, i.e., Raspberry Pi 3 B+ and Nvidia Jetson Nano Board.

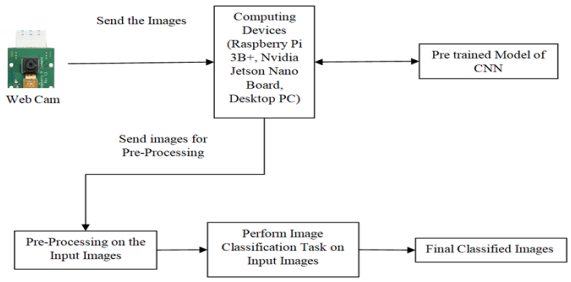

Fig. 4. Image classification over computing devices

The Raspberry Pi 3 B+ consists of a 64-bit Broadcom quad core Cortex-A53 CPU with a clock speed of 1.4 GHz and 2 GB SDRAM. The Raspberry 3B+ operating system runs on the Raspbian 10 operating environment, stored on the SD card. The pre-trained model is tested on a Raspberry Pi 3 B+ board to classify images for different datasets. The implementation procedure in the Nvidia Jetson Nano board is similar to Raspberry Pi 3 B+ for image classification tasks. The Nvidia Jetson contains greater number of computational units compared to the Raspberry Pi 3 B+. This makes Nvidia Jetson more reliable for handling complex computer vision tasks [12, 13]. The Nvidia Jetson Nano board runs on a quad core ARM Cortex-A57 with

4 GB of SDRAM and consumes a maximum power of 10 W. The Nvidia Jetson Nano board runs on the Jetpack SDK (Ubuntu 18.4), which is stored on the SD card. Here, the same pre-trained CNN models are used to classify plant leaves, garbage, and garbage images. The CNN model shows effective results with the Jetson board [10, 15].

Fig. 4 shows the workflow of the image classification task computed over different hardware devices. These devices are connected to the camera module from where they receive the frames. The captured frames are then passed to the CNN model, classifying the images into their respective classes. Finally, the device returns the classification result based on the evaluated CNN prediction.

3.3. Datasets

The existing trash, garbage, and PlantVillage datasets are taken from an openly accessible data repository, namely Kaggle and Gale. PlantVillage dataset has images of different fruits and vegetables, whereas the trash dataset has various images of paper, cardboard, plastic, grass, metal, etc. Otherside, the garbage dataset contains images like kitchen_Garbage, City_Garbage, Village_Garbage, Forest_Garbage, Hazardous_Garbage, and all. The descriptions related to datasets are presented in Table 2.

Table 2. Description of Datasets

Description of datasets			
Features	Plant village	Trash dataset	Garbage dataset
Dataset repository link	**https://www.kaggle.com/emmarex/plantdisease**	**https://gale31.github.io/TIDY/**	**https://www.kaggle.com/asdasdasasdas/garbage-classification**
Classes	Apple_Healthy, Blueberry_Healthy, Grape_Healthy, Peach_Healthy, Potato_Healthy, Soyabean_Healthy, Strawberry_Healthy, Tomato_Healthy	cardboard, metal, paper, plastic, and grass	kitchen_Garbage, City_Garbage, Village_Garbage, Forest_Garbage, Hazardous_Garbage
No of training images	6000	4000	3000
No of testing images	1219	976	786
Total images	7219	4976	3786

4. Experimental results

In this section, the hardware performance of desktop PC and two embedded devices is analyzed on CNN architecture for multithreading workloads. This also provides the training results of classification when executed on the proposed model with three

datasets, and the prediction time for test images on both embedded devices and a desktop PC. A detailed description of all experimental setups is mentioned in Table 3.

Table 3. Experimental setup details

Feature	Desktop PC	Nvidia Jetson Nano Board	Raspberry Pi 3 B+
CPU	Ryzen 7	Quad-core ARM Cortex-A57 (4 core),	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC@ 1.4GHz
CPU core	Octa-core	Quad-core	Single-core
GPU	4 GB Nvidia 1650Ti	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores	Broadcom video core-IV
RAM	8 GB	4 GB	4 GB
Operating System & its version	Window 10 Home	Ubuntu 18.4	Raspbian 5.4.51
Peak performance	1792 Gflops	500 Gflops	200 Gflops
Power required	120 W	10 W	1.5 W to 6.7 W
Weight	250 gm	85 gm	50 gm
Price	$250	$89	$35

4.1. Results

This section presents the experimental results of the proposed CNN architecture evaluated over a desktop PC and two embedded devices. Table 4 shows the experimental results of the proposed model performed on the PlanVillage datasets. This model takes 2412 s (40.2 min) in the entire training process for batch size 4, which is the least time for training the deep learning model compared to other batch sizes. The proposed model has achieved the highest accuracy at batch size 4, which is up to 98.57%.

Table 4. Experimental Results of Plant Village Dataset

Iteration	Batch size (4)				Batch size (8)				Batch size (16)			
	Training		Validation		Training		Validation		Training		Validation	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
20	97.16	0.360	97.14	0.130	95.73	0.480	95.71	0.350	96.04	0.450	90.00	0.800
40	98.22	0.220	97.14	0.310	97.39	0.310	97.14	0.230	97.90	0.280	97.14	0.320
60	98.87	0.150	98.57	0.130	97.76	0.280	98.57	0.230	97.89	0.250	98.57	0.110
80	99.01	0.150	95.71	0.250	98.88	0.140	95.71	0.300	98.81	0.150	97.14	0.250
100	98.98	0.150	97.14	0.180	98.44	0.210	98.57	0.120	98.90	0.140	97.57	0.130
120	98.80	0.130	**98.57**	0.090	98.72	0.180	**95.71**	0.380	99.22	0.100	**97.63**	0.200
Execution time	Time per Epoch= 20 s 10 ms Total execution time= 120×20.10=2412 s				Time per Epoch= 20 s 30 ms Total execution time= 120×20.30=2436 s				Time per Epoch= 20 s 58 ms Total execution time= 120×20.58=2469.6 s			

Table 5 contains the training and validation results for the Trash dataset at various epoch levels. Here, the efficient training time taken by batch size 4 is 1104 s (18.4 min), which is the least amount of time taken for training compared in case of

other batch size. The highest accuracy is achieved at batch size 16, which is up to 71%.

Table 5. Training results of Trash Dataset

Iteration	Batch size (4)				Batch size (8)				Batch size (16)			
	Training		Validation		Training		Validation		Training		Validation	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
20	66.84	0.293	58.00	0.344	70.49	0.273	56.00	0.350	66.78	0.296	50.00	0.349
40	74.99	0.226	64.00	0.329	74.06	0.237	54.00	0.361	71.25	0.264	54.00	0.342
60	79.60	0.197	65.00	0.361	81.02	0.191	62.00	0.367	77.03	0.219	57.00	0.353
80	84.06	0.162	65.00	0.372	83.60	0.161	66.00	0.327	82.75	0.176	67.00	0.311
100	87.89	0.135	67.00	0.373	87.38	0.134	66.00	0.343	81.86	0.179	61.00	0.337
120	87.39	0.129	**67.00**	0.393	90.01	0.116	**67.00**	0.329	87.35	0.143	**71.00**	0.314
Execution time	Time per Epoch= 9 s 20 ms Total execution time = 120×9.20=1104 s				Time per Epoch= 9 s 37 ms Total execution time = 120×9.37=1124.4 s				Time per Epoch= 9 s 74 ms Total execution time = 120×9.74 =1168.8 s			

Table 6 presents the experimental results for the proposed model evaluated over the garbage dataset. The experimental outcomes show an interesting thing because the training accuracy of each batch size is different from each other. The reason behind this is the variation in a class image of the garbage dataset. The highest validation accuracy is achieved as 90 % with batch size 4. The efficient execution time is found using batch size 8, i.e., 1458.6 s (24.30 min).

Table 6. Training results of Garbage Dataset

Iteration	Batch size (4)				Batch size (8)				Batch size (16)			
	Training		Validation		Training		Validation		Training		Validation	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
20	67.09	0.282	71.11	0.240	36.88	0.454	27.78	0.478	33.73	0.467	27.78	0.504
40	72.93	0.262	76.67	0.206	42.65	0.434	34.44	0.448	42.87	0.432	40.00	0.471
60	77.84	0.234	83.33	0.191	47.42	0.389	48.89	0.416	47.39	0.406	44.44	0.402
80	80.70	0.199	83.33	0.168	59.49	0.348	61.11	0.355	52.72	0.369	55.56	0.356
100	80.91	0.190	84.44	0.161	64.28	0.323	70.00	0.289	57.77	0.347	58.89	0.330
120	83.23	0.176	**90.00**	0.126	67.50	0.289	**70.00**	0.300	61.05	0.314	**66.67**	0.309
Execution time	Time per Epoch=12 s 73 ms Total execution time =120×12.73=1527.6 s				Time per Epoch = 12 s 15 ms Total execution time = 120×12.15=1458.6 s				Time per Epoch = 12 s 29 ms Total execution time = 120×12.29 =1474.8 s			

To better understand the effect of hyper-parameter on the CNN model, we performed hyperparameter tuning on all datasets with various activation functions shown in Table 7. We used ReLU, Tanh, and LeakyReLU on Convolutional Layer and ReLU, Tanh on fully connected layer. At last, we used the sigmoid function at the output layer of the model.

Table 7. Hyper-parameter tuning for CNN model

Dataset	Model	Batch size	Dropout	Activation function			Optimizer	Epochs	Training		Validation	
				Convolutional layer	FC layer	O/P layer			Accuracy	Loss	Accuracy	Loss
Plant Village	A1	4	0.30	Relu	Relu	Softmax	Adam	100	**98.32**	0.312	**98.78**	0.208
	A2	8	0.30	Tanh	Tanh	Softmax	Adam	100	97.19	0.389	97.34	0.247
	A3	16	0.30	LeakyRelu	Relu	Softmax	Adam	100	97.32	0.312	97.48	0.224
	A4	4	0.30	Relu	Relu	Softmax	Ada Delta	100	98.17	0.301	97.54	0.212
	A5	8	0.30	Tanh	Tanh	Softmax	Ada Delta	100	98.68	0.325	98.18	0.230
	A6	16	0.30	LeakyRelu	Relu	Softmax	Ada Delta	100	98.08	0.112	98.22	0.210
Trash	B1	4	0.30	Relu	Relu	Softmax	Adam	100	**86.23**	0.345	**78.00**	0.497
	B2	8	0.30	Tanh	Tanh	Softmax	Adam	100	84.01	0.215	66.00	0.413
	B3	16	0.30	LeakyRelu	Relu	Softmax	Adam	100	87.35	0.215	72.00	0.314
	B4	4	0.30	Relu	Relu	Softmax	Ada Delta	100	87.49	0.310	67.00	0.369
	B5	8	0.30	Tanh	Tanh	Softmax	Ada Delta	100	85.01	0.747	66.00	0.258
	B6	16	0.30	LeakyRelu	Relu	Softmax	Ada Delta	100	87.35	0.852	72.00	0.247
Garbage	C1	4	0.30	Relu	Relu	Softmax	Adam	100	**68.91**	0.287	**78.00**	0.156
	C2	8	0.30	Tanh	Tanh	Softmax	Adam	100	67.50	0.369	72.00	0.360
	C3	16	0.30	LeakyRelu	Relu	Softmax	Adam	100	62.37	0.425	67.67	0.358
	C4	4	0.30	Relu	Relu	Softmax	Ada Delta	100	72.91	0.217	62.00	0.874
	C5	8	0.30	Tanh	Tanh	Softmax	Ada Delta	100	67.50	0.369	72.00	0.488
	C6	16	0.30	LeakyRelu	Relu	Softmax	Ada Delta	100	62.37	0.425	67.67	0.458

In Table 7, we see that the performance of the proposed model for various datasets in the presence of different modeling parameters varies. Model A1 for the plantvillage dataset has achieved the highest validation accuracy (98.78%) and training accuracy (98.32%) compared to other variants. Model B1 for the trash dataset has achieved the highest validation and training accuracy, up to 78.00% and 86.23%, respectively, compared to other variants. In contrast, the C1 variant of the garbage dataset has also acquired 78.00% validation and 68.91% training accuracy, which is most efficient compared to other variants.

4.2. Result analysis

The graph in Fig. 5 presents the timing observed for the training process of the deep learning model, for the PlantVillage, garbage, and trash datasets, with different batch sizes, i.e., 4, 8, and 16. After analyzing the training time for each dataset, it is found that the PlantVillage dataset takes more time in training at batch sizes 4, 8, and 16 than the other datasets because it has a total of 7219 images with eight classes of images. The training is performed on GPU enabled desktop PC, and it took a fair time, averaging 2400 s, i.e., around 40 min. Faster training becomes possible with available parallel Cuda cores in Nvidia GPU.

The same trained model is run on all three hardware to analyze their performance. The graph in Fig. 6 presents the timing analysis of the classification task on these hardwares. On analyzing the graph in Figure 6, we find a significant difference in the response time for all hardware. Here, desktop PC simultaneously takes 2× and 3× less time to classify an image than Nvidia Jetson Nano Board and Raspberry Pi 3 B+. This significant difference in prediction time is seen due to several reasons. The first reason is the specifications of embedded devices and desktop PC shown in Table 2, which means if any device has a parallel and application-specific

processor, it may take less time and effort to complete the task. The second reason is the availability of Cuda cores of GPU in devices. GPUs rend images more quickly than a CPU because of their parallel processing architecture, allowing them to simultaneously perform multiple calculations across data streams. Due to this, Raspberry Pi takes a lot of time compared to Nvidia and desktop PC because it has only 128 Cuda cores.

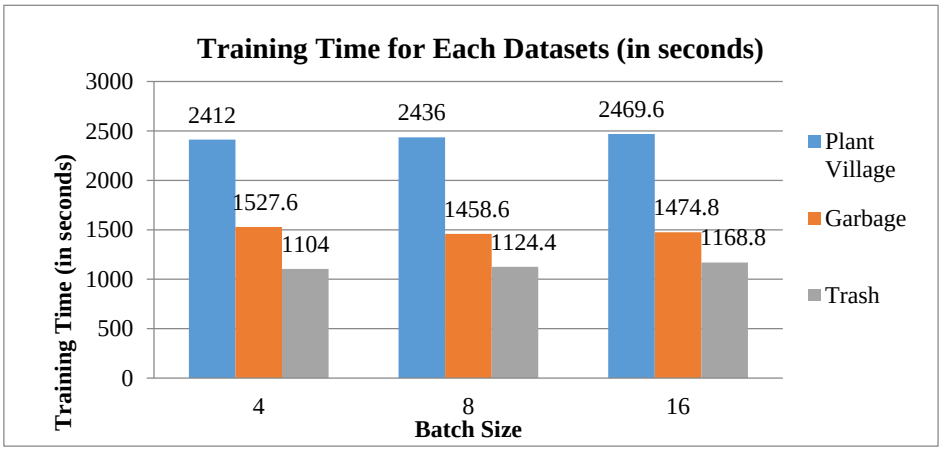

Fig. 5. Training time distribution of the model on different datasets

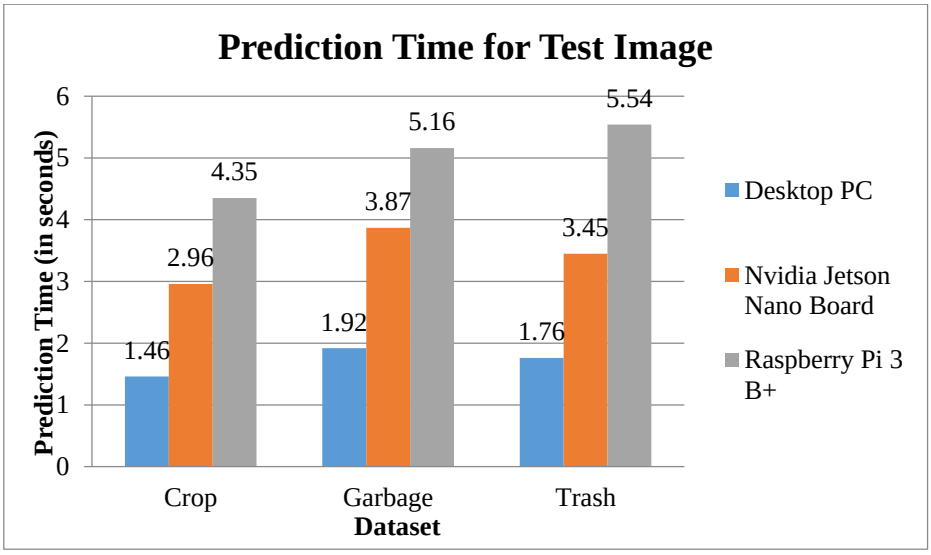

Fig. 6. Comparison of response time for classification

All experiments have been carried out in the Window's environment on a machine configured with 9th generation i5, 8 GB RAM, and 256 GB SSD. CUDA 10.0 with CUDNN 7.5 libraries used to effectively accelerate the GPU-based processing. The dataset used in the experiments is detailed as follows.

5. Conclusion

This paper presents a study of embedded devices and a desktop PC for various neural network applications on multiple datasets and also reviewes the key feature of the Nvidia Jetson Nano Board, Raspberry Pi 3 B+, and a Desktop PC. This paper discusses CNN-level optimization and hardware performance activities under various conditions. The experimental results have shown that the Nvidia Jetson Nano board is 50% more efficient than the Raspberry Pi board due to the availability of a large number of Cuda cores (i.e., 128 Cuda cores) with Cortex-A7 CPU and Maxwell generation GPU. Hence, the Nvidia Jetson Nano board takes less time in the image classification task than a raspberry Pi device. On the other side, Desktop PC is almost two times (2x) more efficient than the Nvidia Jetson Nano because it has a 1536 Cuda core compared to the 128 Cuda core of the Nvidia Jetson Nano Board. As a result, it is found that a Desktop PC with larger processing units are more efficient than the embedded devices (i.e., the Nvidia Jetson Nano board and the Raspberry Pi 3B+ board).

References

1. Zu, Y. Deep Learning Parallel Computing and Evaluation for Embedded System Clustering Architecture Processor. – Design Automation Embedded System, Vol. **24**, 2020, pp. 145-159.
2. Gomatheeshwari, B., J. Selvakumar. Appropriate Allocation of Workloads on Performance Asymmetric Multicore Architectures via Deep Learning Algorithms. – Microprocessors and Microsystems, Vol. **73**, 2020, 102996. ISSN 0141-9331.
3. Mittal, S. A Survey on Optimized Implementation of Deep Learning Models on the NVIDIA Jetson Platform. – Journal of Systems Architecture, Vol. **97**, 2019, pp. 428-442. ISSN 1383-7621.
4. Demir, B., S. Ertürk. Improving SVM Classification Accuracy Using a Hierarchical Approach for Hyperspectral Images. – In: Proc. of 16th IEEE International Conference on Image Processing (ICIP'09). IEEE, 2009.
5. Sultana, F., A. Sufian, P. Dutta. Advancements in Image Classification Using Convolutional Neural Network. – In: Proc. of 4th International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN'18), IEEE, 2018.
6. Salimi, B. S., B. Dewantara, I. K. Wibowo. Visual-Based Trash Detection and Classification System for Smart Trash Bin Robot. – In: Proc. of International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC'18), Bali, Indonesia, 2018, pp. 378-383. DOI: 10.1109/KCIC.2018.8628499.
7. Ramcharan, A., et al. Deep Learning for Image-Based Cassava Disease Detection. – Frontiers in Plant Science, Vol. **8**, 2017, p. 1852.
8. Basulto-Lantsova, J., A. Padilla-Medina, F. J. Perez-Pinal, A. I. Barranco-Gutierrez. Performance Comparative of OpenCV Template Matching Method on Jetson TX2 and Jetson Nano Developer Kits. – In: Proc. of 10th Annual Computing and Communication Workshop and Conference (CCWC'20), 2020, pp. 0812-0816.
9. Kim, S., S. Song, J. Kim, Z. Yuan, J. Cho. Fast Rotation-Invariant Template Matching with Candidate Reduction Using CUDA. – In: Proc. of International Symposium on Consumer Electronics (ISCE'15), 2015, pp. 1-2.
10. Hangün, B., Ö. Eyecioğlu. Performance Comparison between OpenCV Built in CPU and GPU Functions on Image Processing Operations. – International Journal of Engineering Science and Application, Vol. **1**, 2017, pp. 34-41.

11. T a i r A. s a l i h, M o h a m m a d B a s m a n G h. A Novel Face Recognition System Based on Jetson Nano Developer Kit. – IOP Conference Series: Materials Science and Engineering, Vol. **928**, 2020, No 3, IOP Publishing.
12. K u m a r, L., D. K. S i n g h. Analyzing Computational Response and Performance of Deep Convolution Neural Network for Plant Disease Classification Using Plant Leave Dataset. – In: Proc. of 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT'21), 2021, pp. 549-553. DOI: 10.1109/CSNT51715.2021.9509632.
13. S i n g h, D u s h y a n t K u m a r. 3D-CNN Based Dynamic Gesture Recognition for Indian Sign Language Modeling. – Procedia Computer Science, Vol. **189**, 2021, pp. 76-83.
14. S h a h i d, A., M. M u s h t a q. A Survey Comparing Specialized Hardware and Evolution in TPUs For Neural Networks. – In: Proc. of 23rd International Multitopic Conference (INMIC'20). IEEE, 2020.
15. G a o, F., Z. H u a n g, S. W a n g et al. Optimized Parallel Implementation of Face Detection Based on Embedded Heterogeneous Many-Core Architecture. – Int. J. Pattern Recognit. Artif. Intell., Vol. **31**, 2017, No 7, 1756011.
16. Y i n, S., O. P e n g, S. T a n g et al. A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications. – IEEE J. Solid State Circuits, Vol. **53**, 2018, No 4, pp. 968-982.
17. W e n, S., H. W e i, Z. Z e n g et al. Memristive Fully Convolutional Network: An Accurate Hardware Image-Segmentor in Deep Learning. – IEEE Trans. Emerg. Top Comput. Intell., Vol. **2**, 2018, No 5, pp. 324-334.
18. S u g i e, T., T. A k a m a t s u, T. N i s h i t s u j i et al. High-Performance Parallel Computing for Next-Generation Holographic Imaging. – Nat Electron, Vol. **1**, 2018, No 4, pp. 254-259.
19. T h o m a n, P., K. D i c h e v, T. H e l l e r et al. A Taxonomy of Task-Based Parallel Programming Technologies for High-Performance Computing. – J. Supercomput., Vol. **74**, 2018, No 4, pp. 1422-1434.
20. A n s a r i, M. A., D. K. S i n g h. Review of Deep Learning Techniques for Object Detection and Classification. – In: Proc. of International Conference on Communication, Networks and Computing. Springer, Singapore, 2018.
21. H a y a s h i, N., et al. Advanced Embedded Packaging for Power Devices. – In: Proc. of IEEE 67th Electronic Components and Technology Conference (ECTC'17), 2017, pp. 696-703. DOI: 10.1109/ECTC.2017.215.
22. L e c h n e r, M., A. J a n t s c h. Blackthorn: Latency Estimation Framework for CNNs on Embedded Nvidia Platforms. – IEEE Access, Vol. **9**, 2021, pp. 110074-110084. DOI: 10.1109/ACCESS.2021.3101936.
23. V r e č a, J., et al. Accelerating Deep Learning Inference in Constrained Embedded Devices Using Hardware Loops and a Dot Product Unit. – IEEE Access, Vol. **8**, 2020, pp. 165913-165926. DOI: 10.1109/ACCESS.2020.3022824.
24. T y c h a l a s, D., A. K e l i r i s, M. M a n i a t a k o s. Stealthy Information Leakage through Peripheral Exploitation in Modern Embedded Systems. – IEEE Transactions on Device and Materials Reliability, Vol. **2**0, June 2020, No 2, pp. 308-318. DOI: 10.1109/TDMR.2020.2994016.
25. N e e l a m, D., D. K. S i n g h. Review of Deep Learning Techniques for Gender Classification in Images. – In: Proc. of Harmony Search and Nature Inspired Optimization Algorithms. Springer, Singapore, 2019. 1089-1099.
26. L u c k o w, M. C., N. A s h c r a f t, E. W e i l l, E. D j e r e k a r o v, B. V o r s t e r. Deep Learning in the Automotive Industry: Applications and Tools. – In: Proc. of IEEE Int. Conf. Big Data, December 2016, pp. 3759-3768.
27. D e v y a t k i n, V., D. M. F i l a t o v. Neural Network Traffic Signs Detection System Development. – In: Proc. of XXII Int. Conf. Soft Comput. Meas. (SCM'19)), May 2019, pp. 125-128.

Received: 10.01.2022; Second Version: 21.04.2022; Accepted: 08.06.2022