Google - 🤟 American Sign Language Fingerspelling Recognition 🖖

Train fast and accurate American Sign Language fingerspelling recognition models

(ಠ_ಠ) Overview

◯ The goal of the competition is to detect and translate American Sign Language (ASL) fingerspelling into text.

◯ The competition is based on a large dataset of over three million fingerspelled characters produced by over 100 Deaf signers captured via the selfie camera of a smartphone with a variety of backgrounds and lighting conditions.

◯ The competition aims to improve sign language recognition technology, making it more accessible for the Deaf and Hard of Hearing community.

◯ Fingerspelling is an important part of ASL and is often used for communicating names, addresses, phone numbers, and other information commonly entered on a mobile phone.

◯ ASL fingerspelling can be substantially faster than typing on a smartphone's virtual keyboard.

◯ Sign language recognition AI for text entry lags far behind voice-to-text or even gesture-based typing, as robust datasets didn't previously exist.

◯ Participating in this competition could help provide Deaf and Hard of Hearing users with the option to fingerspell words instead of using a keyboard, enabling them to communicate with hearing non-signers more quickly and smoothly.

Table of contents

0. Import all dependencies

```python
# import the desired packages
import numpy as np
import pandas as pd
import json
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "simple_white"


def map_new_to_old_style(sequence):
    types = []
    landmark_indexes = []
    for column in list(sequence.columns)[1:544]:
        parts = column.split("_")
        if len(parts) == 4:
            types.append(parts[1] + "_" + parts[2])
        else:
            types.append(parts[1])

        landmark_indexes.append(int(parts[-1]))

    data = {
        "frame": [],
        "type": [],
        "landmark_index": [],
        "x": [],
        "y": [],
        "z": []
    }
```

```
        for index, row in sequence.iterrows():
            data["frame"] += [int(row.frame)]*543
            data["type"] += types
            data["landmark_index"] += landmark_indexes

            for _type, landmark_index in zip(types, landmark_indexes):
                data["x"].append(row[f"x_{_type}_{landmark_index}"])
                data["y"].append(row[f"y_{_type}_{landmark_index}"])
                data["z"].append(row[f"z_{_type}_{landmark_index}"])

        return pd.DataFrame.from_dict(data)

    # assign desired colors to landmarks
    def assign_color(row):
        if row == 'face':
            return 'red'
        elif 'hand' in row:
            return 'dodgerblue'
        else:
            return 'green'

    # specifies the plotting order
    def assign_order(row):
        if row.type == 'face':
            return row.landmark_index + 101
        elif row.type == 'pose':
            return row.landmark_index + 30
        elif row.type == 'left_hand':
            return row.landmark_index + 80
        else:
            return row.landmark_index

    def visualise2d_landmarks(parquet_df, title=""):
        connections = [
            [0, 1, 2, 3, 4,],
            [0, 5, 6, 7, 8],
            [0, 9, 10, 11, 12],
            [0, 13, 14, 15, 16],
            [0, 17, 18, 19, 20],


            [38, 36, 35, 34, 30, 31, 32, 33, 37],
            [40, 39],
            [52, 46, 50, 48, 46, 44, 42, 41, 43, 45, 47, 49, 45, 51],
            [42, 54, 56, 58, 60, 62, 58],
            [41, 53, 55, 57, 59, 61, 57],
            [54, 53],


            [80, 81, 82, 83, 84, ],
            [80, 85, 86, 87, 88],
            [80, 89, 90, 91, 92],
            [80, 93, 94, 95, 96],
            [80, 97, 98, 99, 100], ]

        parquet_df = map_new_to_old_style(parquet_df)
        frames = sorted(set(parquet_df.frame))
        first_frame = min(frames)
        parquet_df['color'] = parquet_df.type.apply(lambda row: assign_color(row))
        parquet_df['plot_order'] = parquet_df.apply(lambda row: assign_order(row), axis=1)
        first_frame_df = parquet_df[parquet_df.frame == first_frame].copy()
        first_frame_df = first_frame_df.sort_values(["plot_order"]).set_index('plot_order')


        frames_l = []
        for frame in frames:
            filtered_df = parquet_df[parquet_df.frame == frame].copy()
            filtered_df = filtered_df.sort_values(["plot_order"]).set_index("plot_order")
            traces = [go.Scatter(
                x=filtered_df['x'],
                y=filtered_df['y'],
                mode='markers',
                marker=dict(
                    color=filtered_df.color,
                    size=9))]

            for i, seg in enumerate(connections):
```

```
            trace = go.Scatter(
                    x=filtered_df.loc[seg]['x'],
                    y=filtered_df.loc[seg]['y'],
                    mode='lines',
            )
            traces.append(trace)
        frame_data = go.Frame(data=traces, traces = [i for i in range(17)])
        frames_l.append(frame_data)

    traces = [go.Scatter(
        x=first_frame_df['x'],
        y=first_frame_df['y'],
        mode='markers',
        marker=dict(
            color=first_frame_df.color,
            size=9
        )
    )]
    for i, seg in enumerate(connections):
        trace = go.Scatter(
            x=first_frame_df.loc[seg]['x'],
            y=first_frame_df.loc[seg]['y'],
            mode='lines',
            line=dict(
                color='black',
                width=2
            )
        )
        traces.append(trace)
    fig = go.Figure(
        data=traces,
        frames=frames_l
    )


    fig.update_layout(
        width=500,
        height=800,
        scene={
            'aspectmode': 'data',
        },
        updatemenus=[
            {
                "buttons": [
                    {
                        "args": [None, {"frame": {"duration": 100,
                                                  "redraw": True},
                                        "fromcurrent": True,
                                        "transition": {"duration": 0}}],
                        "label": "&#9654;",
                        "method": "animate",
                    },

                ],
                "direction": "left",
                "pad": {"r": 100, "t": 100},
                "font": {"size":30},
                "type": "buttons",
                "x": 0.1,
                "y": 0,
            }
        ],
    )
    camera = dict(
        up=dict(x=0, y=-1, z=0),
        eye=dict(x=0, y=0, z=2.5)
    )
    fig.update_layout(title_text=title, title_x=0.5)
    fig.update_layout(scene_camera=camera, showlegend=False)
    fig.update_layout(xaxis = dict(visible=False),
            yaxis = dict(visible=False),
    )
    fig.update_yaxes(autorange="reversed")

    fig.show()


def get_phrase(df, file_id, sequence_id):
```

```
def get_phrase(df, file_id, sequence_id):
    return df[
        np.logical_and(
            df.file_id == file_id,
            df.sequence_id == sequence_id
        )
    ].phrase.iloc[0]
```

1. Data overview

The goal of this competition is to detect and translate American Sign Language (ASL) fingerspelling into text.

`[train/supplemental_metadata].csv`

- `path` - The path to the landmark file.
- `file_id` - A unique identifier for the data file.
- `sequence_id` - A unique identifier for the landmark sequence. Each data file may contain many sequences.
- `phrase` - The labels for the landmark sequence. The train and test datasets contain randomly generated addresses, phone numbers, and urls derived from components of real addresses/phone numbers/urls. Any overlap with real addresses, phone numbers, or urls is purely accidental. The supplemental dataset consists of fingerspelled sentences. Note that some of the urls include adult content. The intent of this competition is to support the Deaf and Hard of Hearing community in engaging with technology on an equal footing with other adults.

`[train/supplemental]_landmarks/` - the landmark data. The landmarks were extracted from raw videos with the **MediaPipe holistic model**. Not all of the frames necessarily had visible hands or hands that could be detected by the model.

The landmark files contain the same data as in the ASL Signs competition (minus the row ID column) but reshaped into a wide format. This allows you to take advantage of the Parquet format to entirely skip loading landmarks that you aren't using.

- `sequence_id` - A unique identifier for the landmark sequence. Most landmark files contain 1,000 sequences. The sequence ID is used as the dataframe index.
- `frame` - The frame number within a landmark sequence.
- `sequence_id` - A unique identifier for the landmark sequence. Each data file may contain many sequences.
- `[x/y/z]_[type]_[landmark_index]` - There are now 1,629 spatial coordinate columns for the x, y and z coordinates for each of the 543 landmarks. The type of landmark is one of ['face', 'left_hand', 'pose', 'right_hand'].

1.1 American sign language (ASL)

ASL Basics: American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English.

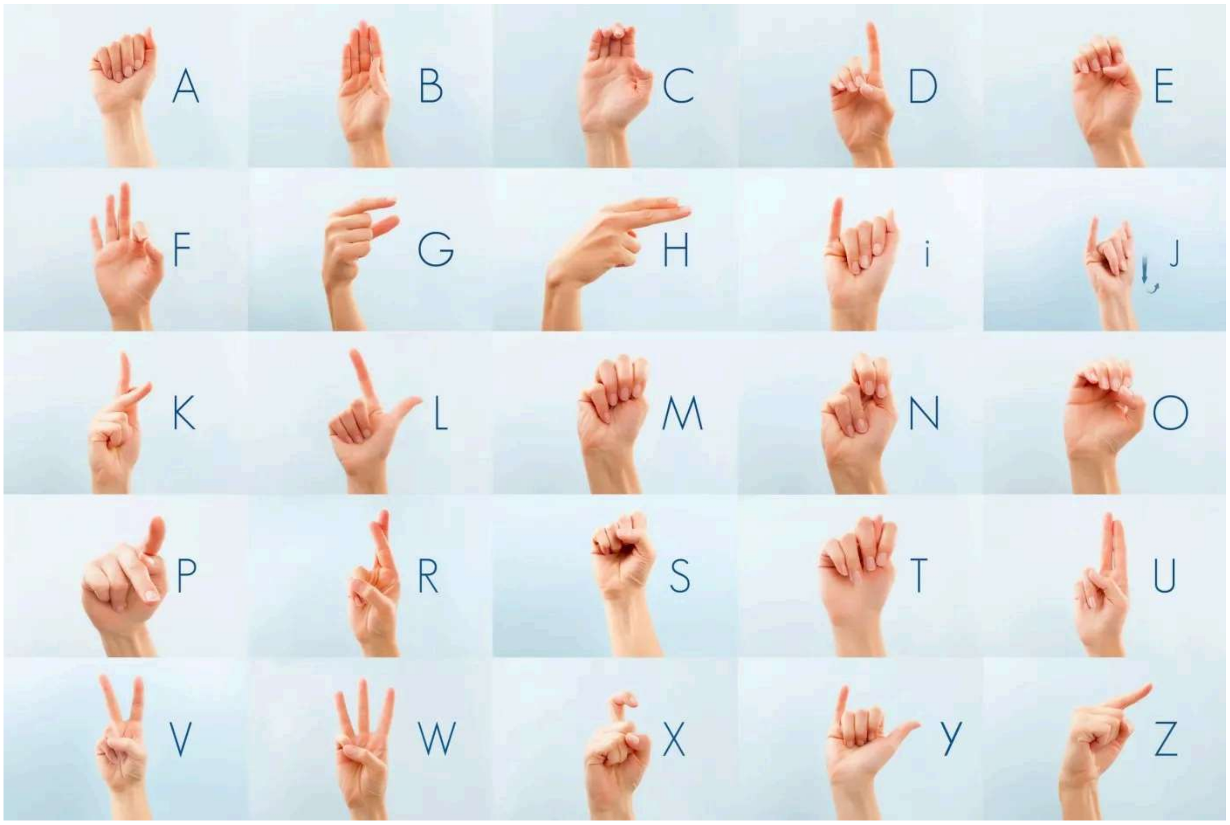
Sign Language in Different Countries: There is no universal sign language. Different sign languages are used in different countries or regions. Some countries adopt features of ASL in their sign languages.

Origins of ASL: No person or committee invented ASL. The exact beginnings of ASL are not clear, but some suggest that it arose more than 200 years ago from the intermixing of local sign languages and French Sign Language (LSF, or Langue des Signes Française).

ASL Compared to Spoken Language: ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. Fingerspelling is part of ASL and is used to spell out English words.

Neurobiology of Language Development: Study of sign language can also help scientists understand the neurobiology of language development. Better understanding of the neurobiology of language could provide a translational foundation for treating injury to the language system, for employing signs or gestures in therapy for children or adults, and for diagnosing language impairment in individuals who are deaf.

Sign Languages Created Among Small Communities: The NIDCD is also funding research on sign languages created among small communities of people with little to no outside influence.



ASL finger alphabet

1.2 MediaPipe holistic model

◯ MediaPipe Holistic Solution is a powerful, easy-to-use software tool that can detect and track multiple human body parts and gestures in real-time video streams. It is open-source and can run on a variety of platforms, including mobile devices, making it an ideal solution for our competition.

◯ **Real-time Perception:** Real-time, simultaneous perception of human pose, face landmarks, and hand tracking can enable impactful applications like fitness analysis, gesture control, and sign language recognition.

◯ **Open-Source Framework:** MediaPipe is an open-source framework designed for complex perception pipelines.

◯ **State-of-the-Art Solution:** MediaPipe Holistic is a solution that provides a state-of-the-art human pose topology, consisting of optimized pose, face, and hand components that each run in real-time.

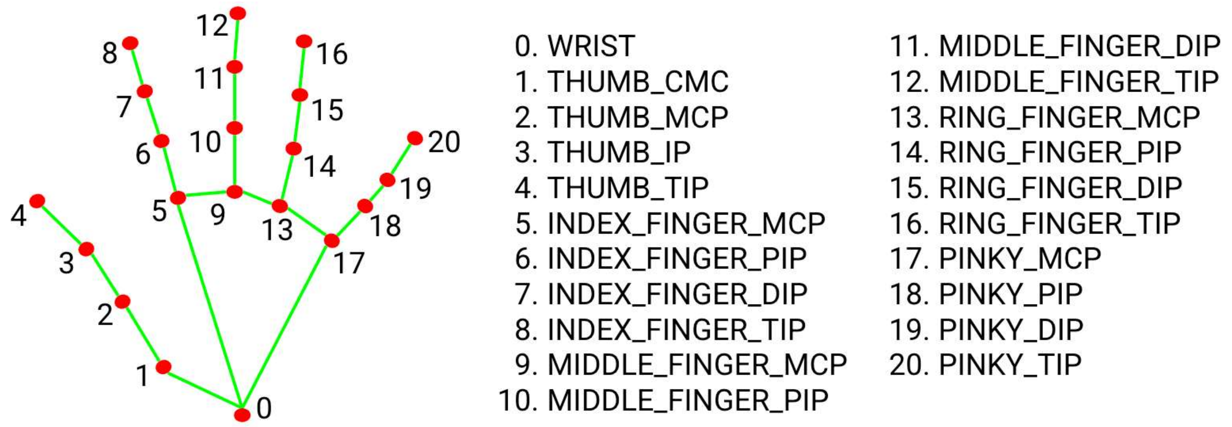
◯ **Unified Topology:** MediaPipe Holistic provides a unified topology for 540+ keypoints and is available on-device for mobile and desktop.

◯ **Separate ML Models For Separate Tasks:** The pipeline integrates separate models for pose, face, and hand components, treating the different regions using a region-appropriate image resolution.

◯ **Significant Model Coordination:** MediaPipe Holistic requires coordination between up to 8 models per frame and optimized machine learning models and pre- and post-processing algorithms for performance benefits.

◯ **Performance Benefits:** The multi-stage nature of the pipeline provides performance benefits, as models are mostly independent and can be replaced with lighter or heavier versions.

[Example of MediaPipe Holistic]



[MediaPipe Landmarks for Hands]

2. Train overview

◯ First load train.csv file.

```
train_df = pd.read_csv("/kaggle/input/asl-fingerspelling/train.csv")
```

◯ Let's look how the data looks like.

```
train_df.head()
```

◯ Show a discribe of the dataframe with its basic information.

```
train_df.describe()
```

🔴 Let's try to visualize sequence `1816796431` from the file `5414471`.

```
sequence_id = 1816796431
file_id = 5414471
```

```
path_to_sign = f"/kaggle/input/asl-fingerspelling/train_landmarks/{file_id}.parquet"
sign = pd.read_parquet(path_to_sign)
```

◯ Here is what is inside this parquet file:

```
sign
```

◯ The number of unique sequences in an `.parquet` file is **1000**.

```
len(np.unique(sign.index))
```

◯ Let's get only sequence `1816796431` from the file.

```
sequence = sign[sign.index == sequence_id]
sequence
```

● Now we can visualize the sequence and the phrase.

```
sequence_phrase = get_phrase(train_df, file_id, sequence_id)
visualise2d_landmarks(sequence, f"Phrase: {sequence_phrase}")
```

```
#TODO: Plot NaN distributions for some features
```

```
#TODO: Text length distribution
```

```
#TODO: Text lenght distribution
```

```
#TODO: Create participant_id distribution
```

```
#TODO: Frames distribution
```

3. Supplemental overview

◯ First load supplemental_metadata.csv file.

```
supplemental_df = pd.read_csv("/kaggle/input/asl-fingerspelling/supplemental_metadata.csv")
```

◯ Let's look how the data looks like.

```
supplemental_df.head()
```

◯ Show a discribe of the dataframe with its basic information.

```
supplemental_df.describe()
```

● Let's try to visualize sequence `1535467051` from the file `33432165`.

```
sequence_id = 1535467051
file_id = 33432165
```

```
path_to_sign = f"/kaggle/input/asl-fingerspelling/supplemental_landmarks/{file_id}.parquet"
sign = pd.read_parquet(path_to_sign)
```

◯ Here is what is inside this parquet file:

```
sign
```

◯ The number of unique sequences in an `.parquet` file is **1000**.

```
len(np.unique(sign.index))
```

◯ Let's get only sequence `1535467051` from the file.

```
sequence = sign[sign.index == sequence_id]
sequence
```

🔴 Now we can visualize and sequence with the phrase.

```
sequence_phrase = get_phrase(supplemental_df, file_id, sequence_id)
visualise2d_landmarks(sequence, f"Phrase: {sequence_phrase}")
```

```
# ( ˘▽˘) WORK STILL IN PROGRESS
```

༼ つ ◕_◕ ༽つ Thank You!

💌 Thank you for taking the time to read through my notebook. I hope you found it interesting and informative. If you have any feedback or suggestions for improvement, please don't hesitate to let me know in the comments.

🚀 If you liked this notebook, please consider upvoting it so that others can discover it too. Your support means a lot to me, and it helps to motivate me to create more content in the future.

❤️ Once again, thank you for your support, and I hope to see you again soon!