Sign Detection using Deep Learning

Jaimin Gajjar (B20AI014), Saurabh Modi (B20EE035), Mukul Shingwani (B20AI023)

Abstract

This project leveraged advanced Deep Learning (DL) tools such as ONNX, Torchscript, and Wandb to develop an accurate machine learning model capable of detecting American Sign Language (ASL) gestures. The model was preprocessed using contour image conversion, segmentation, and resizing techniques to optimize its performance, while data augmentation techniques such as random Gaussian noise and random horizontal flip were applied to enhance its accuracy.
However, the true strength of the model lies in its deployment, as it was seamlessly implemented across various platforms using Pytorch, Torchscript, and ONNX. Furthermore, the use of Wandb allowed for efficient model monitoring and analysis, improving the overall effectiveness of the project. These DL tools demonstrate their efficacy in addressing real-world problems related to sign language detection and interpretation, ultimately improving the quality of life for individuals within the deaf community.

1. American Sign Language Detection

Our project aims to create a model that can recognize fingerspelling hand gestures and combine them to form complete words. The gestures we will train the model on are shown in figure 1. It contains 26 alphabet classes and 1 empty character class.

2. Literature Review

Hand gesture recognition involves four basic steps: data acquisition, data preprocessing, feature extraction, and sign classification. Extensive research has been conducted in this field in recent years.

2.1. Data acquisition

2.1.1 Vision based approach

Vision-based methods use a high resolution 1080p webcam to observe hand movements and positions, allowing for natural interaction with computers. However, detecting hands is challenging due to variability in appearance, skin color, viewpoint, scale, and camera speed. Nonetheless, vision-based methods remain an important area of research in human-computer interaction. Dataset -
Training Dataset
Validation Dataset

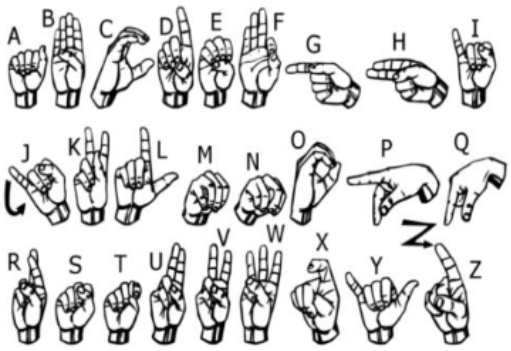

Figure 1. Sign Language

2.2. Data pre-processing and Feature extraction for vision based approach

We now present the preprocessing methods used to prepare images for a machine learning model that recognizes sign language gestures. The primary aim of preprocessing is to transform raw images into a format that can be easily processed by the model, while preserving the essential information required for accurate classification. It aims to improve the quality of the data, reduce noise and increase the accuracy and robustness of the model.

- The preprocessing pipeline used in this project involves several steps. First, we convert the raw image to a contour image to extract the shape of the gesture. Next, we segment the image into a black and white format to remove any background noise and enhance the contrast of the gesture. The resulting segmented image is then resized to a standard size of 224x224 pixels to ensure consistency across all images.

- To improve the robustness of the model, we applied data augmentation techniques such as random Gaus-

sian noise and random horizontal flip. These techniques introduce variations in the data and help prevent overfitting, which is a common problem in deep learning models.

3. Methodology

The approach used by the system is based on vision, where all signs are made using only bare hands, thus avoiding the need for any artificial devices for interaction.

3.1. Data Set Details

- We found a pre-made dataset in the form of raw images that matched our requirements.
- It is a contour image dataset formed using OpenCV library.
- It has 27 classes, 26 for the signs of A-Z and 1 class for an empty character.
- Training images contains 12,845 images. Validation set contains 4,268 images stratified across all classes.
- The size of each image is taken as 224 x 224 in 3 channel format(RGB).

3.2. Gesture Classification

Our approach uses two layers of algorithm to predict the final symbol of the user.

3.2.1 Working Code Pipeline

1. Use OpenCV to capture the video using frames and then taking the input from a pre-defined rectangle on top left of the screen.
2. Wait for the video capture frame to recognize a known class.
3. Send this processed image to the CNN model for prediction. The model returns the predicted class with the highest accuracy. These letters are then used to suggest a word.
4. The blank symbol is used to represent the space between words.

3.3. Finger spelling sentence formation

3.3.1 Implementation

1. The program detects letters when their count exceeds a certain value and there are no other letters nearby.
2. If a letter is detected, it is printed and added to the current string.

Figure 2. Predicted Words

3. If no letter is detected, the current dictionary is cleared to avoid incorrect predictions.
4. If a blank background is detected and the current buffer is empty, no spaces are detected.
5. If a blank background is detected and the current buffer is not empty, a space is printed to indicate the end of a word and the current string is added to the sentence below.
6. A suggestion library such as hunspell_suggest library in Python suggests correct alternatives for incorrect words and presents a list of matching words for the user to choose from. This feature helps to minimize spelling errors and aids in predicting difficult words and may be used.

4. Ablation Study

4.1. Comparative Study of Models

Model	Accuracy
Resnet18	67 %
Densenet	53 %
Efficientnet	58 %
CNN	68%

Figure 3. Comparative Study

1. **DenseNet:** DenseNet is a deep convolutional neural network architecture that connects each layer to every other layer in a feed-forward fashion. The architecture addresses the vanishing gradient problem in deep learning by enabling feature reuse throughout the network. DenseNet models have been shown to perform well on image classification tasks and achieve high accuracy.
2. **ResNet:** ResNet is a deep residual neural network architecture that uses shortcut connections to enable

the flow of information from early layers to later layers. This addresses the vanishing gradient problem and makes it possible to train very deep neural networks. ResNet models have been used successfully in image recognition and object detection tasks.

3. **EfficientNet:** EfficientNet is a neural network architecture that combines the advantages of existing architectures such as ResNet and Inception with an improved scaling method to achieve state-of-the-art performance on image classification tasks. The architecture is designed to be highly efficient in terms of computational resources while achieving high accuracy.

4. **Conventional CNN:** Conventional CNNs are a class of neural network architectures that have been widely used for image classification tasks. They typically consist of convolutional layers, max-pooling layers, and fully connected layers. The architecture is designed to learn hierarchical representations of the input data and has achieved high accuracy on a variety of image classification tasks.

In this project, all these models were evaluated for their accuracy in recognizing sign language gestures. The results showed that the accuracy of the conventional CNN model was highest among all the models tested.

4.2. CNN Model Architecture

Layer No.	Layer Type	Input Size	Output Size	Parameters	Memory	GPU Usage
1	**Conv2d**	3x224x224	80x220x220	(3x5x5x80) + 80 = 6080	107.85 MB	6.03 %
2	**BatchNorm2d**	80x220x220	80x220x220	160	21.33 MB	0.16 %
3	**ReLU**	80x220x220	80x220x220	0	21.33 MB	0 %
4	**MaxPool2d**	80x220x220	80x110x110	0	21.33 MB	0 %
5	Conv2d	80x110x110	80x106x106	(80x5x5x80) + 80 = 160080	1374.43 MB	80.5 %
6	**BatchNorm2d**	80x106x106	80x106x106	160	21.33 MB	0.16 %
7	**ReLU**	80x106x106	80x106x106	0	21.33 MB	0 %
8	**MaxPool2d**	80x106x106	80x53x53	0	21.33 MB	0 %
9	**Linear**	80x53x53	256	(80x53x53x256) + 256 = 2930944	68.22 MB	3.57 %
10	**ReLU**	256	256	0	0.00 MB	0 %
11	**Linear**	256	27	(256x27) + 27 = 6939	0.00 MB	0 %
12	**LogSoftmax**	27	27	0	0.00 MB	0 %
Total				**2970553**	**1645.24 MB**	**90.98 %**

Figure 4. Best Model (CNN) Statistics

- **Convolution layers:** These layers were used as the primary building blocks of the neural network model to extract important features from the input images. They help the model to learn and detect patterns in the image data, such as edges, corners, and shapes.

- **Maxpooling:** This technique was used to reduce the dimensionality of the feature maps generated by the convolution layers. It helps to improve the computational efficiency of the model and prevent overfitting. The maxpooling operation selects the maximum value from a small region of the feature map and discards the rest, which helps to retain the most important information.

- **ReLU activation function:** Rectified Linear Unit (ReLU) activation function was used after the convolutional and pooling layers to introduce non-linearity and make the model more expressive. ReLU returns the input value if it is positive, and zero otherwise. This activation function helps to improve the model's performance by allowing it to learn complex and non-linear relationships between the input and output.

- **Adam optimizer:** Adam optimizer was used to optimize the model parameters during the training process. It is an adaptive learning rate optimization algorithm that helps to achieve faster convergence and better generalization performance. Adam computes adaptive learning rates for each parameter based on the past gradients and updates the parameters accordingly. **Learning rate used : 3e-4**

- **Cross-entropy loss function:** Cross-entropy loss function was used as the objective function to measure the difference between the predicted and actual output labels, and adjust the model parameters accordingly during backpropagation. It is a popular loss function for classification problems that calculates the difference between the predicted and actual probability distributions of the output classes. The model aims to minimize the cross-entropy loss during training to improve its accuracy on the test data.

4.3. Model Robustness

We examined the impact of reducing the number of neurons in a fully connected layer of a neural network. By reducing the number of neurons by half, the number of trainable parameters in the network also reduces by half. However, this reduction in parameters results in a significant drop in accuracy from 128 to 32 neurons. These findings suggest that reducing the number of neurons may not be an effective optimization strategy for neural networks seeking to reduce complexity and size.

4.4. Optimisation through TorchScript

4.5. Power Usage

- The GPU power usage graph generated by the WandB tool during CNN training provides valuable information on the efficiency and performance of the training process. The graph shows the power consumption of the GPU over time and can reveal any anomalies or irregularities in the training process, such as sudden

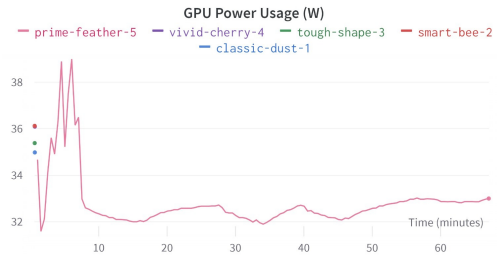

Figure 5. GPU Power Usage

spikes or drops in power usage, which could be indicative of issues such as memory leaks, unstable optimization, or hardware failures.

- Furthermore, the GPU power usage graph can provide insights into the efficiency of the training process. Ideally, the graph should show a smooth and consistent increase in power usage over time, indicating that the model is efficiently utilizing the GPU resources. In contrast, if the graph shows irregular or fluctuating power usage, it could indicate inefficiencies in the training process, such as suboptimal batch sizes, slow data loading, or inefficient optimization.

- Overall, monitoring the GPU power usage graph during the training process using WandB can help identify and troubleshoot issues, optimize the training process, and ensure the efficient use of GPU resources.

5. Optimization through Torchscript and ONNX

Model size reduction is an essential aspect of deploying machine learning models in resource-constrained environments such as mobile devices and embedded systems. Two commonly used techniques for reducing model size are TorchScript and ONNX.

5.1. Torchscript

1. Torchscript is a way to serialize PyTorch models into a format that can be loaded and executed in a variety of environments, including C++, Java, and JavaScript. Torchscript allows PyTorch models to be deployed in production environments without requiring the PyTorch framework to be installed. It also allows PyTorch models to be integrated with other programming languages and frameworks.

2. The memory usage of a PyTorch model that has been optimized using TorchScript will depend on a number of factors, including the specific architecture of the model, the size of the input data, and the hardware being used for inference. However, in general, using TorchScript to optimize a model can reduce the memory usage and increase the inference speed compared to using the original PyTorch model.

3. The accuracy of the model is reduced to 62% from 68%. The memory usage decreased from 1.6GB to 0.9GB.

5.2. ONNX

1. ONNX (Open Neural Network Exchange) is an open-source format for representing deep learning models. It allows models to be trained in one framework and then easily transferred to another framework for inference. ONNX supports a wide range of deep learning frameworks, including PyTorch, TensorFlow, and Caffe.

2. ONNX Quantized is a version of ONNX that supports quantized models. Quantization is a technique used to reduce the memory and computation requirements of deep learning models by representing weights and activations with fewer bits. ONNX Quantized allows quantized models to be easily transferred between different frameworks.

3. The accuracy of the model is reduced to 64% from 68%. The memory usage decreased from 1.6GB to 1.2GB.

6. Experimental Results

6.1. During validation

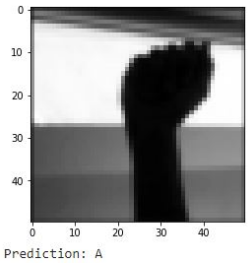

Figure 6. Prediction of label 'A'

6.2. Real-Time Results

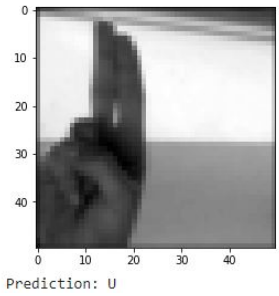

Figure 7. Prediction of label 'U'

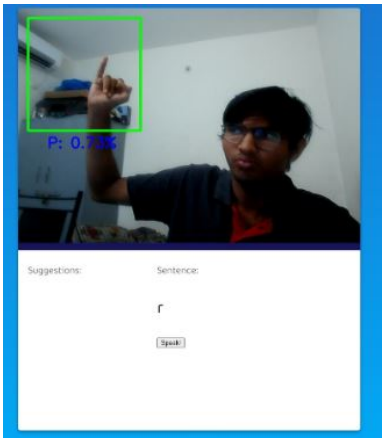

Figure 8. Prediction of a Letter

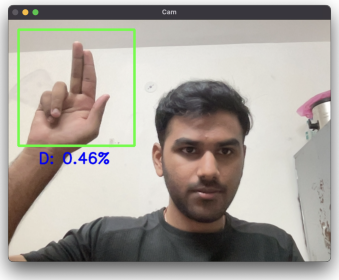

Figure 9. Letters forming a Sentence

7. Future improvements for custom SOC

7.1. Knowledge Distillation

- Knowledge distillation is a technique that transfers knowledge from a larger, complex model (teacher model) to a smaller, simpler model (student model) to improve the performance and efficiency of the student model. The teacher model is trained on a large dataset, and the student model is trained to predict the same outputs as the teacher model, guided by the teacher model's soft targets.
- The use of knowledge distillation can improve the efficiency and robustness of deep learning models, making them suitable for deployment on custom SoCs with limited processing power.
- We have not applied knowledge distillation as we are only training on CNN with few convolution layers and decreasing the parameters further does not make sense. Knowledge distillation is applicable for sequential models like RNN, high grade models like tranformers and GANs.

8. Custom Evaulation

- In our evaluation, we are interested in analyzing the performance of our classification model with respect to each individual class in our dataset. Specifically, we aim to identify instances where our model may have misclassified certain samples as belonging to a different class.
- To achieve this, we perform a comprehensive analysis of misclassifications by comparing each class in our dataset with every other set of classes. By doing so, we can identify instances where our model has incorrectly assigned samples to a class other than their true label.
- Using this information, we then calculate the inter-class precision and inter-class recall for each class in our dataset. These metrics provide us with an indication of the overall accuracy of our model in correctly identifying samples belonging to each class. By analyzing the inter-class precision and recall, we can identify specific areas of our model that may require further optimization or adjustment to improve its performance in accurately classifying samples.

References

1.https://towardsdatascience.com/pytorch-jit-and-torchscript-c2a77bac0fff
2.https://medium.com/axinc-ai/using-the-onnx-official-optimizer-27d1c7da3531
3.https://docs.wandb.ai/