Hand Action Recognition in G-D Video Data

Lucas Jaffe
Stanford University
450 Serra Mall, Stanford, CA 94305
jaffe5@stanford.edu

Repo: *https://github.com/LukeJaffe/cs231a_project*

Abstract

While there has been significant progress for the hand action recognition task in recent years, most state-of-the-art solutions require a complex data-annotation process that is impractical for many real-world applications. This paper seeks to explore methods which require minimal annotation, allowing for rapid prototyping of hand action recognition systems, in particular for RGB + depth video data. We present a modern convolutional neural network-based approach which is capable of classifying hand actions from offline or online video streams of arbitrary duration.

1. Introduction

FS Studio, a software contracting company which focus in-part on computer vision applications, has recently started a project involving human-computer interaction with a ToF camera produced by PMD Technologies. This camera, the CamBoard pico monstar, collects grayscale + depth (G-D) data at up to 60 fps. The work presented here involves an informal collaboration with FS Studios to produce algorithms and software specialized for data from this camera.

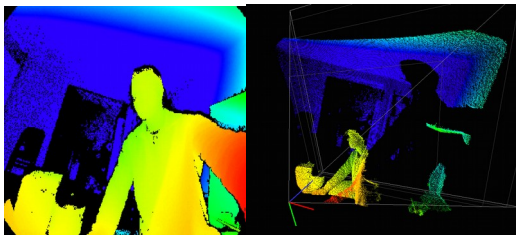

Figure 1: Views from the Pico Monstar showcasing depth. 2D (left) and 3D (right) scenes are shown in the Royale Viewer.

1.1. Problem Statement

The problem is to develop hand action recognition methods from public RGB-D or G-D video data. Once an effective solution is developed, this solution will be applied to G-D video data from the pico monstar camera. There is particular interest in the real-time scenario. This problem is interesting for a variety of applications involving human-computer interaction including video games, virtual/mixed reality, appliance interaction, and surveillance [4].

The target use case for the pico monstar camera will involve a stationary camera with a moving subject. The camera will be facing the user, and the hand will usually take up significant fraction of the FOV. Since the camera can operate at higher frame rates for closer distances, these close-distance use cases are of particular interest. Two views from the pico monstar are shown in Figure 1.

2. Background

To the author's knowledge, all modern methods for hand pose estimation and hand action recognition utilize convolutional neural networks (CNNs) [1, 2, 3, 4, 5, 6, 7]. The BigHand2.2M paper [2] and the First-Person Hand Action paper cite *Spatial Attention Deep Net with Partial PSO for Hierarchical Hybrid Hand Pose Estimation* [5] as the state-of-the-art for pose estimation. This method uses a spatial attention mechanism and Particle Swarm Optimization (PSO) to learn and perform inference on hand pose data. In the Hand Action paper, LSTM-based methods including [6] and other temporal-dependent methods such as [7] are benchmarked for a hand action recognition task.

2.1. Hand Pose Estimation

The state-of-the-art in using RGB-D for hand action recognition is to perform hand pose estimation prior to action recognition- estimating the position of (often) 21 joints in the hand from specially-annotated data. 6D (position and rotation) magnetic sensors on gloves are often used for the data collection process, resulting in highly-detailed hand pose annotations. The paper *First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations* states the following,

> "However, the recent trend is to take advantage of the depth channel [from RGB-D data] to obtain robust body pose estimates and use them directly as a feature to recognize actions. This trend has led to what is known as pose (or skeleton) action recognition, a problem which is usually tackled as a time series problem."

2.2. Hand Action Recognition with Basic Annotations

Since it is often impractical to collect this 3D joint information, a different approach is here discussed. Specifically, hand actions can also be learned without any form of annotation aside from a basic action label for the video as a whole. Assuming that the hand is prominently featured in the video, and few occlusions or distracting motions are present, much simpler approaches can be applied to learn the domain; for instance, an approach which uses CNNs to convert the imagery to some feature space, and then uses a sequential model like LSTM on the resulting feature set. This approach is the primary focus of this work.

Alternatively, simple bounding box annotations or segmentation annotations could be created without the equipment required for 6D joint annotations. A basic GUI labeling tool would be sufficient for this task, though it is time consuming to annotate any significant number of frames in this manner.

For this reason, we focus primarily on the case where only an action label is present for a given video sequence, which can easily be prepared for a dataset collected from a new camera.

3. Approach

3.1. SKIG Dataset

Due to the proprietary nature of collaborating with FS Studio, a number of common datasets are off-limits to this study. The First-Person Hand Action Dataset is probably the most extensive, given that it has 105k frames of 45 different hand-action labels couples with 21-joint annotations [4]. Despite this setback, we have found a dataset which is suitable for the task, and meets our very basic annotation needs.

The Sheffield KInect Gesture (SKIG) dataset consists of 1080 RGB-D hand gesture sequences collected from 6 different subjects using a Microsoft Kinect camera [8]. In addition the videos are annotated with 10 different gestures, 3 different poses, 3 different backgrounds, and 2 different illumination conditions. This variety of conditions is more likely to lead to strong generalization performance of a model, making the dataset a suitable candidate for transfer learning to data from a different camera. The 10 gestures of the dataset are shown in Figure 2.

The SKIG dataset is prepared such that both the RGB and D portions of the data are composed of 3-channels. This allows us to use classical pre-trained architectures on both parts of the data (which are nearly always designed for 3-channel images of some fixed size). However, there is no guarantee that the depth data domain is sufficiently similar to the RGB domain to make use of the pre-trained weights

The data was partitioned such that video data from 4 subjects was used for training, with 1 subject held out for validation, and 1 for testing.

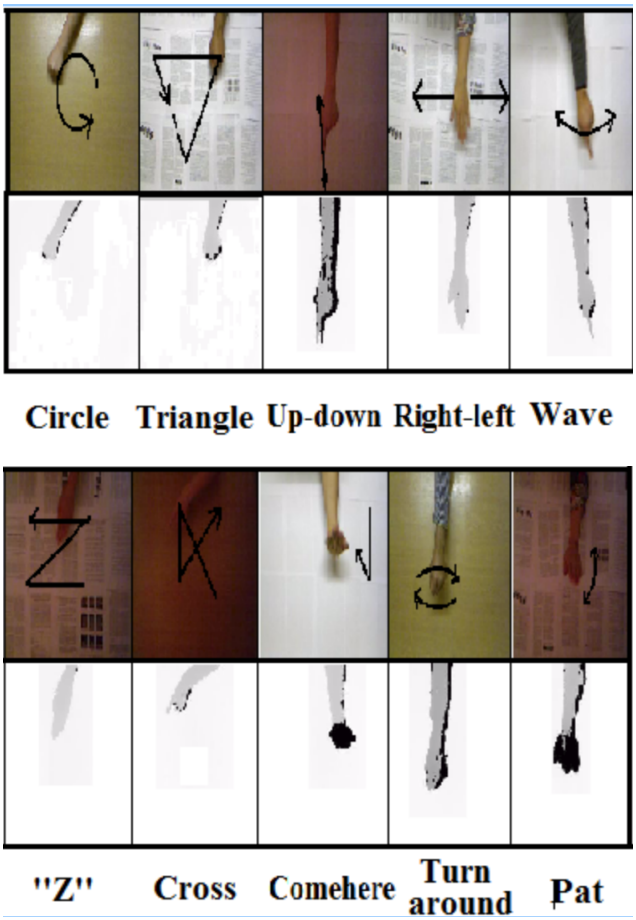

Figure 2: 10 Hand Gestures Composing SKIG Dataset

3.2. Pico Monstar Dataset

The pico monstar camera produces distinct grayscale and depth pixel maps.

The 10 hand gestures from the SKIG dataset were each replicated with the pico monstar under 3 different illumination conditions to produce a total of 30 videos. The varying illumination conditions did not noticeably impact the video quality, which is not surprising given that the sensor uses IR. We tried to produce conditions as close as possible to the SKIG dataset, including close replication of the gestures, positioning of the hand in the scene, and distance of the hand to the camera. One key difference is that the optimal range of the camera for the 10 FPS setting we used is 0.7-5.2m. Since the target hand was around 0.5m from the camera in SKIG, and therefore in our dataset, we were not operating within the optimal range. This means the data recorded was likely not

optimal, though visually it seems of decent quality, as shown in Figure 3.

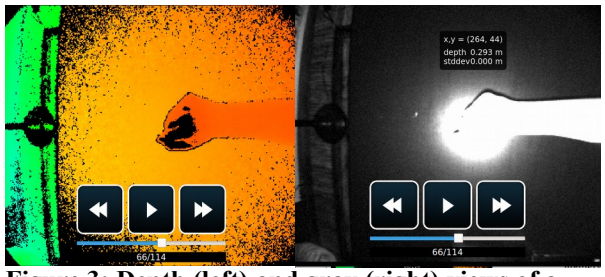

Figure 3: Depth (left) and gray (right) views of a hand in the Pico replicate of the SKIG dataset shown in the Royale Viewer.

3.3. Single Image Feature Extraction

Our initial approach was to train a standard image classifier on the SKIG dataset, using each frame of each video as individual image inputs. We did not expect this would perform exceptionally well since no temporal information is learned, but it serves as a logical baseline. The full RGB and depth data from SKIG was used.

This portion focuses primarily on use of the DenseNet architecture for spatial feature extraction [9]. In particular, the PyTorch version of DenseNet161 architecture is used, which has shown strong experimental results when trained on ImageNet[1]. The pre-trained version of the network will certainly be used for the both the RGB and depth data.

We also recommend that an autoencoder method be attempted for feature extraction, as this is less likely to remove the position-related features that may be lost by the traditional classifier.

3.4. 3D Convolutional Neural Networks

3D CNNs have been used with increasing frequency for video recognition, including in [11][12][13]. The concept is fairly simple: convolutional layers have a filter which is convolved over 2 spatial dimensions (as normal) in addition to 1 dimension of time. As we will demonstrate, classifying consecutive frames with this architecture proves to be far more effective than classifying single frames. We utilize the 3D DenseNet121 architecture used by [11].

3.5. LSTM Network

Still, there is an inherent problem with using either of the above feature extraction methods: neither takes advantage of longer-term temporal features. The 3D CNN features are a step in the right direction, but they still require a fixed-size input, and are impractical to train with more than a few dozen frames seen at a time. We trained with only 16 consecutive frames at a time, or about 1.6 seconds of real time at the 10 FPS rate used.

To fix both the problems of needing dynamically-sized input and utilizing temporal information, we take advantage of recurrent neural network architectures. In particular, the LSTM is commonly used for sequential tasks, and is our architecture of choice. The LSTM network is also a convenient package for processing data in real-time, since all messy accounting for weighting of data is done internally. Without this architecture, some type of weighted average over time would need to be used for classifier scores to make real-time time-domain predictions.

3.6. Real-Time Pipeline

Capturing, processing, and displaying imagery in real-time presents unique engineering challenges that we will discuss here. First, it should be noted that the pico monstar camera has several different capture modes available, with video framerates varying from 5-60 FPS. Lower framerate settings are able to operate effectively at greater distances i.e. pick up objects further from the camera. We mainly utilized the 5 and 10 FPS settings to reduce real-time latency.

Our pipeline follows a standard producer-consumer model in which a producer process captures and stores the data, and a consumer process loads, processes, and displays the data. The Royale SDK offers a C++ API, which we used to activate the camera and receive frames in real time. These frames are then converted to NumPy format[2], and saved to disk. The server/producer process then sends a ZeroMQ message containing the name of the file path to the client/consumer process.

When the client/consumer process receives a ZeroMQ message, the file path is loaded from disk into a NumPy array, then converted to a PyTorch tensor and sent through the neural network pipeline. The frames were stacked in consecutive groups of 16 to feed into the 3D CNN. When the network has made a classification prediction, this prediction can be displayed, along with the original image. In our demo, we display the real-time result in a Jupyter Notebook.

Python3 bindings were written for the C++ producer code, so both the producer and consumer processes can be called from Python3, even within the same script if a multiprocessing library is used.

The main sources of latency in our pipeline are in saving/loading the data to/from disk, and in displaying the result in the Jupyter Notebook. Jupyter is not very efficient with displaying imagery, so this portion could be easily improved by using a different display platform like Qt or OpenCV. The most effective way to reduce latency, then, would be to avoid saving the data to disk to begin with.

[1] http://pytorch.org/docs/master/torchvision/models.html

[2] NumPy objects in C++: https://github.com/rogersce/cnpy

One way to avoid the save to disk would be to make more complete Python3 bindings for the C++ SDK code. If a cross-language queue object were used, this queue could be shared by both the C++ producer code and the Python3 consumer code. The producer could push new NumPy frames into the queue, and they could be read and processed in the consumer. This would be better accomplished with multithreading than multiprocessing, to avoid issues with shared memory, but there would still be challenges with properly handling concurrency control so deadlocks could be avoided. A cross-language mutex could be used to solve this problem, but would likely be a significant engineering challenge to implement properly. Alternatively, the producer and consumer could both be written in C++, but this would make it harder to take advantage of existing neural network infrastructure, such as PyTorch in our implementation.

4. Experiments

All code pertaining to the project can be found at: https://github.com/LukeJaffe/cs231a_project.

4.1. Single Image Features

Preliminary experiments involved using only the RGB data of the SKIG dataset. The first experiment was to evaluate per-frame accuracy when training with the DenseNet161 classifier. This achieved a result of 36.81% accuracy on the validation set.

Following this, features for each video were extracted from the best trained network. During this process, the scores for each frame of a given video were summed and argmax'd to produce a per-video baseline validation set accuracy of 40.23%.

After features were extracted for the training and validation sets, the training features were fed into an LSTM network followed by a fully-connected layer to produce a final estimate on the validation set of 52.22% accuracy. The per-class breakdown was not even, as shown in Table 1.

Table 1: Breakdown of Performance on Validation Set by Class

Gesture	**Classification Accuracy**
Circle	0.50
Triangle	0.67
Up-down	0.22
Right-left	0.56
Wave	0.78
Z	0.06
Cross	0.89
Comehere	0.89
Turn around	0.22
Pat	0.28

4.2. 3D CNN Features

The key breakthrough in this project was in training the 3D DenseNet121 model on the SKIG dataset RGB data. This resulted in a big boost in accuracy from the single image feature methods, up to 85.25% accuracy on the validation set. We also tried the ResNet and ResNeXt codes from the same author, but they did not train well.

4.3. Transfer Learning from SKIG to Pico

Following the successful 3D CNN experiment, our goal was to make the SKIG dataset as close to the Pico data as we could. To do this, we loaded the SKIG RGB data as a single grayscale channel, and loaded the depth data as one channel as well. We then modified the first conv layer of the DenseNet121 architecture to take 2 channels instead of 3, and concatenated the grayscale and depth data into a single tensor (2 channels). This was quite effective, scoring at 84.5% accuracy on the validation set, very close to the RGB performance. We expect that this accuracy could be raised to near 100% if there was more time spent tuning the training procedure.

3D CNN features were then extracted from every consecutive set of 16 frames to produce a feature-only dataset for the training and validation data. This was used to train a separate LSTM, which scored 84.5% accuracy, exactly the same as the score-sum argmax performance. Though the performance is not any better, the real-time capabilities of the LSTM are certainly worth it.

Unfortunately, when this network was applied to the pico SKIG-imitation dataset, performance was only 20% (6/30 videos). In order to utilize the SKIG data (or another hand-action dataset) for testing on data of a different domain such as the pico data, we recommend the utilization of recent *domain adaptation* methods in deep learning. These allow transformation of a target dataset to

the source dataset for which a network is trained, potentially giving much better performance than running the trained network on totally different data out of the box.

4.4. Native Training on Pico Data

In our final experiment, we partitioned 20 of the pico dataset videos for training, and the remaining 10 for testing. We would have liked to use a validation set as well, but time limitations prevented additional data collection. Using the score-sum argmax approach, all 10 test videos were classified correctly after training with the 3D CNN. This was impressive, but can be explained by the relative similarity of the train and test videos, given that the varying illumination conditions likely did not impact the IR sensing significantly.

When the train/test 3D CNN features were extracted and separately trained with the LSTM, *per-frame* accuracy was at 83.68%, with the per-class breakdown shown in Table 2.

Table 2: Breakdown of Performance on Validation Set by Class

Gesture	**Classification Accuracy**
Circle	0.97
Triangle	0.99
Up-down	0.92
Right-left	0.53
Wave	0.72
Z	0.87
Cross	0.72
Comehere	1.00
Turn around	0.92
Pat	0.99

4.5. Real-Time Demo

The real-time Jupyter Notebook demo was lightly tested, to understand some performance heuristics. The network seemed to prefer the up-down and left-right motions to other, but would always shift to an up-down prediction when the hand was moved closed to the camera. This shows that the network was able to learn this depth information well. The closer the demo camera positioning is to the training position, logically, the more effective the demo is. With additional data and training, we predict the real-time scenario would become much more resilient to changes in hand/camera position. Some of these problems could also be solved with random data augmentations (i.e. rotation, flipping) during training. A screenshot of the demo notebook is shown in Figure 4, and a demo is available in the presentation video in the repo.

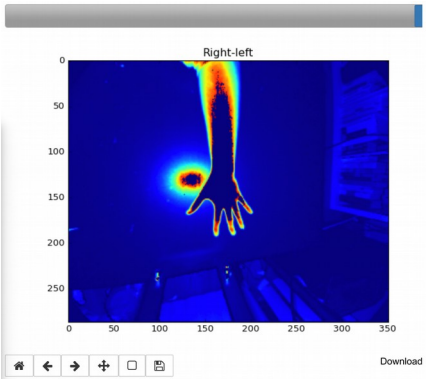

Figure 4: Real-time demo operating in Jupyter Notebook. The original image (gray mask) is shown with the NN prediction atop the plot.

5. Conclusion

This project explored methods for hand-action recognition on grayscale-depth (G-D) video data. While initial results using per-image classifier features were not very effective, features trained a 3D CNN using batches of 16 consecutive frames worked quite well. With more time, more data could be collected, and the training process could be further tuned. In this case, we believe accuracy of the model would approach 100%, and would be resilient to many different scene configurations.

References

[1] Chang, L., Deng, X., Tan, P., Wang, H., Yang, S., & Zhang, Y. (2017). Hand3D: Hand Pose Estimation using 3D Neural Network. *CoRR, abs/1704.02224*.

[2] Yuan, S., Ye, Q., Stenger, B., Jain, S., & Kim, T. K. (2017, July). Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2605-2613). IEEE.

[3] Tompson, J., Stein, M., Lecun, Y., & Perlin, K. (2014). Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, *33*(5), 169.

[4] Garcia-Hernando, G., Yuan, S., Baek, S., & Kim, T. K. (2017). First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations. *arXiv preprint arXiv:1704.02463*.

[5] Ye, Q., Yuan, S., & Kim, T. K. (2016, October). Spatial attention deep net with partial PSO for hierarchical hybrid hand pose estimation. In *European Conference on Computer Vision* (pp. 346-361). Springer, Cham.

[6] Zhu, W., Lan, C., Xing, J., Zeng, W., Li, Y., Shen, L., & Xie, X. (2016, February). Co-Occurrence Feature Learning for Skeleton Based Action Recognition Using Regularized Deep LSTM Networks. In *AAAI* (Vol. 2, p. 8).

[7] Zhang, X., Wang, Y., Gou, M., Sznaier, M., & Camps, O. (2016). Efficient temporal sequence comparison and classification using gram matrix embeddings on a riemannian manifold. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4498-4507).

[8] Liu, L., & Shao, L. (2013, August). Learning Discriminative Representations from RGB-D Video Data. In *IJCAI* (Vol. 1, p. 3).

[9] Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2017, July). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (Vol. 1, No. 2, p. 3).

[10] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

[11] Hara, K., Kataoka, H., & Satoh, Y. (2017). Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?. *arXiv preprint arXiv:1711.09577*.

[12] Ji, S., Xu, W., Yang, M., & Yu, K. (2013). 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, *35*(1), 221-231.

[13] Sun, L., Jia, K., Yeung, D. Y., & Shi, B. E. (2015). Human action recognition using factorized spatio-temporal convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 4597-4605).