A Real-time Hand Gesture Recognition System on Raspberry Pi: A Deep Learning-based Approach

Alyssa Yu*, Cheng Qian†, and Yifan Guo‡
*Poolesville High School, Poolesville, MD, USA 20837
†Department of Computer Science and Information Technology, Hood College, Frederick, MD, USA 21701
‡Department of Computer and Information Sciences, Towson University, Towson, MD, USA 21252
Emails: xiaoyumd@gmail.com, qian@hood.edu, yguo@towson.edu

2024 IEEE 21st Consumer Communications & Networking Conference (CCNC) | 979-8-3503-0457-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCNC51664.2024.10454652

***Abstract*—Recent years have witnessed the deep involvement of hand gesture recognition in Internet of Things (IoT) devices in healthcare, autonomous driving, virtual reality, augmented reality, etc., since hand gestures offer a natural and intuitive way for human beings to interact with IoT devices without relying on traditional input methods such as keyboards, mice, or touchscreens. Thus, accurate gesture recognition is crucial to its development. Consequently, many recognition approaches are proposed, from the traditional computer vision domain to the deep learning domain. Although with promising recognition performance, these approaches typically come with high computational and energy costs relying on graphics processing unit (GPU) accelerations, which cannot be compatible with low-cost and non-GPU IoT edge devices. To this end, in this paper, we develop an artificial intelligence (AI)-enabled system for real-time hand gesture recognition on low-cost edge devices, e.g., Raspberry Pis. Particularly, we first design a simple but effective convolutional neural network (CNN)-based model with residual blocks to handle American Sign Language (ASL) digits tasks, which enables fast-bust-accurate real-time inferences. Then, to fit the low-cost environment for edge devices, we involve model quantization techniques to shrink the model size, thus requiring reduced memory and storage for edge devices. In addition, we plug a motion sensor into our system to automatically turn off our recognition once it detects no people nearby, reducing energy consumption. By evaluating the performance of real-time hand gesture recognition, our system maintains a high prediction accuracy rate, e.g., over 96 %. Moreover, our designed solution has the potential to be privileged to other common low-cost edge devices.**

***Index Terms*—Deep learning, Raspberry Pi, American sign language, Hand gesture recognition, Edge intelligence, Internet of Things**

I. Introduction

The Internet of Things (IoT) has revolutionized modern communication, industry, and innovation [1]. It has allowed us to communicate with smart home devices, monitor patient health at home, and connect home applications, enabling numerous smart systems. In particular, hand gesture recognition plays a vital role in IoT technologies. There are many benefits to incorporating hand gesture recognition into IoT technologies. For instance, people with hearing disabilities can use American Sign Language (ASL) to interact with smart devices. Users can control home settings using simple hand movements. Additionally, hand gesture recognition provides a safe and comfortable way of driving because drivers can change settings and temperature without removing their hands from the steering wheel, which can improve driving safety. Furthermore, with the recent development of virtual reality and augmented reality, companies such as GestureTek and Microsoft have created technologies that allow users to communicate with devices and play video games using hand movements. According to Fortune Business Insights, the gesture recognition market is projected to be valued at 115.70 billion USD by 2031 [2].

To widely deploy real-time gesture detection, it is critical to ensure that gesture detection is compatible with low-cost devices, including smart IoT sensors or actuators. However, most existing hand gesture recognition approaches [3]–[12] focus on designing complicated and sizeable deep learning models with tens or hundreds of millions of parameters, with the aid of multiple GPUs' accelerations, to achieve high accuracy rates. Few studies have considered hand gesture recognition on devices with limited memory and computational power.

In this paper, we develop a deep learning-based system for hand gesture recognition on Raspberry Pi. In particular, to maintain the strict performance requirements of high prediction accuracy, real-time inference, and low power consumption, we conduct a set of optimizations on our system from the following four aspects: (i) *Input Optimization.* We apply calibrated image filters on collected hand gesture images to reduce the image size from 48 KB to 3 KB each, with a 94 % decrease. Also, filtered images remove unimportant features and background noises from gesture images while keeping sharp edges, helping produce a high accuracy rate. (ii) *Model Architecture Optimization.* We design a simple but effective Convolutional Neural Network (CNN) model with residual blocks to handle ASL digits tasks, which enables fast-bust-accurate real-time inferences. It can achieve accuracies greater than 96 % with only 610 K model parameters, which are approximately 10 % of the model parameters of the AlexNet model and 0.44 % of that of the VGG16 model. (iii) *Model Storage Optimization.* We apply post-training model quantization techniques with Tensorflow lite, which converts the high-precision floating point parameters to relatively lower-precision representations with negligible accuracy drops. This further decreases the storage size of the trained model by approximately 75 %, lowering the memory footprint and computational requirements so that it can be deployed on Raspberry Pis for fast hand gesture detection. (iv) *Energy Consumption Optimization.* We integrate

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.

a motion sensor into our system to automatically turn off our recognition once it detects no people nearby, which enables low energy consumption costs.

In summary, the contributions of this paper are listed below:

- We develop an artificial intelligence (AI)-enabled system for hand gesture recognition on low-cost edge devices, e.g., Raspberry Pis. To the extent of our knowledge, this is the first paper to consider AI-enabled real-time hand gesture recognition on low-cost devices.
- We design a simple but effective CNN model with residual blocks to handle ASL digits tasks by providing fast-bust-accurate on-device inferences, which can effectively fit the real-time requirement for hand gesture recognition.
- To fit the low-cost environment for edge devices, we involve model quantization techniques to shrink the model size, thus reducing memory cost when running on edge devices. Also, we integrate a motion sensor to automatically turn off our recognition system once it detects no people nearby, which lowers energy consumption.
- By evaluating the performance of real-time hand gesture recognition, our system maintains a high prediction accuracy rate, e.g., over 96 %. Moreover, our system can potentially be extended to other common low-cost edge devices.

The rest of this paper is organized as follows: Section II introduces the related work on hand gesture recognition and deep learning. Section III presents our testbed environment. Section IV depicts our designed system in detail. Section V presents our experiment methodology and evaluation results. Section VI discusses the potential applications of our work and extensions. Finally, Section VII concludes this paper.

II. Related Work

A. Hand Gesture Recognition and Its Applications

Hand gesture recognition has witnessed a wide range of approaches, including both non-deep learning and deep learning-based ones. Traditional non-deep learning approaches include rule-based methods that rely on geometric features [13], and machine learning techniques using extracted features and classifiers like k-Nearest Neighbors [14] or Support Vector Machines [15]. With the success of deep neural networks in the past decade, deep learning-based approaches have surged in popularity with the advent of CNN excels in image-based gesture recognition [4], [5] and Recurrent Neural Networks (RNN), which are used for sequence-based gesture recognition in videos [6]. Hybrid models combining CNN and RNN have also been developed for recognizing gestures in video sequences [7]. Its applications span from human-computer interaction (HCI) to virtual and augmented reality, autonomous driving, robotics, healthcare, etc.

B. American Sign Language Recognition and Its Applications

In general, ASL recognition technology has broader applications in healthcare, education, and entertainment. For example, it can be used to develop systems that translate ASL into text or speech, facilitating communication with non-signers [16]. Also, it can be used to create interactive learning platforms for teaching ASL to both deaf and hearing individuals [17]. Moreover, ASL recognition can be used to develop video games and entertainment applications that allow users to interact using ASL gestures [18].

ASL recognition, as a special task in hand gesture recognition, can significantly benefit deaf and hard-of-hearing individuals by enabling computers to interpret sign language, improving their access to technology, communication, and information. This can enhance educational and employment prospects, social integration, and overall life quality. By bridging language gaps between the deaf and hearing communities, sign language recognition promotes inclusivity. It also boosts accessibility across industries, such as healthcare, customer service, and emergency services.

For example, Zafrulla *et al.* [8] studied the use of the Kinect depth-mapping camera for carrying out sign language recognition that could be used for educational games with deaf children. Garcia *et al.* [9] utilized the transfer learning techniques for ASL recognition on a pre-trained GoogLeNet model trained on the ILSVRC2012 dataset. Bantupalli *et al.* [10] took video sequences, extracted temporal and spatial features from ASL gestures, and used a CNN for recognizing spatial characteristics and an RNN to train on temporal features correspondingly. Shin *et al.* [11] proposed a media-pipe hands algorithm to estimate hand joints using RGB hand gesture images collected by a web camera. Likewise, Mohammad *et al.* [12] developed several spiking neural network models for classifying static ASL hand gestures (i.e., ASL alphabet, ASL digits). However, all these methods are designed with complicated and large and complex deep learning models with tens or hundreds of millions of parameters, which cannot be compatible with low-cost devices due to limited computational and memory resources.

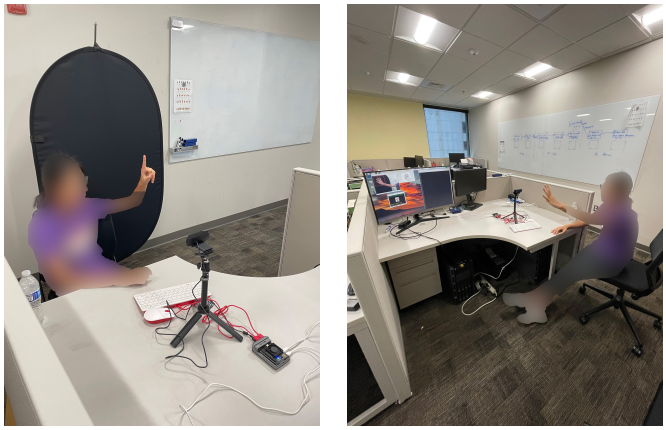
III. Our Testbed Environment

This section introduces our experimental platform for hand gesture recognition on computationally constrained devices. We categorize the function of our testbed environment into three parts: Sensing, Computing, and Demonstrating.

A. Sensing

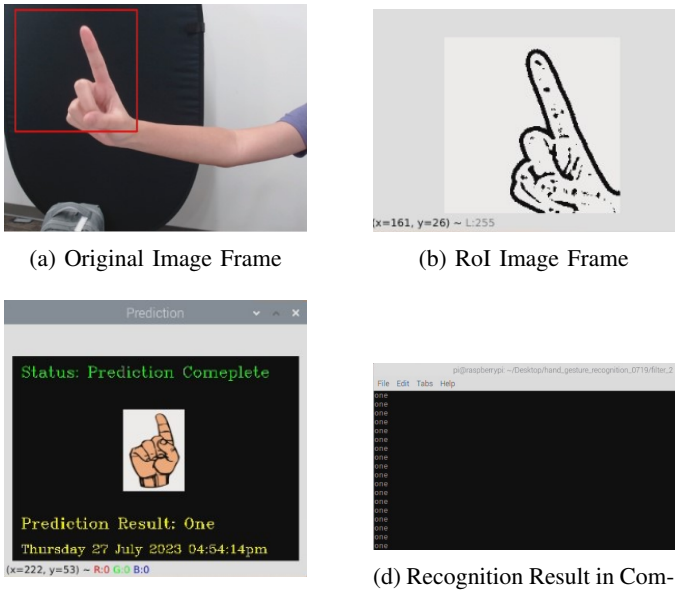
A reliable gesture-capturing device is essential to enable an efficient and low-cost hand gesture recognition system. Various sensors are suitable for this purpose, such as Infra-Red (IR) sensors, web cameras, LiDAR sensors, etc. Notably, according to a study in 2021 [19], 53 % of research in this field has prioritized 2D images with RGB channels for detecting human gestures. Since the clarity and reliability of the input images are critical for detailed and precise gesture recognition, in this testbed, we adopt the Logitech HD 920 webcam, known for its high-definition video capture capabilities, to capture and interpret hand gestures. The HD 920 not only offers 1080p resolution but also comes equipped with advanced autofocus and light correction features. These features ensure that gestures are captured with extreme clarity, even under

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.



(a) Front View (b) Side View

Fig. 1: An Illustration of Our Testbed Environment



(a) Original Image Frame (b) RoI Image Frame

(c) Prediction Frame (d) Recognition Result in Command Line

Fig. 2: A Python-based GUI for the Demonstration of Our Hand Gesture Recognition System

varying lighting conditions. Fig. 1 gives an illustration of our testbed. Particularly, the camera captures images and transmits them to the Raspberry Pi, as shown in Fig. 1a. We also have a keyboard and mouse to access the Raspberry Pi. A monitor is used to showcase the web camera interface and monitor the recognition process, as shown in Fig. 1b.

B. Computing

After capturing hand gesture images, we aim to train a deep learning model to mine the patterns of hand gestures based on acquired data. Considering the limited computation resources for Raspberry Pi, we consider transferring the training process to a local edge server with high computational capabilities, which not only expedites the training process but also enhances the system's usability. To this end, we offload the collected images to the server and download the well-trained model from the server.

Model inference is less resource-intensive than model training. Thus, the recognition task could be executed on a device with constrained computational capacity. The Raspberry Pi, an embedded Linux-based mini-computer, is a suitable choice for our study. Being a cost-effective, edge-based device with limited computational power, the Raspberry Pi can fetch the trained model from the edge server deployed nearby and execute gesture recognition effectively.

C. Demonstrating

During demonstrations, a proper gesture-capturing interface is instrumental in guiding individuals to position their hands correctly, thereby maximizing the accuracy of the recognition system. We utilize a 32-inch monitor as the display interface, where a Python-based graphic user interface (GUI) for the hand gesture recognition system operates, allowing for real-time and simultaneous recognition tasks. Fig. 2 illustrates our designed GUI for hand gesture prediction. As it indicates, there are three frames: one frame to show the original images, one frame to show the filtered RoI (region of interest) images, and one frame to show the prediction results. All frames are flushed at 60 Hz. Additionally, a command line displays the real-time prediction results based on our well-trained model by chronologically feeding captured and processed images.

IV. Our Approach

In this section, we first give an overview of our designed hand gesture recognition system and then introduce our designed image filters, our proposed model architectures, post-training model quantization implementations, and motion detection implementations, respectively, in detail.

A. System Overview

Fig. 3 illustrates our workflow for an edge AI-based hand gesture recognition system. Initially, we capture gesture images using the webcam and store images on the Raspberry Pi. We consider hand gestures with numbers from zero through nine in ASL as our targets. To increase the robustness of recognition under dynamically realistic environments, we consider three different collection scenarios: (i) gesture images against a black background (with no background noise), (ii) gesture images with a regular background (with background noise), and (iii) gesture images with a low-light background (with background noise) as shown in Fig. 4. We collect approximately 64,000 images, with 6,400 images per class in three scenarios in total. After that, we convert RGB images to grayscale images and feed them into two filters: (i) an adaptive threshold filter, and (ii) a guided filter with Canny edge detection, as shown in Fig. 5. After organizing the pre-processed images, we upload the processed images to the server.

We pick a list of CNN models at the edge server as our candidate models. Particularly, in our system, we have put

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.

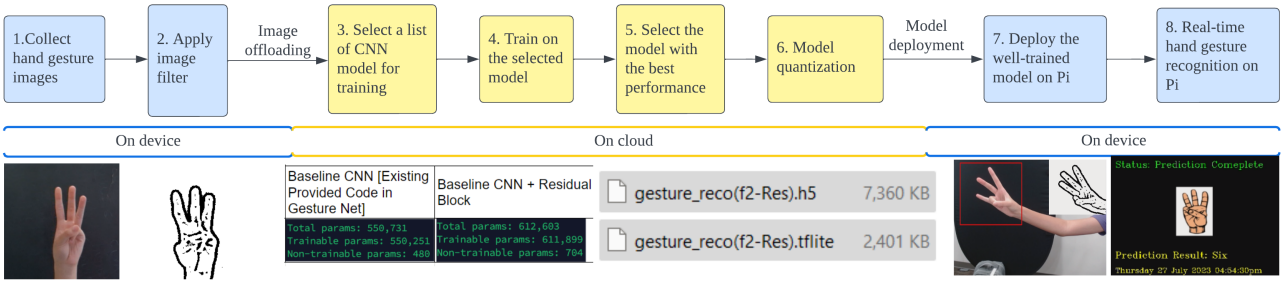


Fig. 3: Edge-based Hand Gesture Recognition System Workflow

Fig. 4: An Illustration of Dynamic Environments: a Low-light Background, a Regular Background, and a Black Background

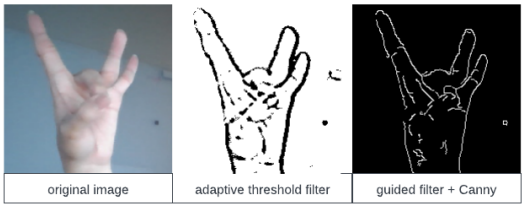


Fig. 5: An Illustration of Image Filters: Original Image, Filtered Image with Adaptive Threshold, and Filtered Image with Guided Filter and Canny Edge Detection

the following five models into consideration: (i) a baseline GestureNet CNN, (ii) a baseline CNN enhanced with a residual block excluding a fully connected layer, (iii) A baseline CNN enhanced with a dense block excluding a fully connected layer, (iv) a baseline CNN with both a residual block and a fully connected layer, and (v) a MobileNet V3 model. Each model will be trained in parallel with the same training configurations. Then, we test the model performance on different CNN models by feeding additional filtered images. Through a thorough analysis of training and validation accuracy among different combinations, we identify the combination with the highest validation accuracy as our final model sending back to the Raspberry Pi. Before deploying it onto the Raspberry Pi with limited computational and memory resources, we need another step to quantize this model using TensorFlow Lite. Consequently, the Raspberry Pi is equipped with fast hand gesture inference.

Finally, the Raspberry Pi downloads the trained and quantized model from the edge server. Based on our provided GUI, as illustrated in Fig. 2, it allows an interactive recognition by varying hand gestures and displays the accurate prediction results on GUI.

B. Image Filters

We now introduce two image filtering methods to preprocess the images: (i) the adaptive threshold filter and (ii) the guided filter with Canny edge detection. Notably, before image filtering, we need to convert 3-channel RGB images to 1-channel grayscale images.

1) Adaptive Threshold Filter: For each original grayscale image, we first apply a Gaussian filter with a 7x7 kernel to the image. The Gaussian filter will blur and smooth images with minimal distortion by convolving each point in the image array using a Gaussian kernel. The formula for Gaussian filtering is

$$O(i,j) = \sum_{x=-\infty}^{\infty} \sum_{x=-\infty}^{\infty} I(i+x, j+y) \cdot K(x,y),$$

where $O(i,j)$ represents the pixel value at position i,j after filtering, $I(i+x, j+y)$ represents the pixel value at position $i+x, j+y$ before filtering, and $K(x,y)$ represents the kernel value at position (x,y). Thus, using a 7x7 kernel to filter the image, we are significantly smoothing the image.

After that, we implement an adaptive threshold filter by utilizing a block size of 9 and a constant of 2. Particularly, the block size is an odd integer, indicating the size of the neighborhood used to determine the threshold value for each pixel. The adaptive threshold filter is used as opposed to the simple threshold filter because the simple threshold filter employs a fixed threshold value for the entire image, whereas the adaptive threshold filter dynamically adjusts the threshold on the basis of intensity variations throughout the image. For the threshold strategy, we use the adaptive Gaussian threshold, which uses a Gaussian distribution to compute the weighted average of surrounding pixel intensities for each pixel to determine the threshold.

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.

2) Guided Filter with Canny Edge Detection: For each original grayscale image, we first apply a guided filter that preserves sharp edges while smoothing the image. The guided filter eliminates gradient reversal, a common drawback in filters like the bilateral filter, and the guided filter's fixed complexity per pixel ensures efficiency. For the hyper-parameters, we use a radius of 40 and a regularization parameter of 20 to add strong blurring and smoothing.

After that, we apply the Canny edge detection technique, which is commonly used for detecting edges in an image with precision and clarity. Canny edge detection produces clean results compared to alternative edge detection algorithms like the Sobel filter. We set the lower bound and upper bound of the edge detection threshold as `int(variance/255*17.25)-15` and `int(variance/255*17.25)+15`, respectively, where `variance` is the variance of pixel values of the image after applying the guided filter. This adjustment accounts for changes in lighting throughout the image. In so doing, we identify and highlight the edges in an image that represent rapid changes in intensity or color.

C. Learning Models

In the following, we introduce our developed learning models. In particular, we first design a vanilla CNN-based model for hand gesture recognition called Gesture Net. Then, inspired by deep residual learning, we propose an improved model, Residual Gesture Net, with a higher accuracy in hand gesture recognition.

1) Gesture Net: We first propose a CNN-based model, called GestureNet, serving as our baseline recognition model, which consists of an input layer, three convolutional blocks, a fully connected layer, and an output layer.

In particular, for the input layer, the model anticipates images of a specific shape defined by its width, height, and depth. In our application, this input shape is $64 \times 64 \times 1$ by feeding grayscale images. The first convolutional block includes a 2D convolutional layer that utilizes 16 kernels, each with a kernel size of 7×7, followed by a ReLU activation each. The kernels assist the CNN in gleaning vital details from the input, while the activation function enables the network to discern intricate patterns within the data. Then, we apply batch normalization to stabilize and speed up the training and a max pooling operation with a pool size of 2x2 with a dropout rate of 0.25. Here, the max pooling operation assists in isolating the most identical features of an object, emphasizing attributes like edges and corners. Dropout operation is commonly used after the pooling layers to reduce overfitting.

For the second and third convolutional blocks, they follow the same structure as the first convolutional block. The only differences lie in the number of kernels and kernel size. For example, in the second convolutional block, the 2D convolutional layer utilizes 32 kernels, each with kernel size of 3×3. In the third convolutional block, the 2D convolutional layer utilizes 64 kernels, each with kernel size of 3×3. After that, there is a flatten layer to reshape the features into a vector, followed by a fully connected layer with 128 neurons and ReLU activation, batch normalization, and dropout with a 0.5 dropout rate. The output layer contains a dense layer with neurons equal to the number of classes and a softmax activation for class probabilities.

2) Residual Gesture Net: Due to the complexity of human gestures, relying solely on baseline CNN models may not achieve the desired accuracy. While augmenting the baseline model with additional layers may increase training complexity, it may result in prolonged operation times on compute-constrained devices such as the Raspberry Pi. To solve this problem, integrating residual blocks into CNN models can improve accuracy and training efficiency and reduce model computation. Thus, we propose an improved model, called Residual Gesture Net, with the involvement of residual blocks. Notably, in our proposed Residual Gesture Net, we utilize the ReLU function as our activation to connect two consecutive layers inside the block, since it is a non-explosive activation function. Also, we use a small learning rate to prevent gradient explosion in the model's training.

D. Post-training Quantization

To better fit the low-cost environments on edge devices, we employ quantization techniques to further reduce the model's storage size implemented with TensorFlow Lite. As an edge-device-oriented learning library, TensorFlow Lite facilitates the deployment of models on mobile devices, microcontrollers, and other edge devices. At its core, Tensorflow Lite capitalizes on the quantization mechanism to reduce the model's size, thereby conserving storage space, reducing the download size, and optimizing memory usage. Through quantization, various computational tasks are streamlined by minimizing the computations executed during inference, while possibly sacrificing a modicum of accuracy.

There are several methodologies to implement quantization with TensorFlow Lite. In our approach, we harness post-training quantization, ensuring minimal accuracy loss without necessitating alterations to the downloaded model. This tactic can potentially reduce the model size by up to 75 %, and reduce memory usage by up to 63 %, respectively, on average, as indicated in Table I.

E. Energy Consumption Reduction

Energy consumption is critical in low-cost IoT devices, which could operate using batteries. To enable energy saving in our designed system, we integrate a distance laser sensor into our testbed, which can automatically turn off our recognition system once it detects no people nearby, leading to lower energy consumption costs. The system uses the RS485 laser ranging sensor for distance detection with an RS485-to-USB converter to connect with the Raspberry Pi.

To evaluate the performance of our energy-saving solution, we deploy the Kasa KP115 energy monitoring smart plug to record the power consumption of our hand gesture recognition system. Initially, the system enters the user detection phase. During this phase, a distance laser sensor gauges the proximity

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Memory and Storage Variations When Applying Model Quantization Techniques

Model	Model Parameters	Model Storage Size (MB)		Memory Usage (MB)	
		Before Quantization	After Quantization	Before Quantization	After Quantization
Baseline CNN	550,731	6.6	2.2	43.8	34.4
CNN + Residual Block	612,603	7.3	2.4	46.2	34.6
MobileNet v3	1,662,299	20.4	6.5	55.5	35.2

TABLE II: Power Consumption of Running Raspberry Pi

System Stage	Idle	Sensing Detection	Gesture Recognition
Power Consumption (W)	2.3	3.0	7.2

of a user to the system. The system shifts to the recognition phase if the user is less than 1 meter away. After the user finishes the recognition and is more than 1 meter away from the sensor, the system reverts to the detection phase. The power consumption of the different system stages is shown in Table II. From the table, by involving sensing detection, we can infer that our system power consumption cost can be reduced by 60 % compared to when the recognition function is not used.

V. Performance Evaluation

This section presents the experimental results to validate the efficacy of our designed system. In the following, we first introduce the evaluation methodology and then discuss the experimental results in detail.

A. Methodology

1) Data Collections: Our experiment utilized a self-collected dataset for the ASL digits task, i.e., recognizing hand gestures for numbers from 0 to 9. This dataset comprises 6,400 samples for each category, totaling 64,000 samples across all categories. It is worth noting that background and illumination play crucial roles in recognition accuracy. To investigate the effects of varying backgrounds and illumination levels on the accuracy, we have collected images and verified the recognition performance in three distinct environments: a regular background, a pure black background, and a background under low illumination.

2) Data Processing and Hyper-parameters Configurations: We allocate 70 % of the data for training and the remaining 30 % for testing. To enhance the model's robustness, the training set was augmented with a 20 % rotation. We set the learning rate to 1×10^{-3} and the batch size to 8. We choose the Adam optimizer for model updating.

3) Model Configurations: For performance evaluation, in addition to the Gesture Net and Residual Gesture Net that we have proposed, we consider three additional models in this experiment to evaluate performance: the baseline as the Residual Gesture Net without one fully connected layer (i.e., removing the dense layer with 128 neurons), the Gesture Net with dense blocks [20], and MobileNet V3 [21]. Additionally, we conduct extensive experiments with two fine-tuned filters, e.g., the adaptive threshold filter and the Canny-based filter, to assess their impact on the performance across the models.

4) Environment Configurations: We use TensorFlow 2.10 as our primary platform. Our CNN model's training is conducted on an edge server with a Ryzen 7 5825U CPU, 40 GB DDR4 RAM, and an AMD Radeon RX Vega8 GPU. Additionally, we deploy the well-trained model on a Raspberry Pi 4B for gesture recognition, which is equipped with 4 GB DDR4 RAM and runs with Raspberry Pi OS.

5) Evaluation Metrics: To evaluate the model performance under different conditions, we keep track of the training, validation, and test accuracy to estimate the recognition performance. In particular, validation accuracy is used to tune hyperparameters in the model's training and to prevent overfitting. The test accuracy is the key factor in validating the model's superiority in recognition.

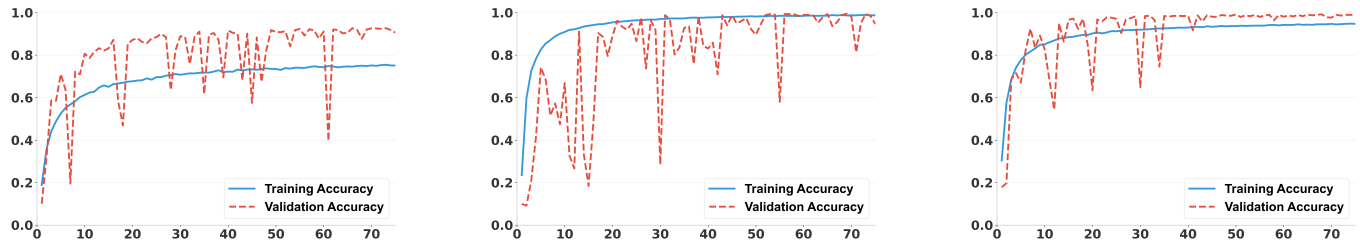
B. Results

We first check on model convergence. Figs. 6 and 7 indicate the convergence curve of each model under the adaptive threshold filter and the guided filter with Canny edge detection, respectively. As we can find, as the training epoch increases, the training accuracy generally increases. More importantly, the validation accuracy follows the same tendency, even though different levels of fluctuations exist for different models. Particularly, the fluctuations for MobileNet V3 are more obvious, since its model size and complexity are relatively higher than other models, making it not easy to be trained to a converged status. The training of the CNN+ResBlock model with filter 3 (guided filter and Canny edge detection) is the most smooth, and the training of the same model with filter 2 (adaptive threshold) is also smooth but contains some fluctuations. Thus, faster convergence occurs during the training of the CNN+ResBlock model.

Then, we check on the training performance in terms of training and validation accuracy. Table III displays each model's training and validation accuracy in different filters. Particularly, by averaging different filters' results, the validation accuracies are 0.81, 0.94, 0.89, 0.97, and 0.97 for five evaluated models: baseline CNN, baseline CNN plus residual block without one FC layer, baseline CNN plus dense block without one FC layer, baseline CNN plus residual block with one FC layer, MobileNet V3, respectively.

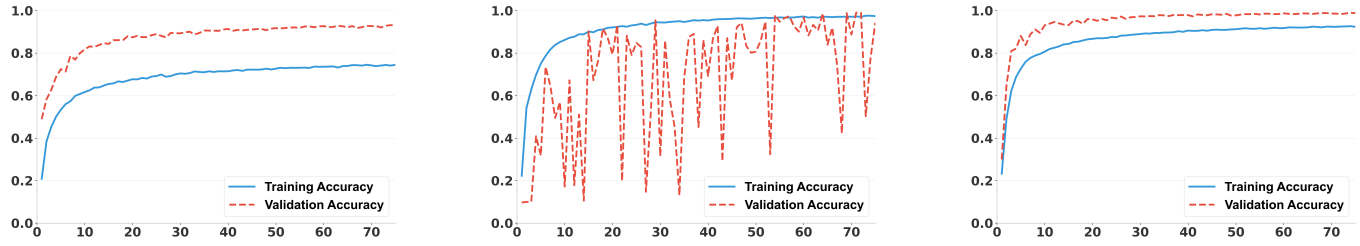
It is evident from the given data that the last two models exhibit superior accuracy compared to the other models. Also, Table IV indicates the number of model parameters for each model. We can observe that the model size of our proposed Residual Gesture Net is only 36 % of that of MobileNet V3. It is also approximately 10 % of the size of AlexNet and 0.44 %

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.



(a) Gesture Net (b) MobileNet V3 (c) Residual Gesture Net

Fig. 6: Convergence Curves with the Adaptive Threshold Filter



(a) Gesture Net (b) MobileNet V3 (c) Residual Gesture Net

Fig. 7: Convergence Curves with the Guided Filter and Canny Edge Detection

TABLE III: Training and Validation Accuracy of Each Model

Training Accuracy vs. Validation Accuracy		Training Model				
		Baseline CNN	Baseline CNN + Residual Block w/o FC	Baseline CNN + Dense Block w/o FC	Baseline CNN + Residual Block w/ FC	MobileNet V3
Filter	Adaptive Threshold Filter	0.75 |0.90	0.94 |0.87	0.88 |0.92	0.93 |0.99	0.95 |0.95
	Guided Filter + Canny Edge Detection	0.74 |0.93	0.92 |0.98	0.79 |0.84	0.89 |0.98	0.97 |0.94
	No Filter	0.64 |0.88	0.94 |0.99	0.94 |0.99	0.95 |1.00	0.99 |0.99

TABLE IV: Number of Model Parameters for Each Model

Training Model	Baseline CNN	Baseline CNN + Residual Block w/o FC	Baseline CNN + Dense Block w/o FC	Baseline CNN + Dense Block w/ FC	MobileNet V3
# of Model Parameters	550,731	74,699	81,403	612,603	1,662,299

TABLE V: Real-time Recognition Results (Filter 1: Adaptive Threshold Filter; Filter 2: Guided Filter with Canny Edge Detection)

Filter	Model Name	Model Size	Regular Background	Black Background	Low Illumination Background
Filter 1	CNN + ResBlock	610K	9/10	10/10	10/10
	MobileNet	1.6M	8/10	8/10	7.5/10
	CNN + Dense Block	70K	8/10	9/10	non-stable (loss texture information)
Filter 2	CNN + ResBlock	610K	10/10	10/10	9/10
	MobileNet	1.6M	8/10	8/10	6/10
	CNN + Dense Block	70K	7.5/10	9/10	non-stable (loss texture information)

of the size of VGG16. This indicates our model is not only effective in recognition but also light-weighted, ideally for edge devices.

Finally, we check on the performance of real-time test accuracy. Table V shows the real-time recognition results with different image filters. We accumulate the correct rate by testing all ASL digit gestures from 0 to 9. The average results are obtained by running the experiment four times. We observe that the combination of CNN and residual blocks attains the highest accuracy among compared models. Particularly, under Filter 1, it achieves a score of 10/10 under dynamic environments. Under Filter 2, it also performs well enough, with only a slight drop in extreme environments, i.e., low-light environments. In contrast, there is a significant dilemma between the training performance and real-time recognition performance for MobileNet V3. We find its real-time recognition results drop significantly compared with its good performance in training. Particularly, we find that its performance

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.

under low-light environments is highly non-stable and greatly impacted by the illustration condition surrounding the camera. Since MobileNet V3 is of high complexity, a slight noise captured by the input image can incur unexpected changes in the model's prediction. Instead, our model is well-designed for edge devices with lower model complexity, which is more tolerant to illustration noise. To some extent, it indicates the robustness of our proposed model in real-time recognition under dynamic environments.

VI. Discussion

Border Impacts: In this paper, we systematically investigate the feasibility of deep learning model deployment on Raspberry Pi and verify the efficacy of our designed hand gesture recognition system to meet the requirements of high prediction accuracy, real-time inference, and low power consumption. In fact, our system could be easily extended to other common low-cost devices, like Nvidia Jetson, Particle, Google Coral, Orange Pi, Odroid, Intel NUC, etc. Our system design methodology is generic and can be applied to other IoT applications. Our research is designed to shed light on future research directions in edge intelligence.

Future Research: The next generation of edge AI is expected to enable the following four functions in an integrated device: on-device data collection, on-device model training, real-time model inference, and on-device model updating. In our proposed system, we have implemented the functions of on-device data collection and real-time model inference with promising results. Next, we will try to handle the bottleneck, e.g., on-device model training and updating, which is a complex task involving optimization designs among hardware, model architecture, data storage, and scheduling. In addition, in our ongoing research, empirical and theoretical analysis could be conducted to investigate the trade-off between high learning performance and low computational and energy costs in different edge computing infrastructures.

VII. Final Remarks

In this study, we developed a hand gesture recognition system on Raspberry Pi using deep learning-based techniques to achieve high prediction accuracy, real-time inference, and low power consumption. Particularly, we first designed a simple but effective CNN model with residual blocks to handle ASL digits tasks, enabling fast-bust-accurate real-time recognition. Then, to fit the low-cost environment for edge devices, we involved model quantization techniques to shrink the model size, thus requiring reduced memory for edge devices. In addition, we integrated a motion sensor into our system to automatically turn off our system once it detects no people nearby, which lowers energy consumption. By evaluating the performance of real-time hand gesture recognition, our designed system maintains a high prediction accuracy rate, e.g., over 96 %. Moreover, our designed solution has the potential to be extended to other common low-cost edge devices and applied to other IoT applications.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[2] "Gesture recognition market size, share covid-19 impact analysis," 2023. [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/gesture-recognition-market-100235

[3] S. Hussain, R. Saxena, X. Han, J. A. Khan, and H. Shin, "Hand gesture recognition using deep learning," in *2017 International SoC Design Conference (ISOCC)*, 2017, pp. 48–49.

[4] F. Zhan, "Hand gesture recognition with convolution neural networks," in *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*. IEEE, 2019, pp. 295–298.

[5] J. Yu, M. Qin, and S. Zhou, "Dynamic gesture recognition based on 2d convolutional neural network and feature fusion," *Scientific Reports*, vol. 12, no. 1, p. 4345, 2022.

[6] Y. Xing, G. Di Caterina, and J. Soraghan, "A new spiking convolutional recurrent neural network (SCRNN) with applications to event-based hand gesture recognition," *Frontiers in neuroscience*, vol. 14, p. 590164, 2020.

[7] K. Lai and S. N. Yanushkevich, "CNN+ RNN depth and skeleton based dynamic hand gesture recognition," in *2018 24th international conference on pattern recognition (ICPR)*. IEEE, 2018, pp. 3451–3456.

[8] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti, "American sign language recognition with the kinect," in *Proceedings of the 13th international conference on multimodal interfaces*, 2011, pp. 279–286.

[9] B. Garcia and S. A. Viesca, "Real-time American sign language recognition with convolutional neural networks," *Convolutional Neural Networks for Visual Recognition*, vol. 2, no. 225-232, p. 8, 2016.

[10] K. Bantupalli and Y. Xie, "American sign language recognition using deep learning and computer vision," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4896–4899.

[11] J. Shin, A. Matsuoka, M. A. M. Hasan, and A. Y. Srizon, "American sign language alphabet recognition by extracting feature from hand pose estimation," *Sensors*, vol. 21, no. 17, p. 5856, 2021.

[12] M. Mohammadi, P. Chandarana, J. Seekings, S. Hendrix, and R. Zand, "Static hand gesture recognition for american sign language using neuromorphic hardware," *Neuromorphic Computing and Engineering*, vol. 2, no. 4, p. 044005, 2022.

[13] T. G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A hand gesture interface device," *ACM Sigchi Bulletin*, vol. 18, no. 4, pp. 189–192, 1986.

[14] M. Z. Alksasbeh, A. H. Al-Omari, B. A. Alqaralleh, T. Abukhalil, A. Abukarki, I. A. Alshalabi, and A. Alkaseasbeh, "Smart hand gestures recognition using K-NN based algorithm for video annotation purposes," *Indones. J. Electr. Eng. Comput. Sci*, vol. 21, no. 1, pp. 242–252, 2021.

[15] P. N. Huu and T. Phung Ngoc, "Hand gesture recognition algorithm using SVM and HOG model for control of robotic system," *Journal of Robotics*, vol. 2021, pp. 1–13, 2021.

[16] G. Latif, N. Mohammad, R. AlKhalaf, R. AlKhalaf, J. Alghazo, and M. Khan, "An automatic Arabic sign language recognition system based on deep CNN: an assistive system for the deaf and hard of hearing," *International Journal of Computing and Digital Systems*, vol. 9, no. 4, pp. 715–724, 2020.

[17] J. Hashemi, T. V. Spina, M. Tepper, A. Esler, V. Morellas, N. Papanikolopoulos, and G. Sapiro, "Computer vision tools for the non-invasive assessment of autism-related behavioral markers," 2012.

[18] B. Shi, A. M. Del Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, and K. Livescu, "American sign language fingerspelling recognition in the wild," in *2018 IEEE Spoken Language Technology Workshop (SLT)*, 2018, pp. 145–152.

[19] N. Mohamed, M. B. Mustafa, and N. Jomhari, "A review of the hand gesture recognition system: Current progress and future directions," *IEEE Access*, vol. 9, pp. 157 422–157 436, 2021.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[21] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1905.02244

Authorized licensed use limited to: UNIVERSITY OF STRATHCLYDE. Downloaded on April 20,2024 at 10:02:42 UTC from IEEE Xplore. Restrictions apply.