GALALA UNIVERSITY
Powered by Arizona State University

Neural Network Report

Hand Gestures Recognition (ASL)

➢ ***Names:***

1. Ahmed Marwan Salman 222100171
2. Mohamed Yasser 222101451
3. Shehab Essam Hassan 222100828
4. Mostafa Karam Yehia 222101620
5. Wael Ahmed El-Zuhairy 222101888

➢ ***Table of Contents:***

- ***Introduction***

A key aspect of human contact is communication, and language is essential for developing mutual understanding. American Sign Language (ASL) is an essential communication tool for the deaf and hard-of-hearing communities.

The objective of this project is to create a strong and effective deep learning model that can identify ASL alphabet actions from images. The tool aims to improve accessibility and communication for people with speech or hearing difficulties by automating the process of converting hand gestures into corresponding alphabetical characters by using advanced artificial intelligence and deep learning algorithms.

This project is based on the ASL alphabet collection, which includes an extensive range of hand gesture images. This project aims to achieve high accuracy features through systematic preprocessing, model training, and evaluation

- ***System Design***
  - ***Dataset And Code wise***

The system for ASL alphabet recognition is designed as a pipeline, consisting of the following key components:

  - Data Preprocessing:

1.Dataset Loading:

- The dataset is stored in folders named after the ASL labels (e.g., A, B).

- The script constructs a list of all file paths (list_path) and their corresponding labels (list_labels) by iterating over the folder structure.

This creates a DataFrame (metadata) that maps each image to its respective label.

2. Data Splitting:

- The dataset is split into training, validation, and testing sets:

- **Training (70%):** For training the model.

- **Validation (15%):** For tuning hyperparameters and checking overfitting.

- **Testing (15%):** For evaluating the model on unseen data.

- The splits ensure class balance using stratify to maintain equal representation across classes.

3.Image Rescaling:

- Images are rescaled to a pixel range of [0, 1] from the default [0, 255]. This normalization speeds up convergence during training.

4. Data Generators:

- The ImageDataGenerator object generates batches of data for training, validation, and testing.

- The generators preprocess images to the required size (64x64) and format

For validation and testing, a separate ImageDataGenerator is used without augmentation but with rescaling.

6. One-Hot Encoding Labels:

- Labels are converted to one-hot encoded vectors since the model predicts probabilities for multiple classes

- Model Architecture

The recognition model is built using a convolutional neural network (CNN) framework, chosen for its proven effectiveness in image classification tasks. The architecture consists of:

Input Layer: Processes pre-processed images as input.

Convolutional Layers: Extracts spatial features through learned filters

Pooling Layers: Reduces spatial dimensions, mitigating overfitting and computational complexity.

Fully Connected Layers: Maps extracted features to the output classes (alphabet letters).

Output Layer: Utilizes a softmax activation function to predict the probability distribution over 26 alphabet classes.

Hyperparameters such as learning rate, batch size, and the number of epochs is configured for optimal performance.

- Model Training and Optimization

The dataset is split into training, validation, and test sets to ensure a reliable evaluation. The training process involves:

Loss Function: Categorical cross-entropy is used to measure the divergence between predicted and true labels.

Optimizer: The Adam optimizer is selected for its adaptability and efficiency in converging to the optimal solution.

Regularization: Techniques like dropout are employed to prevent overfitting.

- Evaluation and Validation

Model performance is assessed using metrics such as accuracy, Visualization tools, including confusion matrices and learning curves, provide insights into the model's strengths and areas for improvement.

- **Hardware Components:**

- ***ESP-32 CAM Module:***

The ESP-32 CAM module will serve as the core hardware for the system. It is a low-cost, low power microcontroller with built-in Wi-Fi and Bluetooth capabilities, coupled with a camera for capturing real-time images of hand gestures.

The ESP-32 CAM is chosen for its compact size, affordability, and ability to run TensorFlow Lite models efficiently on edge devices. The camera will capture images or video frames, which will be processed locally by the ESP-32

- ***Power Supply:***

A suitable power supply such as a rechargeable battery

- ***Programming The ESP-32 CAM Module***

We used Arduino Uno to program the CAM from the TensorFlow Lite model that we made

- ***LED To Indicate the letters***

A Green Colour and B Red Colour

  - **Software Components:**

- **TensorFlow Lite Model:**

The core of the ASL recognition system is a CNN-based model, trained to recognize ASL gestures from image data. TensorFlow Lite is chosen for its ability to run machine learning models on edge devices with limited resources, such as the ESP-32 CAM.

The model will be optimized for real-time inference and will be deployed on the ESP-32 CAM for offline processing.

Training the Model:

A Convolutional Neural Network (CNN) will be used to classify ASL hand gestures. The model will be trained on a large dataset of ASL images, where each image corresponds to a specific gesture or letter. Techniques such as data augmentation and transfer learning may be applied to improve the model's accuracy and generalization.

Model Conversion to TensorFlow Lite:

Once the CNN model is trained, it will be converted into TensorFlow Lite format to make it compatible with the ESP-32 CAM.

- ***System Pipeline:***

The overall workflow is as follows:

1. Image Capture: The ESP-32 CAM captures images or video frames of the user's hand gestures.
2. Preprocessing: The captured images are pre-processed (e.g., resizing, normalization) to match the input requirements of the TensorFlow Lite model.

3. Gesture Recognition: The pre-processed image is passed through the trained TensorFlow Lite model running on the ESP-32 CAM, which performs the classification and identifies the ASL gesture.
4. Output Display: The recognized gesture is displayed on a connected display, such as an LCD screen, or used as input for further actions (e.g., translating the gesture into text or speech).

- ***Implementation***

The code will be provided in the discussion in an outer handout

- ***Problems***

We encountered several challenges during the development of this project.

First, incorporating the entire alphabet significantly increased the size of the model, exceeding the capabilities of the ESP32-CAM. To address this limitation, we reduced the scope to include only three letters (A, and B), ensuring compatibility with our hardware.

Second, we observed that the model struggled to detect hand gestures in the presence of busy backgrounds. To improve its performance, we retrained the model using RGB images with complex backgrounds to enhance its accuracy under such conditions.

Third, we used the edge impulse to encounter the model problem to be uploaded in the ESP-32CAM and to be compatible with it.

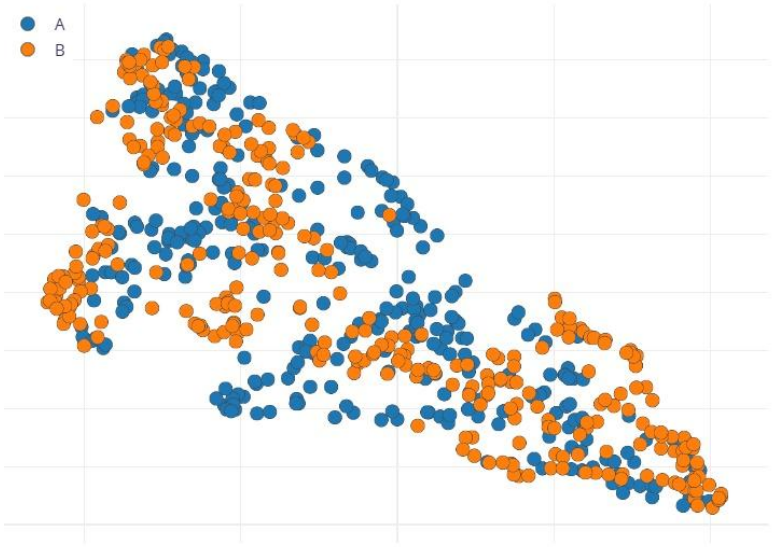
Lastly, while we initially intended to display the results on a liquid crystal display (LCD), budget constraints led us to opt for LEDs as an alternative for indicating predictions. This approach provided a cost-effective solution while meeting the project's requirements

- ***Results***

Model Architecture:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 75)	750
batch_normalization (BatchNormalization)	(None, 28, 28, 75)	300
max_pooling2d (MaxPooling2D)	(None, 14, 14, 75)	0
conv2d_1 (Conv2D)	(None, 14, 14, 50)	33,800
dropout (Dropout)	(None, 14, 14, 50)	0
batch_normalization_1 (BatchNormalization)	(None, 14, 14, 50)	200
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 50)	0
conv2d_2 (Conv2D)	(None, 7, 7, 25)	11,275
batch_normalization_2 (BatchNormalization)	(None, 7, 7, 25)	100
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 25)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 512)	205,312
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 24)	12,312

Dataset Distribution:

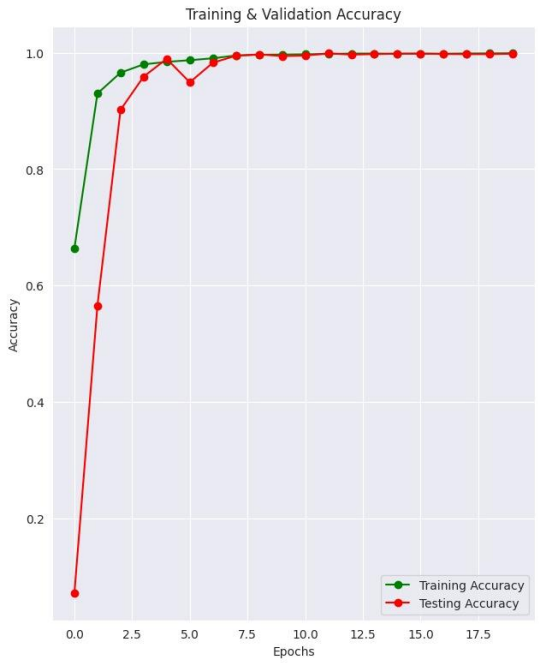


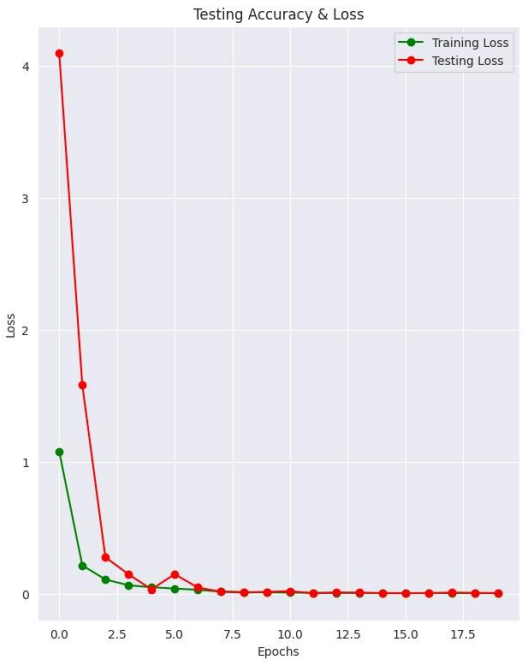
Confusion Matrix:

Confusion matrix (validation set)

	BACKGROUND	A	B
BACKGROUND	**100%**	0%	0%
A	1.7%	**98.3%**	0%
B	0%	0%	**100%**
F1 SCORE	1.00	0.99	1.00

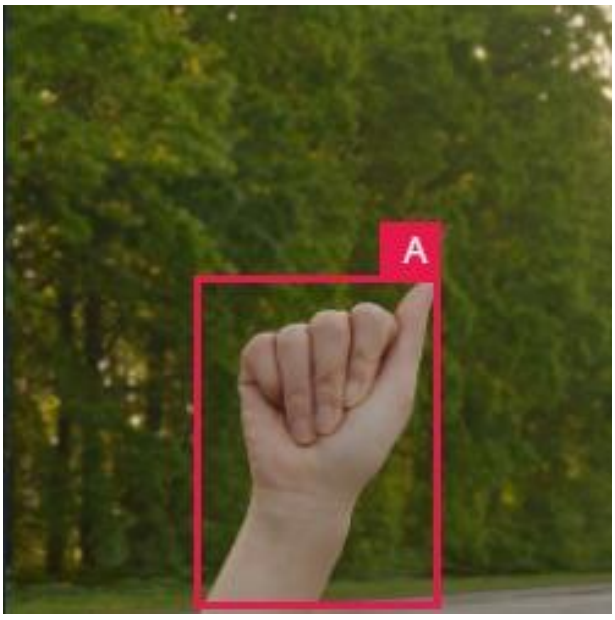
Training Loss and Metrics:

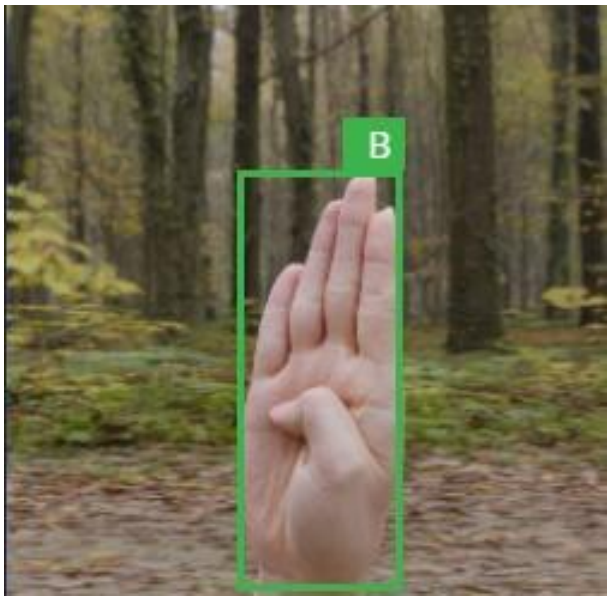




Final Test Accuracy:

Test Results:





Hardware Final Product:

Conclusion

The project successfully demonstrates the development of a deep learning-based system for recognizing American Sign Language (ASL) alphabets. By employing a Convolutional Neural Network (CNN) architecture, optimized with techniques like data augmentation and dropout, the model achieved notable accuracy in classifying ASL gestures by the LEDs

This innovation enhances communication accessibility for deaf and hard-of-hearing communities. Future improvements could include expanding the dataset to dynamic signs and improve our hardware.

- ***References***

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

2. Chollet, F. (2017). Deep Learning with Python. Manning Publications.

3. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image re cognition. arXiv preprint arXiv:1409.1556.

4. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

5. https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet

6.Fritz AI. (n.d.). *Exploring sign language recognition techniques with machine learning*. Retrieved from https://fritz.ai/sign-language-recognition-techniques-with-machine-learning/

7.Stanford University. (2024). *Sign language recognition with convolutional neural networks*. Retrieved from https://cs231n.stanford.edu/2024/papers/sign-language-recognition-with-convolutional-neural-networks.pdf

8. https://studio.edgeimpulse.com/public/606170/live