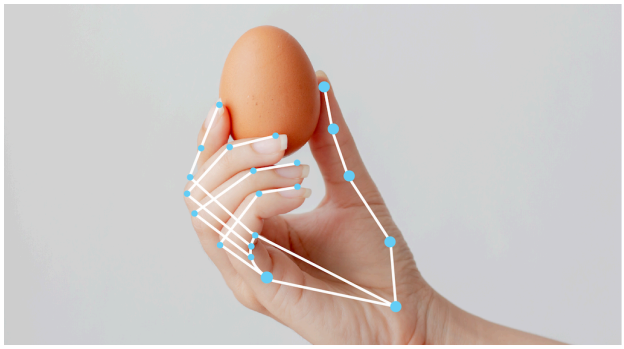
“LISten AId”

Michele Fontana (1905755),
Sofia La Rosa (1885756)

1. Introduction

Hand gesture recognition is an emerging field of research within artificial intelligence and computer vision. Leveraging this technology has the potential to significantly improve communication with others for deaf or hard-of-hearing individuals, allowing them to more easily interact with their surroundings. The use of AI systems for gesture recognition offers an intuitive and natural way to facilitate human-machine interaction, with applications ranging from home automation to the control of wearable medical devices, and more generally for accessibility for users with disabilities, which is what motivated us to undertake this project.

In recent years, numerous studies have emerged to develop algorithms capable of recognizing hand gestures in real time using computer vision and machine learning techniques. Systems like those proposed by Google with the Mediapipe project and by Microsoft with Kinect have shown how the use of advanced sensors and powerful algorithms can translate hand movements into commands interpretable by a computer.



Guide to Hand Landmark Detection with Mediapipe

Despite these advances, however, hand gesture recognition in the context of sign language still represents a significant challenge. The complexities arise not only from the variety of gestures but also from individual differences in how these gestures are performed:

Moreover, most of the existing solutions mainly focus on American Sign Language (ASL), leaving a significant gap for our fellow Italian citizens and LIS. In fact, the only noteworthy project appears to be the one developed by QuestIT, which even made it to Forbes.

This is where our project fits in, with the goal of developing a small AI system capable of recognizing the letters of the LIS alphabet that are executed with *static* hand gestures using a simple webcam. This system should be able to identify the gestures in real time and display the corresponding letter on the screen.

To achieve this goal, we first tackled the not-so-simple task of learning the Italian Sign Language alphabet. Then, we applied the computer vision knowledge acquired during the AI Lab course, implementing and training various models on a dataset of images representing the gestures of all the letters of the LIS alphabet.

In the following sections, we will describe in more detail the method, the datasets, and the models implemented, as well as the results obtained, and especially the *challenges* encountered during the development of this project.

2. Methodology

This project involves training a *Convolutional Neural Network* (CNN) for the classification of images into 22 distinct classes representing letters of the alphabet (excluding G, S, Z, and J because those need movement to be represented).

The dataset is divided into training and testing datasets, and the model's performance is evaluated using accuracy metrics and confusion matrix visualization.

2.1. Model

2.1.1. Libraries and Tools

The project utilizes the following Python libraries and tools:

- ★ PyTorch: For building and training the neural network.
- ★ Torchvision: For dataset handling and image transformations.
- ★ Scikit-learn: For computing the confusion matrix.
- ★ Matplotlib: For plotting the confusion matrix and learning curve.

2.1.2. Data Preparation

- ★ **Image Transformations**: Images are resized to 64x64 pixels, randomly flipped horizontally, converted to tensors, and normalized with mean and standard deviations using `ImageFolder` and are then passed to DataLoader to facilitate batch processing.
- ★ **Dataset and DataLoader**: The training and test datasets are loaded from respective directories.

2.1.3. Model Architecture

The CNN model consists of three convolutional layers, each followed by ReLU activation and max pooling, and two fully connected layers. The final layer outputs probabilities for the 22 classes.

2.1.4. Training Process

- ★ **Loss Function and Optimizer:** The Cross-Entropy Loss function is used for multi-class classification, and the AdamW optimizer is used with a learning rate of 0.001.
- ★ **Training Loop**: The model is trained for a specified number of epochs. For each epoch, the training data is iteratively passed through the model, the loss is computed, and backpropagation is performed to update the model parameters. Training loss is recorded for each epoch.

2.1.5. Evaluation

- ★ **Accuracy Calculation**: After training, the model is evaluated on the test dataset to compute accuracy.
- ★ **Confusion Matrix**: A confusion matrix is plotted to visualize the performance of the model across different classes.
- ★ **Learning Curve**: The learning curve is plotted to show the training loss over epochs.

2.1.6. Annotations

The model was initially trained on **Kaggle** using a dataset of sign language images. The training involved the following steps

Initially, we used **ResNet50** for the CNN model architecture. However, it did not perform as expected, leading us to define a custom CNN architecture with the following layers:

- ★ **Convolutional Layers**: Three convolutional layers with ReLU activation and max-pooling.
- ★ **Fully Connected Layers**: Two fully connected layers, the final one with 22 output nodes corresponding to the 22 classes.

2.2. Implementation

2.2.1. Libraries and Tools

- ★ **PyTorch**: For defining, training, and running the neural network model.
- ★ **Torchvision**: For handling image transformations.
- ★ **OpenCV**: For capturing video from the webcam and processing images.
- ★ **PIL (Pillow)**: For handling image data and transformations.

2.2.2. Loading the Pre-trained Model

The model is loaded from a saved state dictionary.

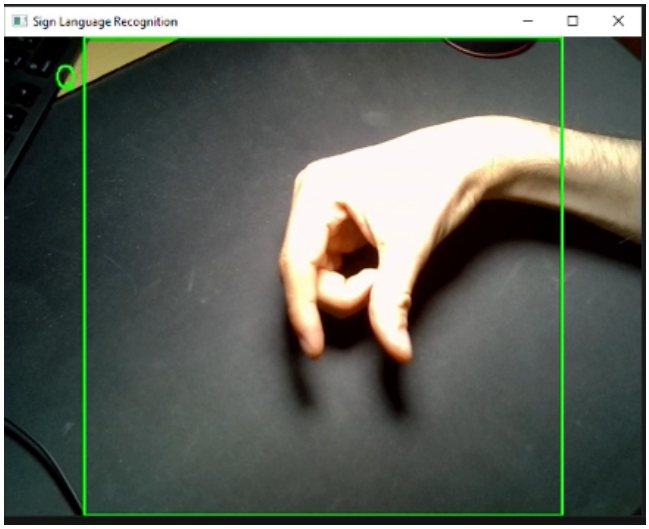
2.2.3. Image Preprocessing

Images are preprocessed using the following transformations:

- ★ **Image Transformations**: Images are resized to 64x64 pixels, randomly flipped horizontally, converted to tensors.
- ★ **The 'preprocess_image' function** is designed to preprocess individual frames captured from a webcam so that they can be fed into the Convolutional Neural Network (CNN) The function performs several steps to convert the raw image data into a format suitable for the model. It converts the color format (from BGR to RGB), converts the NumPy array to a PIL image, applies the image transformations displayed above, and lastly adds a batch dimension using the 'unsqueeze' method.

2.2.4. Real-Time Webcam Recognition

The system captures a video from the webcam, preprocesses frames, and performs inference to display the predicted sign language letter.



3. Results

3.1. Issues Identified

The model exhibits certain issues in terms of misclassifications, which can be attributed to several factors:

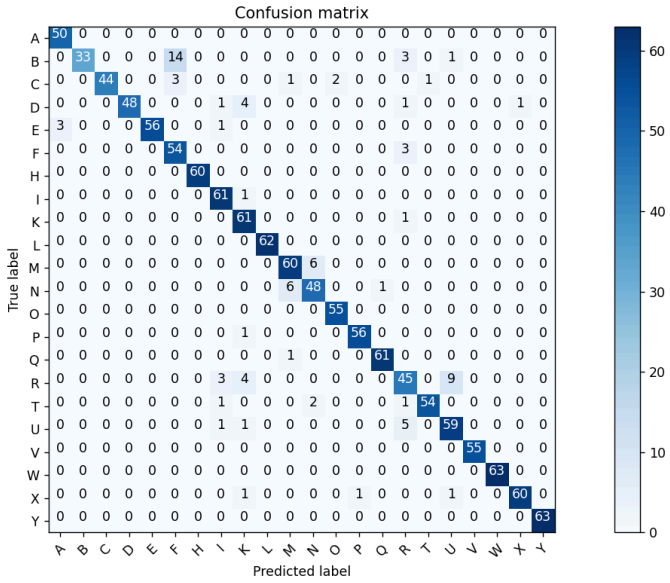
- ★ **Insufficient Dataset Size:** the dataset used for training and testing might not have been big enough, limiting the model's ability to learn diverse features.
- ★ **Specific Lighting and Background Conditions:** the images in the dataset were captured under specific lighting and background conditions. This caused the model to perform well only under similar conditions and struggle with variations in lighting or backgrounds during testing.

3.2. Confusion Matrix Analysis

The confusion matrix displays the performance of the model on the test dataset. Each cell in the matrix shows the number of instances of true labels versus predicted labels for each class.

Key observations:

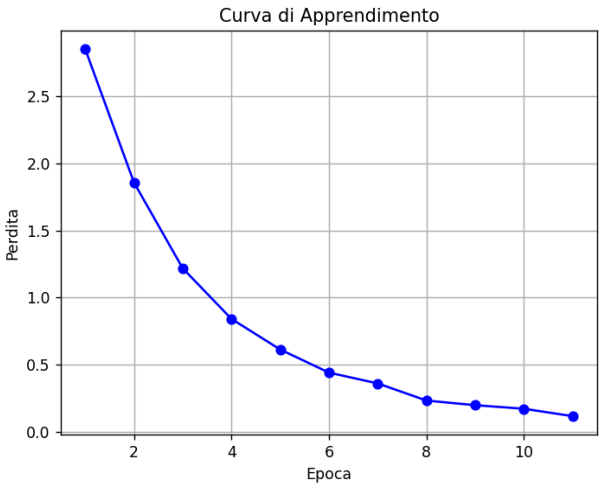
- ★ **Diagonal Dominance:** The majority of the values are concentrated along the diagonal, indicating correct predictions. This shows that the model correctly identifies most of the images for each class.
- ★ **Misclassifications:** Some off-diagonal values are present, indicating misclassifications.



3.3. Learning Curve Analysis

The learning curve shows the loss of the model over 11 epochs. Key observations from the learning curve are:

- ★ **Initial Loss:** The loss starts at a high value of approximately 2.7.
- ★ **Progression:** There is a steep decline in the loss during the first few epochs, indicating that the model is learning rapidly.
- ★ **Convergence:** The loss continues to decrease gradually, reaching a value close to 0.0 by the 11th epoch. This suggests that the model has effectively minimized the training loss.



3.4. Conclusions

The model shows good performance in terms of reducing training loss, as evidenced by the learning curve. However, the confusion matrix highlights certain misclassifications that suggest areas for improvement. To enhance the model's performance, the following steps are recommended:

- ★ **Expand the Dataset:** collect more images to cover a wider range of variations in hand gestures, lighting conditions, and backgrounds. This will help the model generalize better to new data.
- ★ **Data Augmentation:** apply more data augmentation techniques such as varying brightness, contrast, and adding background clutter to make the model robust to different conditions.

We chose this project because we strongly believe in the importance of accessibility and inclusivity. In an increasingly technological world, innovations must be designed for everyone, including those who face daily communication challenges. Our goal was to create a solution that could make a real impact in the lives of deaf or hard-of-hearing individuals, allowing them to communicate more easily and feel more included in society.

Throughout the development of this project, we encountered numerous technical and methodological challenges, but our determination to achieve something meaningful pushed us to overcome them.

This project is not just an academic exercise for us; it is a commitment to a more equitable and inclusive society. We hope that our work can inspire others to follow the same path, contributing to a future where technology is truly accessible to everyone.

3.5. Links

- ★ Dataset