CS445 Final Project Report: Hand Gesture Recognition

Jiahao Zhang and Zeyu Liao

{jiahao4, zeyu9}@illinois.edu

Demo video link: video1, video2, video3

Code link: https://github.com/Zeyu-UIUC/445-finalproject.git

1. Motivation and impact

Recognizing human actions is a challenging yet crucial task in computer vision with wide ranging potential applications, such as healthcare and autonomous driving. In this final project, our focus is on hand gesture recognition, a subfield of human action recognition. Our specific aim is to learn and implement the natural human gesture recognition model in the paper "Deep learning for hand gesture recognition on skeletal data" **[1]**. This paper proposes the use of skeletal ("pose") representations in place of image and video representations, which are lightweight yet very effective. We hope to replicate the result from this paper, and we believe that by leveraging the computational photography techniques we learned in CS445, we can create value by combining them with deep neural networks.

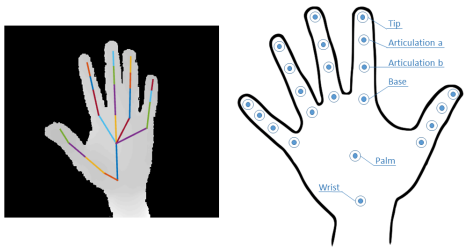

Figure 1. Hand skeletal representation.

2. Approach

Our project comprises three main parts: (1) pre-processing the training data, (2) replicate the model, and (3) developing applications around the model. For the first part, we utilized the SHREC 2017 dataset, which is a 3D Hand Gesture Recognition Skeletal Dataset. The dataset was pre-processed by resizing the sequence length, shuffling, and splitting it into a training set and a test set. Next, we replicated the model proposed in the paper using PyTorch with reference to a small portion of the paper's code. We made sure to properly cite the portion of the code that we used in our implementation. For our first application, we created a Python script that captures a still image and transforms it into skeletal data. We then utilized the model to predict the gesture being made in the image. We extended our first application to incorporate hand gesture recognition using video by transforming each frame into skeletal data, which is then used to make predictions. In our second application, we acquired a Leap Motion 3Di sensor device to monitor hand gesture movements. We developed C# scripts to record the hand tracking data and store it into a CSV data file. Subsequently, we transformed the data from the CSV file into skeletal data, which was then used as input for the model to make predictions."

Figure 2. Leap Motion 3Di sensor device.

3. Implementation Details

For this project, we employ Jupyter Notebook and Python, along with several packages including: sklearn, glob, numpy, pickle, ndimage, train_test_split, itertools, torch, sklearn.utils, time, math, PIL, torchvision.transforms, and tensorboardX.

3.1 Data Processing

The dataset we used for our model is SHREC 2017 **[2]**. The dataset is organized as follows: there are 14 gesture folders, each containing subfolders for the fingers used (finger_1, finger_2, etc.). Within each finger folder, there are subfolders for the subjects (subject_1, subject_2, etc.). Within each subject folder, there are subfolders for each trial (essai_1, essai_2, etc.). Each trial folder contains depth images (depth_0.png through depth_N-1.png), as well as three text files: general_informations.txt, skeletons_image.txt, and skeletons_world.txt.

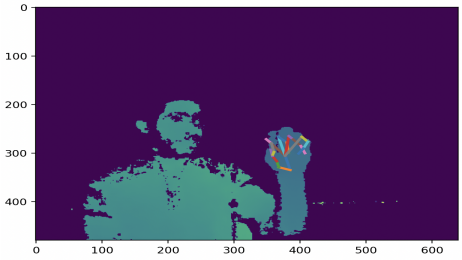

Figure 3. SHREC dataset gesture 1 example.

In loadData.py, we pre-processed the SHREC dataset through several functions. The function load_gestures takes the dataset name, root directory, and other parameters as inputs and returns a list of numpy arrays containing the gesture data, as well as labels for the gestures (either 14 or 28 labels depending on the version_y parameter). The resize_gestures function resizes the gestures to a specified length. The write_data function writes the loaded and processed data to a file using the pickle format. The code then loads the gesture data from the SHREC dataset, resizes the gestures to a length of 100, and splits the data into training and testing sets using the train_test_split function. The training and testing sets, along with their corresponding labels, are then saved to a file using the write_data function. The saved data includes the following variables: x_train, x_test, y_train_14, y_train_28, y_test_14, and y_test_28.

In the main.ipynb file, the pickle format dataset generated from loadData.py was further pre-processed. Initially, the data was shuffled and resized, and then it was converted to a PyTorch tensor. This tensor can now be utilized for training the model.

3.2 Model

In main.ipynb, we have implemented a neural network model for gesture detection. The model's input is a tensor with a data type of tensor and shape (batch_size, duration, n_channels). Each batch contains hand skeletons for a specific duration of time, with each hand skeleton having 22 joints and 3 channels. To extract features from the input data, we first process each channel independently. We employ 1D convolutions to process each channel, and the neural network consists of three convolutional layers and pooling layers. The output of each convolutional layer is concatenated into a single output, which is then utilized as input for the next layer. Finally, the three outputs are concatenated into one output.

The neural network architecture for this model can be summarized as follows:

- Input layer with shape (batch_size, duration, n_channels).

- Three 1D convolutional layers with padding, followed by a max pooling layer.
- Three output layers for each channel, with a concatenation layer at the end.

```
HandGestureNet(
  (all_conv_high): ModuleList(
    (0-65): 66 x Sequential(
      (0): Conv1d(1, 8, kernel_size=(7,), stride=(1,), padding=(3,))
      (1): ReLU()
      (2): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (3): Conv1d(8, 4, kernel_size=(7,), stride=(1,), padding=(3,))
      (4): ReLU()
      (5): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (6): Conv1d(4, 4, kernel_size=(7,), stride=(1,), padding=(3,))
      (7): ReLU()
      (8): Dropout(p=0.2, inplace=False)
      (9): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (all_conv_low): ModuleList(
    (0-65): 66 x Sequential(
      (0): Conv1d(1, 8, kernel_size=(3,), stride=(1,), padding=(1,))
      (1): ReLU()
      (2): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (3): Conv1d(8, 4, kernel_size=(3,), stride=(1,), padding=(1,))
      (4): ReLU()
      (5): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (6): Conv1d(4, 4, kernel_size=(3,), stride=(1,), padding=(1,))
      (7): ReLU()
      (8): Dropout(p=0.2, inplace=False)
      (9): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (all_residual): ModuleList(
    (0-65): 66 x Sequential(
      (0): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (1): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      (2): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (fc): Sequential(
    (0): Linear(in_features=7128, out_features=1936, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1936, out_features=14, bias=True)
  )
)
```

Figure 4. Model Neural Network Structure.

By using 1D convolutions and pooling layers, we can extract meaningful features from the hand skeleton data. The concatenation layer at the end allows us to combine the information from each channel into a single output, which can be used to predict the gesture being performed.

3.3 First Application

For the first implementation, we converted hand data from still images and videos to skeletal data. The demo video can be seen here: video1, video2.

3.4 Second Application

For our second application, we utilized a Leap Motion sensor and Unity, with the Leap Motion tracking software and the Leap Motion Unity package installed. In the unity scene, we have our Leap motion hand with a C# script attached. This C# script in Unity uses the Leap Motion SDK to capture hand tracking data and saves it to a CSV file. The script finds the LeapServiceProvider object, sets up a StringBuilder object and file path, and captures the positions of the palm, wrist, and finger joints every 1/6th of a second. It formats the data as CSV rows using the StringBuilder object and writes it to a CSV file when the application is quit. Then, the main.ipynb file contains a function called unityProcess that processes hand data from a CSV file and converts it to a tensor. This tensor is subsequently utilized to predict the type of gesture being performed. Finally, the demo video for collecting hand data in Unity can be seen here: video3.

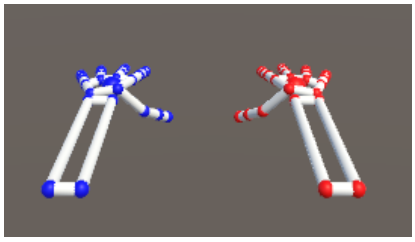

Figure 5. Leap motion Hand in Unity.

4. Result

We trained our model for 10, 20, 30, and 40 epochs and analyzed the Tensorboard results. The analysis indicated that training the model for 20 epochs is adequate to achieve optimal performance without encountering overfitting or underfitting issues. Additionally, our model achieves about the same accuracy score on the training and test sets compared with the paper's result.

```
[INFO] Started to train the model.
Training the model on CPU.
Epoch #001 | Time elapsed : 02m 35s | Loss : 3.3115e+02 | Accuracy_train : 3.2689e-01 | Accuracy_test : 3.1667e-01
Epoch #002 | Time elapsed : 05m 09s | Loss : 1.2053e+02 | Accuracy_train : 5.1639e-01 | Accuracy_test : 4.4524e-01
Epoch #003 | Time elapsed : 07m 54s | Loss : 8.6311e+01 | Accuracy_train : 7.2605e-01 | Accuracy_test : 6.4762e-01
Epoch #004 | Time elapsed : 10m 48s | Loss : 6.3097e+01 | Accuracy_train : 7.9328e-01 | Accuracy_test : 7.1667e-01
Epoch #005 | Time elapsed : 13m 30s | Loss : 5.0130e+01 | Accuracy_train : 8.3025e-01 | Accuracy_test : 7.4048e-01
Epoch #006 | Time elapsed : 16m 35s | Loss : 4.3560e+01 | Accuracy_train : 8.5042e-01 | Accuracy_test : 7.8333e-01
Epoch #007 | Time elapsed : 19m 18s | Loss : 3.7319e+01 | Accuracy_train : 8.7437e-01 | Accuracy_test : 8.1667e-01
Epoch #008 | Time elapsed : 22m 09s | Loss : 3.2202e+01 | Accuracy_train : 9.0420e-01 | Accuracy_test : 8.4524e-01
Epoch #009 | Time elapsed : 24m 51s | Loss : 2.8564e+01 | Accuracy_train : 9.1218e-01 | Accuracy_test : 8.7143e-01
Epoch #010 | Time elapsed : 27m 37s | Loss : 2.5529e+01 | Accuracy_train : 9.1765e-01 | Accuracy_test : 8.7619e-01
Epoch #011 | Time elapsed : 30m 26s | Loss : 2.2935e+01 | Accuracy_train : 9.1723e-01 | Accuracy_test : 8.7381e-01
Epoch #012 | Time elapsed : 33m 31s | Loss : 2.1980e+01 | Accuracy_train : 9.1176e-01 | Accuracy_test : 8.7619e-01
Epoch #013 | Time elapsed : 36m 28s | Loss : 2.1350e+01 | Accuracy_train : 9.2941e-01 | Accuracy_test : 8.8571e-01
Epoch #014 | Time elapsed : 39m 20s | Loss : 1.9156e+01 | Accuracy_train : 9.1176e-01 | Accuracy_test : 8.5952e-01
Epoch #015 | Time elapsed : 42m 08s | Loss : 1.6812e+01 | Accuracy_train : 9.0966e-01 | Accuracy_test : 8.5000e-01
Epoch #016 | Time elapsed : 44m 57s | Loss : 1.6905e+01 | Accuracy_train : 9.3151e-01 | Accuracy_test : 8.7857e-01
Epoch #017 | Time elapsed : 47m 52s | Loss : 1.5218e+01 | Accuracy_train : 9.2143e-01 | Accuracy_test : 8.5714e-01
Epoch #018 | Time elapsed : 50m 39s | Loss : 1.5217e+01 | Accuracy_train : 9.3487e-01 | Accuracy_test : 8.7143e-01
Epoch #019 | Time elapsed : 53m 30s | Loss : 1.3410e+01 | Accuracy_train : 9.5378e-01 | Accuracy_test : 9.0714e-01
Epoch #020 | Time elapsed : 56m 17s | Loss : 1.3599e+01 | Accuracy_train : 9.3235e-01 | Accuracy_test : 8.5952e-01
[INFO] Finished training the model. Total time : 56m 17s.
```

Figure 6. Model training log.

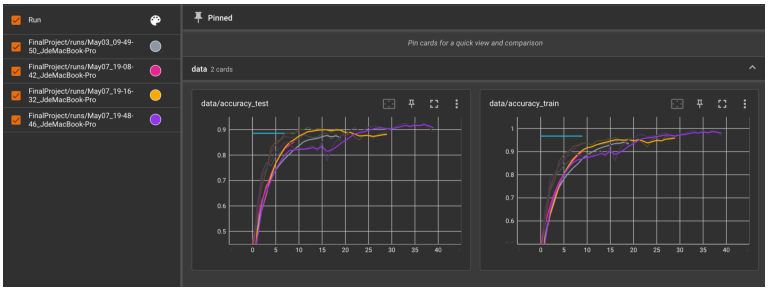

Figure 7. TensorBoard analysis for model in 10, 20, 30, and 40 Epochs

For the Leap Motion hand tracking application, we were successful in converting hand movement data to skeletal data, which we used to predict gestures from the 14 gesture class. This enabled us to accurately detect and classify hand gestures, opening up the potential for a range of exciting real-world applications.

```
// Update is called once per frame
void Update()
{
    // pasuse when 100 frames are being collected
    if (frameCount == 1000) {
        Time.timeScale = 0;
        Debug.Log("pause");
    } else {

        List<Vector3> handPosition = new List<Vector3>();

        if (leapServiceProvider.CurrentFrame != null)
        {
            if (leapServiceProvider.CurrentFrame.Hands.Count > 0) {
                // collect data for every 10 frames. stop when 100 frames are collected.
                if (frameCount % 10 == 0)
                {
                    foreach (Hand hand in leapServiceProvider.CurrentFrame.Hands)
                    {
                        if (hand.IsRight)
                        {
                            Vector3 palmPosition = hand.PalmPosition;
                            handPosition.Add(palmPosition);

                            Vector3 wristPosition = hand.WristPosition;
                            handPosition.Add(wristPosition);

                            foreach (Finger finger in hand.Fingers)
                            {
                                for (int i = 0; i < 4; i++)
                                {
                                    Vector3 jointPosition = finger.Bone((Bone.BoneType)i).NextJoint;
                                    handPosition.Add(jointPosition);
                                }
                            }
```

```
    model = HandGestureNet(n_channels=66, n_classes=14)
    model.load_state_dict(torch.load('gesture_pretrained_model.pt'))
    model.eval()

    # make predictions
    with torch.no_grad():
        unitydata = numpy.genfromtxt('GestureData.csv', delimiter=',')
        gesture_batch = unityProcess(unitydata)
        predictions = model(gesture_batch)
        _, predictions = predictions.max(dim=1)
        print("Predicted gesture classes: {}".format(predictions.tolist()))
[ ] ✓ 0.2s
Predicted gesture classes: [4]
```

Figure 8. Script for Leap Motion data collection and model prediction. Gesture class 4 refers to the gesture "pinch".

The results for still image and video hand gesture recognition using the model proposed by the paper were found to be less accurate. This can be attributed to the fact that the original model was trained primarily on pre-processed skeletal data, and may not be well-suited for handling raw image data. However, we have made significant efforts to enhance the model's functionality and improve its accuracy, particularly in the context of processing normal images and video frames. The results and details can be seen in video1 and video2.

5. Innovation and Challenge

The primary challenge of this project was replicating the model as the official code provided was incomplete and required a good understanding of deep learning concepts to be useful. In addition to resolving errors, we also spent a significant amount of time creating functions for data pre-processing and testing. These functions were necessary to ensure that the model could effectively process the data and generate accurate predictions. Another challenge is that in the original paper, the model is trained using a pre-processed skeletal dataset. There is no provision to convert normal images into skeletal data. However, we have made significant efforts to resolve this problem, which includes thoroughly understanding the structure of the skeletal dataset, the relevant parameters, and the PyTorch tensor. This understanding enabled us to devise an approach to transform normal images into skeletal data, which significantly enhances the model's functionality and applicability.

For us, using the Leap Motion hand tracking sensor and Unity to collect hand movements and convert them into skeletal data was a significant innovation. This approach enabled us to capture the precise movements of the hand and utilize them as prediction input for our pre-trained neural network model.

We have introduced another innovation by extending the original paper's model, which only supports 2D/3D skeleton images, to detect normal images. We have created two versions of gesture recognition. The first version is straightforward, where we take images and manually input them to the model. The second version is based on video frames, and it automatically detects the gesture in the frame and displays the result on the window. These extensions have significantly enhanced the model's functionality and made it more versatile in real-world scenarios. Based on the innovations we have introduced and the challenges we have overcome in this project, we believe that our efforts can be evaluated as deserving of a full score in the innovation and challenge section.

Citation:

[1] Devineau, Guillaume, Fabien Moutarde, Wang Xi, and Jie Yang. "Deep learning for hand gesture recognition on skeletal data." In 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018), pp. 106-113. IEEE, 2018.

[2] Dynamic Hand Gesture Recognition using Skeleton-based Features ,Quentin De Smedt, Hazem Wannous and Jean-Philippe Vandeborre, 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). Download from http://www-rech.telecom-lille.fr/DHGdataset/ and unzip into ./415-finalproject/dataset_dhg1428