Hand Gesture Classifier

Project Final Report

Team Members: Ethan Gruening, Owen Harty

Abstract

Using computer vision to determine a person's hand gestures can be a powerful tool in ASL translations and gesture-triggered actions. Utilizing the HaGrid dataset of hand gestures and Google's MediaPipe library, we can train multiple SVM, CNN, and deep learning models to classify 33 different hand gestures. We are using models trained on a feature set of hand landmarks detected within an image of a hand. The feature set is comprised of coordinates of hand landmarks within the HaGrid dataset identified by Google's MediaPipe hand-tracking software. On a live camera feed, we can track the hand movements and give real-time inferences of hand gestures in both the right and left hands. The bulk of our project lies in the machine learning techniques to increase the performance of models. We have concluded that hand gesture recognition using an SVM, SVM with bagging, and SVM with boosting has a 75.77%, 75.83%, and 75.81% accuracy, respectively. We additionally evaluated hand gesture recognition using a 1D-CNN, 2D-CNN, 1D-DenseNet, and 2D-DenseNet achieving 76.86%, 80.84%, 94.20%, 92.09% accuracy, respectively. A deep learning TabNet module was also developed with 93.51% accuracy.

1 Introduction

Various translators exist for all languages worldwide, but limited translators exist for ASL. Our project is important because in the modernization of our technical world, image recognition and computer vision are growing fields to help communication. Integrating machine learning to translate sign language and hand gestures can help increase communication and improve the lives of those who sign.

Image recognition is computationally intensive and requires neural networks and machine learning models to differentiate and classify images. Since our project requires computer vision, machine learning is necessary to achieve accuracy.

Our hand gesture classifier will use computer vision and machine learning to identify left and right-hand gestures in real time accurately. Examples of hand gestures our project will identify are thumbs up, thumbs down, open palms, okay signs, and peace signs. This project could advance gesture recognition and lead to additional workings with ASL signing for translating non-verbal communication.

To train a machine-learning model on hand gesture data, we utilize HaGrid, a dataset compiled with publicly submitted images of 34 different hand gestures. HaGrid's total dataset is 1.5 terabytes with an annotated set of JSON files containing each image's Google MediaPipe 21 hand landmarks, the image's file name, and the gesture classification. The dataset is pre-scrambled and separated, with 76% of the images in the training dataset, 9% in the validation dataset, and 15% in the test dataset.

Using data from the HaGrid dataset, we will train SVM, neural network, and deep learning classifiers and test them against the testing dataset to determine the most optimal module. A major goal in this process is to integrate machine learning techniques such as ensemble learning, bagging, and boosting to compare and analyze the effects on performance. In our testing process, we will report the complexity of the model through its weights, the accuracy through its percentage of correct classifications of the testing dataset, and its training time.

Testing different models and training techniques, starting with bagging and boosting ensemble learning, we expect an increase in performance from the SVM not utilizing bagging or boosting. Additionally, we expect an increase in accuracy when using 1D and 2D Convolutional Neural Networks, and again for TabNet and DenseNet modules utilizing multiple convolutional deep layers.

2 Related Work

Our project is a unique subset of preexisting hand recognition models. Some of the more popular models, such as DeepASL, SignAll, and OpenSign, all focus on translating ASL (hand signs) into text or speech in real-time; however, our project focuses more on universal hand gestures, as specified above.

2.1 DeepASL

The first related work we looked at was Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation (Fang). DeepASL is more complex in the sense that it uses a novel hierarchical bidirectional deep recurrent neural network (HB-RNN), which is designed to capture both long-range and local dependencies in sequential data (it combines hierarchical structures with bidirectional recurrent layers to improve the model's understanding of sequences).

However, our project is similar in identifying the person's hand landmarks in real time to classify the specified hand gesture correctly. In addition, our project and DeepASL support two-handed gesture recognition and the ability to differentiate between similar signs with different meanings (i.e. DeepASL: 'want' and 'what'; Our Project: 'mute' and 'point').

SignAll and other high-end gesture recognition models utilize deep neural networks (DNNs), an artificial neural network with multiple layers of neurons designed to model complex relationships in data. DNNs are a popular model architecture, not just for ASL recognition but for various tasks.

Our project utilizes a less complex architecture, SVMs, but for a good reason. Unlike DNNs, SVMs require less computational resources and can be trained on smaller amounts of data. In addition, SVMs are easier to understand and break down due to their open nature (we can visualize the decision boundaries). Although DNNs perform significantly better with ample data, it made more sense to go with SVMs due to our circumstances.

Despite the differing methodologies between our project and models like DeepASL and SignAll, the end result will ultimately be the same: accurate, real-time recognition of hand gestures. Both approaches aim to classify and differentiate gestures accurately, ensuring effective communication through gesture recognition, whether it's for universal hand gestures or ASL.

2.2 Sign Language Transformers

Other related work involves Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation (Camgoz) which simultaneously executes Sign Language Recognition (SLR) and Sign Language Translation (SLT) tasks. This research expands on prior work by proposing a joint end-to-end model to handle the tasks at the same time. The SLT architecture involves adapting transformer models to process sign language videos directly, where the input is raw video frames. The key innovations that Camgoz and others bring to the table include 2D Convolutional Embedding which converts video frames into spatiotemporal (space and time related information) tokens and a transformer encoder - captures spatial and temporal dependencies from the videos - and decoder - generates spoken English translations from the encoded information.

This transformer project is similar to our work in the sense that we are both trying to translate hand signs, where we are focusing on hand gestures and they are focusing on American Sign Language (ASL). However, our work differs in the execution of said project, where we use live data to process frames to classify hand gestures based on trained SVM/CNN models. The transformer project uses video sequences as input and translates full-sentence ASL using models trained on spatiotemporal relationships.

2.3 Translating ASL via Sensors and ML

The next related work involves an American Sign Language Recognition System Using Wearable Sensors and Machine Learning (Dibba). This research involves capturing hand and finger movements using devices like gloves or wristbands equipped with motion, flex, or inertial sensors. The data from these sensors is processed using SVMs, Long Short-Term Memory (LSTM), or CNNs to recognize and classify different ASL signs. Finally, the original signs are translated into text or speech (English).

This research is related to our work because they are working to classify hand signs, specifically ASL. In addition, they also use SVMs and shallow CNNs to classify the signs, which is what we initially started our project with. However, they differ with how they gather their input (sensors on gloves). This hardware component is a stark difference between our projects, but the machine learning aspect is relatively similar (SVMs and CNNs). Finally, this related work expands on previous work by using sensors instead of images/video because the authors believed it would "remove the potential obtrusiveness of the camera". Essentially, the ASL could be translated with any respect to location instead of being confined to the position in front of a camera.

2.4 Real-Time Sign Language via LSTM

The fourth related work we looked at was Real-Time Sign Language Detection using MediaPipe and LSTM (Rao). The way the system they made works is it extracts features (3D landmarks) using MediaPipe, capturing the spatial configuration of gestures. Next, they use temporal modeling with LSTM via processed sequences of expected landmarks to recognize dynamic sign language expressions. Lastly, the output from the model is passed through a dense layer to classify the gesture into predefined categories.

This is related to our work as the authors are dealing with hand gestures more than actual sign language. In the tests they classify only three signs (hello, thanks, and iloveyou), which is similar to our six class tests which we'll discuss later. In addition, the authors extract frames to make a prediction; however, these frames are not extracted from a live feed, but a video that was inputted. Also, compared to their project, our project deals with 30 more gestures and uses a wide variety of models, but not LSTM which was used in their project. This research expands on previous work, as the authors aimed to "enhance the performance of the current system in terms of response time and accuracy."

2.5 Real-Time Sign Language via Deep Transfer Learning

The final related work we evaluated was Real-Time Sign Language Recognition using Deep Learning Techniques (Wahane). This project uses video input to capture hand regions from the captured frames and applies image processing techniques to isolate and enhance said hand regions from the images, allowing them to capture spatial characteristics of hand gestures. From there, they utilize pretrained models such as SSD, Faster RCNN, and EfficientDet via YOLO to classify the extracted features into their corresponding sign language alphabets or words (they also mentioned they used Google's Inception v3 for the ASL alphabet detection). Finally, based on the predicted letter/word gesture, the system provides the text equivalent.

This project relates to our work as it utilizes hand gestures from captured frames to retrieve a prediction from the model. In addition, they use a webcam to capture the image frames, which is what we are also doing for our project. However, where their project differs is that they do image pre-processing on the captured frames, which we do not. Continuing, they use transfer learning as the basis for their models via YOLO and Google as opposed to our from scratch models. The paper improves on previous work by incorporating individual ASL letters from the alphabet to help new ASL signers learn the alphabet.

In total, all of these related works have provided us with an idea of how to improve these works. We have taken a piece of each research project and combined it into one to make an even better gesture recognition system. For example, we incorporated Deep Learning models from the first paper, 2D CNNs from the second paper, SVMs from the third paper, hand gestures instead of full ASL sentences from the fourth paper, and using live input from a webcam from the fifth paper. Together, these components have motivated us to create our Hand Gesture Classifier, which we will describe in detail next.

3 Your methods name

3.1 Intuition

The intuition of the project requires 3 main modular tasks: creating a tool for feature collection, implementing an OpenCV Live Feed interface, and testing and training multiple model architectures. The integration of these three tasks will allow for an interface where hand gestures are captures live, analyzed, and fed into a model where its output will be displayed immediately.

3.2 OpenCV Live Feed

The live feed interface is an essential component of our design, where we will use the image and video processing library OpenCV to display a live feed to the user's display. The live feed will repeatedly pull image frames from the camera to not only output to the screen, but also have a place to run feature collection and model prediction on the given image frame. The FPS of the program will be displayed on the screen, as well as a class label for the left hand and the right hand.

3.3 Feature Collection

Our approach to training our gesture recognition model involves gathering online images to build a robust dataset. After evaluating several available datasets, we chose HaGRID as our primary training resource. HaGRID contains diverse images featuring people performing hand gestures under different lighting conditions, angles, and backgrounds. This diversity makes it well-suited for training our models, as it helps improve the model's generalization across different real-world scenarios. Since HaGRID provides a large volume of labeled gesture images, we were able to leverage it as the sole dataset for training our models.

Collecting the features from HaGRID requires a multistep process utilizing Google MediaPipe hand recognition. Firstly, the images must be parsed with the Google MediaPipe library. Secondly, the Google MediaPipe library will map 21 (x,y) coordinates to each hand along the fingers and palm. Thirdly, we will take the 21 coordinates and map each x and y coordinate into a feature vector for a 1D map of 42 features. Lastly, we can insert the 42 feature vector values and its known class name into the model as a single training example.

Once the model is trained, we can analyze its live performance by collecting features from a live feed as opposed to static image files from the HaGRID dataset. Steps 1-3 are the same, mapping the hand landmarks and flattening to a feature vector. Then, the feature vector is inserted into the predict() method of the model as an unknown test sample, where it will then output the class name.

3.4 Models

In this section, we'll talk about the learning algorithms and models we used. For the project, we used support vector machines, 1D-convolutional neural networks, 2D-convolutional neural network, 1D and 2D DenseNet modules, and a deep learning TabNet model as our tools to classify hand gestures.

SVMs are a type of supervised learning algorithm that is particularly well-suited for classification tasks due to their ability to create hyperplanes that separate data points of different classes. In our case, the data consists of hand landmarks captured in real-time, and the SVM is trained to classify these gestures into their respective categories.

The 1D and 2D convolutional neural networks are linear convolutional layers. The 1D CNN will be better suited for temporal sequences while the 2D CNN will better handle spatial relations with 2D projections.

Increasing the complexity of our CNNs, we've implemented a DenseNet model to concatenate features from each dense block, which is then entered through a 1D or 2D convolution and pooled before being concatenated to the next layer. The feature concatenation is the primary technique in DenseNet that allows the output to use data from all layers within the CNN to make its final decision.

3.4.1 SVM

The SVM model is trained using a LinearSVC implementation from sklearn, which provides a straightforward method for handling high-dimensional data like our hand landmark features. Once trained, the model is tested on unseen data (also provided by HaGRID), and we evaluate its performance using a confusion matrix, classification report, and decision function scores (provided via sklearn.metrics). These metrics help us understand the model's ability to generalize across various hand gestures and provide insight into areas for improvement. Additionally, we visualize the results via graphs, including feature weights, decision function distributions, and classification metrics, to better understand how the model interprets the data.

In addition to SVM, we also perform bagging and boosting to reduce variance and bias. Bagging is an ensemble method that trains multiple models (in our case, SVMs) independently on different subsets of the training data and then combines their predictions through majority voting.

This technique reduces variance and helps prevent overfitting by using different bootstrap samples or random subsets of the data to train each model. In SVMwBagging.py, our implementation creates an ensemble of SVM models, where each model is trained on a different resampled version of the training data. The final prediction is made by taking the majority vote from all the models in the ensemble. This technique is particularly effective when the base model (SVM) is prone to overfitting on noisy data, as it helps smooth out predictions' fluctuations.

We also perform boosting, which focuses on improving the accuracy of the ensemble by sequentially training models and giving more weight to the misclassified examples from previous iterations. In SVMwBoosting.py, we implement this method by training a series of SVMs, each time focusing more on the mistakes made by the earlier models. After each iteration, the models are combined using weighted voting, where each model's influence is proportional to its performance. This method reduces bias and is particularly effective in improving the accuracy of weak learners.

Both bagging and boosting are powerful techniques that enhance the performance of individual SVM models by reducing overfitting and increasing accuracy by focusing on mistakes, respectively. Our implementations provide a robust approach to gesture detection, leveraging these ensemble techniques to enhance model generalization and performance. And, despite the simpler architecture compared to more complex deep learning models, our SVM-based approach offers a practical and efficient solution for gesture recognition, with results that match the performance of more resource-intensive methods.

3.4.2 1D and 2D CNNs

The 1D CNN model processes the sequential data, hand landmarks, using 1D convolutional layers to extract the temporal and spatial patterns from the now vectorized inputs. Below is a diagram showing how an example of an architecture with stacked Conv 1D and pooling leads to a classifier (numbers not accurate):

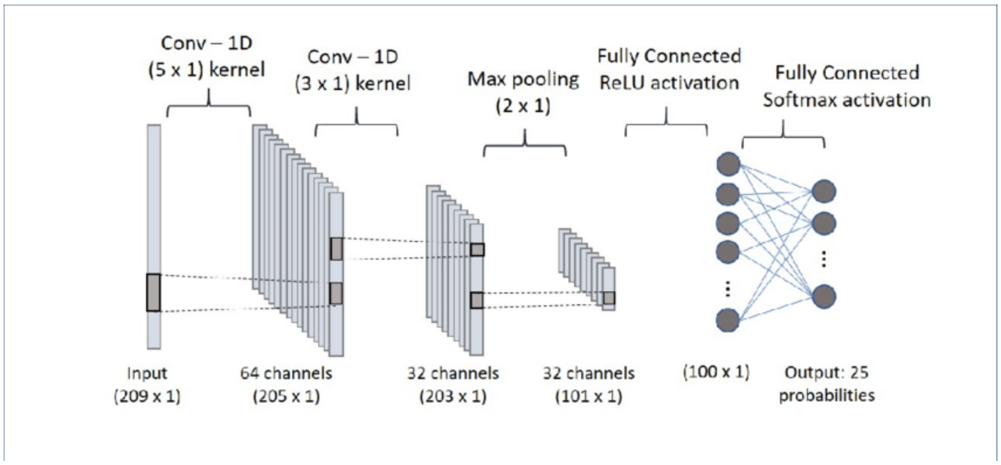

Figure 1: Network Architecture of 1D CNN

The 2D CNN mirrors traditional image classifiers, employing 2D convolutions to capture hierarchical spatial features from input frames. Its structure resembles the 1D variant but operates on image tensors with Conv2D and MaxPooling2D layers:

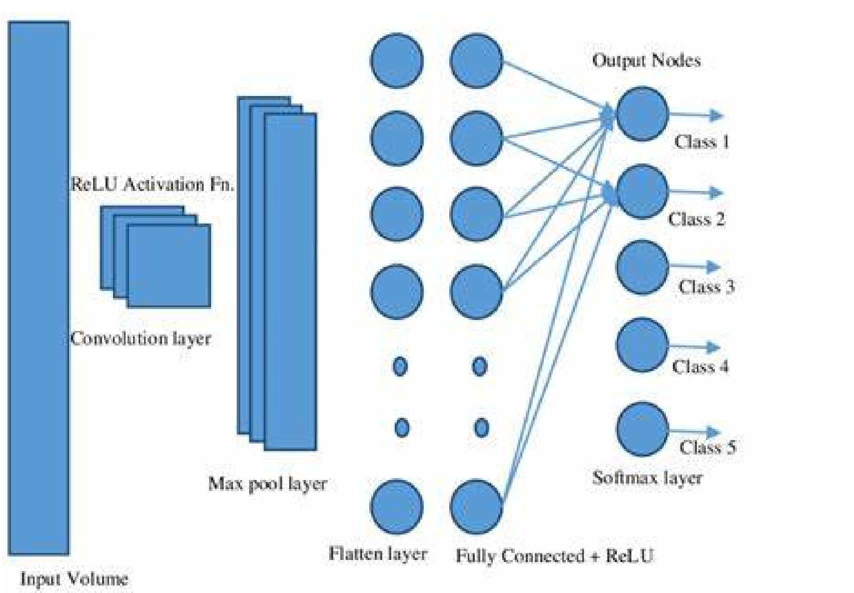

Figure 2: Network Architecture of 2D CNN

Both models utilize TensorFlow with ReLU activations, dropout (0.5), and Adam optimization (lr=0.001). For the 1D CNN, the flattened landmark coordinates were reshaped to (batch_size, 42, 1) while the 2D CNN reshpaed to (height, width, channels). Similarly, both had their data labels encoded via LabelEncoder and the output of the model as class probabilities with softmax. The vectorized labels were then translated back to word labels for simplicity. For testing, we used the classification matrix from sklearn.metrics which reports per-class precision, recall, and F1-score.

3.4.3 1D and 2D DenseNet CNNs

The DenseNet model originated from the ImageNet classifier contest for its ability to concatenate features of previous dense blocks in each layer with convolution and average pooling iterations. Below is a graph depicting a 3-layer 1D-DenseNet model showing the process of convolution, dense block feature concatenation, and pooling steps until a linear output. The 2D-DenseNet model is very similar to the 1D model, except it performs 2D convolutions in an attempt to increase spatial feature recognition.

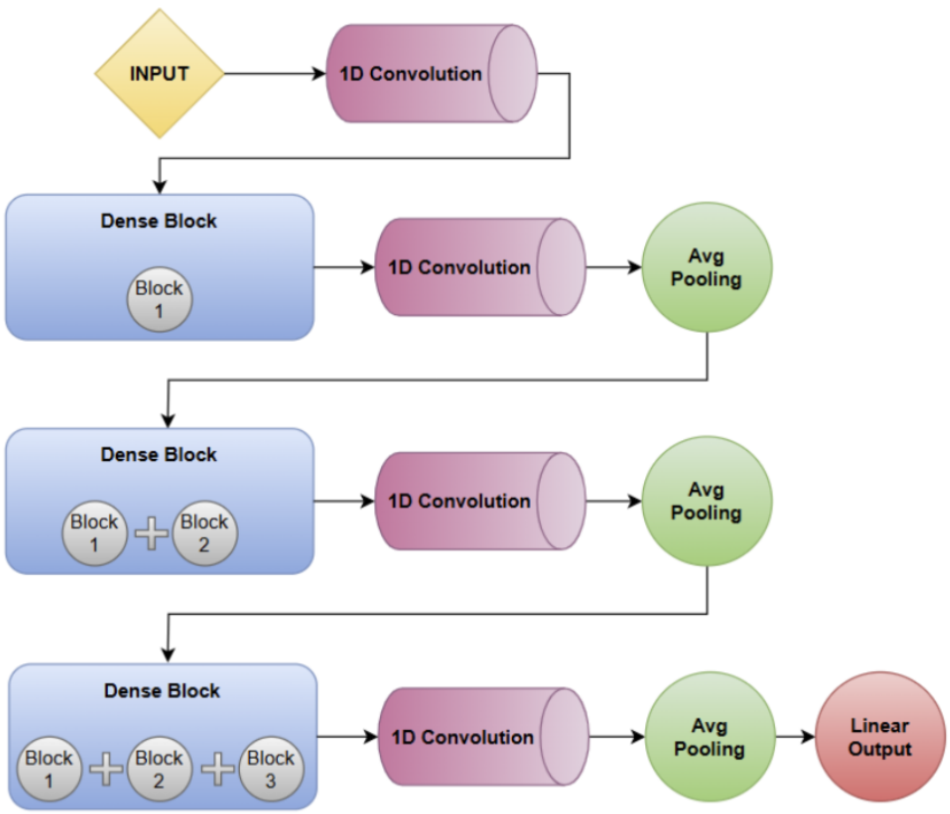

Figure 3: DenseNet Modular Diagram

When developing the DenseNet in Python, the PyTorch Neural Network library is used to build the model's steps for each dense block and transition layer at initialization. All convolutions use ReLU activation functions, a learning rate of .001, and 10 epochs to standardize the hyperparameters with the 1D and 2D CNN mentioned above.

Since neural networks identify classes as integers, the dataset's class names were translated to integers using a LabelEncoder when training. Similarly, when the model would return a prediction or evaluate testing metrics, the integer class returned from the model would be translated back to strings using the LabelEncoder.

Testing of the DenseNet models was done by inserting HaGRID's testing dataset and analyzing its accuracy as well as displaying a full report using sklearn.metric's classification_report library. This gave us information about each class's accuracy, recall, precision, and F1-score.

4 Experimental Results

4.1 Model Accuracy

All models were tested against two collections of the HaGRID dataset. One collection held only 6 gestures: thumbs up, thumbs down, palm, fist, okay sign, and peace sign. The other collection held all 33 gesture classes. Each model was trained on the allocated training data (74%) and tested against the testing data (10%). TabNet used the validation dataset (16%) through each epoch. The success of the 10-epoch DenseNet model prompted us to train a 100-epoch DenseNet model, which had a 95.63% accuracy.

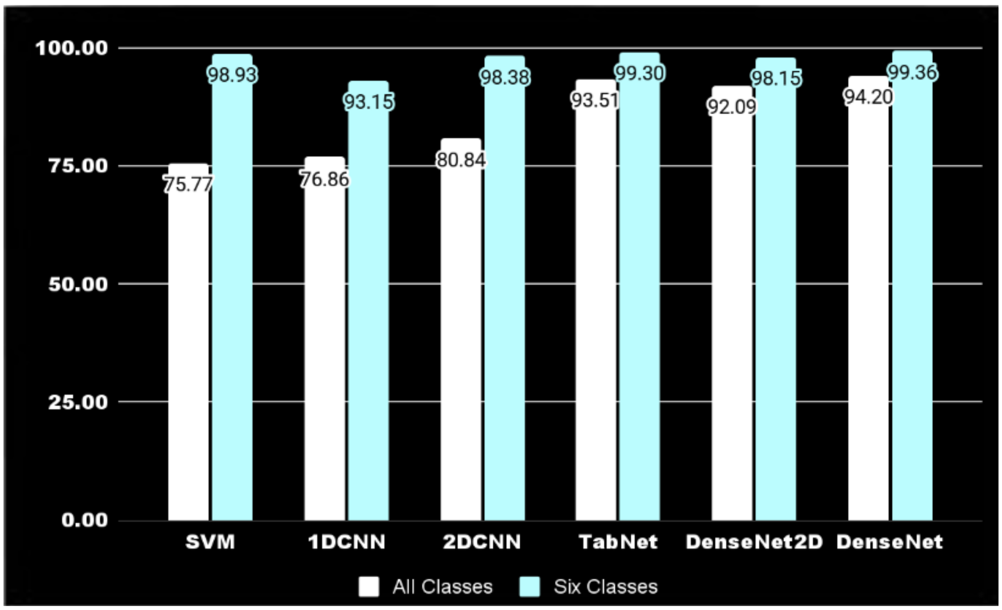

Figure 4: Model Accuracy Chart

4.2 Individual Testing Reports

4.2.1 SVM - 6 Gestures

Below is the output of sklearn.metric's classification_report of the trained SVM model.

Beginning Testing
Accuracy: 0.9893

	precision	recall	f1-score	support
dislike	0.9971	0.9903	0.9937	4932
fist	0.9982	0.9963	0.9972	4877
like	0.9773	0.9953	0.9863	4940
no_gesture	0.9909	0.9555	0.9729	5726
ok	0.9982	0.9986	0.9984	4989
palm	0.9672	0.9998	0.9833	4991
peace	0.9972	0.9945	0.9959	4953
accuracy			0.9893	35408
macro avg	0.9895	0.9900	0.9897	35408
weighted avg	0.9895	0.9893	0.9893	35408

Figure 5: SVM classifying 6 gestures

4.2.2 SVM - All Gestures

Below is the output of sklearn.metric's classification_report of the trained SVM model.

Beginning Testing

Accuracy: 0.7577

	precision	recall	f1-score	support
call	0.7377	0.8533	0.7913	4936
dislike	0.7247	0.9207	0.8110	4932
fist	0.9534	0.9684	0.9608	4877
four	0.7650	0.8754	0.8165	4969
grabbing	0.9031	0.9266	0.9147	4837
grip	0.8631	0.9803	0.9180	4760
hand_heart	0.9638	0.8779	0.9188	8989
hand_heart2	0.8908	0.5501	0.6802	5339
holy	0.6361	0.9354	0.7573	5933
like	0.6298	0.2407	0.3483	4940
little_finger	0.9816	0.9892	0.9854	4918
middle_finger	0.9756	0.9719	0.9737	4845
mute	0.6660	0.8688	0.7540	4922
no_gesture	0.7651	0.9326	0.8406	25867
ok	0.9120	0.9761	0.9430	4989
one	0.8920	0.3943	0.5469	4943
palm	0.8446	0.6045	0.7047	4991
peace	0.6192	0.4741	0.5370	4953
peace_inverted	0.6016	0.8155	0.6925	4939
point	0.8079	0.9197	0.8602	4583
rock	0.9827	0.9782	0.9804	4945
stop	0.7468	0.8793	0.8077	4979
stop_inverted	0.8467	0.7592	0.8006	4941
take_picture	0.7365	0.2794	0.4051	7133
three	0.9733	0.9843	0.9788	4965
three2	0.5910	0.7373	0.6561	4972
three3	0.9785	0.9883	0.9834	4973
three_gun	0.5458	0.9666	0.6977	4968
thumb_index	0.3672	0.9195	0.5248	4956
thumb_index2	0.8729	0.2033	0.3298	9124
timeout	0.2243	0.0387	0.0661	4905
two_up	0.5725	0.5158	0.5427	4936
two_up_inverted	0.6908	0.4071	0.5123	4918
xsign	0.8712	0.8522	0.8616	5954
accuracy			0.7577	201131
macro avg	0.7686	0.7525	0.7324	201131
weighted avg	0.7739	0.7577	0.7357	201131

Figure 6: SVM classifying all gestures

4.2.3 SVM Bagging - 6 Gestures

Below is the output of sklearn.metric's classification_report of the trained Bagging SVM model.

```
Beginning Testing
Accuracy: 0.9895
              precision    recall  f1-score   support

     dislike     0.9971    0.9903    0.9937      4932
        fist     0.9977    0.9963    0.9970      4877
        like     0.9770    0.9953    0.9861      4940
  no_gesture     0.9910    0.9567    0.9735      5726
          ok     0.9982    0.9986    0.9984      4989
        palm     0.9695    0.9998    0.9844      4991
       peace     0.9970    0.9945    0.9958      4953

    accuracy                         0.9895     35408
   macro avg     0.9896    0.9902    0.9898     35408
weighted avg     0.9896    0.9895    0.9895     35408
```

Figure 7: SVM with Bagging classifying 6 gestures

4.2.4 SVM Bagging - All Gestures

Below is the output of sklearn.metric's classification_report of the trained Bagging SVM model.

```
Beginning Testing
Accuracy: 0.7583
                 precision    recall  f1-score   support

           call     0.7406    0.8539    0.7933      4936
        dislike     0.7254    0.9207    0.8115      4932
           fist     0.9537    0.9682    0.9609      4877
           four     0.7657    0.8748    0.8166      4969
       grabbing     0.9046    0.9268    0.9156      4837
           grip     0.8646    0.9805    0.9189      4760
     hand_heart     0.9635    0.8777    0.9186      8989
    hand_heart2     0.8897    0.5546    0.6833      5339
           holy     0.6364    0.9353    0.7574      5933
           like     0.6320    0.2482    0.3564      4940
  little_finger     0.9818    0.9892    0.9855      4918
  middle_finger     0.9733    0.9721    0.9727      4845
           mute     0.6653    0.8698    0.7539      4922
     no_gesture     0.7655    0.9327    0.8408     25867
             ok     0.9119    0.9753    0.9426      4989
            one     0.8931    0.4038    0.5561      4943
           palm     0.8446    0.6065    0.7060      4991
          peace     0.6224    0.4785    0.5410      4953
 peace_inverted     0.6040    0.8145    0.6936      4939
          point     0.8076    0.9195    0.8599      4583
           rock     0.9825    0.9782    0.9803      4945
           stop     0.7473    0.8797    0.8081      4979
  stop_inverted     0.8488    0.7592    0.8015      4941
   take_picture     0.7365    0.2801    0.4059      7133
          three     0.9737    0.9845    0.9791      4965
         three2     0.5896    0.7399    0.6563      4972
         three3     0.9783    0.9883    0.9833      4973
     three_gun      0.5479    0.9660    0.6992      4968
    thumb_index     0.3681    0.9193    0.5257      4956
   thumb_index2     0.8734    0.2005    0.3261      9124
        timeout     0.2232    0.0396    0.0672      4905
         two_up     0.5761    0.5118    0.5420      4936
two_up_inverted     0.6900    0.4124    0.5162      4918
          xsign     0.8723    0.8512    0.8616      5954

       accuracy                         0.7583    201131
      macro avg     0.7692    0.7533    0.7334    201131
   weighted avg     0.7744    0.7583    0.7365    201131
```

Figure 8: SVM with Bagging classifying all gestures

4.2.5 SVM Boosting - 6 Gestures

Below is the output of sklearn.metric's classification_report of the trained Boosting SVM model.

```
Beginning Testing
Accuracy: 0.9894
              precision    recall  f1-score   support

     dislike     0.9969    0.9905    0.9937      4932
        fist     0.9977    0.9961    0.9969      4877
        like     0.9787    0.9953    0.9870      4940
  no_gesture     0.9911    0.9563    0.9734      5726
          ok     0.9978    0.9986    0.9982      4989
        palm     0.9676    0.9998    0.9834      4991
       peace     0.9972    0.9945    0.9959      4953

    accuracy                         0.9894     35408
   macro avg     0.9896    0.9902    0.9898     35408
weighted avg     0.9896    0.9894    0.9894     35408
```

Figure 9: SVM with Boosting classifying 6 gestures

4.2.6 SVM Boosting- All Gestures

Below is the output of sklearn.metric's classification_report of the trained Boosting SVM model.

Beginning Testing
Accuracy: 0.7581

	precision	recall	f1-score	support
call	0.7407	0.8531	0.7930	4936
dislike	0.7243	0.9211	0.8110	4932
fist	0.9541	0.9682	0.9611	4877
four	0.7698	0.8710	0.8173	4969
grabbing	0.9033	0.9272	0.9151	4837
grip	0.8631	0.9803	0.9180	4760
hand_heart	0.9641	0.8777	0.9189	8989
hand_heart2	0.8899	0.5497	0.6796	5339
holy	0.6345	0.9356	0.7562	5933
like	0.6360	0.2476	0.3564	4940
little_finger	0.9814	0.9892	0.9853	4918
middle_finger	0.9754	0.9721	0.9737	4845
mute	0.6689	0.8675	0.7554	4922
no_gesture	0.7659	0.9327	0.8411	25867
ok	0.9125	0.9761	0.9433	4989
one	0.8913	0.3933	0.5458	4943
palm	0.8433	0.6115	0.7089	4991
peace	0.6133	0.4809	0.5391	4953
peace_inverted	0.6059	0.8107	0.6935	4939
point	0.8084	0.9190	0.8602	4583
rock	0.9825	0.9786	0.9805	4945
stop	0.7463	0.8807	0.8079	4979
stop_inverted	0.8483	0.7592	0.8012	4941
take_picture	0.7353	0.2784	0.4039	7133
three	0.9733	0.9843	0.9788	4965
three2	0.5962	0.7291	0.6560	4972
three3	0.9785	0.9883	0.9834	4973
three_gun	0.5457	0.9666	0.6976	4968
thumb_index	0.3665	0.9201	0.5242	4956
thumb_index2	0.8728	0.2046	0.3315	9124
timeout	0.2247	0.0385	0.0658	4905
two_up	0.5694	0.5170	0.5419	4936
two_up_inverted	0.6896	0.4193	0.5215	4918
xsign	0.8711	0.8524	0.8616	5954
accuracy			0.7581	201131
macro avg	0.7690	0.7530	0.7332	201131
weighted avg	0.7742	0.7581	0.7364	201131

Figure 10: SVM with Boosting classifying all gestures

4.2.7 1D CNN

Below is the output of sklearn.metric's classification_report of the trained 1D CNN model.

	precision	recall	f1-score	support
call	0.9624	0.9757	0.9690	4936
dislike	0.9237	0.9627	0.9428	4932
fist	0.6363	0.9818	0.7721	4877
four	0.9826	0.8859	0.9317	4969
grabbing	0.7275	0.9698	0.8314	4837
grip	0.9153	0.9189	0.9171	4760
hand_heart	0.9523	0.9255	0.9387	8989
hand_heart2	0.7643	0.7464	0.7552	5339
holy	0.6912	0.9309	0.7934	5933
like	0.7469	0.9613	0.8407	4940
little_finger	0.9615	0.9858	0.9735	4918
middle_finger	0.9218	0.9437	0.9326	4845
mute	0.7979	0.9376	0.8621	4922
no_gesture	0.7098	0.9843	0.8248	25867
ok	0.9462	0.9302	0.9381	4989
one	0.3457	0.8932	0.4985	4943
palm	0.9507	0.8349	0.8891	4991
peace	0.5150	0.8524	0.6421	4953
peace_inverted	0.5269	0.9393	0.6751	4939
point	0.6754	0.9020	0.7724	4583
rock	0.9492	0.9189	0.9338	4945
stop	0.9119	0.8923	0.9020	4979
stop_inverted	0.9859	0.8891	0.9350	4941
take_picture	0.8336	0.9019	0.8664	7133
three	0.9951	0.9396	0.9665	4965
three2	0.7027	0.9755	0.8169	4972
three3	0.9981	0.9326	0.9642	4973
three_gun	0.9914	0.9084	0.9481	4968
thumb_index	0.0000	0.0000	0.0000	4956
thumb_index2	0.0000	0.0000	0.0000	9124
timeout	0.0000	0.0000	0.0000	4905
two_up	0.0000	0.0000	0.0000	4936
two_up_inverted	0.0000	0.0000	0.0000	4918
xsign	0.0000	0.0000	0.0000	5954
accuracy			0.7686	201131
macro avg	0.6771	0.7594	0.7069	201131
weighted avg	0.6706	0.7686	0.7077	201131

Figure 11: 1D CNN classifying all gestures with 10 epochs

4.2.8 2D CNN

Below is the output of sklearn.metric's classification_report of the trained 2D CNN model.

	precision	recall	f1-score	support
call	0.9944	0.9783	0.9863	4936
dislike	0.9240	0.9811	0.9517	4932
fist	0.8174	0.9914	0.8960	4877
four	0.9849	0.9948	0.9898	4969
grabbing	0.9553	0.9851	0.9700	4837
grip	0.9427	0.9788	0.9604	4760
hand_heart	0.9636	0.9735	0.9685	8989
hand_heart2	0.7304	0.9648	0.8314	5339
holy	0.7101	0.9821	0.8242	5933
like	0.8233	0.9943	0.9008	4940
little_finger	0.9884	0.9908	0.9896	4918
middle_finger	0.9264	0.9893	0.9568	4845
mute	0.6526	0.9707	0.7805	4922
no_gesture	0.9055	0.9671	0.9353	25867
ok	0.9948	0.9882	0.9915	4989
one	0.3353	0.9630	0.4974	4943
palm	0.9890	0.9770	0.9830	4991
peace	0.4923	0.9909	0.6578	4953
peace_inverted	0.5211	0.9812	0.6807	4939
point	0.5797	0.9644	0.7241	4583
rock	0.8881	0.9901	0.9363	4945
stop	0.9202	0.9863	0.9521	4979
stop_inverted	0.9731	0.9745	0.9738	4941
take_picture	0.9592	0.9159	0.9370	7133
three	0.9879	0.9879	0.9879	4965
three2	0.7823	0.9954	0.8761	4972
three3	0.9956	0.9926	0.9941	4973
three_gun	0.9894	0.9924	0.9909	4968
thumb_index	0.0000	0.0000	0.0000	4956
thumb_index2	0.0000	0.0000	0.0000	9124
timeout	0.0000	0.0000	0.0000	4905
two_up	0.0000	0.0000	0.0000	4936
two_up_inverted	0.0000	0.0000	0.0000	4918
xsign	0.0000	0.0000	0.0000	5954
accuracy			0.8084	201131
macro avg	0.6979	0.8071	0.7389	201131
weighted avg	0.7099	0.8084	0.7477	201131

Figure 12: 2D CNN classifying all gestures with 10 epochs

4.2.9 1D DenseNet - 6 Gestures, 10 Epochs

Below is the output of sklearn.metric's classification_report of the trained 1D DenseNet model.

Accuracy: 0.9936

	precision	recall	f1-score	support
dislike	0.9903	0.9937	0.9920	4932
fist	0.9957	0.9932	0.9945	4877
like	0.9907	0.9949	0.9928	4940
no_gesture	0.9919	0.9871	0.9895	5726
ok	0.9988	0.9916	0.9952	4989
palm	0.9909	0.9992	0.9950	4991
peace	0.9970	0.9962	0.9966	4953
accuracy			0.9936	35408
macro avg	0.9936	0.9937	0.9936	35408
weighted avg	0.9936	0.9936	0.9936	35408

Figure 13: 1D DenseNet classifying 6 gestures with 10 epochs

4.3 1D DenseNet - All Gestures, 10 Epochs

Below is the output of sklearn.metric's classification_report of the trained 1D DenseNet model.

Accuracy: 0.9420

	precision	recall	f1-score	support
call	0.9836	0.9587	0.9710	4936
dislike	0.9536	0.9785	0.9659	4932
fist	0.9761	0.9705	0.9733	4877
four	0.9821	0.9843	0.9832	4969
grabbing	0.9640	0.9737	0.9688	4837
grip	0.9468	0.9798	0.9630	4760
hand_heart	0.9541	0.9645	0.9593	8989
hand_heart2	0.9135	0.8867	0.8999	5339
holy	0.8860	0.9322	0.9085	5933
like	0.9752	0.9642	0.9697	4940
little_finger	0.9717	0.9915	0.9815	4918
middle_finger	0.9854	0.9730	0.9791	4845
mute	0.9565	0.9553	0.9559	4922
no_gesture	0.9633	0.9744	0.9688	25867
ok	0.9859	0.9557	0.9706	4989
one	0.9608	0.9632	0.9620	4943
palm	0.9832	0.9391	0.9606	4991
peace	0.9720	0.9310	0.9510	4953
peace_inverted	0.9774	0.9269	0.9515	4939
point	0.9126	0.9474	0.9297	4583
rock	0.9883	0.9893	0.9888	4945
stop	0.9453	0.9711	0.9580	4979
stop_inverted	0.9600	0.9872	0.9735	4941
take_picture	0.9484	0.9071	0.9273	7133
three	0.9902	0.9813	0.9857	4965
three2	0.9897	0.9827	0.9862	4972
three3	0.9935	0.9875	0.9905	4973
three_gun	0.9887	0.9863	0.9875	4968
thumb_index	0.6309	0.6281	0.6295	4956
thumb_index2	0.7894	0.7646	0.7768	9124
timeout	0.8881	0.8239	0.8548	4905
two_up	0.9288	0.9781	0.9528	4936
two_up_inverted	0.9233	0.9664	0.9444	4918
xsign	0.9064	0.9444	0.9250	5954
accuracy			0.9420	201131
macro avg	0.9434	0.9426	0.9428	201131
weighted avg	0.9420	0.9420	0.9418	201131

Figure 14: 1D DenseNet classifying all gestures with 10 epochs

4.4 1D DenseNet - All Gestures, 100 Epochs

Below is the output of sklearn.metric's classification_report of the trained 1D DenseNet model.

Accuracy: 0.9563

	precision	recall	f1-score	support
call	0.9926	0.9751	0.9838	4936
dislike	0.9822	0.9866	0.9844	4932
fist	0.9821	0.9893	0.9857	4877
four	0.9819	0.9938	0.9878	4969
grabbing	0.9735	0.9866	0.9800	4837
grip	0.9838	0.9824	0.9831	4760
hand_heart	0.9804	0.9716	0.9760	8989
hand_heart2	0.9057	0.9494	0.9270	5339
holy	0.9271	0.9434	0.9352	5933
like	0.9861	0.9759	0.9810	4940
little_finger	0.9919	0.9906	0.9913	4918
middle_finger	0.9843	0.9829	0.9836	4845
mute	0.9713	0.9701	0.9707	4922
no_gesture	0.9821	0.9762	0.9792	25867
ok	0.9859	0.9946	0.9902	4989
one	0.9712	0.9688	0.9700	4943
palm	0.9882	0.9888	0.9885	4991
peace	0.9851	0.9847	0.9849	4953
peace_inverted	0.9830	0.9860	0.9845	4939
point	0.9489	0.9688	0.9588	4583
rock	0.9929	0.9903	0.9916	4945
stop	0.9833	0.9817	0.9825	4979
stop_inverted	0.9796	0.9903	0.9849	4941
take_picture	0.9618	0.9317	0.9465	7133
three	0.9891	0.9871	0.9881	4965
three2	0.9970	0.9930	0.9950	4972
three3	0.9925	0.9897	0.9911	4973
three_gun	0.9892	0.9954	0.9923	4968
thumb_index	0.5627	0.8810	0.6867	4956
thumb_index2	0.9031	0.6044	0.7242	9124
timeout	0.8925	0.8852	0.8888	4905
two_up	0.9880	0.9882	0.9881	4936
two_up_inverted	0.9850	0.9750	0.9800	4918
xsign	0.9363	0.9704	0.9531	5954
accuracy			0.9563	201131
macro avg	0.9600	0.9626	0.9594	201131
weighted avg	0.9612	0.9563	0.9565	201131

Figure 15: 1D DenseNet classifying all gestures with 100 epochs

4.5 2D DenseNet - 6 Gestures, 10 Epochs

Below is the output of sklearn.metric's classification_report of the trained 2D DenseNet model.

Accuracy: 0.9815

	precision	recall	f1-score	support
dislike	0.9904	0.9864	0.9884	4932
fist	0.9939	0.9762	0.9850	4877
like	0.9417	0.9915	0.9660	4940
no_gesture	0.9801	0.9527	0.9662	5726
ok	0.9918	0.9920	0.9919	4989
palm	0.9925	0.9842	0.9883	4991
peace	0.9824	0.9917	0.9870	4953
accuracy			0.9815	35408
macro avg	0.9818	0.9821	0.9818	35408
weighted avg	0.9818	0.9815	0.9815	35408

Figure 16: 2D DenseNet classifying 6 gestures with 10 epochs

4.6 2D DenseNet - All Gestures, 10 Epochs

Below is the output of sklearn.metric's classification_report of the trained 2D DenseNet model.

Accuracy: 0.9209

	precision	recall	f1-score	support
call	0.9624	0.9706	0.9665	4936
dislike	0.9549	0.9627	0.9588	4932
fist	0.9443	0.9871	0.9652	4877
four	0.9701	0.9648	0.9674	4969
grabbing	0.9380	0.9696	0.9535	4837
grip	0.9741	0.9637	0.9688	4760
hand_heart	0.9542	0.9559	0.9551	8989
hand_heart2	0.8450	0.9241	0.8828	5339
holy	0.8295	0.9510	0.8861	5933
like	0.9311	0.9735	0.9518	4940
little_finger	0.9908	0.9856	0.9882	4918
middle_finger	0.9863	0.9674	0.9768	4845
mute	0.9646	0.9197	0.9417	4922
no_gesture	0.9671	0.9650	0.9660	25867
ok	0.9787	0.9473	0.9627	4989
one	0.9167	0.9711	0.9431	4943
palm	0.9388	0.9581	0.9483	4991
peace	0.9721	0.9136	0.9419	4953
peace_inverted	0.9886	0.8601	0.9199	4939
point	0.9339	0.9057	0.9196	4583
rock	0.9909	0.9858	0.9883	4945
stop	0.9678	0.9422	0.9548	4979
stop_inverted	0.9788	0.9541	0.9663	4941
take_picture	0.9420	0.9044	0.9228	7133
three	0.9526	0.9831	0.9676	4965
three2	0.9882	0.9793	0.9837	4972
three3	0.9919	0.9817	0.9868	4973
three_gun	0.9776	0.9938	0.9856	4968
thumb_index	0.4352	0.9473	0.5965	4956
thumb_index2	0.9171	0.3032	0.4557	9124
timeout	0.9190	0.7443	0.8225	4905
two_up	0.9171	0.9775	0.9464	4936
two_up_inverted	0.8613	0.9711	0.9129	4918
xsign	0.9434	0.9318	0.9376	5954
accuracy			0.9209	201131
macro avg	0.9331	0.9299	0.9233	201131
weighted avg	0.9361	0.9209	0.9184	201131

Figure 17: 2D DenseNet classifying all gestures with 10 epochs

4.7 TabNet - 6 Gestures, 100 Maximum Epochs

Below is the output of sklearn.metric's classification_report of the trained TabNet model.

```
Accuracy: 0.9930
              precision    recall  f1-score   support

     dislike     0.9963    0.9933    0.9948      4932
        fist     0.9937    0.9955    0.9946      4877
        like     0.9904    0.9852    0.9878      4940
  no_gesture     0.9791    0.9878    0.9834      5726
          ok     0.9990    0.9988    0.9989      4989
        palm     0.9968    0.9968    0.9968      4991
       peace     0.9980    0.9943    0.9962      4953

    accuracy                         0.9930     35408
   macro avg     0.9933    0.9931    0.9932     35408
weighted avg     0.9930    0.9930    0.9930     35408
```

Figure 18: TabNet classifying 6 gestures with 100 max epochs

4.8 TabNet - All Gestures, 100 Maximum Epochs

Below is the output of sklearn.metric's classification_report of the trained TabNet model.

Accuracy: 0.9351

	precision	recall	f1-score	support
call	0.9928	0.9504	0.9711	4936
dislike	0.9625	0.9832	0.9727	4932
fist	0.9664	0.9791	0.9727	4877
four	0.9871	0.9549	0.9707	4969
grabbing	0.9844	0.9524	0.9682	4837
grip	0.9606	0.9834	0.9719	4760
hand_heart	0.9851	0.9531	0.9688	8989
hand_heart2	0.9132	0.9088	0.9110	5339
holy	0.8862	0.9424	0.9134	5933
like	0.9588	0.9848	0.9716	4940
little_finger	0.9848	0.9888	0.9868	4918
middle_finger	0.9854	0.9761	0.9807	4845
mute	0.9620	0.9614	0.9617	4922
no_gesture	0.9643	0.9723	0.9683	25867
ok	0.9845	0.9950	0.9897	4989
one	0.9556	0.9585	0.9571	4943
palm	0.9596	0.9794	0.9694	4991
peace	0.9826	0.9683	0.9754	4953
peace_inverted	0.9868	0.9816	0.9842	4939
point	0.9007	0.9577	0.9283	4583
rock	0.9905	0.9879	0.9892	4945
stop	0.9411	0.9821	0.9612	4979
stop_inverted	0.9636	0.9860	0.9747	4941
take_picture	0.9790	0.9003	0.9380	7133
three	0.9843	0.9742	0.9792	4965
three2	0.9876	0.9934	0.9905	4972
three3	0.9669	0.9879	0.9773	4973
three_gun	0.9670	0.9962	0.9814	4968
thumb_index	0.4512	0.9318	0.6080	4956
thumb_index2	0.9060	0.3571	0.5123	9124
timeout	0.9090	0.7778	0.8383	4905
two_up	0.9637	0.9895	0.9764	4936
two_up_inverted	0.9790	0.9742	0.9766	4918
xsign	0.8973	0.9535	0.9245	5954
accuracy			0.9351	201131
macro avg	0.9456	0.9448	0.9389	201131
weighted avg	0.9473	0.9351	0.9334	201131

Figure 19: TabNet classifying all gestures with 100 max epochs

5 Conclusion

5.1 Experiment Observations

Model Type	All Gestures	6 Gestures
SVM	Struggles with high dimensionality	Excellent linear separability
1D CNN	Limited temporal feature learning	Adequate for basic gestures
2D CNN	Better spatial generalization	Over-engineered for simple cases
DenseNet	Optimal feature reuse	Slight overfitting tendency
TabNet	Effective feature selection	Maintains interpretability

Table 1: A 6-column by 3-row table

Overall, the models trained on the 6 gesture classes performed better than the models trained on the 33 classes, which is what we expected initially, because the 6-class classification is inherently easier than 33-class classification. As we progressed in complexity of models (following the timeline they were taught in class), we can see a general increase in accuracy for the models trained on 33 classes. And, comparing the powerhouse models (DenseNet and TabNet), we can see they both performed surprisingly well, 95.63% and 93.51% respectively. The deep learning models perform better due to their ability to reuse features, mitigate vanishing gradients, and improve parameter efficiency, which make up for simple CNN limitations. Although they are both more computationally expensive, the tradeoff is worth it for being more accurate.

5.2 The Optimal Model

In analyzing our experimental results, the model architecture that best fits our data for the full 33 gesture classification is the DenseNet convolutional neural network. The DenseNet's feature concatenation between each dense block's 1D convolutions create the feature relations needed to build a deep layered network for accurately classifying gestures. The DenseNet model achieved an accuracy of 99.36% on the testing dataset for 6 gestures and 94.20% on the testing dataset for all 33 gestures when running 10 epochs.

The success of the 10-epoch model encouraged further training of a deeper network by iterating the model through 100 epoch training sessions over several hours. This increased our accuracy for classifying all gestures by 95.63% accuracy, which satisfies our project's objective.

Saving the weights of the neural network, we can now integrate the model with the OpenCV live feed to dynamically create the feature vector from real-time hand gestures and insert them into the model for predictions. The predictions are then output to the screen for both the left and right hands. The full live translation interface ran at 20 FPS and output accurate results.

5.3 Future Work

In observing the test results from the SVM model, we saw a strong accuracy for a small set of gesture classes. The SVM performed with a 98.93% accuracy on the 6 original gestures. This reflects SVM's main objective in finding linearly separable data over linearly correlated features.

In improving the speed of our live translator, we can use the SVM for simple, linearly separable gestures, running at 30 FPS, and use the DenseNet model as a "failsafe" approach for when the SVM detects a more complex class. This will increase our performance by utilizing simpler models when the data is simple and reserving the higher computations for the more complex feature patterns.

An additional approach for future development is to integrate transformer layers into the 1D-CNN. This technique can better capture the long-range temporal dependencies in hand movements and improve sequential gesture recognition.

6 References

"HaGRID: Hand Gesture Recognition Image Dataset." Nuzhdin, Dmitry, et al. GitHub, 2024, https://github.com/hukenovs/hagrid.

"Hand Landmarks Detection Guide for Python." MediaPipe Solutions, Google AI Edge, n.d. https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker/python.

"DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation." Fang, Biyi, Jillian Co, and Mi Zhang. arXiv, 18 Oct. 2018, https://arxiv.org/abs/1802.07584.

"Real-time sign language recognition using Deep Learning Techniques." Wahane, Abhishek, et al. 2022 IEEE 7th International Conference for Convergence in Technology (I2CT), 7 Apr. 2022, https://doi.org/10.1109/i2ct54291.2022.9825192.

"Sign Language Transformers: Joint End-to-End Sign Language Recognition and Translation." Camgoz, Necati Cihan, et al. arXiv.Org, 30 Mar. 2020, doi.org/10.48550/arXiv.2003.13830.

"American Sign Language Recognition System using wearable sensors and machine learning." Dibba, Modou, and Cheol-Hong Min. 2023 21st IEEE Interregional NEWCAS Conference (NEWCAS), 26 June 2023, https://doi.org/10.1109/newcas57931.2023.10198199.

"Sign language recognition using LSTM and media pipe." Rao, G. Mallikarjuna, et al. 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS), 17 May 2023, pp. 1086–1091, https://doi.org/10.1109/iciccs56967.2023.10142638.