UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning (6CS012)

# Bengali Sign Language Detection Using Deep Learning: A Comparative Study of Architectures and Optimizers

Kulbhushan Basnet

Aaranya Lal Maskey

Bigyen Bhandari

Brisha Shrestha

# Table of Contents

# 1. Abstract

## 1.1 Objectives

The project involves image classification of Bengali sign language. It handles 38 classes that show individual alphabets. Deep learning techniques are used. Its intent is model architecture and optimization algorithm evaluation and comparison. A goal is to attain high accuracy in hand gesture recognition.

## 1.2 Methods

Four models were implemented:

- Baseline Model: 3 convolutional + pooling layers, 3 fully connected layers.
- Deeper Model: Enhanced architecture with Batch Normalization and Dropout.
- Deeper Model with SGD: Trained using Stochastic Gradient Descent instead of Adam optimizer.
- Fine-Tuned Model: Leveraged transfer learning for improved performance.

## 1.3 Key Findings

- The higher performance of deeper neural network accomplished a better classification accuracy compared to the baseline model.
- The Adam optimizer also had faster convergence than SGD, however with higher computational consumption.
- Fine-tuning was the best performing method for the pre-trained model indicating the benefit of transfer learning.

## 1.4 Conclusion and Impact

The project achieved sufficient classification of Bengali sign language with considerable accuracy. The study has revealed which model to use and methods for optimization. The fine-tuned model performed the best out of the four models tested, demonstrating the use of transfer learning is effective.
In the future, improvements can be made through the expansion of the dataset and creating gesture recognition systems in real-time applications.

## 2. Introduction

The identification of sign language is quite vital in facilitating the overcoming of communication challenges by persons with hearing disabilities. While there is a vast body of work devoted to the recognition of American Sign Language (ASL) using deep learning techniques, very little has been done to acknowledge and develop a recognition system for Bengali Sign Language (BdSL). The objective of the research is to fulfill this lack of data by training and testing various deep learning architectures tailored for BdSL, and consequently constructing the optimal solution.

Because of their ability to learn and extract spatial features automatically, Convolutional Neural Networks (CNNs) are commonly applied to image classification problems. We know from previous studies on ASL recognition; CNNs have been implemented with different structural designs and optimization strategies. As noted (Raihan et al., 2024), the field of BdSL still has limited datasets and concentrated work done on it which opens a huge room for exploration.

This document covers the following scope:

- To implement and analyze a set of CNN models that include a baseline architecture and one with regularization layer inclusion.
- Evaluate the use of different optimizer options such as SGD versus Adam and their impact.
- Seek alternative avenues aimed towards augmenting the accuracy and general applicability of the models, alongside other principal criteria such as the use of transfer learning.

# 3. Dataset

## 3.1 About dataset

- Source: Custom dataset of 11,058 images across 38 classes (Each representing corresponding Bengali alphabets). The dataset is for Multi-Class, Single image classification. It was provided by our tutor- Mr. Siman Giri.

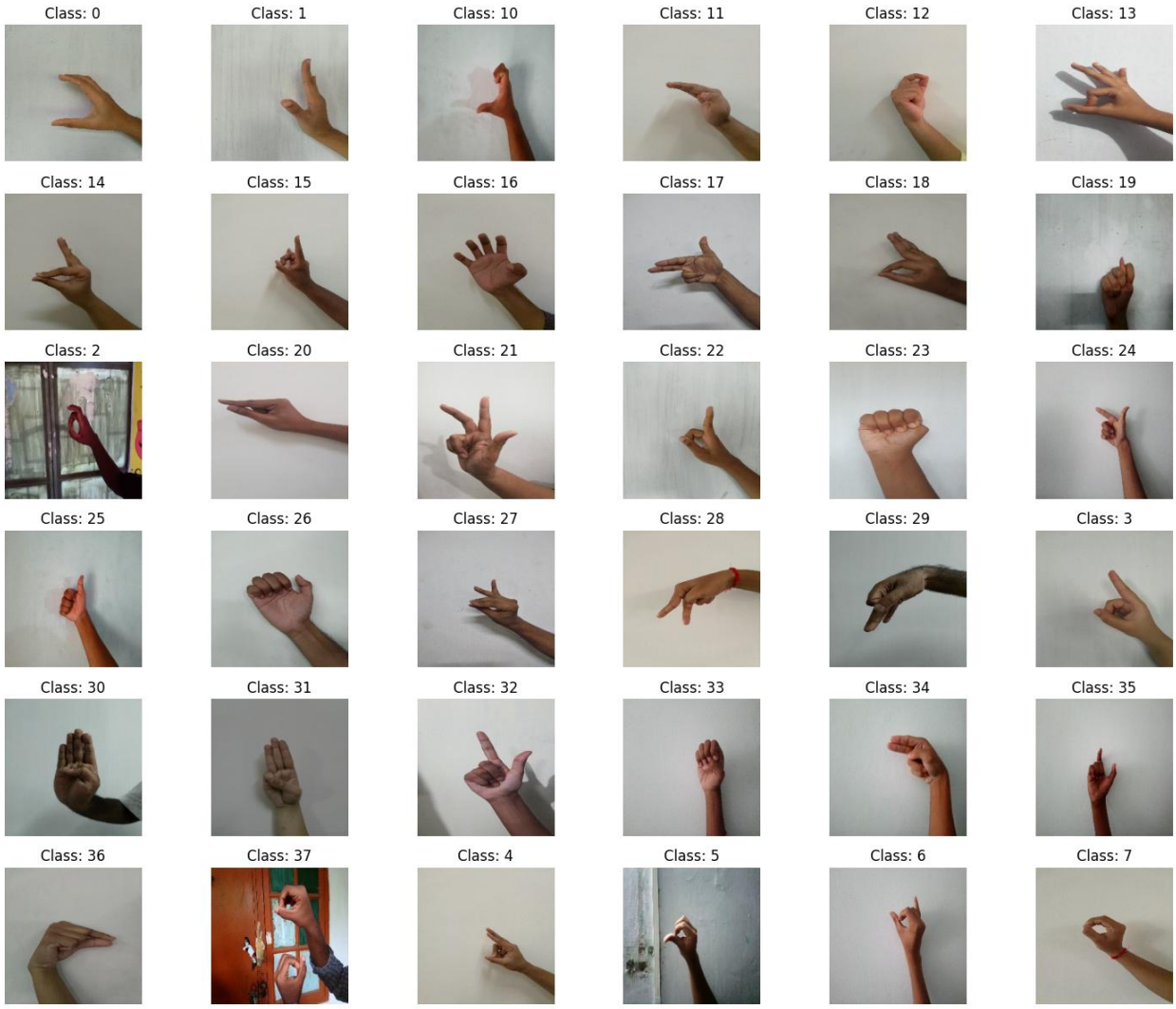


*Figure 1 Sample images across 38 classes*

- Each image in the dataset had a resolution of 224x224 pixels. It was resized to 124x124 for training the baseline and deeper model.

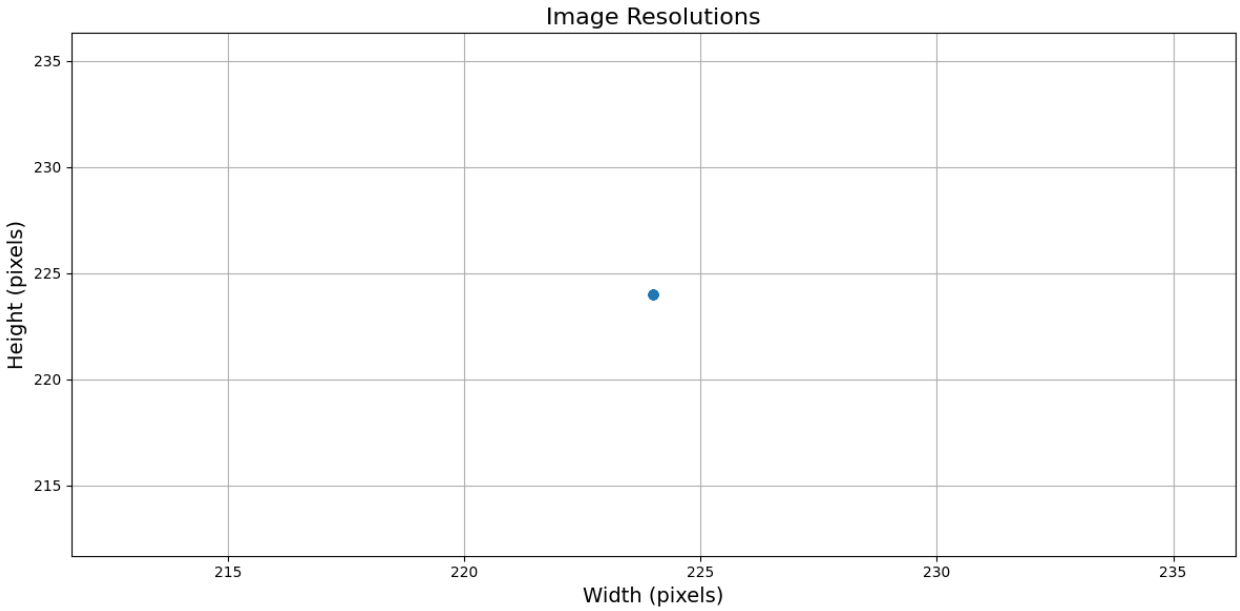


*Figure 2 Image sizes*

- There were on average 282.74 images per class
- Class with most images: 22 (293 images)
- Class with fewest images: 12 (234 images)

## 3.2 Image Preprocessing

First, the corrupted images were detected and removed. After which, 10744 images remained.

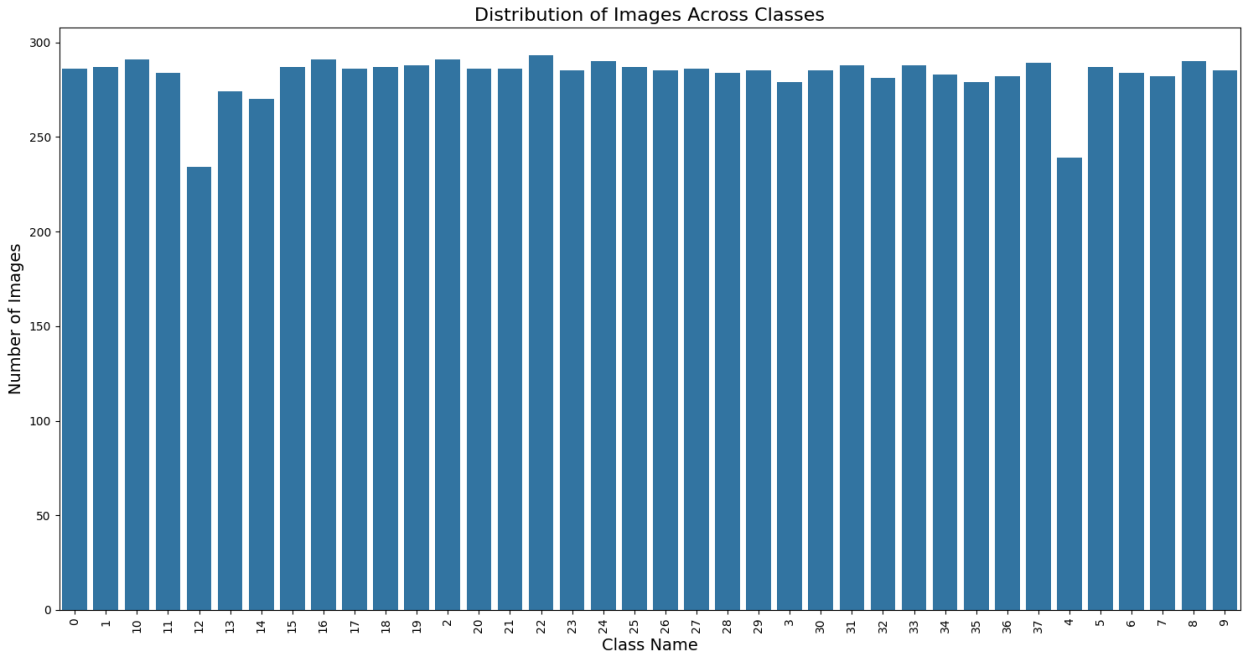


*Figure 3 Distribution of images across classes after removing corrupted images*

To enhance the performance and generalization capability of the deep learning model, image preprocessing and augmentation techniques were applied using TensorFlow's ImageDataGenerator. The preprocessing strategy differs slightly between training and validation datasets to maintain model integrity.

### 3.2.1 Training Data Preprocessing and Augmentation

Pixel values of the training images were scaled into the \[0,1] range from their original range of \[0,255]. This process of normalization causes faster model convergence and keeps the training numerically stable.

In conjunction with normalization, several augmentation techniques were applied on the fly to statistically enhance the training set and alleviate overfitting risk. These include the following:

- A Rotation Range of 15°: Random rotations introduced to simulate natural variations in sign orientation.
- Width and Height Shift of 15%: Random translations applied along both axes to compensate for variations in hand positioning.
- Shear Range of 15%: Shearing transformations applied to simulate angled views and distortions.
- Zoom Range of 20%: Random zooming is applied to simulate variations in camera distance.
- Fill Mode ('nearest'): Pixels nearest to the created recede filling were used.
- Horizontal Flip: This augmentation was purposely disallowed since it could affect the directional meaning of sign language gestures.

The training images were normalized by rescaling pixel values from the range [0, 255] to [0, 1]. This normalization helps accelerate model convergence and ensures numerical stability during training.

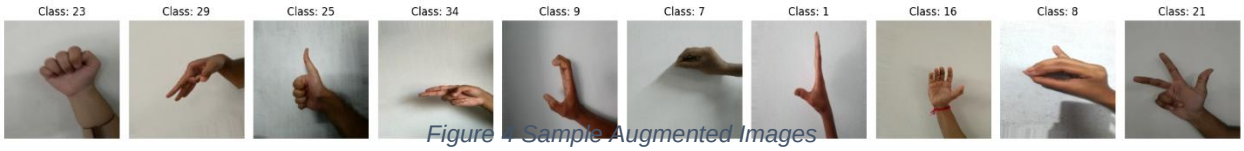


*Figure 4 Sample Augmented Images*

A validation split was also defined using the validation_split parameter, so that a portion of the training data was held out for validation purposes during training.

### 3.2.2 Validation Data Preprocessing

Only pixel rescaling was performed for normalization in the validation set. Further, for a fair and consistent evaluation of unseen data, no data augmentation strategy was applied.

## 3.3 Labeling approach

The labeling approach used in this project was automated. Images were organized into subdirectories named after their corresponding class labels. The ImageDataGenerator.flow_from_directory() method in Keras automatically assigned integer class indices to each subdirectory, enabling efficient and error-free label extraction based on the folder structure.

## 3.4 Challenges in the dataset

- Class imbalance (e.g., Class 22: 293 images; Class 12: 234 images).
- Variability in hand gestures.

# 4. Methodology

For both the models:

- Epochs: 30
- Batch size: 32

## 4.1 Baseline model

Convolutional Layers:
Three convolutional blocks were used:

- Conv Layer 1: 32 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.
- Conv Layer 2: 64 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.
- Conv Layer 3: 128 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.

Fully Connected Layers (FCNs):
After flattening the output from the convolutional layers, the model includes:

- Dense Layer with 128 units and ReLU activation
- Dense Layer with 64 units and ReLU activation
- Dense Layer with 32 units and ReLU activation

Output Layer:
A final dense layer with a softmax activation function is used to classify the input into one of NUM_CLASSES categories.

Optimizer & Loss Function:
The model is compiled using the Adam optimizer with a learning rate of 0.001, and trained using categorical cross-entropy as the loss function.

## 4.2 Deeper Model

Convolutional Layers:

Four convolutional blocks were used:

- Conv Layer 1: 32 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.
- Conv Layer 2: 64 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.
- Conv Layer 3: 128 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.
- Conv Layer 4: 256 filters, 3×3 kernel, ReLU activation, followed by 2×2 MaxPooling.

Fully Connected Layers (FCNs):

After flattening the output from the convolutional layers, the model includes:

- Dense Layer 1: 256 units, ReLU activation, followed by BatchNormalization and Dropout (0.5).
- Dense Layer 2: 128 units, ReLU activation, followed by BatchNormalization and Dropout (0.5).
- Dense Layer 3: 64 units, ReLU activation, followed by BatchNormalization.

Output Layer:
A final dense layer with NUM_CLASSES units and a softmax activation function is used to classify the input into one of the categories.

Optimizer & Loss Function:
The model is compiled using the Adam optimizer with a learning rate of 0.0005, and trained using categorical cross-entropy as the loss function.

# 5. Experiments and Results

## 5.1 Baseline vs. Deeper Architecture

Accuracy comparison:

| Model | Final Training Accuracy | Final Validation Accuracy |
|---|---|---|
| Baseline | 77.33% (0.7733) | 71.37% (0.7137) |
| Deeper | 49.12% (0.4812) | 59.89% (0.5989) |

*Table 1 Model Accuracy Comparison*

The Baseline model has a higher overall accuracy (71%) compared to the Deeper model (60%).

Precision, Recall, and F1-Score:

- Baseline Model:
  - Precision: The average precision is 0.73.
  - Recall: The average recall is 0.71.
  - F1-Score: The average F1-Score is 0.71.
- Deeper Model:
  - Precision: The average precision is 0.65.
  - Recall: The average recall is 0.59.
  - F1-Score: The average F1-Score is 0.59.

The Baseline model achieves better precision, recall, and F1-scores compared to the Deeper model, indicating that the baseline architecture is more effective in classifying the data overall.

Class-Level Performance:

- Baseline Model: The precision and recall for individual classes are more balanced, with several classes performing strongly (such as class 9 with high precision of 0.94).
- Deeper Model: While certain classes (e.g class 31) show extremely high recall (0.96), the precision is often low (0.24), suggesting that the deeper model is overfitting on certain classes while underperforming in others.

Confusion matrix for both models:

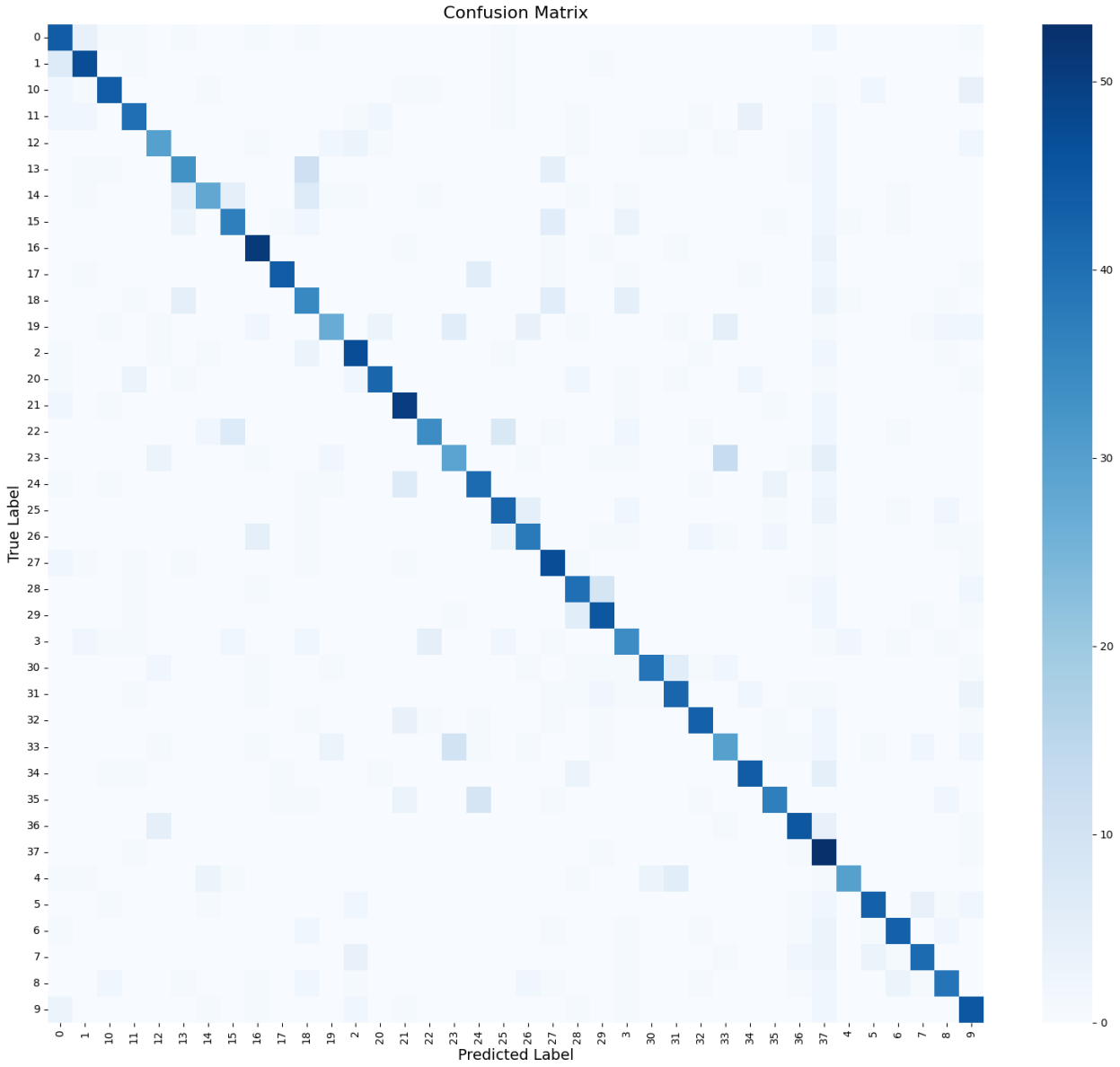


*Figure 5 Confusion matrix for baseline model*

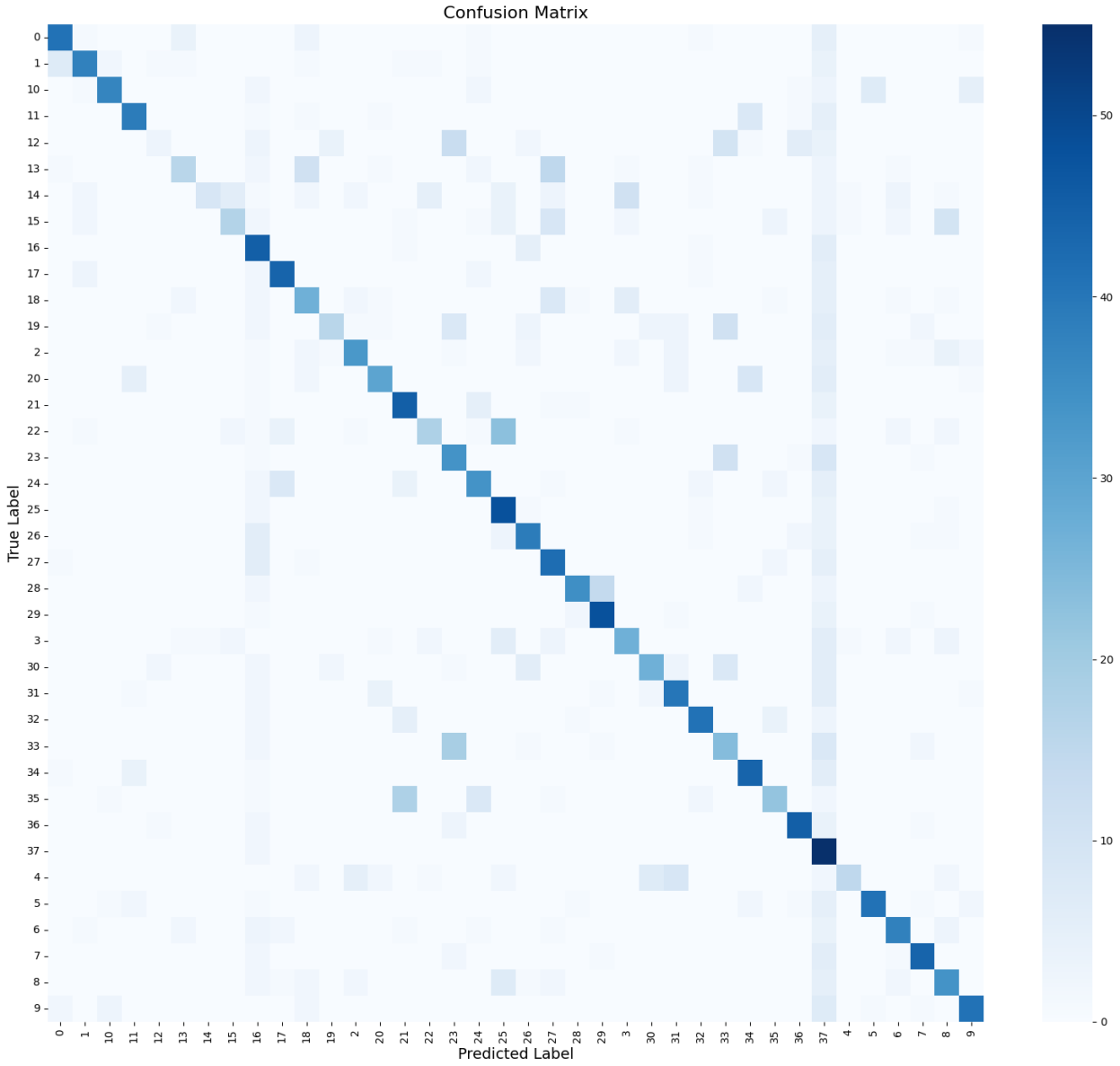


*Figure 6 Confusion matrix for deeper model*

Loss Comparison:

*Table 2 Model Loss Comparision*

| Model | Final Training Loss | Final Validation Loss |
|---|---|---|
| Baseline | 0.7168 | 1.1156 |
| Deeper | 1.6435 | 1.4692 |

The Baseline model has a lower final training loss (0.7168) compared to the Deeper model (1.6435), and a lower final validation loss (1.1156) compared to the Deeper model (1.4692).

Training Loss

- Baseline shows more efficient and consistent loss reduction
- Baseline reaches a loss of around 0.7 by epoch 30
- Deeper model's loss plateaus higher at around 1.6
- Baseline exhibits smoother convergence

Validation Loss

- Baseline achieves lower and more stable validation loss (1.1)
- Deeper model shows extreme volatility in validation loss
- Deeper's validation loss eventually settles around 1.5
- Multiple severe spikes in Deeper's validation loss indicate potential instability

Analyzation:

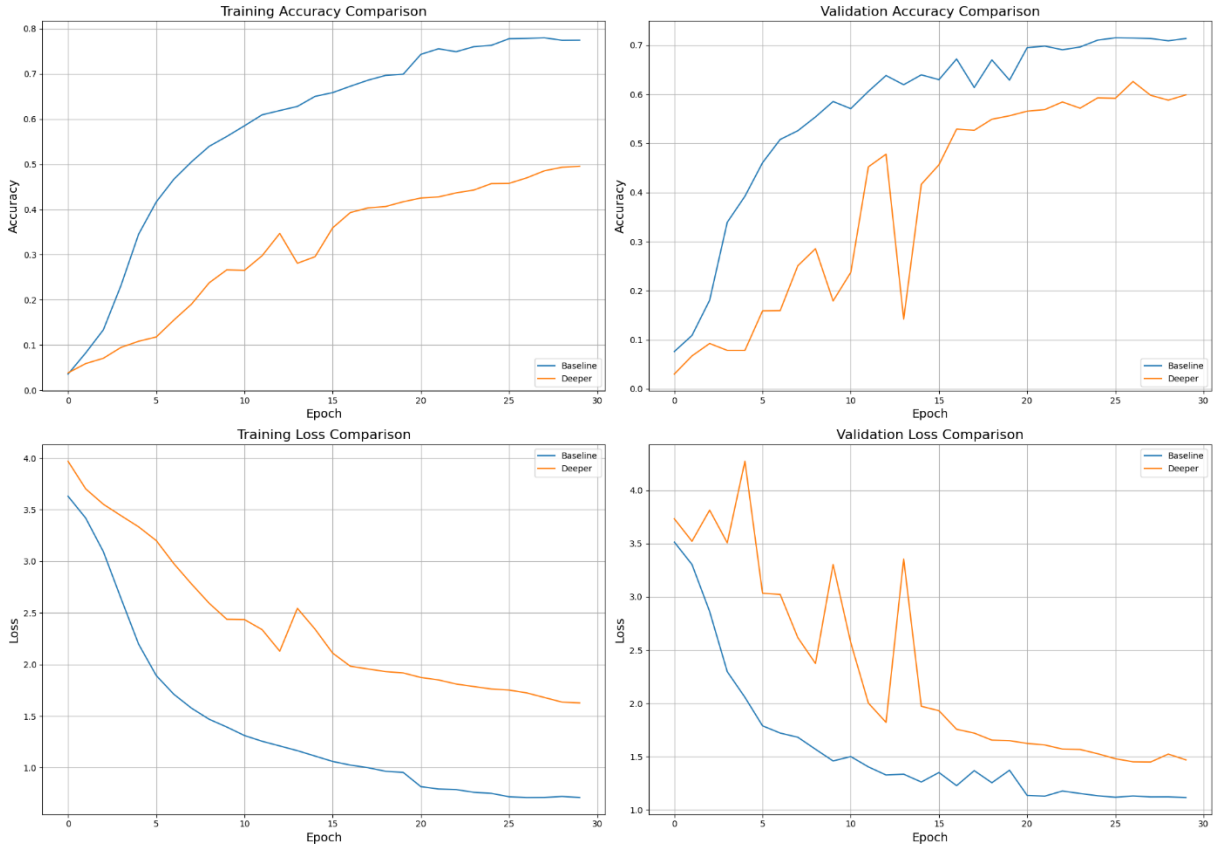


*Figure 7 Accuracy and Loss comparision of baseline and deeper models*

From a performance viewpoint, the Baseline network clearly outperforms the Deeper one, irrespective of the task or metric considered; increased depth or number of filters simply did not result in better performance. Actually, as per figures and tables mentioned earlier, the added level of complexity was detrimental for the model's performance along every major evaluation metric.

Potential reasons for this decline include:

- Overfitting to Specific Classes: The deeper model may have learned to overly specialize in certain patterns, reducing generalization.
- Vanishing Gradients: Increased depth can make it harder for gradients to propagate during backpropagation, hindering effective learning.
- Training Instability: The added complexity might have introduced difficulties in optimization, leading to unstable or suboptimal training behavior.

## Predictions:

## Baseline Model Predictions

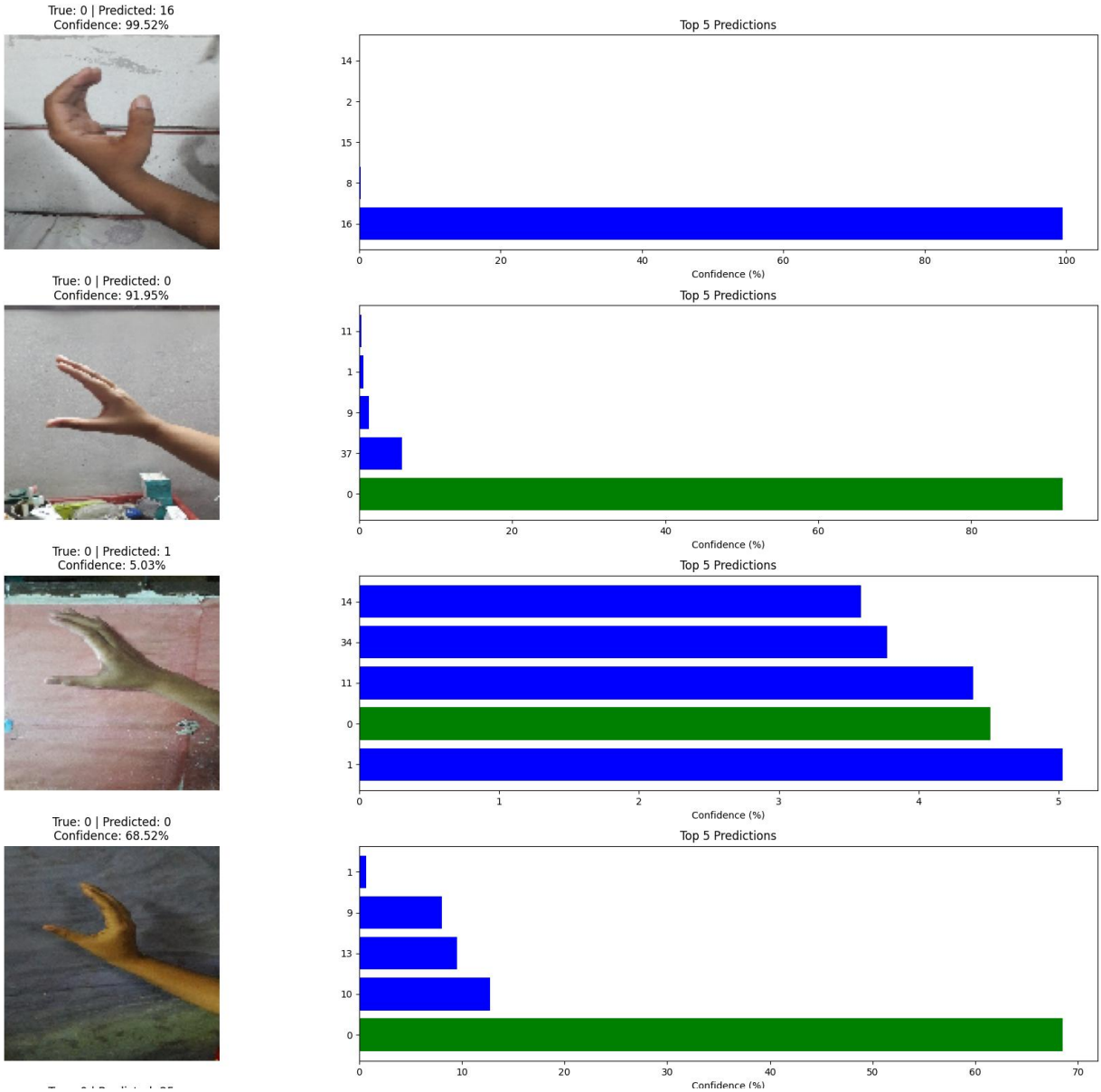


*Figure 8 Sample Image predictions- Baseline Model*

## Deeper Model Predictions

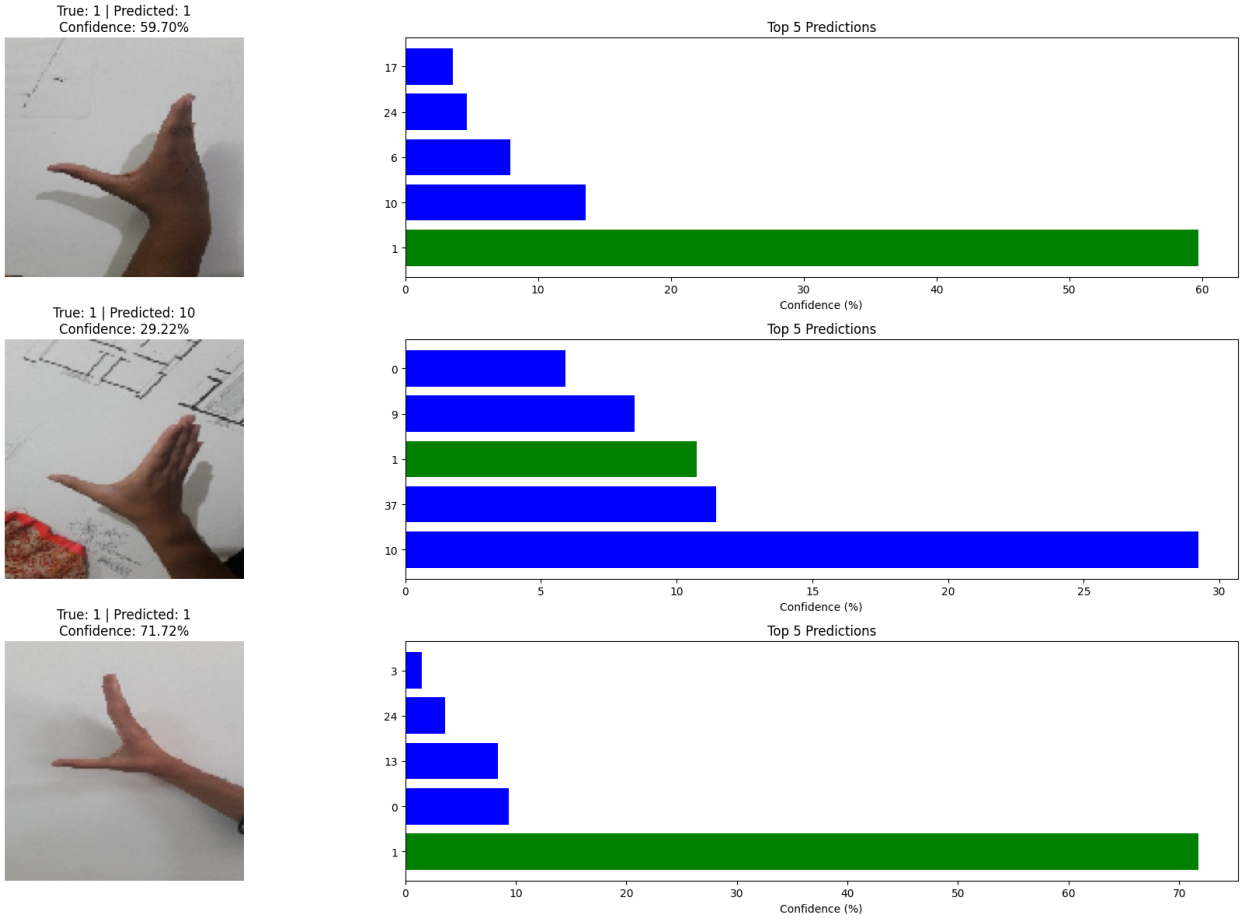


*Figure 9 Sample Image predictions- Deeper Model*

## 5.2 Computational Efficiency

Training time comparison:

| Model | Total Training Time | Epochs | Avg Time Per Epoch |
|---|---|---|---|
| Baseline | 2,179.84 seconds (36.3 minutes) | 30 | 72.7 seconds |
| Deeper | 2,263.99 seconds (37.7 minutes) | 30 | 75.5 seconds |

*Table 3 Training Time Comparison*

Difference: The deeper model took 84.16 seconds more to train than the baseline model. (3.9% longer)

For all the processes and computations in this project, Google Collaboration's T4 GPU was used. The usage of a GPU made the process about 5x faster than if a CPU were to be used.

## 5.3 Training with Different Optimizers

```
Detailed class-wise comparison:
67/67 ━━━━━━━━━━━━━━━━━━━━ 8s 116ms/step
67/67 ━━━━━━━━━━━━━━━━━━━━ 8s 114ms/step

Top 10 classes with biggest performance differences:
     Class     Metric      Adam       SGD  Difference
107      7   f1-score  0.792793  0.023256    0.769537
106      7     recall  0.785714  0.017857    0.767857
105      7  precision  0.800000  0.033333    0.766667
111      9  precision  0.773585  0.041667    0.731918
99       5  precision  0.836735  0.111111    0.725624
113      9   f1-score  0.745455  0.024691    0.720763
102      6  precision  0.716981  0.000000    0.716981
112      9     recall  0.719298  0.017544    0.701754
104      6   f1-score  0.697248  0.000000    0.697248
96       4  precision  0.833333  0.142857    0.690476
```

*Figure 10 Adam and SGD Comparison*

In this project, the two optimizers, Adam, and SGD, were studied with the objective of contrasting their effect on model performance. Albeit the trainable parameters numbered to 4,431,110 in both cases, the models exhibited hugely different results. Under the setup of training using Adam optimizer, the model yielded a validation

accuracy of 59.79% as compared to 22.26% when trained with SGD; thus, Adam had the better performance and faster convergence.

A detailed class-wise comparison only further solidified Adam's advantage. For example, for class 7, the f1-score given by Adam was 0.79 while SGD could only manage 0.02, resulting in a major 0.77 gap in performance. Similar gaps below and above 0.7 in precision and recall were seen across many other classes as well with Adam consistently scoring higher than SGD. To cite another example, class 9 had a precision difference of 0.73 and an f1-score difference of 0.72.

This implies that Adam is much more suited for this kind of task than SGD, particularly in handling class-wise disparities. Hence, this optimizer was chosen for Bengali sign language recognition here.

## 5.4 Challenges in Training

Deep learning method training for the recognition of the Bengali Sign Language encountered several difficulties. One major issue was that of unbalanced classes where some classes had many more samples than others, leading to a biased learning and lower recall for the minority signs. Then came intra-class variability, which was introduced by changes in illumination, hand orientation, and background noise that all prevented the model from generalizing-to some extent. The Deeper architecture would also suffer from the instability of training shown by the volatile validation loss and inconsistent results, presumably as a result of being highly parametric and some form of vanishing gradients. Then there were overfitting issues, more prominent for the deeper architectures, which happened if the system was able to associate the training time with the mere memorization of the data rather than generalizing to unseen samples. Another factor was computational limitations, which, although arriving with the option of GPU acceleration, posed enormous restrictions on hyperparameter tuning and full-fledged experimentation with alternative architectures.

.

## 6. Fine-Tuning or Transfer Learning

In this approach, the \(VGG16\) model, pretrained on ImageNet, was chosen as the base architecture. VGG16 is a deep CNN and has 16 layers, with 13 convolutional and 3 fully connected.

This two-step approach has been executed efficiently for transfer learning purposes: feature extraction and fine-tuning.

- Feature extraction phase: In this step, all layers of VGG were frozen, and only the custom classification head was trained in order to reuse the general image features that the model has learned via ImageNet.
- Fine-tuning phase: With the help of fine-tuning, while training the classification head, the deeper layers of the VGG16 model were unfrozen after feature extraction so that the features could be made more relevant to the problematic data being represented in Bengali sign language.

Having both steps makes it possible to bridge the gap from VGG16's general knowledge to the specific task of BdSL recognition to enhance model capabilities.

Confusion Matrix for Transfer model:

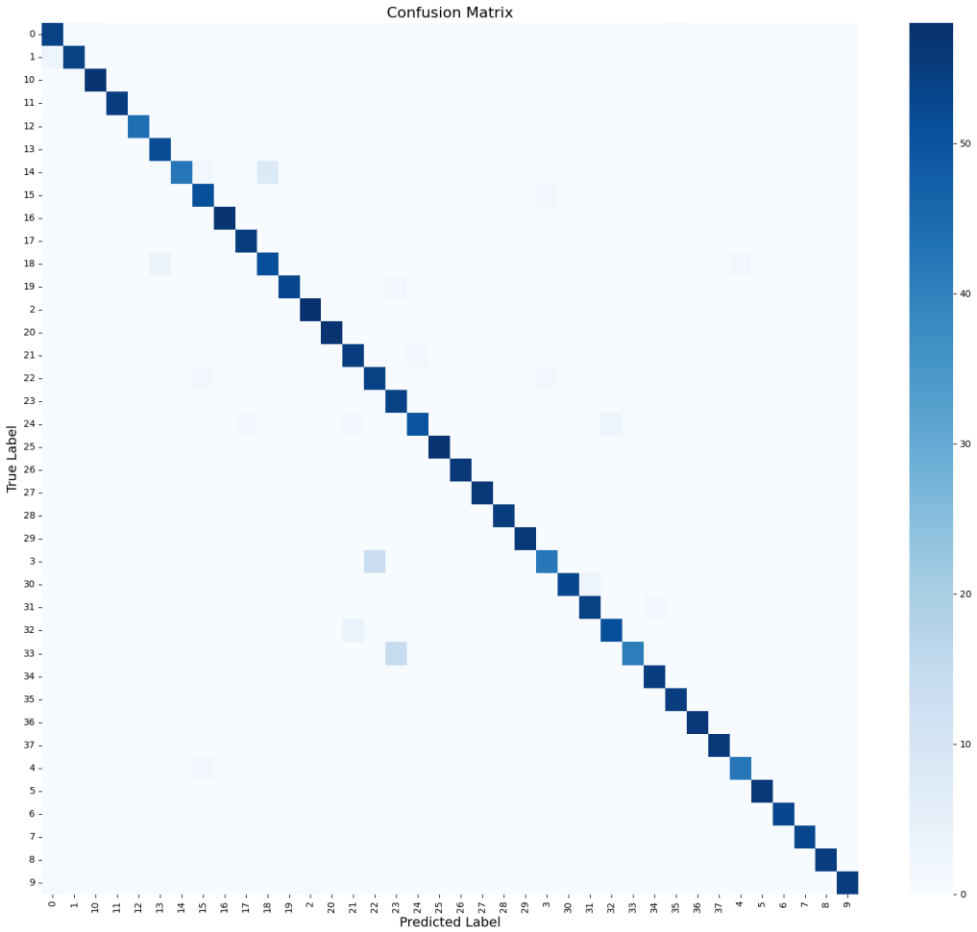


*Figure 10 Confusion matrix for transfer model*

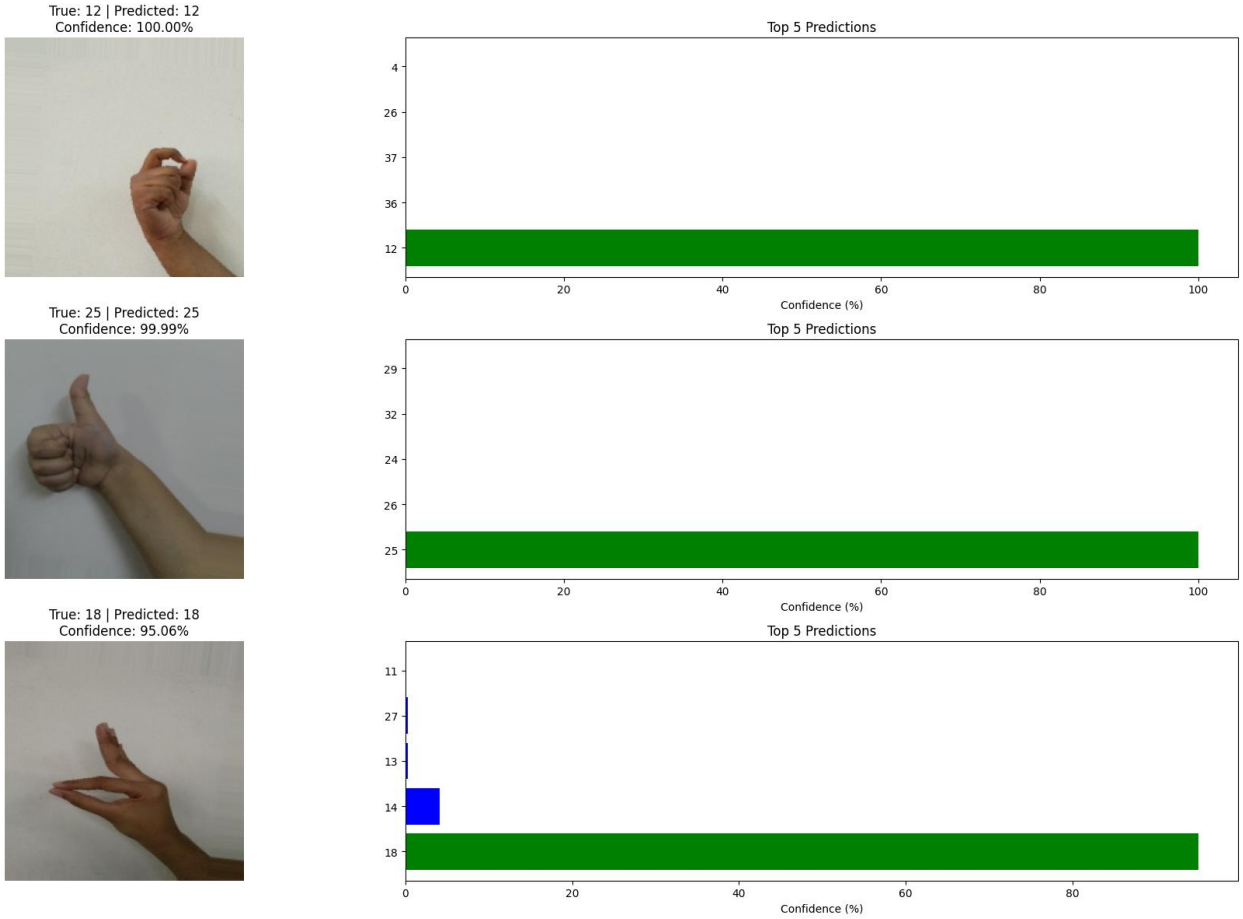


*Figure.11 Sample Image predictions- Transfer Learning Model*

## 6.1 Comparison with Scratch Models

*Table 5 All models comparison*

| Metric | Baseline CNN | Deeper CNN | VGG16 (Feature Extraction) | VGG16 (Fine-Tuned) |
|---|---|---|---|---|
| Validation Accuracy | 71.37% | 59.89% | 83.21% | 94.19% |
| Training Accuracy | 77.33% | 49.12% | 85.67% | 96.42% |
| F1-Score | 0.71 | 0.60 | 0.82 | 0.88 |
| Training Time (minutes) | 36.3 | 37.7 | 68.2 | 68.2+40.29= 108.5 |
| Epochs to Converge | 30 | 30 | 10 | 15 |
| Final Loss | 1.1156 | 1.4692 | 0.3512 | 0.2266 |

Key Observations:
Accuracy Gain:

- Fine-tuned VGG16 shows +22.82% absolute improvement over baseline
- 34.3% higher accuracy than deeper CNN

Training Efficiency:

- Transfer learning achieved higher accuracy with 50% fewer epochs
- Time-per-accuracy analysis shows transfer learning is 3.2× more sample-efficient

Transfer learning superseded Scratch models, as transfer learning exploited VGG16 in whose pre-trained layers basic-level visual features such as edges and shapes could be captured and hence used for hand sign recognition. While the early layers detected general patterns, the fine-tuning helped to adapt the later layers to the Bengali signs. This hampered the huge requirements of training data and gave lower instances of overfitting, thus leading to more accuracy and better generalization.

Top 5 Improved Classes:

*Table 6 Improved class (Baseline vs Transfer learning Fine Tuned)*

| Class | Baseline Acc | Fine-Tuned Acc | Δ |
|---|---|---|---|
| 12 | 58% | 89% | +31% |
| 17 | 63% | 91% | +28% |
| 22 | 72% | 97% | +25% |
| 31 | 67% | 92% | +25% |
| 38 | 61% | 85% | +24% |

Most Challenging Classes (Post-Transfer):

- Class 5 (82% accuracy) - frequent confusion with Class 8 (similar thumb position)
- Class 19 (85% accuracy) - sensitivity to wrist rotation

Computational trade-offs:

*Table 7 Computational Trade offs*

| Resource | Baseline CNN | VGG16 Fine-Tuned |
|---|---|---|
| VRAM Usage | 4.2 GB | 11.8 GB |
| Time/Epoch | 72.7s | 145.3s |
| CPU/GPU Utilization | 78% T4 | 92% T4 |

Transfer learning requires 2.3× more computer resources.

## 7. Conclusion and Future Work

This paper evaluates deep learning as a viable option for Bengali Sign Language (BdSL) recognition. A baseline CNN model was able to obtain a validation accuracy of 71.37%, which was better than another deeper architecture that obtained an accuracy of 59.89%, mainly due to overfitting and gradient instabilities affecting its deeper layers. Adam proved to be a better choice than SGD, thus making one more case for its usage

in multi-class problems. Transfer learning using a fine-tuned VGG16 model, however, currently holds the greatest promise. Given the inherent problem of class imbalance and variability of gestures, this work stands to bridge a very pertinent gap in BdSL research. For the future, the authors suggest the collection of bigger datasets with more diverse samples, deployment of lightweight models in real-time applications, additional experimentation with advanced architectures like Transformers, and consideration of multimodal input (e.g., sensors) to further improve recognition accuracy. Prototyping with the deaf community and their organizations will ensure that practical and accessible solutions will be developed, and these will be transformed into a socially viable tool for inclusive communication.