A.I.R.G.T.R. – Artificial Intelligence for Real-time Gesture-based Tonal Rendering

Shane Mion
Stanford University
Department of Computer Science
smion@stanford.edu

Jacob Rubenstein
Stanford University
Department of Mathematics
jacobr1@stanford.edu

Abstract

We explore the use of computer vision and deep learning to enable hand-based gesture recognition for musical interaction, aiming to create more accessible and intuitive tools for musical expression. As an illustrative use case, we create a vision-based system that mimics guitar playing, interpreting right-hand strumming motions and finger counts to trigger musical output, while using left-hand gestures to modulate sound characteristics such as pitch or echo. To recognize right-hand finger configurations, we cast the task as a five-way image classification problem and evaluate multiple convolutional neural network architectures (MobileNetV2, ResNet18, EfficientNet-B0) on a custom dataset that we create using computer vision. We further assess how data augmentation and environmental variability affect generalization. For the left hand, we develop a complementary pipeline that leverages machine learning on temporal sequences of hand landmarks to recognize dynamic gestures, enabling real-time control over expressive musical parameters such as pitch bends, echo effects, or tonal shifts. Together, these components form a real-time pipeline for camera-based musical control, demonstrating a promising direction for low-cost, intuitive alternatives to traditional instruments.

1. Introduction

Access to music creation tools is often limited by cost, complexity, and availability. High-quality instruments, MIDI controllers, and other digital audio hardware can be prohibitively expensive, difficult to learn, and requires a general barrier to entry that is too high for many. Even when tools are accessible, they often require specialized knowledge or years of training to use effectively. This project aims to lower those barriers by turning a webcam, something nearly everyone already has, into a powerful musical interface. By using simple hand gestures, users will be able to produce and manipulate sound in real time, without needing prior experience with traditional instruments or music software. In doing so, the system opens up new possibilities for learning, experimentation, and play, making music creation more inclusive and approachable.

To make this possible, we're building a system that interprets hand gestures using neural networks and maps them to musical actions. The technical challenge lies in ensuring that this interaction feels natural, responsive, and expressive. The system must detect gestures accurately in real time using just a single camera feed, and translate those gestures into meaningful musical output. By focusing on intuitive interaction and low-cost implementation, we hope to create a tool that's both technically interesting and broadly accessible.

Our approach achieves this through a combination of deep learning and computer vision. We evaluate core components of our system across both classification accuracy and real-time performance. For right-hand finger recognition, we frame scale degree selection as a five-class image classification task over the A minor pentatonic scale, achieving over 99% test accuracy using MobileNetV2 on in-domain data. Left-hand openness controls play mode $\in$ note, chord, with a simple landmark-based classifier reaching 95% accuracy. Octave class $\in 0, 1$ is set using left-hand position, determined through a heuristic angle-based rule that performs reliably in live settings. Although not deep learning-based, our Mediapipe-powered strum detection effectively captures downward hand motion to trigger notes in real time. All gesture outputs are fused and translated into MIDI messages, enabling seamless control of synthesis and playback within Max/MSP. Together, these results demonstrate the feasibility of our approach and highlight key challenges and design decisions in building real-time, vision-based musical instruments.

2. Related Work

The intersection of computer vision, deep learning, and musical expression is one that has been well explored, with various intuitive interfaces for musical interaction emerging. Our work builds upon several key areas of existing research, while introducing novel and unique combinations of methods to fulfill our specific goals.

2.1. Computer Vision and Hand Pose Estimation

The recent advancements in computer vision have been key for 3D hand pose estimations, work that is crucial for musical gesture recognition applications. Pavlakos et. al developed HaMeR [2] (Hand Mesh Recovery), a transformer-based approach that reconstructs hands in 3D from monocular input with significantly improved accuracy and robustness. HaMeR uses a large-scale Vision Transformer architecture and achieves state-of-the-art results on standard 3D hand pose benchmarks. While our system could potentially benefit from such precise 3D hand reconstruction, we opted for a more lightweight approach using 2D landmark detection to ensure real-time performance on consumer hardware.

There is also the jo0707/midiGesture project [1] demonstrates a real-time gesture recognition system that converts hand gestures into MIDI signals using MediaPipe and OpenCV. This approach focuses on simple gestures like index finger pointing and thumb-finger contact for note triggering, which shares similarities with our finger counting classification approach but lacks a lot of the sophisticated musical expression capabilities we implement through left hand modulation.

2.2. Machine Learning and Musical Applications

The development of gesture-controlled music systems has accelerated through advances in computer vision and machine learning, with researchers exploring diverse approaches to bridge physical movement and digital sound generation. A seminal contribution comes from Yang et al.'s MuGeVI system [4], which employs deep neural networks for hand keypoint detection and maps gestures to musical parameters through Open Sound Control (OSC) integration with Max/MSP. While sharing our system's vision-based approach and OSC/Max integration, MuGeVI does not share our guitar-style interaction paradigm that defines our work, which specifically decouples right-hand strumming detection from left-hand modulation to mirror established instrumental techniques.

There are several projects that exist that demonstrate the versatility of MediaPipe for basic music control applications. Sgvkamalakar's Hand Gesture Music Player [3] implements real-time gesture recognition for playback functions, using open palms to play/pause and thumb-index finger separation for volume control. This is a type of system that exemplifies the growing accessibility of vision-based music interfaces but focus on discrete playback controls rather than the expressive performance capabilities enabled by our hybrid architecture. Our approach advances beyond simple trigger mechanisms through finger-count classification using convolutional neural networks (MobileNetV2, ResNet18, EfficientNet-B0) for discrete scale degree selection, combined with temporal gesture recognition for dynamic parameter modulation, which is a dual strategy that supports both melodic precision and real-time manipulation.

2.3. Bridging to Our Work

Our architecture synthesizes these advancements while addressing their limitations through three key innovations: 1) Guitar-inspired interaction design separating strumming detection (via MediaPipe motion tracking) from finger position classification (through CNN architectures), 2) Hybrid static-temporal processing that combines MobileNetV2-based finger-count recognition with heuristic angle rules for octave selection, and 3) MIDI parameter mapping optimized for real-time responsiveness on consumer hardware The integration of both classification paradigms (discrete finger counts via CNNs and continuous strumming detection via landmark trajectories) with additional music capabilities creates a musically intuitive interface that surpasses the powers of systems focusing solely on singular gesture types.

Unlike systems that require specialized hardware or multiple cameras, our approach uses only a standard webcam, making it highly accessible. While other gesture-based music systems focus on general playback control or abstract musical interaction, our system provides structured musical expression through pentatonic scale mapping and chord/note mode selection. Furthermore, our integration of both temporal gesture recognition (for strumming) and static pose classification (for finger counting) creates a hybrid approach that combines the benefits of both methodologies. This differs from systems that rely solely on either continuous motion tracking or discrete pose recognition, providing both rhythmic control and melodic expression capabilities within a single unified interface.

3. Data

Our project centers around enabling expressive, two-handed musical interaction using computer vision. Specifically, we work towards interpreting right-hand finger poses and strumming gestures, and left-hand openness and position, to control real-time musical output. Because no publicly available and well curated dataset exists for the specific forms of interaction we were lookin for (in simpler words, "air guitar" gestures using both hands to simulate pitch, play

mode, and octave selection), we created a custom, multimodal dataset tailored to the needs of our system.

3.1. Data Collection Process

We collected synchronized video data using a standard webcam, recording participants as they mimicked guitar playing with their bare hands. The right hand is used for two simultaneous types of expression: (1) strumming gestures, which trigger note events and are detected via wrist movement, and (2) finger configurations, which correspond to musical scale degrees (1–5 in the A minor pentatonic scale: A, C, D, E, G). Meanwhile, the left hand controls two separate musical modifiers: (1) hand openness to toggle between note and chord play modes, and (2) vertical hand position to indicate octave class (low or high). In each session, participants performed predefined gesture sequences that covered the full range of these musical states.

Each frame or short video clip will be annotated with:

- `scale_degree` $\in \{1, 2, 3, 4, 5\}$ corresponding to a note in the A minor pentatonic scale
- `play_mode` $\in \{\text{note}, \text{chord}\}$ based on left-hand openness
- `octave_class` $\in \{0, 1\}$ (low or high), used only when playing single notes

We use MediaPipe Hands to extract hand landmarks and handedness classification, allowing us to segment and crop left- and right-hand regions from the input frames. The cropped right-hand images (used for finger classification) form the core of our neural training set. To facilitate training, we also developed a custom labeling and collection interface to associate image crops with the appropriate class labels and to support incremental dataset expansion.

3.2. Dataset Size and Composition

The dataset we created consists of over 1,000 labeled images of right-hand finger poses distributed across five classes (1–5 fingers), collected from multiple environments and lighting conditions. The dataset for left-hand gestures consists of over 400 labeled images captured from webcam footage and sorted into two classes, open and closed. These images were collected across varied backgrounds and lighting conditions to improve generalization. These images were then used to train a binary classifier for left-hand state recognition. Octave classification is computed dynamically at runtime using the vertical position of the wrist landmark.

Initially, data was collected under relatively uniform lighting and background conditions. Early results from CNN-based classifiers on this dataset were promising in controlled environments but failed to generalize well during real-time use. Specifically, the models struggled to classify right-hand finger poses accurately in live scenarios with varying lighting, motion blur, or non-neutral backgrounds. This revealed a key gap between training and deployment conditions, which we addressed through two major strategies, which were data augmentation and dataset expansion.

3.3. Data Augmentation

To improve generalization and simulate the visual variability encountered during live webcam use, we applied a suite of data augmentation techniques during training. These include:

- Geometric transforms: random horizontal flips and small-angle rotations simulate changes in hand orientation.
- Color transforms: brightness and contrast jitter simulate lighting shifts due to different environments or camera exposure settings.
- Gaussian blur: added to simulate motion blur or low camera resolution, both of which were common during live use and often degraded model performance.
- Random cropping and resizing: implicitly simulated by jittering hand detection boundaries.

We implemented these augmentations using PyTorch's *transforms, Compose* and applied them dynamically during training. Empirically, models trained with these augmentation strategies demonstrated marked improvements in robustness. For instance, finger-count classification accuracy improved by over 10 percentage points in live webcam tests compared to the non-augmented baseline. To be specific, adding blur proved to be especially impactful, as mirroring the camera artifacts introduced by quick hand motion or suboptimal lighting.

3.4. Expanded Data Collection in Diverse Environments

While augmentation helped address many generalization issues, we also found that some failure cases persisted, particularly when the model encountered extreme lighting differences or occluded hand regions. To address this, we conducted an additional round of data collection for both the left and right hand with explicit environmental diversity: collecting samples in front of windows, under yellow vs. white lighting, at night, and with complex backgrounds. We also varied webcam angles and hand sizes (through distance to the camera) to broaden the distribution of visual inputs. This second dataset, collected under the folder structure *data_diff/*, was appended to the existing dataset to train a more environment-robust model. These additional samples were gathered using the same collection pipeline, which captured cropped hand regions via MediaPipe, then labeling them with finger counts. We observed

that the inclusion of these expanded samples, when combined with data augmentation, led to significantly improved live performance, particularly in environments such as low-light strumming or highly blurred frames. It also reduced model overfitting, as reflected in closer alignment between validation accuracy and live classification behavior.

4. Methods

Our approach to building AirGtr centers on transforming video input of gestures from each hand into expressive musical control signals. We break the system down by hand, treating the right and left hands as separate pipelines with distinct tasks and models. For each hand, we explored multiple technical approaches, including rule based logic and deep learning classifiers, and iteratively refined them in response to both offline and real-time performance. This section details the models, preprocessing steps, and system design choices that we employed for each hand, highlighting how they draw on key ideas from the course, including the way that convolutional networks and computer vision intersect. By combining traditional gesture engineering with modern learning-based techniques, we aimed to create a system that is both responsive and robust in capability during live interaction.

4.1. Right Hand Methods

Our right-hand processing pipeline is responsible for two key musical control tasks, which are detecting strumming events and classifying finger poses to determine scale degrees. We began by testing a rule based method grounded in MediaPipe hand landmarks before transitioning to a data driven approach using a convolutional neural network (CNN). This section outlines both approaches in detail and motivates the shift toward neural methods.

4.1.1 Rule-Based Strumming Using Hand Landmarks

Our right hand pipeline began with a rule based approach using MediaPipe's 21-point hand landmark model. Strumming corresponds to rapid downward motion of the right wrist, and this pattern proved to be cleanly separable using simple kinematic features. Specifically, we tracked the y-coordinate of the wrist landmark across consecutive frames and computed a finite difference between these frames to estimate vertical velocity. When this velocity exceeded a threshold of 0.08 units per frame, a strum event was triggered, which is when a sound would be played during the live experiment. To prevent false positives and double-triggering, we implemented a temporal cooldown of 400ms between consecutive strums. This thresholding strategy allowed for real-time responsiveness and maintained high precision under various conditions, including fast gestures and variable lighting.

Figure 1. captured as 1 finger during a strumming motion

Figure 2. captured as 3 fingers during a strumming motion

Figure 3. captured as 5 fingers during a strumming motion

4.1.2 Different Finger Classification Methods

While our strumming detection pipeline remained rule-based throughout development, leveraging MediaPipe's reliable hand landmark tracking and a simple velocity-based trigger, we decided to take a more comprehensive approach with respect to the classification of right-hand finger poses. To establish a baseline, we built off of the above rule based system to implement a rule based classifier. Our approach consisted of distinguishing between straight and bent fingers using landmark comparisons. For each finger other than the thumb, we compared the y-coordinates of the fin-

gertip and the PIP joint. A finger was considered extended if the tip landmark lay significantly above the PIP joint, by more than a 0.02 normalized units, indicating an upward posture in the image plane. The thumb required some special handling due to its lateral orientation, so we tried to scale and normalize using the thumb's unique joint.

However, we also utilized our learnings from class to evaluate other robust alternatives. We assumed that a neural network trained on labeled images of hand poses could more effectively handle noisy input and diverse visual conditions. Instead of relying on geometric heuristics that can easily vary and be mis-measured, this model would learn implicit visual cues that generalize across users, lighting conditions, and camera perspectives.

We began by evaluating a range of convolutional architectures for image-based classification, including ResNet18, MobileNetV2, and EfficientNet-B0. These models were initialized with ImageNet-pretrained weights and fine-tuned on a custom dataset of cropped hand images, each labeled with a corresponding finger count (1–5). Crops were extracted from full frames using bounding boxes inferred from MediaPipe landmarks, with a margin to ensure the entire hand was included. During training, we also decided to experiment with multiple optimization strategies. We tried both Adam, with its adaptive learning rates and rapid convergence, and also stochastic gradient descent (SGD) with momentum, which we know takes longer to converge but may have higher accuracy.

4.2. Left Hand Methods

Our left-hand processing pipeline also supports two forms of musical modulation, selecting octave range and determining whether a chord or single note should be played. Similar to the right-hand system, we experimented with both rule-based and learning-based techniques. Octave control was implemented through a simple rule-based method using hand height, while hand openness was used to distinguish between chord and note modes, and was addressed through a data-driven image classification approach using convolutional neural networks. The following subsections describe the logic and implementation of each method in detail.

4.2.1 Octave Selection via Wrist Height

To determine which octave class to play from, we used the vertical position of the left hand relative to the camera frame. because MediaPipe returns normalized landmark coordinates in the image plane, we directly extracted the y-coordinate of the wrist landmark. If the wrist's y-position exceeded our set threshold, the gesture was interpreted as belonging to the lower octave. Otherwise, it was treated as a high-octave gesture. This rule-based system provided a very simple and easy, yet robust, mechanism for encoding octave shifts without requiring machine learning. It proved stable across different users and lighting conditions, as long as the hand remained clearly visible to the camera.

4.2.2 Open vs Closed Classification via CNNs

To determine whether the left hand was open or closed during a strum event, we trained a binary image classifier on a dataset of cropped left hand images. Our custom dataset was collected using MediaPipe to isolate and crop hand regions from live webcam input, and each image was manually labeled as either open or closed. We trained multiple convolutional neural network architectures from scratch (with weights=None) to compare performance across model families. Specifically, we chose to evaluate MobileNetV2, ResNet18, and EfficientNet-B0 using a shared training loop and hyperparameters.

We evaluated MobileNetV2, ResNet18, and EfficientNet-B0 as they represent different potential good designs. MobileNetV2's might minimize computational cost, ResNet18's skip connections enable fast convergence with minimal parameters, and EfficientNet-B0's generally achieves strong accuracy-efficiency trade-offs. All three are lightweight enough for quick real-time inference on consumer hardware.

Each model was trained for 10 epochs on a dataset augmented with random horizontal flips, color jitter, and resizing to 128×128 resolution. We used the Adam optimizer with a learning rate of 1e-4 and a batch size of 32, and applied cross-entropy loss for supervision. To ensure comparability, each model was evaluated using the same dataset and training loop. Training loss per epoch was recorded and plotted for all three models.

5. Experiments and Results

We conducted a series of experiments to evaluate the effectiveness and real-time viability of the system we created, focusing on both rule based baselines and learning based models.

5.1. Right Hand Results

We evaluated our right-hand pipeline through a series of experiments aimed at quantifying the effectiveness of different finger-count classification strategies. Our goals were to establish a baseline using a rule-based classifier, and then compare the effectiveness of CNN architectures and optimizers.

5.1.1 Rule Based Finger Classifier

While this approach, as described in our methods section, was intuitive and interpretable, it underperformed signifi-

cantly in practice. On the 65% of test images where MediaPipe successfully identified a right hand, **the rule-based classifier achieved only 40.64% accuracy** (115 correct out of 283 total). Because this method could only hope to work if MediaPipe was fully able to detect a hand, there were 35% of cases where this couldn't even happen, so a prediction was not even possible. When the predictions did happen, the reasons failures were so common was sensitivity to occlusion, side views, and lighting conditions, which made the geometric rules brittle and prone to misclassification. These results validated the reasons that we also implemented a learning-based approach.

5.1.2 CNN Training and Evaluation

We trained a series of CNN classifiers to predict finger count directly from cropped hand images. Each crop was extracted using MediaPipe landmark bounding boxes and resized to 128×128 pixels. We performed an 80/20 train-test split and tracked training and validation performance across 10 epochs.

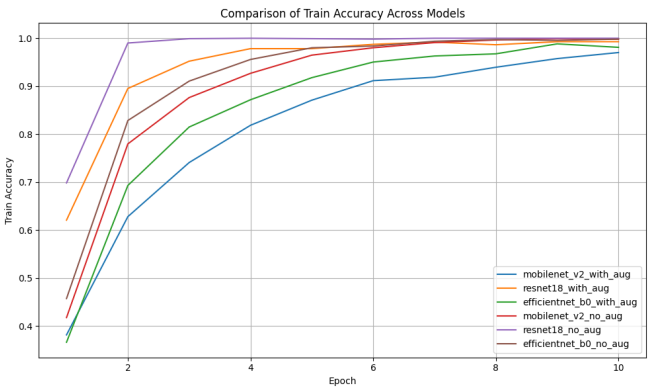

Figure 4. Training accuracy comparisons among model and augmentation combinations

To better understand the learning behavior of each architecture, we tracked training accuracy over 10 epochs for all model and augmentation combinations (Figure 4). All models had relatively good convergence, which makes sense given the simplicity of the classification task, the use of pretrained ImageNet weights, and the clear visual separability between the five finger-count classes in the dataset. With this being said, ResNet18 without augmentation converged the fastest and achieved the highest training accuracy across all models. This rapid learning can be attributed to the ResNet architecture's use of residual connections, which facilitate optimization by enabling gradient flow through deeper layers. Combined with its relatively high capacity and pretrained ImageNet initialization, ResNet18 was able to memorize the training set more quickly than either MobileNetV2 or EfficientNet-B0.

Taken together, these results emphasize that **dataset diversity via environment expansion in the dataset was more impactful than augmentation** in accelerating convergence. Augmentation alone introduced more noise during training, making the optimization landscape harder to navigate. By contrast, real-world samples, meaning collected under varied lighting, background, and camera angles—provided useful priors that better matched live test conditions while still allowing fast learning. As discussed in the data section, we quickly found that our models were almost no better than random guessing in live scenarios with varied lighting or different backgrounds. Once we expanded our set of environments, augmentation, as showed in the graph, was proved to no longer add benefit.

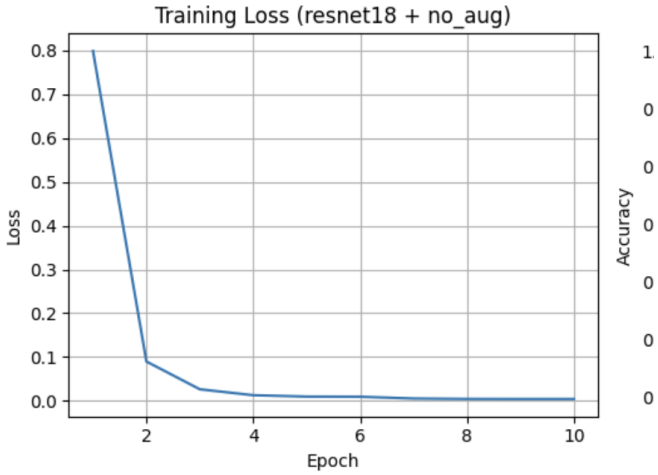

Figure 5. Training loss for our quickest converging model

As shown in Figure 5, the training loss for ResNet18 without augmentation drops steeply within the first two epochs and quickly approaches zero. This rapid convergence is consistent with the architecture's residual learning mechanism and pretrained initialization. The minimal loss plateau after epoch 3 further indicates that the model had effectively memorized the training data, underscoring both its high capacity and the relative simplicity of the classification task. Generally, this behavior also reflects the absence of data augmentation, which would have otherwise slowed convergence by introducing greater input variability. While fast convergence can be beneficial, it also highlights the need for a more diverse training set to ensure generalization, motivating our use of expanded data collection across environments. Using our robust datasets, even with the worry of overfitting, the model works well, which probably speaks to the relative non difficulty of the classification task.

5.2. Left Hand Results

To evaluate left hand gesture classification, we trained MobileNetV2, ResNet18, and EfficientNet-B0 from scratch (no weights at the start) on our binary dataset of open vs. closed hand images. Each model was trained using the same augmented dataset, loss function, and optimizer configuration (Adam, 1e-4 learning rate). Figure 6 shows the training loss per epoch across all models, allowing for a direct comparison of learning behavior.

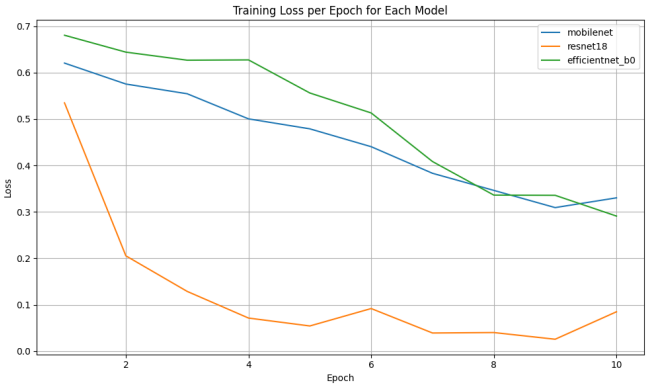

Figure 6. Comparison of losses between models trained from scratch on left hand data

We can quickly observe that ResNet18 converges the fastest and achieves the lowest loss across nearly all epochs, suggesting that it is best suited for this binary task under limited data conditions. This is consistent with ResNet's higher representational capacity and skip connections, which allow deeper layers to optimize effectively without vanishing gradients. In contrast, MobileNetV2 and EfficientNet-B0 exhibit a more gradual reduction in loss, suggesting that they have a slower convergence and potentially less efficient use of training data in this context.

It is important to note that all models demonstrate consistent downward trends in loss, confirming that the open vs. closed classification task is learnable with relatively modest data and light augmentation. The gap in convergence speed also supports our choice to use ResNet18 in the live pipeline, where fast and confident predictions are needed during real-time strumming.

In terms of robustness, we found that models trained with augmentation performed well under varied lighting and backgrounds, with the ResNet18 classifier achieving 95% test accuracy in live webcam tests. While training loss alone does not capture generalization fully, the consistent trends in Figure 6 validate our training loop and model architecture choices. Overall, these results confirm that the left-hand CNN module is both performant and deployable in real-world, latency-sensitive scenarios.

Now that we established that ResNet18 consistently outperformed MobileNetV2 and EfficientNet-B0 in both convergence speed and final accuracy on the left-hand openness classification task, we conducted a focused hyperparameter search to further optimize its performance. Specifically, we explored how different combinations of learning rate, batch size, and optimizer affected generalization. This search was motivated by the need to deploy a lightweight but reliable model in real-time settings, where training budget is limited but robustness matters.

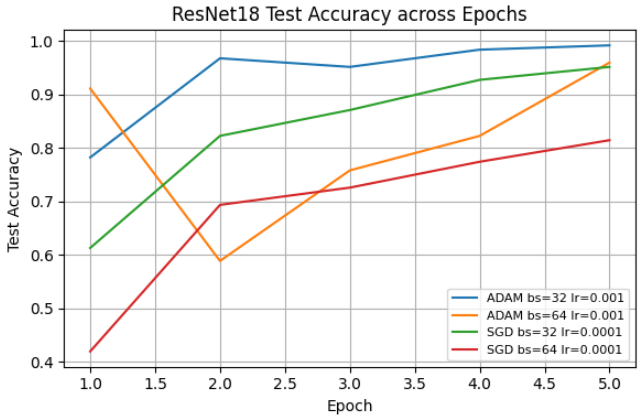

Figure 7. Comparison of different hyperparameter combinations

The results of our hyperparameter grid search are shown in Figure 7, which plots ResNet18 test accuracy across epochs for several optimizer/batch size/learning rate configurations. The most stable and highest performing configuration was Adam with a batch size of 32 and a learning rate of 1e-3, which reached nearly 99% test accuracy by epoch 5. In contrast, larger batch sizes showed less stable training dynamics, particularly for Adam, which dipped early before recovering. This is consistent with known behavior where larger batch sizes can lead to poorer generalization unless compensated for by lower learning rates or longer training.

Interestingly enough, the SGD configurations improved steadily over time, but never matched the performance ceiling of the best Adam setup within the 5-epoch window. This aligns with the idea that Adam's adaptive learning rates offer faster convergence, especially in low data regimes like our binary classification task. The combination of pretrained ResNet18 features and well-tuned optimization parameters proved highly effective, validating our final choice to use Adam, with lr=1e-3 and bs=32 in our deployed system.

5.3. Constructing the Air Guitar Pipeline

After building and evaluating the right and left hand pipelines independently, we integrated the components into a unified gesture to music system. The final AirGtr pipeline processes video input frame-by-frame, using MediaPipe to track both hands and extract hand landmarks. These signals

are then routed through separate logic for each hand, with the right hand controlling pitch and note triggering, and the left hand controlling chord or note decision and octave. By combining both sources of input, we created a twohanded interface capable of expressive control over musical output. The decisions discussed previously are fused into a compact representation of the current gesture state, resulting in a pitch class, a play mode, and an octave modifier.

This output is then sent to Ableton Live using MIDI messages. Each time a strum is detected, a Note On event is issued with the appropriate pitch and velocity, computed from the CNN prediction and left-hand context. Chords are synthesized by triggering multiple notes simultaneously, while octave and mode settings alter the mapping of gestures to sounds. This real time connection between computer vision and audio synthesis enabled the system to feel instrument-like, with decent latency and nuanced expressive range. The result is a prototype that transforms silent air gestures into musically meaningful performances, something verified through our supplementary videos showing music being played on our interface.

6. Conclusion

In this paper, we presented **AirGtr**, a real time, vision based system for gesture-driven musical expression. Our system enables two handed musical interaction using only a standard webcam, with the right hand controlling pitch through finger count classification and note triggering through strumming motion, and the left hand modulating chords/notes and octave. Through a combination of computer vision with rule based logic and deep learning models, we demonstrated that intuitive and expressive musical control is achievable without specialized hardware.

Our key results show the following.

- **ResNet18 consistently outperformed other architectures** (MobileNetV2, EfficientNet-B0) across both the finger-count and open/closed classification tasks, exhibiting faster convergence and higher accuracy, even when trained from scratch.
- **Environmental diversity in the dataset proved more effective than augmentation alone**, with expanded data collection dramatically improving generalization to real-world webcam conditions.
- **Hyperparameter tuning further boosted performance**, with the optimal ResNet18 configuration (Adam, lr=1e-3, bs=32) reaching nearly 99% test accuracy in left-hand classification.

These findings validate that relatively lightweight CNNs can be paired with good training data and tuned carefully, and can then deliver robust performance for gesture-based interaction tasks. We also explore the trade-offs between rule based and learned approaches, showing where each offers simplicity, generalization, or control.

Despite these strengths, the system does have limitations. Gesture recognition performance really degrades under poor lighting or fast motion, and MediaPipe's landmark detection does occasionally fail during rapid movement which leads to dropped inputs. Additionally, while our finger classifier generalizes well across environments, it remains limited by the size and diversity of the training dataset. Another area for future work is refining left hand recognition beyond rule based logic using trainable models.

For future work, there are several promising directions. First, extending the gesture vocabulary beyond five discrete finger poses could allow for richer melodic control or dynamic velocity mapping. We clearly had lightweight models on small datasets, and hope this serves as a proof of concept for more advanced systems. In addition, deploying the pipeline in mobile or embedded contexts could further democratize access to music.

Ultimately, AirGtr proves that we can combine computer vision and deep learning, and expand what it means to play an instrument.

References

[1] jo0707. midigesture: Real-time gesture recognition system to recognize hand gestures and position then convert them into midi signals. https://github.com/jo0707/midiGesture, 2024. Accessed: 2024-XX-XX. 2

[2] G. Pavlakos et al. Hamer: Hand mesh recovery from a single image. https://geopavlakos.github.io/hamer/, 2023. Accessed: 2024-XX-XX. 2

[3] Sgvkamalakar. Hand gesture music player. https://github.com/Sgvkamalakar/Hand-Gesture-Music-Player, 2024. Accessed: 2024-XX-XX. 2

[4] Y. Yang, Z. Wang, and Z. Li. Mugevi: A multi-functional gesture-controlled virtual instrument. In M. Ortiz and A. Marquez-Borbon, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 536–541, Mexico City, Mexico, May 2023. 2