CustomFX:
A Lightweight Hand Tracking Model for Musical Instruments

Gayatridevi Tarcar
Stanford University
devig17@stanford.edu

Sid Yu
Stanford University
sidyu@stanford.edu

Owen Jung
Stanford University
owenjung@stanford.edu

Abstract

Many generalized hand-recognition, tracking, and labeling models exist which can capably be deployed for a variety of static, video, and live applications. This generality, obviously, is a boon in cases of quick deployment and cases where the baseline latency is acceptable.

Using Google's MediaPipe framework for hand-tracking as a reference, we built a purpose-built, lighter-weight model that is trained on specific gestures (rather than a massive dataset of all possible hand gestures) and is deployed for a single purpose. This approach will reduce training time for our specific application (by reducing the size of training sets) and speed up the processing of the hand-recognition models.

The application our vision model is designed to control is a virtual musical effects board. It has five different effects, each keyed live by a different gesture and controlled by hand movement. The model we use uses the Google's MediaPipe to process live webcam video for landmark tracking fed into a custom-built gesture-recognition layer which controls the musical effects.

1. Introduction

1.1. Problem

The primary problem we address is latency in real-time musical performance systems that rely on video-based hand tracking. Although general-purpose models such as Google's MediaPipe are highly capable and optimized for broad gesture recognition, they are not designed for domain-specific responsiveness or customization, particularly in interactive musical settings. Gestural definition as defined within Google's MediaPipe paper (as a set of algorithms computing finger-angles) is somewhat arcane to the lay user; we seek to define a new gestural definition layer that can be quickly trained on any custom set of gestures, even those generated by a user.

To enable fast and expressive control over virtual musical effects, we built a lightweight, task-specific gesture classification model that runs on top of MediaPipe's hand landmark outputs. This custom layer maps five gestures to musical parameters: four pitch modulations (12, 7, 4, and +12 semitones) and one reverb effect.

For music, especially live performance, speed is absolutely key: We want to achieve real-time performance with our application because precise control over rhythm is absolutely key to using these tools in any serious musical context.

1.2. Related Work

As discussed before, the key piece of related work is Google's MediaPipe, which contains a hand-tracking model that first uses a model which already exists, called BlazePalm, for palm tracking, then is trained to detect and track 21 points on the hand (`https://research.google/blog/on-device-real-time-hand-tracking-with-mediapipe/`). It is trained on a manually annotated dataset of 30k images, each of which contains the 21 landmarks to track. The landmark detection is built on a deep CNN architecture. Then comes gesture detection, which is based on the data from the landmarks rather than the entirety of the image. The gesture detection from MediaPipe is encoded via a set of algorithms (it is not completely clear within Google's paper whether these algorithms are fully manually implemented or the partial product of a neural network, but it seems that the gestures they encoded are manually written as combinations of the angles between the fingers, as encoded in tracking points).The clever combination of initialization with palm-tracking and subsequent landmark-only tracking ensures that this model is very fast; it achieves latency speeds of 17ms on CPU and 12ms on GP with Pixel 6.

In our work, we use MediaPipe solely for extracting hand landmarks from live webcam video. We then discard its built-in gesture layer and replace it with a custom Multi-Layer Perceptron (MLP) classifier, trained on a labeled dataset of five gestures, defined in Section 1.3. To optimize for speed and simplicity, we use only 11 of the 21

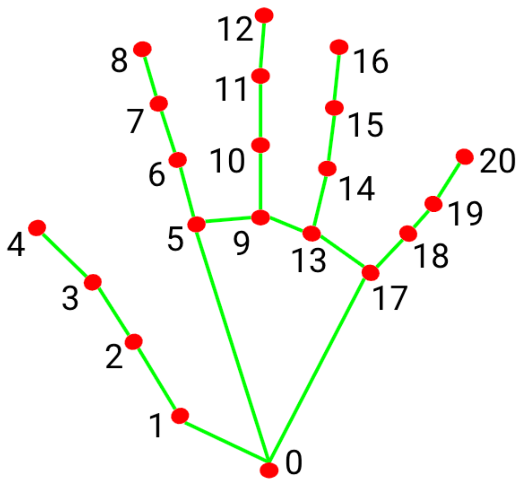

Figure 1. Hand landmarks (full set).

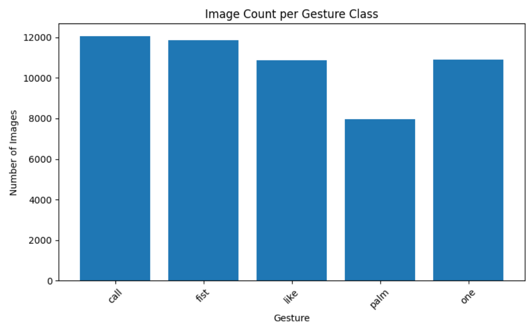

Figure 2. Distribution of images by gesture.

landmarks (33 input dimensions), selected based on which joints were most expressive for our chosen gestures. This MLP was trained from scratch using 5-fold stratified cross-validation and achieves high validation accuracy with low inference overhead. This is a more intuitive and customizable method, where we were able to choose exactly our gestures, creating a very lightweight gesture recognition layer.

Other models exist which recreate full 3-dimensional nets of hands; for example, HaMeR (`https://github.com/geopavlakos/hamer?tab=readme-ov-file`) is a 3d recreator of hands that is trained on a data set called Hint (Hand INTeractions in the wild), which contains images of hands interacting with real-world environments. These methods provide detailed spatial representations and can handle occlusions better than MediaPipe. However, they are computationally intensive and less suited for real-time interaction, particularly in creative applications like music.

More broadly, software environments like TouchDesigner have been widely adopted by artists for visual and audio synthesis using motion data. While powerful, TouchDesigner's gesture logic is embedded in a closed ecosystem, limiting access to underlying models and adaptability for developers. Our system, by contrast, is fully modular and open, enabling precise, real-time gesture control via a minimal and transparent classification layer.

1.3. Data

For our dataset, because we were focusing primarily on gesture recognition, we wanted to find a large dataset of images that was categorized by gesture. We found the dataset HAGRID - HAnd Recognition Gesture Recognition Dataset, which is a huge (1.5 terabyte, 1 million total images) dataset of high-definition (1080p) hand images, annotated by gesture (18 different gestures in the original dataset, with an additional 15 in HAGRID v2) and bounding boxes. Because we wanted to focus on a smaller set of gestures and did not have the compute time or storage to handle such a large dataset, we found a smaller version of HAGRID v1 (500,000 images) on HuggingFace which was downsampled from 1080p to 384p, allowing us the ability to train on a dataset which is storable on a personal computer and doesn't take too much time to load into free version of Google Colab. While the intended function of the application is to process video, an image-only training dataset is sufficient for the hand-recognition task.

The downsampled images, too, allow for faster training with similar accuracy, based on the results of our primary exploration. In our exploratory processing using MediaPipe Hands, to improve speed, we resized each frame to 320×240, reducing computation while retaining gesture fidelity. We recorded the latency of each frame and computed the average, minimum, and maximum processing times. As a result of this optimization, we achieved an average latency of around 30 ms per frame, a significant improvement from the original 60 ms latency reported in default MediaPipe usage on higher-resolution input. This confirms that downscaling input resolution can halve the processing time while still enabling useful landmark detection; an important insight for designing low-latency, real-time virtual instruments.

Figure 2 has the five gestures we used (Call, Fist, Like, Palm, and One - a reduction from 18 original gestures in HAGRID), along with the number of photos of each (between 7000 and 12000, for a total of 53702 samples). We chose the five gestures because of their differentiability and because they represent a broad range of possible motions representable with the eleven tracking points. To ensure our gestures were significantly different, we compared their embeddings in MobileNet V2 and graphed them via T-SNE (see Figure 3). MobileNet V2 is-as far as we can tell-a good approximation of the feature-detection (point-detection) layers of MediaPipe Hands, so we were confident in our dataset for the training of our gesture recognition model.

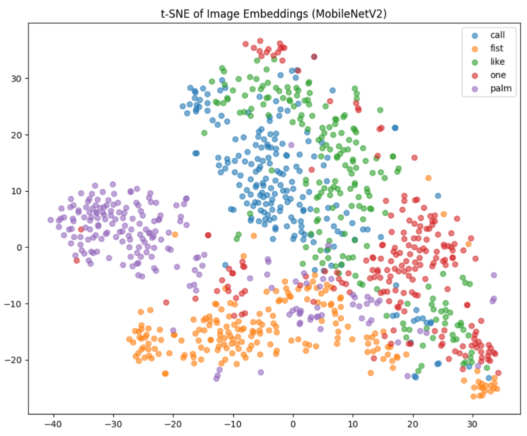

Figure 3. T-SNE embedding of images in MobileNetV2

Finally, we needed to shape our dataset into a format compatible with our pipeline. The HAGRID annotations provided gesture labels, user IDs, and bounding boxes, which we preserved in our metadata. We then used MediaPipe Hands in static image mode to generate 21 landmark coordinates for each image, storing the results in a consolidated .parquet file. From these, we produced two versions of the input: one with all 21 landmarks and one with 11 selected points (removing [1, 2, 6, 7, 10, 11, 14, 15, 18, 19]; see Figure 1) for a smaller and faster model. Unlike MediaPipe's internal training set and gesture layer, which uses fixed heuristics and an opaque gesture vocabulary, our dataset is lightweight, user-labeled, and built for flexible training. Compared to the training data used by MediaPipe, our dataset is lower in image resolution, restricted to five clearly distinguishable gestures, and supports a reduced landmark set. These design choices enable faster training, reduced model complexity, and optimized real-time inference performance for our musical control application.

2. Methods and Experiments

2.1. Methods

MediaPipe Hands, developed by Google, is not fully open-source as the core model weights are not available but its architecture (palm detection ⇒ bounding-box cropping ⇒ hand landmarking ⇒ gesture detection and rectangle generation) is described in detail in accompanying publications. This structure is illustrated in Figure 4.

We used MediaPipe Hands as the foundation for our system, leveraging its palm detection and hand landmarking modules to extract 21 3D hand landmarks from images. Since these models are accessible through the MediaPipe API, we did not attempt to recreate the landmarking step. Instead, we focused on building and training a custom gesture classification layer that operates on MediaPipe's landmark outputs.

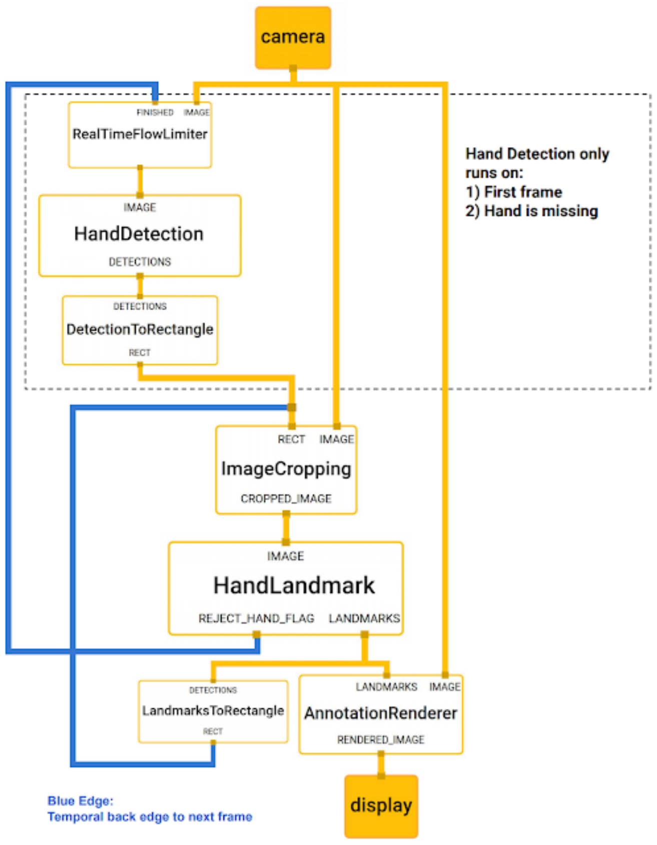

Figure 4. Google's MediaPipe architecture

Our dataset consisted of approximately 53,000 images sampled from the downsampled HAGRID-500k (384p) dataset. We ran each image through MediaPipe in static image mode to annotate the hand with 21 landmarks, which we stored alongside gesture labels in a .parquet file. From this data, we trained two versions of a gesture classification model: one using the full 21-point landmark vector (63 dimensions) and another using a reduced 11-point subset (33 dimensions). Both models share the same architecture and were trained from scratch on our custom gesture set.

Given the small number of distinct gestures in our dataset, we hypothesized that a dedicated, lightweight gesture recognition model could achieve high accuracy with significantly lower computational cost. Rather than relying on MediaPipe's internal gesture logic, we trained our own classifier on landmark vectors, making gesture detection an explicit and customizable part of our pipeline. We briefly considered approaches that would bypass the hand landmarking step entirely and operate directly on image crops using a CNN, given early experiments showed gesture classification accuracy above 90%. However, this added unnecessary complexity compared to the performance of our landmark-based method.

However, our gesture-detection layer was much faster and lighter-weight when built on the reduced output (11 landmark points) provided by MediaPipe's HandLandmark module. Training was also significantly faster with this reduced input, while still achieving high accuracy. We considered using Vision Transformers or other modern architectures, but the upstream MediaPipe pipeline—palm detection, internal cropping, and landmark extraction—already produces a compact and well-structured representation of hand pose. Figure 3 shows that our five gesture classes are clearly separable in feature space after landmark extraction, even without complex encoders. We trained a Multi-Layer Perceptron (MLP) on the flattened landmark vectors, which proved accurate, easy to implement, and fully customizable. This approach eliminated the need for manually defined gesture heuristics and allowed us to train on only the gestures relevant to our musical application.

With our gesture recognition layer, we considered adding a temporal delay mechanism, similar to the logic used in MediaPipe's hand detection stage, where predictions are throttled across frames. The idea was to avoid running the classifier on every single frame to save computation. However, our gesture model was already lightweight—trained on only five distinct gestures—and we observed no meaningful speedup from frame skipping. We also experimented with a "reset" mechanism using the fist gesture, where new gestures would only be detected after first returning to a neutral fist pose. This was conceptually inspired by MediaPipe's use of hand disappearance as a reset condition, but in practice it introduced unnecessary friction: users had to deliberately reset between gestures, which disrupted fluid control. As a result, we opted for continuous gesture recognition without explicit temporal gating or manual resets, but optimized it for musical applications, as detailed below.

Finally, we created a music application on top of the gesture recognition layer. Each recognized gesture was assigned a unique MIDI Control Change (CC) number between 21 and 25. While the MIDI CC protocol is typically used for sending continuous parameter values (ranging from 0 to 127), this system used it in a binary fashion to simulate an on/off switch. When a gesture was detected, a value of 127 was sent; otherwise, a value of 0 was transmitted to indicate the gesture was no longer present. These MIDI messages were sent using the rtmidi Python library, which provides an interface for communicating with MIDI devices or software ports. The data was received in Max/MSP, a visual programming environment for audio and media processing, where each MIDI CC value was routed to control a specific audio effect parameter, like pitch shift or reverb intensity. The outputs of these effects were then combined and sent to the digital-to-analog converter (DAC) for playback.

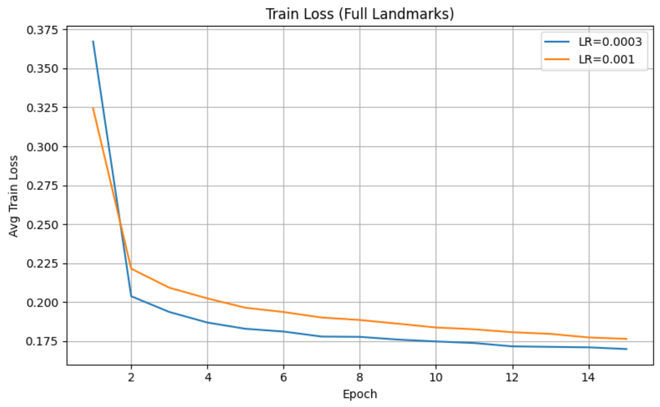

Figure 5. Training the full-size (21 landmark) model.

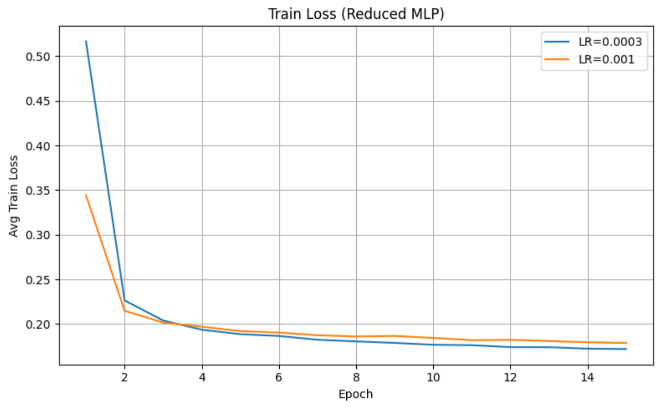

Figure 6. Training the reduced-size (11 landmark) model.

2.2. Experiments

Our early experiments (discussed in Section 1.3) used the standard MediaPipe Hands pipeline on downsampled static images from the HAGRID dataset. These tests confirmed that lower-resolution input (e.g., 384p instead of 1080p) retained sufficient fidelity for accurate landmark extraction, and also hinted that lighter-weight downstream models could still operate effectively. This motivated our decision to explore architectural modifications, especially given that minimizing latency is essential for responsive virtual instrument control.

The main comparison that we made was the effects of reducing the number of landmarks needed to classify a gesture. We remove the two middle landmarks for each feature (Section 1.3), reducing 21 landmarks to 11. The rationale for the removal of these features depended on our gesture set; that is, the gestures we chose were significantly differentiable (Figure 3) even with those features (the middle joints of the fingers) removed, as each of the gestures used different sets of full fingers and could be differentiated with fingertips alone.

We performed a learning rate sweep from 3×10^{-4} to

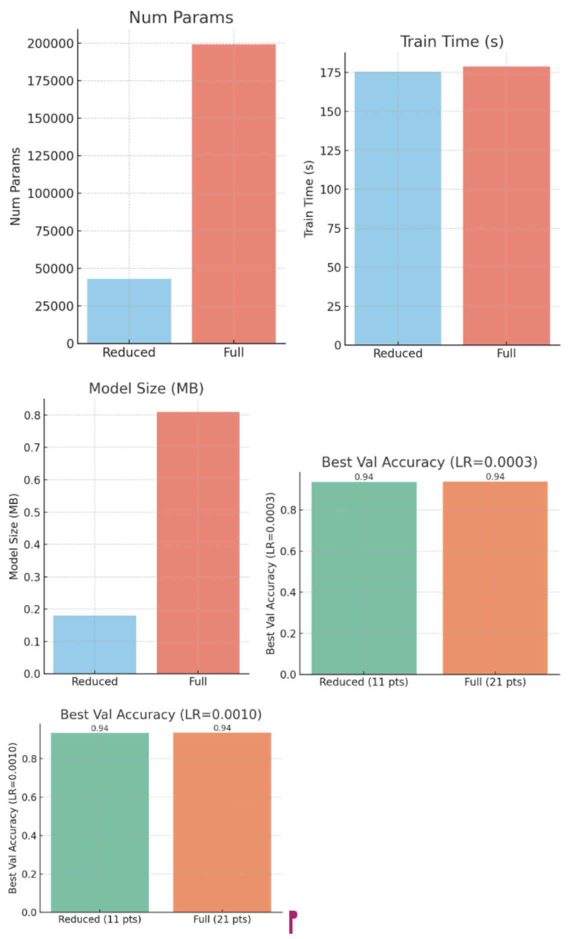

Figure 7. Comparison of accuracy and model size between reduced and full-size model.

1×10^{-3} and found that both model variants converged reliably across the range. Most importantly, the reduced landmark model achieved validation accuracy comparable to the full model, as shown in Figures 5 and 6. These results validated our hypothesis that a significantly downsized model could still deliver strong classification performance without compromising speed.

After this promising result, we wanted to compare our full and reduced models of the gesture-recognition layer in more detail. Considering the real-time and local-first applications of this model, we evaluated the full and reduced landmark gesture MLPs based on the number of parameters, total training time, model size (MB), and best validation accuracy (Figure **??**). The validation accuracy was measured using 5-fold cross-validation. Inspecting the reduced model of the gesture-recognition layer, we found a significantly smaller model, with about half the number of parameters (28,000 vs 58,000) and a quarter of the model size (0.2MB

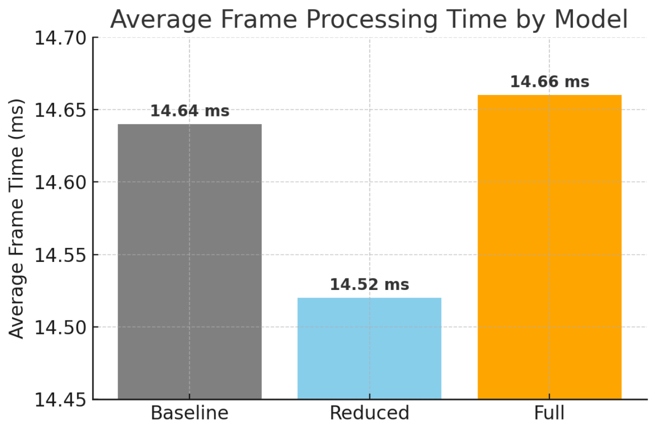

Figure 8. Comparison of accuracy and model size between reduced and full-size model.

vs 0.8MB). We also saw nearly identical validation accuracy (0.94 at learning rates of both 0.001 and 0.0003) for both the reduced and full-size models. The training time was comparable, since both models were trained for the same number of epochs and folds, but the reduced model trained slightly faster due to fewer input features. Overall, the reduced model is significantly lighter and more interpretable, without sacrificing functionality or accuracy.

From there, we tested the latency of video frame gesture recognition across MediaPipe, the full landmark gesture MLP, and the reduced landmark gesture MLP. We measured the performance of our gesture recognition system by processing a 762-frame video in headless mode across three configurations: baseline (MediaPipe only), reduced model (11 landmarks), and full model (21 landmarks) (see Figure 8). For each frame, we tracked MediaPipe processing time, model inference time (when applicable), and total frame processing time. The results revealed that MediaPipe hand detection is the primary bottleneck, taking 12.1–12.3 ms per frame, while model inference adds only 0.11–0.12 ms of overhead. The reduced model achieved the best performance with 68.86 FPS (14.52 ms total), slightly outperforming both the baseline (68.29 FPS, 14.64 ms) and full model (68.23 FPS, 14.66 ms). All configurations maintained real-time performance ($>$ 68 FPS), confirming that the reduced model architecture has an optimal balance between speed and functionality.

We did not expect the reduced model to achieve nearly identical validation accuracy compared to the full landmark model despite having significantly fewer parameters. Initially, we anticipated some drop in classification quality due to reduced input dimensionality. However, the five gestures we trained on remained easily distinguishable, even with just 11 landmarks. As for the latency results, we saw that the full land- mark model had a higher inference time compared to the baseline MediaPipe model. We thought that training on fewer gestures to begin with would improve the inference speed.

3. Conclusion

In conclusion, we found that our method of filtering landmarks to create a more lightweight local model was effective for real-time gesture recognition. This approach reduces computational overhead, making it easier to deploy the model on resource-constrained platforms and at a lower cost.

In terms of improvements in latency, we did not see as significant of a result. There were fractions of a millisecond gained in inference with the reduced landmark model, but this is unnoticeable by a human. Only on tremendous scale (very computationally expensive musical effects) would this improvement in latency be significant.

Looking ahead, we aim to explore more selective filtering of landmarks on a gesture-specific basis, as well as investigate whether additional downsampling could further reduce inference time. Since MediaPipe's HandLandmark module dominates overall processing time, we are also motivated to explore alternatives to the 21- or 11-point detection systems. In particular, we are interested in testing the feasibility of a direct palm-recognition-to-gesture-classification pipeline as discussed in Section 2.1. Ideally, we could collapse the landmark extraction and gesture recognition into a single, efficient classification layer without sacrificing accuracy.

4. References and Acknowledgments

4.1. References

References

[1] Bazarevsky et al. "MediaPipe Hands: On-device Real-time Hand Tracking." Google AI Blog, 2019.

[2] Pavlakos et al. "HaMeR: Hand Mesh Recovery from Occluded Images." CVPR 2022.

[3] Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." CVPR 2018.

[4] Oudah, Munir et al. "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques." Journal of imaging vol. 6,8 73. 23 Jul. 2020, doi:10.3390/jimaging6080073

4.2. Acknowledgments

Sid Yu: I contributed to the idea of filtering landmarks to make gesture classifiers more lightweight. I constructed and annotated the dataset used to train both gesture models. I helped design and develop the models and created evaluations to compare all models.

Gayatridevi Kamat Tarcar: I graphed our annotated landmark data to explore its distribution and underlying patterns, built the baseline gesture recognition model, and extended the system to work with live webcam input for real-time gesture detection. I also designed and implemented the full music interaction protocol—mapping detected gestures to MIDI Control Change messages and building the corresponding Max/MSP patch to control vocal effects such as reverb and pitch shifting. The system was designed to be extensible, with the potential to support additional sound effects. I also wrote the final draft of the report.

Owen Jung: I chose and downloaded the downsampled HAGRID dataset that formed the base of our training data (experimenting with many datasets), as well as . I also contributed to the idea of filtering landmarks to make gesture classifiers more lightweight (along with other unfollowed paths described in Section 2.1). Finally, I wrote initial draft of the paper and formatted it using LaTeX.