Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 18 No. 12 (2024) | OPEN ACCESS

https://doi.org/10.3991/ijim.v18i12.47895

PAPER

On-Device Neural Network for Object Train and Recognition using Mobile

Md. Salah Uddin[1,2](✉), Wolfgang Slany[1], Kazi Jahid Hasan[2]

[1]Institute of Software Technology, Graz University of Technology, Graz, Austria

[2]Multimedia and Creative Technology, Daffodil International University, Dhaka, Bangladesh

headmct@daffodilvarsity.edu.bd

ABSTRACT

A neural network is a machine learning (ML) program or model that processes information and recognizes patterns, similar to the human brain. The neural network algorithm operates by training on data to learn and enhance its accuracy. If there is any mistake in the learning process, the machine will react unexpectedly and produce incorrect information. So, whenever a neural network model is developed, it is mandatory to evaluate its performance and ensure its output is accurate. Cameras, touchscreens, internet connectivity, and powerful CPUs have contributed to the popularity of smartphones. Mobile apps are software applications that run on smartphones. ML enhances mobile app functionality by offering features such as voice recognition, image analysis, natural language processing, personalization, and recommendations. Training ML models using mobile apps is challenging due to limited resources, data, and privacy concerns. Object recognition is a neural network-based technique that enables the identification and localization of objects in an image or video. This technique is used in driverless cars, disease identification, industrial inspection, robotics, and more. In this paper, the author introduces a new neural network-based algorithm that utilizes mobile devices for training and recognizing images. Although mobile devices have some technological limitations for training, well-established guidelines for systematically mapping verification and validation techniques have been used in the proposed neural network to ensure performance and correctness in on-device object training and recognition.

KEYWORDS

mobile application, neural network, object train and recognition, verification and validation

1 INTRODUCTION

In recent years, machine learning (ML) has transformed the world by achieving human-level performance in a variety of tasks, such as speech recognition, image classification, autonomous cars, malware detection, unmanned aerial vehicles (UAVs), fraud detection, heart failure detection, and aircraft collision avoidance systems. However, the ongoing dangerous incidents linked to such systems are

Uddin, M.S., Slany, W., Hasan, K.J. (2024). On-Device Neural Network for Object Train and Recognition using Mobile. *International Journal of Interactive Mobile Technologies (iJIM)*, 18(12), pp. 99–111. https://doi.org/10.3991/ijim.v18i12.47895

Article submitted 2024-01-12. Revision uploaded 2024-03-08. Final acceptance 2024-03-20.

a significant cause for concern. For instance, a Google self-driving car recently collided with a bus due to unforeseen circumstances [7], while a Tesla car collided with a trailer without detecting it as an obstacle [8]. To prevent such mishaps, there is a growing interest in researching testing methodologies for ML systems. In any case, the testing of ML frameworks presents new challenges to the programming testing community. The operation of ML frameworks depends on the sets of input data; a small change in the training data can significantly affect how the framework functions and the outcomes of the learning process. Some academics have labeled such frameworks as "non-testable" using existing methodologies, which necessitates the development of novel systems to test them [9]. Several strategies for testing ML algorithms have recently been developed [10]. These methods can be used to investigate and assess the oversimplification and robustness of ML programs.

Due to constraints such as storage, memory, computing resources, and battery consumption, on-device ML poses a significant barrier for many users. Due to these constraints, training a model on a server or GPU is now feasible. On-device ML will eventually democratize technology, providing mobile developers with the resources to design applications that will benefit consumers worldwide, regardless of their connectivity situation. Data processing can occur on the user's device by leveraging on-device ML. This indicates that anyone can use ML to protect sensitive data that should not leave the device.

Image classification is an ML method that enables a machine to categorize a batch of photographs into multiple categories. Photos can be submitted to a model, whether trained or untrained, for the model to accurately identify all photos using neural networks and deep learning techniques. The model is a pre-programmed set of instructions that the mobile device can apply to a set of photos to classify the dataset into classes or categories. To construct an image training model for recognition on a mobile device, several challenging obstacles must be overcome. Additionally, greater emphasis should be placed on validation and verification procedures to prevent unexpected accidents.

In this paper, systematic mapping (SM) is used to evaluate a newly developed neural network for on-device image training in ML systems.

2 BACKGROUND STUDY

V&V is the formal process of testing and examining a system for dependability, reliability, and assurance. Verification attempts to answer the question, "Is the product being built correctly?" Validation attempts to answer the question, "Is the right product being built?"

Kurd et al. [1] describe the development of a health case to justify the use of artificial neural networks (ANNs) in fundamental security applications. They attempted to establish security measures for ANNs that should be implemented in healthcare settings. Aside from that, they mention issues in balancing execution and security.

Burton et al. [2] highlighted existing safety concerns that arise when using AI algorithms in highly automated driving. To achieve this goal, the author has employed a confirmation case structure based on the Goal Structuring Notation [3], which also highlights the necessity for further research on innovative verification procedures. Special emphasis was placed on potential strategies that can be used to reduce utilitarian limitations in insight capacities based on convolutional neural networks. This study has recently been published in [4]. Cheng et al. [5] discuss the challenges of accrediting reliable brain research groups. As a result, the inadequacy of traditional design techniques based on V-models is highlighted. Based on this, they describe their proposed enhancements to existing design techniques for ANN security validation.

Rao et al. [6] address specific functional safety issues encountered during the development of deep learning techniques for self-driving cars. The author has discussed the challenges of verifying dataset completeness for training and testing, linking them to the requirements of ANNs, and the process of transfer learning.

Kanewala et al. [11] conducted another SLR by incorporating literature related to testing logical programming. The SLR offers four discussion starters on the definitions, challenges, methodologies, and difficulties of testing logical programming. However, it does not focus on ML programs and has different objectives.

The Oracle issue in the realm of programming testing has received a lot of attention from various scientists, and several approaches have been proposed in the past to tackle it. Barr et al. [12] conducted a comprehensive investigation into the Oracle issue and explored various objective approaches used in the literature.

3 RESEARCH TECHNIQUES AND EMPIRICAL STUDY

A systematic mapping study presented by Peterson et al. [13] and Kitchenham et al. [14] provides unbiased assessments, identifies research gaps, and collects evidence for future directions. Before developing an on-device object training and recognition neural network model, a systematic mapping study technique was followed.

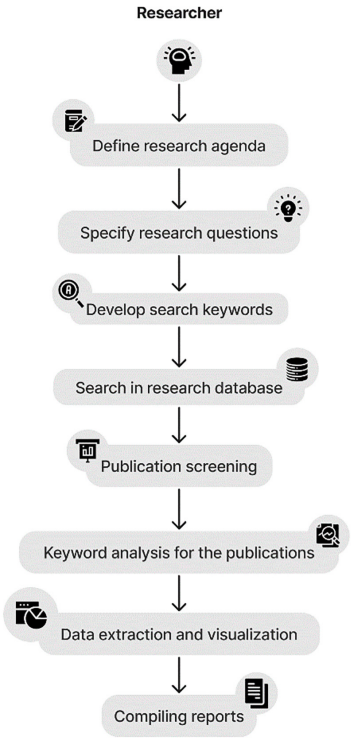

Fig. 1. The research's systematic mapping process

4 PROPOSED METHOD

After conducting a systematic mapping study, a neural network algorithm has been developed that can train images on a device and recognize objects. After the development of the object training and recognition model, various techniques have been employed to verify and validate the proposed algorithm. The on-device neural network working process has been explained in Figure 2.

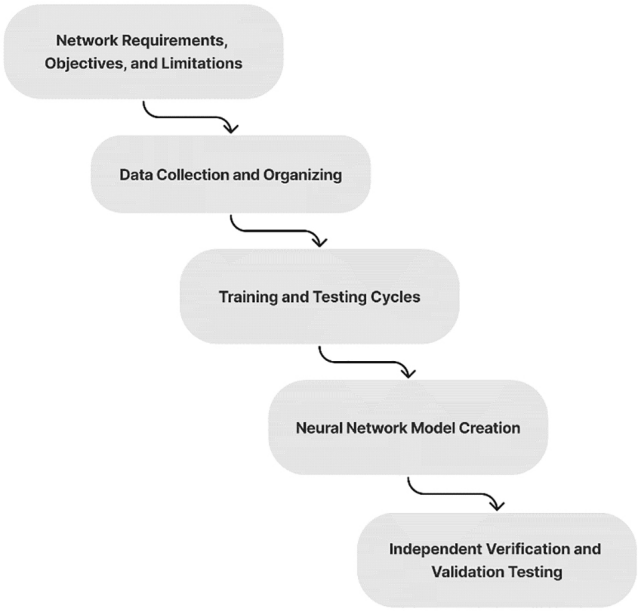

Fig. 2. Proposed on-device neural network development waterfall model

4.1 The architecture of neural network

The learning (or training) phase in the proposed neural network begins by splitting the input into two distinct sets.

1. Training dataset: This dataset enables the neural network to learn the weights of nodes.
2. Test dataset: This dataset is used to calculate the neural network's accuracy and margin of error.

A multilayer neural network architecture has been utilized to develop the proposed algorithm for training data. The proposed neural network consists of three sections, as illustrated in Figure 3. There are three layers: input, hidden, and output. The proposed neural network has been annotated with subscripts L and o to establish a connection between the elements of the hidden and output layers. Furthermore, a neural network has been utilized, with j as the index of the hidden layer elements and i as the indices of the input and output layers.

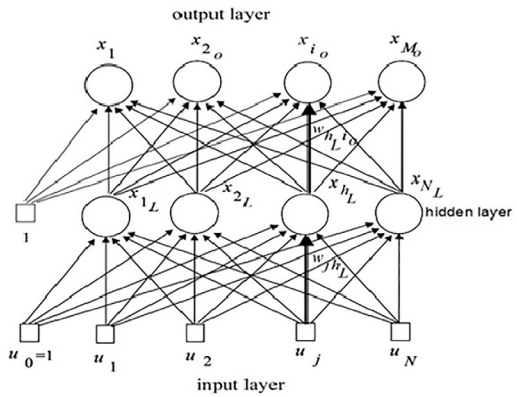

Fig. 3. Neural network architecture

Input layer: The input layer accepts an input image of 28*28 pixels, or 784 input units.

Hidden layer: The input units are associated with weights, so each weight is connected to a hidden unit during the training process. If you create multiple hidden layers, the data training process will be faster. Finally, we used the weight update rule for the hidden layer.

$$w_j h_L(t+1) = w_j h_L(t) - \eta \frac{\partial e^k}{\partial w_j h_L}$$

Output layer: The transfer function at the output layer combines several inputs into one output value by

$$X_{i_0}^k = f_0(W_{i_0}^T X_L^k)$$

Algorithm 1: On-Device Image Training

Step 0. Initialize weights: to small random values;

Step 1. Apply a sample: apply to the input a sample vector $\mathbf{u}^k$ having desired output vector $\mathbf{y}^k$;

Step 2. Forward Phase:

Starting from the first hidden layer and propagating towards the output layer:

Calculate the activation values for the units at layer L as:

If L-1 is the input layer

$$a_{h_L}^k = \sum_{j=0}^{N} w_{jh_L} u_j^k$$

If L-1 is a hidden layer

$$a_{h_L}^k = \sum_{j_{(L-1)}=0}^{N_{L-1}} w_{j_{(L-1)}h_L} x_{j_{(L-1)}}^k$$

Calculate the output values for the units at layer L as:

$$x_{h_L}^k = f_L(a_{h_L}^k)$$

in which use i_o instead of h_L if it is an output layer

Step 4. Output errors: Calculate the error terms at the output layer as:

$$\delta_{i_o}^k = (y_{i_o}^k - x_{i_o}^k) f_o'(a_{i_o}^k)$$

Step 5. Backward Phase Propagate error backward to the input layer through each layer L using the error term

$$\delta_{h_L}^k = f_L'(a_{h_L}^k) \sum_{i_{(L+1)}=1}^{N_{(L+1)}} \delta_{i_{(L+1)}}^k w_{h_L i_{(L+1)}}^k$$

in which, use i_o instead of $i_{(L+1)}$ if L+1 is an output layer;

Step 6. Weight update: Update weights according to the formula

$$w_{j_{(L-1)}h_L}(t+1) = w_{j_{(L-1)}h_L}(t) + \eta \delta_{h_L}^k x_{j_{(L-1)}}^k$$

Step7. Repeat steps 1-6 until the stop criterion is satisfied, which may be chosen as the mean of the total error

$$< e^k > = < \tfrac{1}{2} \sum_{i_o=1}^{M} (y_{i_o}^k - x_{i_o}^k)^2 >$$

4.2 User interface of proposed algorithm

A mobile application has been developed based on the proposed neural algorithm, which focuses on training and recognizing objects on the device. Figure 4 illustrates the number of levels and their names in the training dataset, as well as the neural network architecture for training the dataset. Users are required to specify the number of layers, units per layer, and the number of iterations. If the iteration number is higher, the training dataset will be more accurate, but the training time will also increase. The last image represents the accuracy rates of the training and testing datasets, as well as the training time.

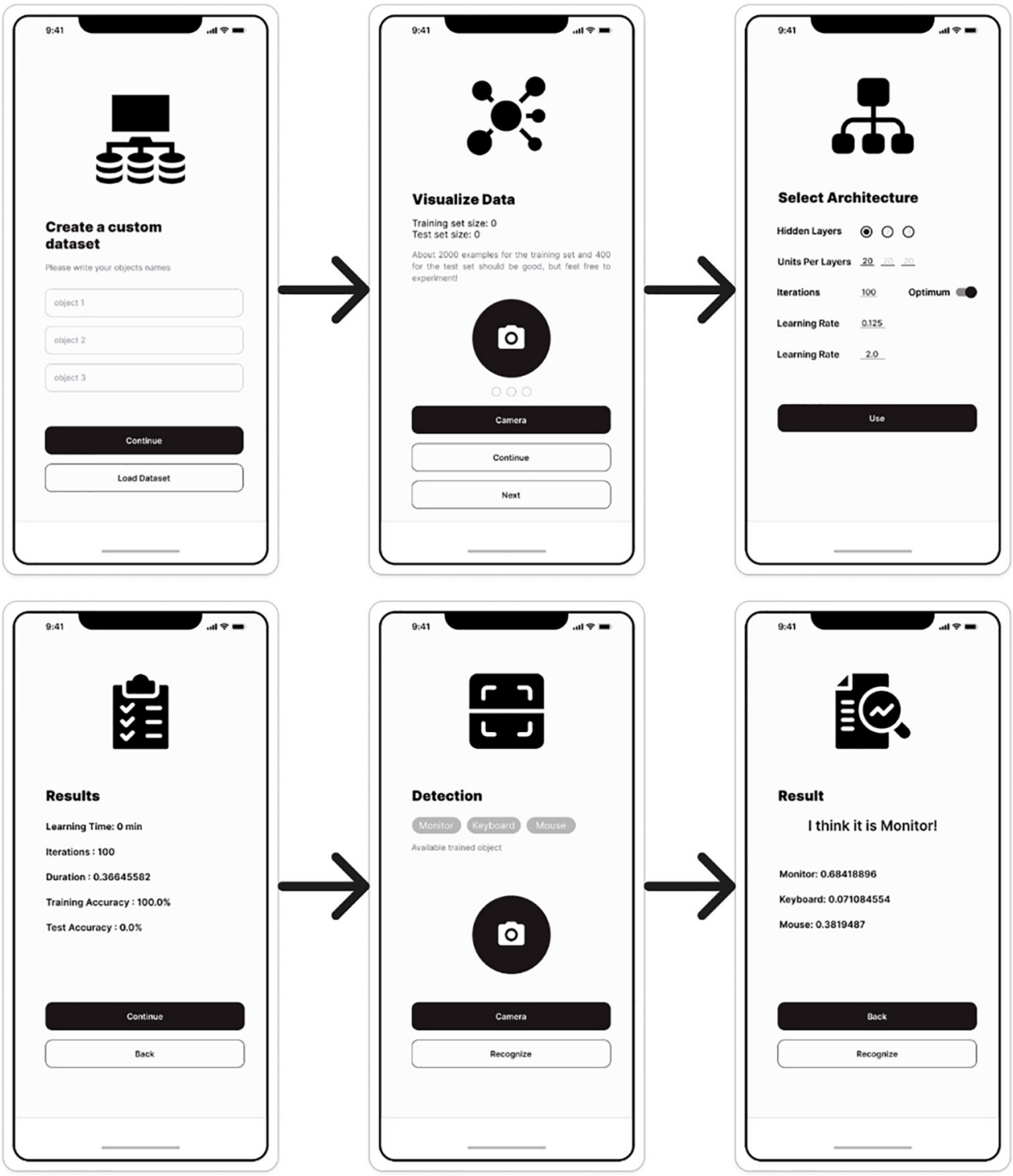

Fig. 4. On-device image train and recognition mobile application

4.3 Suggestions for independent verification of our proposed neural networks

To verify and validate the proposed neural network, the agile framework has been followed. During the verification process, the requirements, design, code, and UI/UX have been tested.

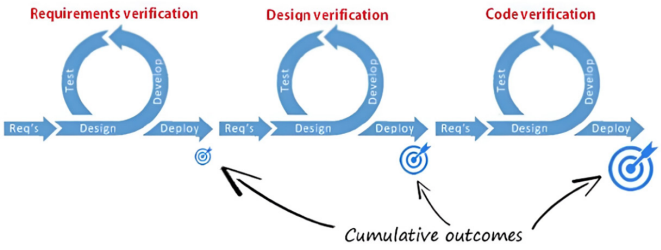

Fig. 5. Agile framework and verification process

Requirement's verification. To verify the performance and accuracy of the proposed neural network, the following criteria have been established: Abstract interpretation, white-box, and branch-and-bound verification approaches have been used.

- Verify the requirements of the learning algorithm.
- Verify neural network performance.
- Verify the accuracy of the stopping criteria.
- Verify the input weights.
- Verify the neural network topology.
- Verify the accuracy of the training set requirements.
- Check the description of the intended output data, the range of those parameters, and the allowed degree of error for optimal system performance.
- Check the validity of each error range for the stability coefficients.
- Check the suitability and application of stability proofs for the adaptive algorithm.
- Check that the weights are not changed as they are input into the system after training.

To verify the proposed neural network, a JUnit testing framework is implemented.

Step 1: Right-click on the "src" folder and select "New" > "Class," then enter a class name and click "OK," The test class location is src/test/java.

Step 2: Add the following code:

```
private val e = 1e-3f // the offset value
private val differenceThreshold = 1e-3 //

private val inputLayerSize = 3
private val numberExamples = 5
private var numberHiddenLayers = 1
private var unitsPerLayer = intArrayOf(5)

@Test
fun checkGradFor1Layer() {
    numberHiddenLayers = 1
    unitsPerLayer = intArrayOf(5)
    checkGrad()
}

@Test
fun checkGradFor2Layer() {
    numberHiddenLayers = 2
    unitsPerLayer = intArrayOf(4,7)
    checkGrad()
}

@Test
fun checkGradFor3Layer() {
    numberHiddenLayers = 3
    unitsPerLayer = intArrayOf(3,5,8)
    checkGrad()
}
```

Fig. 6. JUnit testing implementation

Step 3: Right-click on the Example Test class and select Run As > JUnit Test to see the test result.

Design verification. To start, make sure that the image classification model is trained on a growing number of training photos until it reaches a minimal error with a minimal number of weights. Ensure that the weights are fixed and remain static for future evaluation and implementation using test or production data. Ensure that test data is used to validate the correctness of the image classification model. Finally, compare the neural network output to the intended values to determine the minimum and maximum errors, as well as the average error.

- Examine with a reasonable level of accuracy.
- Conduct sensitivity studies.
- Verify the correct application of scaling.
- Examine the antecedents of the problem.
- Examine the halting criteria.
- Test preparation time
- Monitor the development of the training program.
- Review the practice set.
- Verify to see if the learning algorithm was successfully constructed.

To verify the user interface of the developed application, the Espresso UI testing method has been used.

```
private fun initializeLabels() : ArrayList<FloatArray>{
    val arrayList = ArrayList<FloatArray>(numberExamples)
    for (i in 0 until numberExamples) {
        arrayList.add(vectorizedLabel(i))
    }
    return arrayList
}


private fun vectorizedLabel(i: Int): FloatArray {
    return when (i%3) {
        0-> floatArrayOf(0f,0f,1f)
        1-> floatArrayOf(0f,1f,0f)
        else-> floatArrayOf(1f,0f,0f)
    }
}
```

Fig. 7. Espresso testing implementation

Code and integration verification. White box testing techniques have been used for code and integration verification.

- Check that the PID (parameter identification) data matches the neural network data in the proper format.
- Verify if the trained model accurately classifies the data.

4.4 Instructions for independently validating of proposed neural networks

Validation ensures that the output of the proposed neural network is correct. To ensure validation, the following criteria have been followed:

- Perform unit testing.
- Perform Failure Modes and Effects Analysis (FMEA) testing.
- Test system utilization (on-device memory available, storage, compute resources, and power consumption)
- Perform data security
- Learning time
- Response time for recall
- Reliability
- Reproducibility
- Resistance to faults and robustness

```
private fun calculateNumericGradients(gradsArray: Array<Array<FloatArray>?>?): ArrayList<ArrayList<ArrayList<Float>>>{
    val grads = ArrayList<ArrayList<ArrayList<Float>>>()
    // go over each individual weight
    for (w in gradsArray!!.indices) {
        val rows = ArrayList<ArrayList<Float>>()
        for (r in gradsArray[w]!!.indices) {
            val cols = ArrayList<Float>()
            for (c in gradsArray[w]!![r].indices) {
                val costA = NeuralNetwork.wCostWithWeightModifTest(w,r,c,-e)
                val costB = NeuralNetwork.wCostWithWeightModifTest(w,r,c,e)
                cols.add((costB-costA)/(2f*e))
            }
            rows.add(cols)
        }
        grads.add(rows)
    }
    return grads
}


private fun initializeInput(): ArrayList<FloatArray> {
    val arrayList = ArrayList<FloatArray>(numberExamples)
    for (i in 0 until numberExamples) {
        arrayList.add(FloatArray( size: inputLayerSize+1) { j -> Random.nextDouble(0.0, 1.0).toFloat()})
    }
    return arrayList
}
```

Fig. 8. Espresso testing implementation

5 RESULT AND ANALYSIS

This work has demonstrated the development of an on-device neural network for training and recognizing objects through a mobile application. To check the accuracy rate of the proposed algorithm on the training dataset in a mobile application in this research, a mobile phone with the following specifications was used: CPU:

Octa-core (2x2.4 GHz Cortex-A78 and 6x2.0 GHz Cortex-A55); RAM: 6GB. The following result was observed:

Number of Image	Iteration	Training Accuracy	Test Accuracy	Time
15 (size:3000*4000)	100	100%	0%	0.36645s
21 (size:3000*4000)	100	100%	0%	0.3777s
35 (size:3000*4000)	200	96%	95%	3.9120s

In this research, to determine the accuracy rate of object recognition from an on-device trained model, a total of 50 test images were used. The computer contains 25 data, the mouse 15, and the mobile 10 data. The results show TP = 30, TN = 19, FP = 1, and FN = 0. Applying the confusion matrix formula ((TP + TN) *100%) / (TP + TN + FP + FN), the research achieved 98% accuracy in object recognition.

6 CONCLUSION

In this article, a systematic mapping study explores the construction of on-device object training models using mobile applications, along with the verification and validation of a proposed neural network. All types of mobile devices can utilize the proposed neural network without any additional limitations. Both RAM and image count affect training time.

Furthermore, training time will be optimized with a lower error rate, and the algorithm will be compatible with various platforms.

7 ACKNOWLEDGEMENTS

I thank all my colleagues at the department. I thank the Department of Multimedia and Creative Technology and my doctoral supervisor, Wolfgang Slany, for their feedback and valuable recommendations that helped bring this paper to its final form.

8 REFERENCES

[1] Z. Kurd and T. Kelly, "Establishing safety criteria for artificial neural networks," in *Knowledge-Based Intelligent Information and Engineering Systems (KES 2003)*, Lecture Notes in Computer Science, V. Palade, R. J. Howlett, and L. Jain, Eds., Springer, Berlin, Heidelberg, 2003, vol. 2773, pp. 163–169. https://doi.org/10.1007/978-3-540-45224-9_24

[2] S. Burton, L. Gauerhof, and C. Heinzemann, "Making the case for safety of machine learning in highly automated driving," in *Computer Safety, Reliability, and Security (SAFECOMP 2017),* Lecture Notes in Computer Science, S. Tonetta, E. Schoitsch, and F. Bitsch Eds., Springer, Cham, 2017, vol. 10489, pp. 5–16. https://doi.org/10.1007/978-3-319-66284-8_1

[3] Tim Kelly and Rob Weaver, "The goal structuring notation – A safety argument notation," in *2004 Proc. of Dependable Systems and Networks Workshop on Assurance Cases*, 2004.

[4] L. Gauerhof, P. Munk, and S. Burton, "Structuring validation targets of a Machine learning function applied to automated driving," in *Computer Safety, Reliability, and Security (SAFECOMP 2018)*, 2018, vol. 11093, pp. 45–58. https://doi.org/10.1007/978-3-319-99130-6_4

[5] C. Cheng *et al.*, "Neural networks for safety-critical applications—Challenges, experiments and perspectives," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, 2018, pp. 1005–1006. https://doi.org/10.23919/DATE.2018.8342158

[6] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, New York, NY, USA, ACM, 2018, pp. 35–38. https://doi.org/10.1145/3194085.3194087

[7] C. Ziegler, "A Google self-driving car caused a crash for the first time," *The Verge*, 2016. http://www.theverge.com/2016/2/29/11134344/google-selfdriving-car-crash-report

[8] Tesla-accident. Understanding the fatal tesla accident on autopilot and the NHTSA probe. 2016. Available from: https://electrek.co/2016/07/01/understandingfatal-tesla-accident-autopilot-nhtsa-probe/.

[9] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal*, vol. 26, no. 4, pp. 1373–1413, 2018. https://doi.org/10.1007/s11219-017-9392-4

[10] M. R. Lyu, "Software reliability engineering: A roadmap," in *Future of Software Engineering (FOSE '07)*, 2007, pp. 153–170. https://doi.org/10.1109/FOSE.2007.24

[11] Upulee Kanewala and James M. Bieman, "Testing scientific software: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1219–1232, 2010. https://doi.org/10.1016/j.infsof.2014.05.006

[12] Earl T. Barr, Mark Harman, Phil McMinn, Muazammil Shahbaz, and Shin Yoo, "The Oracle problem in software testing: AYY survey," *IEEE Transaction on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015. https://doi.org/10.1109/TSE.2014.2372785

[13] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015. https://doi.org/10.1016/j.infsof.2015.03.007

[14] Barbara Kitchenham, O. Pearl Brereton, David Budgen, MarkTurner, John Bailey, and Stephen Linkmana, "Systematic literature reviews in software engineering – A systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009. https://doi.org/10.1016/j.infsof.2008.09.009

[15] C. Cheng *et al.*, "Neural networks for safety-critical applications—Challenges, experiments and perspectives," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, 2018, pp. 1005–1006. https://doi.org/10.23919/DATE.2018.8342158

[16] Z. Ma *et al.*, "A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles," *Robotics and Autonomous Systems*, vol. 100, pp. 108–118, 2018. https://doi.org/10.1016/j.robot.2017.10.009

[17] R. R. Zakrzewski, "Verification of performance of a neural network estimator," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, Honolulu, HI, USA, 2002, vol. 3, pp. 2632–2637. https://doi.org/10.1109/IJCNN.2002.1007559

[18] S. Burton, L. Gauerhof, and C. Heinzemann, "Making the case for safety of machine learning in highly automated driving," in *Computer Safety, Reliability, and Security (SAFECOMP 2017),* Lecture Notes in Computer Science, S. Tonetta, E. Schoitsch, and F. Bitsch Eds., Springer, Cham, 2017, vol. 10489, pp. 5–16. https://doi. org/10.1007/978-3-319-66284-8_1

[19] J. Schumann and Y. Liu, Eds., "Application of neural networks in high assurance systems," in *Studies in Computational Intelligence*, vol. 268, pp. 1–19, 2010. https://doi.org/10.1007/978-3-642-10690-3

[20] X. Xu, Y. Lei, and F. Yang, "Railway subgrade defect automatic recognition method based on improved faster R-CNN," *Scientific Programming*, vol. 2018, pp. 1–12, 2018. https://doi.org/10.1155/2018/4832972

[21] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, New York, NY, USA, ACM, 2018, pp. 35–38. https://doi.org/10.1145/3194085.3194087

[22] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks. *Computer Aided Verification. CAV 2010*, Lecture Notes in Computer Science, T. Touili, B. Cook, and P. Jackson, Eds., Springer, Berlin, Heidelberg, 2010, vol. 6174. https://doi.org/10.1007/978-3-642-14295-6_24

[23] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, 2017, vol. 10426, pp. 97–117. https://doi.org/10.1007/978-3-319-63387-9_5

[24] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *Pre Print arXiv no. 1803.04792*, 2018. https://doi.org/10.48550/arXiv.1803.04792

[25] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui.

[26] Xue, Bo Li, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang, "DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," CoRR, abs/1803.07519, 2018.

[27] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 3–18. https://doi.org/10.1109/SP.2018.00058

[28] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M. Pawan Kumar, "Piecewise linear neural network verification: A comparative study," *arXiv,* no. 1711.00455, 2017.

[29] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis, (ATVA 2017)*, Lecture Notes in Computer Science, D. D'Souza and K. Narayan Kumar, Eds., Springer, Cham, 2017, vol. 10482, pp. 269–286. https://doi.org/10.1007/978-3-319-68167-2_19

[30] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *Pre Print arXiv no. 1803.04792*, 2018. https://doi.org/10.48550/arXiv.1803.04792

[31] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating System Principles (SOP '17),* 2017, pp. 1–18. https://doi.org/10.1145/3132747.3132785

[32] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18),* 2018, pp. 109–119. https://doi.org/10.1145/3238147.3238172

[33] L. T. S. Valdez, R. Y. Cafino, A. T. Isla Jr., E. J. T. Ignacio, P. J. Labra, and L. C. P. Velasco, "Mobile applications in Otology: A scoping review," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 18, no. 4, pp. 124–144, 2024. https://doi.org/10.3991/ijim.v18i04.45063

[34] J. Chi, C. Kim On, H. Zhang, and S. S. Chai, "A review of deep convolutional neural networks in mobile face recognition," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 17, no. 23, pp. 4–19, 2023. https://doi.org/10.3991/ijim.v17i23.40867

[35] N. I. Nife and M. Chtourou, "Improved detection and tracking of objects based on a modified deep learning model (YOLOv5)," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 17, no. 21, pp. 145–160, 2023. https://doi.org/10.3991/ijim.v17i21.45201

9 AUTHORS

Md. Salah Uddin is an Assistant Professor and the head of the Department of Multimedia and Creative Technology at Daffodil International University. He is a PhD researcher at the Technical University of Graz, Austria. His research interests include machine learning, data science, big data, cloud computing, game theory, and AR/VR. He can be contacted at salah.mct@diu.edu.bd.

Wolfgang Slany is a Professor at the Institute of Software Technology at Graz University of Technology and the head and founder of the Catrobat nonprofit free open-source project. His research interests include poverty alleviation through coding education for teenagers, with a focus on girls, refugees, and adolescents in developing countries. Slany received a Ph.D. in applied computer science and artificial intelligence from the Vienna University of Technology. He conducts research, teaches, and consults on sustainable large-scale agile software development and user experience topics for mobile platform projects.

Kazi Jahid Hasan is a Lecturer in the Multimedia and Creative Technology department at Daffodil International University. His primary research interests include 3D modeling, lighting, and rendering, as well as human-computer interaction, extended reality (XR), and 3D printing.