(MIS-64061-003)

**American Alphabets Sign Languages Detection**

|  |  |
| --- | --- |
| Jyothsna Peddireddigari  Pranay Kumar Kodeboyina  Ramya Krishna Singavarapu |  |
| Venkateswara Rao Jammula |  |
|  |  |

Contents

[Summary 3](#_Toc152708389)

[Introduction 4](#_Toc152708390)

[Assumptions 4](#_Toc152708391)

[Project Implementation 5](#_Toc152708392)

[Data Creation 5](#_Toc152708393)

[Data Processing and Exploration 6](#_Toc152708394)

[Data Split 6](#_Toc152708395)

[Model Building 7](#_Toc152708396)

[Convolutional Neural Network 7](#_Toc152708397)

[Convolutional Neural Network with Image Augmentation 9](#_Toc152708398)

[Pretrained Model: ResNET 11](#_Toc152708399)

[Recommendations 13](#_Toc152708400)

[Conclusion 13](#_Toc152708401)

# Summary

This project aims to develop a user-friendly tool for recognizing American Sign Language (ASL) gestures, which can improve communication accessibility for children with hearing impairment. To achieve this, the team has built convolutional neural networks (CNNs) using a carefully curated dataset and conducted thorough data processing and exploration. The CNN model was tested with and without image augmentation. Additionally, another model is built using a pre-trained model called the ResNET model. The team recommends further hyperparameter tuning, exploring additional pre-trained models, and considering real-time deployment for practical applications to get performance feedback.

*Keywords: CNN, Google Colab, Teachable Machine, Image Augmentation, ResNET*

# Introduction

According to the National Institute on Deafness and Other Communication Disorders (NIDCD), an estimated 2 to 3 out of every 1,000 children in the United States are born with detectable hearing loss. This statistic suggests that approximately 6,000 to 9,000 deaf children are born each year in the United States. Additionally, per the Centers for Disease Control and Prevention (CDC) findings, approximately 1.4 per 1,000 children in the United States experience bilateral hearing loss of 40 decibels or more, equating to an estimated 4,200 children affected by such hearing loss.

Sign language is crucial for those who are deaf or hard of hearing, enabling them to communicate and express themselves.

This project aims to create a tool that assists children with varying degrees of hearing impairment in recognizing alphabets through American Sign Language gestures. This will be achieved by leveraging convolutional neural networks for accurate and efficient gesture identification. The overarching goal is to provide a user-friendly and accessible solution that enhances communication for children with hearing impairment. The project is limited to exploring gesture recognition and sign language within the American Sign Language (ASL) context. However, anyone could consider the analysis in the project to extend the analysis to English words and other languages.

Project study aims to enhance communication accessibility and sign language recognition for the hearing impaired, with a primary focus on developing a machine learning model to accurately recognize and classify ASL gestures for each English alphabet letter. This requires creating a robust model capable of effectively interpreting diverse hand gestures.

# Assumptions

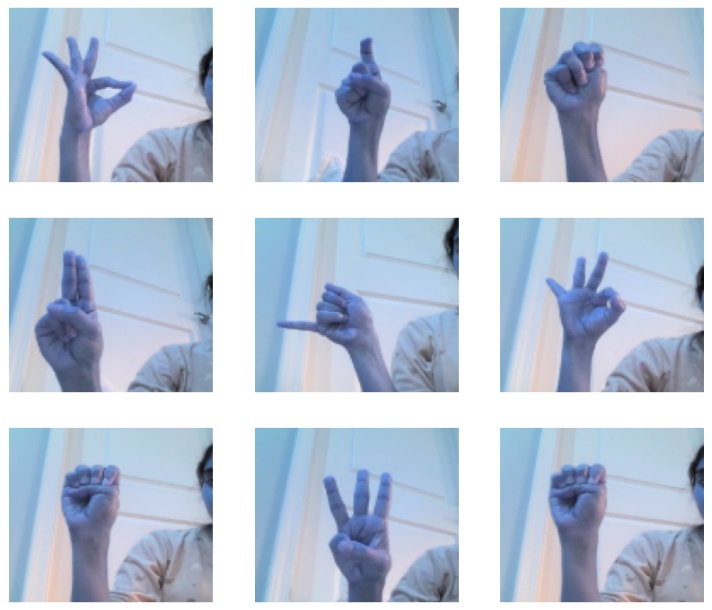
1. The training dataset is representative of the real-world scenarios the model will encounter, and it sufficiently captures the diversity of objects, backgrounds, and conditions.
2. The annotations provided in the training data are accurate, consistent, and properly align with the objects or features the model is expected to detect.

# Project Implementation

The provided code marks the initiation of an Alphabet Sign Language Detection project, where a team has harnessed various Python libraries to undertake critical preparatory steps. These steps encompass data creation, data processing, visualization, and importation of essential machine learning packages.

## Data Creation

Image snapshots used in the project are manually taken snapshots using the google platform techablemachine(https://teachablemachine.withgoogle.com/train). No augmentation is performed using the platform. Platform allowed us to capture our hand sign images from different angles when we move our hand and allows us to download them as a zip file. It is a manual process, but capturing thousands of images is easy because of the automatic captures when start recording enabled in the platform. When the number of captures reached 500, person stopped the capture and deleted extra images if captured more than the desired number. Then, created a folder for each alphabet image taken on our desktop and created the dataset.



## Data Processing and Exploration

Notably, the project is hosted on Google Collab, utilizing Google Drive for streamlined access to the dataset. The dataset itself is meticulously organized into subfolders, each corresponding to an English alphabet letter, facilitating efficient data loading. A custom function is employed to read and process the images, assigning numerical labels based on the alphabet's structure. The code then explores the dataset through visualizations, offering an initial understanding of the diverse hand gestures represented. Subsequently, the dataset is split into training and testing sets, ensuring a balanced distribution of classes. Crucially, the categorical labels undergo one-hot encoding to prepare them for multi-class model training.

The dimensions of various data variables are meticulously checked, providing crucial insights for configuring the neural network layers. This initial phase concludes by seamlessly transitioning to subsequent stages of model development, training, and evaluation. The code not only establishes a solid foundation for machine learning tasks but also emphasizes the collaborative and systematic approach of the project team.

In delving into the specific code sections, the team begins by importing an array of Python libraries, spanning from foundational numerical operations with NumPy to data visualization tools like Matplotlib and Seaborn, culminating in machine learning powerhouses TensorFlow and Keras. The integration of Google Drive is pivotal, underscoring the collaborative and accessible nature of the project. The meticulous organization of the dataset into subfolders aligned with each alphabet letter reflects a thoughtful approach to data structuring. The custom data loading function demonstrates adaptability, handling image processing intricacies such as resizing for standardization. The subsequent visual exploration of a subset of images is a critical step, enabling the team to qualitatively assess the dataset's quality and diversity.

One-hot encoding of categorical labels is a crucial transformation, preparing the data for multi-class classification tasks. The meticulous checking of variable dimensions ensures data integrity and compatibility with the envisioned neural network architecture. The code concludes by seamlessly transitioning to the subsequent phases, setting the stage for model building, training, and evaluation.

## Data Split

Moving forward, the team conducts a strategic split of the dataset into training and testing sets(70% and 30%), utilizing the stratify parameter to ensure an equitable distribution of classes in both subsets. This thoughtful data partitioning is essential for the robustness of the subsequent machine learning model.

In essence, this code snippet encapsulates the conscientious and systematic approach of the project team, starting with data pre-processing and exploration, laying a solid foundation for the subsequent implementation of advanced machine learning models for Alphabet Sign Language Detection.

## Model Building

### Convolutional Neural Network

One of the initial models in the project is a Convolutional Neural Network (CNN) that uses the Keras library. Keras is a high-level neural networks API that runs on top of TensorFlow. The purpose of this CNN is to classify images into 26 classes, with each class representing an English alphabet. Constructed the neural network with specific layers, such as convolutional layers, max-pooling layers, flattening layers, and densely connected layers. Additionally, a sequential model allows the linear stack of layers.

Convolutional layers are added to capture hierarchical features from the input images. The specified parameters include the number of filters, kernel size, and activation function (ReLU). The ReLU activation function is used to introduce non-linearity. Max-pooling layers are used to down-sample the spatial dimensions and added the Flatten layer to transform the 2D output from the convolutional layers into a 1D vector for preparing the data for input into densely connected layers.

Densely connected layers are introduced to perform classification based on the extracted features. The ReLU activation function is applied, and dropout layers with a dropout rate of 0.5 are included for regularization to prevent overfitting. The final layer is a densely connected layer with 26 neurons, representing the 26 classes in the classification task. The softmax activation function is used for multi-class classification, providing probability scores for each class.

The model is designed to handle input images of size (120, 120, 3) and to classify them into one of 26 classes. The layered architecture is chosen to balance model complexity and generalization, and dropout layers contribute to the model robustness.

The EarlyStopping callback imported from the TensorFlow/Keras library is designed to monitor a specified metric during training and halt the training process if the metric does not improve over a certain number of epochs.

Validation loss and patience metrics are used in the early stopping. The validation loss is the metric being monitored, and training will stop if this metric fails to improve. The patience parameter allows the training to continue for two additional epochs after the validation loss stops improving. Training will be stopped if there is no improvement after these two epochs.

The fit method is utilized to train the neural network on the provided training data, consisting of input features (X\_train) and categorical labels (y\_cat\_train). Three key parameters govern the training process:

Firstly, with epochs=10, the model undergoes 10 complete passes through the training dataset, allowing it to iteratively learn patterns and refine its parameters. The batch\_size=64 parameter indicates that training is performed in batches of 64 samples, contributing to computational efficiency and memory utilization. This approach enables the model to update its parameters based on subsets of the data rather than the entire dataset in each iteration.

The verbose=2 parameter controls the level of information displayed during training. A verbosity level of 2 provides detailed progress information for each epoch, including metrics such as loss and accuracy. This real-time feedback is crucial for monitoring the model's performance and identifying trends or potential issues.

We can assess the model's performance by comparing its loss, accuracy, and confusion matrix heatmap. The train and validation loss have decreased consistently after two epochs and have followed a similar trend. Similarly, the accuracy of both the train and validation sets has been increasing with each epoch and has also followed a similar trend. The confusion matrix shows the number of correctly and incorrectly classified classes.

A graph of a line

Description automatically generatedA graph of a curve

Description automatically generated with medium confidence

A graph of numbers and letters

Description automatically generated with medium confidence

### Convolutional Neural Network with Image Augmentation

The second model is the convolutional network with image augmentation. As Deep learning relies heavily on vast amounts of data for effective training, the quality and diversity of this data significantly impact model performance. In real-world scenarios, images often exhibit orientation, position, and scale variations. Training models solely on unaugment data can lead to overfitting and poor generalization to unseen variations.

This is where image data augmentation comes into play. Image data augmentation refers to techniques that artificially manipulate the training data to increase diversity by applying various transformations to the original images, such as:

*Rotation*: Randomly rotating images within a predefined range helps the model become invariant to object orientations.

*Width and Height Shifts*: Simulating horizontal and vertical shifts introduces variations in object positioning, enhancing the model's ability to generalize to different spatial arrangements.

*Shearing*: This technique involves tilting parts of the image, mimicking a shearing effect, and helps the model recognize objects from diverse perspectives.

*Zooming*: Randomly zooming in or out of the images allows the model to learn features at different scales, improving its robustness to variations in image size.

*Horizontal Flip*: Flipping images horizontally is instrumental when object interpretation is independent of its spatial orientation.

Image data augmentation can reduce overfitting, enhance robustness and improve model performance.

*Reduce overfitting*: By exposing the model to a wider range of variations, the model is less likely to memorize specific patterns in the training data, leading to improved generalization.

*Enhance robustness*: The augmented data better reflects real-world scenarios, where images rarely appear in pristine, controlled settings. This equips the model with the ability to handle noise, distortions, and other imperfections, improving its overall robustness.

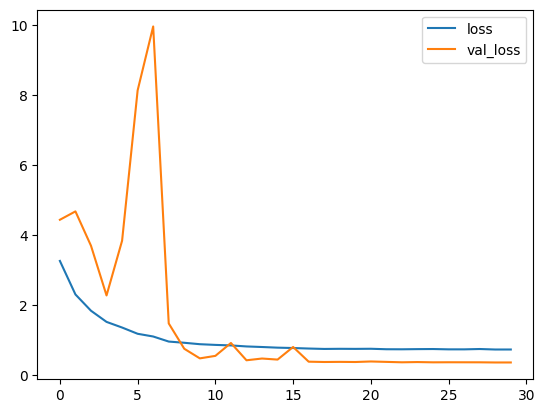
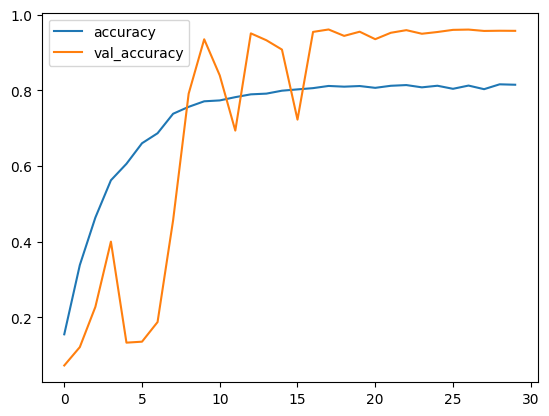
*Improve model performance*: Studies have consistently demonstrated that using image data augmentation leads to significant improvements in model accuracy and generalizability across various deep learning tasks.

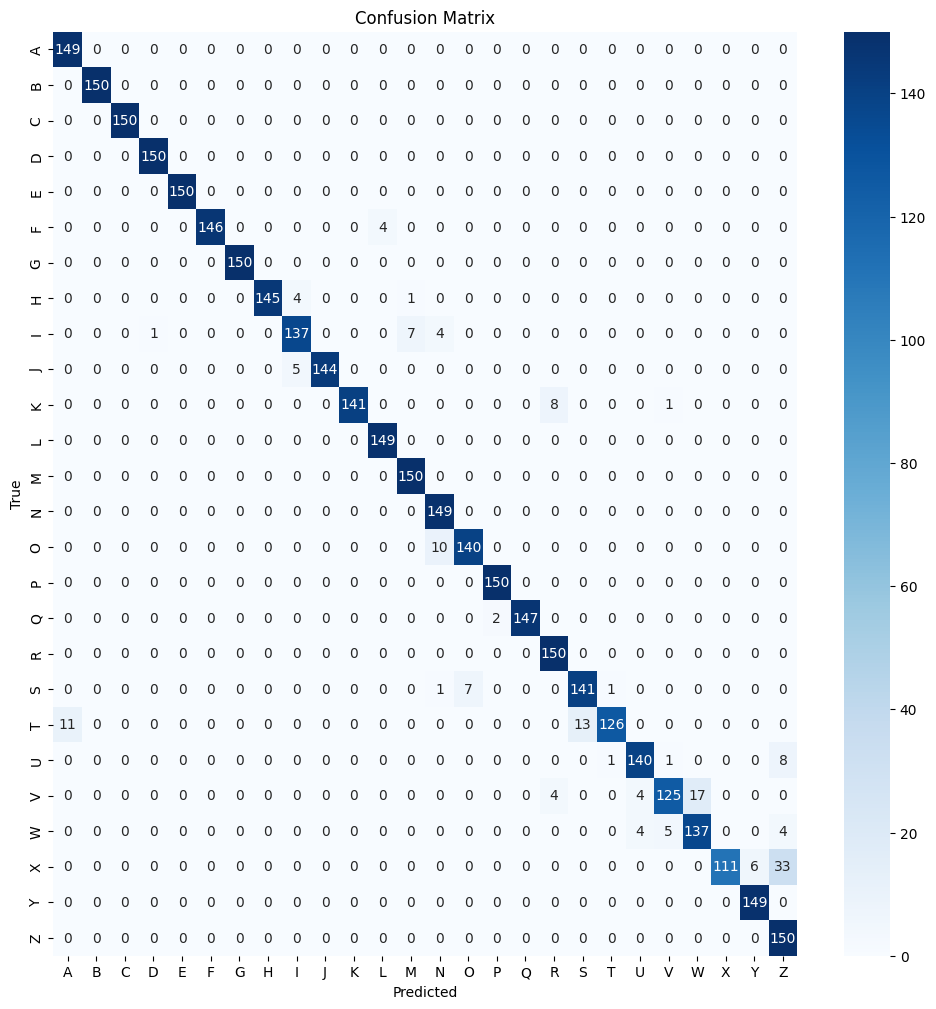
Image augmentation for the project is performed with random rotations within a 20-degree range, horizontal and vertical shifts of up to 20% of the image dimensions, shearing effects to simulate tilting, random zooming in or out by 20%, horizontal flipping for increased variation, and a fill mode of 'nearest' that replicates the values of the nearest existing pixels when creating new ones. These augmentations collectively contribute to a more varied and robust training dataset, enhancing the model's ability to generalize to different orientations, positions, and scales of images, ultimately improving overall model performance. Using data augmentation in this project led to confusion among similar-looking alphabet signs. As a result, the model's performance with data augmentation was not as good as the other two models.

In addition to the setting used in the first model, Model 2 introduces additional techniques like L2 regularization, batch normalization, and dropout for improved generalization and training stability. These techniques are particularly useful when dealing with deeper networks and complex datasets.

In summary, Model 2 is more sophisticated in terms of regularization and normalization techniques, potentially offering better performance, especially in scenarios with a large amount of data and deeper network architectures. However, the effectiveness of these techniques may depend on the specific characteristics of the dataset and the problem being addressed.

Let us evaluate the model's performance by comparing its loss, accuracy, and confusion matrix heatmap. During the initial few epochs, the training loss was lower than the validation loss but stabilized after nine epochs. On the other hand, the validation error increased at five epochs and then decreased, becoming less than the training error and stabilizing around 15 epochs. The accuracy of the training dataset gradually increased with the number of epochs and stabilized after 12 epochs, while the accuracy of the validation dataset fluctuated until 15 epochs and stabilized after that. The confusion matrix shows the number of correctly and incorrectly classified classes.



### Pretrained Model: ResNET

The experimental evaluation involved the implementation of various pre-trained models, including ResNet50 and ResNet101V2, as feature extractors. Notably, the ResNet101V2 model emerged as the most effective choice, as evidenced by its superior performance. The model was loaded without its top (fully connected) layers, and a bespoke architecture was constructed atop it. To preserve the knowledge acquired during pre-training, the base model's weights were frozen, preventing updates during subsequent training phases.

The constructed Sequential model, denoted as model2, is structured with the ResNet101V2 base model, followed by a flattening layer to transform the output into a 1D tensor. Subsequent to flattening, two densely connected layers were introduced, incorporating non-linearity through the application of the Rectified Linear Unit (ReLU) activation function in the initial dense layer. The final layer utilized a Softmax activation function, tailored for the multi-class classification task involving 26 distinct classes.

The detailed model summary provides comprehensive insights into the architecture, encompassing layer types, output shapes, and the total number of parameters involved.

A screenshot of a computer program

Description automatically generated

This architectural design capitalizes on the ResNet101V2 model's robust feature extraction capabilities, derived from pre-training on a substantial image dataset. By customizing the model with additional layers, the network was fine-tuned for the specific task of sign language detection with 26 distinct classes. The decision to freeze the base model's layers was strategic, aiming to preserve the valuable knowledge acquired during pre-training, while the inclusion of custom layers facilitated adaptability to project-specific requirements, particularly the 26-class image classification task applied to input images sized at (120, 120, 3).

Analyzing the model's performance progression across epochs revealed consistent improvement, culminating in an outstanding outcome. The model demonstrated exceptional efficacy on test data, achieving a accuracy rate of 100%.

A table of numbers with numbers and symbols

Description automatically generated with medium confidence

However, Pretrained model seems to perform well but there is a higher chance of overfitting and biased which could not guarantee the generalization on unseen data.

# Recommendations

1. Experiment with Hyperparameters: Fine-tune hyperparameters such as learning rates, dropout rates, and layer configurations to optimize model performance.

2. Transfer Learning: Explore additional pre-trained models (e.g., from TensorFlow Hub) for transfer learning to leverage knowledge from larger datasets.

3. Model Deployment: Consider deploying the trained model as part of a real-time sign language recognition application, possibly through web or mobile interfaces.

# Conclusion

|  |  |  |
| --- | --- | --- |
| **Model Name** | **Train Accuracy (%)** | **Test Accuracy (%)** |
| Convolutional Neural Networks | 98.6 | 99.97 |
| Convolutional Neural Networks with Image Augmentation | 81.5 | 95.7 |
| Pretrained model – ResNETV10 | 100 | 99.98 |

After analyzing the performance metrics of the three models, it is clear that all models have intense training and validation accuracies, indicating their ability to learn and generalize from the provided data. The Convolutional Neural Network (CNN) achieves an impressive 99% validation accuracy, demonstrating the effectiveness of the base architecture in recognizing American Sign Language gestures. CNN with image augmentation slightly lags in both training and validation accuracy, using data augmentation led to confusion among similar-looking alphabet signs. As a result, the model's performance was not as good as the other two models compared to their performance. Notably, the pre-trained ResNetV10 model achieves perfect training accuracy and an impressive 99% validation accuracy. However, it is essential to exercise caution to avoid potential overfitting. Overall, the findings highlight the effectiveness of convolutional neural networks for sign language recognition. Further fine-tuning and exploring hyperparameters could enhance the model's performance and robustness.

In summary, the project demonstrates a solid understanding of building and training CNN models for image classification tasks, with room for further exploration and enhancement.

The project code effectively implements a sign language recognition system using CNNs. The two models provide options for varying levels of complexity, and the use of data augmentation enhances the model's robustness.

***References:***

[***https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing***](https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing)

[***https://arxiv.org/abs/1603.05027***](https://arxiv.org/abs/1603.05027)