Hand Gesture Recognition Using Few Shot Learning

Akash Sinha (7015406), Akhil Juneja (7015523), Tulika Nayak (7025773)
Universität des Saarlandes, Saarland, Germany

Abstract

Hand gesture recognition is essential in human-computer interaction in order to capture different human gestures and their intentions. Most prior works are based on classical machine learning techniques and deep neural networks which have to be trained on huge training datasets. However, the existing hand gesture database could not meet the needs of gesture interaction in varied fields. Imagine a case when the user wants to define a new gesture, the user will have to collect several samples of the new gesture and re-train the entire model. The existing networks fail to complete the optimization of the model with small size training samples. In an attempt to solve that we have used Few Shot Learning techniques to work with limited training datasets. We have obtained different results with different few-shot learning algorithms while performing dynamic hand gesture recognition using sensor data from RGB 2D cameras.

1. Introduction

A gesture is a movement usually of the body or limbs that expresses or emphasizes an idea, sentiment, or attitude [1]. To understand that we need gesture recognition. In this report, we are discussing our work in implementing hand gesture recognition that enables users to record their own hand gestures, using 5 samples of each class, to train a deep learning model using few shot learning approaches, with the classification accuracy of 60.7%.

2. Related Work

Gesture recognition is an active research area so, there is a large body of work within this field. We reviewed a lot of them to concretely agree on our project idea, but here are the most relevant publications directly related to our project.

First of these works, and the most relevant, would be Project Prague by Microsoft [4], which is a Software Development Kit (SDK) that enables developers to define a set of gestures to control applications. However, Project Prague needs a depth camera like Intel RealSense or Microsoft Kinect, in addition to it being an SDK so, normal users might find it a bit difficult to use.

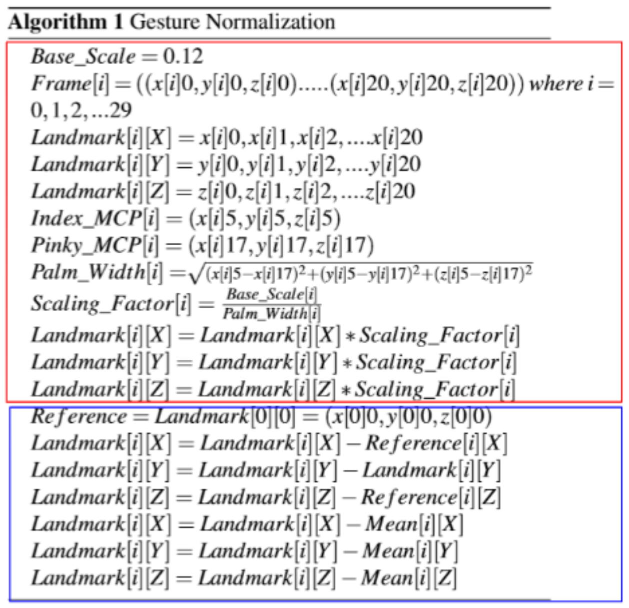
In order to choose gestures that our system can be tested on, and to include as the predefined set of gestures, we utilized the work from a recent publication by Sharma et al. 2021 [6], in which they test a variety of micro-gestures that can be performed by any of the five fingers, while holding various objects. They performed various gestures, but we chose three to test out, which were tap, flexion or swipe-up, and extension or swipe-down. We chose to follow this work, because they have a concept of gestures that can be performed, uniformly, by all five fingers within one hand.

Although very little work exists for hand gesture recognition using few-shot learning, one such work is done by Elahe Rahimian et al. 2020 [7]. They implemented a few-shot learning approach to classify hand gestures via sEMG(surface ElectroMyography) signals. They claimed to achieve an accuracy of 81.89% for a 5-way 5-shot classification. We implemented a similar network architecture as proposed in their work for classification of gestures captured by 2D cameras and we have compared our results with the results they obtained.

3. Methodology

3.1. Gesture Capturing

One of the important aspects in the work that we wanted to achieve, was to use a normal laptop camera or webcam for capturing the gestures so that the need of expensive depth cameras could be avoided. In order to achieve that, we used the **MediaPipe** Hands framework [12], which is an open source and freely available solution by Google, that can track and recognize hand and finger positions using normal webcams and produces 21 landmarks in each frame and each landmark holds three coordinates which are **x**: the width, **y**: the height, and **z**: the depth, but the z coordinate represents the depth from the camera with the wrist being the origin. We used Python Programming Language for our implementation [13]. In order to make use of MediaPipe, we also used OpenCV (Open Source Computer Vision Library) which is an open source computer vision and ML software library [14]. All the gestures were captured within a 30-frame box window, due to software limitations from



Algorithm 1 Gesture Normalization

```
$Base\_Scale = 0.12$
$Frame[i] = ((x[i]0, y[i]0, z[i]0).....(x[i]20, y[i]20, z[i]20))\ where\ i = 0,1,2,...29$
$Landmark[i][X] = x[i]0, x[i]1, x[i]2, ....x[i]20$
$Landmark[i][Y] = y[i]0, y[i]1, y[i]2, ....y[i]20$
$Landmark[i][Z] = z[i]0, z[i]1, z[i]2, ....z[i]20$
$Index\_MCP[i] = (x[i]5, y[i]5, z[i]5)$
$Pinky\_MCP[i] = (x[i]17, y[i]17, z[i]17)$
$Palm\_Width[i] = \sqrt{(x[i]5 - x[i]17)^2 + (y[i]5 - y[i]17)^2 + (z[i]5 - z[i]17)^2}$
$Scaling\_Factor[i] = \frac{Base\_Scale[i]}{Palm\_Width[i]}$
$Landmark[i][X] = Landmark[i][X] * Scaling\_Factor[i]$
$Landmark[i][Y] = Landmark[i][Y] * Scaling\_Factor[i]$
$Landmark[i][Z] = Landmark[i][Z] * Scaling\_Factor[i]$
$Reference = Landmark[0][0] = (x[0]0, y[0]0, z[0]0)$
$Landmark[i][X] = Landmark[i][X] - Reference[i][X]$
$Landmark[i][Y] = Landmark[i][Y] - Landmark[i][Y]$
$Landmark[i][Z] = Landmark[i][Z] - Reference[i][Z]$
$Landmark[i][X] = Landmark[i][X] - Mean[i][X]$
$Landmark[i][Y] = Landmark[i][Y] - Mean[i][Y]$
$Landmark[i][Z] = Landmark[i][Z] - Mean[i][Z]$
```

Figure 1. Gesture Normalization

MediaPipe being able to only capture 15-20 frames per second. So, some gestures took less than 30 frames, but the maximum number of frames for the recorded gestures was 30, which amounts to approximately a second and a half. For the gestures that were less than 30 frames, we padded them by adding frames filled with zeros (0, 0, 0) so that all the captured gestures were 30 frames.

3.2. Gestures Set

We recorded gestures which included performing tapping, swipe up, and swipe down with all five fingers. So we had 24 different gestures collected from two participants. We collected 18 gestures (G1, G2, ..., G18) to train our model in which participant one recorded (G1, G2, ..., G12) and participant two recorded (G13, G14, ..., G18). We collected 6 gestures (G7, G8, ..., G12) for validation and 6 gestures (G19, G20, ..., G24) to test our model both recorded by participant 2. Each gesture has twenty five samples, while in some cases more data was recorded to further test the system as a whole.

Before actually sending the data to the learning models, we had to normalize the data, which is discussed in detail, in the following part.

3.3. Data Preprocessing

Before sending the data captured from MediaPipe to the classifier, we needed to perform a normalization step, because the x and y coordinates that MediaPipe captures, are normalized with respect to the computer screen, meaning that the top left corner represents (0, 0) and the bottom right corner represents (1, 1) with the step size depending on the camera resolution. Our normalization algorithm works by scaling the hand coordinates according to the palm width, which is the distance between the Metacarpophalangeal (MCP) joints of the index and pinky fingers, then we reference all the coordinates according to the wrist joint of the first frame, which becomes the origin (0, 0, 0), and finally we subtract the mean so that we end up with values ranging from (-1, 1). To summarize or better understand all of this, the process of how a single frame is normalized, is seen in figure 1

The complete data pipeline of gesture capturing and normalization can be seen in figure 2

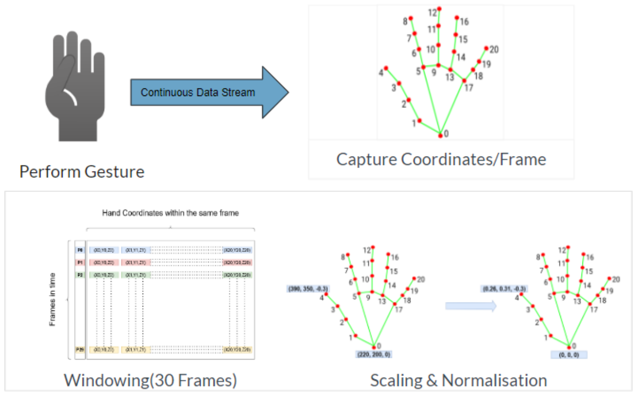


Figure 2. Data capturing and Normalisation

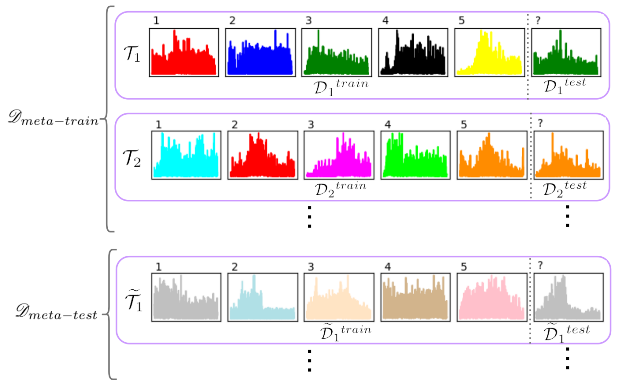

$\mathscr{D}_{meta-train}$
$\mathcal{T}_1$
$\mathcal{D}_1^{train}$
$\mathcal{D}_1^{test}$
$\mathcal{T}_2$
$\mathcal{D}_2^{train}$
$\mathcal{D}_2^{test}$
$\mathscr{D}_{meta-test}$
$\widetilde{\mathcal{T}}_1$
$\widetilde{\mathcal{D}}_1^{train}$
$\widetilde{\mathcal{D}}_1^{test}$
1
2
3
4
5
?

Figure 3. Meta learning framework

3.4. Meta Learning Framework(N Way K Shot)

Meta-learning aims to address the lack of data by expanding the learner's(neural network) focus to a variety of related tasks. Rather than training learners on a single task, meta-learners train learners on distributions of similar tasks in order to learn strategies that generalize to related but unseen tasks from distributions of similar tasks. A framework of a few shot learning approach is shown in the Figure 3

Here, each task mimics a few-shot scenario, so in N-way

K-shot classification, each task contains K different examples of N classes. These are called the support set for a task and are used to learn how to solve that task. Additionally, there are other examples of the same classes(N classes used in the support set of the task) called query sets, which are used to evaluate the performance of this task. No two tasks are exactly the same. The idea is that during training, the system repeatedly checks for instances (tasks) that match the structure of the last few-shot task, but contain different classes.

We use a series of test tasks to evaluate performance on a small number of shots. Each contains only hidden classes that are not included in any training task. For each, we measure query set performance based on our knowledge of the support set.

3.5. Learning Models

To actually be able to recognize the different gestures, we needed to try different algorithms to see which ones work out best. We started by testing various algorithms to test the performance.

3.5.1 1D CNN and a Fully connected layer

This is not a few-shot learning based classifier. We used this classifier to test the efficiency of our data pre-processing steps.

The network consists of two primary components - embedding layer and fully connected dense layer. The embedding layer consists of two 1D CNN layers, the first consisting of 128 kernels, while the second layer consisting of 64 kernels and Tanh as an action activation function after each layer. We applied Batch normalization to the output of convolutional layer and also used a Dropout layer after the convolutional layer for regularization. The features extracted after the second convolutional layer are then passed to a fully connected dense layer, where a Softmax activation is used, which then classifies the extracted feature into one of the gesture classes. The whole approach can be seen in Figure 4

3.5.2 Prototypical Network

In prototypical network [8], there exists a embedding space in which points forms clusters around a single prototype representation for each class. This is performed by learning non-linear mapping of the input into an embedding space using a neural network and class's prototype is taken as the mean of its support set in the embedding space. Classification is performed by finding the nearest class prototype for an embedded query point.

Embedding Module Embedding module has 4 1D CNN layers. First layer consists of kernel size=3, input channel=63, output channel=128. Second, third and fourth layers consists of kernel size=3, input channel=128, output channel=128, ReLU as activation function and applied batch normalization after every output. Embedding module can be seen in the figure 5. We used cosine similarity to find distance between the embedding.

3.5.3 Relation Network

Two-branch Relation Network (RN) [9] performs few-shot recognition by learning to compare query data against few-shot labeled support data. This architecture is shown in figure 6

First, an embedding module generates representations of the query and support data. Embedding module 3.5.2 is similar to the one we used in previous section.

Second, these embeddings are compared by a relation module(learnable non-linear comparator, instead of a fixed linear comparator) that determines if they are from matching categories or not. Relation module has two 1D CNN layers followed by two fully connected layers. First 1D CNN layer consists of kernel size=3, input channel=256, output channel=64, second 1D CNN layer consist of kernel size=3, input channel=64, output channel=64. Each layer has ReLU as an activation function and batch normalization is applied after every output. First fully connected layer has ReLU as activation function and second fully connected layer has softmax as activation function. Relation module can be seen in the figure 8

3.5.4 SNAIL architecture

A Simple Neural Attentive Meta-Learner(SNAIL) [10] is a sequence based meta-learning approach where the entire support set is ingested as a sequence of tuples x, y each containing a data example x and a label y. As shown in Figure 7, the last tuple of the task consists of just a data example from the query set and the system is trained to predict the missing label. The parameters of the sequence model are updated so that the classifier classifies the query set accurately across different tasks. In our case, x corresponds to the embedding generated from the embedding module mentioned in section 3.5.2.

3.6. Experiments and Results

In this section, we describe the experiments we performed and the results we obtained. We performed four experiments in which except the first one the rest use different few shot learning approaches. To evaluate our data-collection pipeline and data-prepossessing steps, we trained a 1D-CNN based classifier without using few shot learning approach. We trained the model on 6 different classes with each containing 25 samples from each participant with 70-

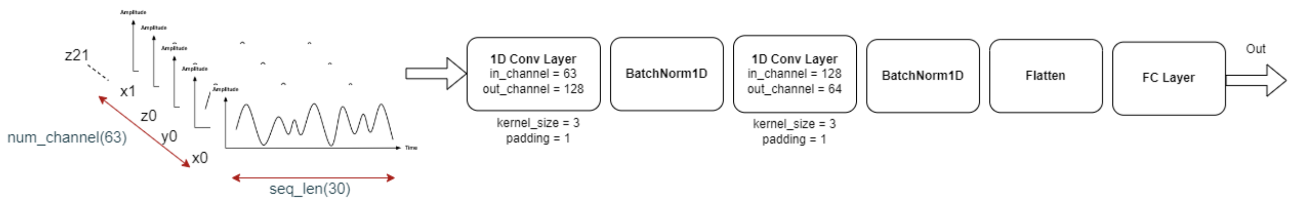


Figure 4. 1D CNN Classifier

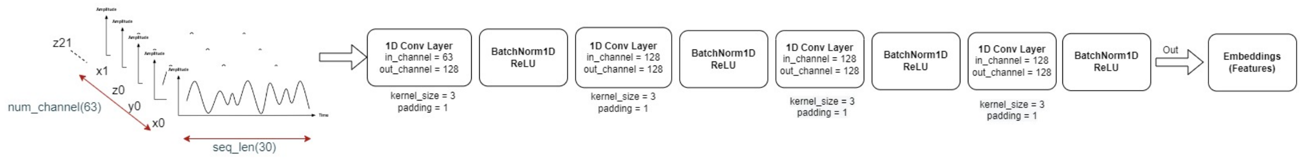


Figure 5. Embedding Module

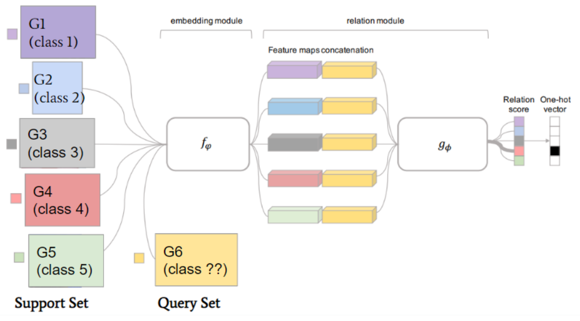


Figure 6. Relation Network Architecture

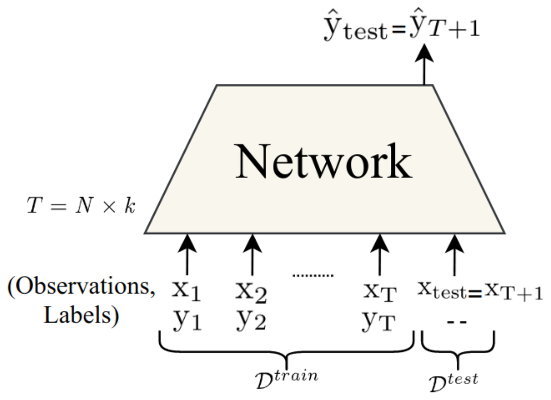


Figure 7. For each task T_j , the set of observations and labels are concatenated together and sequentially fed to the model. The final example is concatenated with a null label instead of True label. The network is supposed to predict the missing label of final example based the previous labels that it has seen. In N -way k -shot classification, N shows the number of classes which are sampled from whole set of labels, and k shows the examples that are sampled from each of those N classes

15-15 as train, validation and test split and got an accuracy of 95.25% on test data.

Prototypical Networks and Relational Networks are trained in multiple iterations. In each iteration, a task containing the support set and query set is forward propagated and the loss is calculated based on error in classification of query set labels. This loss is then back-propagated to update the weights of the network. Therefore, for these two networks, weight updation happens after every iteration. Also, the tasks for these two networks can have more than one example in their query set. We used 18 different classes for training and used 6 similar classes (same as in training) for validation and 6 new classes for testing. As the training happens through stochastic gradient descent, the training process was unstable and the training loss and accuracy fluctuated a lot.

SNAIL architecture was trained using an approach similar to mini-batch gradient. In this case, multiple tasks are clubbed together in a mini-batch and the weights of the model is updated after forward-propagation of each batch. The training process was stable in this case; and the training loss showed a continuous descent with epochs, with training accuracy reaching to around 100%. We used early-stopping to extract the model which performs best on test-data. We attained an accuracy of 52% on the test set reported in section 3.2 for 5-way 5-shot classification. However, when we increased the data size by 10 samples in each class in training set, the accuracy of classification for the same test set increased to 60.7%.

The results of our experiment is summarised in table 1 where the accuracies obtained in different experiments is also compared with the accuracy reported in [7] for 5-way

Model	Class - Shots	Test Accuracy
FS-HGR (Baseline) [7]	5 way - 5 shot	81.08 %
Relation Network	5 way - 5 shot	34.22 %
Prototypical Network	5 way - 5 shot	35.07 %
Prototypical Network	3 way - 5 shot	54.33 %
SNAIL architecture	5 way - 5 shot	52.00 %
SNAIL architecture(increased dataset by 10 samples/class)	5 way - 5 shot	60.70 %

Table 1. Model comparison along with baseline system

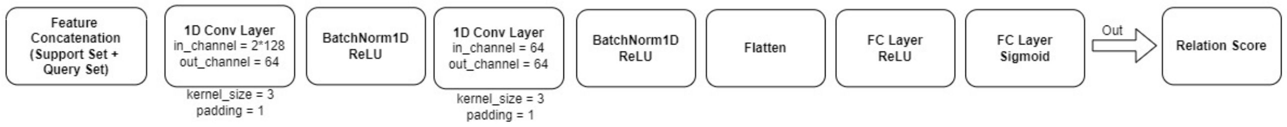

Feature Concatenation (Support Set + Query Set)
1D Conv Layer in_channel = 2*128 out_channel = 64
kernel_size = 3 padding = 1
BatchNorm1D ReLU
1D Conv Layer in_channel = 64 out_channel = 64
kernel_size = 3 padding = 1
BatchNorm1D ReLU
Flatten
FC Layer ReLU
FC Layer Sigmoid
Out
Relation Score

Figure 8. Relation Module

5-shot classification accuracy based on new subjects with few-shot.

3.7. Conclusion

Based on our experiments, we concluded that SNAIL network is a more stable way of few shot learning as the training was done in batches of episodes. With SNAIL architecture, training accuracy reached to 100%, while the highest test accuracy we obtained was 60.7%. This means, that we need to increase our training dataset to improve the test accuracy. Prototypical and Relation networks are trained using SGD. Therefore, the training was not stable and training loss and accuracy fluctuated a lot.

References

[1] https://www.merriam-webster.com/dictionary/gesture 1

[2] Silverio-Fernández, M., Renukappa, S. & Suresh, S. What is a smart device? - a conceptualisation within the paradigm of the internet of things. Vis. in Eng. 6, 3 (2018). https://doi.org/10.1186/s40327-018-0063-8

[3] What is IoT? [Blog post]. Retrieved from https://www.oracle.com/internet-of-things/what-is-iot/

[4] Microsoft Project Prague - Hand Gestures SDK https://www.microsoft.com/en-us/research/project/project-prague/ 1

[5] Nguyen, Trong Khanh & Ha, Bui & Pham, Cuong. (2019). Recognizing Hand Gestures for Controlling Home Appliances with Mobile Sensors. 10.1109/KSE.2019.8919419.

[6] Adwait Sharma, Michael A. Hedderich, Divyanshu Bhardwaj, Bruno Fruchard, Jess McIntosh, Aditya Shekhar Nittala, Dietrich Klakow, Daniel Ashbrook, and Jürgen Steimle. 2021. ”SoloFinger: Robust Microgestures while Grasping Everyday Objects”. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* Association for Computing Machinery, New York, NY, USA, Article 744, 1–15. https://doi.org/10.1145/3411764.3445197 1

[7] E. Rahimian, S. Zabihi, A. Asif, D. Farina, S. F. Atashzar and A. Mohammadi, ”FS-HGR: Few-Shot Learning for Hand Gesture Recognition via Electromyography”, *in IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 29, pp. 1004-1015, 2021, doi: 10.1109/TNSRE.2021.3077413. https://arxiv.org/pdf/2011.06104.pdf 1, 4, 5

[8] J. Snell, K. Swersky, and R. S. Zemel. ”Prototypical networks for few-shot learning”. In NIPS, 2017. https://doi.org/10.48550/arXiv.1703.05175 3

[9] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr and T. M. Hospedales, ”Learning to Compare: Relation Network

for Few-Shot Learning," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1199-1208, doi: 10.1109/CVPR.2018.00131. https://arxiv.org/pdf/1711.06025.pdf 3

[10] Mishra, Nikhil, Mostafa Rohaninejad, Xi Chen and P. Abbeel. "A Simple Neural Attentive Meta-Learner." ICLR (2018). https://arxiv.org/pdf/1707.03141.pdf 3

[11] Yi-Ping Phoebe Chen, Elena P. Ivanova, Feng Wang, Paolo Carloni, 9.15 - Bioinformatics, Editor(s): Hung-Wen (Ben) Liu, Lew Mander, Comprehensive Natural Products II, Elsevier, 2010, Pages 569-593, ISBN 9780080453828, https://doi.org/10.1016/B978-008045382-8.00729-2

[12] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C. L., and Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking. arXiv preprint arXiv:2006.10214. https://google.github.io/mediapipe/ 1

[13] Van Rossum, G., and Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace. 1

[14] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools. 1

[15] Kingma, Diederik & Ba, Jimmy. (2014). "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations*. https://www.researchgate.net/publication/269935079_Adam_A_Method_for_Stochastic_Optimization