EchoGest: A Highly Scalable Unseen Gesture Recognition System Based on Feature-wise Transformation

Yunshu Wang, Weiyu Chen, Weiwei Lu, Yanbo He, Yongpan Zou, Member, IEEE, Kaishun Wu, Fellow, IEEE, and Victor C. M. Leung, Life Fellow, IEEE

Abstract—Recent research studies have made significant progress in acoustic-based gesture recognition. However, existing methods lack the capability to expand to customized gestures and adapt to different practical environments. We propose a highly scalable gesture recognition system called EchoGest which integrates a well-designed feature-wise transformation layer into prototypical network framework, and accomplishes unseen gesture recognition with a device’s built-in speaker and microphone. Our key insight gains involving the similarity between query sample representations and class prototypes in the embedding space, and thus enabling the scalability to unseen gestures. Meanwhile, we introduce a feature transformation layer to linearly adjust feature maps and propose an efficient two-stage training strategy to obtain regularized parameters for this layer. Specifically, this layer employs affine transformation to enhance intermediate feature activations and yield more diverse feature distributions for cross-domain recognition, and it improves recognition accuracy by 10% in 1-shot cases. We train the system with a collected a letter gestures (i.e., writing ‘A’ to ‘Z’) dataset and test it on a digit gestures (i.e., writing ‘0’ to ‘9’) dataset with 10 volunteers. The results show that EchoGest can recognize unseen digit gestures with an accuracy of 93.7% in 2-shot cases, and 93.2% in the leave-one-user-out testing setting. We also explore a semi-supervised clustering approach in which each user’s data can be used to update his or her prototypes for personalized customization. The comprehensive experiments also verify that EchoGest remain good performance across various environments, age groups, and different devices.

Index Terms—Gesture recognition; Acoustic Sensing; Prototypical Network

I. INTRODUCTION

N-air gesture recognition has attracted widespread attention as an emerging human-computer interaction method [2], [23]. With the popularity of smart devices like smartwatches, the demand for text input in limited screen space has increased [19]. However, traditional touch screen input is not convenient enough due to the limited screen size. In-air gesture recognition, as shown in Fig. 1, offers users a more natural and convenient way of writing in the air, enhancing the practicality and functionality of smart devices in daily life [42]. Researchers have explored various non-contact gesture recognition methods based on Wi-Fi [12], [29], mmWave [24], RFID [34], [37], and inertial measurement units (IMU) [1], [11]. Nevertheless, they often require specialized devices (e.g., RFID readers and tags, Wi-Fi transceivers) or demand additional hardware on users’ devices. IMU-based solutions typically require users to hold or wear extra hardware when writing in the air. In a summary, these methods have certain limitations in practical application. In contrast, present commercial smart devices are commonly equipped with built-in speakers and microphones, making acoustic-based gesture recognition systems nearly cost-free to deploy rapidly on smart devices. Despite various background noise in daily life, utilizing modulated high-frequency acoustic signals for gesture recognition can significantly reduce noise interference, and human ears cannot perceive high-frequency signals, avoiding the sound disturbance.

Fig. 1. The potential scenarios of acoustic-based in-air gesture recognition.

However, existing research on acoustic-based gesture recognition often involves extracting features and inputting them into traditional convolutional neural networks for classification. This means that the number of recognized gestures needs to be fixed during the training process, and if the user needs to customize new gestures or use the system in a completely new environment, a lot of data collection or retraining is required. Additionally, overfitting must be considered, limiting practical applications. Earlier works such as Dolphin [21], Soundwrite [48], and later high-precision, fine-grained works like Ultracesture [14], RobuCIR [38], AMT [15], Amaging [36], and AO-Finger [44] only accept predefined gestures and conditions without considering new customized (i.e., user-defined) gestures from users and extensions to different envi-
environments, devices, and orientations. Limited scalability results in data collection overhead and training overhead for users. To collect enough samples of each unseen gesture class for a new user and then retrain the model is inconvenient in practice.

The emerging field of few-shot learning (FSL) [39] has provided a promising solution to the problem. Unlike traditional deep learning approaches, it focuses on training models with minimal data to quickly adapt to new tasks based on a small amount of relevant data. Some works attempt to apply FSL methods to human gesture recognition. DHGR [46] uses Doppler radar to capture gestures and performed gesture recognition using weighted relation network (RN) [9]. WiGr [49] develops a Wi-Fi-based gesture recognition system with a dual-path prototypical network for domain-invariant feature extraction and cross-domain recognition. OneFi [43] uses meta-learning for one-shot fine-tuning to recognize Wi-Fi gestures. In order to realize zero training-cost for using, we choose metric-learning methods in FSL, which only needs to provide shots for comparison without fine-tuning the model.

In our feasibility study, we initially implement a simple gesture recognition system using prototypical networks (PN) [28] and evaluate its usability. We develop an original system on Android, collect ‘A’-‘Z’ letters data from the first 10 volunteers, train a ResNet18-backbone PN and have another 20 volunteers test our application. According to volunteers’ feedback, we draw Fig. 2 as a schematic diagram. In ideal conditions, the application occupy less storage space, have a higher recognition accuracy, and it can reduce the time to prepare customized gestures as much as possible, that is, provide as few shots as possible but still achieve a high accuracy. During testing, volunteers can customize their own gestures for recognition. To provide a quantifiable indicator, we ask them to use ‘0’ to ‘9’ digit gestures as a set of customized gestures. This feasibility analysis reveals that the basic model achieves around 80% accuracy for these ten digits with 2-shot, which is not high. We use cross-task problem to describe this problem. Another practical problem is that we have observed users have their own usage habits, such as placing the phone horizontally rather than vertically, or holding on hand instead of on table. In addition, real-world noise is also a factor. All of the above factors differ from the training data collected in the lab. When faced with these situations, a gesture recognition system encounters obvious performance degrading, making it challenging to meet real-world usage requirements. We use cross-domain problem to describe this problem in the following. In addition, shots (serve as prototypes) provided by users are biased [16] due to the data scarcity in few-shot scenarios, which is also a problem that cannot be ignored.

To overcome the cross-task and cross-domain problem, we analyze that the reason of low accuracy is relatively inabundant distribution of our training dataset, indicating a need for simulating a broader range of feature distributions, and incorporate ideas from FSL methods [3], [20], [26], [30], [31]. We design a feature transformation layer for linearly transforming feature maps, providing appropriate and effective parameter training methods. The layer enhances intermediate feature activation through affine transformations to generate a more diverse feature distribution, which facilitates cross-domain recognition. Moreover, our training process uses an Multi-Layer Perceptron (MLP) to output the required hyper-parameters of our feature transformation layer instead of optimizing directly with the gradient of hyper-parameters [30], which is also the key to our approach. Experimental results also support that by optimizing MLP parameters, the desired hyper-parameters can be obtained indirectly rather than directly to improve the model performance. Our method increases model size and inference cost minimally while significantly improving accuracy across various gesture type, experimental conditions and environments. When users only provide 1-shot, the accuracies in cross-domain cases are improved by about 10%.

To address the problem that the users’ provided shots might be biased and writing habits changes over time, we explore updating prototypes using users’ own handwritten data. Drawing inspiration from semi-supervised PN [25], we collect each user’s unlabeled data through semi-supervised clustering. Updating the prototypes through clustering achieves a user-customized effect. We demonstrate its effectiveness and potential through tests on the collected data.

In this paper, we propose and design a real-time acoustic gesture recognition system, EchoGest, that integrates our feature-wise transformation layer into prototypical network framework. We train with 26 uppercase letter gestures and test with 10 digit gestures which are considered as unseen gestures. Note that those unseen gestures can be extended to any other customized gestures as also evaluated in Sec. VI-D. The 2-shot accuracy reaches 93.7%, and 93.2% accuracy in cross-person testing. The top-2 accuracy of different gestures all exceed 98%. Moreover, its accuracies in various cross-domain tests also exceed 90%. We also explore a semi-supervised clustering approach to update prototypes using each user’s own data, achieving personalized customization.

Our contributions are listed as follows:

1) We design a feature transformation layer to linearly transform feature maps and proposed an efficient two-stage training strategy. By training a MLP, we can obtain all hyper-parameters for MLP.
outputs. During testing, hyper-parameters remain fixed, adding minimal model size and inference cost while improving accuracy across different settings.

2) We explore using user’s own writing data to update prototypes. By leveraging semi-supervised clustering, results show that updating prototypes can consistently improve model performance.

3) We conduct a series of experiments to validate the effectiveness of our designed modules and the overall robustness of our real-time acoustic gesture recognition system. Additionally, experiments on smartwatches show the potential of acoustic gesture recognition technology in smartwatch applications.

The remainder of this paper is organized as follows. Sec. II reviews the related works. Sec. III introduces preliminaries. Sec. IV shows our system design, detailing the entire process and classification framework. Sec. V introduces our feature-wise transformation layer, including its principles, functions, training methods and how to integrate it into PN framework. Furthermore, this section briefly outlines updating prototypes using clustering. Sec. VI shows system implementation and evaluation in various aspects. Sec. VI and Sec. VIII are our discussion and conclusion.

II. RELATED WORKS

A. Acoustic-based Gesture Recognition

The use of Doppler effect in soundwave perception is a direct approach in gesture recognition technology. Besides Doppler frequency shift, other methods such as Frequency Modulated Continuous Wave (FMCW) and Channel Impulse Response (CIR) are also utilized. The early research Soundwave [8] explores the use of commercial off-the-shelf audio components to recognize hand gestures in the air. SoundWrite [48] utilizes MFCC [17] to describe handwritten features, employing KNN to match capture features. Recent research advancements in gesture recognition have aimed for fine-grained granularity, higher accuracy, smaller training set sizes, and more targets. With the development of transfer learning [41], few-shot learning [39], and generative adversarial networks [40], these techniques have also been applied in the field of gesture recognition. The work [22] conducts few-shot learning for gesture recognition using electromyography signals. Despite its relatively basic approach, the accuracy for 5-way 5-shot recognition of new gestures is only 73%. The work [35] applies GAN to generate virtual samples from real samples, testing it on millimeter-wave data. Although different in methods and carriers, these approaches provide valuable insights for acoustic-based gesture recognition.

In the last few years, more related works have appeared. Ipanel [5] utilizes sounds generated by fingers sliding on a table to recognize writing contents with an average accuracy of 85% for 10 digits and 26 letters. UbiWriter [45] treats handwriting signals as complete trajectories and achieves recognition an accuracy of 69.23% for lowercase letters and 91.8% for words. Ultragesture [14] achieves high-precision gestures tracking based on CIR measurements. RobuCIR [38] employs frequency hopping to reduce frequency selective fading and accurately recognize 15 kinds of hand movements. Echowrite2.0 [51] proposes a new model training strategy and dataset augmentation method to minimize required training data. DHGR [46] makes use of meta-learning methods to recognize 5 different gestures with an accuracy of 91% with one shot. AO-Finger [44] designs a wristband with a microphone and two high-speed optical motion sensors, and proposes a multimodal CNN-Transformer model to recognize light tapping, pinching, light patting, and no motion. However, compared with these works, EchoGest has the following unique advantages: 1) the utilization of commercial devices without any additional hardware equipment; 2) the extension of recognizing unseen gestures and adaptation to different domains; 3) the exploration of a semi-supervised clustering method to update prototypes with unlabeled data to boost the recognition accuracy. We have also summarized the comparison in Table I.

B. Few-shot learning Methods

Few-shot learning (FSL) is a machine learning task aims at effectively classifying or recognizing test data from a target domain with very few labeled samples. Most of the training data distribution differs from the test data distribution. Common strategies include data augmentation, optimization-based, and metric-based approaches. Data augmentation methods like Mixup [47] generate new training samples through linear interpolation between different training samples, enhancing diversity and model generalization. Optimization-based methods like MAML (Model-Agnostic Meta-Learning) [6] enable the model to quickly fine-tune using limited data from the target domain through meta-learning on multiple tasks. Metric-based approaches focus on learning an embedding function that maps the input space (e.g., images) to a new embedding space, where a similarity metric distinguishes different classes. Typical works include Siamese networks [4], matching networks [33], prototypical networks (PN) [28], and relation networks [9].

In our work, we aim for the ability to directly provide a few samples (shots) from users without the need for retraining the model, achieving fast deployment and real-time requirements. Considering this goal, we mainly focus on metric-based optimization approaches, particularly the prototypical network method which performs well in few-shot learning. Furthermore, to prioritize high real-time performance, we avoid overly complex model structures. Complex meta-learning schemes may increase computational complexity and inference time, while our work emphasizes usability and efficiency in real-world scenarios. Although some FSL methods have been applied to gesture recognition such as OneFi [43] and WiGr [49], we want the model to be used directly without fine-tuning, so we don’t adopt the method in [43]. The dual-path PN structure in WiGr [49] is somewhat complex, and it only addresses the cross-domain problem without changing the recognition class, i.e., does not address our cross-task problem and the possible bias of prototypes.

In a summary, works closely related to ours are EchoWrite [42], EchoWrite2.0 [51], and DHGR [46]. Compared to EchoWrite and EchoWrite2.0, our signal collection method is
similar, but we enable users to directly write in a complete and natural manner without requiring them to memorize additional rules for simple gestures and concatenation. Additionally, we specifically focus on practical few-shot scenarios, using a prototypical network framework instead of manually extracting features followed by machine learning methods or simply applying CNN networks. By leveraging the prototypical network structure along with our proposed feature transformation layer, we achieve effective recognition with only a few samples provided by users, without the need for retraining the model. Furthermore, we can recognize new gestures and different environments without these data appearing in the training set. In comparison to DHGR [46], their approach requires a Doppler radar as an auxiliary device, whereas we are device-free, relying only on smart devices equipped with speakers and microphones. Moreover, their work uses a basic weighted relation network as the classification model, while we introduce the feature transformation layer and a training approach for its hyper-parameters. Our work does not have the limitation of DHGR [46] that the number of training classes needs to be the same as test classes. Our model with the feature transformation layer exhibits stronger generalization capabilities across different environments and device orientations, which is the key to enhancing performance in our approach.

### III. Preliminaries

We introduce some background knowledge including the relationship between Doppler effect and gesture recognition, few-shot classification, and the meaning of cross-domain.

#### A. Doppler frequency shift and its application

When users perform airwriting gestures near the microphone, the received sound wave signals are composed of three parts: the directly transmitted sound waves from the speaker, the sound waves emitted by the speaker and reflected by the hand before reaching the microphone, and the sound waves reflected by the surrounding environment, such as walls, before reaching the microphone. Considering that the environment at close proximity remains relatively stable during the gesture recognition process, we can assume that the sound waves received by the microphone from both the directly transmitted sound waves and the environment-reflected sound waves are essentially consistent with the source frequency. As for the sound waves reflected by the hand, when the hand moves relative to the wave source with a velocity \( v_h \) and the sound waves propagate in the air at a speed \( v \), and the frequency of the sound waves emitted by the speaker as the wave source is \( f_s \), the frequency of the sound waves received by the hand as the receiver can be described by Equation 1.

\[
    f_h = \left( \frac{v \pm v_h}{v} \right) \times f_s \quad (1)
\]

The operator \( \pm \) is determined based on the direction of relative motion. The sound waves received by the microphone, which are reflected by hand movement, can be considered as emitted by the hand. In this case, when the sound waves propagate to the microphone, the frequency of the received sound waves can be described by Equation 2.

\[
    f = \left( \frac{v}{v \mp v_h} \right) \times f_h \quad (2)
\]

By substituting Equation 1 into Equation 2, we can obtain the frequency difference \( f \) generated by hand movement, as shown in Equation 3.

\[
    \Delta f = |f - f_s| = \left( \frac{2v_h}{v \mp v_h} \right) \times f_s \quad (3)
\]

#### B. Few-shot classification and metric-based method

Few-shot classification is commonly represented as \( N_w \)-way (number of classes) and \( N_s \)-shot (number of labeled examples per class, often denoted as \( K \)). Metric-based algorithms typically consist of a feature encoder \( E \) and a metric function \( M \). During each iteration in the training phase, the algorithm randomly selects \( N_q \) classes and constructs a task \( T \). Denote the input image set as \( X = \{x_1, x_2, ..., x_n\} \), and their corresponding class labels as \( Y = \{y_1, y_2, ..., y_n\} \). Task \( T \) consists of a support set \( S = \{ (X_s, Y_s) \} \) and a query set \( Q = \{ (X_q, Y_q) \} \). For each \( N_w \) class, \( n \) samples are randomly chosen to form the support set \( S \), and \( N_q \) samples to form the

### TABLE I

<table>
<thead>
<tr>
<th>Work</th>
<th>Extra Hardware Required</th>
<th>Recognition Content - Accuracy</th>
<th>Unseen Gesture Recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UbiWriter [45]</td>
<td>No</td>
<td>26 letters – 69.23% 91.8%</td>
<td>No</td>
</tr>
<tr>
<td>RobuCIR [38]</td>
<td>No</td>
<td>15 gestures – 98.4%</td>
<td>No</td>
</tr>
<tr>
<td>Echowrite2.0 [51]</td>
<td>No</td>
<td>26 letters – 73.2% 85.3%</td>
<td>No</td>
</tr>
<tr>
<td>DHGR [46]</td>
<td>Yes</td>
<td>10 gestures (1-shot) – 88.99%</td>
<td>Yes</td>
</tr>
<tr>
<td>AO-Finger [44]</td>
<td>Yes</td>
<td>4 action types – 94.83%</td>
<td>No</td>
</tr>
</tbody>
</table>

**EchoGest (Ours)** No 26 letters (cross-environment, 2-shot) – 88.9% 10 digits (cross-task, 2-shot) – 93.7% 8 gestures (cross-task, 2-shot) – 95.8%
Fig. 3. An overview of the architecture of EchoGest. The system comprises data collection, data processing, feature similarity measurement and classification. In Sec. IV, we briefly introduce the basic structure of the feature extractor $E_{\theta_e, \theta_f}$. Details about training $\theta_f$ parameters and feature transformation will be introduced in Sec. V.

query set $Q$. The feature encoder $E$ first extracts features from both support and query images. Then, the metric function $M$ predicts the class of query image $X_q$ based on the labels of support images $Y_s$, the encoded query image $E(X_q)$, and the encoded support image $E(X_s)$, which can be represented as:

$$\hat{Y}_q = M(Y_s, E(X_q), E(X_s))$$  \hspace{1cm} (4)

Finally, the training objective of the metric-based framework is the classification loss of query images in the query set, denoted as:

$$L = L_{cls}(\hat{Y}_q, Y_q)$$  \hspace{1cm} (5)

We use the prototypical network (PN) [28] as the basic framework, and the metric function $m$ can use the Euclidean distance to measure similarity of vectors. The major advantage of applying the few-shot strategy is that the model does not require retraining, and it helps to avoid overfitting more effectively.

C. Cross-domain recognition

Our target includes both cross recognition categories and cross environment scenarios. In fact, few-shot problems usually involve cross recognition categories. For example, the common minilImagenet dataset [27] follows a benchmark of training with 64 classes and testing with 16 different classes. In our case, cross recognition categories refer to training data consisting of 26 uppercase letters and testing data including 10 digits and some other extended gestures. Additionally, cross environment issues are frequently encountered in the Internet of Things field. Here, the term ‘environment’ can encompass a broader range of settings, such as the orientation of our device (horizontal or vertical), its placement on a table or held upright by hand, the level of environmental noise, and variations in device types, among others. For example, if we train the model with the device placed flat on a table but test it while being handheld upright, the considerable differences in spectral representations due to physical changes would lead to lower classification accuracy using conventional classifiers. In such cases, few-shot methods would show better performance.

IV. System design

In this section, we present our designed in-air gesture recognition system, EchoGest, including overview of architecture, data collection, extract doppler profile, and the classifier backbone based on the prototypical network.

A. Overview of architecture

Fig. 3 presents the overview of our system design, which mainly includes four stages: Data Collection, Data Processing, Similarity measure, and Classification. Specifically, the mobile device’s speaker emits high-frequency signals to capture hand movements, while the microphone receives the reflected echoes from the hand. After filtering out-of-band noise, the time-domain signal sequence is transformed into a spectrogram through Short-Time Fourier Transform (STFT). Then the final gesture feature map is obtained through image binarization and median filtering. For feature map recognition, feature extractor $E_{\theta_e, \theta_f}$ removes the fully connected layer of ResNet18, which serves as the backbone. Here, $\theta_e$ represents all convolutional layer-related parameters of ResNet18, and $\theta_f$ represents the $\theta_e$ and $\theta_\beta$ parameters required for the integrated feature transformation layer. $E_{\theta_e, \theta_f}$ will obtain fixed parameters through a two-stage training process. The input query image $X_q$ goes through the feature extractor $E_{\theta_e, \theta_f}$, resulting in a feature vector. The final Classification is performed by comparing the query feature vector’s $\ell_1$ distance with each gesture support set feature vector, selecting the class with the smallest distance, i.e., the most similar vector, as the classification result.

We will discuss Data Collection in Sec. IV-B, Data Processing in Sec. IV-C, and introduce the basic prototypical network-based feature extractor and classifier in Sec. IV-D. In Sec. V, we will further present our feature transformation layer and the final prototypical network model structure used for classification after incorporating the feature-wise linear transformation layer.

B. Sensing signal and data collection

To meet the requirements of device-free and privacy protection, we utilize 19 kHz ultrasonic signals (since some devices only support up to 20 kHz) to avoid affecting human ears and potential interference. The sampling rate is set at 44.1 kHz. In this study, we do not focus on accurately segmenting the motion in the signals. During the construction of the training dataset, participants are instructed to perform the
gestures at specified intervals, and each data recording lasted for 2 seconds. Similarly, in the testing phase, users are asked to perform a gesture within a 2-second timeframe. Detailed guidelines for these gestures will be presented in Sec. V.

Fig. 4 shows a frame-by-frame illustration of performing the gesture ‘0’ in in-air writing. The upper part demonstrates the process of our writing, while the lower part exhibits the microphone-received sound waves with frequency shifts in both time and frequency domains, visualized using the audacity\(^1\) audio processing software. In Sec. IV-C, we will provide a detailed explanation of how we process and obtain the spectrogram and gesture feature maps to make them suitable inputs for the deep learning framework.

C. Extracting Doppler profile

Since the speed of sound propagation in the air is approximately 346 m/s, Soundwave [8] demonstrated that the finger movement speed \(v_h\) for gesture writing is around 0.2 m/s to 3.5 m/s. With the speaker and microphone sampling rate set at 44.1 kHz, and the speaker’s emitted sound wave frequency \(f_s\) at 19 kHz, we can use equation 3 to calculate the received frequency difference \(\Delta f\) to be approximately between 20 Hz to 300 Hz. Therefore, our processing steps are illustrated in Fig. 5: first, we apply a 6th-order Butterworth bandpass filter to retain the signal within the range of \([18700, 19300]\) Hz. Next, a 3rd-order Butterworth bandstop filter is used to remove the frequency band near the center frequency \([18980, 19020]\) Hz for further signal enhancement. Afterward, we use STFT to transform the one-dimensional signal into a two-dimensional spectrogram, revealing the Doppler frequency shift caused by the gestures.

We execute STFT on each signal sequence with a frame length of 8192 data points (corresponding to 0.186s) and an overlap length of 7168 data points (corresponding to 0.163s). The determination of these two key parameters involves a trade-off between time and frequency resolution, providing a frequency resolution of 5.38 Hz and a time resolution of 23 ms.

We restrict the frequency range to \([18700, 19300]\) Hz and save it as the spectrogram. An example of writing is shown in Fig. 6(a), which presents the spectrogram of letter ‘A’ captured by participant \(p1\) in the laboratory environment using a Samsung Galaxy Tab S2 device, while Fig. 6(d) shows the same content but recorded using a Xiaomi Tab Mix2 device with other settings remaining the same. Although there is a common frequency shift pattern in the writing, different settings still have a significant impact on the generated spectrogram. Therefore, we use a binarization method to eliminate background interference as much as possible. We directly erase the background with no information at the top and bottom of the spectrogram using two rectangular boxes and then perform binarization using the common OTSU method [18] to obtain the binarized result as shown in Fig. 6(b). Finally, to remove noise, we apply the median blur method (medianBlur\(^2\)) to obtain the result as shown in Fig. 6(c). In this way, the binarization process may cause the loss of some depth information in the image. We also attempt to retain the depth information by performing a bitwise AND operation between the binarized data and the original image, preserving the RGB depth of the pixels with information after binarization, as shown in Fig. 6(e) and 6(f).

Fig. 7 displays some spectrograms obtained after processing the input signals. Fig. 7(a), 7(b), and 7(c) represent the digit ‘0’, digit ‘1’, and the letter ‘X’ written by participant

\(^1\)https://www.audacityteam.org
\(^2\)https://docs.opencv.org/4.x/dc/d66/group__cudafilters.html

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on June 20, 2024 at 15:36:39 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
where \( k \) represents the true class of sample \( x_i \), and \( \theta \) denotes the learnable parameters of the prototypical network. In our implementation, we adopt a prototypical network architecture based on ResNet18 as the backbone. We modify its first convolutional layer from a \( 7 \times 7 \) kernel to a \( 5 \times 5 \) kernel (as we observe improved feature extraction with this change) and remove the final fully connected layer of ResNet18. Apart from these modifications, the basic backbone remains unchanged.

Regarding the learnable parameters, a regular prototypical network undergoes a single global-stage training, where \( \theta \) represents all the convolutional layer parameters in ResNet18. To enhance cross-domain recognition accuracy, we design a feature transformation layer to enable more generalized feature extraction. This feature transformation layer has hyper-parameters denoted as \( \theta_f \), while the other original convolutional layer parameters are denoted as \( \theta_c \). The training process involves two stages: first, we pretrain \( \theta_c \), and then we use data with different distributions from the pre-training data to train and obtain the hyper-parameters \( \theta_f \). Once both stages of training are completed, all model parameters are fixed. In Sec. V, we will provide a detailed explanation of the design of our feature transformation layer and how we obtain \( \theta_f \).

V. IMPROVED METHODOLOGY

In this section, we present the feature transformation layer designed to enhance the model’s generalization ability during inference in different environments. We also explore the architecture of the prototypical network classifier with the integration of the feature transformation layer. Additionally, we introduce a two-stage training process in order to obtain hyper-parameters of the feature transformation layer. Furthermore, we briefly describe the extension of updating prototypes using clustering methods.

A. Feature-wise linear transformation layer

Due to variations in feature distributions across different domains in the task, features obtained from the task might overfit to the training dataset and struggle to generalize well to other domains. To address this, we integrate a feature transformation layer into the feature encoder \( E \) to enhance intermediate feature activations with affine transformations. This intuitively generates more diverse feature distributions, capable of meeting the requirements of tasks like gesture spectrogram analysis, which is not inherently complex. Our feature transformation layer’s specific structure is illustrated in Fig. 8, representing a linear transformation applied to the feature maps obtained from the \( l \)th layer of the network. Given the intermediate feature activation map \( x \) of dimensions \( C \times H \times W \) from the feature encoder, a linear feature transformation is performed using hyper-parameters \( \theta_{\gamma} \) of size \( C \times 1 \times 1 \), and \( \theta_{\beta} \) of size \( C \times 1 \times 1 \):

\[
\hat{x}_{c\times h\times w} = \theta_{\gamma} \times x_{c\times h\times w} + \theta_{\beta}
\]

Fig. 9 provides a more intuitive illustration of the effect of the linear modulation. In practice, we apply the feature transformation layer only after the batch normalization of the
last convolutional layer in ResNet18. This adds a mere 1024 additional hyper-parameters, which is negligible compared to the total number of convolutional parameters. The concept and implementation are initially introduced in the work of FiLM [20]. Subsequent works such as CNAPs [26], SimpleCNAPs [3], and the work [30] are influenced by the FiLM layer. The primary distinction of our work from these is how the hyper-parameters $\theta_{\gamma}$ and $\theta_{\beta}$ are trained and determined.

Fig. 8. Feature-wise linear transformation layer

In contrast to CNAPs and SimpleCNAPs which draw inspiration from the Conditional Neural Process (CNP) [7] concept, their $\theta_{\gamma}$ and $\theta_{\beta}$ required by FiLM necessitate an additional complex encoder. Their encoder takes the images from the support set as input and utilizes convolutional and fully connected layers to compute $\theta_{\gamma}$ and $\theta_{\beta}$. As a consequence, they are not deterministic, and the intricate encoder also needs to be deployed on mobile devices, incurring significant training, storage, and computational overhead. In comparison to the work [30], where $\theta_{\gamma}$ and $\theta_{\beta}$ follow a Gaussian distribution rather than specific values, training involves straightforward computation of the loss for $\theta_{\gamma}$ and $\theta_{\beta}$, updated through gradient descent. Although the work [30] shares a similar methodology with ours, in our experiments, we observe that directly updating $\theta_{\gamma}$ and $\theta_{\beta}$ through gradient descent yielded suboptimal results during testing. To address this, we train a Multi-Layer Perceptron (MLP) to map randomly initialized values to optimal $\theta_{\gamma}$ and $\theta_{\beta}$, leading to improved performance during testing. Fig. 10 shows the model architecture after integrating the feature transformation layer into the base feature extractor. It is evident that the incorporation of this feature transformation layer introduces minimal additional parameters and computational complexity.

### B. Training parameters with a two-stage process

Formally, our dataset is divided into three parts: 1) Pre-training data $P'$, used to optimize the $\theta_{\gamma}$ parameters in the feature extractor, akin to training a standard prototypical network; 2) Meta-training data $S'$, employed to optimize the $\theta_{f}$ parameters in the feature extractor. In practice, post the pre-training of the conventional prototypical network, we employ the $S'$ dataset to train an MLP capable of mapping the initially randomly initialized $\theta_{\gamma}$ and $\theta_{\beta}$ parameters to their optimal counterparts; 3) Testing data $T'$, for evaluating the model’s recognition accuracy across various scenarios. In essence, our pre-training data consists of letter samples written on a Samsung tablet in a vertical orientation (0°), the meta-training data involves letters rotated to a horizontal orientation (90°), and the testing data encompass diverse settings such as digit categories, noisy environments, and varying hand poses.

Our MLP architecture, as illustrated in Fig. 11, consists of three fully connected layers combined with the ELU (Exponential Linear Unit) activation function. In contrast to ReLU, ELU returns a value close to zero for negative inputs, avoiding zero itself, while preserving similar properties to ReLU in the positive range. The input, a randomly initialized but fixed vector $z$, a 128-dimensional vector, undergoes the three-layer MLP, to produce $z'$, a 1024-dimensional vector. Here, $\theta_{f} = \{\theta_{\gamma}, \theta_{\beta}\}$, where both $\theta_{\gamma}$ and $\theta_{\beta}$ are $512$-dimensional vectors, serving as hyper-parameters for the feature transformation layer within the feature extractor. The specific training process is divided into two stages: the initial phase, termed pre-training, focuses on optimizing $\theta_{f}$ parameters; the second phase, referred to as meta-training, involves training the MLP to obtain the desired $\theta_{f} = \{\theta_{\gamma}, \theta_{\beta}\}$ parameters. The training procedure is outlined in Algorithm 1 for clarity.

Upon completing the optimization of the MLP, the parameters $\theta_{f} = \{\theta_{\gamma}, \theta_{\beta}\}$ are also determined, finalizing the optimization of $E_{\theta_{f}}$. Concerning the loss function $J(\phi)$ during the meta-training phase, we introduce an $\ell_{2}$ regularization term to constrain the values of $\theta_{f}$ and $\theta_{\gamma}$, where $\alpha$ is set to 0.001. Without regularization, the training performance is notably poor. Additionally, we experiment with the training approach proposed in the work [30]: $\theta_{f}^{t+1} = \theta_{f}^{t} - \alpha \nabla_{\theta_{f}} J(\phi)_{\text{cls}}$, where $t$ represents a training step. However, training the feature transformation layer in this manner doesn’t yield the desired outcomes; the accuracy decrease when dealing with cross-domain scenarios. We will continue our exploration in subsequent endeavors, investigating the distinctions between such training techniques and the introduction of an MLP for

---

**Fig. 10. Overall Model structure integrated with Feature-wise linear transformation Layer (3-51, only 1024 Param #).** The corresponding number of channels $C$ is 512.
Algorithm 1 Learning network parameters with feature-wise linear transformation layer. Parameters \( \theta_e \) in feature extractor \( E \) remain fixed during Meta-training.

**Input:** Pre-training data \( P' \), Meta-training data \( S' \), randomly determined vector \( z \)

**Output:** Optimized parameters \( \theta_e, \theta_f \)

1: Initialize \( \theta_e \) and \( \theta_f = \{ \theta_1, \theta_2 \} \)

2: while Pre-training do

3: \( J(\theta_e) = 0 \) //Training parameters \( \theta_e \)

4: Select support set \( S_r \) and query examples from \( P' \)

5: Compute prototype: \( P_r = \{ \frac{1}{|S_r|} \sum_{(x_i, y_i) \in S_r} E_{\theta}(x_i) \} \)

6: Query \( x_i \): \( p(y = r|x_i) = \frac{\exp(-d(E_{\theta_{\gamma}}(x_i), P_r))}{\sum_{r'} \exp(-d(E_{\theta_{\gamma}}(x_i), P_{r'}))} \)

7: Update \( \theta_e \) by \( J(\theta_e) = -\log p(y = k|x_i) \)

8: end while

9: while Meta-training do

10: \( J(\phi) = 0 \) //Training parameters \( \phi \) to get \( \theta_f \)

11: Select support set \( S_r \) and query examples from \( S' \)

12: Get hyper-parameters: \( \theta_f = \{ \theta_1, \theta_2 \} = MLP_{\phi}(z) \)

13: Compute prototype: \( P_r = \{ \frac{1}{|S_r|} \sum_{(x_i, y_i) \in S_r} E_{\theta_{\gamma}}(x_i) \} \)

14: Query \( x_i \): \( p(y = r|x_i) = \frac{\exp(-d(E_{\theta_{\gamma}}(x_i), P_r))}{\sum_{r'} \exp(-d(E_{\theta_{\gamma}}(x_i), P_{r'}))} \)

15: Update \( \phi \) by \( J(\phi) = -\log p(y = k|x_i) + \alpha (\| \theta_1 \|_2^2 + \| \theta_2 \|_2^2) \) //With \( \ell_2 \) regularization term

16: end while

C. Updating prototypes with user data

Given that users’ writing habits and writing environments may change over time, a more intuitive solution is to collect the user’s own unlabeled data to update the corresponding gesture prototypes. This section represents an extension we have made in the algorithm. The motivation behind this lies in the fact that using only 2 or 3 data points as prototypes may not accurately represent an entire class. Our aim is to update prototypes using real user data. Inspired by Semi-supervised Prototypical Networks [25], consider this scenario:

if we can collect unlabeled data from each user’s writing process (this only requires user authorization and saving the feature maps of the writing process, without any additional steps), through semi-supervised clustering methods like \( K \)-means, each prototype is initially treated as a cluster center. The refinement process adjusts the cluster centers’ positions through clustering, ensuring that the final cluster centers better match the true data distribution. In paper [25], semi-supervised clustering employs soft \( K \)-means as it requires differentiability during the training process. In contrast, we directly use hard \( K \)-means to keep the process as simple as possible. We don’t introduce clustering operations during training; instead, we update prototypes using \( K \)-means with unlabeled data only when necessary for classification. Our subsequent tests indicate that hard \( K \)-means enhances accuracy. This implies that the classification testing process is equivalent to classifying samples within the final clustering clusters. The process of updating prototypes using \( K \)-means can be summarized by the formula 10, where \( U_r \) denotes the set of unlabeled data points assigned to class \( r \).

\[
\tilde{P}_r = \frac{1}{|S_r|} \sum_{(x_i, y_i) \in S_r} E_{\theta}(x_i) + \frac{1}{|U_r|} \sum_{(x_j, y_j) \in U_r} E_{\theta}(x_j)
\] (10)

Fig. 12 provides an illustrative explanation of our approach. The initial prototype’s classification boundaries can be modified after undergoing clustering. The more unlabeled data a user contributes, the more closely the cluster centers should align with the true center positions, resulting in more precise classification boundaries. Currently, we have only experimented with this algorithm extension on a dataset. In the future, it could potentially be extended to the application, allowing each user to update their own prototypes using their unlabeled data, thereby achieving personalized user objects.

VI. IMPLEMENTATION AND EVALUATION

A. Experiment setup

To construct datasets, we initially develop an Android mobile application as a data collection tool. This application control the speaker in the mobile device to emit a 19 kHz
sinusoidal modulated audio signal. Simultaneously, the microphone within the same device is controlled to receive echoes reflected from writing fingers and other nearby objects at a sampling rate of 44.1 kHz. It’s worth noting that, although we set amplitude of the emitted audio to the maximum in our software, adjusting system’s media volume still affects the power of the emitted sound waves. Due to variations in hardware across different devices, we take the example of the speaker configuration on the Samsung Galaxy S9 phone. In a 35 dB environment measured using a sound level meter, when the media volume on the phone is set to 1/3, 2/3, and 1, the sound level at a distance of 10 cm from the speaker reads 39 dB, 49 dB, and 62dB, respectively (these high-frequency sound waves are inaudible to the human ear). We observe that when the media volume is set to less than 1/3, the resulting spectrograms appear disordered. On the other hand, when the volume is set higher than 2/3, the spectrograms become consistent and are no longer influenced by the volume, as shown earlier. For the subsequent experiments, we set the volume to the maximum.

For our experiments, we recruit ten volunteers from the school (6 males, 4 females) between the ages of 18 and 25. Subsequently, as part of the extended experiments detailed later, we recruit three additional on-campus volunteers (3 males, aged 18-25) to write on different devices and at various distances. Additionally, we recruit three special volunteers aged 9, 35, and 59 to illustrate that our system can be used stably by individuals of different ages, including children and the elderly. This is especially significant when considering that these groups may exhibit smaller frequency shifts due to fine-grained finger movements or slower writing speeds.

Our handwritten gestures primarily consist of three types: digit gestures from ‘0’ to ‘9’, uppercase letter gestures from ‘A’ to ‘Z’, and an additional set of 8 gestures commonly used in human-computer interaction applications [50]. For the ‘0’ to ‘9’ digit gestures, we don’t impose any specific writing standards, allowing volunteers to write in their accustomed manner. Regarding the ‘A’ to ‘Z’ uppercase letters, due to the greater variety of characters, we provide slightly more specific writing guidelines as illustrated in Fig. 13. The 8 common gestures and their associated content will be discussed in Sec. VI-D. In the basic experiments, 10 volunteers are required to write on Samsung tablet devices under various conditions, including different writing angles, noise interference levels, and postures. The extended experiments also encompass assessments involving individuals of varying age groups, different writing gestures, distinct devices, and varying distances from the device.

Regarding the conditions considered in our experiments:

- **Angle:** This denotes the relative orientation of the device and the writing area. 0° indicates that the device is in a vertical position, with the writing area under. 90° represents a horizontal device orientation, with the writing area on the right side of the device.

![Fig. 13. Some writing norms of uppercase English letters.](image)


- **Environment:** An office setting with noise levels ranging from 35-50 dB, and a relaxation area, which is open and exhibits noise levels exceeding 50 dB.
- **Posture:** This refers to the method of device usage, either placed flat on a tabletop (on-table) or held in the hand (in-hand). In the in-hand condition, we practically employ a smartphone stand to simulate the upright posture as if the device is held in hand.

We divide the collected data into the required training and testing sets as described below:

- **Pre-training:** Using Samsung tablets at 0° in a lab environment, users wrote ‘A’-‘Z’ letters placed flat on the table. There are 10 volunteers, each writing each letter 20 times, resulting in 5200 samples (26 letters × 20 repetitions × 10 participants).
- **Meta-training:** Similar to Pre-training, but the orientation is changed to 90°, resulting in another 5200 samples.
- **Meta-validation:** Using Pre-training dataset for meta-validation.
- **Testing:** In the basic testing phase, each condition (angle, environment, posture) is tested with 10 volunteers for ‘0’-‘9’ digits, resulting in 2000 testing samples for each condition (10 digits × 20 repetitions × 10 participants). Typically, the basic testing utilizes the 0° orientation, lab environment and on-table. Further details regarding testing data will be explained in the corresponding chapters.

The models and associated parameters we employ have been comprehensively detailed in previous sections and Fig. 10. All our experiments are conducted using PyTorch 1.8.1 on a remote server equipped with a NVIDIA RTX A6000 GPU featuring CUDA 11.1 framework, 48GB of memory, and an Intel Xeon E5-2686 v4 2.30GHz CPU processor for evaluation. We utilize the Adam optimizer [10], and the initial learning rate for all methods is consistent at 5 × 10⁻⁴ (with potential variations in methods like fine-tuning). The activation function employ uniformly across all methods is the Rectified Linear Unit (ReLU).

### B. Overall performance

In terms of overall performance, we employ average recognition accuracy as the key metric. We evaluate recognition accuracy across different random seeds, various handwritten gestures, and participants. Accuracy is defined as \( \frac{N_{\text{correct}}}{N_{\text{total}}} \), where \( N_{\text{correct}} \) represents the number of correctly recognized samples, and \( N_{\text{total}} \) represents the total number of test samples. We conduct evaluation experiments with three different random seeds.
We evaluate the accuracy of ‘0’ to ‘9’ digit gestures recognition in the Testing dataset with conditions of 0°, lab, and on-table. This basic test is cross-task but not cross-environment. When not specified, it will serve as our benchmark test result. Fig. 14 illustrates the confusion matrix obtained by testing the support set with a 2-shot per user directly from the Testing dataset. Since each user has 20 samples for each gesture, during testing, we randomly draw the support set 10 times, using the remaining data as the query set for testing (the sampling process follows 1). This approach serves as our default testing method. Despite the spectral similarities between digits ‘0’ and ‘6’ and ‘1’ and ‘7’, the confusion matrix does show some misclassifications of these gestures. Nevertheless, our system, EchoGest, still achieves an average recognition accuracy of 93.7%.

![Confusion Matrix](image)

**Fig. 14.** The confusion matrix of EchoGest for recognizing unseen gestures with 2-shot.

Fig. 15 displays the results of cross-user testing on ten volunteers using a leave-one-user-out strategy. In this approach, data from the individual being tested only appears in the test set and is absent from the Pre-training and Meta-training datasets. For this test, we continue to provide a 2-shot support set per individual. Our cross-user testing yields an impressive recognition accuracy of 93.2%, affirming the effectiveness of our proposed method. It is suitable for real-world cross-user scenarios without the need for training on user-specific data. Recognition is possible with a small number of shot samples.

![Recognition Accuracy](image)

**Fig. 15.** Recognition accuracies of EchoGest obtained for different persons using leave-one-user-out cross validation.

In cross-domain experiments involving factors like angle, environment, and posture, we conduct further evaluations of our system’s performance. For each cross-domain experiment, only one domain factor is altered, with the baseline being 0°, a quiet laboratory environment, and the device placed flat on a desk. We use three different random seeds: 1, 42, and 100. Under each seed, ten different support sets are extracted for all testing datasets. The average accuracy across all of these is calculated to provide an objective evaluation result.

![Recognition Accuracy](image)

**Fig. 16.** Recognition accuracy varies with number of 10 testing gestures. In 2-shot cases, EchoGest’s accuracy exceeds 90% in various testing scenarios, with accuracy rates of 93.7%, 90.5%, 91.0%, and 94.7%. The performance even surpasses the baseline when the phone is placed upright, which may be due to the closer microphone, suggesting that a substantial amount of training data is required to achieve effective recognition.

In our main experimental tests, which displays the accuracy results for our system when the device is rotated 90° horizontally, in a noisier environment (>50 dB), and when the system is used handheld (simulated by placing the phone on a stand). 'wh/o’ indicates scenarios where our feature-wise linear transformation layer is not used, only with basic PN classification. It is shown that in 2-shot cases, EchoGest’s accuracy exceeds 90% in various testing scenarios, with accuracy rates of 93.7%, 90.5%, 91.0%, and 94.7%. The performance even surpasses the baseline when the phone is placed upright, which may be due to the closer microphone-
to-handwriting distance when the phone is upright, resulting in more noticeable frequency shifts.

When the phone is placed flat on the desk at a 90° angle, our feature transformation layer brings about the most significant improvement, with only a slight increase in parameters leading to an average accuracy improvement of 8%. Under all 3-shot conditions, our accuracy consistently exceeds 94% in various cross-domain tests, maintaining high recognition precision. Results also show that our approach has a significant advantage when users only provide 1-shot: the accuracy improvement is nearly 10% in all cross-domain cases. Adding only a very small number of parameters can result in a huge increase in accuracy. It’s worth noting that our testing phase directly employs the most basic few-shot N-way K-shot testing. In practice, the ability to select template shots could further enhance accuracy, as random shot selection may include some low-quality data. More detailed comparative and ablation experiments are presented in Sec. VI-C.

Regarding EchoGest’s real-time running performance, Table II presents the time required for our system to process data from the user’s input to displaying the classification results. During our practical testing, we discovered that sending data to the server for inference execution is more time-efficient than directly utilizing mobile deep learning frameworks on the client side. Consequently, we continue to employ a server-client architecture, even though this incurs some additional network traffic. We plan to optimize this aspect as part of our future work. What’s more, this approach offers another two advantages. First, it enables the application to be used on smartwatches since they are currently unable to execute complex PyTorch inferences. Second, it eliminates the need for users to locally store the required .pth or .onnx model for classification, thereby reducing local storage usage.

Table II presents the total time for testing 10 digit gestures on different terminal devices acting as clients and various devices serving as servers, all under a 50 Mbps network speed. For terminal devices, we utilize both smartphones and smartwatches, while for servers, we experiment with a laptop (ROG Zephyrus M16, with a NVIDIA RTX 3060 GPU, 16GB of memory, and i7-11800H CPU) and a large server as described in Sec. VI-A. It is observed that, for smartphone with laptop serving as a server, the data upload and storage time on the server is approximately 0.03 seconds, the time taken to convert to a spectrogram on the server averages 0.24 seconds, and the model inference time is only 0.1 seconds. Overall, the total response time is less than 0.4 seconds which is sufficient for daily interaction needs. When a smartwatch serves as user’s device, the transmission time increases slightly, but the total required time remains below 0.5 seconds. Additionally, we conduct separate tests on the last layer of the prototypical network using ℓ₁ and ℓ₂ distances. Their accuracies are quite similar, but the ℓ₁ distance is more time-efficient and computationally faster. Therefore, we opt for the ℓ₁ distance to achieve faster inference speeds.

We also assess the usage of CPU and memory with the use of our application. During the evaluation process, we employ a Samsung Galaxy S9 phone, monitor CPU and memory usage in USB debugging mode via Android Studio. Fig. 19 shows our CPU and memory usage. The testing duration is 15 seconds, during which we perform three writing activities. The maximum memory and CPU usage observed are 335.6 MB and 25.4%, respectively, with low CPU usage when there is no activity.

In terms of power consumption, we use the same device to monitor power consumption of our application, within one hour. For comparison, we also provide power consumption when the application is not in use and when only a music player continuously plays music. The phone has a 3000 mAh battery, and when not using it, its screen remains on, and Wi-Fi is enabled. When using our gesture application, we made certain adjustments in the software to emit a continuous 19 kHz sound wave without the need for direct interaction. We set it to emit the maximum volume sound wave for 2 minutes every 5 minutes, indicating that users spend 40% of the time in continuous writing, which is a substantial portion. For the comparison with music player, we use earphones (which consume less power than external speakers) and control the volume at one-third, simulating practical usage. Fig. 20 shows our power consumption results. After one hour, the phone’s

---

**Table II**

<table>
<thead>
<tr>
<th>Item</th>
<th>mobile devices - servers</th>
<th>Transmission</th>
<th>Spectrogram</th>
<th>Inference</th>
<th>Feedback</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone</td>
<td>laptop server</td>
<td>0.027</td>
<td>0.239</td>
<td>0.101</td>
<td>&lt;0.001</td>
<td>0.367</td>
</tr>
<tr>
<td></td>
<td>server cluster</td>
<td>0.023</td>
<td>0.289</td>
<td>0.071</td>
<td>&lt;0.001</td>
<td>0.383</td>
</tr>
<tr>
<td>watch</td>
<td>laptop server</td>
<td>0.068</td>
<td>0.238</td>
<td>0.103</td>
<td>&lt;0.001</td>
<td>0.409</td>
</tr>
<tr>
<td></td>
<td>server cluster</td>
<td>0.097</td>
<td>0.287</td>
<td>0.069</td>
<td>&lt;0.001</td>
<td>0.453</td>
</tr>
</tbody>
</table>

---

**Fig. 18**. Accuracies of cross-domain evaluation. We show the performance difference with(w) and without(w/o) feature-wise linear transformation layer.

**Fig. 19**. The CPU and memory usage under different usage states.

**Fig. 20**. The power consumption under different usage states.
battery decreases to 95%, 74%, and 82% from 100% in these respective scenarios. In actual usage, our software’s power consumption is somewhat similar to continuously playing music on the phone’s speakers and does not significantly deplete the battery, especially when users do not need to write frequently.

C. Comparison and ablation study

Our comparative experiments, as shown in Table III, primarily assessed various aspects of the methods. These assessments included the number of parameters in the method, performance when trained with only 0° letters or both 0° and 90° letters, the 2-shot testing accuracy for 10 digits, and deployability. Deployability in this context, refers to whether users can directly utilize the method without retraining the model. Importantly, Deployability also means that our model does not need to be retrained once it is trained, preventing users from having to incur training costs or time. Fine-tuning methods require users to provide data for retraining. Given that 0° and 90° letters exhibit significant differences in the spectrum, our strategy for fine-tuning, PN [28], RN [9], and other methods that only require one-time training is to initially train with 90° letters. After obtaining this pretrained model, we proceed to further training with 0° letters, effectively utilizing the results from the 90° letter training as an initialization for the model.

The first three compared methods are as follows: Normal training of a ResNet18 with a fully connected layer, followed by removal of the fully connected layer, using convolutional layers as feature extractors, and comparing the obtained feature vectors based on \( \ell_2 \) distance and user-shot; Fine-tuning the modified fully connected layer with user-shot after changing its dimensions; Fine-tuning all the layers of the ResNet18 with user-shot, while reducing the learning rate. Result shows that fine-tuning tends to achieve higher accuracy, surpassing 85%.

These, along with the MAML method [6], RN method [9], will serve as the baselines for comparative experiments, while the PN method [28] will be the baseline for ablation experiments. It is worth noting that the feature extractor structure is ResNet18 for all of them, with possible adjustments in other parts to match the dimensions. PN method [28] achieves the highest accuracy among the baseline methods, with a 2-shot testing accuracy of 88.3% when both 0° and 90° letters are used for training.

‘PN + feature transformation’ is our primary approach. When PN is integrated with our feature-wise linear transformation layer, the accuracy reaches 93.7%, significantly improving accuracy with the addition of very few parameters. The extension part in combination with clustering methods can further enhance accuracy. It’s worth noting that clustering methods require access to all raw unlabeled data, and can only be used when the data distribution is known, which imposes certain limitations.

We also compared our work with some closely related methods. The work [30] is the most relevant method to ours, but our re-implemented accuracy is only 86.7%, indicating that its feature transformation had limited effect. We have provided a detailed analysis of this in Sec. V-B, which also highlights the effectiveness of our two-stage training parameter approach and our \( \ell_2 \) regularization term for \( \theta_f \). DN4 [13] is a method suitable for our image data in the context of few-shot learning. Its idea is to utilize deep local descriptors, which means, instead of applying global average pooling on the ResNet features, it treats each corresponding position vector as a local descriptor. Using the local invariant features from two images for similarity matching, rather than relying solely on their image-level representations, might be more suitable for our data. This is because our writing content is composed of smaller strokes, and local features could better capture this.

### Table III

<table>
<thead>
<tr>
<th>Method</th>
<th>Methodology</th>
<th>No. Parameters</th>
<th>Accuracy (2-shot, 0° digits testing)</th>
<th>Deployability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Feature extractor</td>
<td>transfer learning</td>
<td>44.69M</td>
<td>70.2%</td>
<td>✓</td>
</tr>
<tr>
<td>Fine-tune: FC layer</td>
<td>transfer learning</td>
<td>46.74M</td>
<td>85.8%</td>
<td>✓</td>
</tr>
<tr>
<td>Fine-tune: All layers</td>
<td>transfer learning</td>
<td>46.74M</td>
<td>86.7%</td>
<td>✓</td>
</tr>
<tr>
<td>MAML [6]</td>
<td>meta-learning</td>
<td>46.74M</td>
<td>87.4%</td>
<td>✓</td>
</tr>
<tr>
<td>Prototypical Network (PN) [28]</td>
<td>few-shot learning</td>
<td>44.69M</td>
<td>87.9%</td>
<td>✓</td>
</tr>
<tr>
<td>Relation Network (RN) [9]</td>
<td>few-shot learning</td>
<td>47.77M</td>
<td>82.3%</td>
<td>✓</td>
</tr>
<tr>
<td>PN + feature transformation (our main proposed method)</td>
<td>few-shot learning &amp; feature transformation</td>
<td>44.70M</td>
<td>93.7%</td>
<td>✓</td>
</tr>
<tr>
<td>PN + clustering</td>
<td>Integrated approach</td>
<td>44.70M</td>
<td>91.3%</td>
<td>✓</td>
</tr>
<tr>
<td>PN + feature transformation + clustering</td>
<td>few-shot learning &amp; feature transformation</td>
<td>44.70M</td>
<td>91.2%</td>
<td>✓</td>
</tr>
<tr>
<td>Cross-domain few-shot classification [30]</td>
<td>few-shot learning &amp; feature transformation</td>
<td>44.69M</td>
<td>91.6%</td>
<td>✓</td>
</tr>
<tr>
<td>DN4 [13]</td>
<td>few-shot learning &amp; local descriptor</td>
<td>47.80M</td>
<td>82.8%</td>
<td>✓</td>
</tr>
</tbody>
</table>

* Method 1 ‘Only Feature Extractor’ means that after training a ResNet18 model, we retain only Conv2d layers, removing the fully connected layers used for classification and directly use \( \ell_2 \) distance to measure similarity of feature vectors. Methods 2 and 3 respectively mean that using user’s shots to fine-tune either only the last fully connected layer or fine-tune all ResNet18 layers with a smaller learning rate.
While it does improve accuracy, it comes with a significant increase in computational complexity. The computational cost of the final layer increases by more than 50 times with our $8 \times 8$ feature maps, making it challenging to achieve real-time performance. Finally, we compared our work with DHGR [46]. DHGR [46] builds upon the RN structure and concatenates the Relation Score to obtain the final result through FC layers. When reproducing this method, we encountered a problem: the vector obtained after combining needs to pass through fully connected layers, which require that the number of training and testing shots remain the same and cannot be altered. As a part of our reproduction, we set train-way to 10 instead of 26, which may have caused some decrease in accuracy, resulting in a maximum accuracy of only 83.7%. In contrast, we achieved a 93.7% accuracy in the 10-way 2-shot classification and a 96.0% accuracy in the 10-way 3-shot classification without using additional Doppler radar devices. This demonstrates the practicality of our system, surpassing the performance of DHGR [46], which reported a 10-way-3-shot accuracy of 91.88%.

Fig. 21 and Fig. 22 illustrate the visualization of feature vector distributions obtained using the t-SNE [32] method. In Fig. 21, only PN is used, while Fig. 22 presents the results after incorporating the feature-wise linear transformation layer. The recognition accuracy for the same data improved from 88.5% to 92.5%. The visualization results show that our approach can push the spacing between classes to a greater extent, which is more conducive to classification for prototypical network infrastructures. In this particular set of test data, the device is oriented vertically. Due to this orientation, the Doppler effect results in a more pronounced frequency shift during vertical finger movements compared to horizontal movements. As a result, spectrograms of ‘1’, ‘2’, and ‘7’ appear more similar. Intuitively, our feature transformation also appears to effectively increase the distance between ‘1’ and ‘2’, as well as ‘6’, yielding a noteworthy effect.

D. Performance under other different settings

1) Impact of gestures: We present extended experiments with some variations in the experimental settings. These experiments may have a smaller dataset compared to Sec. VI-B, and we will clarify this when describing the experimental content. First, we conduct tests on 26 letters and 8 new customized gestures. For the 26 letters classes, we use a total of 5200 letters data points previously written by 10 volunteers in an in-hand posture. In this scenario, the test does not involve cross-task testing but does involve cross-environment testing. For another customized gestures, we define 8 customized gestures based on common writing actions in the Wi-Fi field [50], as shown in Fig. 23. These gestures are relatively more distinguishable compared to the set of 10 digit gestures. We collect a total of 320 customized gesture data samples from volunteers $p_2$ and $p_3$. Fig. 24 displays the Top-1 and Top-2 recognition accuracy for the three major categories of gestures under the 2-shot condition. Top-2 denotes the two gestures with the highest classification confidence, including the correct gesture classification. Our Top-2 accuracy for all tests exceeds 98%. For 2-shot tests of digits, letters, and customized gestures, the Top-1 accuracy rates are 93.7%, 88.9%, and 95.8%, respectively.

2) Impact of ages: Next, we conduct tests with participants from different age groups. As mentioned earlier, we note that younger users might have less noticeable frequency shifts due to fine-grained finger movements, while older users might exhibit slower writing speeds, which could also lead to less pronounced frequency shifts. We collect data from three participants representing different age groups (9, 35 and 59 years old), each contributing 100 experimental data. We do not impose any specific restrictions on the devices used for their writing experiments. Fig. 25(a), 25(b), and 25(c) respectively
illustrate their writing conditions, while Fig. 25(d), 25(e), and 25(f) correspond to the frequency spectrum of the handwritten digit ‘0’. These frequency spectrograms are processed into binarized feature maps using the method described in Sec. IV-C and subsequently used for classification. Fig. 26 presents the accuracy results of these tests. It is evident that, although the 1-shot accuracy is relatively lower, the 2-shot accuracy for these age groups reach 89.7%, 86.7%, and 91.3%, respectively. The 3-shot accuracy also consistently exceeds 90%, demonstrating that the accuracy remains high. This highlights the reliability of our system.

3) Impact of devices: We also conduct a specific evaluation of the impact of different devices on the writing process. Our experimental devices include the Samsung Galaxy Tab S2, Samsung Galaxy S9 Phone, and Oppo Watch3 Pro. We recruit two male participants, aged 22, and collect 200 instances of ‘0’-‘9’ digits for each device. Fig. 27(a), 27(b), and 27(c) display the appearance of these three devices, while Fig. 27(d), 27(e), and 27(f) correspond to the frequency spectrum of a handwritten digit ‘0’ obtained with each of these devices. To our surprise, the frequency spectrum associated with the smartwatch is the clearest, exhibiting the most pronounced frequency shifts with a consistent pattern. Although not all smartwatches meet the hardware requirements of emitting and capturing 19 kHz ultrasonic waves, and not all support Android development, our experimental results clearly demonstrate the potential of future smartwatches for touchless writing. Fig. 28 presents the testing accuracy results, with their 2-shot testing accuracy reaching 93.7%, 89.5%, and 93.2% respectively, while the 3-shot testing accuracy consistently surpasses 94%.

It’s worth mentioning that, even though we haven’t provided specific details about the evaluation of image binarization in main text, it remains a crucial component, especially evident in this cross-device testing here and tests involving different age groups mentioned earlier. We conduct tests using spectrograms without binarization as training and testing data. The results, with the only difference being the presence or absence of image preprocessing, shows that 2-shot cross-device recognition accuracy is below 60% without binarization, while exceeds 90% with it. Therefore, image binarization not only aligns with our intuition but also significantly benefits the model’s recognition capabilities.

4) Impact of distances: We assess the impact of different writing distances, as shown in Fig. 29. It’s observed that greater distances result in less noticeable Doppler frequency shifts. In previous experiments, we primarily assume ‘near’ range writing. In this evaluation, we invite five volunteers, p2, p3, p4, p5, and p8, to write a total of 2000 digit gestures at ‘mid’ and ‘far’ distances using the Samsung Galaxy Tab S2 device. The testing results in Fig. 30 indicate that accuracy indeed decreases as the writing distance increases. In the case of 2-shot testing, accuracy for ‘mid’ and ‘far’ distances are 89.7% and 85.3%, respectively. Even the farthest distance achieved 90.5% accuracy in 3-shot testing, and ‘Near’ distance is already suitable for daily use. In the future, we hope to enhance the algorithm or leverage the dual-microphone hardware present in most modern smartphones to further improve accuracy.

5) Impact of complex environments: Finally, we supplement tests for more complex environments, including scenarios with much noisier backgrounds (exceeding 60 dB) and outdoor settings. The experiments are conducted in both laboratory server room and outdoor locations on the school campus. Two male participants, each 22 years old, provide 400 testing instances of digits ‘0’-‘9’ for each scenario. Fig. 31(a) and 31(b) show the experimental setups, and Fig. 31(c) and 31(d) shows the corresponding spectrograms for digit ‘0’, respectively. The results in Fig. 32 demonstrate that in both noisier and outdoor environments, the 2-shot accuracies are 90.9% and 89.6%, respectively. And the 3-shot accuracies in both environments exceed 91%. Compared to the cross-environment scenario shown in Fig. 18, the accuracy remains relatively
and network latency. Table II presents the time consumption when a user device communicates with the server. While the results in Table II demonstrate the feasibility of our system, deploying our software to a large number of users may introduce additional latency. We leave this as future work. Additionally, considering resource limitations, especially in scenarios with constrained memory, relying on mobile devices for the inference process may be challenging. Currently, our edge devices only need to emit ultrasound, record echoes, and transmit data to the server. Depending on the server for processing results is not without drawbacks, involving additional data transfer and potential privacy concerns. Hence, we will continue to explore more suitable approaches.

C. Utilization of sensing capacity

The present version of EchoGest only makes use of a single pair of microphone and speaker. But many commercial smart devices are now equipped with multiple speakers and/or microphones. This provides an advantage for more accurate and robust acoustic-based gesture recognition systems. More specifically, if we can combine the signals from multiple speakers or microphones, we envision that the features of different gestures reflected in the Doppler spectrograms will be more evident. To the best of our knowledge, there seems to be no current improvement in acoustic-based gesture recognition technology utilizing dual speakers or microphones available in commercial products. We believe that analyzing the frequency shift patterns of dual speakers-microphone pairs could lead to a better enhancement in recognition accuracy.

VIII. Conclusion

In this paper, we propose an acoustic in-air gesture recognition system, EchoGest, which is scalable to recognize various unseen gestures. Our system emits high-frequency sound waves and records the echo signals resulting from gesture movements. We process audio signals to get spectrogram and use our classification model to recognize. The classification model that we propose integrates well-designed feature-wise transformation layer into prototypical network framework. The layer enhances feature activations through affine transformations, generating more diverse feature distributions, thus facilitating cross-task and cross-domain recognition. We obtain required regularization parameters of this layer through a two-stage training, which is more suitable for our gesture recognition task. Our improved method results in a 10% accuracy increase in 1-shot cases. We test the recognition accuracy with ‘0’-‘9’ digits as customized gestures. The 2-shot accuracy achieves 93.7%, and top-2 accuracy in recognizing various type of gestures all over 98%. We also explore semi-supervised clustering to update prototypes with each user’s data to achieve a personalized customization effect. We conduct a series of experiments, all of which show EchoGest’s usability, as well as the potential for smartwatch devices.

ACKNOWLEDGMENTS

We really appreciate the kind effort of the AE and anonymous reviewers, for giving precious suggestions to improve the
manuscript. This research was supported in part by the China NSFC Grant (No. 62172286, No. U2001207), Guangdong NSF (No. 2022A151011509), Guangdong Provincial Key Lab of Integrated Communication, Sensing and Computation for Ubiquitous Internet of Things (No. 2023B1212010007), the Project of DEGP (No. 2023KCXTD042), and Tencent “Rhinoceros Birds” - Scientific Research Fund for Young Researchers of Shenzhen University. Yongpan Zou is the corresponding author.

REFERENCES


