Indonesian Journal of Electrical Engineering and Computer Science
Vol. 34, No. 3, June 2024, pp. 1603∼1617
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v34.i3.pp1603-1617

TQU-HG dataset and comparative study for hand gesture recognition of RGB-based images using deep learning

Van-Dinh Do[1], Van-Hung Le[2], Huu-Son Do[2], Van-Nam Phan[2], Trung-Hieu Te[2]
[1]Department of Electrical, Sao DO University, Chi Linh, Vietnam
[2]Department of Information Technology, Tan Trao University, Tuyen Quang, Vietnam

Article Info

Article history:

Received Jan 26, 2024
Revised Feb 18, 2024
Accepted Mar 10, 2024

Keywords:

Comparative study
Convolutional neural networks
Deep learning
Hand gesture dataset
Hand gesture recognition

ABSTRACT

Hand gesture recognition has great applications in human-computer interaction (HCI), human-robot interaction (HRI), and supporting the deaf and mute. To build a hand gesture recognition model using deep learning (DL) with high results then needs to be trained on many data and in many different conditions and contexts. In this paper, we publish the TQU-HG dataset of large RGB images with low resolution (640×480) pixels, low light conditions, and fast speed (16 fps). TQU-HG dataset includes 60,000 images collected from 20 people (10 male, 10 female) with 15 gestures of both left and right hands. A comparative study with two branches: i) based on Mediapipe_TML and ii) Based on convolutional neural networks (CNNs) (you only look once (YOLO); YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, single shot multiBox detector (SSD)_VGG16, residual network (ResNet)18, ResNext50, ResNet152, ResNext50, MobileNet_V3_small, and MobileNet_V3_large), the architecture and operation of CNNs models are also introduced in detail. We especially fine-tune the model and evaluate it on TQU-HG and HaGRID datasets. The quantitative results of the training and testing are presented (F1-score of YOLOv8, YOLO-Nas, MobileNet_V3_small, ResNet50 is 98.99%, 98.98%, 99.27%, 99.36%, respectively on the TQU-HG dataset and is 99.21%, 99.37%, 99.36%, 86.4%, 98.3%, respectively on the HaGRID dataset). The computation time of YOLOv8 is 6.19 fps on the CPU and 18.28 fps on the GPU.

Corresponding Author:
Van-Hung Le
Department of Information Technology, Tan Trao University
Tuyen Quang, Vietnam
Email: van-hung.le@mica.edu.vn

1. INTRODUCTION

Hand gesture recognition has great applications in human-computer interaction (HCI) [1], home device control [2], human-robot interaction (HRI) [3] building sign language understanding systems for the hearing impaired [4]. Previously, to solve this problem, traditional machine learning (TML) models are often used (support vector machine (SVM) [5], AdaBoost [6], and random forest (RF) [7]). Hand gesture recognition results based on TML still have large errors and depend heavily on the learning dataset. Currently, with the advent of deep learning (DL), many very convincing results have been achieved in solving the problem of hand gesture recognition, especially when using convolutional neural networks (CNNs). This is not a new research problem in computer vision and automation. However, it still contains many challenges such as input image quality, lighting, hand occlusion, and speed of hand gestures [8]. Despite impressive results on hand gesture recognition, CNNs require a very large amount of training data for model building. To adapt the data requirements for building a hand gesture recognition model, Kapitanov *et al.* [9] have published the HaGRID dataset with a huge number of frames for model training (554,800 full-HD RGB images collected from 37,583 subjects). For a practical application that uses the results of hand gesture recognition, it is necessary to require a model that can operate in low light conditions, an inexpensive camera (obtained images with low resolution), and the moving speed of quick hand movements.

Therefore, the problem of hand gesture recognition still needs to be researched further. The hand gesture recognition/detection and classification model was evaluated and compared with some models of CNNs [9]: ResNet18 [10], ResNet152 [10], ResNext50 [11], ResNeXt101 [11], MobileNet_V3_small [12], MobileNet_V3_large [12], MobileNet_V3_large + (single shot multiBox detector) SSDLite [12], and YoloV7 tiny [13]. Currently, there are many new versions of you only look once (YOLO) such as YOLOv8 [14], and YOLO-Nas [15] that have been developed and have better results than YOLOv7 [16] or Mediapipe [17] for hand gesture recognition/detection and classification, especially these methods can be performed on the CPU. In this paper, we perform research and comparison on some of these methods. From there, decided to a DL model for building a hand gesture recognition model for building a practical control system. For commercial products using hand gesture recognition software, the price of the product is very important, as the cost of image sensors plays a huge role in reducing product costs. To reduce product costs, the software needs to be able to operate on images obtained from expensive image sensors/cameras and CPU. At the same time, the product's processing time also needs to be fast according to practical requirements. In this paper, we publish a large dataset to build a hand gesture recognition model from image data collected from a cheap camera. This dataset includes 60,000 RGB images (low-resolution images are 640×480 pixels) collected from 20 people (10 male, 10 female) with 15 different types of gestures, called the "TQU-HG" dataset.

CNNs to solve the object classification/recognition problem of computer vision are divided into two types [18]: one-stage CNNs YOLO series/family [19], SSD series/family [20]) and two-stage CNNs. In this paper, we propose a taxonomy (as presented in Figure 1) and perform a comparative study to fine-tune the hand gesture recognition model on the HaGRID and TQU-HG datasets. The comparative study we propose includes two branches, i) the first branch is the combination of Mediapipe [17] for hand detection and hand pose estimation on RGB images and then uses a TML model (SVM and RF) for training the hand gesture recognition model, ii) the second branch is to use the CNNs of YOLO family with versions (v) (including YOLOv5 [21], YOLOv6 [22], YOLOv7 [13], YOLOv8 [14], YOLO-Nas [15]), SSD family (including SSD_VGG16) and other CNNs networks like ResNet18, ResNet50, ResNet152, ResNext50, MobileNet_V3_small, MobileNet_V3_large to train the hand gesture recognition model based on marked hand gesture label data. With the advantage of computational time, one-stage CNNs and regular CNNs will be of interest to us in evaluating and analyzing the results. At the same time, the results of precision, recall, and F1-score are also presented.

The main contributions of our paper are as follows: i) we have proposed a large TQU-HG dataset to train the model and evaluate the hand gesture recognition model with about 60,000 RGB images collected from a cheap camera. TQU-HG dataset is captured from 20 people with 15 common gestures in the daily life of sign language. ii) We have systematized the architecture of some one-stage CNNs of the YOLO (YOLOv5, YOLOv6, YOLOv7, YOLOv8, and YOLO-Nas) and SSD (SSD_VGG16) families and other CNNs like ResNet18, ResNet50, ResNet152, ResNext50, MobileNet_V3_small, and MobileNet_V3_large for object classification. iii) We propose a taxonomy of comparative research on hand gesture recognition that includes two branches: the first is a combination of Mediapipe hands and TML, and the second is CNNs. iv) We conduct a study comparing the results of hand gesture recognition based on TML and CNNs on the TQU-HG and HaGRID datasets. The structure of the paper is organized as follows. A taxonomy of comparative research for evaluating hand gesture recognition is presented in section 2. The background on machine learning and CNNs methods for hand gesture recognition is presented in section 2.1. and section 2.2. The dataset and experimental results, discussion, and challenges will be presented in section 3. We finally conclude and give some ideas for future works will be presented in section 4.

2. A TAXONOMY OF COMPARATIVE RESEARCH

As shown in Figure 1, we propose a taxonomy for the study of comparing hand gesture recognition with RGB image input. This comparative study is divided into two branches. The first branch (Mediapipe_TML) is to perform hand gesture recognition based on the estimated hand skeleton using Mediapipe hands [17] for hand detection and hand skeleton estimation. Then use TML methods such as SVM and RF for gesture recognition. The second branch is to use one-stage CNNs (YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, SSD_VGG16) and other CNNs networks like MobileNet_V3_small, MobileNet_V3_large, ResNet18, ResNet50, ResNet152, ResNext50 to classify/recognize hand gestures on the TQU-HG and HaGRID datasets.

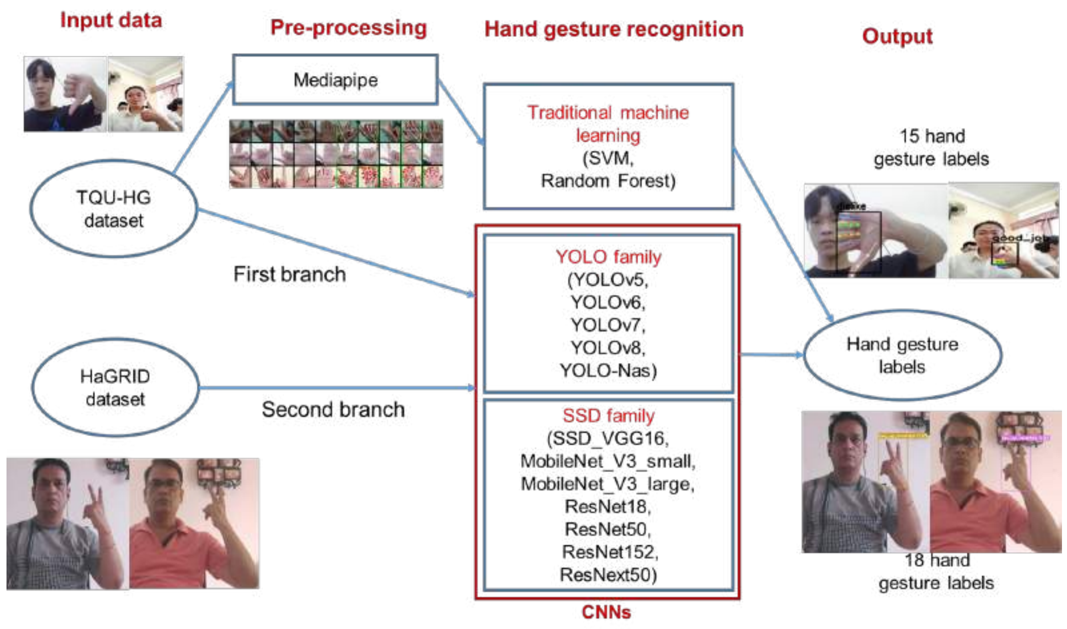
Input data
Pre-processing
Hand gesture recognition
Output
Mediapipe
Traditional machine learning (SVM, Random Forest)
15 hand gesture labels
TQU-HG dataset
First branch
YOLO family (YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas)
Hand gesture labels
HaGRID dataset
Second branch
SSD family (SSD_VGG16, MobileNet_V3_small, MobileNet_V3_large, ResNet18, ResNet50, ResNet152, ResNext50)
18 hand gesture labels
CNNs

Figure 1. A taxonomy of comparative studies

2.1. Mediapipe_TML for hand gesture recognition

MediaPipe is a collection of cross-platform machine learning solutions such as Android/iOS), Desktop/Cloud, Web, and internet of things (IoT) devices, that can interact with lightweights. This is an open-source platform that can be freely adjusted according to applications such as detecting, estimating hand skeletons, detecting hand gestures, estimating human posture, and detecting and recognizing faces. MediaPipe hands [17] is an ML pipeline consisting of multiple models working together like a palm detection model that conducts prediction on the full image and returns the hand bounding box. The hand landmark model performs prediction on the cropped image region determined by the palm detector and returns highly accurate 3D hand keypoints. Previously, to solve the problem of hand gesture recognition, TML models were often used. In this paper, we experiment on SVM [23] and RF [24] models. The features of the 2D hand skeleton estimated from Mediapipe hands are extracted and included in training the hand gesture classification model.

2.2. CNNs for hand gesture recognition

One-stage CNNs such as YOLO or SSD are often applied to object detection problems. However, they can also be applied to the problem of object classification/direct detection of object labels. In this paper, we use YOLO and SSD to fine-tune the object class label detection model, this problem is similar to object classification.

YOLO [25] is a typical and most-used CNN for object detection problems. YOLO has a fairly basic architecture, including a base network and extra layers. The base network is a convolution network responsible for extracting image features. Extra layers are the final layers used to analyze features and detect objects. The base network commonly used is Darknet. The input image is divided into a grid of $S \times S$ cells, also known as cells. Here there is no real image division, but the nature of image division is dividing the output and target into a matrix A of size $S \times S$. If the image, the center of the object is in the $(i, j)^{th}$ cell, then the corresponding output will be in $A[i, j]$. YOLO's implementation process includes 2 steps: the first is a convolution network that extracts image features. The second is extra layers (fully connected layers) to analyze and detect objects.

YOLOv5 [21]: YOLOv5 focuses on speed and ease of use. YOLOv5 model architecture [21] has three important parts: Backbone, Neck, and Head. The backbone aims to extract rich features from the input image. A cross stage partial network (CSPNet) is used as a model backbone. CSPNets are faster than deeper networks. YOLOv5 improves YOLOv4's CSPResBlock [26] into a new module, little more than a convolution layer called a C3 module. In YOLOv5 still use CSPDarkNet53 for predicting the original object candidates according to the boxes. In the activation function, YOLOv4 [26] uses Mish or LeakyReLU for the lightweight version, while in YOLOv5, the activation function used is SiLU. The Neck constructs feature pyramids. This component helps the model to detect and identify the same objects at different sizes and scales. This leads to better performance with unlearned data. Different types of feature pyramids are used by many models, like FPN, BiFPN, and PANet which is used in YOLOv5. The proposed SPP-fast (SPPF) module is similar to SPP in YOLOv5 with twice the speed. Instead of using parallel MaxPooling as in SPP, YOLOv5 SPPF uses sequential MaxPooling. Furthermore, the kernel size in the MaxPooling of SPPF is all 5 instead of (5, 9, 13) like the SPP of YOLOv4. The Head performs the final detection. It applies anchor boxes on generated features and outputs final vectors containing class probabilities, objectiveness scores, and BBs. YOLOv5 has the same model head as YOLOv3 and YOLOv4.

YOLOv7 [27]: YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 fps to 160 fps and has the highest accuracy at $AP = 0.568$ when performed on the V100 GPU, the object detection results are up to 30 fps. This architecture also includes three main parts: The Backbone used the ELAN (YOLOv7-p5, YOLOv7-p6) and E-ELAN (YOLOv7-E6E). The Neck used the SPPCSPC + (CSP-OSA)PANet (YOLOv7-p5, YOLOv7-p6) + RepConv. The Head used the YOLOR + Auxiliary Head (YOLOv7-p6).

YOLOv7 was announced in 2022 and has been several improvements over previous versions. One of the main improvements is the use of anchor boxes. Anchor boxes are a set of predefined boxes with different aspect ratios that are used to detect objects of different shapes. YOLOv7 uses nine anchor boxes, allowing YOLO to detect a wider range of object shapes and sizes than previous versions, the process of predicting an object's BB based on the anchor boxes, thus helping to reduce the number of wrong identifications. Another important improvement of YOLOv7 is using a new loss function called "focal loss". Previous versions of YOLO used the standard cross-entropy loss function, which is known to be less effective at detecting small objects. Therefore, the results of detecting small-sized objects are not good.

YOLOv8 [28]: YOLOv8 is the current latest version of the YOLO family to be published. Like other versions of YOLO, YOLOv8 also includes two main components: Backbone and Head. YOLOv8 was developed by ultralytics, who also created the YOLOv5 model. YOLOv8 includes many changes and improvements in architecture and developer experience compared to YOLOv5. The YOLOv8 architecture features several new enhancements and combinations introduced by ultralytics: first, the Backbone of YOLOv8 is the same as YOLOv5, it uses the CSPDarknet53 feature extractor. It has some changes like $C2f$ replacing $C3$ to combine high-level features with contextual information to improve detection accuracy.

Second, YOLOv8 has an improvement in using an anchor-free model with a detached Head to handle object, classification, and regression tasks independently. This model allows each prediction branch to focus on its task and improves the overall accuracy of the prediction model. The sigmoid function as the activation function for the feature score is used in the output layer of YOLOv8.

Third, two convolutions (#10 and #14 in the YOLOv5 config) were removed. YOLOv8 provides a semantic segmentation model called the YOLOv8-Seg model to achieve state-of-the-art results on various object detection and semantic segmentation benchmarks while maintaining high speed and efficiency. The loss functions for BB loss and binary cross-entropy for classification loss and to improve the predicted results on small objects. Fourth, the bottleneck in YOLOv8 is still the same as in YOLOv5, except that the kernel size of the first convolution has been changed from 1×1 to 3×3. This change represents a change to the ResNet block.

YOLO-Nas: YOLO-Nas [15] is developed by Deci.AI company [28] and it offers state-of-the-art (SOTA) performance with superior speed and accuracy performance compared to other versions of YOLO such as YOLOv5, YOLOv6, YOLOv7, and YOLOv8. The neural architecture search (NAS) is abbreviated from NAS. There are some new points as follows: First, the use of QSP and QCI blocks [29] combines the advantages of re-parameterization and 8-bit quantization. The blocks above allow for minimal loss function of precision during post-training quantization. Second, AutoNAC was used to determine the optimal size and structure of the stages in the backbone and neck parts, including block type, number of blocks, and number of channels in each stage. Third, a quantization method that combines selective quantization of certain parts of the model, reducing information loss and balancing latency and accuracy. Standard quantization affects all model layers, often leading to a significant loss of accuracy. Their hybrid method optimizes quantization to maintain accuracy by only quantizing certain layers while leaving other layers untouched. Their layer selection algorithm considers the impact of each layer on accuracy and latency, as well as the impact of converting between 8-bit and 16-bit quantization on overall latency. Fourth, the pre-training mode includes automatically labeled data, self-distillation, and large datasets. Fifth, YOLO-Nas architecture is available under an open-source license. Its pre-trained weights are available for (non-commercial) research use on SuperGradients, Deci's open-source, PyTorch-based, computer vision training library.

SSD [10]: Like most other object detection architectures, the input to SSD is the BB coordinates of the object (also known as the offsets of the BB) and the label of the object contained in the BB. The special feature that makes the SSD model's speed is that the model uses a single neural network. Its approach is based on object recognition in feature maps (which is a 3D shape output of a deep CNN network after removing the last fully connected layers) with different resolutions $(300 \times 300, 512 \times 512)$. The model will create a grid of squares called grid cells on the feature maps, each cell is called a cell and from the center of each cell determines a set of default boxes for frame prediction capable of surrounding objects.

The training process is also the process of fine-tuning the label and bounding truth box probabilities to match the ground input values of the model (including labels and BB offsets). Thus, the SSD model will be a combination of two steps: i) Extracting feature maps from CNNs (VGG16 [30], VGG19 [31], ResNet [32], InceptionNet [33], and MobileNet). ii) Applying convolutional (conv) filters (or kernel filters) to detect objects on feature maps with different resolutions (revolutions). As described above about SSD_VGG16, VGG16 is the backbone for the feature extraction process and building feature maps for the training process of the SSD network.

VGG16 [30] is a CNN with better accuracy than AlexNet proposed in 2014. The architecture of VGG16 is similar to AlexNet but has the following improvements: i) The VGG16 architecture is deeper, including 13 2-way conv layers (whereas AlexNet only has 5 layers) and 3 fully connected layers. ii) For the first time in VGG16 the concept of conv blocks appeared. These are architectures consisting of a set of similar repeating CNN layers. iii) VGG16 inherits the ReLU activation function in AlexNet. iv) VGG16 is also the first architecture to change the order of blocks when stacking multiple CNN + max pooling layers instead of alternating just one CNN + max-pooling layer. Deeper layers of CNN can extract better features than just 1 layer of CNN. v) VGG16 only uses small 3×3 size filters instead of multiple filter sizes like AlexNet. Small filter sizes will help reduce the number of parameters for the model and provide more computational efficiency. The VGG16 network is deeper than AlexNet and its number of parameters is up to 138 million parameters. This is one of the networks that has a large number of parameters. There is also another version of VGG, called VGG19, with 3 additional layers of depth.

As presented in VGG16, networks are designed with an increasingly deep trend. Adding layers will increase complexity and the ability to represent more complex information and, of course, logically increase accuracy. However, reality is proving the opposite, as the number of layers increases, the phenomenon of higher training error occurs. Therefore, the deeper the network, the greater the amount of computation. To reduce the number of calculations, ResNet [10] developed the idea to reduce the number of calculations by performing residual mapping to copy information from previous shallow layers to deeper layers. Suppose the output of the shallow layer is x. During the forwarding process of the network, it is put through a linear transformation $F(x)$. Suppose the output of this linear transformation is $H(x)$. A residual between the deep layer and shallow layer is $F(x, W_i) = H(x) - x$. Where W_i are the parameters of the CNN model with the F transformation and it is optimized during training. In ResNet's architecture, there are 4 convolution layers in each module, a total of 8 residual blocks with 2 conv layers in each block, plus the first convolution layer and the last fully connected layer. Therefore the model with a total of 18 layers should be named ResNet18, similarly, the number of channels and residual blocks in the module can be varied to create different models. ResNet is different, for example, 50 layers model of ResNet50, 152 layers model of ResNet152.

Residual Next (ResNeXt) [11] is an improved version of ResNet, announced in 2016. The comparative architecture between ResNet and ResNext is shown in Figure 2. Figure 2(a) is a residual block of ResNet [10] and Figure 2(b) is a combination of several ResNet blocks into one ResNext block. Based on ResNext's architecture, when data is injected into the network, channels of that data are divided into smaller parts (cardinality groups). This creates the possibility of learning many different types of features simultaneously from input data. These cardinality groups operate independently of each other. Each group will learn its characteristics from the data. This allows the model to learn a wide variety of information from input data.

After each cardinality group has learned its features, these features will be integrated to create the final output (split-transform-merge strategy). This integration process is typically a weighted total, meaning that each cardinality group contributes a portion in generating the final output of the model.

MobileNet model has been proposed since 2017 [34]. Unlike other CNNs such as LeNet [35], AlexNet [36], VGG16, VGG19, GoogLeNet [37], and ResNet have a very large number of parameters because these networks perform conventional 2-dimensional convolution. 2-dimensional convolution will normally be computed over the entire depth (channel-c). Therefore, the number of model parameters will increase significantly depending on the depth of the previous layer.

MobileNetV3 [12] added squeeze and excitation (SE) to the residual block to create a more precise architecture. The difference between the residual architecture of regular ResNet and SE-ResNet (applying SE). SE is a fairly simple network consisting of only a few layers to enhance information between channels, thereby increasing the representation quality of the CNN model. SE does this by using all the information and then selectively emphasizing each channel with important features and paying less attention to less important channels. It can be seen that at the 3rd layer, there is a SE branch with size (width $\times$ height) equal to (1×1) that synthesizes the global context. This branch in turn goes through the transformations FC $\rightarrow$ ReLU $\rightarrow$ FC $\rightarrow$ hard sigmoid (FC is a fully connected layer). Finally, it is multiplied directly into the input branch to scale the input according to the global context. The remaining architecture is the same as MobileNetV2 [37]. In this paper, we conduct experiments on two models MobileNetV3-large and MobileNetV3-small. These models target high and low resource utilization scenarios, respectively. Models were created through the application of platform-aware Nas and NetAdapt to search the network and incorporate network enhancements.

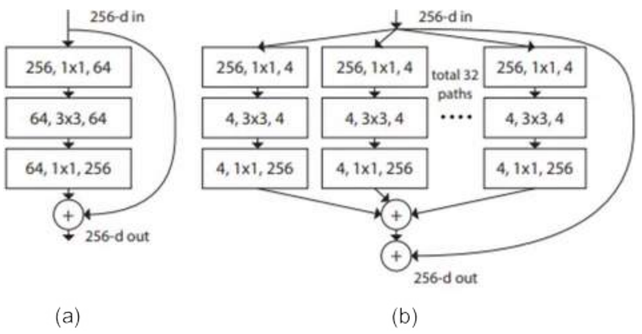

Figure 2. The architecture of the ResNext block is combined from ResNet blocks (a) A residual block of ResNet [10] and (b) a block of ResNext [11]

3. EXPERIMENTAL RESULTS

3.1. Data collection

TQU-HG dataset: in this paper, we collect color image data sets from 20 participants (10 male, 10 female) using a 50 MP RGB camera. Data collection camera integrated on Xiaomi Redmi Note 12 smartphone connected to the Laptop via Droid cam software 50 MP, f/1.8. Although images can be obtained with a resolution of (1080×2400) pixels (full HD+). However, in our experiment, the resolution of the collected images is 640×480 pixels. The data acquisition speed is 16 frames/s, number of color bits when capturing images is $640 \times 480 \times 24 = 7,372,800$ color bits. The images are collected in a low light environment of the classroom at Tan Trao University, Tuyen Quang province, Vietnam, data collection time is in the afternoon. Each image includes only one hand gesture from a single person. Each participant performed 15 different hand gestures ("peace", "hang_loose", "loser", "high_five", "talk_to_the_hand", "you", "good_job", "dis_like", "power_to", "ok", "a_hole", "good_luck", "bang_bang", "rock", "call_me") as shown in Figure 3, with each gesture capturing 200 images, of which 100 were of the left-hand and 100 were of the right-hand. Therefore, the resulting data set has a total of 60,000 images of 15 hand gestures, each gesture has 4,000 images, including 2,000 left-hand images and 2,000 right-hand images.

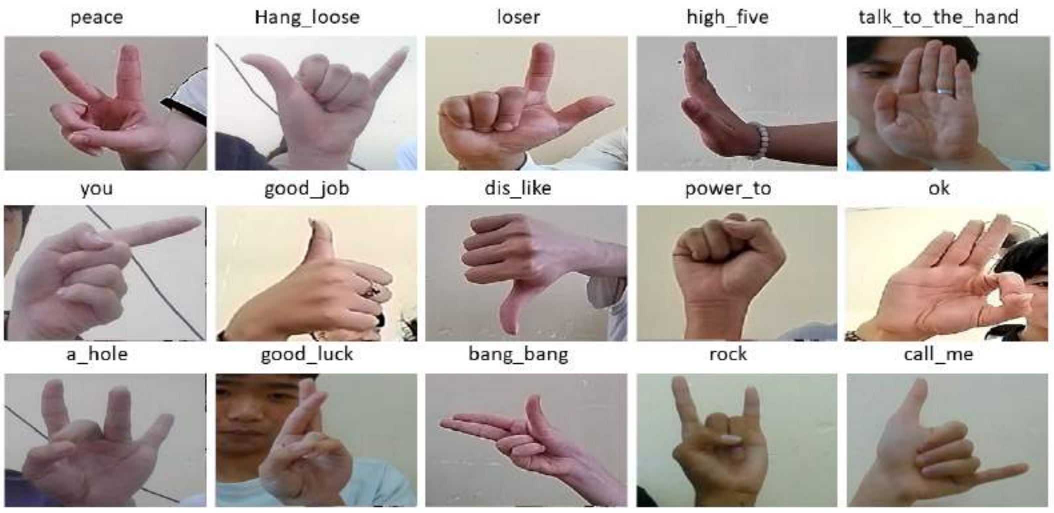

Figure 3. Illustration of hand gestures in the TQU-HG dataset

To prepare the annotation box data and hand gesture labels (15 labels) for training the hand gesture recognition model. We use the software "labelImg" [38]. The process of preparing the annotation data is done entirely manually. To train the hand gesture recognition model, we randomly divide the data set according to each hand gesture, and each person, each type of hand in a ratio of 80-10-10 (80% for training, 10% for validation, 10% for testing). All data of the TQU-HQ dataset is presented in the link (web linknya).

HaGRID datasets [9]: the HaGRID dataset is a large and diverse dataset consisting of images of hand gestures. This dataset was created to improve the performance and evaluate the performance of hand gesture recognition algorithms under real-world conditions, akin to everyday gesture communication. The HaGRID dataset was created to meet the need for a hand gesture dataset with high resolution, high heterogeneity, and a huge number of samples. HaGRID includes approximately half a million full-HD RGB images sized at (1920×1080) pixels with 18 hand gestures and a "no gesture" label. This dataset is collected in many different scenes, from indoors to outdoors, from adults to children, from male to female, with many different lighting conditions and distances. The dataset includes 554,800 full-HD RGB images collected from 37,583 subjects.

3.2. Evaluation metrics

To evaluate hand gesture recognition results, the prediction/recognition parameters are defined as follows: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). The evaluation formulas are shown as follows: accuracy (ACC), precision (P), recall (R), and F1-score, as presented in (1).

$$ACC = \frac{TN + TP}{TN + FP + TP + FN}; P = \frac{TP}{TP + FP}; R = \frac{TP}{TP + FN}; F1 - score = 2 * \frac{P * R}{P + R} \quad (1)$$

To build hand gesture recognition models, we fine-tune the hand gesture recognition model using the source code of the provided CNNs: SSD_VGG16, ResNet18, ResNet50, ResNet152, ResNeXt50, MobileNet_V3_small, MobileNet_V3_large, YOLOv5, YOLOv6, YOLOv7, YOLOv8, and YOLO-Nas. In this paper, we deployed the fine-tuning and testing models on a server with NVIDIA GeForce RTX 2080 Ti, 12 GB GPU. The programs were written in the Python language ($\geq$3.7 version) with the support of the CUDA 11.2/cuDNN 8.1.0 libraries. In addition, there are several libraries such as OpenCV, Matplotlib, Mmcls$\geq$0.20.1, Numpy, Packaging, Prettytable, and PyTorch 1.5+. With CNNs, we perform model fine-tuning over 100 epochs.

3.3. Results, discussions and challenges

As presented in the research taxonomy of this paper (section 2), we build a hand gesture recognition model based on two directions: first, we use SVM and RF of TML to build the hand gesture recognition; the second is to use CNNs for fine-tuning the recognition model with two families: i) SSD family with the CNNs: SSD_VGG16, ResNet18, ResNet50, ResNet152, ResNext50, MobileNet_V3_large, MobileNet_V3_small; ii) is the YOLO family with versions: YOLOv5, YOLOv6, YOLOv7, YOLOv8, and YOLO-Nas. The hand gesture recognition results of SVM, RF, YOLOv5, YOLOv6, YOLOv7, and YOLOv8 on the TQU-HG dataset are shown in Table 1.

The hand gesture recognition results of YOLO-Nas, MobileNet_V3_large, MobileNet_V3_small, ResNet152, ResNet18, ResNet50 on the TQU-HG dataset are shown in Table 2. The hand gesture recognition results of ResNext50, SSD_VGG16 on the TQU-HG dataset are shown in Table 4. Based on the results in Table 1, Table 2, and Table 4, the average results of hand gesture recognition with F1-score measures of SVM, RF, YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, MobileNet_V3_large, MobileNet_V3_small, ResNet152, ResNet18, ResNet50, ResNext50, SSD_VGG16 are 97.38%, 98.83%, 98.48%, 97.31%, 97.88%, 98.99%, 98.98%, 99.08%, 99.27%, 99.34%, 99.22%, 99.33%, 99.36%, and 96.03% respectively. The results show that SSD-VGG16 has the lowest result with an F1-score of 97.31%. The result of ResNext50 has the highest result with an F1-core of 99.36%. The results of all methods have F1-score greater than 95%. Figure 4(left) shows the confusion matrix of Mediapipe_TML (SVM) on the TQU-HG dataset. The results show that the gesture "high_five" is confused the most. Figure 4(right) shows the confusion matrix of YOLO-Nas on the TQU-HG dataset. The results show that the gesture "call_me" is confused the most but there are only a few cases, where the recognition results of hand gestures are greater than 99%.

Table 1. Hand gesture recognition results of SVM, RF, YOLOv5, YOLOv6, YOLOv7, and YOLOv8 on the TQU-HG dataset

	SVM				Random forest				YOLOv5			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)
peace	85.5	90.57	97.11	93.72	93.26	96.26	98.63	97.43	98.93	99.89	99.90	99.89
hang_loose	93.33	97.59	98.91	98.24	97.14	99.73	96.00	97.82	98.48	97.54	97.57	97.55
loser	94.69	97.2	99.75	98.45	93.37	95.72	99.48	97.56	98.65	99.79	99.87	99.80
high_five	70.86	91.84	99.67	95.59	97.87	99.73	100	99.86	98.39	98.05	98.01	98.07
talk_to_the _hand	97.22	99.22	93.43	96.22	97.18	97.93	99.74	98.82	98.67	99.07	98.98	99.02
you	98.68	98.92	98.92	98.92	95.51	99.18	96.79	97.97	98.85	99.38	99.03	99.20
good_job	95.98	97.45	99.74	98.58	98.48	98.73	100	99.36	99.43	98.43	99.14	98.78
dislike	94.47	96.39	99.77	98.05	96.67	97.54	100	98.75	99.64	98.19	98.61	98.04
power_to	95.19	97.30	100	98.63	94.67	97.51	98.99	98.24	99.22	98.67	99.22	98.94
ok	97.76	99.24	95.15	97.15	96.50	99.48	98.47	98.97	99.04	98.93	98.23	98.58
a_hole	94.63	95.10	98.98	97	96.10	98.50	99.49	98.99	99.35	99.60	97.32	98.45
good_lluck	87.66	91.51	98.82	95.02	93.30	99.43	98.86	99.14	98.00	96.59	96.46	96.52
bang_bang	97.51	98.99	98.99	98.99	92.71	96.85	99.19	98	98.44	99.66	96.51	98.06
rock	97.50	98.99	98.99	98.99	98.76	100	99.50	99.74	99.25	98.24	99.68	98.95
call_me	92.18	96.13	98.47	97.28	95.99	98.40	98.94	98.66	98.11	98.65	96.16	97.39
Average	92.87	96.42	98.44	97.38	95.83	98.33	98.93	98.83	98.71	98.31	98.48	98.48
	YOLOv6				YOLOv7				YOLOv8			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)
peace	99.35	98.84	97.12	97.97	98.04	98.79	96.54	97.65	99.85	98.51	99.45	98.98
hang_loose	98.41	97.48	96.39	96.93	98.63	99.30	95.04	97.12	99.04	98.58	99.71	99.14
loser	99.86	99.37	97.39	97.36	99.63	99.02	98.63	98.82	99.36	99.92	98.48	99.20
high_five	98.75	97.93	98.68	98.30	99.69	99.19	96.20	97.72	98.53	99.69	98.23	98.95
talk_to_the _hand	99.83	97.89	99.85	98.86	99.63	99.10	96.66	97.86	99.75	98.62	99.20	98.91
you	98.83	97.35	96.30	96.82	98.40	97.04	95.79	96.41	99.23	98.25	98.99	98.62
good_job	98.33	96.21	97.46	96.83	98.13	96.53	95.42	95.97	99.48	98.86	98.73	98.79
dislike	99.17	97.06	95.91	96.48	98.84	98.50	97.68	98.09	99.05	98.33	99.23	98.78
power_to	99.16	95.48	96.37	95.92	98.35	99.11	99.17	99.14	99.28	99.25	99.99	99.62
ok	98.92	97.63	95.34	96.47	98.55	96.60	95.18	95.88	98.33	98.96	98.18	98.57
a_hole	99.39	95.59	95.50	95.54	99.75	98.92	99.73	99.32	98.58	99.8	98.29	99.08
good_luck	98.48	95.96	99.47	97.68	99.26	98.10	99.10	98.60	98.57	98.36	98.98	98.67
bang_bang	99.68	98.44	99.26	98.85	98.80	98.76	99.01	98.88	99.33	98.61	98.02	98.31
rock	99.69	96.25	97.42	96.83	98.92	97.89	98.85	98.37	98.64	98.30	98.84	98.57
call_me	98.58	98.71	99.10	98.90	99.07	97.69	99.21	98.44	98.16	99.71	98.36	99.03
Average	99.09	97.34	97.43	97.31	98.91	98.30	97.48	97.88	99.01	98.91	98.84	98.99

The results of hand gesture recognition with F1-score measures of SVM, RF, YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, MobileNet_V3_large, MobileNet_V3_small, ResNet152, ResNet18, ResNet50, ResNext50, and SSD_VGG16 on the HaGRID dataset are shown below. The results of hand gesture recognition of YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, and SSD_VGG16 on the HaGRID dataset are presented in Table 3. The results of hand gesture recognition of ResNet152, ResNet18, ResNext50, SSD_VGG16, MobileNet_V3_small, and MobileNet_V3_large on the HaGRID dataset are shown in Table 5. Based on the hand gesture recognition results of the HaGRID dataset shown in Table 3, and Table 5. The result of classifying 18 gesture activities with the YOLOv7 model is the best, with an average of 99.46%, the lowest classification result is MobileNet_V3_small with F1-score of 86.4%. Figure 5 (left) and Figure 5 (right) show the confusion matrices of YOLOv7, and YOLO-Nas, respectively when testing on the HaGRID dataset.

Table 2. Hand gesture recognition results of YOLO-Nas, MobileNet_V3_large, MobileNet_V3_small, ResNet152, ResNet18, and ResNet50 on the TQU-HQ dataset

	YOLO-Nas				MobileNet_V3_large				MobileNet_V3_small			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)
peace	98.69	99.93	98.60	99.26	98.64	99.56	99.50	99.53	98.52	99.22	99.19	99.20
hang_loose	98.80	99.04	98.29	98.66	99.23	99.09	99.39	99.24	99.94	99.56	99.00	99.28
loser	98.66	99.27	98.68	98.98	98.11	98.69	99.09	98.89	99.85	98.93	99.73	99.33
high_five	99.50	99.20	99.19	99.19	99.81	98.36	99.36	98.86	98.83	98.77	99.73	99.25
talk_to_the _hand	99.15	99.11	99.32	99.21	98.84	98.35	99.09	98.72	98.51	99.68	99.69	99.68
you	99.43	99.85	98.48	99.16	98.50	98.04	99.06	98.55	99.29	99.14	99.45	98.17
good_job	98.37	98.20	98.54	98.37	99.77	98.80	99.48	99.14	99.24	99.98	99.82	99.90
dislike	99.54	99.53	98.11	98.81	99.93	99.72	98.73	99.22	98.17	99.56	99.01	99.28
power_to	98.99	98.90	100.00	99.45	99.48	98.99	99.31	99.15	99.75	99.98	98.98	99.48
ok	99.15	99.77	98.73	99.25	99.34	98.84	98.75	99.29	98.62	99.70	98.93	99.31
a_hole	98.88	99.70	98.07	98.21	98.69	98.90	99.48	99.19	99.93	99.75	99.44	99.59
good_luck	99.66	98.88	99.63	99.25	98.55	99.99	98.56	99.27	99.69	99.79	99.37	99.58
bang_bang	99.18	98.23	98.34	98.28	99.04	99.39	99.13	99.26	98.30	99.29	99.78	99.53
rock	98.68	98.66	99.42	99.04	98.70	98.90	99.18	99.04	99.46	98.32	99.47	98.89
call_me	99.78	99.57	99.60	99.58	98.02	99.60	98.25	98.92	98.20	98.08	99.15	98.61
Average	99.09	99.18	98.86	98.98	98.97	99.01	99.09	99.08	99.08	99.31	99.38	99.27
	ResNet152				ResNet18				ResNet50			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)
peace	99.40	99.90	98.77	99.33	99.33	98.78	99.40	99.09	98.54	99.07	99.68	99.37
hang_loose	99.86	98.90	99.60	99.25	99.27	99.24	98.97	99.10	98.01	98.75	98.83	98.79
loser	99.36	99.61	98.88	99.24	99.28	99.06	99.84	99.45	99.53	99.47	99.07	99.27
high_five	99.58	99.39	99.27	99.33	98.00	99.91	99.36	99.63	98.58	99.74	98.92	99.33
talk_to_the _hand	99.25	99.25	99.08	99.16	99.96	98.87	98.91	98.89	98.47	98.95	99.60	99.27
you	98.53	99.96	98.92	99.44	98.53	99.36	99.98	99.67	98.42	98.81	99.82	99.31
good_job	99.84	99.72	99.88	99.80	99.34	98.71	99.12	98.91	99.31	99.03	99.10	99.06
dislike	99.15	98.99	99.97	99.48	98.17	99.02	99.40	99.21	99.73	99.03	99.73	99.38
power_to	99.23	99.05	99.50	99.27	98.28	98.87	99.12	98.99	98.36	99.85	99.14	99.49
ok	98.29	98.97	99.31	99.14	99.12	99.87	99.16	99.51	99.50	99.50	99.81	99.65
a_hole	98.12	99.05	99.61	99.33	99.45	98.97	98.72	98.84	99.04	99.83	99.44	99.63
good_luck	99.56	98.94	99.13	99.03	99.50	99.04	99.58	99.31	98.65	98.73	99.56	99.14
bang_bang	98.75	99.49	98.78	99.13	99.72	99.44	98.88	99.16	98.70	98.91	99.36	99.13
rock	99.56	99.60	99.82	99.71	99.93	98.98	99.81	99.39	99.13	99.47	99.45	99.46
call me	98.93	99.61	99.62	99.61	98.27	99.09	99.38	99.23	99.97	99.62	99.93	99.77
Average	99.16	99.36	99.34	99.34	99.07	99.14	99.30	99.22	98.92	99.25	99.42	99.33

The results are shown in Table 1, Table 2, Table 3, and Table 5 show that YOLOv8 and YOLO-Nas are the two latest improved versions of YOLO with the best sustained results. With F1-score, YOLO8, and YOLO-Nas have results greater than 98%. In particular, YOLO can perform calculations in the CPU, so it is very suitable for building a hand gesture recognition model based on YOLOv8 or YOLO-Nas applied to control systems using table gestures. hands in reality. The versions of MobileNetV3 and Resnet have lower results and must perform calculations on the GPU, so they are not suitable for applying and building control systems using commercial hand gestures. In this paper, we also perform the processing time of the two branches with each approach of Mediapipe_TML model and CNNs. The models are evaluated on both CPU and GPU devices. The results in terms of computation time are shown in Table 6. Table 6 shows the processing time of the networks when tested on the test set. CPU for Mediapipe + SVM, Mediapipe + RF, GPU for MobileNet_V3_large, MobileNet_V3_small, ResNet152, ResNet18, ResNet50, ResNext50, SSD_VGG16, and running on both CPU and GPU for YOLOv5, YOLOv6, YOLOv7, YOLOv8, and YOLO-Nas. The results show that when running on CPU, Mediapipe_TML feedforward can reach speeds up to 15.78 fps while CNNs show the lowest speed of 3.44 fps with YOLOv5, and the highest is 6.19 fps for YOLOv8. When running on GPU, CNNs with small models like MobileNet_V3_small, and new models like YOLOv8 have speeds of 18.41 fps and 18.28 fps respectively. These are the models with the fastest processing times.

Table 3. The results of hand gesture recognition of YOLOv5, YOLOv6, YOLOv7, YOLOv8, YOLO-Nas, and SSD_VGG16 on the HaGRID dataset

	YOLOv6				YOLOv7				YOLOv8			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)
call	99.5	99.5	99.5	99.5	99.74	99.96	99.67	99.74	99.39	99.55	99.34	99.44
dislike	99.5	99.6	99.5	99.54	99.76	99.94	99.93	99.91	99.17	99.11	98.81	98.96
fist	99.5	99.8	99.6	99.69	99.78	99.79	99.71	99.72	99.09	99.44	99.98	99.71
four	99.5	99.3	99.0	99.14	99.77	99.75	99.68	99.65	98.67	99.84	99.34	99.59
like	99.5	99.5	99.0	99.24	99.67	99.75	99.66	99.64	99.17	98.95	99.42	99.18
mute	99.5	99.7	99.9	99.79	99.80	99.73	99.65	99.67	98.91	99.36	98.90	99.13
ok	99.5	99.4	99.2	99.29	98.90	98.77	97.89	98.24	99.92	98.73	99.58	99.15
one	99.4	99.0	98.6	98.79	99.78	99.46	98.32	98.87	98.73	99.0	98.93	98.96
palm	99.3	99.4	99.7	99.54	98.76	98.79	99.60	99.19	99.02	98.72	99.09	98.90
peace	99.3	99.0	98.5	98.74	99.30	99.65	99.67	99.78	99.90	99.25	99.13	99.19
peace_inverted	99.5	99.8	98.7	99.24	99.72	99.89	99.65	99.76	98.66	99.42	98.95	99.18
rock	99.5	99.5	99.3	99.39	98.59	98.76	98.60	98.65	98.41	99.15	98.93	99.04
stop	99.4	98.8	99.8	99.29	99.76	99.50	100	99.74	99.43	99.0	98.70	98.85
stop_inverted	99.5	99.7	99.7	99.7	98.54	98.47	98.77	98.61	98.02	99.86	98.91	99.38
three	99.5	99.0	98.6	98.79	99.78	99.63	99.81	99.71	98.58	99.85	99.41	99.63
three2	99.5	99.0	98.6	98.79	99.74	99.82	99.87	99.84	99.34	98.73	99.04	98.88
two_up	99.5	99.3	98.7	98.99	99.72	100	99.82	99.90	99.84	99.85	99.02	99.43
two_up_inverted	99.5	99.3	99.2	99.24	99.75	99.83	99.81	99.82	99.08	99.68	98.96	99.30
Average	99.46	99.36	99.17	99.26	99.49	99.52	99.45	99.46	99.07	99.30	99.13	99.21
	YOLO-Nas				YOLOv5				SSD_VGG16			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P(%)	R (%)	F1-score (%)
call	99.5	99.6	98.9	99.24	99.5	99.4	99.4	99.4	97.99	97.08	97.59	97.33
dislike	99.5	99.2	99.6	99.39	99.5	99.3	99.6	99.44	95.59	95.40	94.50	94.95
fist	99.5	99.8	99.4	99.59	99.5	99.7	99.6	99.64	97.52	96.30	95.62	95.96
four	99.4	99.2	98.9	99.04	99.5	98.9	99.1	98.99	95.10	96.90	94.89	95.88
like	99.4	99.4	98.8	99.09	99.5	99.2	99.1	99.14	97.91	96.88	94.35	95.60
mute	99.5	99.8	99.9	99.84	99.5	99.6	99.9	99.74	97.07	95.18	95.13	
ok	99.5	99.5	98.9	99.19	99.5	99.5	99.1	99.29	95.51	95.88	95.50	95.69
one	99.4	98.9	98.7	99.79	99.5	98.7	99.1	98.89	97.80	94.99	93.66	94.32
palm	99.4	99.4	99.3	99.34	99.4	99.2	99.6	99.39	95.85	97.95	95.42	96.67
peace	99.2	98.6	98.4	99.49	99.1	98.7	98.8	98.74	97.11	96.34	96.50	96.42
rock	99.5	99.6	99.4	99.49	99.5	99.7	99.6	99.64	97.67	97.95	94.77	96.33
stop	99.3	99.1	99.8	99.44	99.4	99.1	99.8	99.44	97.58	95.33	95.04	95.18
stop_inverted	99.5	99.9	99.6	99.74	99.5	99.6	99.8	99.69				
three	99.5	98.8	98.2	99.49	99.5	98.8	98.7	98.74	97.90	96.48	96.24	96.36
two_up	99.5	99.1	98.7	98.89	99.4	99.1	98.9	98.89	97.67	97.29	93.29	95.25
two_up_inverted	99.5	99.6	99.0	99.29	99.5	99.3	99.3	99.3	95.56	97.73	96.03	96.87
three2	99.4	99.8	98.8	99.29	99.5	99.7	98.9	99.28	95.81	97.11	97.42	97.26
peace_inverted	99.5	99.8	98.6	99.19	99.5	99.7	99.1	99.39	97.86	96.92	96.75	96.83
Average	99.44	99.39	99.05	99.37	99.46	99.28	99.30	99.27	96.91	96.57	95.45	96.05

Table 4. Hand gesture recognition results of ResNext50, SSD_VGG16 on the TQU-HG dataset

	ResNext50				SSD_VGG16			
Hand gestures	ACC (%)	P (%)	R (%)	F1-score (%)	ACC (%)	P (%)	R (%)	F1-core (%)
peace	99.59	99.12	99.51	99.31	94.55	97.02	95.03	96.01
hang_loose	99.12	99.98	98.86	99.42	95.57	97.11	93.59	95.32
loser	99.00	99.37	99.91	99.64	97.40	95.87	94.96	95.41
high_five	98.11	99.16	99.78	99.47	96.62	95.01	93.88	
talk_to_the_hand	98.05	99.56	98.97	99.26	96.71	96.27	96.26	96.26
you	98.76	99.56	99.46	99.51	97.03	95.29	96.46	95.87
good_job	98.17	98.78	99.2	98.99	97.88	96.88	94.26	95.55
dislike	99.33	99.06	99.20	99.13	96.64	96.67	95.49	96.08
power_to	98.83	99.92	98.86	99.39	97.05	95.18	95.72	95.45
ok	98.56	98.80	99.26	99.03	97.22	97.63	94.09	95.83
a_ole	99.18	99.89	99.42	99.65	96.56	97.12	96.66	96.89
good_luck	99.64	99.22	99.36	99.29	97.45	95.90	96.54	96.22
bang_bang	99.56	99.36	99.79	99.57	97.14	97.12	95.46	96.28
rock	98.02	99.40	99.32	99.36	97.42	96.30	97.72	97.0
call me	98.67	99.40	99.77	99.40	96.58	97.47	95.15	96.30
Average	98.83	99.37	99.37	99.36	96.82	95.89	95.80	96.03

Table 5. The results of hand gesture recognition of ResNet152, ResNet18, ResNext50, SSD_VGG16, MobileNet_V3_small, and MobileNet_V3_large on the HaGRID dataset

Models	ResNext50	ResNet152	ResNet18	MobileNet_V3_small	MobileNet_V3_large
Matrix	F1-score (%)	F1-score (%)	F1-score (%)	F1-score (%)	F1-score (%)
Average	98.3	95.5	97.5	86.4	91.9

Table 6. Processing time (frame/second-fps) of inference method on the testing set of the TQU_HG dataset

Models	Processing time (fps)	
	Devices	
	CPU	GPU
Mediapipe + SVM	14.54	_
Mediapipe + RD	15.78	_
MobileNet_V3_large	_	16.22
MobileNet_V3_small	_	18.41
SSD_VGG16	_	13.1
ResNet152	_	10.23
ResNet18	_	13.63
ResNet50	_	11.87
ResNext50	_	10.92
YOLOv5	3.44	15.51
YOLOv6	4.2	14.65
YOLOv7	3.59	15.72
YOLOv8	6.19	18.28
YOLO-Nas	5.63	16.88

Figure 6 shows 10 results of correctly recognizing hand gestures based on Mediapipe_TML (SVM), the skeleton of the hand is drawn with Mediapipe hands on the TQU-HG dataset. The hand data area is defined by a green bounding box, and the label of the recognized hand gesture is shown in red text. Figure 7 shows 10 hand gesture misrecognition results based on Mediapipe_TML (SVM). The results on the TQU-HG dataset based on CNNs are very high, often over 99%. The result of the Mediapipe_TML method is lower. The TQU-HG dataset also presents some challenges such as low image resolution (640×480) pixels, and low image brightness due to collection in low light conditions. Many frames show the hand in a tilted position. In such cases, Mediapipe hands can not detect and can not estimate the hand pose. In the TQU-HG dataset, only 96.85% of the frames have the hand detected and the hand skeleton estimated.

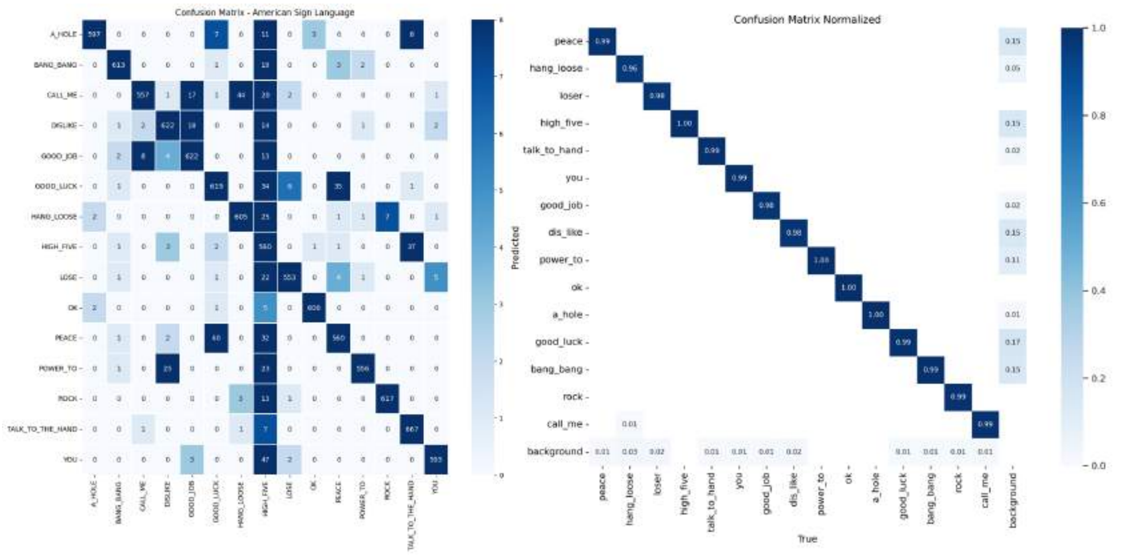

Figure 4. Left is the confusion matrix of Mediapipe_TML (SVM) on the TQU-HG dataset, right is the confusion matrix of YOLO-Nas on the TQU-HG dataset

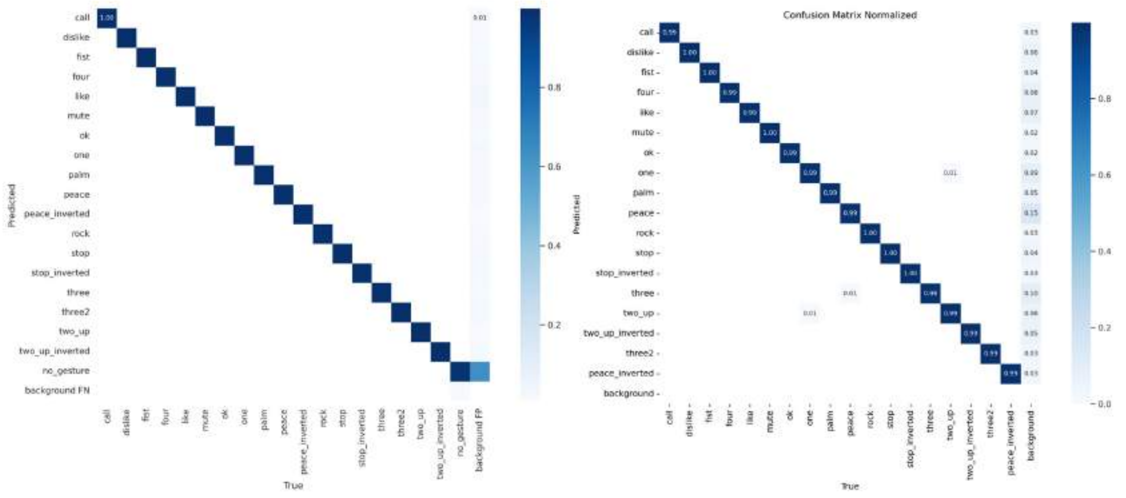

Figure 5. Left is the confusion matrix of YOLOv7 on the HaGRID dataset, right is the confusion matrix of YOLOv-Nas on the HaGRID dataset

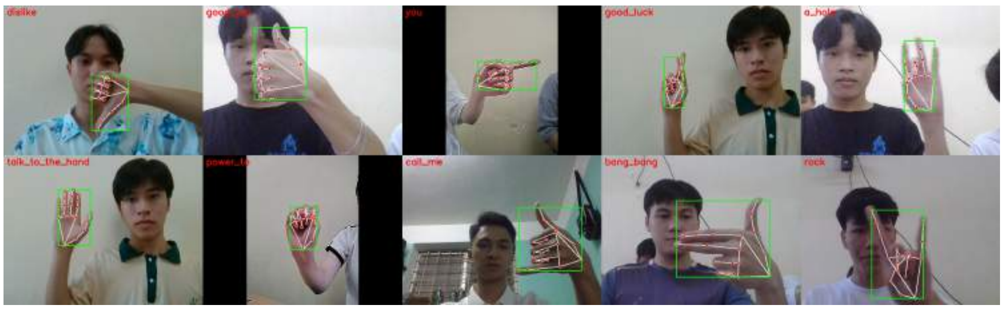

Figure 6. Illustration of 10 results of correct hand gesture recognition based on Mediapipe_TML (SVM)

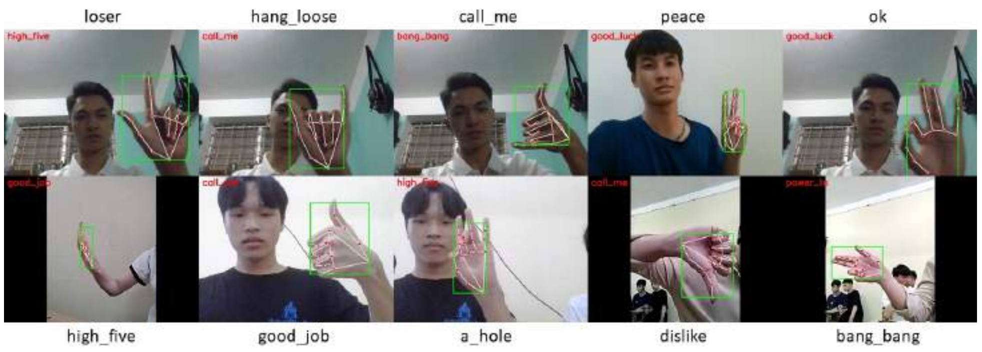

Figure 7. Illustration 10 hand gesture misrecognition results based on Mediapipe_TML (SVM)

4. CONCLUSIONS AND FUTURE WORKS

Hand gesture recognition is not a new problem in computer vision. But to build systems with high accuracy and meet requirements in many different contexts and conditions, the recognition model needs to be fine-tuned in many different environments and conditions. To solve that problem, in this paper, we have published the TQU-HG dataset including 60,000 photos collected from 20 people in low light conditions, with low image resolution (640×480) pixels. At the same time, we also performed a comparative study with two branches, Mediapipe_TML and CNNs. The results of the comparative study were evaluated on the TQU-HG and HaGRID datasets. The highest result on the TQU-HG dataset is ResNext50 with an average F1-score of 99.36%, and on the HaGRID dataset is the YOLOv7 with an average F1-score of 99.46%. At the same time, we also show and analyze in detail the challenges and errors of hand gesture recognition when performed on the TQU-HG dataset. Shortly, we will apply a hand gesture recognition model trained from many datasets, with many different conditions and contexts, such as building a hand gesture recognition model using YOLOv8 based on pre-training the model on the TQU-HG and HaGRID datasets. From there, develop practical applications for human-machine communication and machine translation based on hand gesture language recognition for deaf and mute people.

ACKNOWLEDGMENT

This research is funded by Tan Trao University in Tuyen Quang province, Vietnam.

REFERENCES

[1] V. Chang, R. O. Eniola, L. Golightly, and Q. A. Xu, "An exploration into human–computer interaction: hand gesture recognition management in a challenging environment," *SN Computer Science*, vol. 4, no. 5, 2023, doi: 10.1007/s42979-023-01751-y.

[2] B. I. Alabdullah *et al.*, "Smart home automation-based hand gesture recognition using feature fusion and recurrent neural network," *Sensors*, vol. 23, no. 17, 2023, doi: 10.3390/s23177523.

[3] J. Qi, L. Ma, Z. Cui, and Y. Yu, "Computer vision-based hand gesture recognition for human-robot interaction: a review," *Complex and Intelligent Systems*, 2023, doi: 10.1007/s40747-023-01173-6.

[4] K. Lakshmi, Samiya, M. Sri Lakshmi, M. Rudra Kumar, and P. K. Singuluri, "Real-time hand gesture recognition for improved communication with deaf and hard of hearing individuals," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 6s, pp. 23–37, 2023.

[5] T. Evgeniou and M. Pontil, "Support vector machines: theory and applications," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001, vol. 2049 LNAI, no. May, pp. 249–257, doi: 10.1007/3-540-44673-7_12.

[6] C. Tu, H. Liu, and B. Xu, "AdaBoost typical algorithm and its application research," in *MATEC Web of Conferences*, 2017, vol. 139, doi: 10.1051/matecconf/201713900222.

[7] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001, doi: 10.1023/A:1010950718922.

[8] S. Escalera, V. Athitsos, and I. Guyon, "Challenges in multimodal gesture recognition," *Journal of Machine Learning Research*, vol. 17, pp. 1–54, 2016, doi: 10.1007/978-3-319-57021-1_1.

[9] A. Kapitanov, K. Kvanchiani, A. Nagaev, R. Kraynov, and A. Makhliarchuk, "HaGRID - HAnd gesture recognition image dataset," *arXiv:2206.08219*, 2022, [Online]. Available: http://arxiv.org/abs/2206.08219.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2016, vol. 2016-Decem, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[11] S. Xie, R. Girshick, and P. Doll, "Aggregated residual transformations for deep neural networks," *arxiv*, 2016, https://arxiv.org/abs/1611.05431v2.

[12] A. Howard *et al.*, "Searching for mobileNetV3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, vol. 2019-Octob, pp. 1314–1324, doi: 10.1109/ICCV.2019.00140.

[13] M. Wu *et al.*, “Deep convolutional neural networks for multiple histologic types of ovarian tumors classification in ultrasound images,” *Frontiers in Oncology*, vol. 13, p. 1154200, 2023, doi: 10.3389/fonc.2023.1154200.
[14] D. Reis, J. Kupec, J. Hong, and A. Daoudi, “Real-time flying object detection with YOLOv8,” *arxiv*, 2023, [Online]. Available: http://arxiv.org/abs/2305.09972.
[15] YOLO-NAS, “A next-generation, object detection foundational model generated by deci’s neural architecture search technology,” *deci*, 2023.
[16] K. Eugene and S. Harpreet, “YOLO-NAS by deci achieves SOTA performance on object detection using neural architecture search,” *deci*, 2023.
[17] F. Zhang *et al.*, “MediaPipe hands: on-device real-time hand tracking,” *arxiv*, 2020, [Online]. Available: http://arxiv.org/abs/2006.10214.
[18] M. Carranza-Garcia, J. Torres-Mateo, P. Lara-Benitez, and J. Garcia-Gutiérrez, “On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data,” *Remote Sensing*, vol. 13, no. 1, p. 89, 2020.
[19] T. Diwan, G. Anirudh, and J. V. Tembhurne, “Object detection using YOLO: challenges, architectural successors, datasets and applications,” *Multimedia Tools and Applications*, vol. 82, no. 6, pp. 9243–9275, 2023, doi: 10.1007/s11042-022-13644-y.
[20] W. Liu *et al.*, “SSD: Single shot multibox detector,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[21] Y. Zhu, S. Li, W. Du, and Y. Du, “Identification of table grapes in the natural environment based on an improved Yolov5 and localization of picking points,” *Precision Agriculture*, vol. 24, no. (4), pp. 1–22, 2023.
[22] C. Li *et al.*, “YOLOv6: a single-stage object detection framework for industrial applications,” *arxiv*, 2022, [Online]. Available: http://arxiv.org/abs/2209.02976.
[23] A. Ansari, “Classifying data using support vector machines (SVMs) in Python,” *GeeksforGeeks*, 2023.
[24] Amandp, “Random forest classifier using scikit-learn,” *geeksforgeeks*, 2023, https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/, 2022, (accessed 19-December-2023).
[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: unified, real-time object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 779–788, doi: 10.1109/CVPR.2016.91.
[26] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: optimal speed and accuracy of object detection,” *arxiv*, 2020. http://arxiv.org/abs/2004.10934 (accessed Oct. 13, 2022).
[27] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” *arxiv*, pp. 7464–7475, 2022, [Online]. Available: http://arxiv.org/abs/2207.02696.
[28] deci.ai, “he deep learning platform production-grade performance, faster,” *deci*, 2023. https://deci.ai/ (accessed Sep. 19, 2023).
[29] X. Chu, L. L. Bo, and Z. Meituan, “Make RepVGG greater again: a quantization-aware approach,” *arxiv*, 2022, https://arxiv.org/pdf/2212.01593.pdf.
[30] W. Liu *et al.*, “SSD: single shot multibox detector,” in *European Conference on Computer Vision*, 2016, vol. 1, pp. 21–37, doi: 10.1007/978-3-319-46448-0.
[31] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.
[32] V. Sudha and T. R. Ganeshbabu, “A convolutional neural network classifier VGG-19 architecture for lesion detection and grading in diabetic retinopathy based on deep learning,” *Computers, Materials and Continua*, vol. 66, no. 1, pp. 827–842, 2021, doi: 10.32604/cmc.2020.012008.
[33] C. Szegedy *et al.*, “Going deeper with convolutions,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1–9, 2015, doi: 10.1109/CVPR.2015.7298594.
[34] A. G. Howard *et al.*, “MobileNets: efficient convolutional neural networks for mobile vision applications,” *arxiv*, 2017, [Online]. Available: http://arxiv.org/abs/1704.04861.
[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998, doi: 10.1109/5.726791.
[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, pp. 1097–1105, 2012.
[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 4510–4520, doi: 10.1109/CVPR.2018.00474.
[38] lsell Lauren Sell, “Label studio is a modern, multi-modal data annotation tool,” *github*, 2016, https://github.com/HumanSignal/labelImg (accessed 19-September-2023).

BIOGRAPHIES OF AUTHORS

Van-Dinh Do 1998: Graduated from University with major in Automation, Hanoi University of Science and Technology; 2005: Graduated with Master’s degree in Automation Engineering, Hanoi University of Science and Technology; 2018: Graduated with a doctorate in Control and Automation Engineering, Hanoi University of Science and Technology; current job: klecturer, administrator, Sao Do University; the main researches are the application of artificial intelligence in measurement, control and automation solutions, and intelligent measuring devices. He can be contacted at email: dinh.dv@saodo.edu.vn.

Van-Hung Le received M.Sc. degree at Faculty Information Technology Hanoi National University of Education (2013). He received Ph.D. degree at International Research Institute MICA HUSTCNRS/UMI-2954-INP Grenoble (2018). Currently, he is a lecture of Tan Trao University. His research interests include Computer vision, RANSAC and RANSAC variation and 3-D object detection, recognition; machine leaning, and deep learning. He can be contacted at email: van-hung.le@mica.edu.vn.

Huu-Son Do was born in 1997. Currently he is a Bachelor student majoring in computer science, Faculty of Information Technology, Tan Trao University, Tuyen Quang, Vietnam. His main research interests are artificial Intelligence, computer vision, machine learning, and deep learning. He can be contacted at email: dosonhytq@gmail.com.

Van-Nam Phan was born in 1997. Currently he is a Bachelor student majoring in computer science, Faculty of Information Technology, Tan Trao University, Tuyen Quang, Vietnam. His main research interests are artificial Intelligence, computer vision, machine learning, and deep learning. He can be contacted at email: nampv0903@gmail.com.

Trung-Hieu Te was born in 1997. Currently he is a Bachelor student majoring in computer science, Faculty of Information Technology, Tan Trao University, Tuyen Quang, Vietnam. His main research interests are artificial Intelligence, computer vision, machine learning, and deep learning. He can be contacted at email: ttrunghieu97@gmail.com.