Sign Language Recognition

Myrto Potamiti, Panagiotis Korovesis, Giannis Fourfouris

Abstract

This project presents the development of an automated system for recognizing hand gestures in various sign languages through deep learning techniques. Sign language plays a crucial role in facilitating communication for individuals who are deaf or hard of hearing, yet access to interpreters can be limited. To address this barrier, we explore multiple deep learning models trained on diverse datasets representing different sign languages, including American Sign Language, Bengali Sign Language, Kenyan Sign Language and Azerbaijan Sign Language. The study evaluates the performance of these models, highlighting the advantages of transfer learning and fine-tuning techniques over traditional neural networks. Our findings indicate that leveraging pre-trained models significantly improves accuracy while minimizing the reliance on large labeled datasets, making this approach efficient for sign language recognition tasks.

1 Introduction

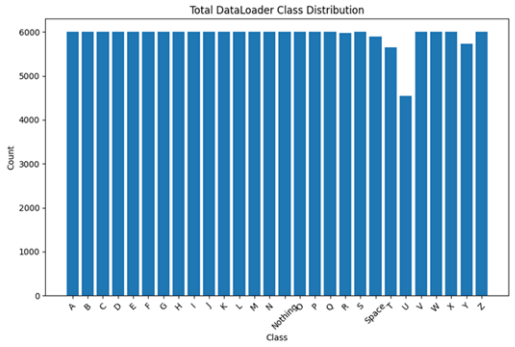
This project aims to develop an automated system for recognizing hand gestures in sign language using deep learning techniques. By classifying gestures into their respective categories, the system can help bridge communication gaps for individuals who rely on sign language. To achieve this, we explore multiple deep learning models and evaluate their performance across various datasets. The study includes an analysis of different model architectures, a comparison of their effectiveness and key observations based on experimental results.

2 Datasets Overview

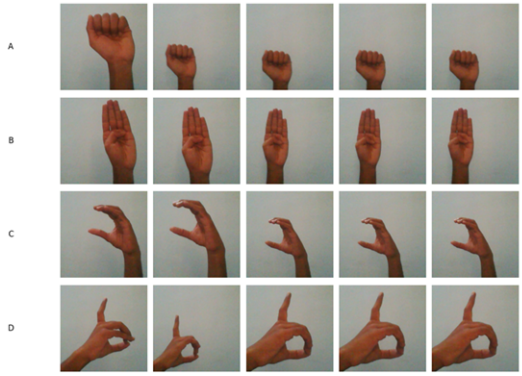
To evaluate the effectiveness of our models, we utilize six different sign language datasets. These datasets encompass a diverse range of sign languages, varying in gestures, alphabets and expressions. A detailed description of each dataset is provided below.

2.1 American Sign Language

The dataset is designed for American Sign Language (ASL) alphabet classification using neural networks to address the communication barrier faced by sign language users. Since sign language interpreters are not always available, this dataset provides a foundation for developing machine learning models that can recognize ASL hand signs. It consists of colored images representing various ASL alphabets, along with a special “nothing” and a “space” class. More info can be found in [1]



(a) Class Distribution

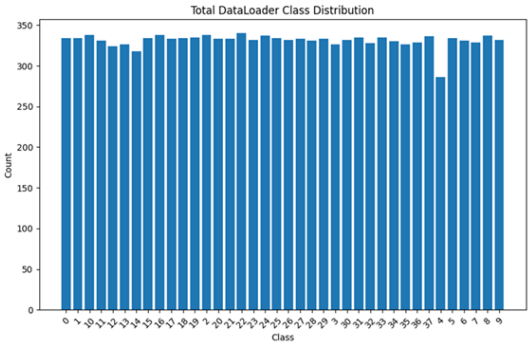


(b) Sample Images

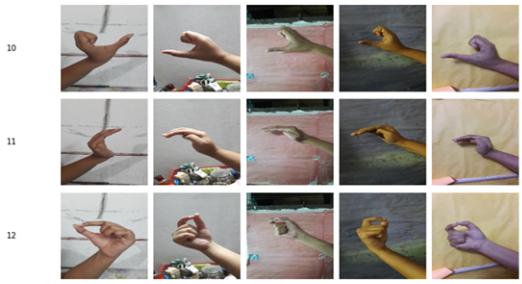
Figure 1: American Sign Language Dataset

2.2 Bengali Sign Language

The dataset focuses on Bengali Sign Language (BdSL) alphabet recognition to help address the communication challenges faced by the Deaf and Dumb (D&D) community, particularly due to the scarcity of BdSL interpreters. It includes 12,581 images representing 38 BdSL alphabets, collected in collaboration with the National Federation of the Deaf [2].



(a) Class Distribution

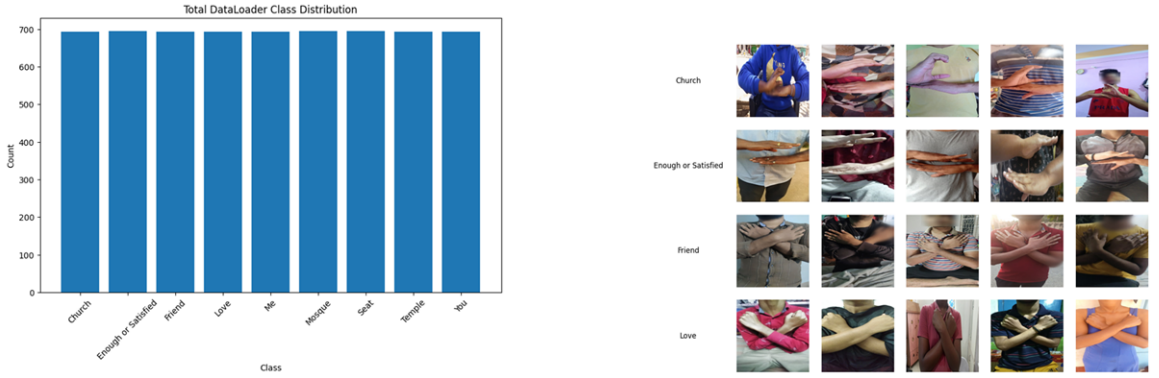


(b) Sample Images

Figure 2: Bengali Sign Language Dataset

2.3 Kenyan Sign Language

The KSLC Kenyan Sign Language dataset [3] includes 9 classes: Church, Enough or Satisfied, Friend, Love, Me, Mosque, Seat, Temple and You. Each class is represented by approximately 693 photos, aimed at enabling the classification of Kenyan Sign Language signs through machine learning models.



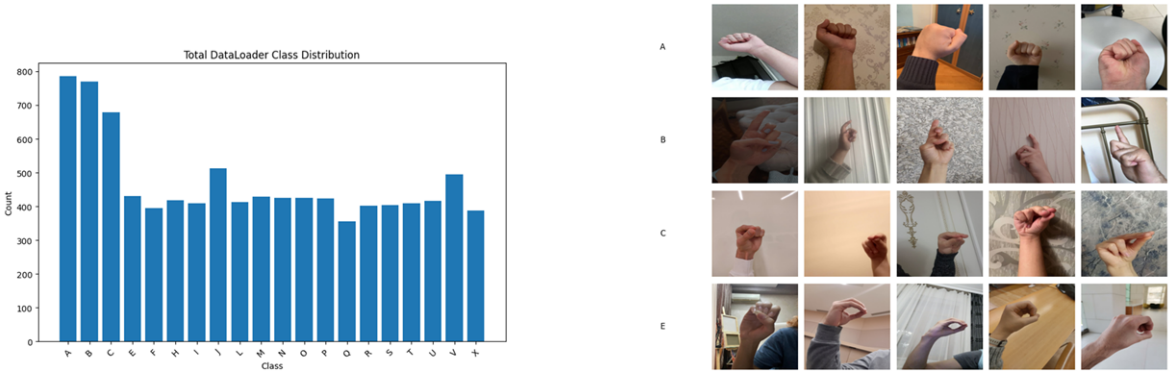
(a) Class Distribution

(b) Sample Images

Figure 3: Kenyan Sign Language Dataset

2.4 Azerbaijan Sign Language

The Azerbaijan Sign Language (AzSL) alphabet [4] consists of 32 letters, with 24 static letters interpreted through a single frame and 8 dynamic letters that involve hand motion. The dataset contains approximately 11,000 images for the static letters and 2,500 videos for the dynamic letters. However, only the images and the corresponding classes are being utilized for this project, focusing on the static representation of the AzSL alphabet.



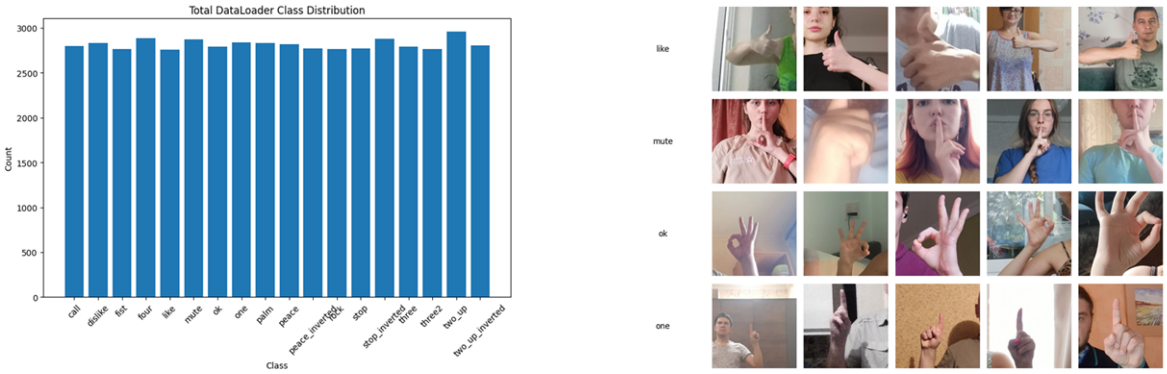
(a) Class Distribution

(b) Sample Images

Figure 4: Azerbaijan Sign Language

2.5 HAnd Gesture Recognition Image

The HaGRID (HAnd Gesture Recognition Image Dataset) is a large-scale resource designed for hand gesture recognition (HGR) systems, suitable for tasks such as image classification or detection. The dataset contains 552,992 FullHD (1920×1080) RGB images, organized into 18 gesture classes and an additional *no_gesture* class with 123,589 samples, capturing instances where a second free hand is present in the frame. With a size of 716,GB, the data is split into training (92%) and testing (8%) sets, with 509,323 images for training and 43,669 images for testing. The dataset features 34,730 unique individuals and scenes, covering subjects aged 18 to 65 years old. Collected primarily indoors, the images exhibit significant variations in lighting conditions, including both artificial and natural light, as well as extreme conditions such as facing or backing to windows. Gestures were performed at distances between 0.5 to 4 meters from the camera, making this dataset a comprehensive resource for developing HGR systems. Due to the large size of the HaGRID dataset, a 10% subset was created, maintaining the original class proportions. Additionally, all images have been resized to 512×512 pixels. This formatted subset provides a more manageable version of the data while preserving the balance of gesture classes. The dataset is described in detail in [5].

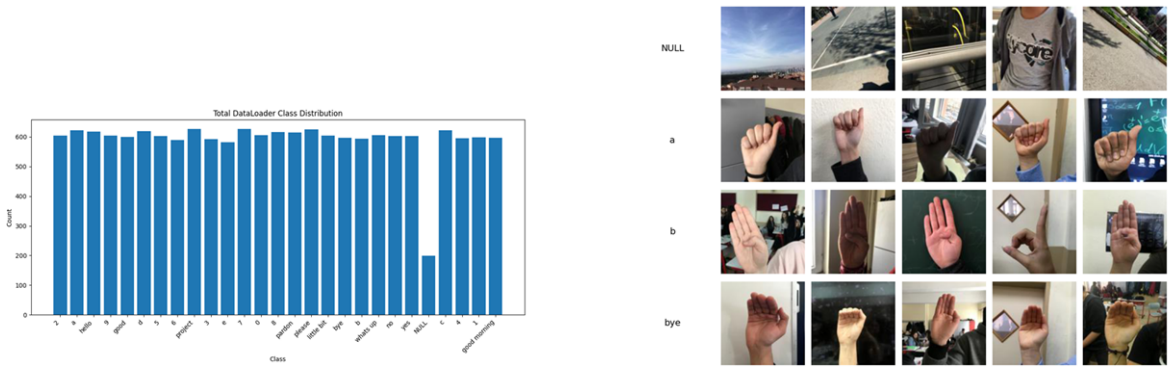


(a) Class Distribution (b) Sample Images

Figure 5: HAnd Gesture Recognition Image Dataset

2.6 27 Class Sign Language

The dataset aims to address communication barriers for speech-impaired individuals by providing a collection of American Sign Language (ASL)-based photographs gathered from 173 volunteers. It includes 27 classes: 26 for numbers, letters and expressions from ASL and 1 NULL class. The numbers range from 0-9, the letters include A, B, C, D, and E and the expressions cover common phrases like Hello, Yes, No, Good, Bye and others. The images were taken with various angles, backgrounds and lighting conditions to ensure diversity and the NULL class contains 314 photos of environments without any sign language gestures. Details of the dataset are published in the paper [6].



(a) Class Distribution (b) Sample Images

Figure 6: 27 Class Sign Language Dataset

Table 1 provides an overview of the six sign language datasets used in our study, detailing their image dimensions (width and height) and the total number of samples in each dataset.

Dataset	Width	Height	Number of Images	Number of Classes
American Sign Language Dataset	400	400	165782	28
Bengali Sign Language Dataset	224	224	12581	38
Kenyan Sign Language Dataset	512	512	6249	9
Azerbaijan Sign Language Dataset	224 (resized)	224 (resized)	9384	20
HAnd Gesture Recognition Image Dataset (10%)	512	512	50698	18
27 Class Sign Language Dataset	128	128	22801	27

Table 1: Summary of Sign Language Datasets

3 Custom Neural and Convolutional Networks

In order to establish a baseline and explore different training methods, three distinct neural network models were developed for the task.

3.1 Models Architecture

3.1.1 Simple Feedforward Neural Network

The first model is a fully connected neural network that utilizes a simple feedforward architecture. It begins with a Flatten layer, which converts the input image into a one-dimensional vector. This flattened representation is then passed through a fully connected (Linear) layer with 500 hidden units, followed by a ReLU activation function to introduce non-linearity. Finally, the network includes an output layer with units equal to the number of classes, allowing it to make classification predictions. The model is trained using stochastic gradient descent (SGD) with a learning rate of 0.001 and Nesterov momentum set to 0.9 to improve convergence speed and stability. Figure 7[1]. illustrates the network architecture.

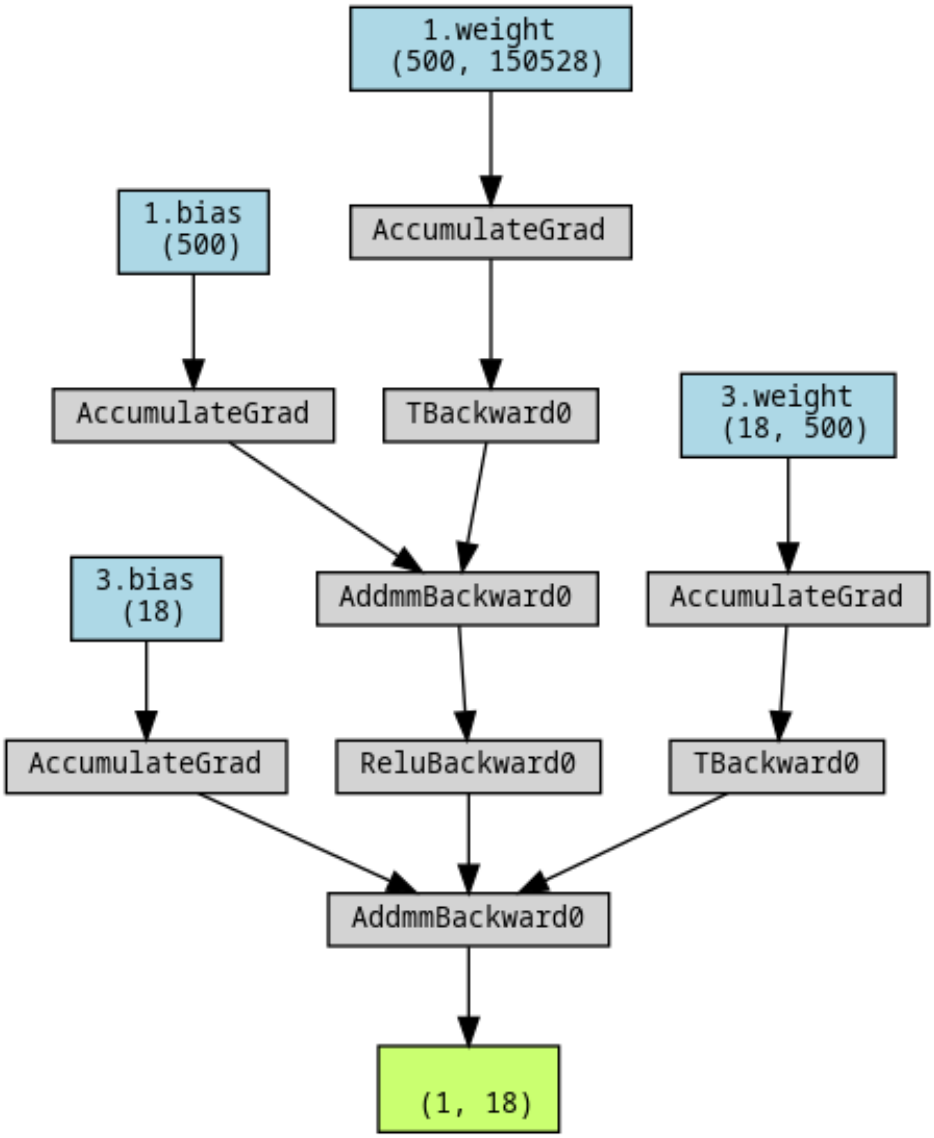


Figure 7: Neural Network Architecture

[1]All model architectures use a sample of 18 number classes.

3.1.2 Simple Convolutional Neural Network

The second model, as depicted in Figure 8, introduces convolutional layers, making it a convolutional neural network (CNN) better suited for image classification tasks. The architecture consists of two convolutional layers: the first layer applies 32 filters of size 5×5 with padding, followed by a ReLU activation function, while the second layer applies 16 filters of size 3×3 with padding, also followed by ReLU activation. After the convolutional feature extraction, the output is flattened into a one-dimensional vector and passed through a fully connected layer, which maps it to the final classification output. Similarly to the first model, this network is optimized using SGD with a learning rate of 0.001 and Nesterov momentum set to 0.9.

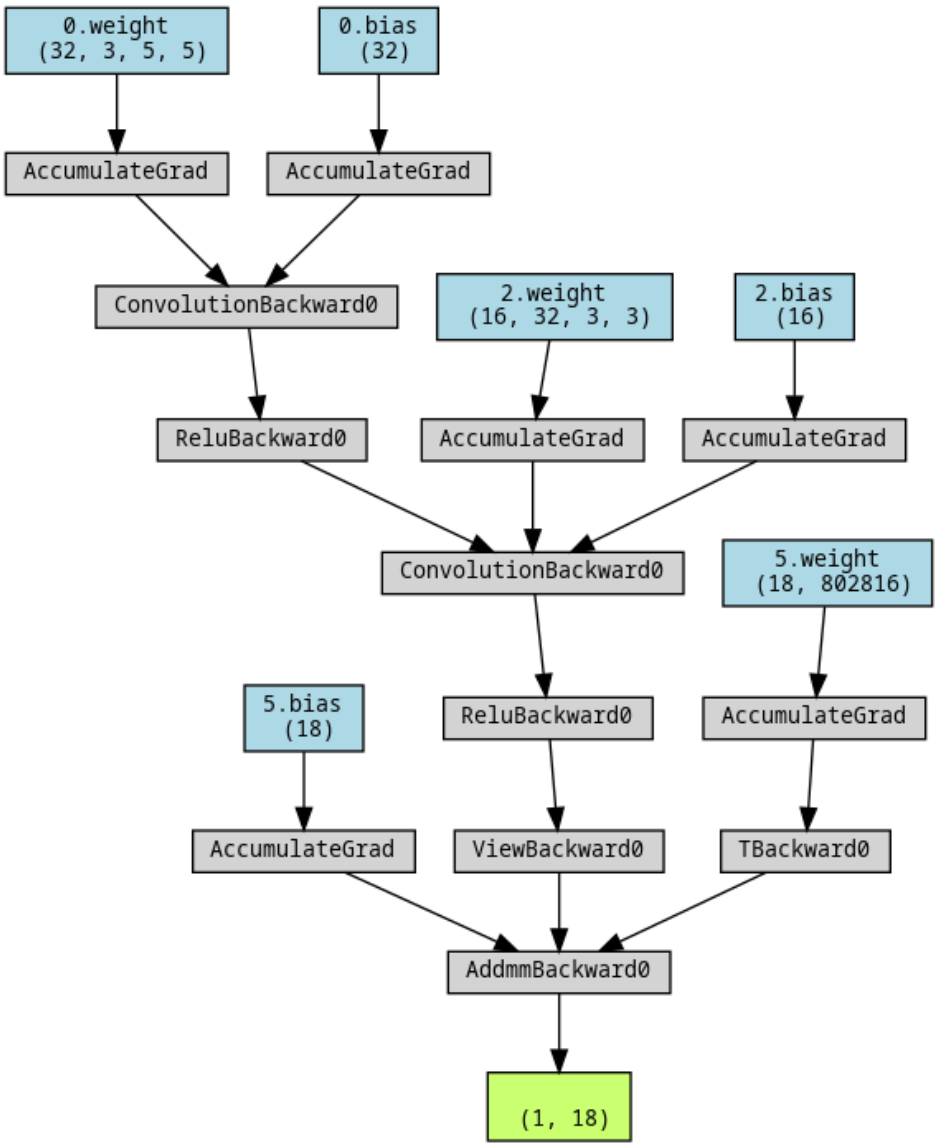


Figure 8: Convolutional Neural Network Architecture

3.1.3 Enhanced Convolutional Neural Network with Dropout

The third model builds upon the previous convolutional architecture but incorporates additional regularization techniques to enhance performance. It starts with a convolutional layer consisting of 32 filters of size 5×5, followed by batch normalization to stabilize learning and a ReLU activation function. A max-pooling layer with a kernel size of 2×2 is applied to reduce spatial dimensions and improve computational efficiency. The second convolutional layer consists of 16 filters of size 3×3, followed by

another batch normalization layer, a ReLU activation function and a second max-pooling operation. After feature extraction, the output is flattened and a dropout layer is introduced to prevent overfitting. Finally, the fully connected layer maps the extracted features to the number of output classes. Due to max-pooling operations, the input to this layer is reduced to a quarter of the original spatial dimensions. The model is trained using the same SGD optimizer with Nesterov momentum and early stopping with a value of 5 is enabled to halt training when performance stops improving. The final structure is shown in Figure 9.

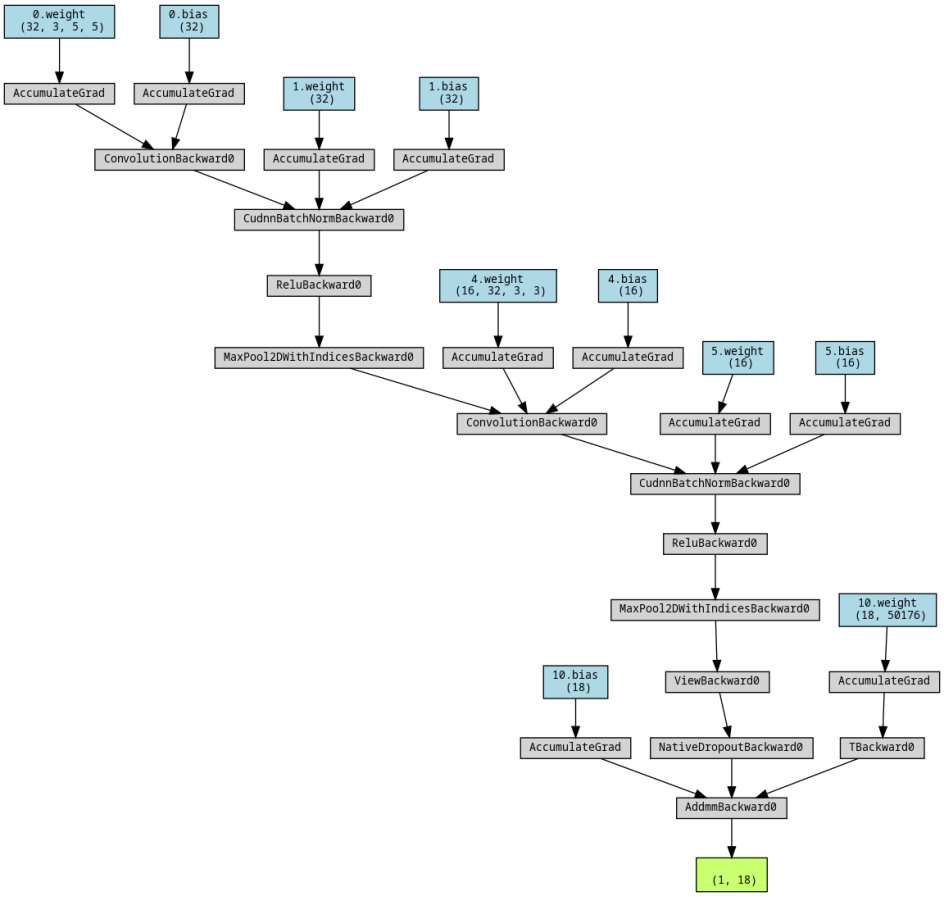


Figure 9: Enhanced Convolutional Neural Network Architecture

3.2 Training and Evaluation process

The training process [2] follows a standard supervised learning pipeline, where model parameters are iteratively updated using mini-batch stochastic gradient descent (SGD). To evaluate the model's performance and mitigate overfitting, the dataset is split into three distinct sets: training, validation and test. The model is trained on the training set, while the validation set is used throughout the process to monitor the model's performance. Additionally, normalization is applied to the input data to ensure consistent scaling and improve convergence during training.

During each epoch, the training loop iterates through the training set, performing forward and backward passes for each mini-batch. In the forward pass, input images are passed through the network to compute class scores and the loss is calculated using cross-entropy. In the backward pass, gradients of the loss with respect to the model parameters are computed. These gradients are then used to update the model's parameters via the optimizer.

The validation set is periodically used to assess the model's generalization performance and help detect overfitting. Finally, the test set is reserved for evaluating the model's final performance after

[2]Training was conducted on a local NVIDIA Quadro M4000 GPU (8GB RAM).

training, ensuring that the results are unbiased and reflect the model's ability to generalize to unseen data.

After training, the model is evaluated on the test dataset, with performance quantified using accuracy or the proportion of correct predictions. Additionally, a confusion matrix for each class is generated to visually represent the true positives, false positives, true negatives and false negatives, providing an overview of the model's classification performance.

3.3 Results

Dataset	Epochs	Elapsed time (s)	Train Accuracy (%)	Val Accuracy (%)	Test Accuracy (%)	Train Loss	Val Loss
NN							
American Sign Language	5	6512.31	100.00	99.97	99.98	0.000898	0.00
Bengali Sign Language	20	829.57	75.86	36.64	36.29	1.0286	2.37
Kenyan Sign Language	10	843.69	99.98	31.46	31.62	0.06037	2.65
Azerbaijan Sign Language	20	6383.10	95.25	24.73	25.95	0.53	2.99
Hand Gesture Recognition Image	20	12947.88	78.09	21.67	21.81	0.9427	4.71
27 Class Sign Language	20	307.26	97.06	52.56	53.96	0.3709	1.84
CNN							
American Sign Language	5	13094.71	99.96	99.75	99.72	0.0	0.01
Bengali Sign Language	20	1316.89	100.00	26.58	24.46	0.00648	5.80
Kenyan Sign Language	10	1665.05	100.00	32.90	34.66	0.00289	3.10
Azerbaijan Sign Language	20	6592.40	100.00	22.97	23.72	0.00243	6.36
Hand Gesture Recognition Image	20	25668.94	100.00	20.17	21.61	0.000364	10.58
27 Class Sign Language	20	668.49	99.99	58.68	58.22	0.00234	2.98
Enhanced CNN & Dropout							
American Sign Language	5/5	10601.88	99.94	99.80	99.72	0.0305	0.01
Bengali Sign Language	8/20	467.40	98.08	39.54	41.62	0.6928	2.93
Kenyan Sign Language	8/10	1058.79	63.31	27.62	33.86	1.3081	2.28
Azerbaijan Sign Language	8/20	2605.08	98.42	29.74	27.23	0.4337	3.46
Hand Gesture Recognition Image	7/20	9013.47	99.74	20.97	20.96	0.0478	6.66
27 Class Sign Language	14/20	420.94	97.38	67.58	65.89	0.4554	1.29

Table 2: NN and CNNs Overall Results

The experimental results, as depicted in Table 2, reveal significant variations in model performance across the different sign language datasets and the three baseline models. The Neural Network model performed exceptionally well on the American Sign Language dataset, achieving a high test accuracy of 99.98% with minimal loss, indicating the dataset's simplicity. However, its performance dropped considerably on other datasets, such as Bengali Sign Language and HaGRID Classification, with test accuracies as low as 21.81%, highlighting the challenges posed by dataset complexity. Convolutional Neural Networks (CNNs) showed improved training accuracy, often reaching 100% across datasets, but validation and test accuracies remained low in most cases, signaling overfitting. This is further evidenced by the noticeable gap between training and validation loss, where models achieved near-zero training loss but much higher validation loss (e.g., 10.58 for HaGRID with CNN), suggesting that the models memorized training data but failed to generalize well to unseen data. The Enhanced CNN & Dropout model, while mitigating overfitting to some extent, still struggled with generalization, particularly on the HaGRID and Azerbaijani Sign Language datasets, as validation loss remained high despite the reduced training loss.

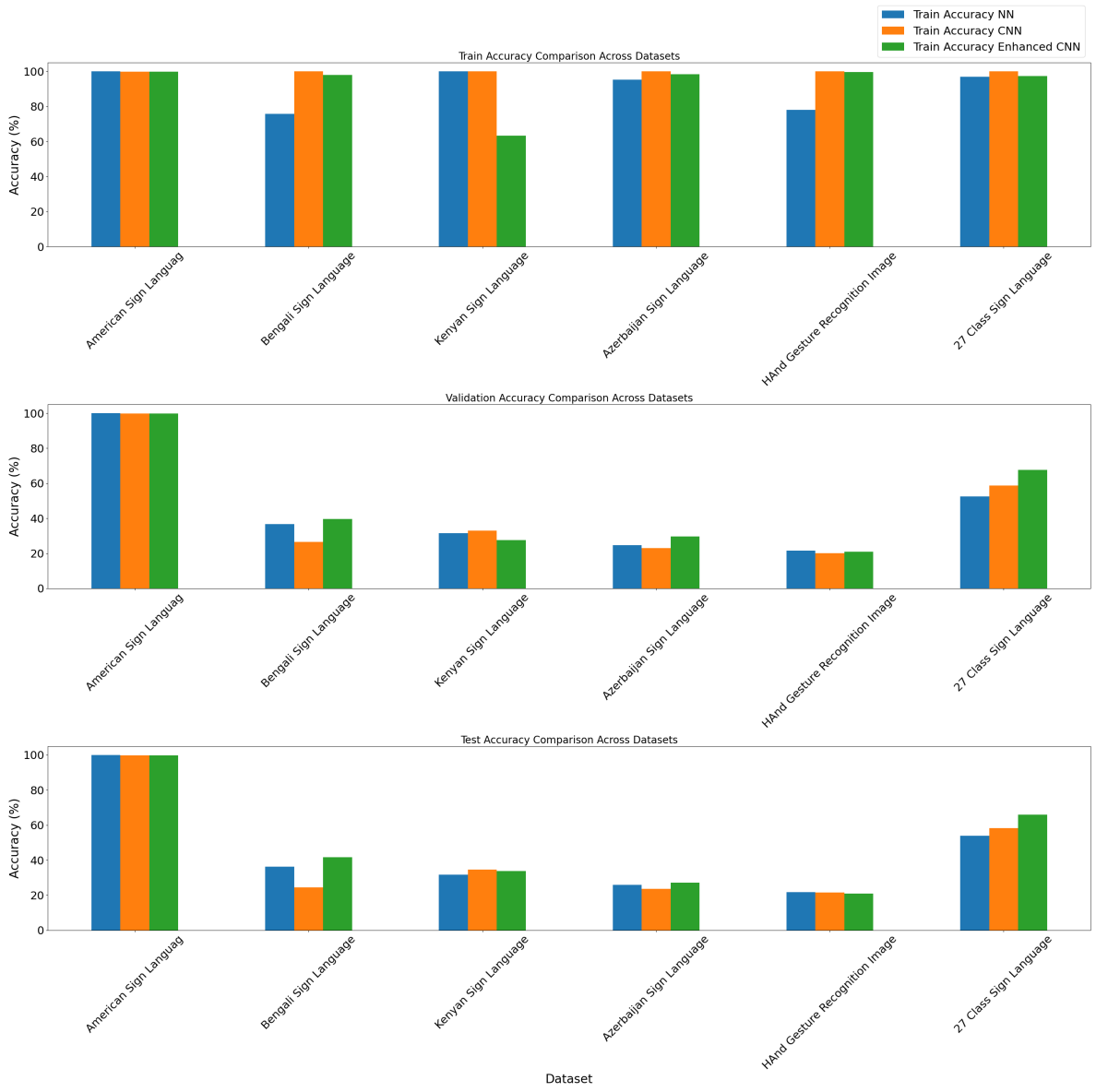


Figure 10: Accuracies for Each Model Across Different Datasets

Additionally, the observed loss patterns, as shown in 11, provide valuable insights into model performance. For instance, a low training loss combined with a high validation loss, as seen in the American Sign Language and Hand Gesture Recognition datasets, indicates overfitting. This suggests that while the model performs well on the training data, it struggles to generalize to new, unseen data. In contrast, high training and validation losses, as observed in Bengali and Kenyan Sign Languages, point to difficulties in learning from the training data, likely due to its complexity or poor quality. This trade-off emphasizes that low training loss with high validation loss signals overfitting, while high losses across both sets may suggest underfitting or insufficient model capacity.

Notably, the 27 Class Sign Language Dataset demonstrated consistent improvement across all model types, with the Enhanced CNN & Dropout model achieving the highest test accuracy (65.89%) and the lowest validation loss (1.29), indicating better generalization. A complete overview of the accuracies is presented in Figure 10. Due to sizing limitations, accuracy and loss versus epoch plots for each dataset and model, obtained during the training phase, are included in the Appendix section. Overall, these results underscore the need for dataset-specific architectural tuning and robust regularization techniques to enhance generalization.

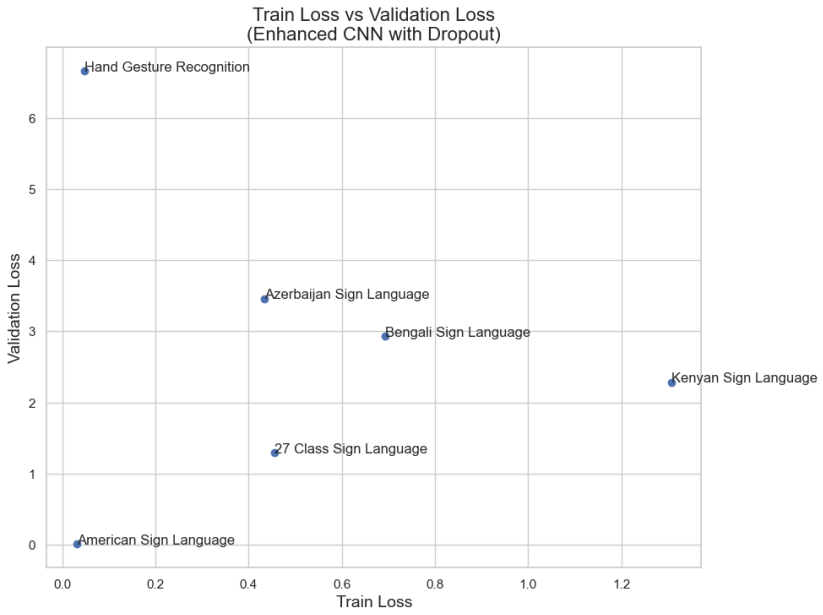


Figure 11: Train Loss vs Validation Loss - Enhanced CNN with Dropout

Regarding the training elapsed times the models vary significantly across different datasets and model types as shown in Figure 12. For instance, the American Sign Language dataset, which achieved near-perfect accuracy, required 6512.31 seconds for the NN model and 13094.71 seconds for the CNN model, indicating that the CNN architecture, while more complex, took approximately twice as long to train. Similarly, the HaGRID Classification 512p (10%) dataset, which is larger and more complex, required the longest training times: 12947.88 seconds for the NN model and 25668.94 seconds for the CNN model. In contrast, smaller datasets like the 27 Class Sign Language Dataset had much shorter training times, with 307.26 seconds for the NN model and 668.49 seconds for the CNN model. The Enhanced CNN & Dropout model generally showed reduced training times compared to standard CNNs, as seen in the Bengali Sign Language Dataset (467.40 seconds vs. 1316.89 seconds) and the KSLC Kenyan Sign Language Classification (1058.79 seconds vs. 1665.05 seconds). This reduction in training time is primarily due to early stopping, which halts training once the model's performance on the validation set stops improving, thereby preventing unnecessary computation. While this approach improves computational efficiency, it also helps avoid overfitting and enhances generalization. Overall, the elapsed times reflect a trade-off between model complexity, dataset size, and the use of optimization techniques like early stopping.

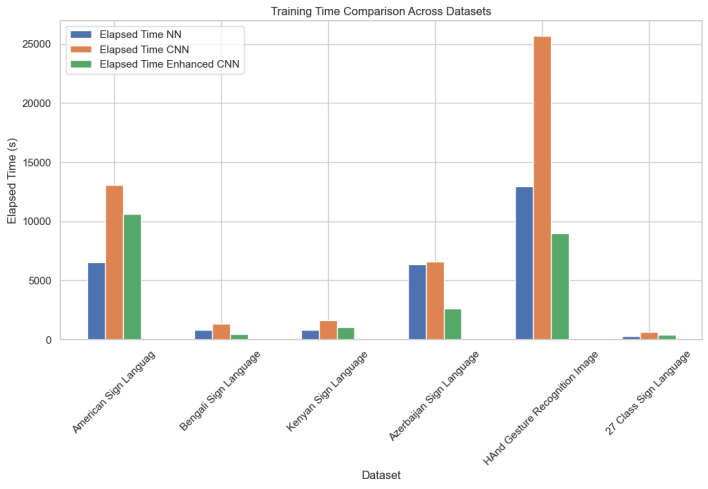


Figure 12: Training Times for Each Model Across Different Datasets

4 Transfer Learning

The second approach to address the image classification problem is relying on pre-trained models and fine-tune them for each of our datasets.

Transfer Learning allows us to leverage the knowledge of already trained models and, often achieve satisfactory performance with a fraction of the training time. There are several approaches when considering transfer learning, each with its own advantages and considerations.

One of the simplest methods is to freeze the pre-trained model and create a new classification head. This allows us to train only the classification head, which essentially maps the features produced by the base model to our specific dataset classes. The main benefit of this method is the reduced train time, as the number of trainable parameter depends only on the classification head, which is considerable smaller, when compared to the entire network. However, this approach has some significant limitations. The final performance is heavily linked with the dataset the original model was trained on. If our dataset shares similarities with the original one we can expect to see a comparable performance. Training on a substantially different dataset will lead to sub-optimal performance, as the features the model extracts may be irrelevant for the classification of dataset's images

The second approach, which is not explored in this research, involves gradually un-freezing layers from the base model, starting from the ones just before the classification head and going backwards. This method can improve the performance of a model that struggles when using only the classification head. Allowing the fine-tuning of more layers can help the model better adapt to the characteristics of our dataset, producing higher results. The optimal number of layers to un-freeze for the task depends on the dataset the architecture of the base model. Typically, this number is determined through trial and error. The apparent drawback is the clear trade-off between layers and train time. Increasing the number of trainable layers, and thus the trainable parameters leads to longer train times. A sub-category of the gradual un-freezing of layers is to fine-tune the entire model. However, this carries a higher computational cost and depending on the dataset may not be necessary.

4.1 Model Architecture

We perform extensive experiments on each dataset using two established models as the base for our fine-tuning. To ensure a fair comparison we made sure that both of the selected models were trained on the *ImageNet* dataset.

4.1.1 MobileNetV2

The first model is **MobileNetV2** [7]. It is comprised of two types of blocks. The first is a residual block with a stride of 1 and the second a block with stride 2 for downsizing. There are in total 3 layers for each type of block. The first layer is a point-wise convolution (1x1) which is used to create new features by combining the different channels. The second layer is a depth-wise convolution. It applies a single convolutional filter per input channel. The last layer is a point wise convolution but without any non-linearity. It's also important to note that the authors of this model decided to use the *ReLU6* activation function to mitigate the exploding gradients issue. The complete model architecture can be seen in Figure 13.

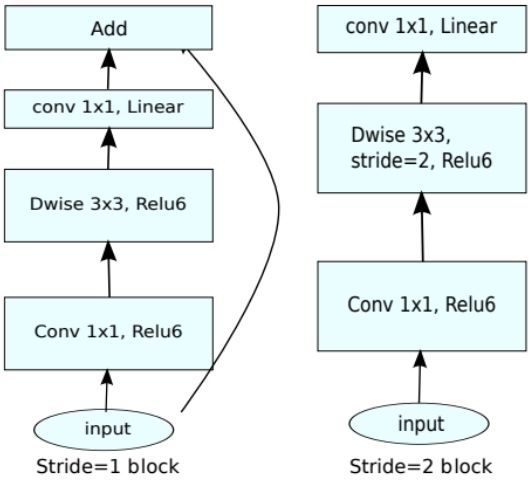


Figure 13: MobileNetV2 Architecture

4.1.2 ConvNetXt

The second model is **ConvNetXt** [8]. Architecture wise, it's essentially ResNet50, a classic convolution model, with many significant improvements. To name a few the authors replaced the ReLU activation function with *GeLU* and the Adam optimizer with *AdamW*. Inspired from the Swin Transformer, they changed the ratio of blocks per layer to the one used in the Swin architecture. For the stem layer they replaced the existing convolution kernels with non-overlapping ones with a size of 7x7. Figure 14 showcases the differences between a typical ResNet Block and the modified ConvNetXt Block.

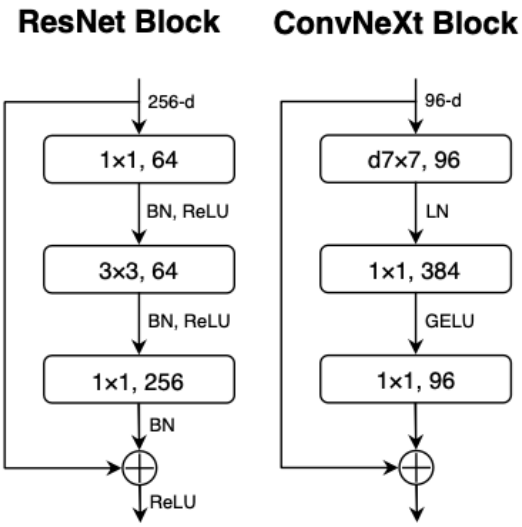


Figure 14: ResNet - ConvNetXt Block comparisson

4.2 Training

To train and evaluate the models on our sign language datasets we chose two approaches.

First we replaced the classification head with a new one and froze the rest of the network. Our reasoning behind this decision was to observe the models ability to extract features, when faced with other, un-known datasets and then compare their performances.

Second, we chose the best model from the first step and fine-tuned the entire model. Our reasoning behind this was to observe the performance gain by allowing all the model's parameters to be trainable

to underline the trade-off between training time and trainable parameters.

Due to hardware restrictions, the entire model fine-tune for each dataset was conducted on the model that scored the highest in the Classification head experiments. Our reasoning behind this was to observe the performance gain by allowing all the model's parameters to be trainable.

4.3 Experiments

4.3.1 Setup

For each dataset we performed *7* experiments.

The first *3* use **MobileNetV2** [7] as the base model. We use a *GlobalAveragePooling2D* layer to reduce the number of input parameters for the classification head. We then use two *Dense* layers as the classification head with a *BatchNormalization* Layer between them. ReLU is used as the activation function.

The secod *3* use **ConvNetXtBase** [9] as the base model. The classification head remains exactly the same to ensure a fair comparison.

The **hyperparameter** in both cases is the *hidden size* of the first Dense layer. We experiment with three values: 128, 256, 512. To avoid over-fitting the data, we use the **Early Stopping** functionality provided by the Keras library. Our optimizer of choice is **Adam** and the learning rate is set to 0.001.

When fine-tuning the entire network, we opted for a reduced learning rate of 0.0001. This choice aligns with common practices in the research community, as a lower learning rate helps to stabilize the training process. By preventing the model's weights from shifting too drastically, we can maintain the existing correlations learned during the initial training phase.

4.3.2 Results

The complete results for the fine-tuning process can be found on Table 3. The bold numbers show the best test accuracy for each model with each hidden size.

An observation that applies across all datasets is the **higher performance of MobileNet** [7] regardless of hidden size. This finding is interesting given that MobileNetV2 is smaller in size compared to ConvNetXtBase [9]. One explanation for this behavior could be the dataset size. It is reasonable to assume that since ConvNetXtBase is larger, it requires a larger and more diverse dataset to prevent over-fitting the train data. This claim is further supported by the fact than in some datasets such as *Azerbaijan, Bengali* and *Hagrid* the optimal ConvNetXt was the one with the smallest available hidden size, whereas MobileNetV2 performed best with the largest available. In the Appendix section, we provide the accuracy and loss versus epoch plots for each dataset and for each model.

The second conclusion we can extract by looking at the results regards the **complexity of each dataset**. It's clear that the *American Sign Language* dataset was too simple, which lead to a 0.99% test accuracy for both models, using the smallest hidden size. Figure 1 proves that the dataset lacks diversity such as different backgrounds. Additionally, it features only the hand gestures, without any representation of the individuals performing them.

We can also observe a **correlation between the hidden size and the number of epochs**. As the hidden size increases, the number of epochs decrease, to prevent the model from over-fitting. This is particularly evident for the *27 Class* dataset, where the hidden size of 128 allowed both models to perform a training of 10 epochs, whereas all other hidden sizes lead to a 3 epoch training process. A larger classification head can capture more information and is thus more prone to over-fitting. This persists, despite the large dropout rate of 0.4 that we have incorporated in the classification head architecture.

Finally, studying the scores produced by MobileNetV2, we can see that allowing the **entire model to train** resulted in significant increases in the accuracy of the model. The average increase is 32.40%, with the highest being 47.69% for the Hagrid dataset. It's also impressive that for the *27 Class Sign Language* dataset, the accuracy reached 0.99%.

The number of total epochs remained the same between the classification head and the entire model with the exception of the *27 Class* dataset, which caused the epochs to drop from 10 to 8 when training the entire network. The **total train time** also increased with an astonishing rate of 358%. However, for these datasets, the overall train time still remains small and we could confidently say that the performance gains outweigh the addition train overhead.

Dataset	Model	Head Only	HS	Train Acc	Val Acc	Test Acc	Train Loss	Val Loss	Test Loss	Elapsed Time	Epochs
American Sign Language	MobileNetV2	YES	128	0.9997	0.9999	**0.9999**	0.00024	0.00016	0.00069	6.55 min	2
American Sign Language	ConvNeXtBase	YES	128	0.9994	1.000	**0.9999**	0.0034	2.9016e-4	2.9016e-4	?	?
Azerbaijan Sign Language	MobileNetV2	YES	128	0.8144	0.6562	-	0.6383	1.1579	-	7.6 min	7
Azerbaijan Sign Language	MobileNetV2	YES	256	0.8847	0.6581	-	0.4172	1.0664	-	7.6 min	7
Azerbaijan Sign Language	MobileNetV2	YES	512	0.9519	0.6977	**0.6266**	0.2210	1.0578	1.2575	7.8 min	7
Azerbaijan Sign Language	ConvNeXtBase	YES	128	0.7162	0.5409	**0.5143**	0.5409	1.4811	1.5030	11.26 min	8
Azerbaijan Sign Language	ConvNeXtBase	YES	256	0.5742	0.4724	-	1.3781	1.7600	-	4.53 min	3
Azerbaijan Sign Language	ConvNeXtBase	YES	512	0.6290	0.4886	-	1.2182	1.6811	-	4.51 min	3
Azerbaijan Sign Language	MobileNetV2	**NO**	512	0.9211	0.9021	**0.9026**	0.2917	0.4113	0.4181	20.48 min	7
Bengali Sign Language	MobileNetV2	YES	128	0.8997	0.8077	-	0.3246	0.6050	-	4.66 min	10
Bengali Sign Language	MobileNetV2	YES	256	0.9358	0.8125	-	0.2213	0.6199	-	3.81 min	9
Bengali Sign Language	MobileNetV2	YES	512	0.9671	0.8125	**0.8094**	0.1215	0.6614	0.6055	4.15 min	10
Bengali Sign Language	ConvNeXtBase	YES	128	0.8472	0.7535	**0.6630**	0.5032	0.8443	1.0555	46.71 min	10
Bengali Sign Language	ConvNeXtBase	YES	256	0.8701	0.6751	-	0.4190	1.0855	-	41.22 min	9
Bengali Sign Language	ConvNeXtBase	YES	512	0.9032	0.6643	-	0.3139	1.2082	-	42.6 min	10
Bengali Sign Language	MobileNetV2	**NO**	512	0.9639	0.9445	**0.9401**	0.1458	0.2635	0.3224	20.86 min	10
Hagrid 10	MobileNetV2	YES	128	0.7366	0.6565	-	0.7708	1.0357	-	20.13 min	10
Hagrid 10	MobileNetV2	YES	256	0.7930	0.6732	-	0.6085	1.0280	-	18.15 min	10
Hagrid 10	MobileNetV2	YES	512	0.8489	0.6746	**0.6589**	0.4428	1.1124	1.1529	17.53 min	10
Hagrid 10	ConvNeXtBase	YES	128	0.6705	0.6127	**0.6150**	0.9856	1.2157	1.2235	167.30 min	10
Hagrid 10	ConvNeXtBase	YES	256	0.7084	0.6110	-	0.8730	1.3036	-	185.51 min	10
Hagrid 10	ConvNeXtBase	YES	512	0.7351	0.6019	-	0.7915	1.3432	-	185.61 min	10
Hagrid 10	MobileNetV2	**NO**	512	0.9722	0.9676	**0.9691**	0.0947	0.1307	0.1129	60.34 min	10
Kenyan Sign Language	MobileNetV2	YES	128	0.8502	0.6262	**0.6565**	0.4207	1.1691	1.1114	2.776 min	10
Kenyan Sign Language	MobileNetV2	YES	256	0.8820	0.6195	-	0.3296	1.2517	-	2.25 min	9
Kenyan Sign Language	MobileNetV2	YES	512	0.8611	0.5915	-	0.3909	1.3393	-	1.63 min	6
Kenyan Sign Language	ConvNeXtBase	YES	128	0.7442	0.5661	**0.5856**	0.7358	1.2580	1.1951	16.01 min	7
Kenyan Sign Language	ConvNeXtBase	YES	256	0.7886	0.5541	-	0.6140	1.5261	-	15.93 min	7
Kenyan Sign Language	ConvNeXtBase	YES	512	0.8368	0.5327	-	0.4450	1.9528	-	22.85 min	10
Kenyan Sign Language	MobileNetV2	**NO**	128	0.8823	0.8785	**0.8871**	0.4211	0.4105	0.3716	22.85 min	10
27 Class Sign Language	MobileNetV2	YES	128	0.9123	0.8325	**0.8553**	0.2799	0.5012	0.4507	3.61 min	10
27 Class Sign Language	MobileNetV2	YES	256	0.8302	0.8026	-	0.5536	0.6067	-	1.25 min	3
27 Class Sign Language	MobileNetV2	YES	512	0.8748	0.8285	-	0.4107	0.5349	-	1.25 min	3
27 Class Sign Language	ConvNeXtBase	YES	128	0.8421	0.6803	**0.7469**	0.4682	1.0213	0.7637	38.06 min	10
27 Class Sign Language	ConvNeXtBase	YES	256	0.7459	0.6298	-	0.8193	1.1035	-	11.73 min	3
27 Class Sign Language	ConvNeXtBase	YES	512	0.7799	0.6741	-	0.6848	0.9495	-	11.73 min	3
27 Class Sign Language	MobileNetV2	NO	128	0.9834	0.9842	**0.9905**	0.0829	0.0921	0.0506	13.73 min	8

Table 3: Transfer Learning Results on six Sign Language Datasets

Figure 15 contains the **accuracies of the best models**. By best models, we define the models that performed the best on the test dataset, for each sign language dataset. This figure serves two purposes.

First, it verifies the quality of the training process and the lack of overfitting. This is evident by the fact that the validation and test accuracies are close to each other and have the same relevant distance from the train accuracy.

Second, it gives a visual representation of the benefits of training the whole model instead of the classification head. In each dataset, the *green* line, which corresponds to the entire model fine-tuning, is considerably higher than all the rest.

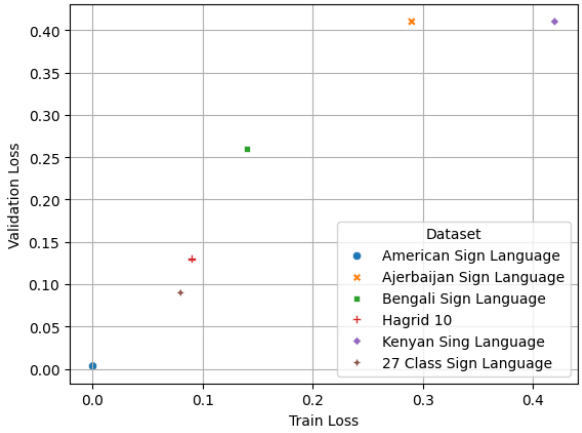


Figure 16: Train Loss vs Validation Loss - Optimal Transfer Learning Models

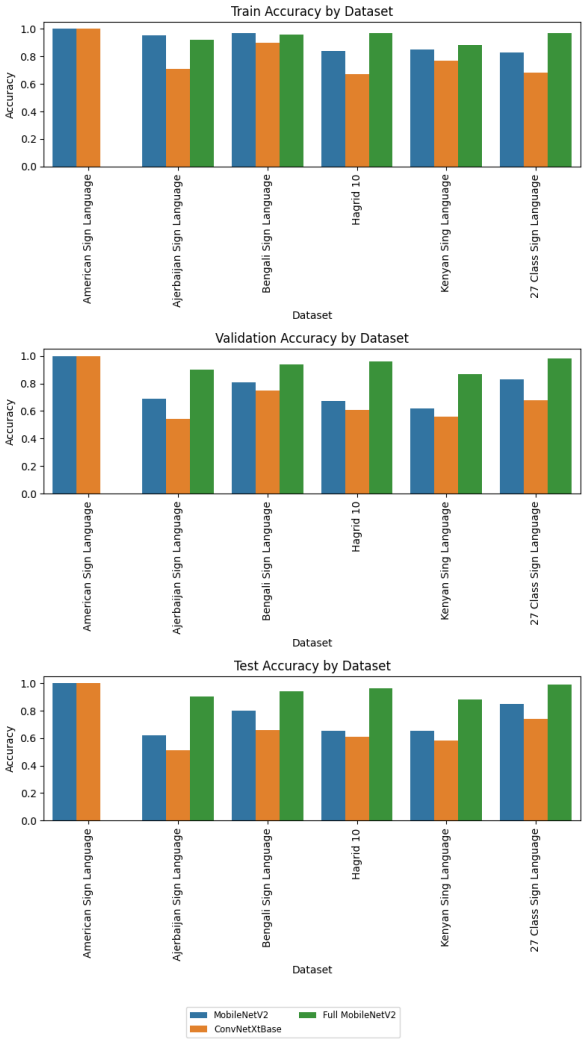


Figure 15: Accuracies per Dataset for the Best Models

Figure 16 provides another visualization of the train and validation loss achieved by the best model on each dataset. A successful training process should have similar train and validation losses. This proves that the model did not over-fit the train data and can generalize well into new unseen images. The majority of the points are close to the diagonal with the exception of the Ajerbaijan Sign Language

dataset. However, the difference between the train and the validation loss is about 0.1 which translates to a 2% difference between the train and validation accuracy.

5 Vision Transformers

Vision Transformers (ViTs)[10] are an adaptation of the transformer architecture for image processing. They treat images as sequences of patches, similar to words in a sentence, and leverage self-attention mechanisms to model dependencies between them. This enables ViTs to capture both local and global relationships in images more effectively than traditional Convolutional Neural Networks (CNNs). The vision transformer architecture consists of the following components, and is presented on 17:

- Patch Embedding: The image is divided into fixed-size patches, which are flattened and projected into a higher-dimensional space.
- Transformer Encoder:
- Multi-Head Self-Attention: Captures dependencies between image patches.
- Feed-forward Neural Network: Processes the attention outputs.
- Residual Connections and Layer Normalization: Stabilize training.
- Classification Head: A special [CLS] token is passed through the transformer layers and then projected to class logits.

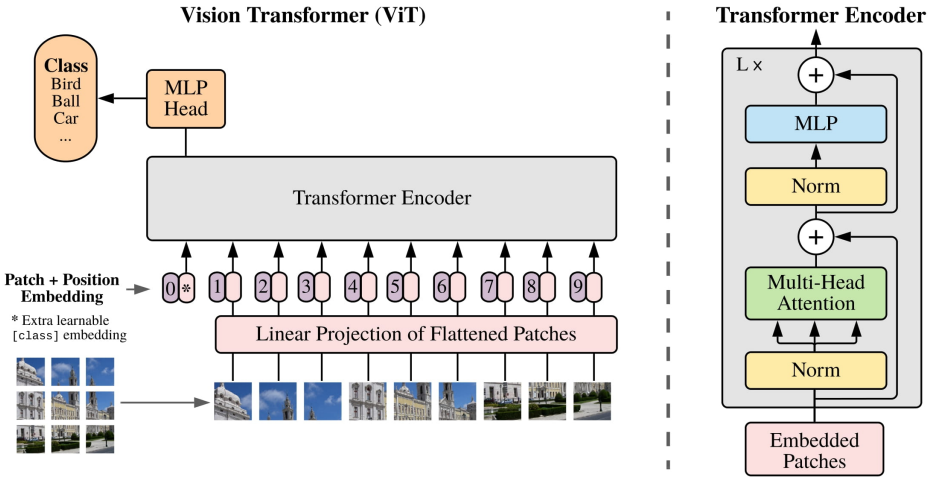


Figure 17: Visual Transformer Architecture

Typically, ViTs are pre-trained on large-scale datasets (e.g., ImageNet-21k, JFT-300M) and then fine-tuned on smaller downstream tasks. During fine-tuning, the pre-trained prediction head is removed and replaced with a zero-initialized feed-forward layer adapted to the number of target classes.

Unlike CNNs, which use fixed-size convolutional filters and hierarchical feature extraction, ViTs rely on self-attention to process spatial information. This architecture makes them highly effective for image classification and other computer vision tasks, particularly when trained on large datasets. However, because ViTs lack the built-in spatial biases of CNNs, they require significantly more data to generalize well. Without sufficient training data, ViTs are prone to over fitting.

5.1 Models

To experiment with multiple Vision Transformer architectures we used the timm (Torch Image Models) library. It is a repository of pre-trained deep learning models for image classification. To evaluate the Vision transformers performance in our sign language recognition task, several architectures were tested, as presented on the table below:

Model	Params (M)	Pre-training Dataset	Image size
ViT-Tiny	9.7	ImageNet-21k	224 x 224
ViT-Base	86.6	ImageNet-21k	224 x 224
EfficientViT-M5	12.5	ImageNet-1k	224 x 224
DeiT-III Small	22.1	ImageNet-22k	224 x 224

Table 4: ViT architectures for the experiments

5.1.1 ViT-Base/Tiny

ViT-Base is an image classification vision transformer model. It is trained on ImageNet-21k at resolution 224x224 and fine-tuned on ImageNet. This is the largest vision transformer model we experimented with, containing 86.6 M parameters. Hence, training the entire ViT-Base model is computationally expensive. Therefore, we trained only the classification head, keeping the transformer layers frozen, so that their weights are not updated during training. The classification head is a fully connected layer at the end of the transformer, and this is the part of the model that is updated with this method. In this approach the features learned during pre-training are retained and only the final layer is adapted. This allows for faster training and lower memory usage. This method tends to perform more efficiently on small datasets.

ViT-Tiny is the most lightweight variant of ViT-Base and the smaller model we tested with, because it contains 9.7M parameters. It is trained on ImageNet-21k at image resolution 224x224. For this model we conducted two experiments:

- Training only the classification head while keeping the transformer layers frozen.
- Fine-tuning the entire model by unfreezing all layers.

Fine-tuning the whole model is computationally more expensive but it contains substantially less parameters than the ViT-base and it is interesting to examine the trade-off of training the entire model with computational time, on a more scalable model. Unfreezing all transformer layers enables a better adaptation to the target dataset and improves the model's performance. Also it is more flexible for datasets where the pre-trained models might not extract relevant features to the given dataset. However fine-tuning the whole model might lead to overfitting compared to only training the classification head.

5.1.2 DeiT-III Small

DeiT-III[11] is a distilled data-efficient Image Transformer pre-trained on ImageNet-22k at resolution 224x224 and fine-tuned on ImageNet-1k at resolution 224x224. The small version used for the experiments contains 22.1M parameters, substantially smaller than our biggest model ViT-Base. Consequently for this model we unfreeze all the layers for training, to achieve better performance better performance, as training the whole model is scalable.

The distinguishing feature between classic ViTs and DeiTs is the use of a distillation token. DeiT architectures introduce this token, in addition to the CLS token, to improve training efficiency, by leveraging knowledge distillation from a teacher model. This enables to transfer knowledge from the teacher model to the DeiT model, the student. The distillation token is learned through back propagation, by interacting with the class CLS and patch tokens through the self-attention layers. The distillation procedure is presented in figure 18 The teacher model is usually a pre-trained CNN, for example ResNet or EfficientNet. The final output uses both the CLS token. This method reduces the amount of data required for effective training and bridges the gap between pure transformers and CNNs.

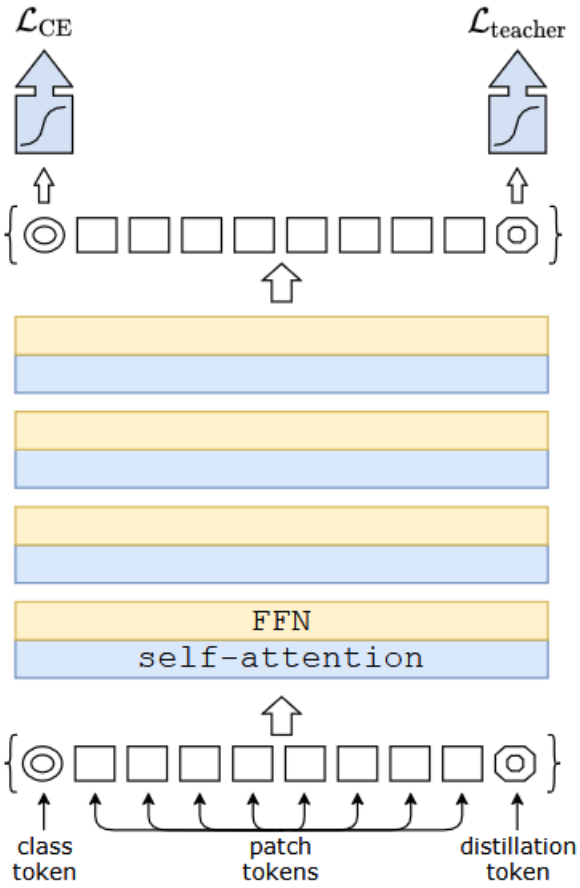


Figure 18: DeiT architecture and distillation procedure

5.1.3 EffientViT-M5

EfficientVit-M5[12] is a lightweight transformer model optimized for efficiency. It is trained on ImageNet-1k(1000 classes, 1.28M images) with a resolution of 224x224 and contains 12.4M parameters. We fine-tuned the entire model in this case, as EfficientViT's architecture is designed for lower computational overhead, making full fine-tuning feasible even on limited hardware.

EfficientViT addresses the high computational overhead of vision transformers by redesigning the transformer block. This includes using a single multi-head attention (MHSA) layer between lightweight feed-forward layers, reducing memory usage compared to the ViT's traditional multi-head self-attention, which processes the entire input with each attention head. EfficientViT also introduces cascading processing, as shown in the Sandwich Layout block, where the input features are split into smaller groups. Each attention head processes a unique subset of the input features, leading to more efficient computations by ensuring that each head focuses on different aspects of the input. The Cascaded Group Attention mechanism further optimizes this by merging the outputs from the different groups to maintain comprehensive feature representation while reducing redundancy. The architecture of EfficientViT is presented in figure 19.

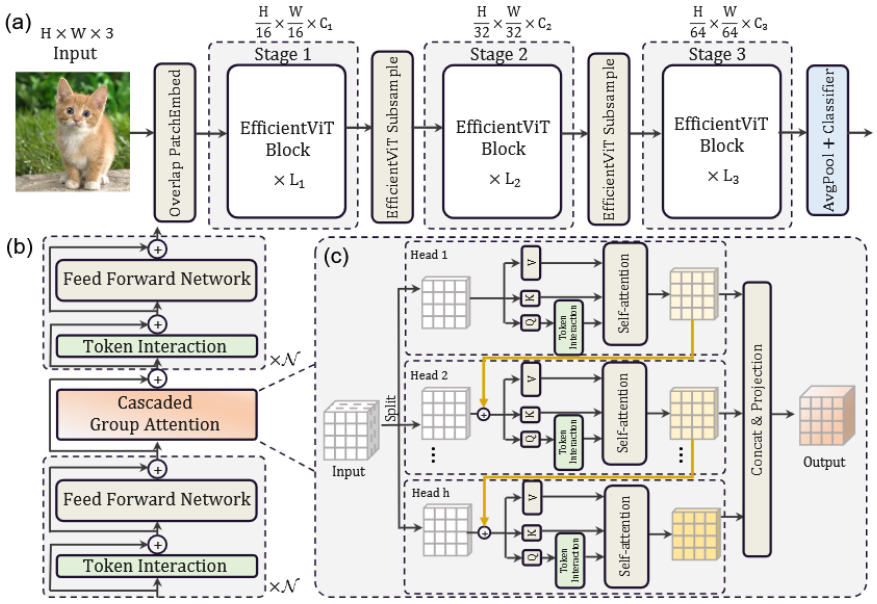


Figure 19: EfficientViT architecture

5.1.4 Training Process

The training of the Vision Transformer is performed using PyTorch, with the timm library for model initialization and pre-trained weights. The training is conducted in GPU-enabled environments: NVIDIA Geforce RTX 4GB (local developments) and two T4 GPU on kaggle. A custom Trainer class is developed to manage the training and evaluation process. The class includes a method that handles the training loop, performing forward passes, loss computation with Cross-Entropy Loss, and backpropagation. In addition, an evaluation method computes the validation loss and accuracy without updating the model's weights, operating in evaluation mode. The Trainer class also supports learning rate scheduling based on validation performance, for dynamic learning rate adjustment to improve convergence.

The dataset images are normalized based on the mean and standard deviation values of the dataset used, rather than using ImageNet statistics, ensuring better adaptation to the specific data distribution. The dataset is divided to 70% for the training set, 20% for the validation set to evaluate the generalization of the model , and 10% for the test set. In some datasets the test set is provided. In this case we split the dataset to 80% for training and 20% for validation. After training, performance is evaluated by calculating test accuracy, and a confusion matrix is generated to analyze misclassifications, providing a deeper understanding of the model's classification behavior. To further analyze the model's training behavior, accuracy and loss curves were plotted for both the training and validation sets across epochs. These plots provide insights into the model's convergence, the effectiveness of learning rate adjustments, and potential overfitting.

5.2 Exeperiments

5.2.1 Setup

A total of five ViT models were trained, each with different learning rate and weight decay configurations to identify the best-performing setup. When fine-tuning only the classification head, a higher learning rate (e.g 1e-3) and lower weight decay (1e-3) were used to allow faster adaptation. In contrast, when training the entire model, lower learning rates (e.g 1e-4) with higher weight decay (e.g 1e-2) were applied for stable learning. For smaller datasets, lower weight decay (e.g 1e-4) was preferred to maintain flexibility and prevent excessive regularization. On datasets with poor performance, where the validation accuracy remained stagnant across many epochs, early stopping was also applied. The optimizer used is AdamW and the loss function used is Cross-Entropy Loss, suitable for multi-class classification tasks. It is worth noting that after experimenting with learning rate schedulers to try

to improve the performance on models with lower validation accuracy, we observed that it did not contribute to the model's performance. Therefore, it was not applied.

5.2.2 Results

The results of all the experiments are presented in the table 5. By observing the results across all datasets, we deduce that EfficientViT-M5 224 was the best-performing model. This proves that if we train all the layers of a small vision transformer we can achieve satisfactory performance. This transformer achieves the highest test accuracy on the Bengali dataset at 92.57% and the 27 class sign language at 98.38%. It also achieves test accuracy above 87%, with the lowest occurring at the Azerbaijan dataset at 87.04%, across all datasets. This memory efficient vision transformer maintains scalable computation times, with the highest occurring on the American Sign Language Dataset at 38.8 minutes, and the lowest at the 27 Class Sign Language at 4.96 minutes. The second more per formant is the DeiT-III small model proving that vision transformer variants can perform better than the classic architecture. The second-best performer is the DeiT-III Small model, demonstrating that modern vision transformer variants can surpass classical architectures. DeiT-III achieves competitive test accuracies, such as 91.97% on the Azerbaijan dataset, 91.0% on the Bengali dataset, and 94.46% on HaGRID. Furthermore, it achieves a perfect score of 100% accuracy on the American Sign Language dataset. This model also exhibits relatively low training times, with the fastest training time recorded at 8.25 minutes for HaGRID.

Regarding the classic vision transformer architectures, fine-tuning the classifier head of ViT-Base offers satisfactory performance, with the lowest accuracy observed on the Hagrid dataset at 72.81%. However, even training the classifier head results in high computational time. The smallest elapsed time is detected on the Kenyan dataset at 24.83 minutes. However it is worth noting that this experiment is conducted on an NVIDIA GeForce RTX 3050 4GB, which affects the computational time. Also in several datasets it requires several epochs to achieve accuracy. For example, in the Bengali dataset to reach 75.26% test accuracy the classifier needs to be trained for 12 epochs. Furthermore the ViT-Tiny architecture when training only the classifier of the model we need a high learning rate and lower weight decay for the classifier to learm more efficiently. This is the least performant model, even when trained for several epochs. The highest accuracy reached is on the easiest dataset tested, the American SIgn language, where it converges in 2 epochs, achieving 100% accuracy. On the contrary on more complex datasets, such as either the Azerbaijan or the Kenyan dataset, the model achieves 41.99% and 48.24%, and did not further converge with more epochs. Hence, training a small ViT classifier is only effective on simpler and bigger datasets, to generalize better. However, when training the whole ViT-Tiny model, as it is the smallest model, we can improve its performance without increasing significantly the computational time. This model is capable of reaching test accuracy above 90%. The lowest accuracy achieved, occurs on the azerbaijan dataset at 92.7% at 16.26 minutes. It is worth highlighting though, that this is the best model for this dataset.

Dataset	Model	Head Only	LR	WD	Train Acc	Val Acc	Test Acc	Train Loss	Val Loss	Test Loss	Elapsed Time [min]	Epochs
American Sign Language	ViT Tiny Patch16-224	NO	2e-4	1e-2	0.9935	0.9995	1.0000	0.0266	0.0035	0.0009	36.76	1
American Sign Language	ViT Tiny Patch16-224	YES	5e-4	1e-2	1.0000	0.9999	1.0000	0.0028	0.0015	0.0007	121.5	2
American Sign Language	ViT Base Patch16-224	YES	5e-4	1e-2	1.0000	1.0000	1.0000	0.0027	0.0014	0.0005	81.33	2
American Sign Language	EfficientViT-M5 224	NO	2e-4	1e-2	0.9991	1.0000	1.0000	0.0047	0.0006	0.0005	38.8	2
American Sign Language	DeiT-III Small Patch16-224	NO	2e-4	1e-2	0.9940	1.0000	1.0000	0.0234	0.0001	0.0000	34.28	1
Azerbaijan Sign Language	ViT Tiny Patch16-224	NO	1e-4	5e-4	0.9984	0.9100	0.9270	0.0128	0.3181	0.2917	16.36	5
Azerbaijan Sign Language	YES	1e-3	1e-3	0.5089	0.4189	0.4199	1.5701	1.9594	1.8162	64.25	20	
Azerbaijan Sign Language	ViT Base Patch16-224	YES	2e-3	1e-3	0.8775	0.6313	0.5903	0.4334	1.2214	1.3623	51.6	11
Azerbaijan Sign Language	EfficientViT-M5 224	NO	1e-4	5e-4	0.9876	0.8675	0.8704	0.0938	0.4586	0.4499	31.83	10
Azerbaijan Sign Language	DeiT-III Small Patch16-224	NO	1e-4	1e-3	0.9439	0.9306	0.9197	0.1762	0.2617	0.2644	9.23	2
Bengali Sign Language	ViT Tiny Patch16-224	NO	2e-4	1e-4	0.9773	0.9218	0.9092	0.0766	0.2806	0.3157	6	4
Bengali Sign Language	ViT Tiny Patch16-224	YES	1e-3	1e-2	0.7226	0.6240	0.5711	0.9197	1.3257	1.4323	12	15
Bengali Sign Language	ViT Base Patch16-224	YES	5e-4	1e-2	0.8851	0.7605	0.7526	0.5230	0.8332	0.8345	28.65	12
Bengali Sign Language	EfficientViT-M5 224	NO	1e-4	5e-3	0.9947	0.9146	0.9257	0.0770	0.3089	0.2973	8.36	10
Bengali Sign Language	DeiT-III Small Patch16-224	NO	2e-4	1e-2	0.9439	0.9306	0.9105	0.1762	0.2617	0.2882	9.23	2
HaGRID Classification 512p (10%)	ViT Tiny Patch16-224	NO	2e-4	1e-2	0.9783	0.9734	0.9761	0.0658	0.0827	0.0744	423	3
HaGRID Classification 512p (10%)	ViT Tiny Patch16-224	YES	2e-3	1e-3	0.4143	0.4205	0.4199	1.8414	1.8167	1.8162	230	10
HaGRID Classification 512p (10%)	ViT Base Patch16-224	YES	5e-4	1e-3	0.7314	0.7259	0.7281	0.8051	0.8193	0.8297	2228.47	10
HaGRID Classification 512p (10%)	EfficientViT-M5 224	NO	2e-4	1e-2	0.9752	0.9536	0.9533	0.0836	0.1593	0.1492	21.18	3
HaGRID Classification 512p (10%)	DeiT-III Small Patch16-224	NO	2e-4	1e-2	0.9637	0.9442	0.9446	0.1792	0.2967	0.2878	8.25	1
Kenyan Sign Language	ViT Tiny Patch16-224	NO	1e-5	5e-2	0.9907	0.8680	0.8608	0.0805	0.4643	0.4550	10.35	10
Kenyan Sign Language	ViT Tiny Patch16-224	YES	2e-3	1e-4	0.5799	0.4820	0.4824	1.1888	1.5276	1.5014	8.17	10
Kenyan Sign Language	ViT Base Patch16-224	YES	2e-3	1e-3	0.9255	0.7120	0.9003	0.2813	0.9815	0.7240	24.83	10
Kenyan Sign Language	EfficientViT-M5 224	NO	1e-4	5e-3	0.9960	0.8750	0.8840	0.0487	0.4406	0.4365	11.4	13
Kenyan Sign Language	DeiT-III Small Patch16-224	NO	2e-4	1e-2	0.9322	0.9150	0.9040	0.2251	0.2959	0.3189	6.35	2
27 Class Sign Language	ViT Tiny Patch16-224	NO	2e-4	1e-2	0.9753	0.9671	0.9601	0.0852	0.1168	0.1335	3.4	2
27 Class Sign Language	ViT Tiny Patch16-224	YES	1e-3	1e-3	0.6246	0.5656	0.5498	1.1470	1.3530	1.3714	22.15	23
27 Class Sign Language	ViT Base Patch16-224	YES	1e-3	1e-3	0.8949	0.7932	0.7904	0.3606	0.6186	0.6344	71.34	15
27 Class Sign Language	EfficientViT-M5 224	NO	2e-4	1e-2	0.9881	0.9800	0.9838	0.0587	0.0788	0.0783	4.96	4
27 Class Sign Language	DeiT-III Small Patch16-224	NO	2e-4	1e-2	0.9786	0.9787	0.9754	0.0755	0.0820	0.0884	6.87	2

Table 5: ViT models results across all datasets

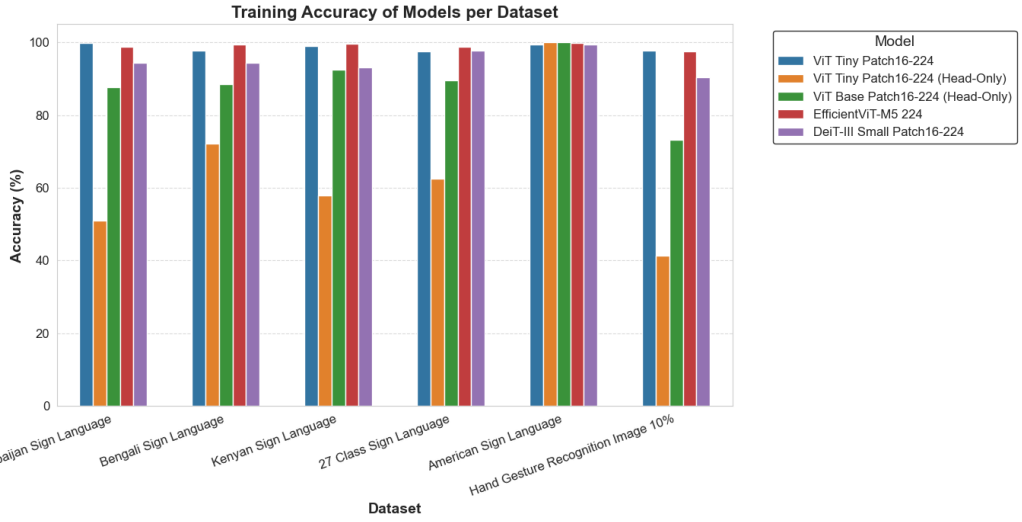


Figure 20: Training accuracy across datasets for each model

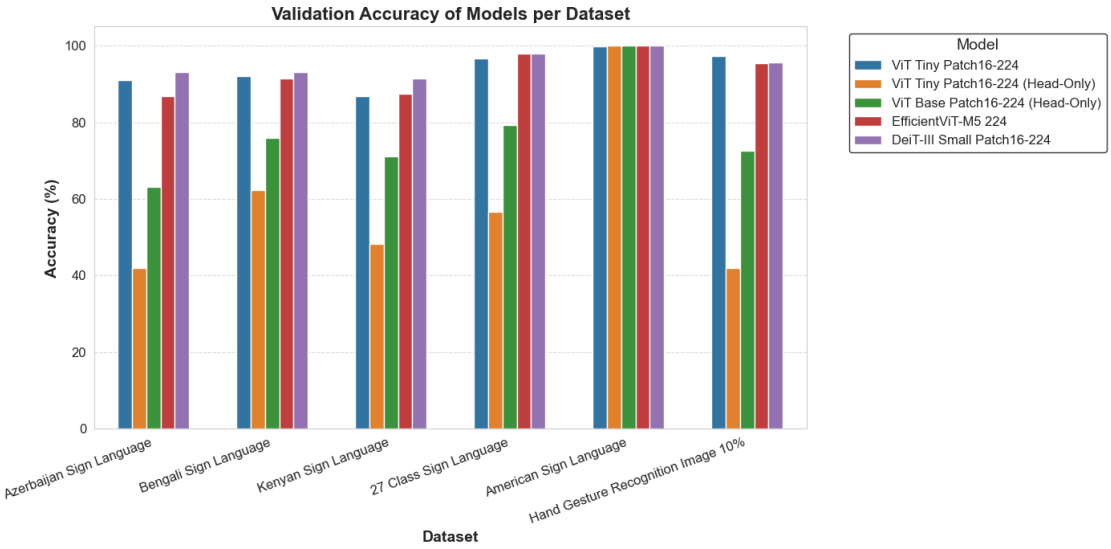


Figure 21: Validation accuracy across datasets for each model

Figure 22: Test accuracy across datasets for each model

On the Figures 20-22 an overview of the train, validation and test accuracies is present for each model across all datasets. From the results, we can observe that the American Sign Language (ASL) dataset shows exceptional performance across all models, with accuracies nearing 100% in both training, validation, and test sets, indicating this is a simple dataset. Similarly, the 27 Class Sign Language dataset also demonstrates strong generalization with minimal performance differences between training and test accuracies. This indicates good generalization, but it is worth noting this is also a simpler dataset. However, certain datasets show signs of overfitting, particularly Kenyan Sign Language and Azerbaijan Sign Language. For these datasets, training accuracies are much higher than test accuracies, especially for models like ViT-Base Patch16-224 and ViT-Tiny Patch16-224 with only the classification head trained. The Bengali sign language dataset also overfits with ViT-Base Patch16-224. This indicates that models where only the classifier head is trained do not generalize well to unseen data, especially for more complex and small datasets. This proves the disadvantage of visual transformers as they are data hungry. Furthermore, on some models both validation and training accuracy are low indicating under-fitting. This occurs on the Hand Gesture Recognition image with ViT-Tiny Patch16-224 with only the classification head trained, as well as on the 27 class sign language dataset,

and Kenyan dataset with the same model. This suggests that training only the classifier head on this model does not capture the underlying patterns in the data.

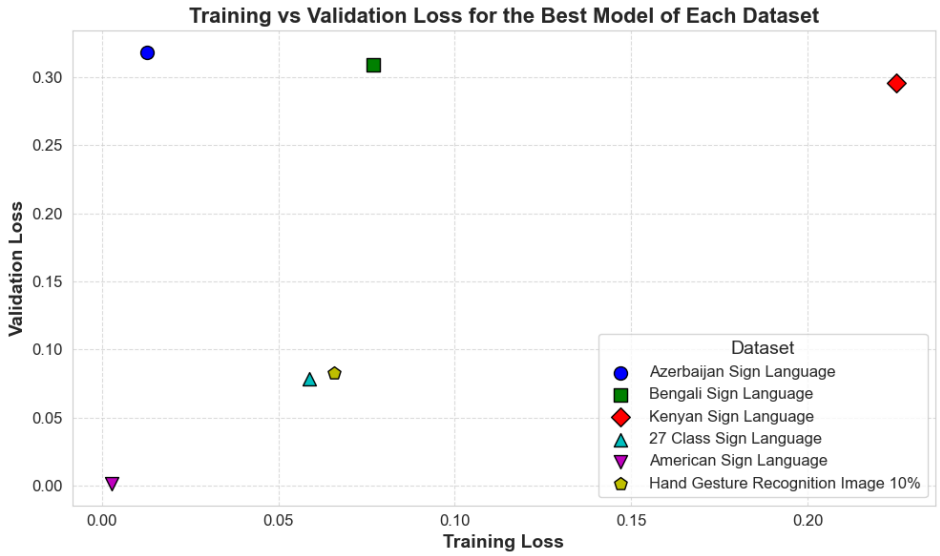


Figure 23: Accuracy and Validation loss for best ViT model of each dataset

The plot 23 comparing training and validation losses for the best model of each dataset reveals key insights into model performance. Datasets like Azerbaijan Sign Language and Bengali Sign Language show a noticeable gap between low training loss and higher validation loss, suggesting overfitting. Conversely, the 27 Class Sign Language dataset, the American Sign Language and the Hand Gesture Recognition Image dataset exhibit low and similar training and validation losses, indicating good generalization. To elaborate, the American Sign Language dataset stands out with very low training and validation losses. Furthermore, on the Kenyan dataset both the accuracy and training loss is the highest, indicating possible underfitting. To conclude, most datasets do not feature high discrepancies between the validation and training loss, indicating good generalization.

6 Best Models Comparison

Table 6 presents the top-performing models that achieved the best performance metrics for each dataset among the various models tested.

Dataset	Train Acc [%]	Val Acc [%]	Test Acc [%]	Train Loss	Val Loss	Head Only	Model
Custom Neural and Convolutional Networks							
American Sign Language	100	99.97	99.98	0.000898236	0	-	NN
Bengali Sign Language	75.86	36.64	36.29	1.028	2.37	-	NN
Keynan Sign Language	100	32.9	34.66	0.002896843	3.1	-	CNN
Azerbaijan Sign Language	98.42	29.74	27.23	0.433727324	3.46	-	Enhanced CNN
27 Class Sign Language	97.38	67.58	65.89	0.455358595	1.29	-	Enhanced CNN
Hand Gesture Recognition Image	78.09	21.67	21.81	0.942707419	4.71	-	NN
Transfer Learning on Pretrained Models							
American Sign Language	99.99	99.99	99.99	0.00024	0.00016	YES	MobileNetV2
Bengali Sign Language	96.39	94.45	94.01	0.1458	0.2635	NO	MobileNetV2
Keynan Sign Language	88.23	87.85	88.71	0.4211	0.4105	NO	MobileNetV2
Azerbaijan Sign Language	92.11	90.21	90.26	0.2917	0.4133	NO	MobileNetV2
27 Class Sign Language	98.34	98.42	99.05	0.0829	0.0929	NO	MobileNetV2
Hand Gesture Recognition Image	97.22	96.76	96.91	0.0947	0.1307	NO	MobileNetV2
Fine-tune Vision Transformers							
American Sign Language	1.00	1.00	1.00	0.00270	0.00140	YES	ViT Base Patch16-224
Bengali Sign Language	99.47	91.46	92.57	0.07700	0.30890	NO	EfficientViT-M5 224
Keynan Sign Language	93.22	91.50	90.40	0.22510	0.29590	NO	DeiT-III Small Patch16-224
Azerbaijan Sign Language	99.84	91.00	92.70	0.01280	0.31810	NO	ViT Tiny Patch16-224
27 Class Sign Language	98.81	98.00	98.38	0.05870	0.07880	NO	EfficientViT-M5 224
Hand Gesture Recognition Image	97.83	97.34	97.61	0.0658	0.0827	NO	ViT Tiny Patch16-224

Table 6: Best Models for the Different Datasets per Training Type

The results reveal distinct performance variations across the training methods for each dataset. For the American Sign Language dataset, all approaches achieved near-perfect accuracy, with fine-tuned Vision Transformers (ViT) reaching 100% across all splits, while MobileNetV2 and a custom NN followed closely behind. It is worth highlighting that for this dataset, in the transfer learning and the vision transformer approach, only training the classification head was sufficient to achieve such performance. In contrast, for the Bengali Sign Language dataset, the neural network struggled significantly, achieving only 36.29% test accuracy, whereas fine-tuned EfficientViT-M5 224 improved performance to 92.57% and transfer learning with MobileNetV2 further enhanced it to 94.01%. A similar trend is observed in the Kenyan Sign Language dataset, where the custom CNN performed poorly with 34.66% test accuracy, while MobileNetV2 improved results to 88.71% and DeiT-III Small Patch16-224 reached 90.40%, highlighting the advantages of leveraging pre-trained features. The Azerbaijan Sign Language dataset saw the enhanced CNN achieve only 27.23% test accuracy, but transfer learning with MobileNetV2 (90.26%) and fine-tuning with ViT Tiny Patch16-224 (92.70%) delivered significant improvements. For the 27 Class Sign Language dataset, the enhanced CNN achieved moderate results (65.89%), while MobileNetV2 (99.05%) and EfficientViT-M5 224 (98.38%) demonstrated near-perfect generalization. Finally, the Hand Gesture Recognition Image dataset saw the custom NN achieve only 21.81% test accuracy, but MobileNetV2 (96.91%) and ViT Tiny Patch16-224 (97.61%) indicated substantial gains. Overall, traditional neural networks struggled to generalize effectively, while transfer learning and fine-tuned Vision Transformers consistently delivered superior results across all datasets, as indicated also in Figure 24.

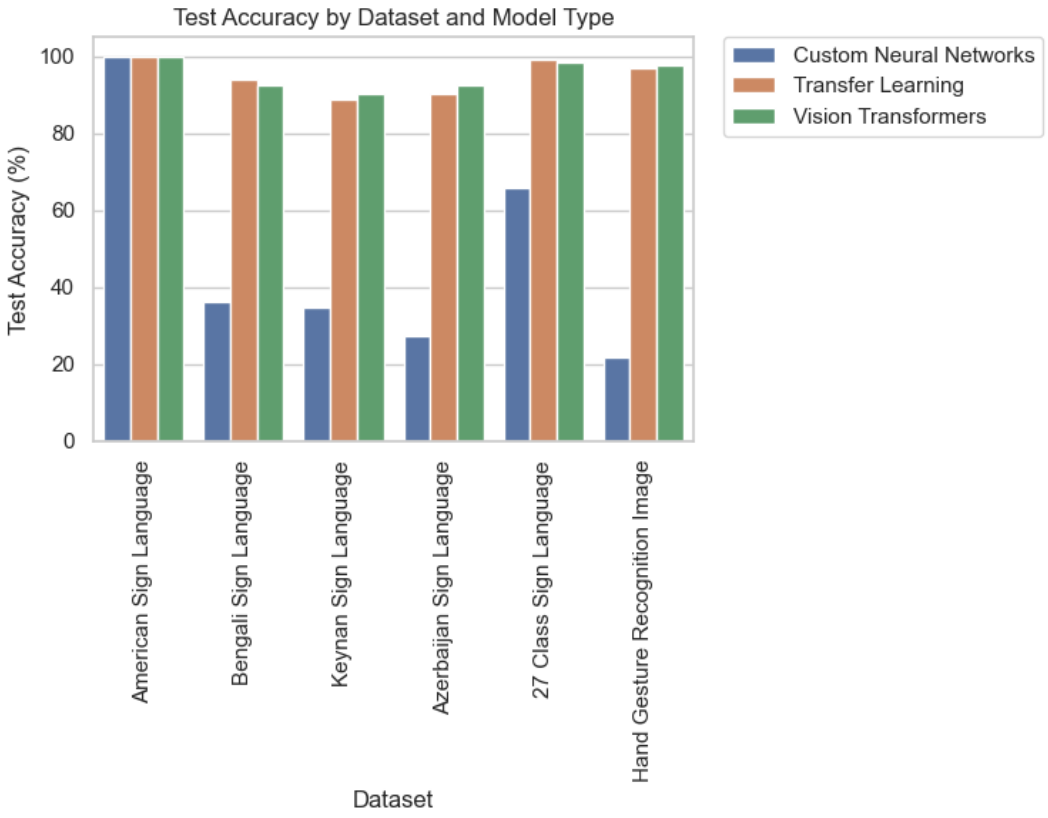


Figure 24: Test Accuracy by Dataset and Model Type

Concluding, designing and training custom neural networks and convolutional models from scratch is a complex and resource-intensive process. It requires careful architecture design and substantial amounts of labeled data to achieve good generalization. As seen in our results, these models often suffer from severe overfitting, particularly when data is limited, leading to poor validation and test accuracy despite high training accuracy. In contrast, transfer learning and fine-tuning pre-trained models require significantly less effort while achieving superior results. Pre-trained models like MobileNetV2 and Vision Transformers (ViT) leverage vast amounts of data and extensive prior training, allowing them

to extract meaningful features even from smaller datasets. This reduces the need for extensive training and computational resources while maintaining high accuracy. The results demonstrate that even without large-scale datasets, transfer learning and fine-tuning provide a more effective and efficient approach to sign language recognition, outperforming custom models with minimal additional training effort.

7 Conclusion and Future Work

In this study, we explored the performance of various models, including Custom Neural Networks, Transfer Learning models and Vision Transformers, across multiple sign language datasets. Our analysis highlighted the strengths and limitations of each model type in terms of test accuracy, training efficiency and generalization capabilities. While some models achieved near-perfect accuracy on specific datasets, others struggled with generalization, particularly on datasets with higher variability. These findings underscore the importance of selecting the right model architecture and training strategy based on the dataset characteristics and application requirements.

Our Future Work includes:

1. Data Augmentation: To enhance model generalization and robustness, we plan to implement advanced data augmentation techniques. By augmenting the training data with transformations such as rotation, scaling, noise addition and synthetic data generation, we aim to help models learn more invariant features. This will improve their performance on unseen data, particularly in scenarios where sign language gestures vary in orientation, scale, or environmental conditions.

2. Utilize Video Datasets to Extract Images and Perform Training: Currently, our work is limited to static image datasets. However, sign language is inherently dynamic and relies heavily on temporal information. In the future, we aim to leverage video datasets by extracting frames and incorporating temporal modeling techniques. This approach will enable us to capture the sequential nature of sign language gestures, leading to more accurate and context-aware recognition systems.

3. Real-Time Processing: Developing models capable of real-time sign language recognition is a critical next step. Real-time processing requires optimizing model architectures for speed and efficiency without compromising accuracy. We plan to explore lightweight models and deploy them on edge devices or mobile platforms. Additionally, techniques like model quantization and pruning will be investigated to reduce computational overhead and enable seamless real-time inference.

References

[1] Kapil Londhe. American sign language, 2021.

[2] Abdul Muntakim Rafi, Nowshin Nawal, Nur Sultan Nazar Bayev, Lusain Nima, Celia Shahnaz, and Shaikh Anowarul Fattah. Image-based bengali sign language alphabet recognition for deaf and dumb community. In *2019 IEEE global humanitarian technology conference (GHTC)*, pages 1–7. IEEE, 2019.

[3] Gaurav Dutta. Kslc kenyan sign language classification challenge, 2023. Dataset published on Kaggle.

[4] Aykhan Nazimzada. Azsl dataset, 2022. Dataset published on Kaggle.

[5] Alexander Kapitanov, Karina Kvanchiani, Alexander Nagaev, Roman Kraynov, and Andrei Makhliarchuk. Hagrid–hand gesture recognition image dataset. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4572–4581, 2024.

[6] Arda Mavi and Zeynep Dikle. A new 27 class sign language dataset collected from 173 individuals. *arXiv preprint arXiv:2203.03859*, 2022.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[8] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

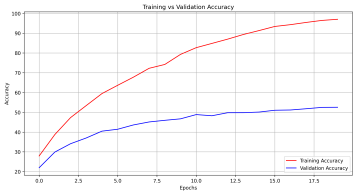
[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[11] Hugo Touvron, Matthieu Cord, and Herve Jegou. Deit iii: Revenge of the vit. *arXiv preprint arXiv:2204.07118*, 2022.

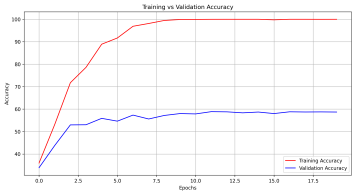
[12] Xinyu Liu, Houwen Peng, Ningxin Zheng, Yuqing Yang, Han Hu, and Yixuan Yuan. Efficientvit: Memory efficient vision transformer with cascaded group attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Appendix

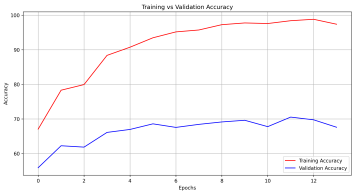
A NN and CNN - Training Plots



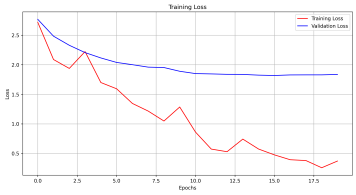
(a) Neural Network Accuracy



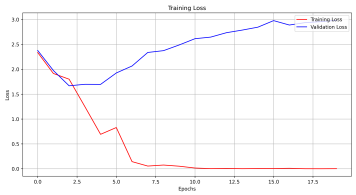
(b) Simple Convolutional Neural Network Accuracy



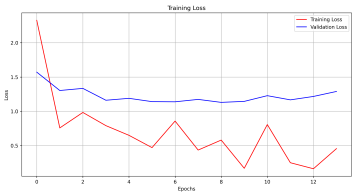
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss



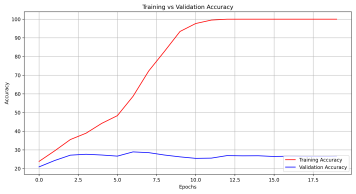
(e) Simple Convolutional Neural Network Loss



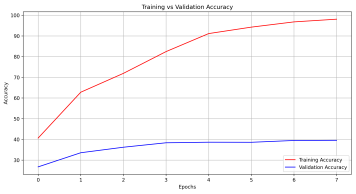
(f) Enhanced Convolutional Neural Network Loss

Figure 25: American Sign Language - Accuracy and Loss vs Epochs

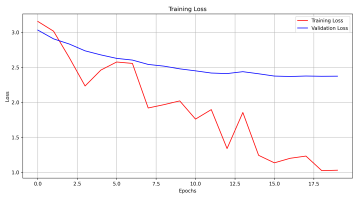
(a) Neural Network Accuracy



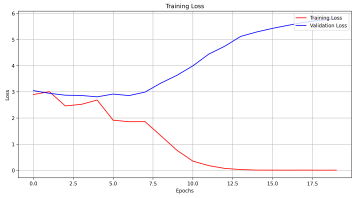
(b) Simple Convolutional Neural Network Accuracy



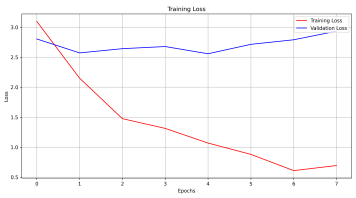
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss

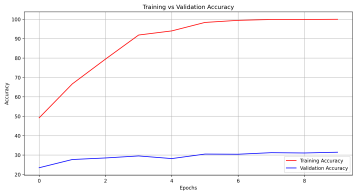


(e) Simple Convolutional Neural Network Loss

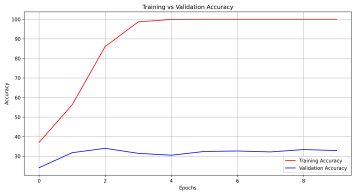


(f) Enhanced Convolutional Neural Network Loss

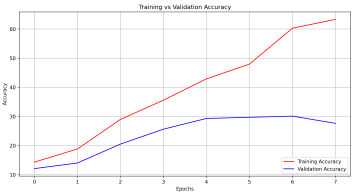
Figure 26: Bengali Sign Language - Accuracy and Loss vs Epochs



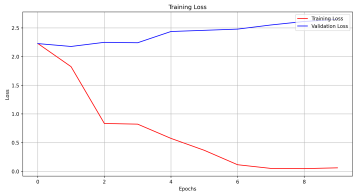
(a) Neural Network Accuracy



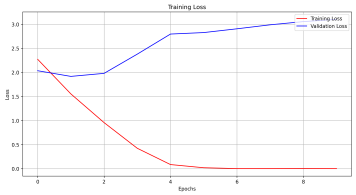
(b) Simple Convolutional Neural Network Accuracy



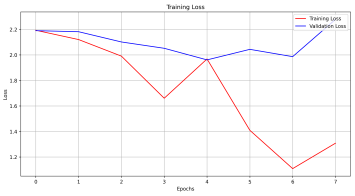
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss

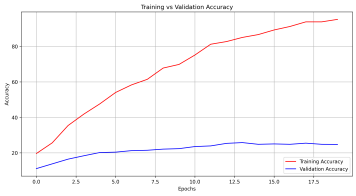


(e) Simple Convolutional Neural Network Loss

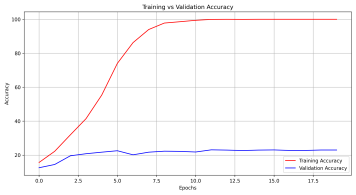


(f) Enhanced Convolutional Neural Network Loss

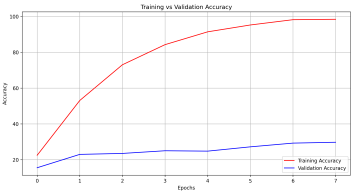
Figure 27: Kenyan Sign Language - Accuracy and Loss vs Epochs



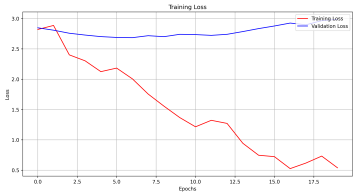
(a) Neural Network Accuracy



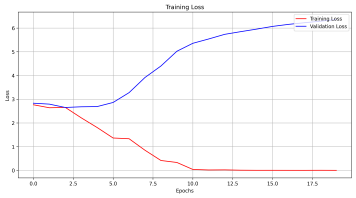
(b) Simple Convolutional Neural Network Accuracy



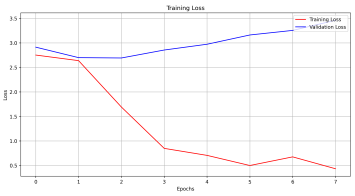
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss

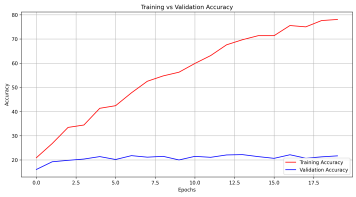


(e) Simple Convolutional Neural Network Loss

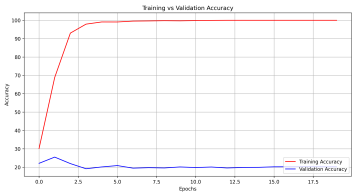


(f) Enhanced Convolutional Neural Network Loss

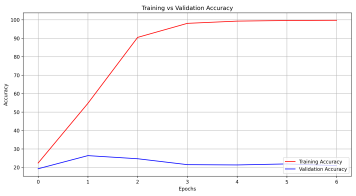
Figure 28: Azerbaijan Sign Language - Accuracy and Loss vs Epochs



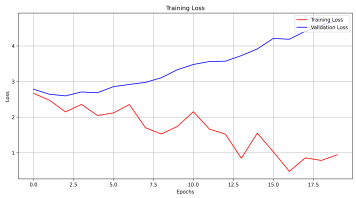
(a) Neural Network Accuracy



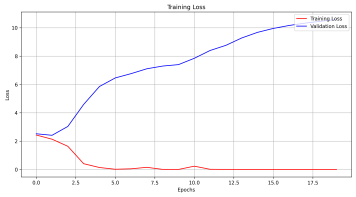
(b) Simple Convolutional Neural Network Accuracy



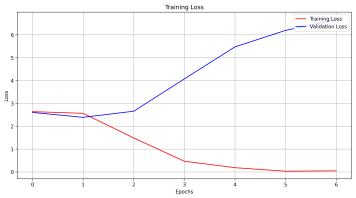
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss

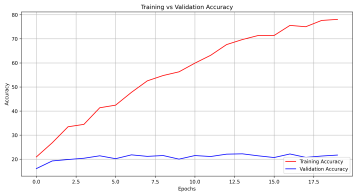


(e) Simple Convolutional Neural Network Loss

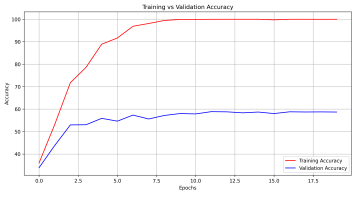


(f) Enhanced Convolutional Neural Network Loss

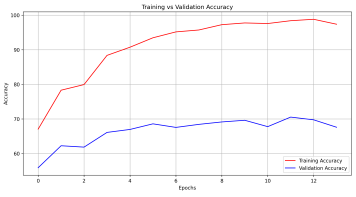
Figure 29: Hand Gesture Recognition Image - Accuracy and Loss vs Epochs



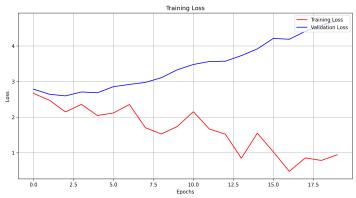
(a) Neural Network Accuracy



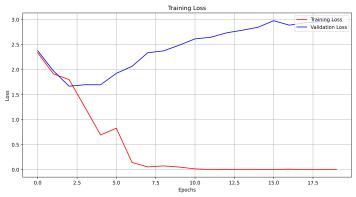
(b) Simple Convolutional Neural Network Accuracy



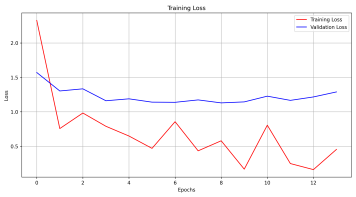
(c) Enhanced Convolutional Neural Network Accuracy



(d) Neural Network Loss

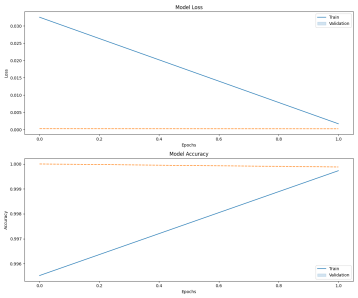


(e) Simple Convolutional Neural Network Loss

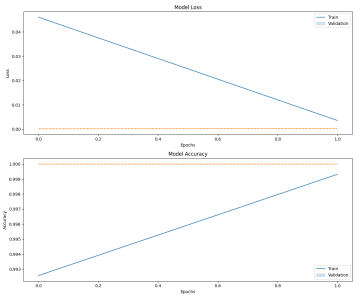


(f) Enhanced Convolutional Neural Network Loss

Figure 30: 27 Class Sign Language - Accuracy and Loss vs Epochs



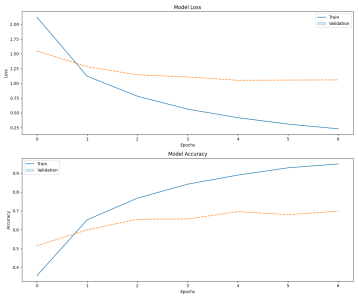
(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch



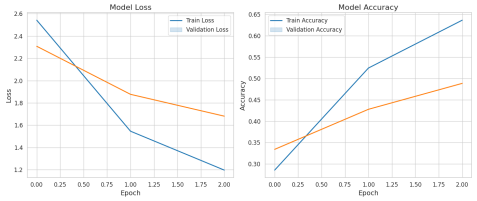
(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 31: American Sign Language - Accuracy and Loss vs Epochs

B Transfer Learning - Training Plots

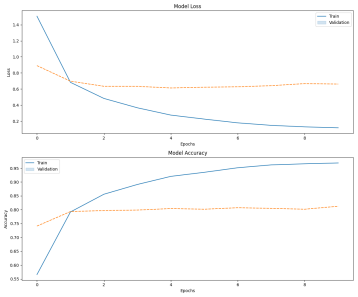


(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch

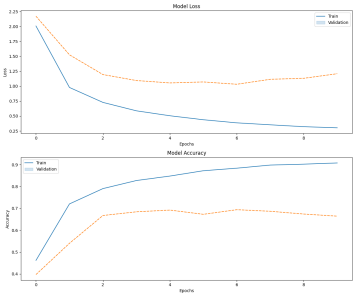


(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 32: Azerbaijan Sign Language - Accuracy and Loss vs Epochs

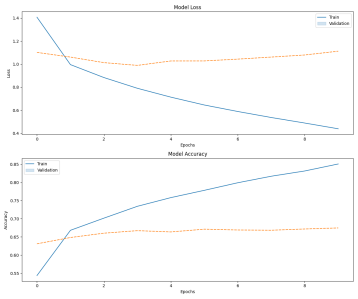


(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch

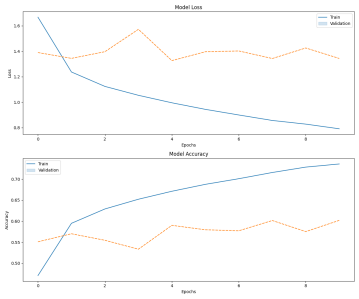


(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 33: Bengali Sign Language - Accuracy and Loss vs Epochs

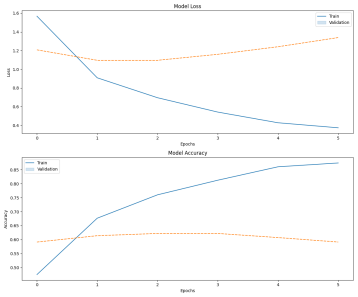


(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch

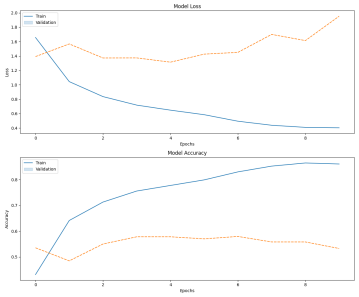


(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 34: HAnd Gesture Recognition Image Dataset - Accuracy and Loss vs Epochs

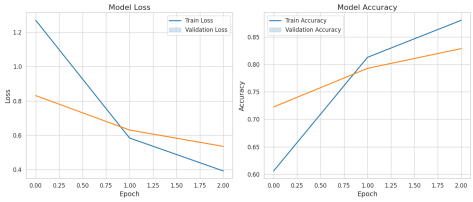


(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch

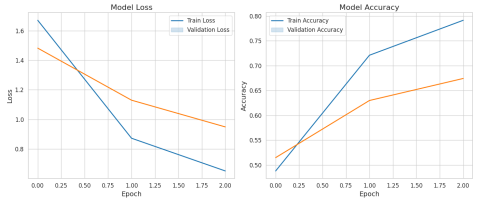


(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 35: Kenyan Sign Language Dataset - Accuracy and Loss vs Epochs



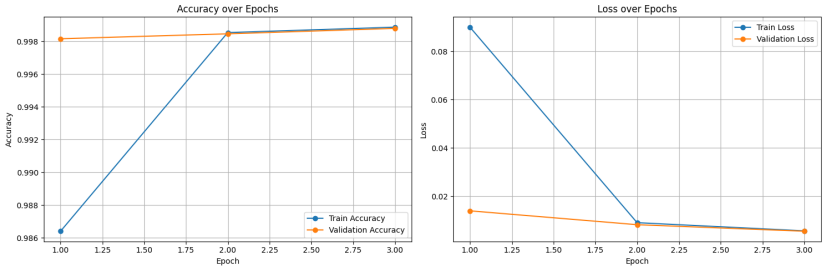
(a) MobileNetV2 Train - Validation Accuracy && Train - Validation Loss per epoch



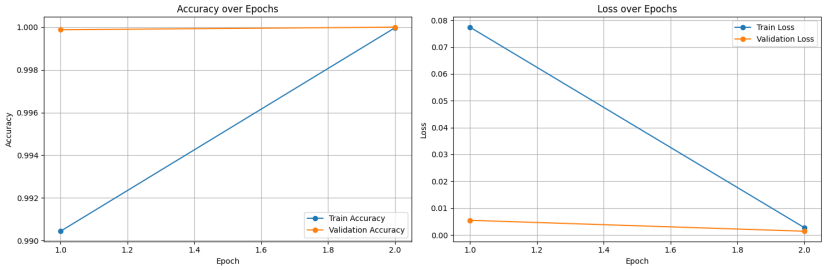
(b) ConvNetXtBase Train - Validation Accuracy && Train - Validation Loss per epoch

Figure 36: 27 Class Sign Language Dataset - Accuracy and Loss vs Epochs

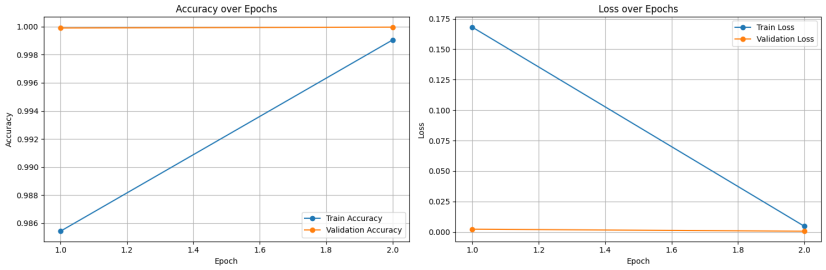
C Vision Transformers - Training Plots



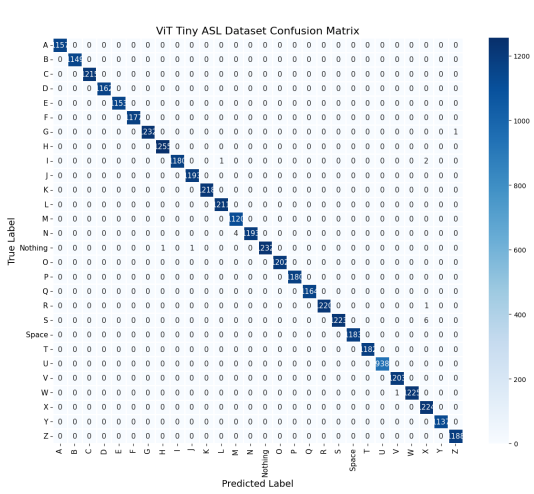
(a) ViT Tiny Patch16-224 (Head Only) ASL Train - Validation Accuracy && Train - Validation Loss per epoch



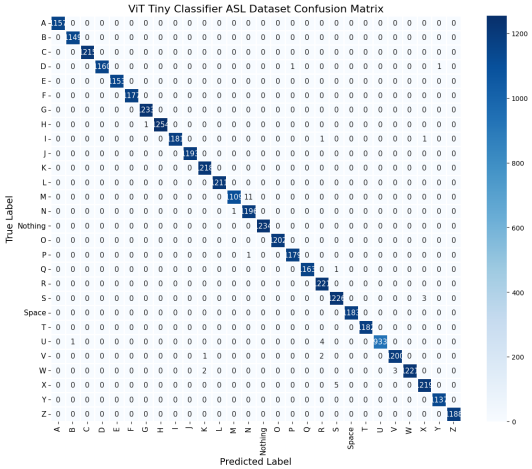
(b) ViT Base Patch16-224 (Head Only) ASL Train - Validation Accuracy && Train - Validation Loss per epoch



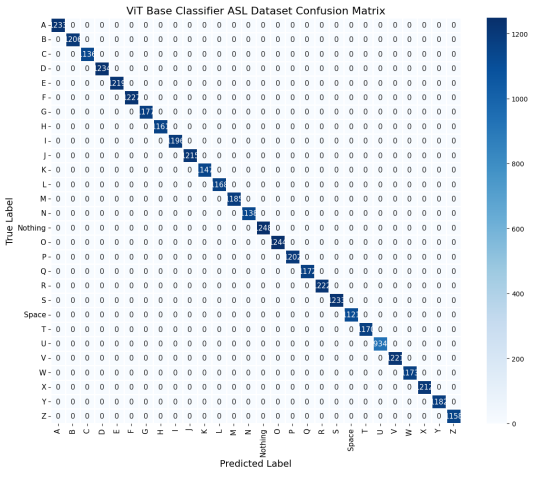
(c) EfficientViT-M5 Tiny ASL Train - Validation Accuracy && Train - Validation Loss per epoch



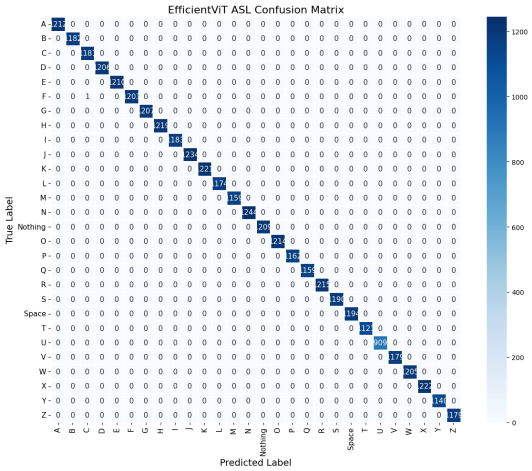
(a) ViT Tiny Patch16-224 ASL Confusion Matrix



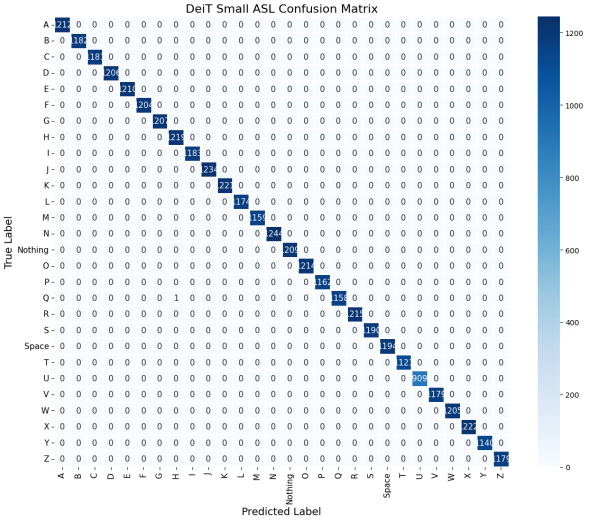
(b) ViT Tiny Patch16-224 (Head Only) ASL Confusion Matrix



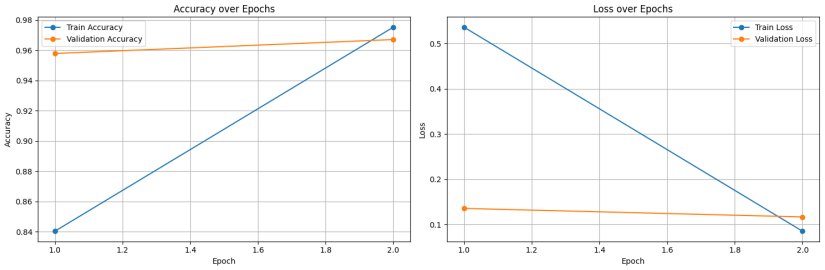
(a) ViT Base Patch16-224 (Head Only) Patch16-224 ASL Confusion Matrix



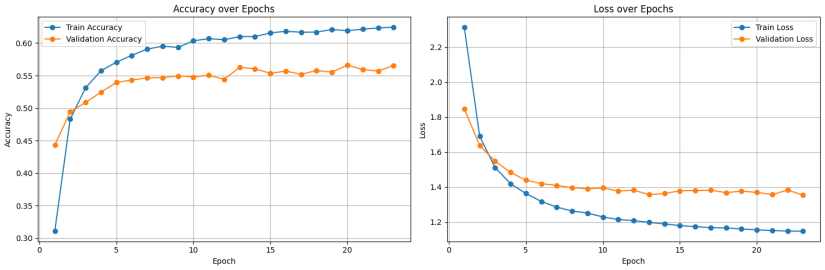
(b) EfficientViT-M5 tiny ASL Confusion Matrix



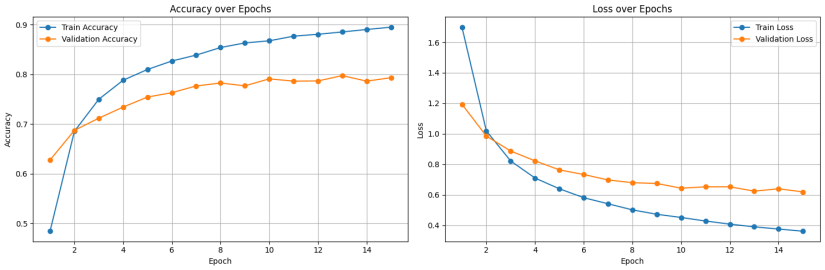
(c) Deit-III Small ASL Confusion Matrix



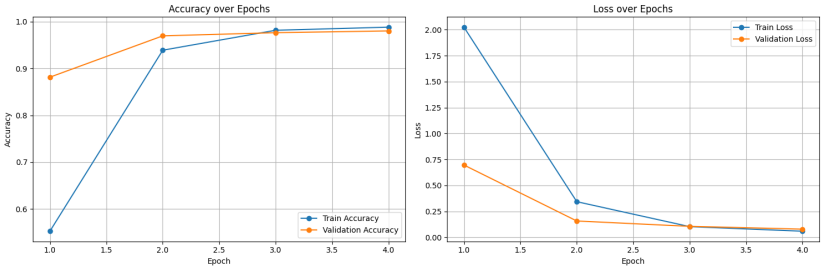
(a) ViT Tiny Patch16-224 27 class ASL Train - Validation Accuracy && Train - Validation Loss per epoch



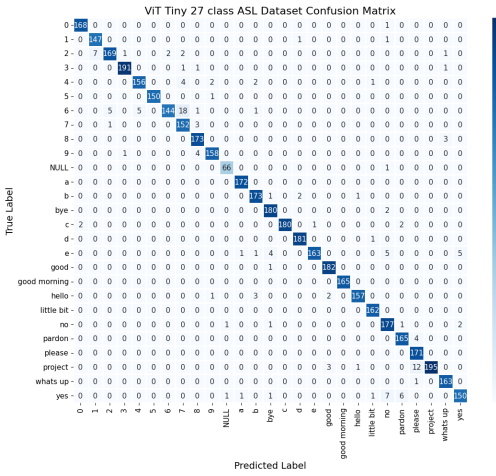
(b) ViT Tiny Patch16-224 (Head Only) 27 class ASL Train - Validation Accuracy && Train - Validation Loss per epoch



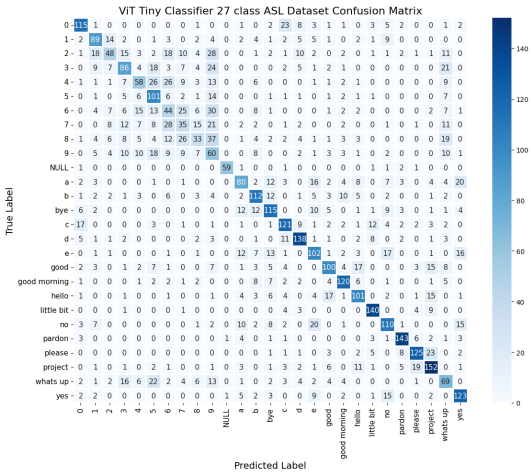
(c) ViT Base Patch16-224 (Head Only) 27 class ASL Train - Validation Accuracy && Train - Validation Loss per epoch



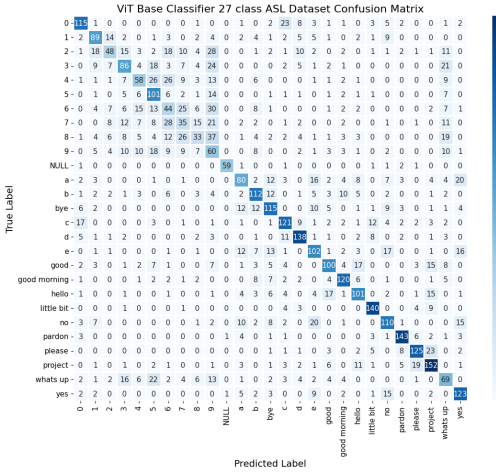
(d) EfficientViT-M5 Tiny 27 class ASL Train - Validation Accuracy && Train - Validation Loss per epoch



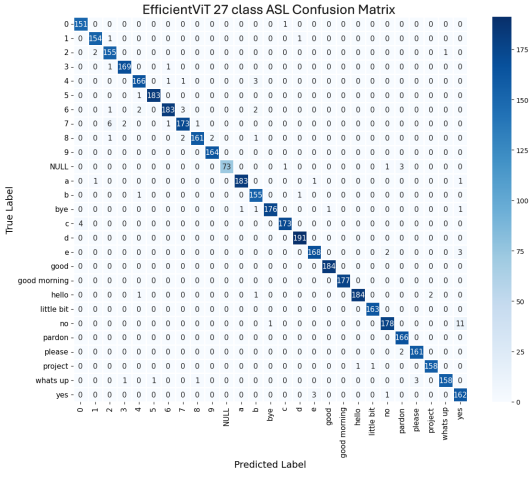
(a) ViT Tiny Patch16-224 27 class ASL Confusion Matrix



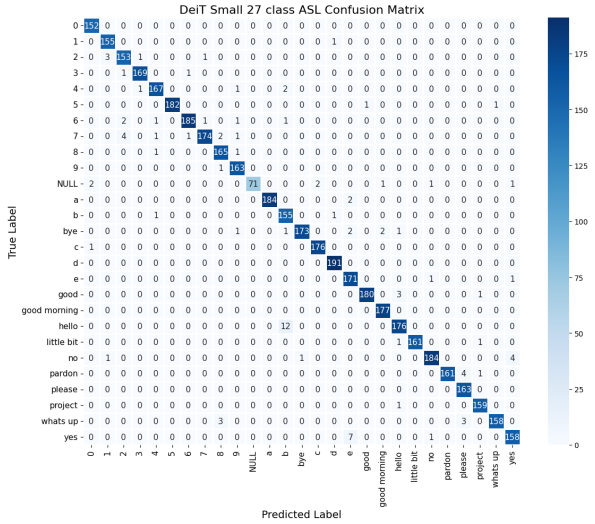
(b) ViT Tiny Patch16-224 (Head Only) 27 class ASL Confusion Matrix



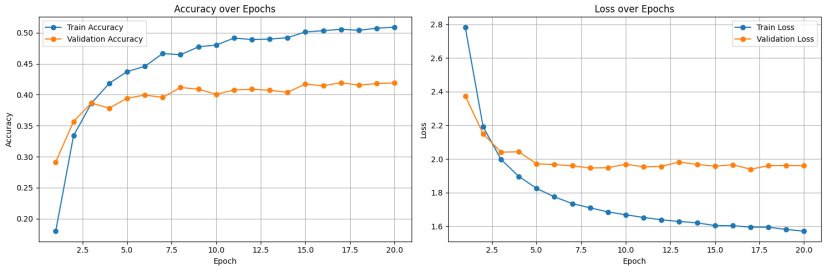
(a) ViT Base Patch16-224 (Head Only) 27 class ASL Confusion Matrix



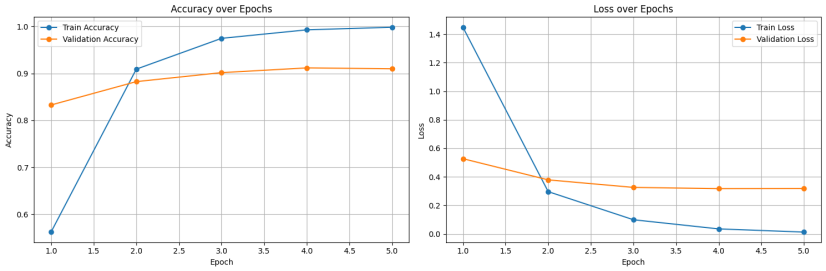
(b) EfficientViT-M5 Tiny 27 class ASL Confusion Matrix



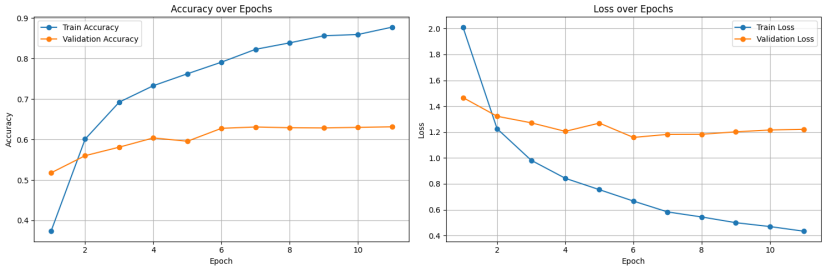
(c) Deit-III Small 27 Class ASL Confusion Matrix



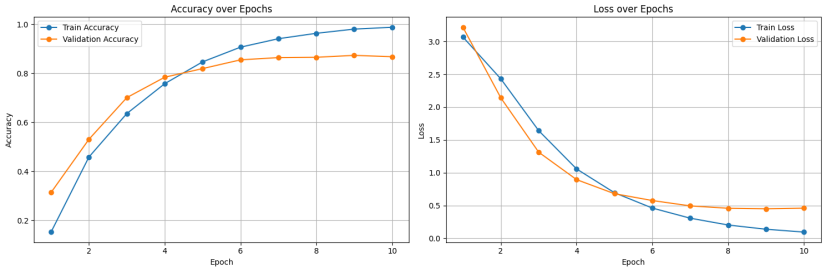
(a) ViT Tiny Patch16-224 (Head Only) Azerbaijan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



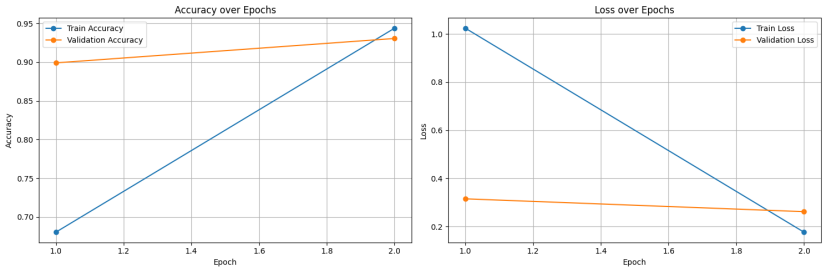
(b) ViT Tiny Patch16-224 Azerbaijan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



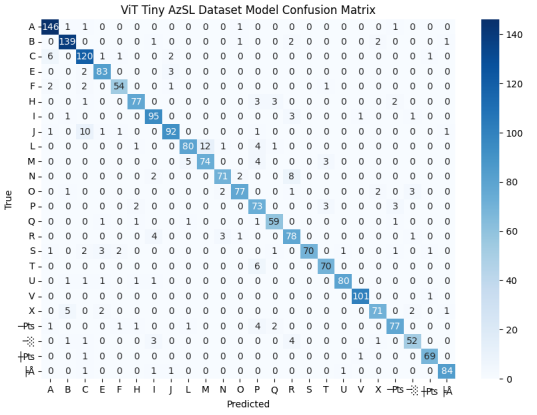
(c) ViT Base Patch16-224 (Head Only) Azerbaijan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



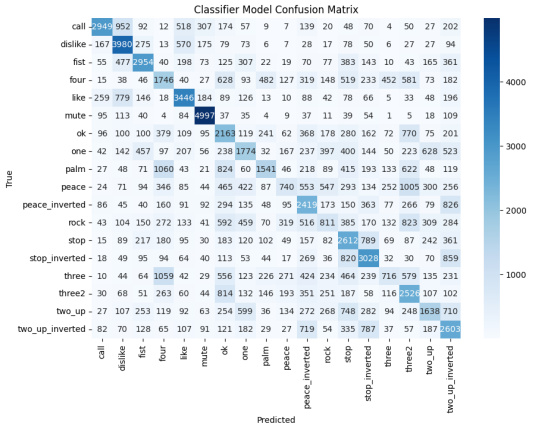
(d) EfficientViT-M5 Tiny Azerbaijan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



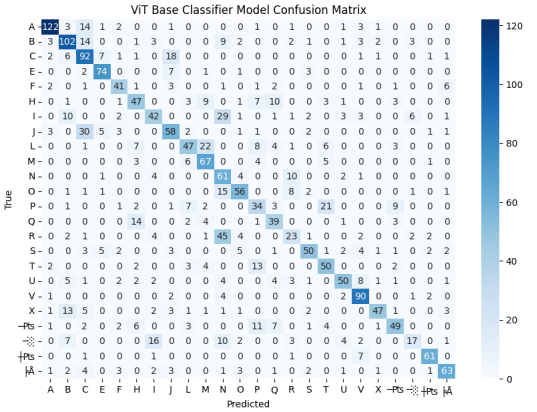
(e) Deit-III Small Tiny Azerbaijan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



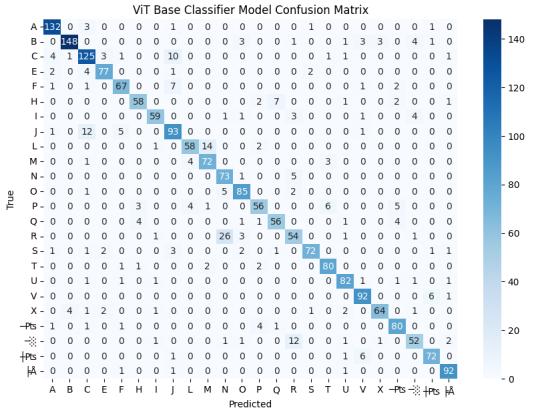
(a) ViT Tiny Patch16-224 Azerbaijan Sign Language Confusion Matrix



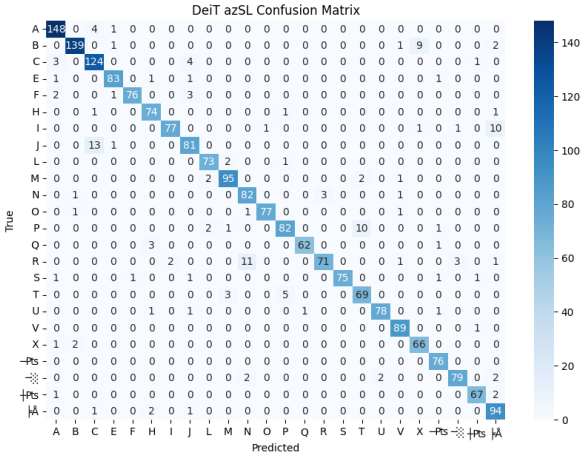
(b) ViT Tiny Patch16-224 (Head Only) Azerbaijan Sign Language Confusion Matrix



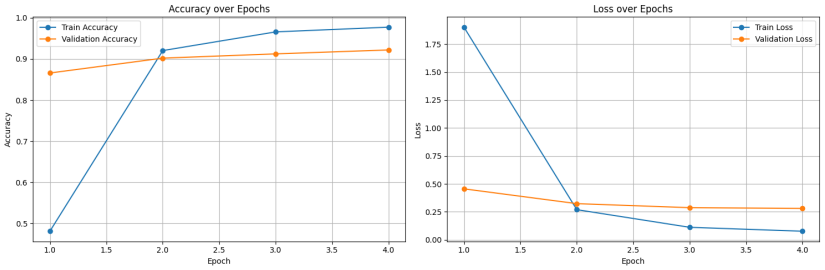
(a) ViT Base Patch16-224 (Head Only) Azerbaijan Sign Language Confusion Matrix



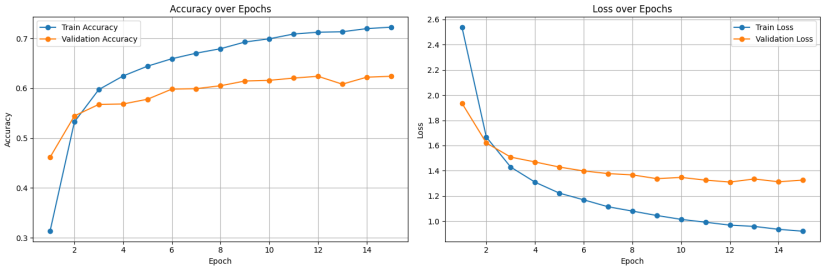
(b) EfficientViT-M5 Tiny Azerbaijan Sign Language Confusion Matrix



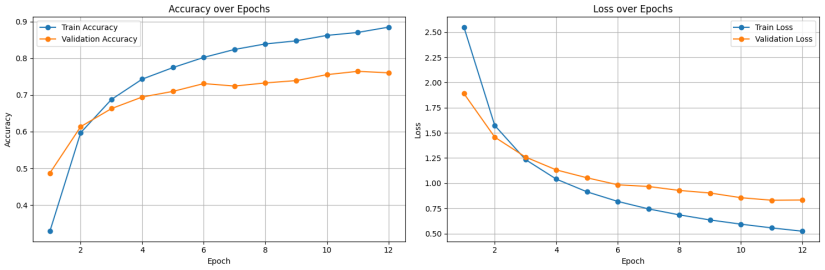
(c) Deit-III Small Azerbaijan Sign Language Confusion Matrix



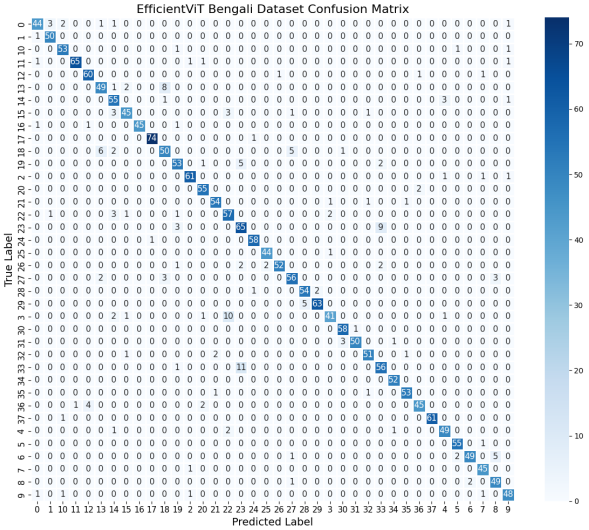
(a) ViT Tiny Patch16-224 Bengali Sign Language Train - Validation Accuracy & Train - Validation Loss per epoch



(b) ViT Tiny Patch16-224 Bengali Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



(c) ViT Base Patch16-224 (Head Only) Bengali Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



(d) EfficientViT-M5 Tiny Bengali Confusion Matrix

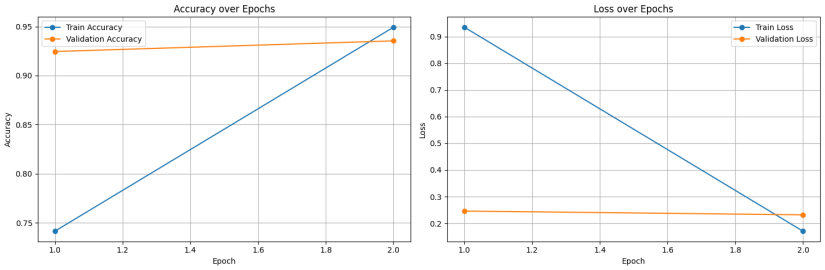
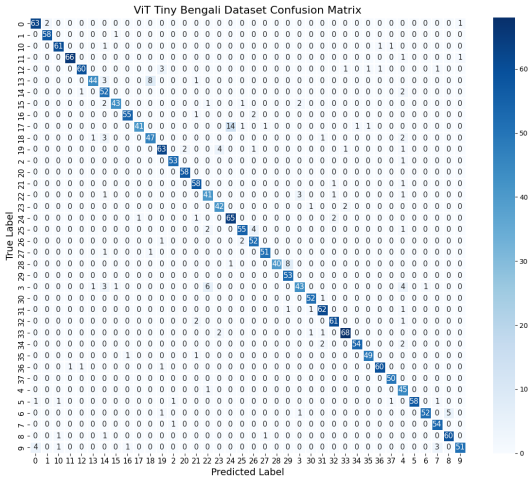
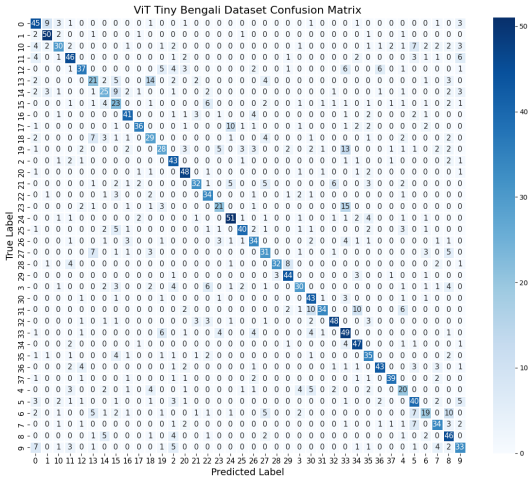


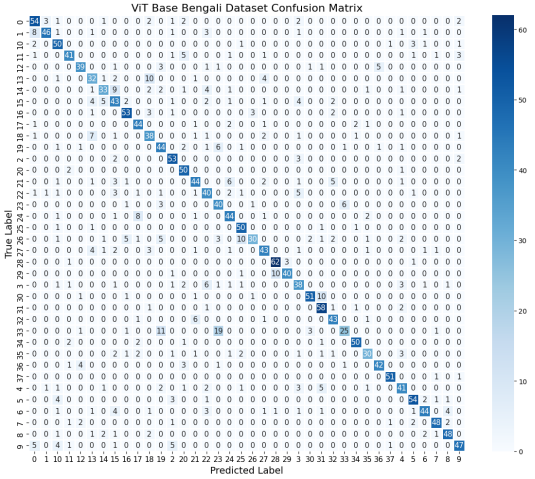
Figure 47: DeiT-III Small Tiny Bengali Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



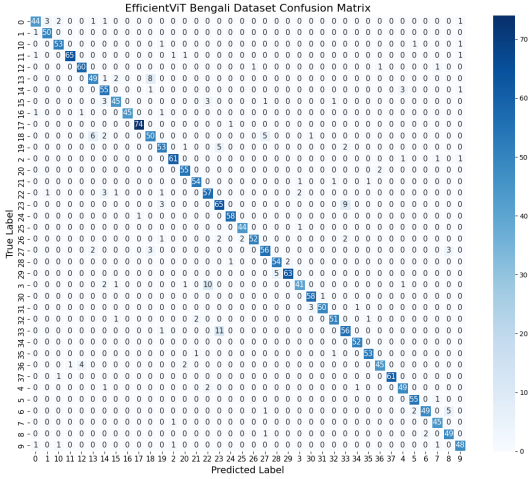
(a) ViT Tiny Patch16-224 Bengali Sign Language Confusion Matrix



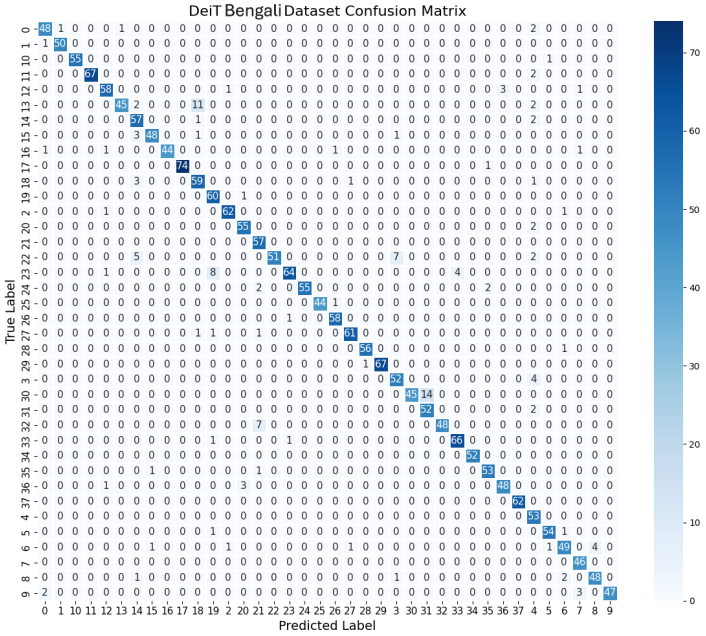
(b) ViT Tiny Patch16-224 (Head Only) Bengali Sign Language Confusion Matrix



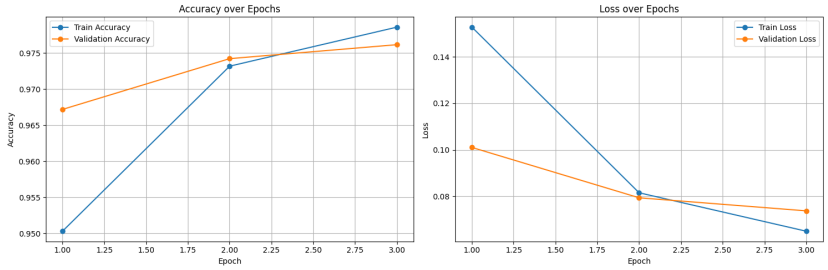
(a) ViT Base Patch16-224 (Head Only) Bengali Confusion Matrix



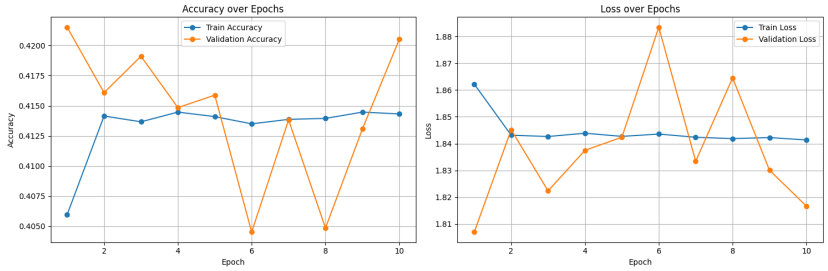
(b) EfficientViT-M5 tiny Bengali Confusion Matrix



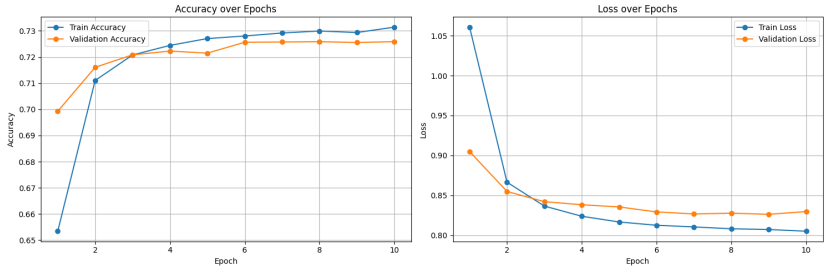
(c) Deit-III Small Bengali Sign Language Confusion Matrix



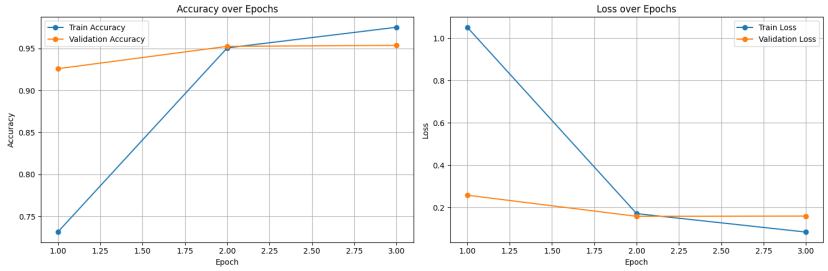
(a) ViT Tiny Patch16-224 Hand Gesture Recognition 10% Train - Validation Accuracy && Train - Validation Loss per epoch



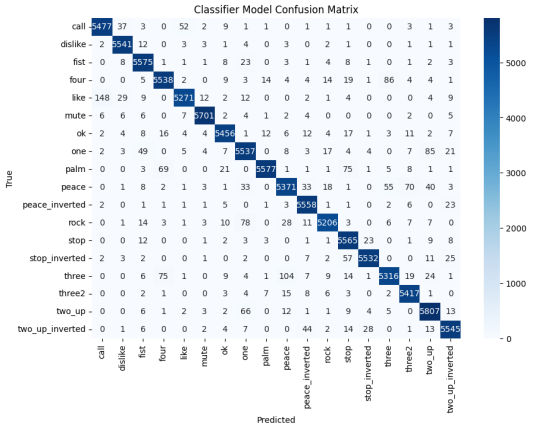
(b) ViT Tiny Patch16-224 Hand Gesture Recognition 10% Train - Validation Accuracy && Train - Validation Loss per epoch



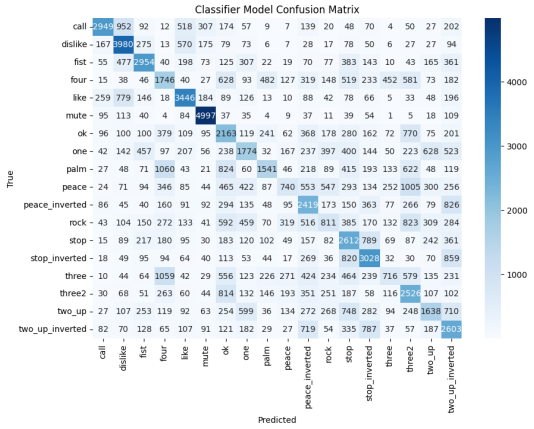
(c) ViT Base Patch16-224 (Head Only) Hand Gesture Recognition 10% Train - Validation Accuracy && Train - Validation Loss per epoch



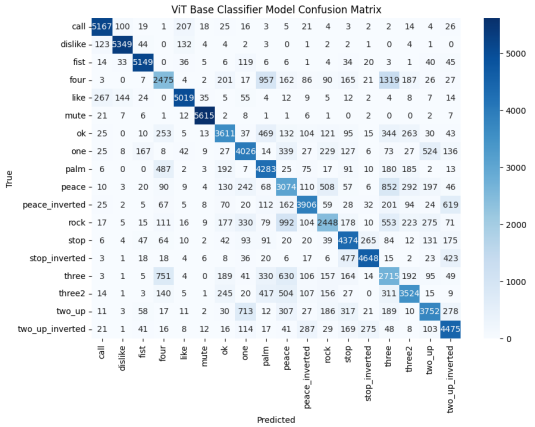
(d) EfficientViT-M5 Tiny Hand Gesture Recognition 10% Train - Validation Accuracy && Train - Validation Loss per epoch



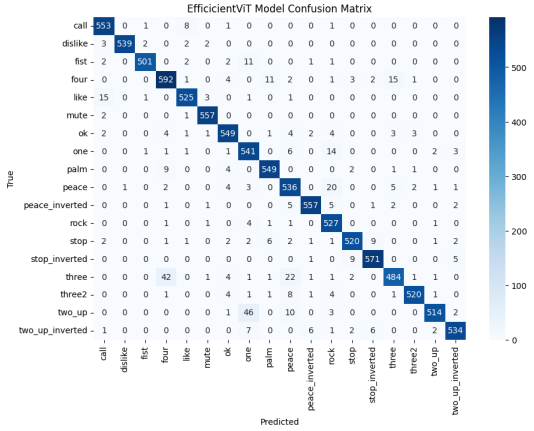
(a) ViT Tiny Patch16-224 Hand Gesture Recognition 10% Confusion Matrix



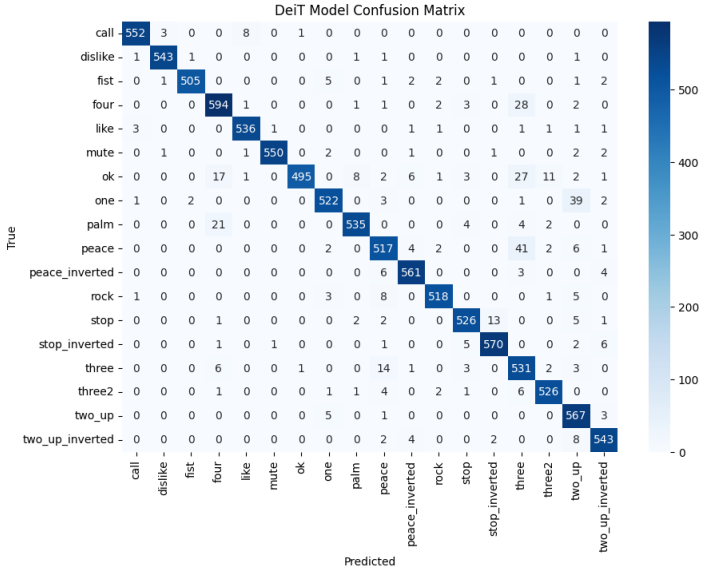
(b) ViT Tiny Patch16-224 (Head Only) Hand Gesture Recognition 10% Confusion Matrix



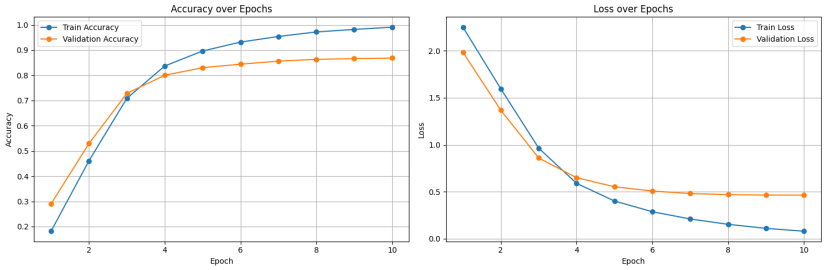
(a) ViT Base Patch16-224 (Head Only) Hand Gesture Recognition 10% Confusion Matrix



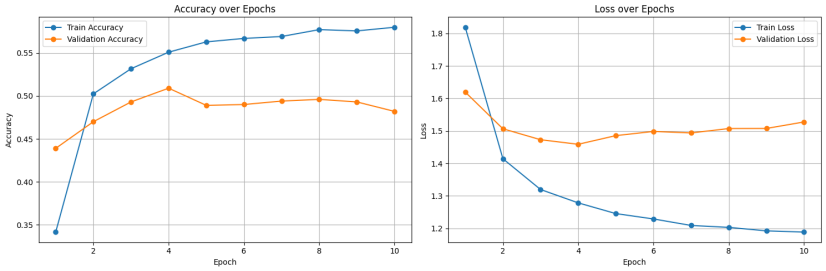
(b) EfficientViT-M5 Tiny Hand Gesture Recognition 10% Confusion Matrix



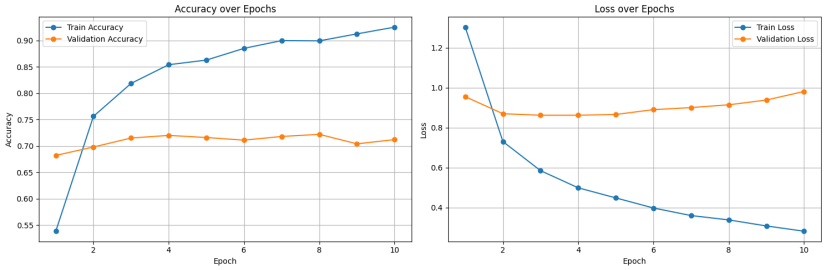
(c) Deit-III Small Hand Gesture Recognition 10% Sign Language Confusion Matrix



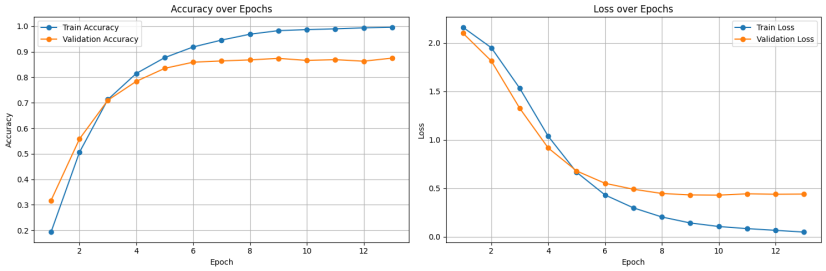
(a) ViT Tiny Patch16-224 Kenyan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



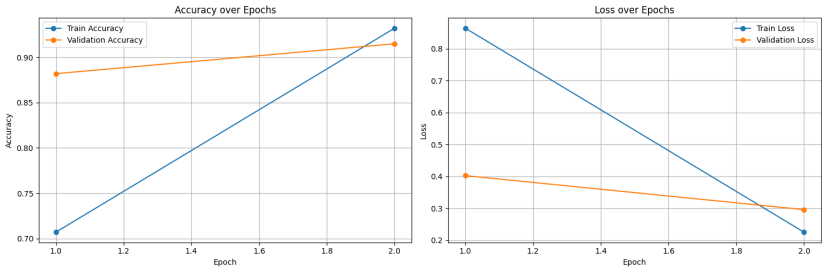
(b) ViT Tiny Patch16-224 Kenyan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



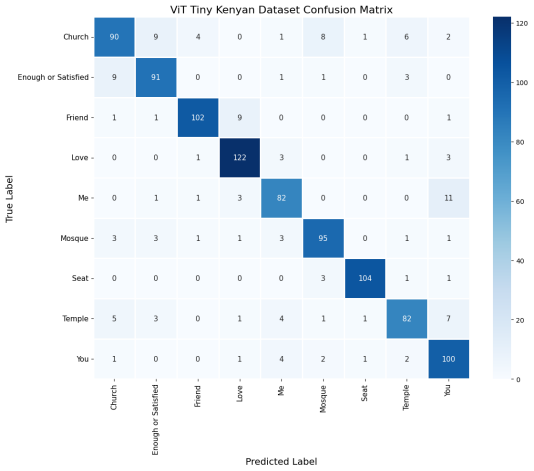
(c) ViT Base Patch16-224 (Head Only) Kenyan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



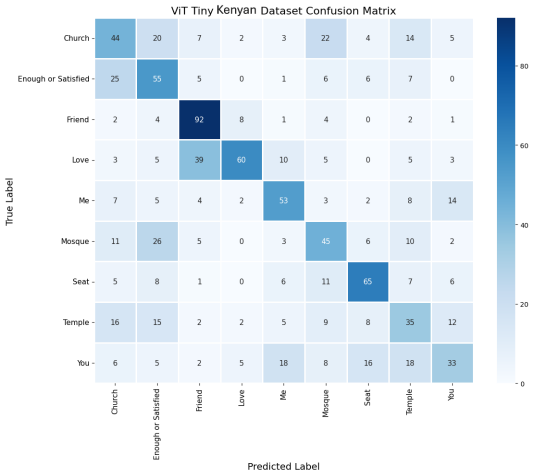
(d) EfficientViT-M5 Tiny Kenyan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



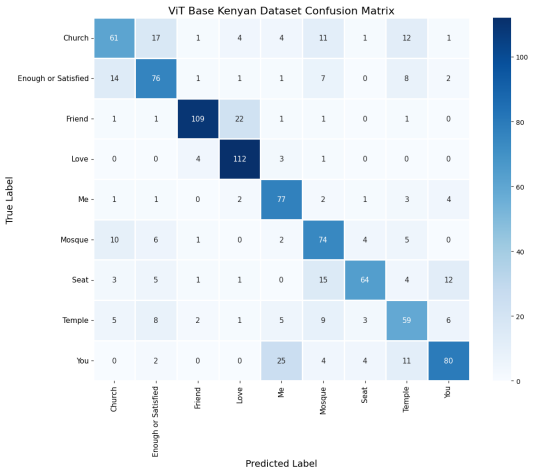
(e) Deit-III Small Kenyan Sign Language Train - Validation Accuracy && Train - Validation Loss per epoch



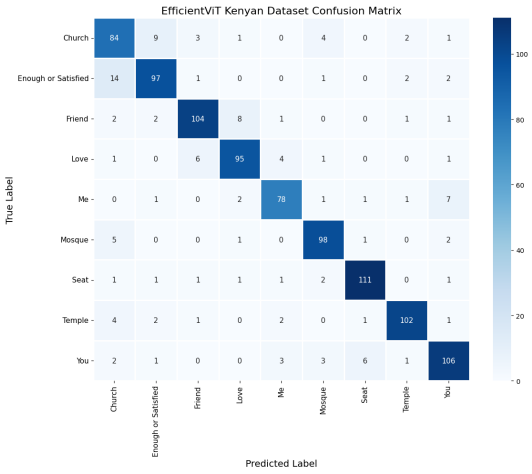
(a) ViT Tiny Patch16-224 Kenyan Sign Language Confusion Matrix



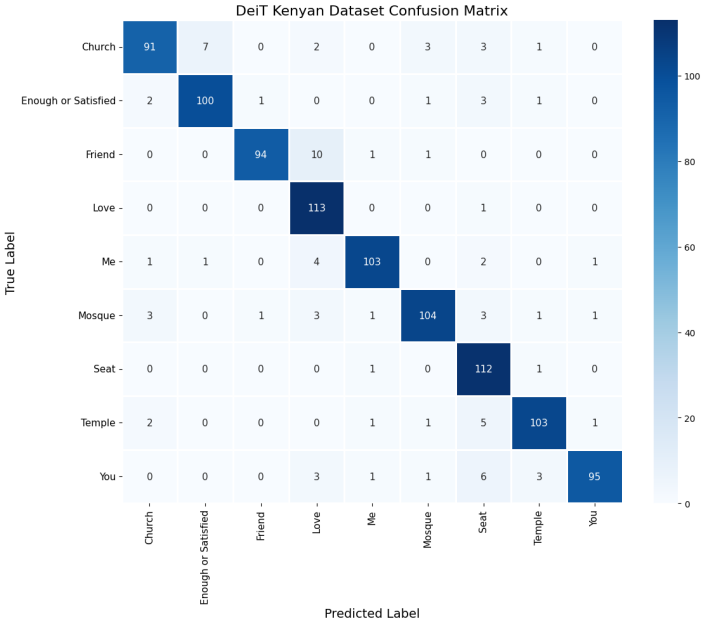
(b) ViT Tiny Patch16-224 (Head Only) Kenyan Sign Language Confusion Matrix



(a) ViT Base Patch16-224 (Head Only) Kenyan Sign Language Confusion Matrix



(b) EfficientViT-M5 Tiny Kenyan Sign Language Confusion Matrix



(c) Deit-III Small Kenyan Sign Language Sign Language Confusion Matrix