LUPINE PUBLISHERS

ISSN: 2643-6736

Advances in Robotics & Mechanical Engineering

DOI: 10.32474/ARME.2023.04.000182

Research Article

Exploring Automated Deep Learning with Satellite Image Data for Constrained-RAM Microcontroller-based Analysis

Euhid Aman[1]*, Hwang-Cheng Wang[2], Prakhar Mishra[1] and Pallavi R[1]

[1]*Vision and Learning Lab, Presidency University, Bangalore, India*

[2]*National Ilan University, Yilan, Taiwan*

***Corresponding author:** Euhid Aman, Vision and Learning Lab, Presidency University, Bangalore, India

Received: August 21, 2023 **Published:** August 28, 2023

Abstract

Deep Learning has evolved a lot in the past few years. But most neural network applications are for devices with memory capacities exceeding gigabytes. This paper focuses on creating deep learning models for memory-constrained devices. This paper demonstrates the feasibility of creating and deploying highly accurate neural network models on a Raspberry Pi-Pico microcontroller, i.e., the RP2040 chip, possessing just 264KB of memory. The proposed model will be first trained on a large satellite imagery dataset, with the main goal of creating a highly accurate model for automated geospatial image data inference. Broader applications of this paper will lead to the creation of cheaper and cost-effective satellites, which can be used autonomously for land classification, disaster management, and environmental monitoring with the help of the Raspberry Pi Pico microcontroller. This research paper explores the various model architectures, encompassing state-of-the-art designs and transfer learning techniques, all aimed at enhancing the accuracy of the baseline model tailored explicitly for the microcontroller deployment. This study showcases the adaptability of deep learning in resource-constrained environments and, thus, will expand the horizons of automated and intelligent neural network applications.

Impact Statement: Artificial satellites are objects intentionally placed around celestial bodies, such as the Earth and the moon. They generally provide detailed information about those bodies around which they revolve. It takes a lot of money and tremendous technology to create and deploy those artificial objects in space. This research paper introduces the concept of intelligent satellites, which can be created using cheap microcontrollers and can classify geospatial images with an accuracy above 90% due to the deployment of lightweight neural network models on the latest Raspberry Pi Pico microcontroller. This research can therefore offer entirely new insights into creating cost-effective automated space satellites with the help of low-powered microcontrollers.

Keywords: Deep Learning; Image Classification; Microcontrollers; Satellites; Transfer Learning

Introduction

SATELLITE imagery has become a valuable tool in a variety of applications, such as environmental monitoring, disaster response, and law enforcement. The ability to continuously gather from ground-based measurements, making it a crucial technology for a wide range of fields, including urban planning, agriculture, and natural resource management. With the large land area to be covered, automating land area classification from satellite images is essential to save time [1]. The classification process must also require high accuracy, surpassing the performance of existing basic artificial neural network models [2]. This paper aims to create intelligent lightweight deep learning models for deployment on cost-effective microcontrollers, which can be further launched in space for autonomous geospatial image data inference. The concept of transfer learning and model compression is used to achieve better accuracy in satellite image classification and to deploy the neural network models on memory-constrained microcontrollers. Transfer learning [3] is a technique that employs a pre- trained model as a starting point for a new model. Instead of beginning from

scratch, the new model learns from the features extracted from the baseline pre-trained model, reducing the amount of training data needed and thereby improving the new model's accuracy. Model compression in neural networks refers to the different techniques that reduce the size and complexity of neural networks without significant loss in performance. This involves methods like pruning, quantization, and knowledge distillation. The main goal behind this paper is to make accurate and lightweight models more efficient for deployment on resource- constrained devices. This can significantly save costs and resources and benefit the space industry.

Hardware Parameters

Microcontroller Raspberry Pi Pico

The entire automated intelligent satellite system is based on embedded systems. The important component of this project is the Raspberry Pi (Rpi) Pico [4] microcontroller (Figure 1), which has the RP2040 microchip with dual-core Arm Cortex-M0+ processor, 264KB of SRAM, 2MB of on-board QSPI (quad serial peripheral interface) flash storage, 26 GPIO pins, and a variety of interface choices. It is a single-sided PCB with a micro-USB interface and a form factor of only 21mm x 51mm. This microcontroller is responsible for intelligently inferencing and classifying different geospatial image data.

Raspberry Pi 3B+

As shown in (Figure 2), Raspberry Pi 3B+ [5] is a compact single- board computer known for its versatility and affordability. Equipped with a quad-core ARM Cortex-A53 processor and 1GB of RAM, it offers improved performance. It also features built-in Wi-Fi, Bluetooth connectivity, and GPIO pins for various projects. Rpi 3B+ is the main interface, which will process the inference data from the Rpi-Pico microcontroller. This device will receive all the data from the RP2040 chip and display it in a visually pleasing manner.

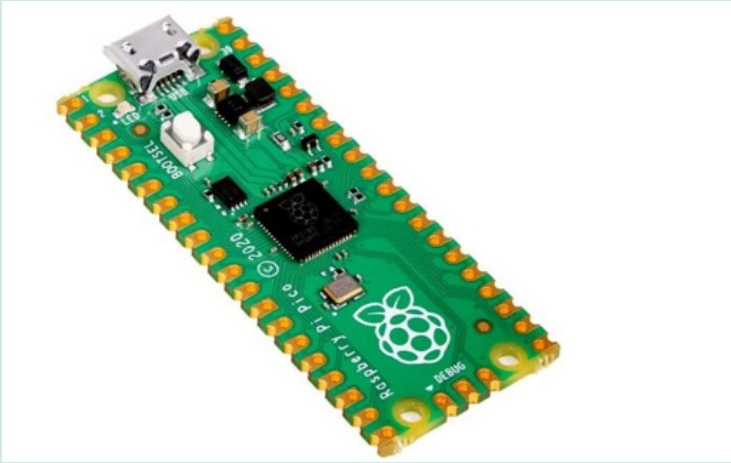

Figure 1: Raspberry Pi - Pico microcontroller.

Figure 2: Raspberry Pi 3B+.

Software Parameters

Raspberry Pi OS – a Linux-Based OS

Raspberry Pi OS [6], a Debian-based Linux distribution, is optimized to leverage the hardware capabilities of Raspberry Pi boards. With its LXDE desktop environment, it provides essential utilities and a repository of software packages. Its utilization of ARM architecture, GPU acceleration, and GPIO access empowers enthusiasts to undertake a broad spectrum of projects. This is the main operating system running on the Raspberry Pi 3B+, which can parse our inference results. This is the platform, which can understand the Rpi-Pico's results.

Tensorflow Lite for Microcontrollers

TensorFlow Lite for Microcontrollers (TFLite Micro) [7] is a small framework designed for microcontrollers and other low- resource devices. It provides a set of tools and frameworks for deploying and optimizing TensorFlow models on small-scale hardware. Kera's, a high-level neural network API built in Python, may be used with TensorFlow as its backend. Kera's provides an easy-to-use interface for deep learning researchers and practitioners to create, train, and deploy neural networks.

Methodology

The intelligent satellite creation workflow, as depicted by the flowchart in (Figure 3), starts with data collection. A suitable and large amount of labeled data is collected. The baseline model is created on that data, which will be further converted into the format to be uploaded on the Rpi-Pico. Otherwise, a different model is created based on transfer learning with the help of pre-trained state-of-the-art models. Once the model is created and stored in the uf2 format, its deployment on the microcontroller allows for immediate initiation of image inference upon receiving external image inputs. The inference outcomes are subsequently recorded within an internal CSV file, facilitating analysis through connection to a Raspberry Pi device acting as an interface. The entire inference happens automatically, and intelligently. No human interaction is needed, which makes the entire system a stand-alone device capable of producing great results without plenty of resources.

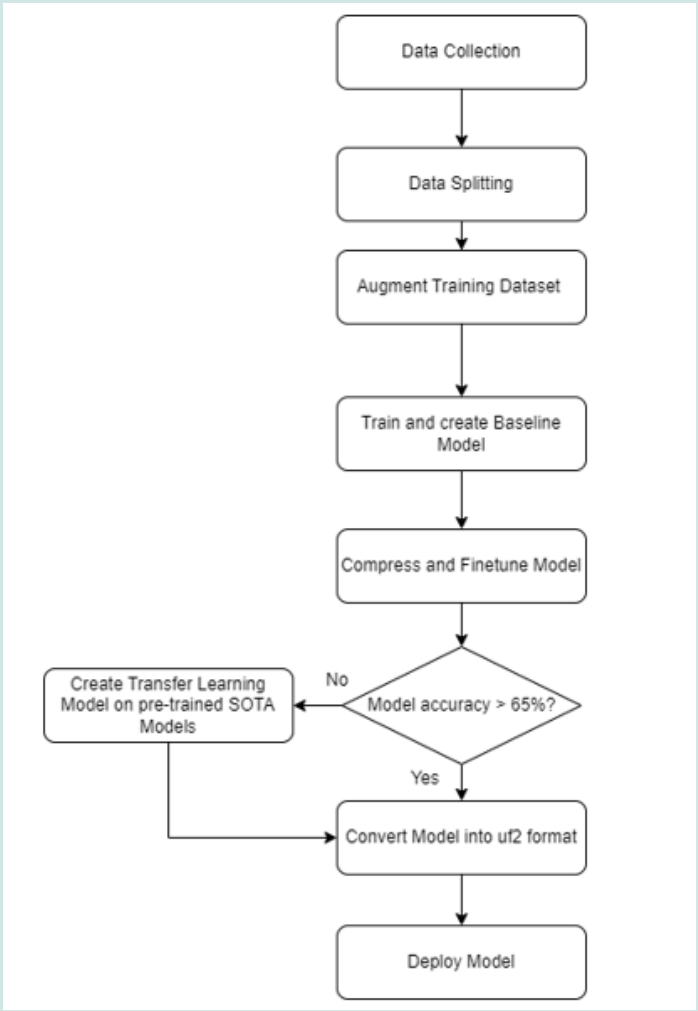

Figure 3: Automated Satellite Model Creation flowchart for Rpi-Pico.

Data Collection and Splitting

A dataset is always a prerequisite for creating any neural network model [8]. For this research project, a combination of the Euro SAT dataset and UC Merced land use dataset is used. The Euro SAT dataset comprises satellite images of land cover across Europe, containing 10 diverse classes like urban, forest, and agricultural areas. With over 27,000 images, it serves as a benchmark for remote sensing tasks. Conversely, the UC Merced dataset focuses on land use in the US, encompassing 21 classes such as residential, airplane, and river. Featuring 2,100 images, it aids research in image classification and analysis. Both datasets are pivotal for advancing machine learning applications in Earth observation and geospatial analysis. These datasets are downloaded and partitioned accordingly, with 70% belonging to training data, 20% as validation data, and 10% as test data, for developing the neural network model. Data augmentation [9], illustrated in (Figure 4), is also performed to increase the size of the individual partitions. It is the technique of artificially expanding a dataset by applying various transformations to the existing data.

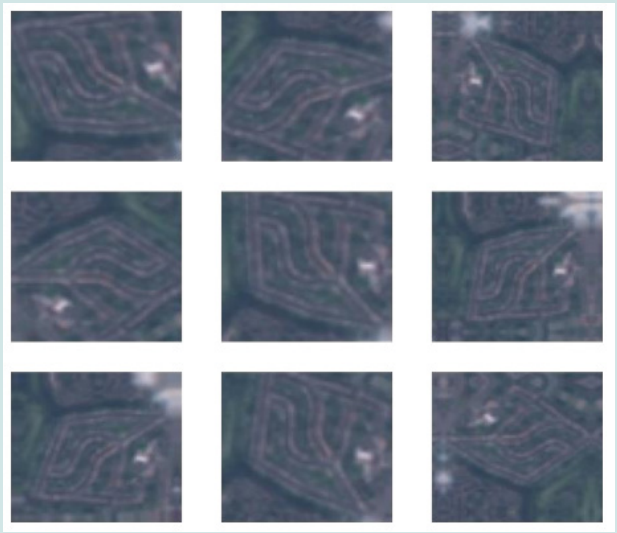

Figure 4: Data Augmentation of a single geospatial image.

Preprocessing Data and Preparing Baseline

Given the Euro SAT dataset's 10 classes, the initial random baseline stands at 10% [10], but a more effective benchmark can be set through non-deep learning approaches. A meticulous process is employed to define this foundational reference point. Firstly, every image within the Euro SAT dataset is partitioned into four distinct squares, each measuring 32x32 pixels. These individual squares undergo a thorough analysis as the red, green, and blue color channels are independently averaged for each square. The ensuing values, amounting to 12 (3 channels x 4 squares), are then channeled into a single-layer, fully connected network.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 12)]              0

 dense (Dense)                (None, 10)                130

=================================================================
Total params: 130
Trainable params: 130
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Simple baseline model for Image Inference, built with EuroSAT dataset.

Remarkably, the achieved accuracy through this approach greatly surpasses the mere 10% represented by the random baseline. This method showcases the potential inherent in non-deep learning techniques for setting a more insightful performance benchmark. Therefore, to create the baseline, a single layer fully connected neural network model is created, as shown in (Figure 5). The model was able to achieve a validation accuracy of 30.85%, which is greater than the random baseline of 10%., however other techniques must be utilized to create automated inference for the geospatial data, with higher accuracy.

Exploring and Evaluating Alternative Models for Image Classification

Simple Fully Connected Neural Network

This is the most basic type of neural network model [11]. As shown in (Figure 6), it uses only dense layers to carry out image classification. It overtakes the baseline in terms of accuracy, by having a validation accuracy of 48.74% after training for only 20 epochs. However, this accuracy is still comparatively low for creating a good inference model. So, due to its limited scope and high loss values, a different model is created with a comparatively greater number of convolutional layers.

Basic ConvNet Models with 4 Conv2D Layers

This model is the first convolutional neural network implementation of the Euro SAT dataset. The model comprises 4 Conv2D layers [12] with escalating numbers of filters, resulting in a validation accuracy of 87.59%. This achievement is obtained by using more than 390,000 trainable parameters. Subsequent experimentation has revealed that the Raspberry Pi Pico microcontroller lacks the capacity to run such expansive models due to their size. Nevertheless, the training and analysis of these models remain essential, as they offer insights into the most effective Configuration of neural network layers required for crafting an optimized model. As shown in (Figure 7) and (Figure 8), there appears to be only slight overfitting in the model.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
rescaling (Rescaling)	(None, 64, 64, 3)	0
flatten (Flatten)	(None, 12288)	0
dense (Dense)	(None, 512)	6291968
dense_1 (Dense)	(None, 10)	5130

Total params: 6,297,098
Trainable params: 6,297,098
Non-trainable params: 0

Figure 6: Simple fully connected model architecture.

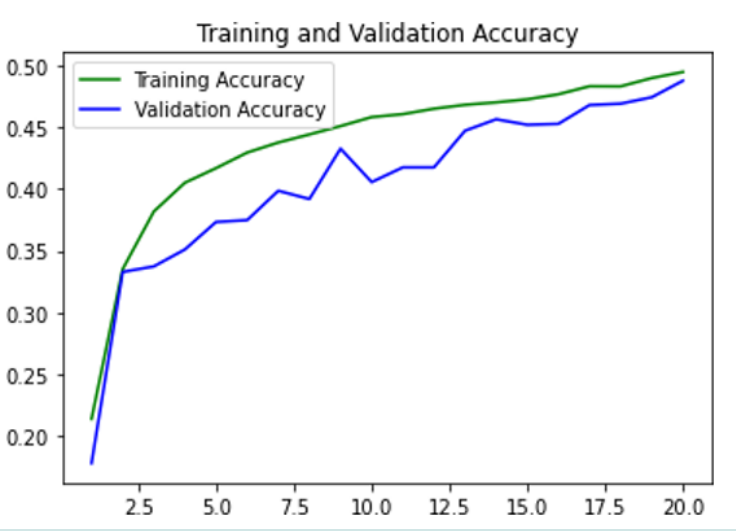

Figure 7: Training and Validation accuracy graph for model with 4 Conv2D layers over 20 epochs.

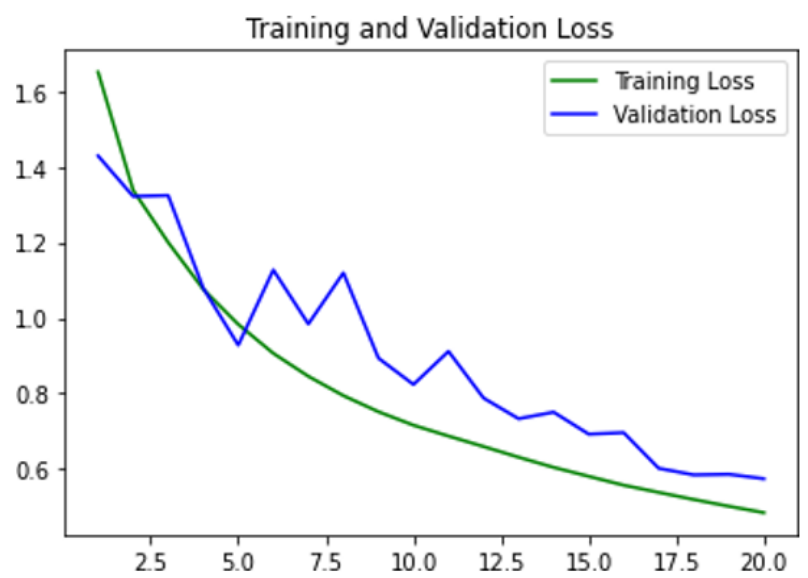

Figure 8: Training and Validation loss graph of Simple Baseline Model for Image Inference, built with EuroSAT dataset over 20 epochs.

CNN model with 4 Conv2D layers and Data Augmentation

This is an upgradation on the previously created model. It adds data augmentation to the mix while creating the neural network model. This is supposed to improve the model accuracy, as the model will see variation of each individual image on every epoch. However, in practicality during the creation of this model, the accuracy decreased from 87.59% to 83.44% after training for 20 epochs. Also, as shown in (Figure 9), this model exhibits high overfitting. Overfitting [13] in neural networks occurs when a model becomes excessively tailored to the training data, performing well on it but poorly on the validation and new data. It signifies the model has memorized noise rather than learned general patterns, resulting in diminished real-world performance. There are various regularization techniques that help mitigate overfitting in neural network models.

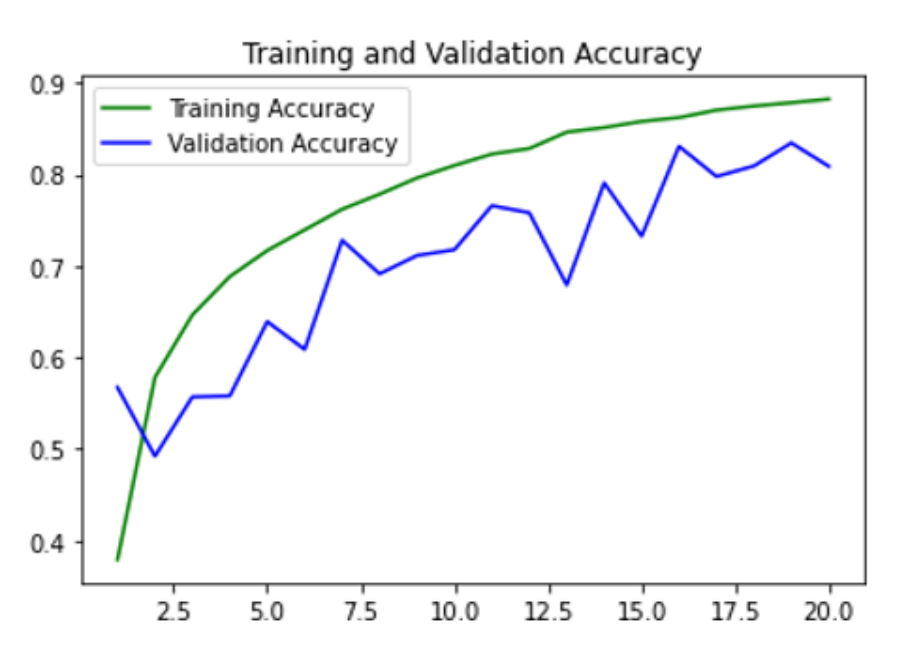

Figure 9: Training and Validation loss graph of Simple Baseline Model for Image Inference, built with EuroSAT dataset over 20 epochs.

Optimal Selection: Creating and Evaluating Alternative Smaller Models for deployment on Rpi-Pico

Raspberry Pi Pico has just 2MB of flash and 264KB of RAM, making it a highly resource-constrained microcontroller. So, the deployed neural network model should also be much smaller for it to act as an independent, autonomous satellite capable of inferencing individual geospatial images. It was previously induced that Conv2D models need under 30 thousand parameters, and SeparableConv2D [14] models need under 20 thousand parameters to run on Rpi-Pico. So higher accuracy models must be created for inference, which also fit into these resource-constrained conditions. Moreover, achieving such accuracy with just 20 epochs of training is impossible. Thus, along with a larger number of epochs, changes must be made to the various parameters, such as dropout rate, learning rate, regularizes, and residual networks, by trial-and-error methods. In this research, several smaller models were created using an iterative approach.

Small Conv2D Model [15]

According to the model structure shown in (Figure 10), this model has only 29,290 parameters, and an additional dropout layer. This layer randomly deactivates a proportion of its neurons during each training iteration, reducing overfitting by promoting more robust and generalized feature learning. The learning rate is also 5 times higher than the default value. Thus, based on these parameters, it was also able to achieve a high accuracy of 87.3% as depicted by the line graph in (Figure 11), which is quite remarkable given that only large models were able to achieve this feat.

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 64, 64, 3)]	0
rescaling_2 (Rescaling)	(None, 64, 64, 3)	0
conv2d_6 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 31, 31, 32)	0
conv2d_7 (Conv2D)	(None, 29, 29, 32)	9248
max_pooling2d_5 (MaxPooling 2D)	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 12, 12, 64)	18496
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650

Total params: 29,290
Trainable params: 29,290
Non-trainable params: 0

Figure 10: Small Conv2D Model structure.

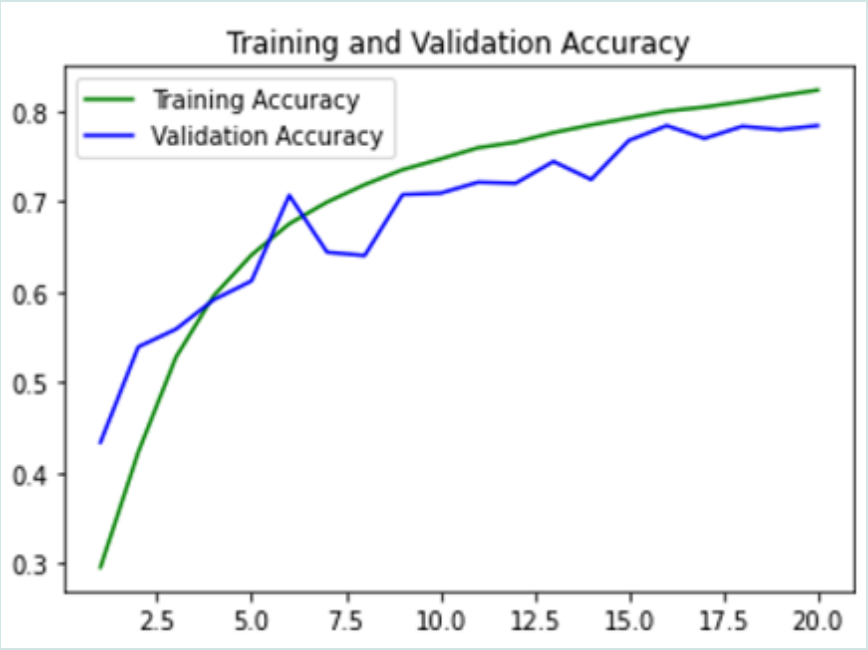

Figure 11: Training and Validation accuracy graph of Small Conv2D model, over 20 epochs.

CNNs with Residuals

During this research, a deliberate strategy involves the incorporation of Residuals [16] and SeparableConv2D layers along with basic Conv2D residual layers. Even though the resulting model is smaller than the previous model, it can achieve a high evaluation accuracy rate of 92.30%, which is clearly illustrated by the graphs in (Figure 12) and (Figure 13). While the original Conv2D layer from the preceding model remains in place due to the strong interrelation among RGB color channels, the focus pivots towards the application of Depth wise Separable Convolution. This dual-pronged approach aims to expedite training and curtail the count of learnable parameters, leading to a noteworthy enhancement in accuracy. Because of this, it can achieve such a high accuracy by training over 60 epochs, with only a total of approximately 18 thousand parameters. But even though this model can achieve such extraordinary accuracy, it cannot be deployed on the Rpi-Pico microcontroller because residuals are not supported on the device. An error was produced during the conversion of the model to TensorFlow Lite version, stating that "Arena size is too small for all buffers." This is also because the Pico has a mere 264KB of memory, whereas the operation and deployment of the model requires a minimum of approximately 385KB of memory to accommodate the model itself, along with the buffers for all inputs and outputs. Therefore, more RAM would be required to run the model.

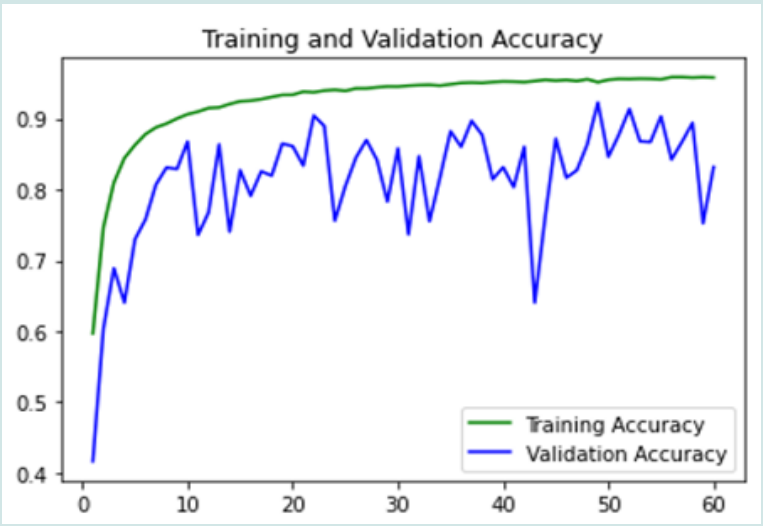

Figure 12: Training and Validation accuracy graph for CNN model with residuals, over 60 epochs.

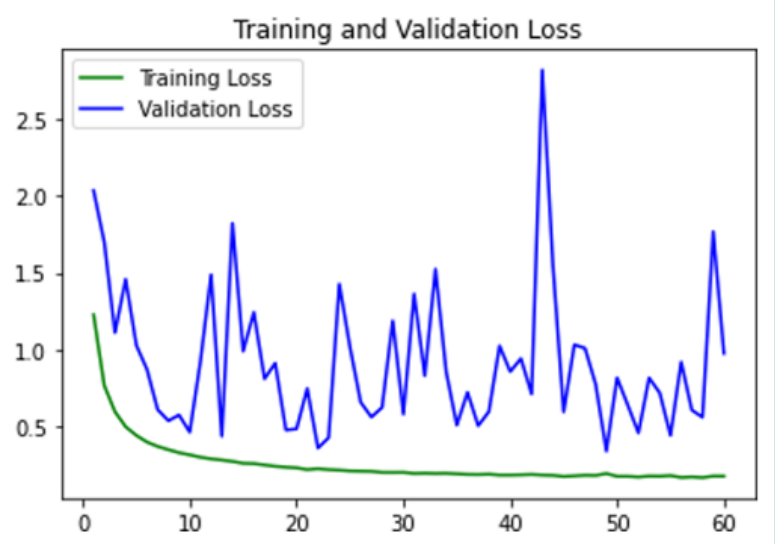

Figure 13: Training and Validation loss graph for CNN model with residuals, over 60 epochs.

Final Neural Network model for Rpi-Pico

Based on insights gained from prior models and the challenges encountered while either creating these models or converting them into compressed forms, the most optimal approach appears to involve employing a simple model architecture combined with an increased number of training epochs to achieve an increased level of accuracy. So instead of training for only 20-30 epochs, the neural network model must be trained for around 125 epochs. By employing this approach, the model achieved an accuracy of 91.33%, which is demonstrated in (Figure 14) and (Figure 15). And, this is comparatively a high accuracy result, compared to the other previously created small neural network models.

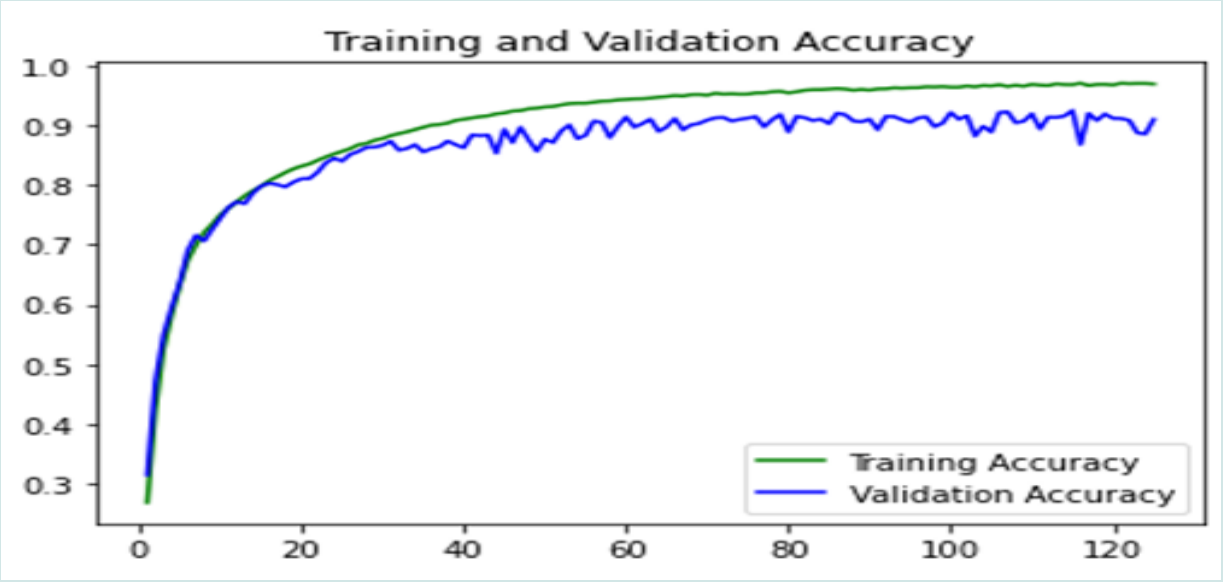

Figure 14: Training and Validation Accuracy graph of Simple small model using only Conv2D and SeparableConv2D layers trained over 125 epochs.

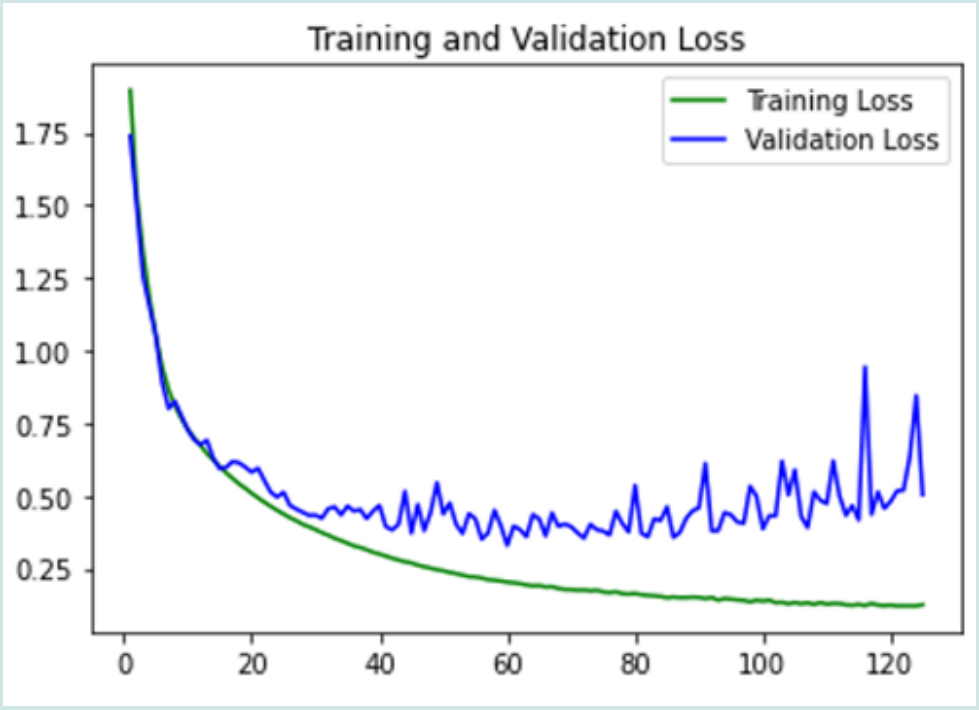

Figure 15: Training and Validation Loss graph of Simple small model using only Conv2D and SeparableConv2D layers trained over 125 epochs.

Model Quantization and Deployment on Rpi-Pico

The training of the model is executed using the Kera's backend. However, the Kera's model itself cannot be directly deployed onto the microcontroller. To facilitate its deployment, a crucial step involves converting it into a lightweight TFLite format [17]. The process of converting a Kera's model to the TFLite format encompasses multiple stages: the model must first be loaded, followed by initializing the converter and configure ring optimizations. Additionally, a pivotal role is played by a Representative Dataset Generator, which supplies sample data to the converter. This ensures that the model is appropriately configured red for production, guaranteeing Tensor Arenas's correctness in terms of types and sizes. While the TFLite model is indeed written to disk, its direct use on the Pico microcontroller is hindered due to the absence of a filesystem for loading files. As a workaround, the TFLite binary undergoes a transformation into a C Header file, thereby enabling seamless embedding of the model directly into C++ codebase, which makes it easier for converting model format. The final TFLite model is subsequently transformed into the compatible Pico format, aligning with the C-style architecture referred to as uf2 format, accompanied by supplementary binary files. Ultimately, the microcontroller is employed for inference by effortlessly dragging and dropping the uf2 file. It is worth noting that the conversion process from Kera's to TFLite introduces optimizations, including quantization, aimed at significantly reducing the model's size. However, these optimizations do entail a minor trade-off in terms of accuracy. To ensure that the model's performance is maintained at a satisfactory level, continuous inference can be executed to gauge its accuracy under operational conditions.

One of the model quantization methods used to compress our neural network models is to convert the floating-point weights into fixed-point weights [18]. This helps by reducing the storage requirements and improving the computational efficiency of neural networks. Fixed-point numbers use a fixed number of bits to represent both the integer and fractional parts of a value, while floating-point numbers typically require more bits and additional storage for the exponent and sign bits. This leads to reduced memory usage for storing model parameters, which is particularly important for deploying models on resource-constrained devices. The process of transitioning from floating-point to fixed- point begins by determining the necessary number of bits, denoted as m, for representing the unsigned integer component:

$$m = 1 + \frac{[\log_2(\max |\, x_i \,|]}{1 \le i \le N} \qquad (1)$$

In Eq. (6), xi represents an element within the floating-point vector x which has a length N. When m is positive, it signifies the need for m bits to express the absolute integer part. Conversely, a negative m implies m unused leading bits for the fractional part. This approach enhances precision for vectors containing values below 2-1. By replacing redundant leading bits with additional trailing bits, greater precision can be achieved. And thus, the remaining n bits for the fractional part can be computed as:

$$n = w - m - 1 \qquad (2)$$

where w is the width of the datatype (e.g., 16 bits). In this equation, subtraction of 1 account for the extra bit needed to accommodate signed number representation. When n is positive, it indicates the availability of n bits for fractional part representation. Conversely, a negative n implies the inability to represent the fractional part and compromises the full precision of the integer part representation. The value of an element, denoted as $xfixed$i, within the fixed-point vector $xfixed$, is calculated based on the element xi from the floating-point vector x in the following manner:

$$xfixed_i = trunc(x_i \times 2^n) \quad (3)$$

The function $trunc(y)$ is utilized to truncate a real number y to its integer component. The scale factor s is defined by the formula:

$$s = 2^{-n} \quad (4)$$

This adjusts the range of values to fit within a limited range, preserving precision by using a smaller bit representation for efficient computation and storage. After model quantization, the conclusive model achieved its peak validation accuracy at 91.33%. However, following the conversion of Kera's model to the compressed TFLite format, the validation accuracy stands at 90.96%, representing a marginal decline of 0.37 percentage points.

Model Inference and Evaluation on Rpi-Pico

After the microcontroller flash has been loaded with the built image of the neural network model, a dedicated script running on the Raspberry Pi's Linux-based operating system orchestrates the automatic sequential transmission of captured images to the Pico for inference, utilizing the SPI port. However, due to the inherent constraints of the SPI port, a transformation process is necessary to convert the images into their corresponding binary format. Subsequently, the inference process is autonomously executed on the Rpi-Pico device in a continuous manner. The outcomes of the model's inference on the captured images are then generated and cataloged as results within a CSV file, as visually depicted in (Figure 17) The outcomes of the inference process are carefully retained, accompanied by a detailed breakdown of results across distinct classes. Considering the extensive scope of the Euro SAT dataset [19], encompassing a total of 10 distinct classes ('Annual Crop', 'Forest', 'Herbaceous Vegetation', 'Highway', 'Industrial', 'Pasture', 'Permanent Crop', 'Residential', 'River', 'Sea Lake'), the output is systematically categorized and aligned in accordance with these specific classes. This segmentation enables a comprehensive evaluation of how the model's predictions correlate with the diverse range of land cover types present within the dataset. Illustrated in (Figure 16) are the detailed findings derived from the analysis of a set of 1000 acquired satellite images. Notably, the operational model showcases a commendable accuracy level, hovering at approximately 93%. This robust performance underscores the effective functioning of the automated satellite system, demonstrating its capability to discern and categorize land formations with a remarkable degree of precision, thereby minimizing errors to a considerable extent. Hence, the achieved outcome effectively highlights the viability of employing deep learning, particularly in the realm of image classification, even within the constraints of a resource limited microcontroller.

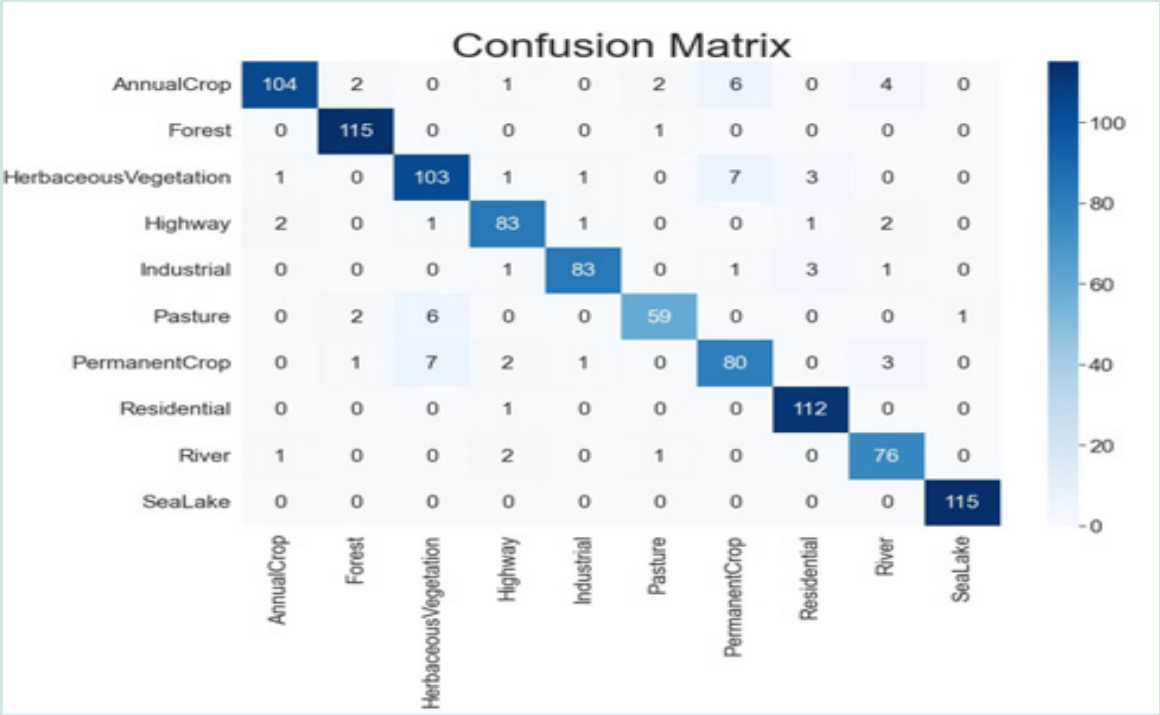

Figure 16: Confusion Matrix for automated inference results by the Raspberry Pi - Pico on 1000 captured geospatial images.

ImagePath	TrueValue	PredValue	Time
Img147_1.bin	1	3	1063
Img148_5.bin	5	2	1064
Img149_3.bin	3	3	1063
Img150_7.bin	7	6	1064
Img151_1.bin	1	1	1064
Img152_2.bin	2	2	1064
Img153_0.bin	0	0	1064
Img154_7.bin	7	7	1065

Figure 17: Sample inference on captured images by the Raspberry Pi Pico microcontroller, saved as a CSV file.

Alternative Works

An alternative transfer learning method can be applied to the pre-trained state-of-the-art models to improve their performance. Transfer learning in neural networks involves leveraging knowledge gained from one task to improve performance on another task. Pre-trained models, typically on vast datasets, are fine-tuned or used as feature extractors, aiding cases with limited data or resources and enhancing efficiency and accuracy. The state-of-the-art models explored in our research for additional comparisons are the ResNet50 neural network and VGG16 models. ResNet-50V2 is a variant of the ResNet-50 architecture renowned for its effectiveness in image analysis. It tackles the challenge of training deep networks by mitigating gradient vanishing by employing residual connections. This version refines the original model, enhancing performance and facilitating smoother gradient flow. ResNet-50V2 is extensively applied in computer vision tasks such as image classification and object recognition, owing to its capacity to extract intricate features from visual data. Its architectural stability and notable performance make it a popular choice for extracting rich hierarchical representations from images in deep learning. So, transfer learning is applied on the ResNet50V2 model [20] by just changing the parameters of the head and tail of the pre-trained model to produce inference of only the necessary 10 classes of geospatial images. In doing so, the final ResNet50V2 transfer learning model was able to achieve a validation accuracy of 89.78%, as depicted in (Figure 18), by training for only 10 epochs. This is because the final model has around 23.5 million total trainable parameters. However, this model also shows some overfitting, as reflected in its much higher training accuracy of 98%. Similar work was also done on the VGG16 model [21]. But according to (Figure 19), it was able to achieve a validation accuracy of only 82.52% over 10 epochs, and its training accuracy was 78.79%. The comparatively low accuracy was because VGG16 has a total of around 14.7 million parameters, which is lower than the ResNet50V2 model. Hence, the ResNet50V2 model demonstrated remarkable proficiency in classifying and inferring from satellite images. This success, therefore, prompts further exploration of other analogous or enhanced models aimed at validating their accuracy and potential deployment on the Rpi-Pico microcontroller.

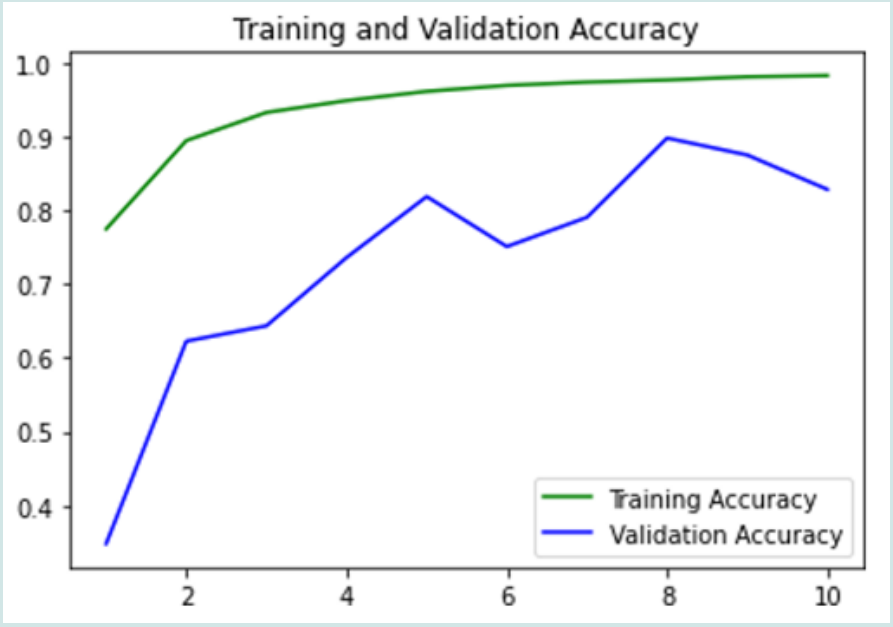

Figure 18: Training and Validation Accuracy graph for creating and applying transfer learning on ResNet50V2 model, over 10 epochs.

Conclusion

The research study aimed to create a cost-effective autonomous satellite system using deep learning on microcontrollers. The chosen microcontroller, Raspberry Pi Pico, had limited resources. Datasets were merged for training, and various neural network models were trained and analyzed. A compact model with conv2D and separableConv2D layers was selected for optimal size and efficiency. This model was optimized for deployment on the Raspberry Pi Pico and achieved 93% validation accuracy in geospatial image inference. The research demonstrated the potential of microcontrollers for autonomous satellites, offering a balance between power and efficiency. It emphasized that deep learning and optimized architectures can enhance satellite autonomy without constant human intervention. The study highlighted microcontrollers' viability and capacity for intelligent tasks in satellite systems, promoting cost-effective and efficient space exploration. On a separate development, satellites are anticipated to be an essential constituent of non-terrestrial networks in 6G mobile communication [22]. Hence, satellites equipped with constrained-resource microcontrollers capable of autonomous learning capability will play a crucial role in the future.

Acknowledgment

This work was partially supported by the National Science and Technology Council, Taiwan, under grant number NSTC 112- 2221-E-197-015.

References

1. Abburu S, Golla SB (2015) Satellite image classification methods and techniques: A review. International Journal of Computer Applications 119(8).
2. Aman E, Jana S, Athikary KG, Suryanarayana RC (2023) AI Inspired ATC, Based on ANN and Using NLP 2023-01-0985. SAE Technical Paper.
3. Torrey L, Shavlik J (2010) Transfer learning. In Handbook of Research on Machine Learning Applications and Trends: Algorithms Methods and Techniques 9(6): 242-264.
4. Bell C (2022) Introducing the Raspberry Pi Pico. In Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects pp. 1-42.
5. Nath O (2020) Review on Raspberry Pi 3B+ and its scope. Int J Eng Appl Sci Technol 4(9): 157-159.
6. McManus S, Cook M (2021) Raspberry Pi for dummies. John Wiley & Sons.
7. Warden P, Situnayake D (2019) Tinyml: Machine learning with TensorFlow lite on Arduino and ultra-low-power microcontrollers. O'Reilly Media.
8. Roh Y, Heo G, Whang SE (2019) A survey on data collection for machine learning: a big data-ai integration perspective. IEEE Transactions on Knowledge and Data Engineering 33(4): 1328-1347.
9. Shorten C, Khoshgoftaar TM (2019) A survey on image data augmentation for deep learning. Journal of big data 6(1): 148.
10. Chan TH, Jia K, Gao S, Lu J, Zeng Z, et al. (2015) PCANet: A simple deep learning baseline for image classification. IEEE Transactions on Image Processing 24(12): 5017-5032.
11. Guo T, Dong J, Li H, Gao Y (2017) Simple convolutional neural network on image classification. In 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA) 721-724.
12. Agarwal M, Gupta S, Biswas KK (2020) A new Conv2D model with modified ReLU activation function for identification of disease type and severity in cucumber plant. Sustainable Computing: Informatics and Systems 30: 100473.
13. Dietterich T (1995) Overfitting and under computing in machine learning. ACM Computing Surveys (CSUR) 27(3): 326-327.

14. Shao J, Qian Y (2019) Three convolutional neural network models for facial expression recognition in the wild. Neurocomputing 355: 82-92.

15. Balaha HM, Hassan AES (2023) A variate brain tumor segmentation, optimization, and recognition framework. Artificial Intelligence Review 56(7): 7403-7456.

16. He F, Liu T, Tao D (2020) Why Resnet works? Residuals generalize. IEEE Transactions on Neural Networks and Learning Systems 31(12): 5349-5362.

17. Verma G, Gupta Y, Malik AM, Chapman B (2021) Performance evaluation of deep learning compilers for edge inference. In 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) pp. 858-865.

18. Novac PE, Boukli Hacene G, Pegatoquet A, Miramond B, Gripon V (2021) Quantization and deployment of deep neural networks on microcontrollers. Sensors 21(9): 2984.

19. Helber P, Bischke B, Dengel A, Borth D (2019) Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 12(7): 2217-2226.

20. Reddy ASB, Juliet DS (2019) Transfer learning with ResNet-50 for malaria cell-image classification. In 2019 International Conference on Communication and Signal Processing (ICCSP) pp. 0945-0949.

21. Tammina S (2019) Transfer learning using VGG-16 with deep convolutional neural network for classifying images. International Journal of Scientific and Research Publications (IJSRP) b9(10): 143-150.

22. I Kang Fu, Gilles Charbit, Abdelkader Medles, Debby Lin, Shao Chou Hung, et al. (2023) Satellite and terrestrial network convergence on the way toward 6G. IEEE Wireless Communications 30(1): 6-8.

DOI: 10.32474/ARME.2023.04.000182