# SIGN LANGUAGE RECOGNITION USING CNN

PATTERN RECOGNITION PROJECT

**Azzahra Adissa Sharya Darsono**
22/492185/PA/21085

**Khalisha Fadiya Khansa**
22/496155/PA/21313

# INTRODUCTION

The topic that we've chosen to cover for our Pattern Recognition final project is a Real-Time Sign Detection application, which will include the importance of 4 Sustainable Development Goals. The SDGs in question are as follows :

- SDG 10 (Reduced Inequalities)
- SDG 4 (Quality Education)
- SDG 9 (Industry, Innovation, and Infrastructure)
- SDG 11 (Sustainable Cities and Communities)

# MOTIVATION

- Although communication is a fundamental human right, it is still hindered by the lack of accessible communication tools for millions of people globally.
- According to *WHO*, over 430 million people require rehabilitation to address their hearing loss disability, including 34 million children.
- By 2050, over 700 million people, or 1 in every 10 people, will have disabling hearing loss.
- This underscores a significant portion of our society that is often overlooked in digital advancements in today's world.

*World Health Organization

# MOTIVATION

INCLUSIVITY

- Our project is mainly motivated by the vision of a world where everyone has equal opportunities to communicate and learn -- We must be inclusive.
- By using pattern recognition methods, we aim to create a program that recognizes sign language, more specifically *ASL*.
- We hope a program like ours not only promotes inclusivity, but also contributes to a number of the previously mentioned SDGs.

*American Sign Language

# CNN

CNN, or Convolutional Neural Network, is a type of neural network widely used in pattern recognition projects. Its structure is somewhat similar to a recurrent neural network, but it incorporates the concept of filters. Simply put, filters extract essential features from the dataset. For instance, in image classification, crucial features might include vertical and horizontal edges within the image.

# PATTERN RECOGNITION METHOD

- Data Collection
- Preprocessing
- Feature Extraction
- Model Building
- Model Training
- Evaluation

# 01 Data Collection

## ASL Alphabet

Image data set for alphabets in the American Sign Language

**The data set is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes. The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.**

# 02 Data Preprocessing

- **Creating our training data and printing the number of samples to be used for our model**

```
def create_training_data():
    training_data = []
    for category in categories:
        path = os.path.join(data_dir, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_array = cv2.resize(img_array, (img_size, img_size))
                training_data.append([resized_array, class_num])
            except Exception as e:
                pass
    return training_data

training_data = create_training_data()
print(f"Number of samples: {len(training_data)}")
```

```
Number of samples: 87000
```

- Reading images from each category directory.
- Converting them to grayscale.
- Resizing them to a uniform size.
- Appending the processed images and their corresponding labels to a list.
- Handling any potential errors during image processing.

# 02 Data Preprocessing

- Shuffling and splitting the data

```
import random
random.shuffle(training_data)

X = []
y = []

for features, label in training_data:
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, img_size, img_size, 1) / 255.0
y = to_categorical(y, num_classes=len(categories))
```

- **random.shuffle(training_data) ensures that the dataset is randomly ordered, preventing any bias from the original ordering.**
- **Loop iterates through the shuffled training_data, separating the features (images) and labels (class indices) into two lists, X and y.**
- **The list X is converted into a NumPy array and reshaped to match the expected input shape for the CNN. The pixel values are normalized to the range [0, 1].**
- **The labels y are transformed into a one-hot encoded format, making them suitable for use in multi-class classification.**
- **This preparation step is crucial to ensure that the data is in the correct format and properly randomized, enabling the neural network to learn effectively from the training data.**

# 03 Feature Extraction - CNN

# CNN

convolutional neural network

**The convolutional and pooling layers extract hierarchical features from the image, starting from low-level features like edges and textures to high-level features like shapes and specific patterns unique to each gesture.**

# 04 Model Building

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()

# First convolutional layer
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(img_size, img_size, 1)))
model.add(MaxPooling2D((2, 2)))

# Second convolutional layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten and dense layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(len(categories), activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

# 05 Model Training

Model training involves feeding the training data into the model, defining the loss function, optimizer, and metrics, and iteratively adjusting the model parameters to minimize the loss.

- Loss Function: The model is trained using a loss function, typically sparse categorical cross-entropy for multi-class classification tasks. This function measures the difference between the predicted probabilities and the actual labels.
- Optimization: Adam optimizer adjusts the weights of the network to minimize the loss function.

Training was conducted over 10 epochs with early stopping and learning rate reduction callbacks to enhance convergence and avoid overfitting.

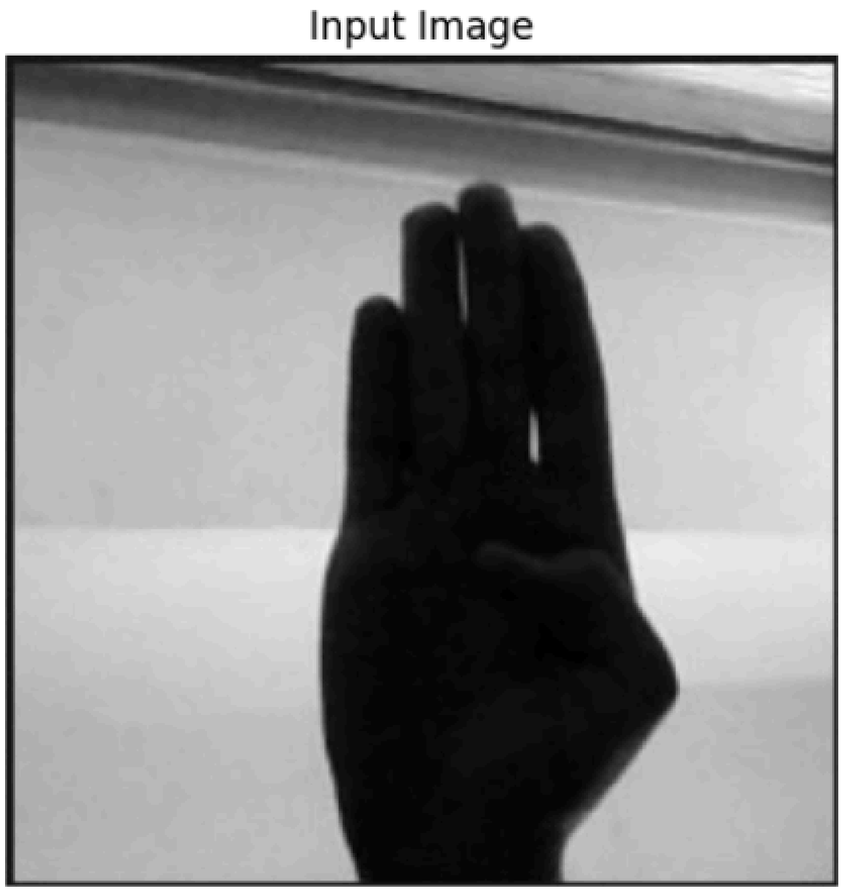
06

# Evaluation - Classification Report

```
2719/2719 [==============================] - 141s 52ms/step - loss: 0.0093 - accuracy: 0.9975
Test accuracy: 0.9974597692489624
```

# 07 Results and Discussion - Input Test Image B

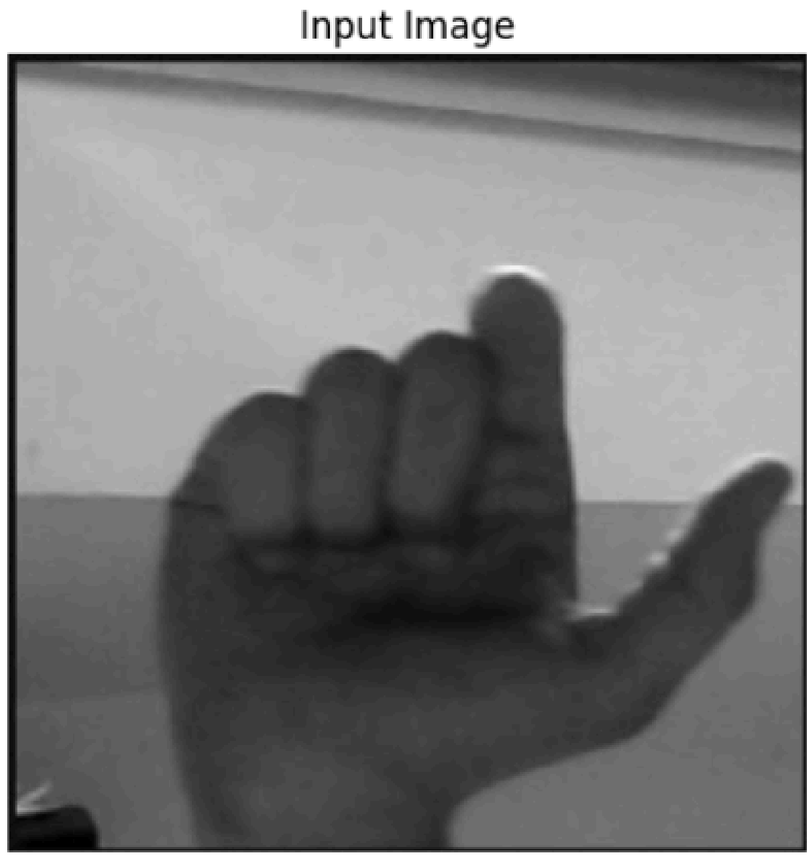
```
predict_and_display('/kaggle/input/asl-alphabet/asl_alphabet_test/asl_alphabet_test/B_test.jpg')
```




```
1/1 [==============================] - 0s 28ms/step
Predicted Sign: B
```

# 07 Results and Discussion - Input Test Image T

```
predict_and_display('/kaggle/input/asl-alphabet/asl_alphabet_test/asl_alphabet_test/T_test.jpg')
```




```
1/1 [==============================] - 0s 35ms/step
Predicted Sign: T
```

# 07 Results and Discussion - Input Test Image S

```
predict_and_display('/kaggle/input/asl-alphabet/asl_alphabet_test/asl_alphabet_test/S_test.jpg')
```


```
1/1 [==============================] - 0s 31ms/step
Predicted Sign: S
```

# 07 Results and Discussion

The high accuracy of the model suggests that the CNN effectively learned the distinguishing features of each ASL sign. The use of multiple convolutional layers, dropout layers, and batch normalization contributed significantly to the model's performance. However, the near-perfect scores indicate potential overfitting or insufficient variability in the test set. There are a few factors that improved the accuracy, such as the amount of data we trained. Tes

## Limitations

- It took forever (1 hour) to train the model.
- The input can only be an image.
- This model is only lmited to detecting American Sign Language, so other sign languages (e.g. Bisindo) cant be detected using this model.
- The model only detects letters not numbers.

# CONCLUSION

Our project succeeded in employing CNN in recognizing sign languages, indicated by the high accuracy percentage and the demonstration. However, there are a few limitations. For future work, we can modify so that the input can be videos or even a real-time camera, integrate it to applications for the deaf and mute community, and more data to the dataset for words to form sentences in ASL.

## Recommendations

- Alter the program so that we can have videos as input, and implement real-time detection
- Add a dataset on numbers and train the model with it to recognize numbers as well