Handle: An Open Source Tool For Accurate Gesture Recognition In Real-Time Systems Across Static And Dynamic Modalities

Aadarsh Jha[1][0000−0001−9848−3925] and Raahul Natarrajan[1]

Vanderbilt University, Department of Computer Science

Abstract. Herein, Handle presents an end-to-end open-source tool for accurate gesture recognition in real-time systems for static and dynamic modalities. The goal of this project is to utilize large-scale hand gesture databases to create new, and refine existing, models of hand gesture recognition (HGR) to demonstrate state-of-the-art approaches via an application, from static to dynamic. For the static sub-problem, consistently strong performance was seen on the Hand Gesture Recognition Database (HGRD) and ASL Dataset (ASLMNIST) (accuracy of > 0.8 and > 0.9, respectively), with optimal performance achieved in DenseNet paired with ASLMNIST (accuracy of 0.9918). Additionally, limited performance (accuracy of 0.2041 and 0.3862) was achieved through an additional transfer learning study between HGRD and ASLMNIST. On the dynamic side, we experimented with various pretrained models, such as IPN pretrained 3D CNN, Jester pretrained 3D CNN, EgoGesture pretrained 3D CNN, Kinetics pretrained 3D CNN, ImageNet pretrained CNN-LSTM, and HowTo100M pretrained TimeSformer space-time attention model. After experimentation, we found that 3D CNN models pretrained on HGR datasets performed much better than 3D CNNs pretrained on other datasets, ImageNet pretrained CNN-LSTM, and TimeSformer. Overall, on a deployed application, it was qualitatively found that the dynamic models significantly outperformed static models on user-generated data. The code for this project is available online at: https://github.com/aadarshjha/handle

Keywords: Deep Learning · Hand Gesture Recognition · Real-Time Systems · Full-Stack Application.

1 Introduction

The recognition of natural hand gestures, generalized as the HGR problem, is the use of algorithmic techniques to allow computers to recognize and classify movements created by an end-user [12]. More specifically, continuous HGR is essential in human-computer interaction, with a specific application in Augmented, Mixed, Virtual, and General Extended Reality systems so as to allow users to engage in digital assets in an immersive fashion [8]. In research, Deep Learning

applied to continuous HGR is a new problem, and the lack of real-world datasets presents a challenge. Thus, accurate models in deployable, productionized systems are not yet at optimal performance due to lack of model exploration and data diversity [21].

The overarching goal of this paper is to utilize large scale hand gesture databases to create and refine existing models of HGR to demonstrate state-of-the-art approaches via a deployed application, from static to dynamic. More specifically, this translates to three project sub-goals and contributions:

- **Experimentation**: Our project serves as a comparative analysis by creating and adopting existing DL architectures (Convolutional Neural Networks, Long Short-Term Memory, Transformers) in order to obtain optimal performance across static and dynamic contexts on existing databases.
- **Transparency**: Our application allows for transparency by enabling the end-user to experiment with the models created in this study. To this end, our motivation is to ease the gap in experimenting in the field of HGR, as most studies are difficult to use and apply.
- **Summary**: Our study will serve as a comparative analysis of a large set of data from different subproblems in HGR. We explore various SOTA architectures like 2D and 3D CNNs, LSTMs, and Transformers and analyze the inner workings of each of these architectures. In effect, we provide a holistic benchmark for future studies to reflect on various DL approaches in HGR.

Overall, our paper investigates two sub-problems of HGR, which include *Static* and *Dynamic* recognition. These domains are discussed further in the **Related Works**.

1.1 Project Dissemination

In order to meet the project specifications, as noted in the **Appendix**, Raahul and Aadarsh divided the project as shown in **Table 1**. Specifically, Aadarsh was primarily in charge of the static recognition portion of the project which includes: 1) experimentation, 2) deployment, and 3) application development. To address 1), Aadarsh varied DL architectures across transfer learning and custom models (CNNs, DenseNet Pretrained, VGG Pretrained, and MobileNet Pretrained) in order to discover optimal performance on two static HGR datasets. Additionally, model adaptation between the two static HGR datasets were also studied. To address 2), the best-performing models from experimentation were compiled and then deployed onto our open-source application for real-time inference. Finally, for 3), Aadarsh worked primarily on the full-stack development of the application, integrating deployed models with an easy-to-use interface that captured custom user data.

The dynamic recognition problem is very time-intensive to explore due to the sheer size of the video data needed to train and test models and the complexity of the models themselves. So, Raahul focused on three main tasks for this problem: 1) dataset refinement, 2) experimentation, and 3) deployment. Raahul

addressed 1) by creating a new minimal dataset called `MiniIPN` by extracting only three classes of gesture data from the original IPN Hand dataset [5]. Additionally, to address 2), Raahul explored various deep learning architectures for dynamic gesture recognition by applying transfer learning on various pretrained models to evaluate these models' performance on the dynamic problem. Specifically, Raahul trained and evaluated the following models: IPN pretrained 3D CNN, Jester pretrained 3D CNN, EgoGesture pretrained 3D CNN, Kinetics pretrained 3D CNN, ImageNet pretrained CNN-LSTM, and HowTo100M pretrained TimeSformer space-time attention model. After training and evaluating these models, Raahul deployed the best-performing models onto the application as part of accomplishing 3). Raahul additionally stress tested the application and debugged any issues that arose.

Table 1. Project Dissemination Of Handle.

Aadarsh	Raahul
Static Recognition Experimentation	Dynamic Recognition Experimentation
Static Model Deployment Onto Application	Dynamic Model Deployment Onto Application
Full Stack Development Of Application	User And Stress Testing Of Application
Final Presentation, Final Write-Up, Check-In Presentation	Final Presentation, Final Write-Up, Check-In Presentation
	Creating New Dynamic Dataset

2 Related Works

Herein, previous literature related to Static and Dynamic HGR is presented in order to contextualize the methodologies employed in this paper.

2.1 Static HGR

Static recognition is defined as a subproblem of HGR, which emphasizes the processing of a single image in order to classify certain gestures [24]. Conventional approaches towards static HGR include support vector machines (SVMs) [15,20], nearest neighbor approaches [6], graphs [26], and distributed locally linear embedding [13]. Recent DL approaches in Static HGR have shown state-of-the-art performance against standard datasets. Specifically, recent works have applied CNNs [3] to various static gesture databases, such as the American Sign Language (ASL) domain [2], in order to achieve competitive performance to more standard approaches [25,11,16,24]. More advanced approaches have also explored data augmentation [16], parallel CNNs [11], and transfer learning [22].

2.2 Dynamic HGR

Dynamic recognition involves the processing of several frames, or image sequences, to recognize gestures, which typically require more complex HGR approaches. With the success of deep convolutional neural networks (CNNs) on object detection and image classification tasks like ImageNet [9], CNNs were also applied to video data as 3D CNNs. 3D CNNs account for the temporal aspect of video data in addition to the spatial information contained in each video frame, unlike 2D CNNs, which do not have the ability to account for temporal trends across video frames. Köpüklü et al. apply a ResNeXt based architecture with 3-D convolutional layers to classify hand gestures in real-time video [17]. Most dynamic recognition papers mainly focus on applying 3D convolutions to understand dynamic gestures due to their SOTA performance. However, by treating the dynamic problem as a video classification problem, we can explore additional popular architectures. The dynamic HGR problem is essentially video classification for hand gestures. And so, adjacent to hand gesture recognition in the domain of human action recognition, Baccouche et al. further the 3D CNN approach and propose a 3D-CNN-LSTM model [4]. Similar to the LSTM approach, Zhu et al. propose a recurrent neural network (RNN) called FASTER to process and classify video clips [29]. With the rise of self-attention proposed by Vaswani et al. in NLP tasks, various approaches have been taken to integrate self-attention into both the image and video classification problems [27,14,10]. Specifically, the TimeSformer approach proposed by Bertasius et al. uses an almost convolution free approach to video classification by applying the concept of space-time self-attention to video data [7]. Essentially, the main approaches boil down to 3D CNNs, variations of the CNN-LSTM architecture, self-attention-based networks, or an amalgamation of all these architectures.

3 Methods

In this section, a three-fold methodology is presented which drives the experimentation of the project: 1) Static HGR Experimentation, 2) Dynamic HGR Experimentation, and 3) Application Development.

3.1 Static HGR

Overview To begin, the Static HGR experimentation applies four unique DL architectures across two datasets. The two datasets are the Hand Gesture Recognition Database (`https://www.kaggle.com/datasets/gti-upm/leapgestrecog`) and the American Sign Language MNIST dataset (`https://www.kaggle.com/datasets/datamunge/sign-language-mnist`). The four DL architectures that are used include: 1) a custom CNN; 2) a pretrained VGG-16; 3) a pretrained MobileNet; and 4) a pretrained DenseNet-121. All experiments are conducted in a Google Colab environment, which utilize a typical workstation with Intel Xeon CPU 2.2 GHz, 13 GB RAM, 33 GB Disk Space, 12 GB NVIDIA Tesla K80 GPU, and CUDA 10.1

Dataset and Preprocessing The Hand Gesture Recognition Database (HGRD) is a set of near infrared images acquired by the Leap Motion sensor. HGRD contains 10 classes, including: Palm, L, Fist, Fist Moved, Thumb, Index, Ok, Palm Moved, C, and Down. HGRD contains images of an initial size of `640x240`. Each DL pipeline began with the same preprocessing steps which included: 1) a resizing to `90x90`, and 2) a conversion to an RGB, 3-channel image. The American Sign Language MNIST dataset (ASLMNIST) is a dataset of `28x28` images with grayscale values between `0-255`. Similar to HGRD, these images are normalized via a factor of `255`, resampled to a `90x90` image size for consistency, and are also converted to three-channel RGB images. Uniquely, data augmentation strategies are also employed in this dataset, wherein Tensorflow's [1] `ImageDataGenerator` is utilized to employ rotation, zoom, width and height shift, as well as vertical and horizontal flips.

Deep Learning Architectures And Training Process Four models are applied to the processed HGRD and ASLMNIST data, including: 1) a custom CNN; 2) a pretrained VGG-16; 3) a pretrained MobileNet; and 4) a pretrained DenseNet-121. For both HGRD and ASLMNIST, the same exact network architecture is utilized for the *pretrained* experiments (VGG, MobileNet, DenseNet). In particular, each pretrained experimentation is left to default implementation as specified by TensorFlow [1]. The pretrained networks utilize weights from the ImageNet [9] dataset. Via TensorFlow, VGG, MobileNet, and DenseNet serve as a feature extractor by excluding the classification layers at the top of the network. In particular, the final dense layers (i.e., "top") of each model are removed, and replaced. To this end, the "top" of MobileNet is replaced with a global average pooling layer, three intermediate Dense layers, and a final output Dense layer. The "top" of DenseNet is replaced with a Flattening operation, two intermediate layers of Batch Normalization-Dense Layer-Dropout, and a final Batch Normalization layer followed by an output Dense layer. The "top" of VGG simply adds a output Dense layer. Figure 1 demonstrates the networks for HGRD, and Figure 2 visualizes the networks for ASLMNIST.

The main difference lies in the custom-built CNN, which was based on top-performing results on Kaggle. Namely, for HGRD, the structure of the CNN is three units of 2D convolutional layers, followed by max pooling layers. Additionally, a flattening layer is present, with an intermediate Dense layer, and a final output Dense layer. On the other hand, the ASLMNIST CNN is deeper, due to the simplicity and low-resolution of the ASLMNIST dataset. A deeper network is needed here to capture the intricate differences between classes in this dataset. Specifically, this CNN contains four units of a 2D convolution, followed by a max pooling layer, as well as dropout. These units are then followed by a flattening layer, that precedes a two Dense-Dropout intermediate layers, and a final Dense output layer.

Both datasets utilize a 5-fold validation technique in order to holistically measure the accuracy of the model, as well as log both macro and micro averaged metrics by way of Precision, Recall, Accuracy, F1 Scores, as well as

Confusion Matrices. In all, for both HGRD and ASLMNIST, the train/validation/test split is 80-10-10%. Both datasets utilize a batch size of `32`. For the HGRD dataset, the learning rate optimizer is Adam, and the loss value is `sparse_categorical_crossentropy`. For the ASLMNIST dataset, the learning rate optimizer is also Adam, but the loss value is simply `categorical_crossentropy`. Both datasets are trained for 10 epochs, per fold. All other hyperparameters are left to a default value as specified by TensorFlow [1].

Model Domain Transfer To further analyze the generalizability of the models trained across the HGRD and ASLMNIST, a study on the transferability of such models is also conducted. In this sense, transfer learning is applied on the CNN-based models trained in the previous section, between the two datasets. More specifically, the CNN-based model trained from the HGRD database is evaluated on the ASLMNIST dataset, and vice versa. To achieve this, the CNN models trained serve as a feature extractor by excluding the dense layers at the top of the network (i.e., "top") and replacing them. When applying the HGRD model to the ASLMNIST dataset, the "top" of the network is replaced with a flattening layer, three units of Batch Normalization-Dense-Dropout layers, followed by a final unit of Batch Normalization and an output Dense layer. When applying the ASLMNIST model to the HGRD dataset, the "top" of the network is replaced with a flattening layer, a unit of Batch Normalization-Dense-Dropout unit, with a final Batch Normalization and output Dense layer. When applying the HGRD model to ASLMNIST data, only a three-fold validation approach is used due to computational memory limits, but when ASLMNIST data to HGRD, a 5-fold validation is utilized. This yields a train/validation/test split of 80/10/10%. All other optimizers, hyperparameters, and salient variables are the same as described in the previous section. Figure 3 demonstrates a visualization of this network architecture.

3.2 Dynamic HGR

Overview For the dynamic HGR experimentation, we tested six different models across one dataset. The one dataset we used is MiniIPN, a variation of the IPN Hand dataset from Benitez-Garcia et al. that we created for this project [5]. The reasoning behind creating MiniIPN and details about MiniIPN itself are provided in **Dataset and Preprocessing**. We test the following models: 1) IPN pretrained 3D CNN; 2) Jester pretrained 3D CNN; 3) EgoGesture pretrained 3D CNN; 4) Kinetics pretrained 3D CNN; 5) ImageNet pretrained CNN-LSTM; and 6) HowTo100M pretrained TimeSformer space-time attention model. The experiments were conducted on a laptop computer with the following specifications: Intel i7-9750H 2.6GHz, 16 GB RAM, 50 GB disk space, 6GB NVIDIA RTX 2060 GPU, and CUDA 11.3.

Dataset and Preprocessing As stated before, we are using MiniIPN, a variation of the IPN Hand dataset from Benitez-Garcia et al. to develop and evaluate

our models [5]. We created MiniIPN since all of the pre-existing datasets like Jester, IPN Hand, and NVGesture contained too much data to process for the course of this project using our computational resources [5,18,19]. To create MiniIPN, we extracted video frame data for only three classes from the IPN Hand dataset out of the thirteen classes available into a new dataset. We chose the three classes based on the diversity in their portrayed action. The three classes tested were "No Gesture", "Pointing With One Finger", and "Double Click With One Finger." Since MiniIPN is a smaller subset of the IPN Hand dataset, MiniIPN is a dataset containing video data acquired from a front-facing web camera that captures participants performing hand gestures. These hand gestures are gestures that can be mapped to computer inputs like double click and pointing.

To preprocess the MiniIPN data, we first resize the individual video frames to `112x112` pixel frames. Then, we crop out a set of `32` frames from the center of the video clip to pass into the model itself. So, we will be passing a `32` frame long video into our model for prediction For the training session, we augment the video data by applying random spatial and temporal cropping of frames alongside elastic spatial distortion to train a more robust model. Additionally, we standardized the video frame tensors using the calculated mean and standard deviation values from the original IPN dataset [5].

Architectures As stated before, we tested the following six models on the MiniIPN dataset: 1) IPN pretrained 3D CNN; 2) Jester pretrained 3D CNN; 3) EgoGesture pretrained 3D CNN; 4) Kinetics pretrained 3D CNN; 5) ImageNet pretrained CNN-LSTM; 6) and HowTo100M pretrained TimeSformer space-time attention model. To set up each model for evaluation on the downstream MiniIPN task, we replaced the final linear layer in each architecture with a new linear layer that allows for classification on the MiniIPN dataset. We then train this linear layer as part of our transfer learning paradigm. This allows these models to serve as feature extractors to see how well they perform on the task at hand without having been explicitly pretrained on the task. We follow this transfer learning evaluation paradigm due to our lack of resources in training all of these architectures from scratch.

We now present a general overview and insight into each of these models, which serve as state-of-the-art models in the general video understanding problem. A 3D CNN model is a convolutional neural network which uses a 3D kernel to pass over 3D data as opposed to the more familiar 2D kernel over 2D data. The 3D data we have in our case in video frames over a set amount of time steps. For our experimentation, we evaluated the ResNeXt-101 architecture with 3D CNNs. ResNeXt-101 is a CNN architecture, which relies on skip connections in order to obtain higher quality learned representations [28]. ResNeXt-101 has mainly been contextualized in the 2D image understanding problem. However, by replacing the 2D convolutions with 3D convolution layers, we can use ResNeXt-101 to classify video in the dynamic hand gesture domain. The CNN-LSTM we investigate in our project is essentially a two-part network: a feature extractor

CNN and a memory unit LSTM. The CNN allows the network to extract salable features for the model to store and remember through its iteration over video frames and the LSTM handles the memory portion by remembering past states in the passed video frames. By combining the CNN with an LSTM, we create a video classification model that takes in one frame at a time and finalizes a prediction after all frames are processed by the CNN-LSTM duo, unlike the 3D CNN which reads all frames at one so that it can create 3D convolutional maps. For our project, we use ImageNet as a pretrained based for our CNN to serve as a feature extractor and train the LSTM portion to understand the passed-in video data. The TimeSformer model is a fairly recent innovation in video understanding proposed in 2021 by Bertasius et al. [7]. TimeSformer applies the concept of space-time attention, where patches of video frames are passed into a set of self-attention blocks from a window of time in the video. The self-attention blocks look to determine what areas in the video to follow closely to understand what is going on in the video at any point in time. This approach is similar to what happens in a Transformer block in NLP tasks, where a sentence is split into certain blocks and then passed into a self-attention scheme. In this case, each video frame is split into a fixed set of patches and a set of patched video frames from a contiguous set of time steps is passed into the space-time attention model [7]. This approach does not require a convolution heavy architecture like the previously mentioned ResNeXt-101 with numerous 3D CNN blocks.

To evaluate the performance of these models, we split MiniIPN into three parts: training, validation, and testing. Specifically 72-14-14% split between the training, validation, and testing sets respectively to keep the proportions approximately balanced to a standard 80-10-10 while also giving more data to the validation and testing sets. This splitting will let us run more testing samples and get more consolidated testing results. For specific hyperparameters, we followed PyTorch's recommended settings for its `SGD` (stochastic gradient descent) optimizer with momentum and dampening [23]. We also used `CrossEntropyLoss` as we are dealing with a classification problem.

3.3 Handle: Real-Time HGR

As proposed in this paper, Handle is a real-time HGR application, which deploys the experimental models created in this research. On a technical level, the methods for this application to realize a full-stack application. The front-end is written in React, and allows the end-user to: 1) capture custom static and dynamic data; 2) select models from which inferences are produced; 3) view the captured user-data; and 4) visualize the predictions from different models and datasets. Handle also contains a backend, written in Python. This Flask-based backend allows for model deployment and inference. Specifically, the best performing models are deployed onto the application, and via TensorFlow and OpenCV integration, are able to process user-data from the front-end in a RESTful manner. An inference is produced from processed data, and that is communicated to the front-end for the user to see.

4 Results

Herein, results are presented for the experimentation in HGR across Static and Dynamic modalities, as well as discussing the efficacy and progress on Handle. The training graphs are demonstrated in the Appendix section for reference.

4.1 Static HGR

Table 2. The Precision, Recall, F1, and Accuracy scores of HGRD.

	CNN	DenseNet	MobileNet	VGG
Precision	0.9865	0.9069	0.9692	0.9796
Recall	0.9865	0.8789	0.9670	0.9790
F1	0.9865	0.8845	0.9668	0.9790
Accuracy	0.9865	0.8789	0.9670	0.9790

Table 3. The Precision, Recall, F1, and Accuracy scores of ASLMNIST.

	CNN	DenseNet	MobileNet	VGG
Precision	0.9615	0.9931	0.9837	0.9868
Recall	0.9596	0.9918	0.9820	0.9867
F1	0.9598	0.9917	0.9819	0.9866
Accuracy	0.9596	0.9918	0.9820	0.9867

Table 4. The Precision, Recall, F1, and Accuracy scores of domain transfer.

	HGRD Into ASL Model	ASL Into HGRD Model
Precision	0.1857	0.4990
Recall	0.2041	0.3862
F1	0.1409	0.3525
Accuracy	0.2041	0.3862

HGRD Dataset Across the board, the accuracy of the four tested models (CNN, DenseNet, MobileNet, VGG) exceeded that of `0.8`, demonstrating accurate performance on the HGRD dataset. In particular, as shown in **Table 2**, the accuracy of the methods are in the order of a the custom CNN, VGG, MobileNet, and then DenseNet. DenseNet suffered the least accuracy, with a value of `0.8789`, whereas the most optimal performance was achieved by the custom CNN, with a value of `0.9865`. In general, each model demonstrated a level of consistency across all four models and each dimension of metric, indicating strong performance and stability.

ASLMNIST Dataset Similar to HGRD, the efficacy of the four tested models exceed a value of `0.9` on ASLMNIST. On average, this demonstrates higher accuracy metrics and consistency than the HGRD dataset. More specifically, in the case of ASLMNIST, the accuracy of the models are in the order of DenseNet, VGG, MobileNet, and then the custom CNN (**Table 3**). This particular ordering is unique from the HGRD dataset. The custom CNN demonstrated the least accuracy, with a value of `0.9596`, whereas the most optimal performance was achieved by DenseNet, with a value of `0.9918`. Similar to HGRD, each model demonstrated a good level of consistency across all four models and each dimension of metric, indicating strong performance and stability.

Model Domain Transfer **Table 4** demonstrates the results of the model domain transfer between HGRD and ASL. When training the ASL model on HGRD, an accuracy score of 0.2041 is achieved. Similarly, when training the HGRD Model on ASL data, an accuracy score of 0.3962 is achieved. Across all metrics, it is clear that the trained models underperformed against initial trials, as shown in **Table 2** and **Table 3**. **Figures 4-13** demonstrate the training and validation (accuracy, loss) values for reference, in the appendix.

4.2 Dynamic HGR

Table 5. The Accuracy, Precision, Recall, and Balanced Accuracy (BACC) scores on MiniIPN.

Model	Accuracy	Precision	Recall	BACC
IPN 3D CNN	0.91	0.90	0.84	0.84
Jester 3D CNN	0.83	0.55	0.60	0.60
EgoGesture 3D CNN	0.77	0.81	0.67	0.67
Kinetics 3D CNN	0.75	0.50	0.54	0.54
ImageNet CNN-LSTM	0.57	0.19	0.33	0.33
TimeSformer	0.58	0.39	0.42	0.42

Based on the results on the testing set, we find that the IPN 3D CNN model performed the best across all metrics with an accuracy of 91%, 0.9 precision, 0.84 recall, and 84% balanced accuracy. We can also see that the EgoGesture 3D CNN performed the second best in terms of precision, recall, and balanced accuracy. In terms of accuracy, the Jester 3D CNN performed the second best but the precision, recall, and balanced accuracy performance show that the model did not do well in predicting some imbalanced classes. So, Jester 3D CNN performed the third best. The Kinetics 3D CNN was the least performant of the 3D CNN models but more performant than and the CNN-LSTM and TimeSformer. After the 3D CNN models, we find neither the CNN-LSTM nor TimeSformer had good

performance on MiniIPN. The least performing model is the ImageNet CNN-LSTM with the lowest metrics across accuracy, precision, recall, and balanced accuracy followed by TimeSformer as the next worst model. **Figures 14-19** demonstrate training and validation metrics for the dynamic problem.

4.3 Handle: Real-Time HGR

The features that were achieved in this portion include: 1) a front-end where the user can select a modality (static or dynamic), as well as different types of models (CNN, DenseNet, VGG, MobileNet, ResNext, LSTM, TimesFormer); and 2) a back-end where created models are deployed and are used to inference across several types of datasets (HGRD, ASLMNIST, EgoGesture, IPN, Jester, Kinetics). From general usage, it seems as though the static recognition portion of the application fails to generalize well to user-generated custom data. Specifically, the model seems to be just as good as random guessing, meaning its unable to fit well, or inference correctly, on this unseen data. However, the dynamic portion of the application sees much better performance, and is able to correctly identify differential gestures from the user in a majority of cases, across all tested datasets (EgoGesture, IPN, Jester, Kinetics). Despite this differing model performance, the overarching goal of easing the gap in experimenting in the field of HGR is achieved. Visualizations of the interface can be seen in **Figure 20** and **Figure 21**.

5 Limitations And Future Works

Generally, the static inference results are not promising, particularly in the case of being deployed in a real-time application. More methods and techniques to improve HGR from a single image or frame would be optimal. These inaccurate inferences in real-time applications is due to a lack of noisy, diverse data. In this sense, future works will likely focus on curating large-scale datasets, similar to what exists for dynamic HGR, and retraining similar network pipelines as in this study.

Although we applied transfer learning by copying over the weights for each of these models and finetuning those models on MiniIPN in the dynamic problem, we think that the models' pre-existing weights themselves determined their performance on MiniIPN as opposed to architectural differences. Since we did not have the time to train these models from scratch on a full dynamic hand gesture recognition dataset, we suppose that if we did train all of these architectures from scratch on MiniIPN, we would be able to better hypothesize the performance differences relevant to network architectures. As networks like the 3D CNNs and TimeSformer are especially parameter heavy and complex, it was not feasible for us to train from scratch on all layers of the model due to GPU computation and time constraints. So based on the results, we can conclude that the models pretrained on hand gesture datasets more relevant to MiniIPN's gestures performed better than the other models. Future work would involve training these

models from scratch to obtain better insight into each architecture's performance without considering pretraining.

6 Conclusion

Herein, Handle presents an end-to-end open-source tool for accurate gesture detection in real-time systems for static and dynamic modalities. In the static problem, we investigated the effect of a custom CNN, DenseNet, MobileNet, and VGG on HGRD and ASLMNIST, while also following up with an additional model domain transfer study. It was found that consistently strong results are achieved across-the-board (> 0.8 for HGRD, and > 0.9 for ASL). For HGRD, optimal performance is seen with the custom CNN, with a score of 0.9865. For ASLMNIST, optimal performance is seen with the pre-trained DenseNet, for a score of 0.9918. The study in model domain transfer demonstrates limited results, however. This demonstrates the limited generalizability of the ASLMNIST and HGRD models (accuracy of 0.2041 and 0.3862, respectively). As for the dynamic problem, we studied and evaluated various SOTA pretrained models such as the IPN pretrained 3D CNN, Jester pretrained 3D CNN, EgoGesture pretrained 3D CNN, Kinetics pretrained 3D CNN, ImageNet pretrained CNN-LSTM, and HowTo100M pretrained TimeSformer space-time attention model. We followed a transfer learning paradigm to retrain and evaluate the performance of these models on a new dataset. For the transfer learning method, we created a new dataset called MiniIPN derived from the IPN Hand dataset so that we were able to actually evaluate all of the models in a reasonable time frame. We found that the IPN 3D CNN performed the best followed by EgoGesture 3D CNN, Jester 3D CNN, Kinetics 3D CNN, and TimeSformer. We also verified that the ImageNet CNN-LSTM performed the worst at the dynamic HGR task. Based on our analysis, we found that the original pretrained weights had a significant impact on a model's performance where a model originally pretrained on an HGR dataset performed better on the evaluation task compared to a model not pretrained on an HGR dataset. Considering the deployed application, while all project goals are achieved, from user study it was found that the static deployed models are highly limited in accuracy on custom data, reaffirming the results from the model domain transfer. On the other hand, the dynamic deployed models demonstrated competitive, consistent results with real-time data.

Overall, in this paper, a comparative analysis of SOTA techniques in static and dynamic HGR is demonstrated in order to investigate optimal techniques, as well as deploy custom-models in a holistic application for experimentation and transparency.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg,

J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org

2. Abdulhussein, A.A., Raheem, F.A.: Hand gesture recognition of static letters american sign language (asl) using deep learning. Engineering and Technology Journal **38**(6), 926–937 (2020)
3. Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: 2017 international conference on engineering and technology (ICET). pp. 1–6. Ieee (2017)
4. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Sequential deep learning for human action recognition. In: International workshop on human behavior understanding. pp. 29–39. Springer (2011)
5. Benitez-Garcia, G., Olivares-Mercado, J., Sanchez-Perez, G., Yanai, K.: Ipn hand: A video dataset and benchmark for real-time continuous hand gesture recognition. In: 25th International Conference on Pattern Recognition, ICPR 2020, Milan, Italy, Jan 10–15, 2021. pp. 4340–4347. IEEE (2021)
6. Van den Bergh, M., Carton, D., De Nijs, R., Mitsou, N., Landsiedel, C., Kuehnlenz, K., Wollherr, D., Van Gool, L., Buss, M.: Real-time 3d hand gesture interaction with a robot for understanding directions from humans. In: 2011 Ro-Man. pp. 357–362. IEEE (2011)
7. Bertasius, G., Wang, H., Torresani, L.: Is space-time attention all you need for video understanding? In: Proceedings of the International Conference on Machine Learning (ICML) (July 2021)
8. Choi, J., Jeong, H., Jeong, W.K.: Gadget arms: Interactive data visualization using hand gesture in extended reality. Journal of the Korea Computer Graphics Society **25**(2), 31–41 (2019)
9. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
10. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
11. Gao, Q., Liu, J., Ju, Z., Li, Y., Zhang, T., Zhang, L.: Static hand gesture recognition with parallel cnns for space human-robot interaction. In: International Conference on Intelligent Robotics and Applications. pp. 462–473. Springer (2017)
12. Garg, P., Aggarwal, N., Sofat, S.: Vision based hand gesture recognition. World academy of science, engineering and technology **49**(1), 972–977 (2009)
13. Ge, S.S., Yang, Y., Lee, T.H.: Hand gesture recognition and tracking based on distributed locally linear embedding. Image and Vision Computing **26**(12), 1607–1620 (2008)
14. Girdhar, R., Carreira, J., Doersch, C., Zisserman, A.: Video action transformer network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 244–253 (2019)
15. Huang, D.Y., Hu, W.C., Chang, S.H.: Gabor filter-based hand-pose angle estimation for hand gesture recognition under varying illumination. Expert Systems with Applications **38**(5), 6031–6042 (2011)

16. Islam, M.Z., Hossain, M.S., ul Islam, R., Andersson, K.: Static hand gesture recognition using convolutional neural network with data augmentation. In: 2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR). pp. 324–329. IEEE (2019)
17. Köpüklü, O., Gunduz, A., Kose, N., Rigoll, G.: Real-time hand gesture detection and classification using convolutional neural networks. In: 2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019). pp. 1–8. IEEE (2019)
18. Materzynska, J., Berger, G., Bax, I., Memisevic, R.: The jester dataset: A large-scale video dataset of human gestures. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops. pp. 0–0 (2019)
19. Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., Kautz, J.: Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4207–4215 (2016)
20. Otiniano-Rodrıguez, K., Cámara-Chávez, G., Menotti, D.: Hu and zernike moments for sign language recognition. In: Proceedings of international conference on image processing, computer vision, and pattern recognition. pp. 1–5 (2012)
21. Oudah, M., Al-Naji, A., Chahl, J.: Hand gesture recognition based on computer vision: a review of techniques. journal of Imaging **6**(8), 73 (2020)
22. Ozcan, T., Basturk, A.: Transfer learning-based convolutional neural networks with heuristic optimization for hand gesture recognition. Neural Computing and Applications **31**(12), 8955–8970 (2019)
23. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
24. Pinto, R.F., Borges, C.D., Almeida, A., Paula, I.C.: Static hand gesture recognition based on convolutional neural networks. Journal of Electrical and Computer Engineering **2019** (2019)
25. Smith, J.W., Thiagarajan, S., Willis, R., Makris, Y., Torlak, M.: Improved static hand gesture classification on deep convolutional neural networks using novel sterile training technique. Ieee Access **9**, 10893–10902 (2021)
26. Triesch, J., Von Der Malsburg, C.: A system for person-independent hand posture recognition against complex backgrounds. IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(12), 1449–1453 (2001)
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
28. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1492–1500 (2017)
29. Zhu, L., Tran, D., Sevilla-Lara, L., Yang, Y., Feiszli, M., Wang, H.: Faster recurrent networks for efficient video classification. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 13098–13105 (2020)

7 Appendix

7.1 Project Rubric

Project Goal:
To utilize large-scale hand gesture databases to create new, and refine existing, models of hand gesture recognition (HGR) to demonstrate state-of-the-art approaches via an application, from static to dynamic.

Project Dissemination
In effect, our goals are three-fold:

1. *Experimentation*: Our goal in the experimentation phase is to create and test models on existing databases, this includes the following subproblems:
   (a) Static
      i. We select the Hand Gesture Recognition Database and the ASL Database. Through these, we create an end-to-end deep learning pipeline by which seven unique models are created: Custom CNN, DenseNet Pretrained, MobileNet Pretrained, and VGG Pretrained. Additional studies in model domain adaptation between the Hand Gesture Recognition Database and American Sign Language Database will be performed.
   (b) Dynamic
      i. Due to time constraints and lack of computational resources, we frame the dynamic problem as a transfer learning problem. To this end, we evaluate the following transfer learning models on a single dataset. For the experimentation, we will use MiniIPN, a custom-made, smaller version of the IPN Dataset with three classes of hand gestures as opposed to thirteen classes.We evaluate the following models on the MiniIPN dataset: IPN pretrained 3D CNN, Jester pretrained 3D CNN, EgoGesture pretrained 3D CNN, ImageNet pretrained LSTM, and TimeSFormer space-time attention model.
2. *Transparency*: Our motivation is to ease the gap in experimenting in the field of HGR, as most studies are difficult to use and apply via an application:
   (a) Application
      i. The application will allow for custom images and video to be captured by the user, as well as will deploy our static and dynamic models to allow for inference from each unique dataset. In effect, the user can choose the static or dynamic subproblem from a menu, as well as the particular model to be executed, and predictions will be made from each dataset.
3. *Summary*: Our study will serve as a comparative analysis of a large set of data from different subproblems in HGR. We explore various SOTA architectures like 2D and 3D CNNs, LSTMs, and Transformers and analyze the inner workings of each of these architectures.

Grading Basis

1. An "A" grade achieves all three goals (experimentation, transparency, summary) to full completion.
2. A "B" grade achieves either: 1) 2/3 of the presented goals, or 2) all three goals, but with moderate completion.
3. A "C" grade achieves either: 1) 1/3 of the presented goals, or 2) all three goals, but with minimal completion.
4. A "D" grade achieves 0/3 of the presented goals.

7.2 Alterations To Project Rubric

Herein, an enumeration is provided on the changes that were made to the project rubric:

1. The singular change to the *static* experimentation is that instead of running seven unique experiments (Custom CNN, DenseNet, DenseNet Pretrained, MobileNet, MobileNet Pretrained, ResNet, ResNet Pretrained), the scope of the experimentation was reduced to: Custom CNN, DenseNet Pretrained, MobileNet Pretrained, and VGG Pretrained. This change was made with approval from Dr. Oguz, and was enacted due to the high redundancy and lack of value in the seven distinct experiments. For our purposes, four experiments were sufficient for a holistic coverage.
2. The one change we made to the *dynamic* experimentation is that instead of testing an NVGesture pretrained 3D CNN model, Raahul tested a Kinetics pretrained 3D CNN model. This substitution occured since the NVGesture model downloaded from an online repository was not compatible with our 3D CNN implementation due to a mismatch in weights in the NVGesture model file. We obtained approval from Dr. Oguz to replace the testing model as this change simply replaces the model tested without changing the amount of work done for this project.

7.3 Self Evaluation

Aadarsh's Evaluation Referencing the Project Rubric, I believe that Raahul and I should be awarded an "A" grade. More specifically, the criterion for an "A" includes" achieving all three goals (experimentation, transparency, summary) to full completion. To this end, this study was able to complete all goals related to experimentation across Static and Dynamic modalities with respect to the selected datasets, with minimal alterations of our original goal. Additionally, with respect to transparency, Raahul and I were successfully able to create an application which deploys our models and allows a user to produce real-time, custom gestures. Our application is able to show the inferences across several architectures and the data to which they correspond. Finally, our paper and experimentation allow for a holistic baseline and further demonstrate a comparative analysis of SOTA techniques, which can benefit future researchers and hobbyists down-the-line when working in similar domains of HGR.

Raahul's Evaluation Based on the Project Rubric, I think that Aadarsh and I should receive an "A" grade since we achieved all three goals of experimentation, transparency, and summary for an "A" grade. We were able to complete all the static and dynamic experiments with a few changes to the original rubric. Additionally, we managed to deploy a working application that demonstrates all of our models in real time as part of our transparency goal. Finally, we finished this paper and experimentation to demonstrate our analysis and insight into various SOTA techniques for dyanmic and static hand gesture recognition as part of our summary goal. Therefore, since we accomplished all of our goals, we should receive an "A".

7.4 Figures

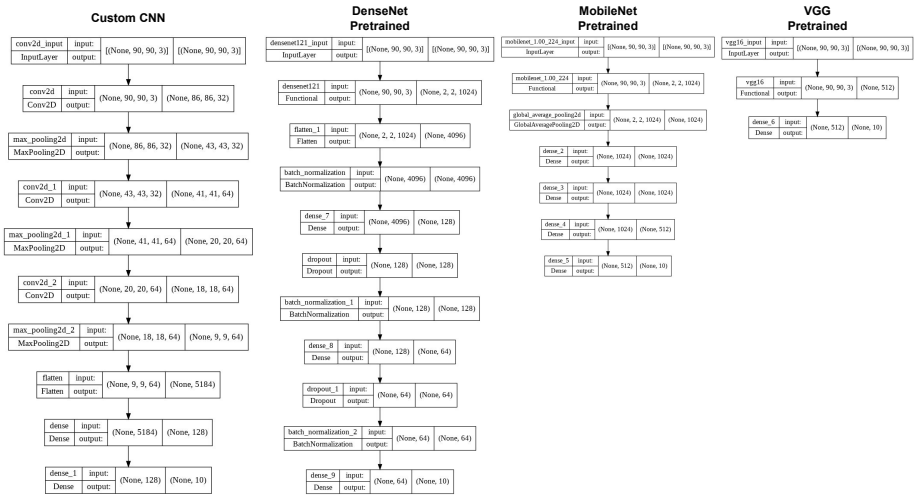

Fig. 1. This figure summarizes the architectural setup of the custom CNN, a pretrained DenseNet, a pretrained MobileNet, and a pretrained VGG. These are applied to the HGRD dataset.

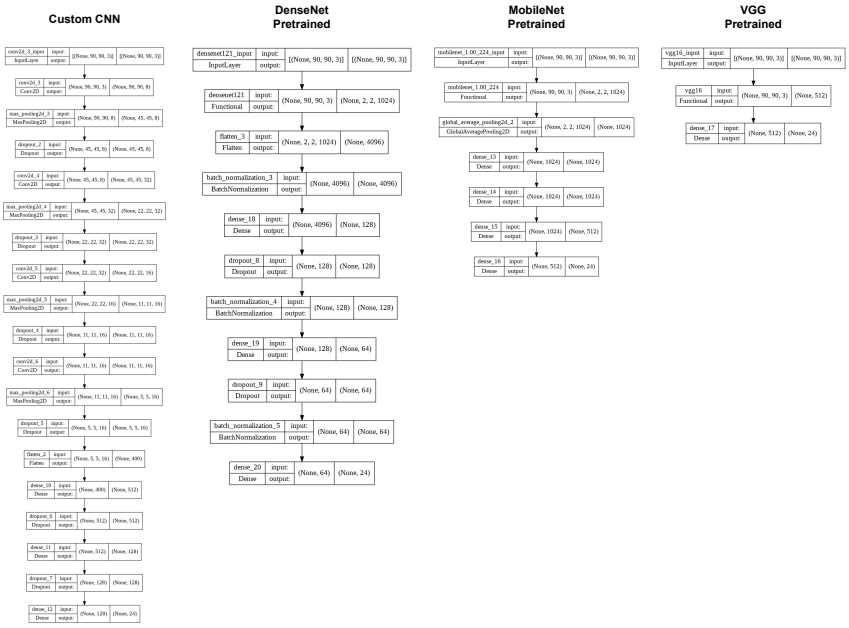

Fig. 2. This figure summarizes the architectural setup of the custom CNN, a pretrained DenseNet, a pretrained MobileNet, and a pretrained VGG. These are applied to the ASLMNIST dataset.

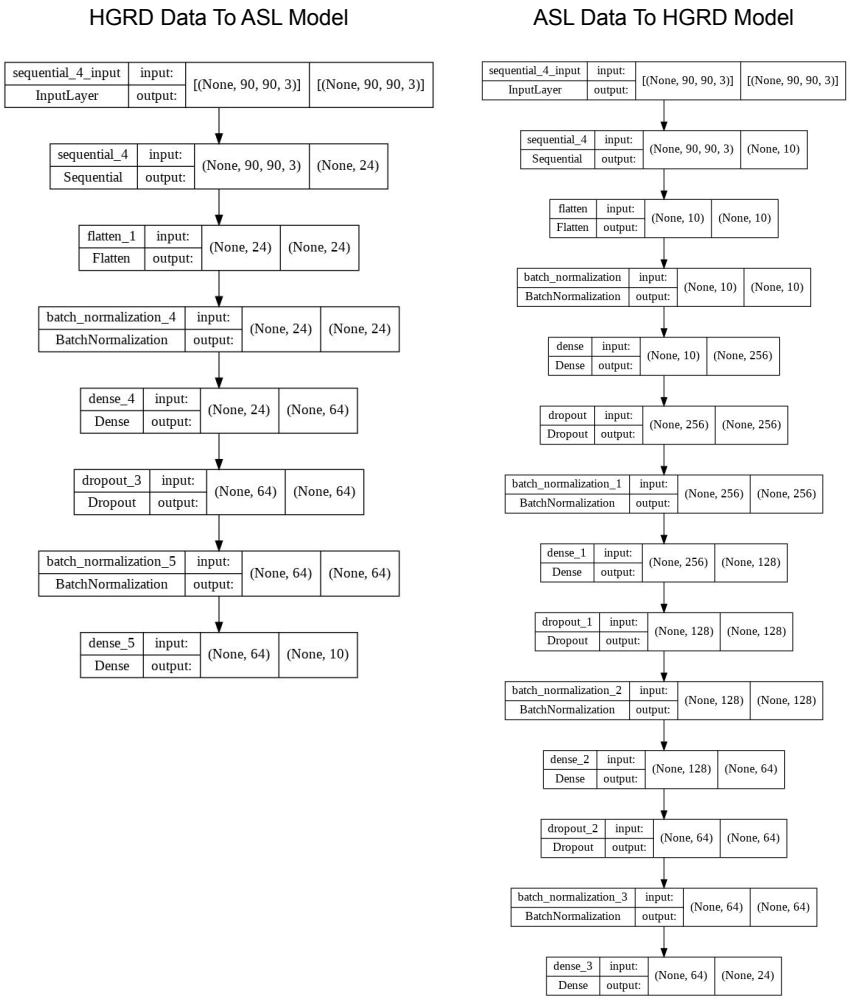

Fig. 3. This figure summarizes the architectures used in the domain transfer studies.

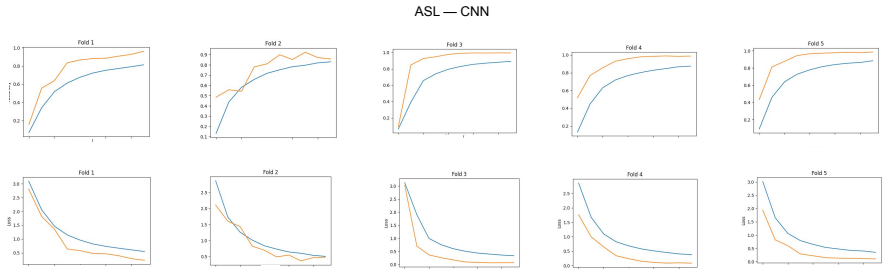

Fig. 4. This figure summarizes accuracy (top row) and loss (bottom row) of training a CNN on ASLMNIST. The orange is validation. The blue is training.

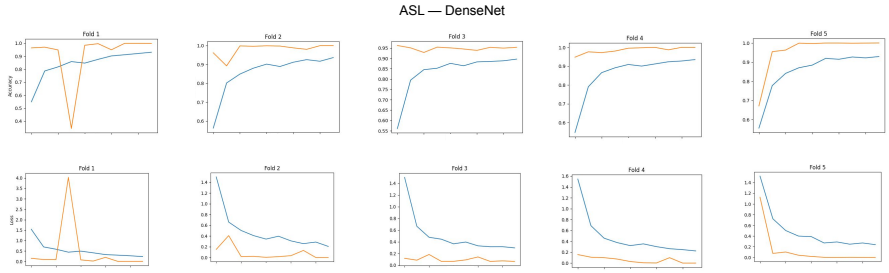

Fig. 5. This figure summarizes accuracy (top row) and loss (bottom row) of training a DenseNet on ASLMNIST. The orange is validation. The blue is training.

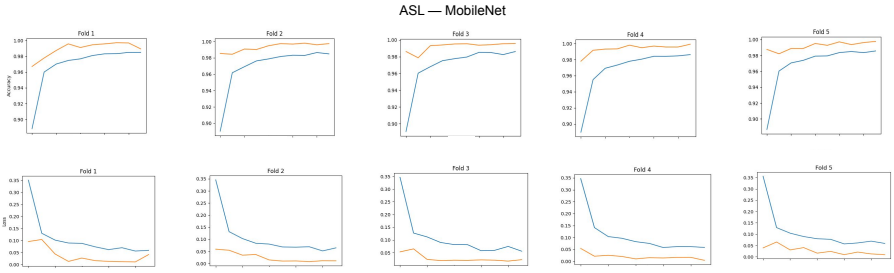

Fig. 6. This figure summarizes accuracy (top row) and loss (bottom row) of training a MobileNet on ASLMNIST. The orange is validation. The blue is training.

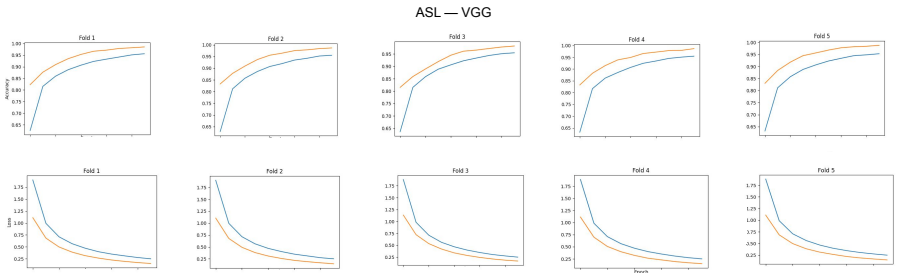

Fig. 7. This figure summarizes accuracy (top row) and loss (bottom row) of training a VGG on ASLMNIST. The orange is validation. The blue is training.

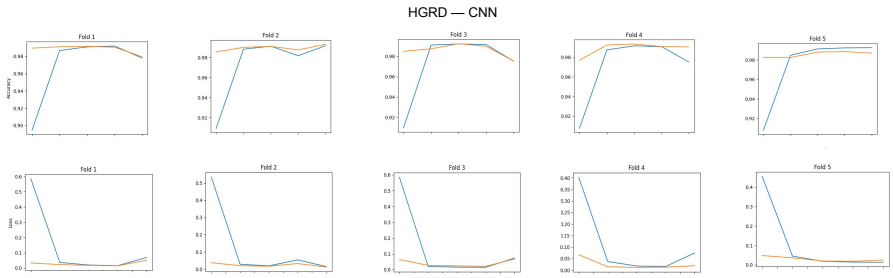

Fig. 8. This figure summarizes accuracy (top row) and loss (bottom row) of training a CNN on HGRD. The orange is validation. The blue is training.

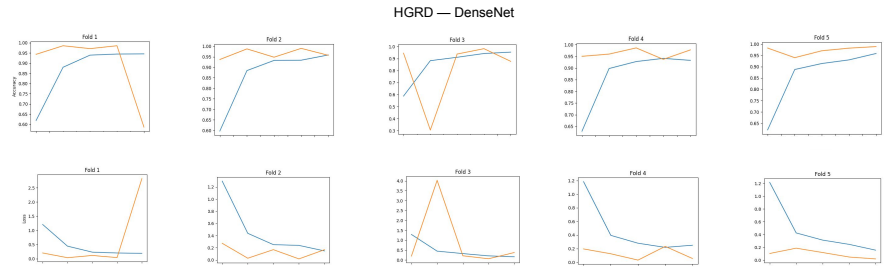

Fig. 9. This figure summarizes accuracy (top row) and loss (bottom row) of training a DenseNet on HGRD. The orange is validation. The blue is training.

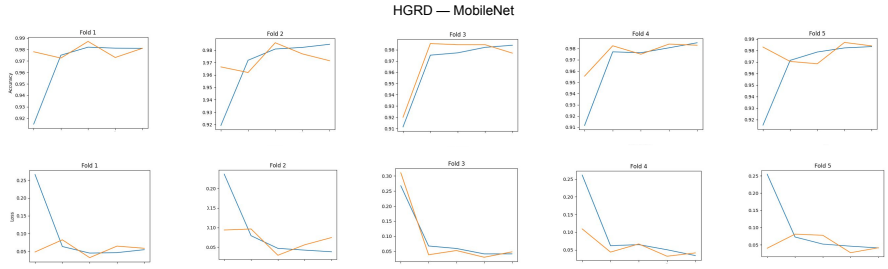

Fig. 10. This figure summarizes accuracy (top row) and loss (bottom row) of training a MobileNet on HGRD. The orange is validation. The blue is training.

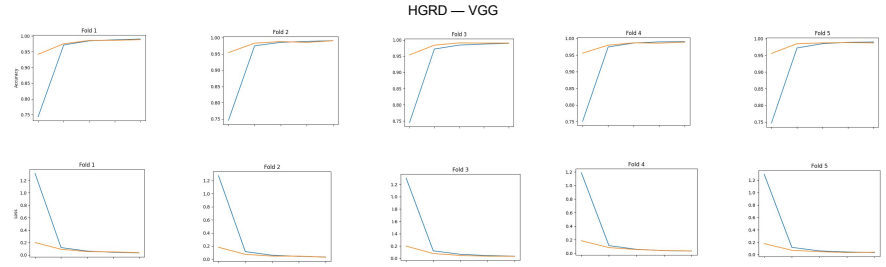

Fig. 11. This figure summarizes accuracy (top row) and loss (bottom row) of training a VGG on HGRD. The orange is validation. The blue is training.

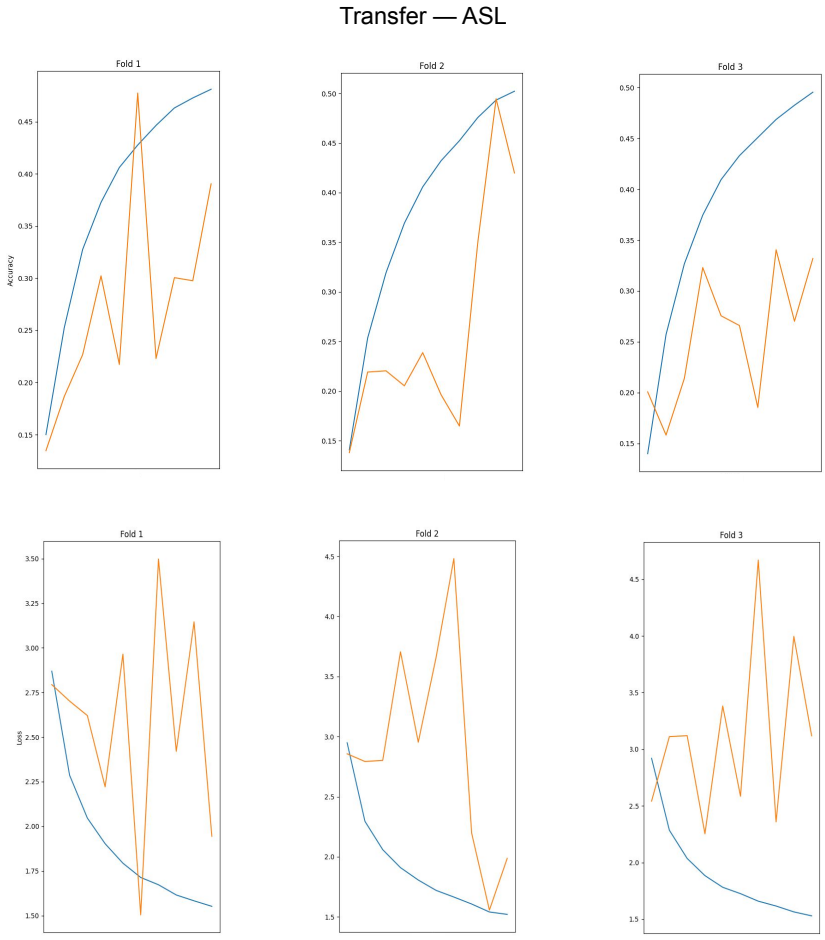

Fig. 12. This figure summarizes accuracy (top row) and loss (bottom row) of transfer learning on ASLMNIST. The orange is validation. The blue is training.

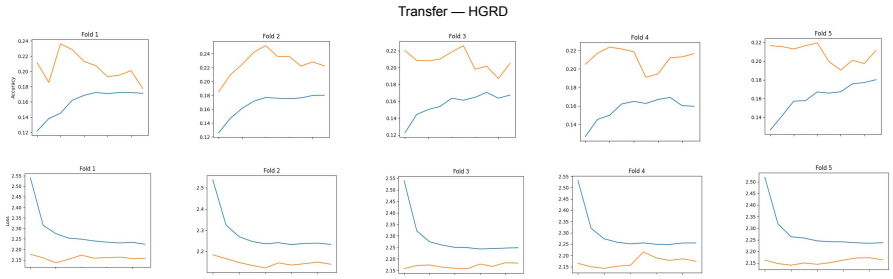

Fig. 13. This figure summarizes accuracy (top row) and loss (bottom row) of transfer learning on HGRD. The orange is validation. The blue is training.

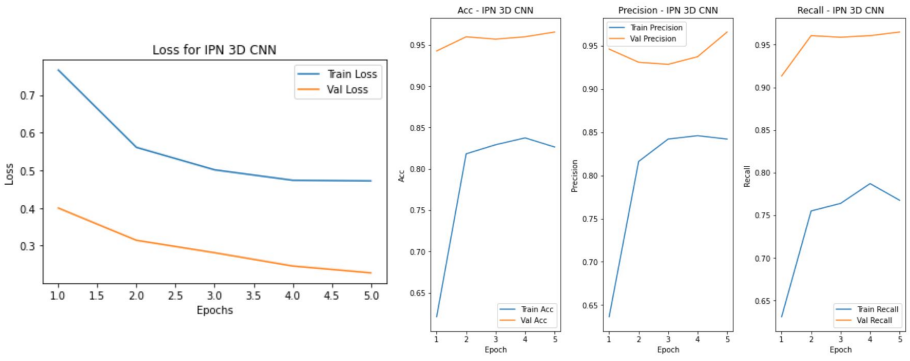

Fig. 14. This figure shows the training loss, validation loss, and metrics for IPN pre-trained 3D CNN during training

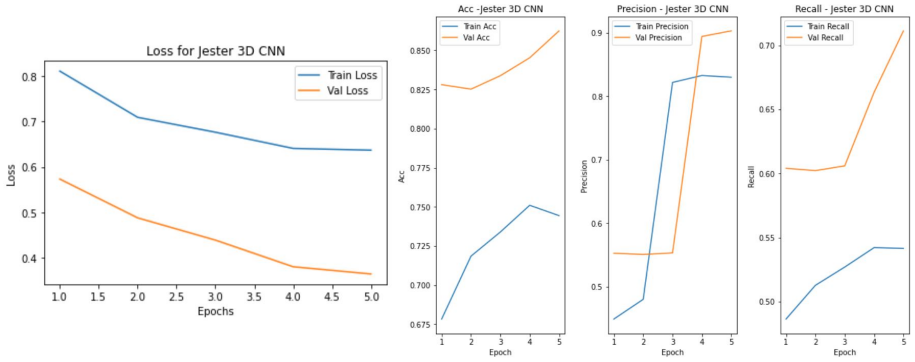

Fig. 15. This figure shows the training loss, validation loss, and metrics for Jester pretrained 3D CNN during training

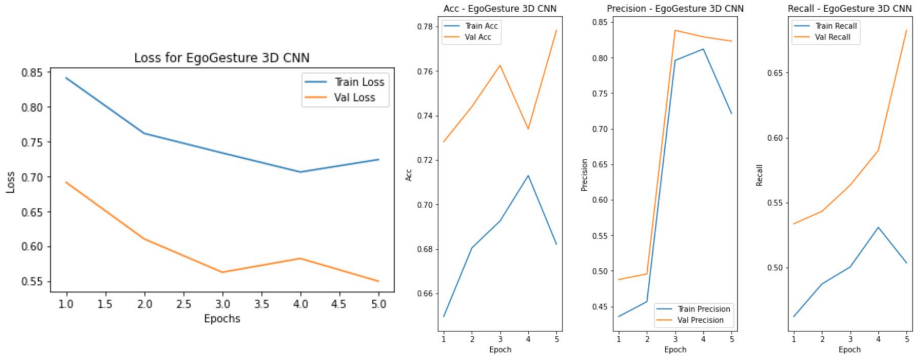

Fig. 16. This figure shows the training loss, validation loss, and metrics for EgoGesture pretrained 3D CNN during training

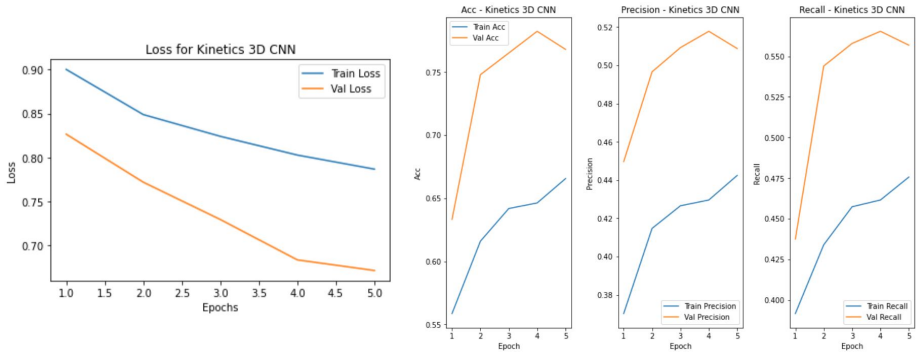

Fig. 17. This figure shows the training loss, validation loss, and metrics for Kinetics pretrained 3D CNN during training

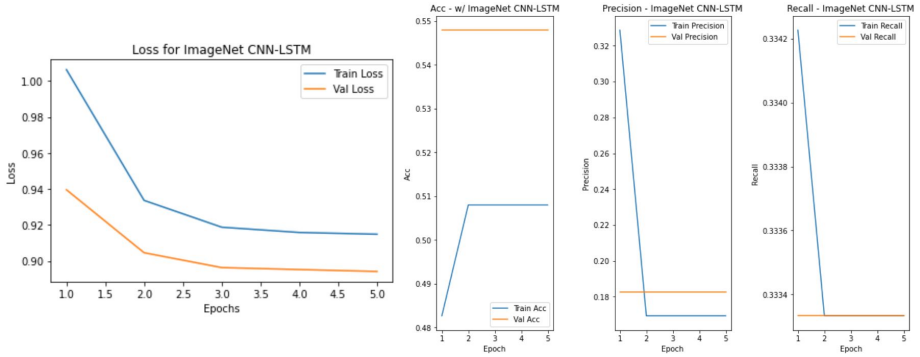

Fig. 18. This figure shows the training loss, validation loss, and metrics for ImageNet pretrained CNN-LSTM during training

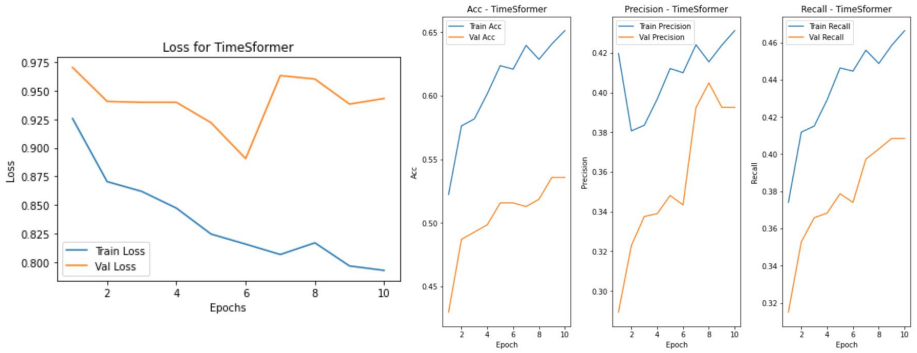

Fig. 19. This figure shows the training loss, validation loss, and metrics for HowTo100M pretrained TimeSformer during training

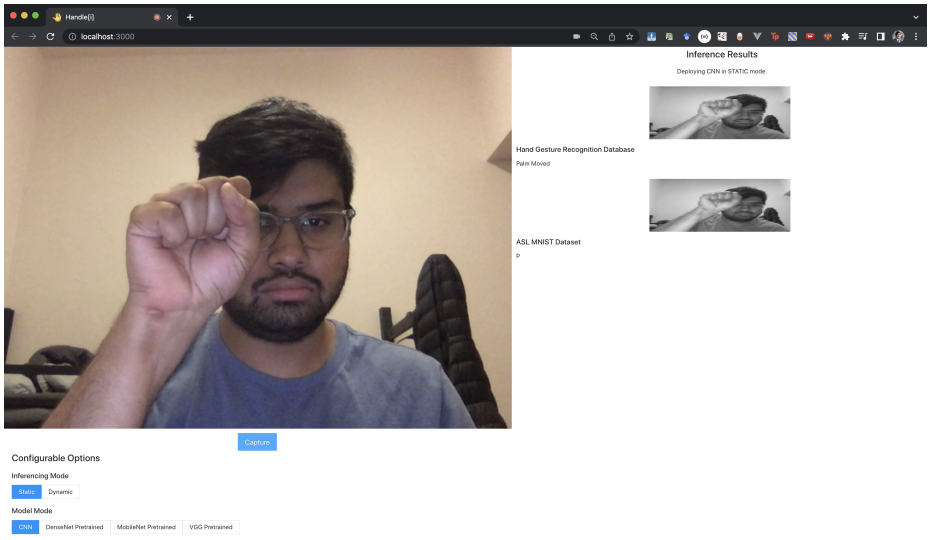

Fig. 20. This figure shows the result of an inference of a static user image. There are: 1) configurable options, 2) a capture button, and 3) inference results.

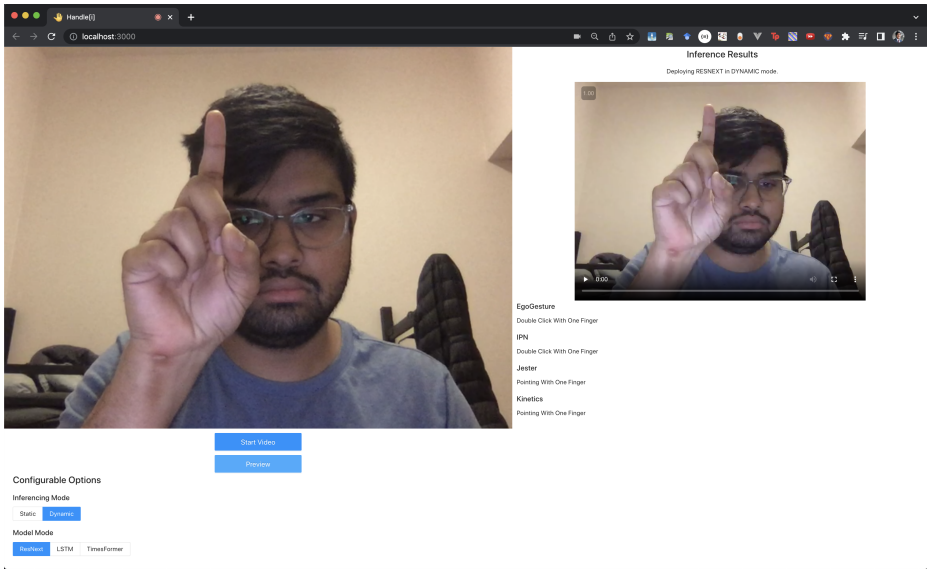

Fig. 21. This figure shows the result of an inference of a dynamic user image. There are: 1) configurable options, 2) a capture button for video, and 3) inference results.