AAI521_Final_Jeremy_V1

December 11, 2023

[ ]:
```
!pip install numpy pandas matplotlib opencv-python tensorflow
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(1.5.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-
packages (4.8.0.76)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-
packages (2.14.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
```

```
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.2)
Requirement already satisfied: tensorboard<2.15,>=2.14 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.41.3)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (3.5.1)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (2.31.0)
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.15,>=2.14->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-
auth-oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (3.2.2)
```

[ ]:
```
# Install the Kaggle API
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages
(1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-
packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from kaggle) (2023.7.22)
Requirement already satisfied: python-dateutil in
```

```
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from kaggle) (4.66.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-
packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-
packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->kaggle) (3.4)
```

[ ]:
```
# Upload the Kaggle API token
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

[ ]:
```
# Move the uploaded file to the required directory
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

[ ]:
```
# Download the ASL dataset from Kaggle
!kaggle competitions download -c asl-signs
```

```
Downloading asl-signs.zip to /content
100% 37.3G/37.4G [05:44<00:00, 115MB/s]
100% 37.4G/37.4G [05:44<00:00, 116MB/s]
```

[ ]:
```
# Unzip the downloaded dataset
!unzip -q asl-signs.zip
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from tqdm.notebook import tqdm
import tensorflow as tf
```

```
import json
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.utils import to_categorical
```

[ ]:
```
# Read the CSV file
df = pd.read_csv('train.csv')

# Display basic information about the dataset
df
```

[ ]:
```
                                                   path  participant_id  \
0      train_landmark_files/26734/1000035562.parquet           26734
1      train_landmark_files/28656/1000106739.parquet           28656
2       train_landmark_files/16069/100015657.parquet           16069
3      train_landmark_files/25571/1000210073.parquet           25571
4      train_landmark_files/62590/1000240708.parquet           62590
…                                                 …              …
94472   train_landmark_files/53618/999786174.parquet           53618
94473   train_landmark_files/26734/999799849.parquet           26734
94474   train_landmark_files/25571/999833418.parquet           25571
94475   train_landmark_files/29302/999895257.parquet           29302
94476   train_landmark_files/36257/999962374.parquet           36257

       sequence_id    sign
0       1000035562    blow
1       1000106739    wait
2        100015657   cloud
3       1000210073    bird
4       1000240708    owie
…              …      …
94472    999786174   white
94473    999799849    have
94474    999833418  flower
94475    999895257    room
94476    999962374   happy

[94477 rows x 4 columns]
```

[ ]:
```
# Check for missing values
print(df.isnull().sum())
```

```
path              0
participant_id    0
sequence_id       0
```

```
sign              0
dtype: int64
```

[ ]:
```
# Load the sign index mapping from the JSON file
with open('sign_to_prediction_index_map.json', 'r') as f:
    sign_index_mapping = json.load(f)
```

[ ]:
```
# Convert the sign index mapping to a DataFrame
sign_index_df = pd.DataFrame(list(sign_index_mapping.items()), columns=['sign',␣
↪'sign_info'])

# Display basic information about the sign index DataFrame
sign_index_df.head()
```

[ ]:
```
        sign  sign_info
0         TV          0
1      after          1
2   airplane          2
3        all          3
4  alligator          4
```

[ ]:
```
# Merge the two DataFrames based on the 'sign' column
merged_df = pd.merge(df, sign_index_df, how='left', on='sign')

# Display basic information about the merged DataFrame
merged_df.head()
```

[ ]:
```
                                            path  participant_id  sequence_id  \
0  train_landmark_files/26734/1000035562.parquet           26734   1000035562
1  train_landmark_files/28656/1000106739.parquet           28656   1000106739
2   train_landmark_files/16069/100015657.parquet           16069    100015657
3  train_landmark_files/25571/1000210073.parquet           25571   1000210073
4  train_landmark_files/62590/1000240708.parquet           62590   1000240708

    sign  sign_info
0   blow         25
1   wait        232
2  cloud         48
3   bird         23
4   owie        164
```

[ ]:
```
sample_fillede\['type'].unique()
```

[ ]:
```
array(['face', 'left_hand', 'pose', 'right_hand'], dtype=object)
```

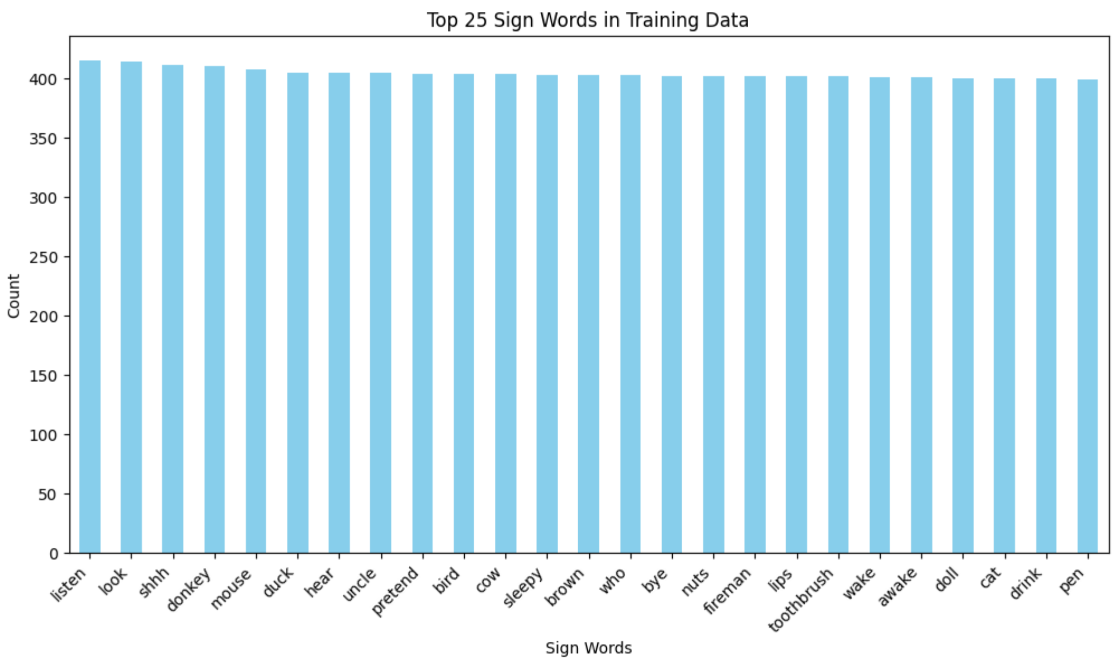
[ ]:
```
sample['frame'].unique()
```

[ ]:
```
array([17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28], dtype=int16)
```

[ ]:
```python
# Assuming your DataFrame is named 'train_df'
top_signs = df['sign'].value_counts().head(25)

# Display the top 25 sign words
print(top_signs)
```

```
listen        415
look          414
shhh          411
donkey        410
mouse         408
duck          405
hear          405
uncle         405
pretend       404
bird          404
cow           404
sleepy        403
brown         403
who           403
bye           402
nuts          402
fireman       402
lips          402
toothbrush    402
wake          401
awake         401
doll          400
cat           400
drink         400
pen           399
Name: sign, dtype: int64
```

[ ]:
```python
import matplotlib.pyplot as plt

# Plot the top 25 sign words
plt.figure(figsize=(12, 6))
top_signs.plot(kind='bar', color='skyblue')
plt.title('Top 25 Sign Words in Training Data')
plt.xlabel('Sign Words')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.show()
```



[ ]:
```python
#Read a Parquet file and set a sample
file_path = '/content/train_landmark_files/25571/1000210073.parquet'
landmark_sample = pd.read_parquet(file_path)

# Display the loaded data
landmark_sample
```

[ ]:
```
      frame              row_id        type  landmark_index         x         y  \
0        17           17-face-0        face               0  0.495870  0.478694
1        17           17-face-1        face               1  0.492222  0.447209
2        17           17-face-2        face               2  0.492067  0.457237
3        17           17-face-3        face               3  0.480419  0.415996
4        17           17-face-4        face               4  0.492035  0.437453
…      …                  …           …               …        …         …
6511     28  28-right_hand-16  right_hand              16  0.506396  0.868416
6512     28  28-right_hand-17  right_hand              17  0.323227  0.835990
6513     28  28-right_hand-18  right_hand              18  0.435733  0.848917
6514     28  28-right_hand-19  right_hand              19  0.476093  0.867098
6515     28  28-right_hand-20  right_hand              20  0.488775  0.885244

             z
0    -0.037412
1    -0.067939
2    -0.035722
3    -0.050779
```

```
4    -0.072314
…          …
6511 -0.139545
6512 -0.136632
6513 -0.156200
6514 -0.149442
6515 -0.142629

[6516 rows x 7 columns]
```

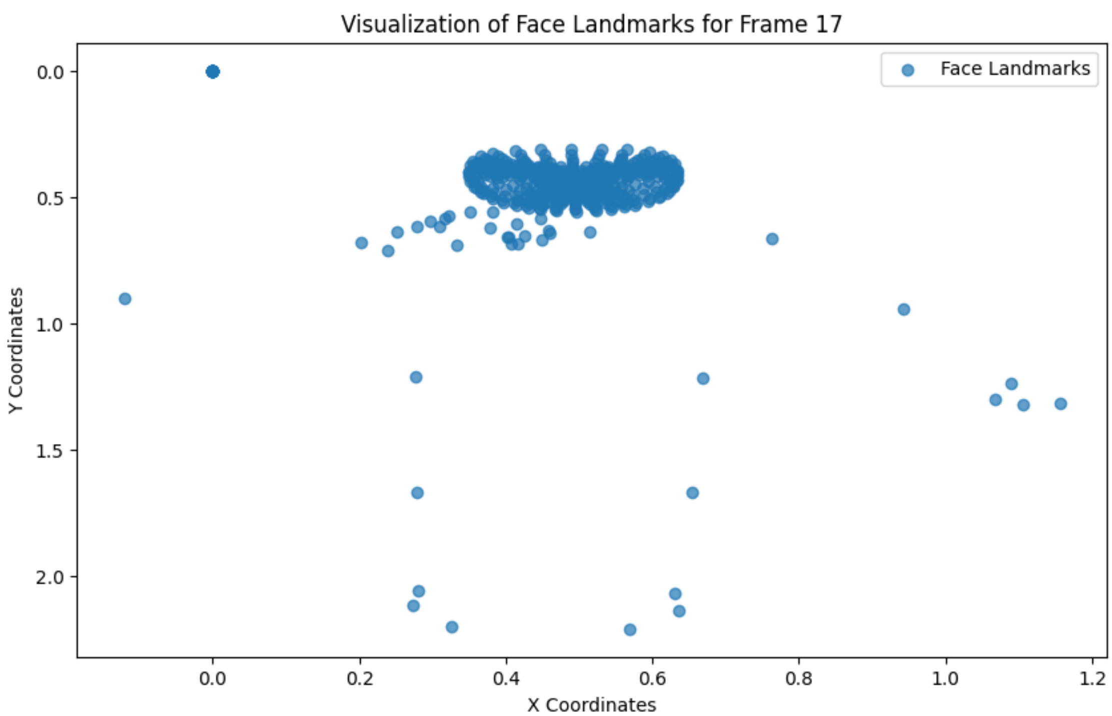
[ ]:
```python
# Replace all NaN values with 0
sample = landmark_sample.fillna(0)

# Display the DataFrame with null values replaced
sample
```

[ ]:
```
      frame             row_id        type  landmark_index         x         y  \
0        17          17-face-0        face               0  0.495870  0.478694
1        17          17-face-1        face               1  0.492222  0.447209
2        17          17-face-2        face               2  0.492067  0.457237
3        17          17-face-3        face               3  0.480419  0.415996
4        17          17-face-4        face               4  0.492035  0.437453
…     …                   …          …               …        …         …
6511     28  28-right_hand-16  right_hand              16  0.506396  0.868416
6512     28  28-right_hand-17  right_hand              17  0.323227  0.835990
6513     28  28-right_hand-18  right_hand              18  0.435733  0.848917
6514     28  28-right_hand-19  right_hand              19  0.476093  0.867098
6515     28  28-right_hand-20  right_hand              20  0.488775  0.885244

             z
0    -0.037412
1    -0.067939
2    -0.035722
3    -0.050779
4    -0.072314
…          …
6511 -0.139545
6512 -0.136632
6513 -0.156200
6514 -0.149442
6515 -0.142629

[6516 rows x 7 columns]
```

[ ]:
```python
# Filter the DataFrame for a specific frame
frame_103_face = sample[(sample['frame'] == 17)]
```

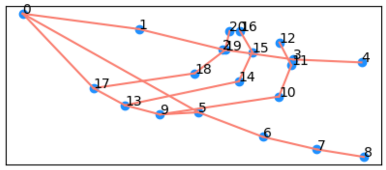
```
# Plot each type of landmark separately
plt.figure(figsize=(12, 8))

# Scatter plot for 'face' landmarks with reversed y-axis
plt.figure(figsize=(10, 6))
plt.scatter(frame_103_face['x'], frame_103_face['y'], label='Face Landmarks',␣
↪alpha=0.7)

# Reverse the y-axis
plt.gca().invert_yaxis()

# Set plot properties
plt.title('Visualization of Face Landmarks for Frame 17')
plt.xlabel('X Coordinates')
plt.ylabel('Y Coordinates')
plt.legend()
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

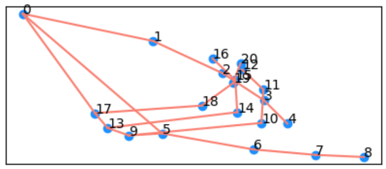


[ ]:
```
# pick the left hand and right hand points
sample_left_hand = sample[sample.type == "left_hand"]
sample_right_hand = sample[sample.type == "right_hand"]
```

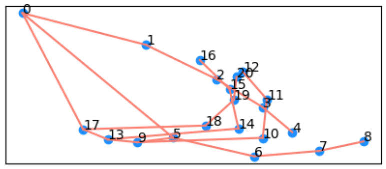
```
# display(sample_left_hand)

# edges that represents the hand edges
# How he knows the edges, so a mystery
edges =␣
↪[(0,1),(1,2),(2,3),(3,4),(0,5),(0,17),(5,6),(6,7),(7,8),(5,9),(9,10),(10,11),(11,12),
         (9,13),(13,14),(14,15),(15,16),(13,17),(17,18),(18,19),(19,20)]

# plotting a single frame into matplotlib
def plot_frame(df, frame_id, ax):
    df = df[df.frame == frame_id].sort_values(['landmark_index'])
    x = list(df.x)
    y = list(df.y)

    # plotting the points
    ax.scatter(df.x, df.y, color='dodgerblue')
    for i in range(len(x)):
        ax.text(x[i], y[i], str(i))

    # plotting the edges that represents the hand
    for edge in edges:
        ax.plot([x[edge[0]], x[edge[1]]], [y[edge[0]], y[edge[1]]],␣
↪color='salmon')
        ax.set_xlabel(f"Frame no. {frame_id}")
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_xticklabels([])
        ax.set_yticklabels([])

# plotting the multiple frames
def plot_frame_seq(df, frame_range, n_frames):
    frames = np.linspace(frame_range[0],frame_range[1],n_frames, dtype = int,␣
↪endpoint=True)
    fig, ax = plt.subplots(n_frames, 1, figsize=(5,25))
    for i in range(n_frames):
        plot_frame(df, frames[i], ax[i])

    plt.show()

plot_frame_seq(sample_right_hand, (17,28), 10)
```



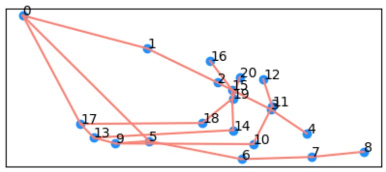
Frame no. 17



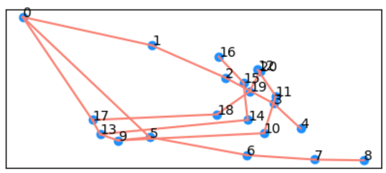
Frame no. 18



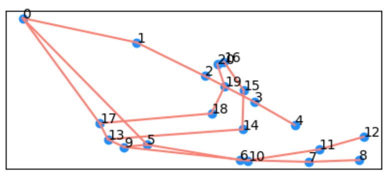
Frame no. 19



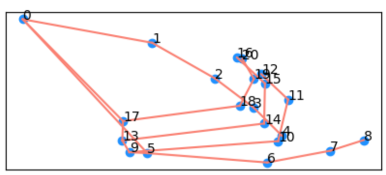
Frame no. 20



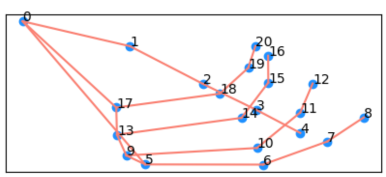
Frame no. 21



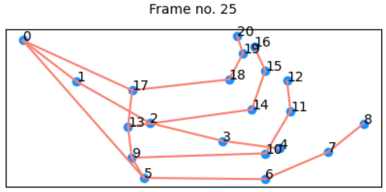
Frame no. 23



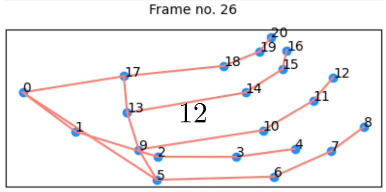
Frame no. 24



Frame no. 25



Frame no. 26



Frame no. 28

[ ]:
```python
# Load landmark data (assuming you have a function to load Parquet files)
landmark_df = load_landmark_data('path/to/landmark_data.parquet')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-49-53b7f909f2e5> in <cell line: 1>()
----> 1 merged_df1 = pd.merge(df, sample, on='participant_id', how='inner')

/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/merge.py in␣
 ↪merge(left, right, how, on, left_on, right_on, left_index, right_index, sort,␣
 ↪suffixes, copy, indicator, validate)
    108     validate: str | None = None,
    109 ) -> DataFrame:
--> 110     op = _MergeOperation(

    111         left,
    112         right,

/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/merge.py in␣
 ↪__init__(self, left, right, how, on, left_on, right_on, axis, left_index,␣
 ↪right_index, sort, suffixes, indicator, validate)
    701             self.right_join_keys,
    702             self.join_names,
--> 703         ) = self._get_merge_keys()

    704
    705         # validate the merge keys dtypes. We may need to coerce

/usr/local/lib/python3.10/dist-packages/pandas/core/reshape/merge.py in␣
 ↪_get_merge_keys(self)
   1160                         rk = cast(Hashable, rk)
   1161                         if rk is not None:
-> 1162                             right_keys.append(right.
 ↪_get_label_or_level_values(rk))
   1163                         else:
   1164                             # work-around for␣
 ↪merge_asof(right_index=True)

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in␣
 ↪_get_label_or_level_values(self, key, axis)
   1848             )
   1849         else:
-> 1850             raise KeyError(key)
   1851
   1852         # Check for duplicates
```

```
KeyError: 'participant_id'
```

[ ]:
```
# Function to load landmark data from Parquet files in a folder
def load_landmark_data(folder_path):
    combined_meta = {}
    for root, dirs, files in os.walk(folder_path):
        for file_name in tqdm(files):
            if file_name.endswith(".parquet"):
                file_path = os.path.join(root, file_name)
                example_landmark = pd.read_parquet(file_path)

                # Replace null values with 0
                example_landmark.fillna(0, inplace=True)

                # Get the number of landmarks with x, y, z data per type
                meta = example_landmark.dropna(subset=["x", "y", "z"])["type"].
 ↪value_counts().to_dict()
                meta["frames"] = example_landmark["frame"].nunique()

                # Calculate additional statistics if needed
                xyz_meta = (
                    example_landmark.agg(
                        {
                            "x": ["min", "max", "mean"],
                            "y": ["min", "max", "mean"],
                            "z": ["min", "max", "mean"],
                        }
                    )
                    .unstack()
                    .to_dict()
                )

                for key in xyz_meta.keys():
                    new_key = key[0] + "_" + key[1]
                    meta[new_key] = xyz_meta[key]

                combined_meta[file_path] = meta

    return combined_meta

# Specify the path to the root folder containing participant folders
root_folder_path = '/content/train_landmark_files'

# Load landmark data from all participant folders
combined_meta_all = {}
for participant_folder in tqdm(os.listdir(root_folder_path)):
```

```python
    participant_folder_path = os.path.join(root_folder_path, participant_folder)

    if os.path.isdir(participant_folder_path):
        participant_combined_meta = load_landmark_data(participant_folder_path)
        combined_meta_all.update(participant_combined_meta)

# Create a DataFrame from the combined metadata
metadata_df = pd.DataFrame.from_dict(combined_meta_all, orient='index').
↪reset_index()
metadata_df.rename(columns={'index': 'file_path'}, inplace=True)

# Display the resulting DataFrame
print(metadata_df.head())
```

```
  0%|          | 0/21 [00:00<?, ?it/s]
  0%|          | 0/4826 [00:00<?, ?it/s]
  0%|          | 0/4841 [00:00<?, ?it/s]
  0%|          | 0/4753 [00:00<?, ?it/s]
  0%|          | 0/4677 [00:00<?, ?it/s]
  0%|          | 0/4810 [00:00<?, ?it/s]
  0%|          | 0/4563 [00:00<?, ?it/s]
  0%|          | 0/3499 [00:00<?, ?it/s]
  0%|          | 0/4968 [00:00<?, ?it/s]
  0%|          | 0/4900 [00:00<?, ?it/s]
  0%|          | 0/3865 [00:00<?, ?it/s]
  0%|          | 0/4545 [00:00<?, ?it/s]
  0%|          | 0/3502 [00:00<?, ?it/s]
  0%|          | 0/4722 [00:00<?, ?it/s]
  0%|          | 0/4563 [00:00<?, ?it/s]
  0%|          | 0/4782 [00:00<?, ?it/s]
  0%|          | 0/3338 [00:00<?, ?it/s]
  0%|          | 0/4656 [00:00<?, ?it/s]
  0%|          | 0/4848 [00:00<?, ?it/s]
  0%|          | 0/4648 [00:00<?, ?it/s]
  0%|          | 0/4896 [00:00<?, ?it/s]
  0%|          | 0/4275 [00:00<?, ?it/s]
```

```
                                         file_path   face  pose  left_hand  \
0  /content/train_landmark_files/55372/2802786652…   7956   561        357
1  /content/train_landmark_files/55372/3403106688…  14508  1023        651
2  /content/train_landmark_files/55372/1127624485…   8424   594        378
3  /content/train_landmark_files/55372/1559766834…   8424   594        378
4  /content/train_landmark_files/55372/657631983…   6552   462        294

   right_hand  frames     x_min     x_max    x_mean  y_min     y_max  \
0         357      17 -0.087367  1.199376  0.448886    0.0  2.479705
1         651      31 -0.240969  1.178582  0.419019    0.0  2.441859
2         378      18 -0.146753  1.073904  0.404705    0.0  2.518284
3         378      18 -0.069765  1.265994  0.388586    0.0  2.612595
4         294      14 -0.423106  1.303239  0.403132    0.0  2.532954

     y_mean     z_min     z_max    z_mean
0  0.386801 -3.059139  3.362435 -0.055478
1  0.370088 -2.872532  1.589201 -0.058870
2  0.403468 -2.520643  1.895188 -0.038549
3  0.378294 -2.927297  2.471197 -0.022748
4  0.364148 -2.680002  2.279785 -0.042742
```

[ ]:
```
metadata_df
```

[ ]:
```
                                             file_path   face  pose  \
0      /content/train_landmark_files/55372/2802786652…   7956   561
1      /content/train_landmark_files/55372/3403106688…  14508  1023
2      /content/train_landmark_files/55372/1127624485…   8424   594
3      /content/train_landmark_files/55372/1559766834…   8424   594
4      /content/train_landmark_files/55372/657631983…   6552   462
…                                                  …    …    …
94472  /content/train_landmark_files/27610/1696867677…  54756  3861
94473  /content/train_landmark_files/27610/2975578577…  49608  3498
94474  /content/train_landmark_files/27610/4223702977…  37440  2640
94475  /content/train_landmark_files/27610/558510995…   4680   330
94476  /content/train_landmark_files/27610/314634651…  11232   792

       left_hand  right_hand  frames     x_min     x_max    x_mean  y_min  \
0            357         357      17 -0.087367  1.199376  0.448886    0.0
1            651         651      31 -0.240969  1.178582  0.419019    0.0
2            378         378      18 -0.146753  1.073904  0.404705    0.0
3            378         378      18 -0.069765  1.265994  0.388586    0.0
4            294         294      14 -0.423106  1.303239  0.403132    0.0
…              …         …       …         …        …        …
94472       2457        2457     117 -0.117516  0.951807  0.261810    0.0
94473       2226        2226     106 -0.147046  0.976050  0.365789    0.0
94474       1680        1680      80 -0.079137  1.134678  0.466686    0.0
94475        210         210      10 -0.066891  0.939062  0.431534    0.0
```

```
94476        504         504      24 -0.053534  0.954912  0.390166    0.0

          y_max    y_mean     z_min     z_max    z_mean
0      2.479705  0.386801 -3.059139  3.362435 -0.055478
1      2.441859  0.370088 -2.872532  1.589201 -0.058870
2      2.518284  0.403468 -2.520643  1.895188 -0.038549
3      2.612595  0.378294 -2.927297  2.471197 -0.022748
4      2.532954  0.364148 -2.680002  2.279785 -0.042742
…           …         …         …         …         …
94472  2.467156  0.463274 -2.706611  1.537550 -0.037171
94473  2.550603  0.493334 -2.661751  1.042857 -0.050042
94474  2.357260  0.496470 -2.643354  1.984523 -0.032431
94475  2.559409  0.609520 -2.620925  1.503764 -0.050653
94476  2.431209  0.508731 -2.105850  1.693273 -0.027899

[94477 rows x 15 columns]
```

[ ]:
```python
# Assuming your DataFrame is named 'df'
metadata_df['file_path'] = metadata_df['file_path'].str.replace('/content/', '')

# Display the updated DataFrame
metadata_df
```

[ ]:
```
                                          file_path   face  pose  left_hand  \
0      train_landmark_files/55372/2802786652.parquet   7956   561        357
1      train_landmark_files/55372/3403106688.parquet  14508  1023        651
2      train_landmark_files/55372/1127624485.parquet   8424   594        378
3      train_landmark_files/55372/1559766834.parquet   8424   594        378
4       train_landmark_files/55372/657631983.parquet   6552   462        294
…                                               …      …     …          …
94472  train_landmark_files/27610/1696867677.parquet  54756  3861       2457
94473  train_landmark_files/27610/2975578577.parquet  49608  3498       2226
94474  train_landmark_files/27610/4223702977.parquet  37440  2640       1680
94475   train_landmark_files/27610/558510995.parquet   4680   330        210
94476   train_landmark_files/27610/314634651.parquet  11232   792        504

       right_hand  frames     x_min     x_max    x_mean  y_min     y_max  \
0             357      17 -0.087367  1.199376  0.448886    0.0  2.479705
1             651      31 -0.240969  1.178582  0.419019    0.0  2.441859
2             378      18 -0.146753  1.073904  0.404705    0.0  2.518284
3             378      18 -0.069765  1.265994  0.388586    0.0  2.612595
4             294      14 -0.423106  1.303239  0.403132    0.0  2.532954
…             …       …         …         …         …      …
94472        2457     117 -0.117516  0.951807  0.261810    0.0  2.467156
94473        2226     106 -0.147046  0.976050  0.365789    0.0  2.550603
94474        1680      80 -0.079137  1.134678  0.466686    0.0  2.357260
94475         210      10 -0.066891  0.939062  0.431534    0.0  2.559409
```

```
94476         504      24 -0.053534  0.954912  0.390166    0.0  2.431209

         y_mean     z_min     z_max    z_mean
0      0.386801 -3.059139  3.362435 -0.055478
1      0.370088 -2.872532  1.589201 -0.058870
2      0.403468 -2.520643  1.895188 -0.038549
3      0.378294 -2.927297  2.471197 -0.022748
4      0.364148 -2.680002  2.279785 -0.042742
…           …         …         …         …
94472  0.463274 -2.706611  1.537550 -0.037171
94473  0.493334 -2.661751  1.042857 -0.050042
94474  0.496470 -2.643354  1.984523 -0.032431
94475  0.609520 -2.620925  1.503764 -0.050653
94476  0.508731 -2.105850  1.693273 -0.027899

[94477 rows x 15 columns]
```

[ ]:
```python
# Assuming your DataFrame is named 'df'
metadata_df.rename(columns={'file_path': 'path'}, inplace=True)

# Display the updated DataFrame
metadata_df
```

[ ]:
```
                                                path   face  pose  left_hand  \
0      train_landmark_files/55372/2802786652.parquet   7956   561        357
1      train_landmark_files/55372/3403106688.parquet  14508  1023        651
2      train_landmark_files/55372/1127624485.parquet   8424   594        378
3      train_landmark_files/55372/1559766834.parquet   8424   594        378
4       train_landmark_files/55372/657631983.parquet   6552   462        294
…                                                ...    …     …          …
94472  train_landmark_files/27610/1696867677.parquet  54756  3861       2457
94473  train_landmark_files/27610/2975578577.parquet  49608  3498       2226
94474  train_landmark_files/27610/4223702977.parquet  37440  2640       1680
94475   train_landmark_files/27610/558510995.parquet   4680   330        210
94476   train_landmark_files/27610/314634651.parquet  11232   792        504

       right_hand  frames     x_min     x_max    x_mean  y_min     y_max  \
0             357      17 -0.087367  1.199376  0.448886    0.0  2.479705
1             651      31 -0.240969  1.178582  0.419019    0.0  2.441859
2             378      18 -0.146753  1.073904  0.404705    0.0  2.518284
3             378      18 -0.069765  1.265994  0.388586    0.0  2.612595
4             294      14 -0.423106  1.303239  0.403132    0.0  2.532954
…             …       …         …         …         …      …
94472        2457     117 -0.117516  0.951807  0.261810    0.0  2.467156
94473        2226     106 -0.147046  0.976050  0.365789    0.0  2.550603
94474        1680      80 -0.079137  1.134678  0.466686    0.0  2.357260
94475         210      10 -0.066891  0.939062  0.431534    0.0  2.559409
```

```
94476         504      24 -0.053534  0.954912  0.390166    0.0  2.431209

         y_mean     z_min     z_max    z_mean
0      0.386801 -3.059139  3.362435 -0.055478
1      0.370088 -2.872532  1.589201 -0.058870
2      0.403468 -2.520643  1.895188 -0.038549
3      0.378294 -2.927297  2.471197 -0.022748
4      0.364148 -2.680002  2.279785 -0.042742
…           …         …         …         …
94472  0.463274 -2.706611  1.537550 -0.037171
94473  0.493334 -2.661751  1.042857 -0.050042
94474  0.496470 -2.643354  1.984523 -0.032431
94475  0.609520 -2.620925  1.503764 -0.050653
94476  0.508731 -2.105850  1.693273 -0.027899

[94477 rows x 15 columns]
```

[ ]:
```
# Merge the train and parquet DataFrames on the 'file_path' column
merged_df = pd.merge(metadata_df, df, on='path')
```

[ ]:
```
merged_df
```

[ ]:
```
                                                path   face  pose  left_hand  \
0      train_landmark_files/55372/2802786652.parquet   7956   561        357
1      train_landmark_files/55372/3403106688.parquet  14508  1023        651
2      train_landmark_files/55372/1127624485.parquet   8424   594        378
3      train_landmark_files/55372/1559766834.parquet   8424   594        378
4       train_landmark_files/55372/657631983.parquet   6552   462        294
…                                                …      …     …          …
94472  train_landmark_files/27610/1696867677.parquet  54756  3861       2457
94473  train_landmark_files/27610/2975578577.parquet  49608  3498       2226
94474  train_landmark_files/27610/4223702977.parquet  37440  2640       1680
94475   train_landmark_files/27610/558510995.parquet   4680   330        210
94476   train_landmark_files/27610/314634651.parquet  11232   792        504

       right_hand  frames     x_min     x_max    x_mean  y_min     y_max  \
0             357      17 -0.087367  1.199376  0.448886    0.0  2.479705
1             651      31 -0.240969  1.178582  0.419019    0.0  2.441859
2             378      18 -0.146753  1.073904  0.404705    0.0  2.518284
3             378      18 -0.069765  1.265994  0.388586    0.0  2.612595
4             294      14 -0.423106  1.303239  0.403132    0.0  2.532954
…             …     …         …         …         …      …
94472        2457     117 -0.117516  0.951807  0.261810    0.0  2.467156
94473        2226     106 -0.147046  0.976050  0.365789    0.0  2.550603
94474        1680      80 -0.079137  1.134678  0.466686    0.0  2.357260
94475         210      10 -0.066891  0.939062  0.431534    0.0  2.559409
94476         504      24 -0.053534  0.954912  0.390166    0.0  2.431209
```

```
         y_mean     z_min     z_max    z_mean  participant_id  sequence_id  \
0      0.386801 -3.059139  3.362435 -0.055478           55372   2802786652
1      0.370088 -2.872532  1.589201 -0.058870           55372   3403106688
2      0.403468 -2.520643  1.895188 -0.038549           55372   1127624485
3      0.378294 -2.927297  2.471197 -0.022748           55372   1559766834
4      0.364148 -2.680002  2.279785 -0.042742           55372    657631983
…           …         …        …        …              …            …
94472  0.463274 -2.706611  1.537550 -0.037171           27610   1696867677
94473  0.493334 -2.661751  1.042857 -0.050042           27610   2975578577
94474  0.496470 -2.643354  1.984523 -0.032431           27610   4223702977
94475  0.609520 -2.620925  1.503764 -0.050653           27610    558510995
94476  0.508731 -2.105850  1.693273 -0.027899           27610    314634651

            sign
0            any
1         vacuum
2           look
3      yesterday
4            can
…          …
94472        hot
94473       talk
94474     cowboy
94475       bird
94476        yes

[94477 rows x 18 columns]
```

[ ]:
```
# Specify the path where you want to save the CSV file
csv_file_path = '/content/drive/MyDrive/AAI521/Final Project/merged_df.csv'

# Save the DataFrame to a CSV file
merged_df.to_csv(csv_file_path, index=False)
```

[ ]:
```
# Load the DataFrame from the saved CSV file
merged_df = pd.read_csv('/content/drive/MyDrive/AAI521/Final Project/merged_df.
 ↪csv')
```

[ ]:
```
# Extract features and labels
feature_columns = ["face", "pose", "left_hand", "right_hand", "frames",
                   "x_min", "x_max", "x_mean", "y_min", "y_max", "y_mean",
                   "z_min", "z_max", "z_mean"]

X = merged_df[feature_columns]

label_encoder = LabelEncoder()
```

```
y = label_encoder.fit_transform(merged_df["sign"])
y = to_categorical(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:

```
# Reshape the input data to include a timestep dimension
X_train_reshaped = X_train.values.reshape((X_train.shape[0], 1, X_train.
↪shape[1]))
X_test_reshaped = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, input_shape=(X_train_reshaped.shape[1],␣
↪X_train_reshaped.shape[2])))
model.add(Dense(units=len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

# Train the model
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32,␣
↪validation_data=(X_test_reshaped, y_test))
```

```
Epoch 1/10
2362/2362 [==============================] - 18s 5ms/step - loss: 5.5311 -
accuracy: 0.0038 - val_loss: 5.5295 - val_accuracy: 0.0038
Epoch 2/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5292 -
accuracy: 0.0039 - val_loss: 5.5280 - val_accuracy: 0.0041
Epoch 3/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5291 -
accuracy: 0.0039 - val_loss: 5.5299 - val_accuracy: 0.0040
Epoch 4/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5294 -
accuracy: 0.0036 - val_loss: 5.5296 - val_accuracy: 0.0037
Epoch 5/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5291 -
accuracy: 0.0038 - val_loss: 5.5303 - val_accuracy: 0.0039
Epoch 6/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5293 -
accuracy: 0.0041 - val_loss: 5.5296 - val_accuracy: 0.0029
Epoch 7/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5291 -
accuracy: 0.0039 - val_loss: 5.5313 - val_accuracy: 0.0039
```

```
Epoch 8/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5297 -
accuracy: 0.0038 - val_loss: 5.5282 - val_accuracy: 0.0041
Epoch 9/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5292 -
accuracy: 0.0039 - val_loss: 5.5317 - val_accuracy: 0.0040
Epoch 10/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5291 -
accuracy: 0.0039 - val_loss: 5.5306 - val_accuracy: 0.0040
```

[ ]: <keras.src.callbacks.History at 0x7b880a200fd0>

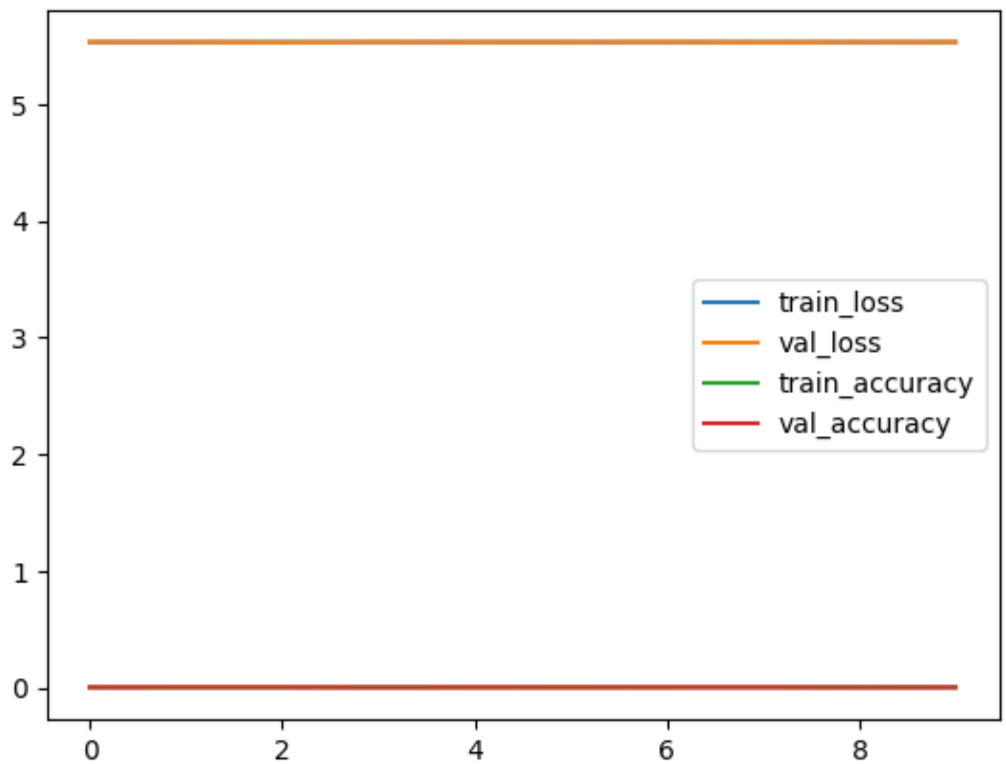
[ ]:
```
loss, accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
591/591 [==============================] - 2s 3ms/step - loss: 5.5306 -
accuracy: 0.0040
Test Loss: 5.530604839324951, Test Accuracy: 0.003969093784689903
```

[ ]:
```
history = model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32,␣
 ↪validation_data=(X_test_reshaped, y_test))

# Plot training history
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/10
2362/2362 [==============================] - 11s 4ms/step - loss: 5.5292 -
accuracy: 0.0037 - val_loss: 5.5305 - val_accuracy: 0.0035
Epoch 2/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5294 -
accuracy: 0.0039 - val_loss: 5.5305 - val_accuracy: 0.0043
Epoch 3/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5293 -
accuracy: 0.0038 - val_loss: 5.5276 - val_accuracy: 0.0042
Epoch 4/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5292 -
accuracy: 0.0043 - val_loss: 5.5310 - val_accuracy: 0.0029
Epoch 5/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5294 -
accuracy: 0.0039 - val_loss: 5.5296 - val_accuracy: 0.0037
Epoch 6/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5295 -
accuracy: 0.0041 - val_loss: 5.5300 - val_accuracy: 0.0035
Epoch 7/10
```

```
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5292 -
accuracy: 0.0039 - val_loss: 5.5313 - val_accuracy: 0.0036
Epoch 8/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5296 -
accuracy: 0.0041 - val_loss: 5.5280 - val_accuracy: 0.0039
Epoch 9/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5293 -
accuracy: 0.0039 - val_loss: 5.5305 - val_accuracy: 0.0029
Epoch 10/10
2362/2362 [==============================] - 10s 4ms/step - loss: 5.5299 -
accuracy: 0.0039 - val_loss: 5.5299 - val_accuracy: 0.0035
```



[ ]:

EDA_Bin

December 11, 2023

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
!pip install -q pandas pyarrow
!pip install -q mediapipe
```

```
                                         34.5/34.5 MB
45.3 MB/s eta 0:00:00
```

[ ]:
```
# Data Folder Directry
main_dir = '/content/drive/MyDrive/Colab Notebooks/Data/asl-signs/'
```

[ ]:
```
import pandas as pd
import os

metadata_sub_dir = 'train.csv'
metadata_full_file_path = os.path.join(main_dir, metadata_sub_dir)
df_metadata = pd.read_csv(metadata_full_file_path)
# Read the .parquet file
#df = pd.read_parquet(file_path)
df_metadata
```

[ ]:
```
                                                path  participant_id  \
0       train_landmark_files/26734/1000035562.parquet           26734
1       train_landmark_files/28656/1000106739.parquet           28656
2        train_landmark_files/16069/100015657.parquet           16069
3       train_landmark_files/25571/1000210073.parquet           25571
4       train_landmark_files/62590/1000240708.parquet           62590
…                                                 …              …
94472    train_landmark_files/53618/999786174.parquet           53618
94473    train_landmark_files/26734/999799849.parquet           26734
94474    train_landmark_files/25571/999833418.parquet           25571
94475    train_landmark_files/29302/999895257.parquet           29302
94476    train_landmark_files/36257/999962374.parquet           36257

       sequence_id    sign
```

```
0        1000035562    blow
1        1000106739    wait
2         100015657   cloud
3        1000210073    bird
4        1000240708    owie
…               …      …
94472     999786174   white
94473     999799849    have
94474     999833418  flower
94475     999895257    room
94476     999962374   happy

[94477 rows x 4 columns]
```

[ ]:
```python
N_SAMPLES = len(df_metadata)
```

[ ]:
```python
import json

signmap_sub_dir = 'sign_to_prediction_index_map.json'
signmap_full_file_path = os.path.join(main_dir, signmap_sub_dir)

# Load the sign to index mapping
with open(signmap_full_file_path, 'r') as file:
    sign_to_index = json.load(file)

# Map the labels in the dataframe
df_metadata['sign_index'] = df_metadata['sign'].map(sign_to_index)
```

[ ]:
```python
df_metadata
```

[ ]:
```
                                                path  participant_id  \
0      train_landmark_files/26734/1000035562.parquet           26734
1      train_landmark_files/28656/1000106739.parquet           28656
2       train_landmark_files/16069/100015657.parquet           16069
3      train_landmark_files/25571/1000210073.parquet           25571
4      train_landmark_files/62590/1000240708.parquet           62590
…                                                …             …
94472   train_landmark_files/53618/999786174.parquet           53618
94473   train_landmark_files/26734/999799849.parquet           26734
94474   train_landmark_files/25571/999833418.parquet           25571
94475   train_landmark_files/29302/999895257.parquet           29302
94476   train_landmark_files/36257/999962374.parquet           36257

       sequence_id    sign  sign_index
0        1000035562    blow          25
1        1000106739    wait         232
2         100015657   cloud          48
```

```
3       1000210073    bird          23
4       1000240708    owie         164
…              …      …          …
94472    999786174   white         238
94473    999799849    have         108
94474    999833418  flower          86
94475    999895257    room         188
94476    999962374   happy         105

[94477 rows x 5 columns]
```

[ ]:
```
samplefile_dir = df_metadata['path'][0]
samplefile_full_file_path = os.path.join(main_dir, samplefile_dir)
print(samplefile_full_file_path)

# Read the .parquet file
df_samplefile = pd.read_parquet(samplefile_full_file_path)
df_samplefile
```

```
/content/drive/MyDrive/Colab Notebooks/Data/asl-
signs/train_landmark_files/26734/1000035562.parquet
```

[ ]:
```
       frame           row_id        type  landmark_index         x  \
0         20         20-face-0        face               0  0.494400
1         20         20-face-1        face               1  0.496017
2         20         20-face-2        face               2  0.500818
3         20         20-face-3        face               3  0.489788
4         20         20-face-4        face               4  0.495304
…      …                 …         …              …        …
12484     42  42-right_hand-16  right_hand              16  0.001660
12485     42  42-right_hand-17  right_hand              17  0.042694
12486     42  42-right_hand-18  right_hand              18  0.006723
12487     42  42-right_hand-19  right_hand              19 -0.014755
12488     42  42-right_hand-20  right_hand              20 -0.031811

              y         z
0      0.380470 -0.030626
1      0.350735 -0.057565
2      0.359343 -0.030283
3      0.321780 -0.040622
4      0.341821 -0.061152
…         …        …
12484  0.549574 -0.145409
12485  0.693116 -0.085307
12486  0.665044 -0.114017
12487  0.643799 -0.123488
12488  0.627077 -0.129067
```

```
[12489 rows x 7 columns]
```

[ ]:
```
df_samplefile['type'].unique()
```

[ ]:
```
array(['face', 'left_hand', 'pose', 'right_hand'], dtype=object)
```

[ ]:
```
df_samplefile['frame'].unique()
```

[ ]:
```
array([20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
       37, 38, 39, 40, 41, 42], dtype=int16)
```

[ ]:
```
df_singleframe = df_samplefile[df_samplefile['frame']==20]
df_singleframe
```

[ ]:
```
     frame           row_id        type  landmark_index         x         y  \
0       20        20-face-0        face               0  0.494400  0.380470
1       20        20-face-1        face               1  0.496017  0.350735
2       20        20-face-2        face               2  0.500818  0.359343
3       20        20-face-3        face               3  0.489788  0.321780
4       20        20-face-4        face               4  0.495304  0.341821
..     …              …          …             …         …         …
538     20  20-right_hand-16  right_hand             16  0.422241  0.390434
539     20  20-right_hand-17  right_hand             17  0.282980  0.457257
540     20  20-right_hand-18  right_hand             18  0.313736  0.412344
541     20  20-right_hand-19  right_hand             19  0.350728  0.399582
542     20  20-right_hand-20  right_hand             20  0.385796  0.401101

            z
0   -0.030626
1   -0.057565
2   -0.030283
3   -0.040622
4   -0.061152
..        …
538 -0.049388
539 -0.038326
540 -0.052699
541 -0.060217
542 -0.064718

[543 rows x 7 columns]
```

[ ]:
```
df_singleframe['type'][522]
```

[ ]:
```
'right_hand'
```

[ ]:
```
df_singleframe[df_singleframe['type']=='left_hand']
```

[ ]:

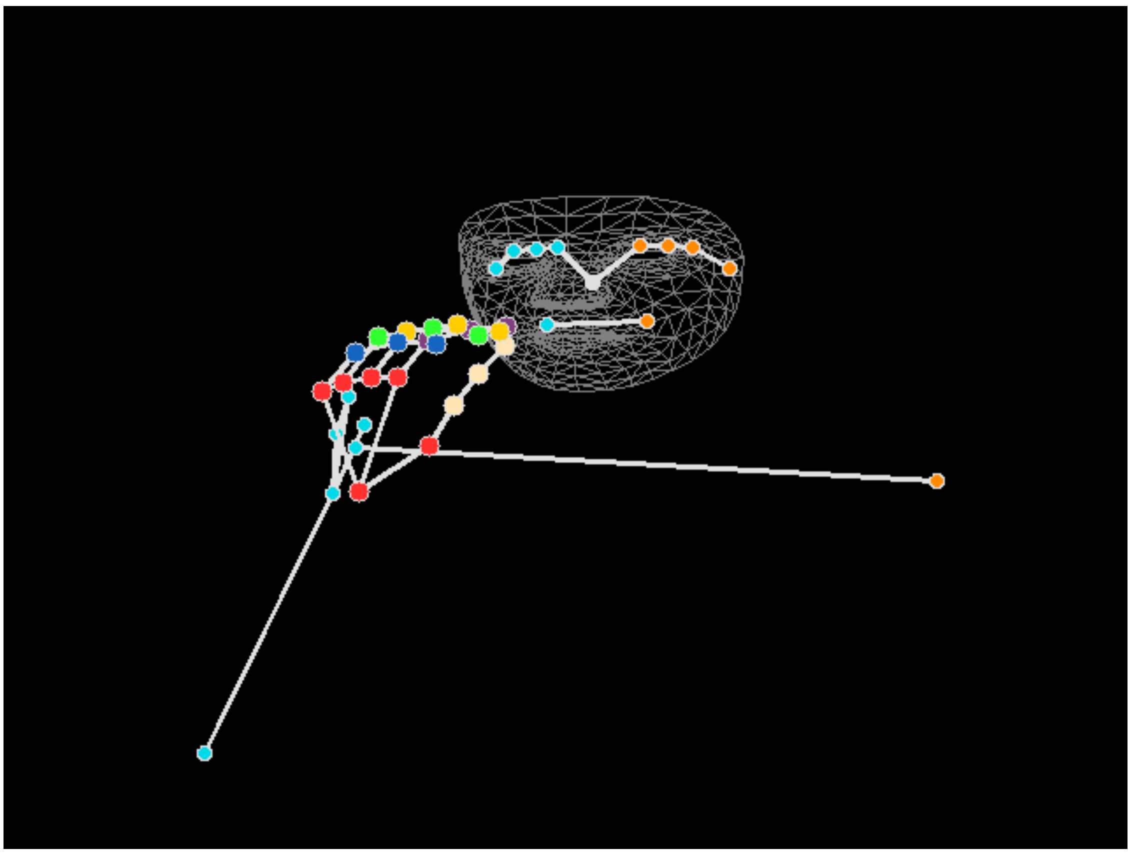
```
     frame            row_id       type  landmark_index    x    y    z
468     20    20-left_hand-0  left_hand               0  NaN  NaN  NaN
469     20    20-left_hand-1  left_hand               1  NaN  NaN  NaN
470     20    20-left_hand-2  left_hand               2  NaN  NaN  NaN
471     20    20-left_hand-3  left_hand               3  NaN  NaN  NaN
472     20    20-left_hand-4  left_hand               4  NaN  NaN  NaN
473     20    20-left_hand-5  left_hand               5  NaN  NaN  NaN
474     20    20-left_hand-6  left_hand               6  NaN  NaN  NaN
475     20    20-left_hand-7  left_hand               7  NaN  NaN  NaN
476     20    20-left_hand-8  left_hand               8  NaN  NaN  NaN
477     20    20-left_hand-9  left_hand               9  NaN  NaN  NaN
478     20   20-left_hand-10  left_hand              10  NaN  NaN  NaN
479     20   20-left_hand-11  left_hand              11  NaN  NaN  NaN
480     20   20-left_hand-12  left_hand              12  NaN  NaN  NaN
481     20   20-left_hand-13  left_hand              13  NaN  NaN  NaN
482     20   20-left_hand-14  left_hand              14  NaN  NaN  NaN
483     20   20-left_hand-15  left_hand              15  NaN  NaN  NaN
484     20   20-left_hand-16  left_hand              16  NaN  NaN  NaN
485     20   20-left_hand-17  left_hand              17  NaN  NaN  NaN
486     20   20-left_hand-18  left_hand              18  NaN  NaN  NaN
487     20   20-left_hand-19  left_hand              19  NaN  NaN  NaN
488     20   20-left_hand-20  left_hand              20  NaN  NaN  NaN
```

[ ]:

```python
import cv2
import mediapipe as mp
import pandas as pd
import numpy as np
from google.colab.patches import cv2_imshow
from mediapipe.framework.formats import landmark_pb2

# Initialize MediaPipe solutions
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_face_mesh = mp.solutions.face_mesh
mp_pose = mp.solutions.pose
mp_hands = mp.solutions.hands  # Add this line for hand landmarks

# Load the landmark data (replace this with your actual file path)
df_landmark = df_samplefile

# Create a black image
image_height, image_width = 480, 640
image = np.zeros((image_height, image_width, 3), dtype=np.uint8)

# Function to draw landmarks using MediaPipe's utility
def draw_mediapipe_landmarks(image, df, landmark_type):
    # Convert DataFrame to MediaPipe Landmark list
```

```
    landmarks = []
    for _, row in df.iterrows():
        if pd.isna(row['x']) or pd.isna(row['y']):
            continue
        landmark = landmark_pb2.NormalizedLandmark(
            x=row['x'], y=row['y'], z=row.get('z', 0))
        landmarks.append(landmark)

    landmark_list = landmark_pb2.NormalizedLandmarkList(
        landmark=landmarks)

    # Draw landmarks
    if landmark_type == 'face':
        mp_drawing.draw_landmarks(
            image, landmark_list,
            mp_face_mesh.FACEMESH_TESSELATION,
            landmark_drawing_spec=None,
            connection_drawing_spec=mp_drawing_styles.
↪get_default_face_mesh_tesselation_style())
    elif landmark_type == 'pose':
        mp_drawing.draw_landmarks(
            image, landmark_list,
            mp_pose.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.
↪get_default_pose_landmarks_style())
    elif landmark_type == 'right_hand':
        mp_drawing.draw_landmarks(
            image, landmark_list,
            mp_hands.HAND_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.
↪get_default_hand_landmarks_style())
    elif landmark_type == 'left_hand':
        mp_drawing.draw_landmarks(
            image, landmark_list,
            mp_hands.HAND_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.
↪get_default_hand_landmarks_style())

# Draw landmarks for a specific frame and type
frame_number = 20  # Example frame number
df_frame = df_landmark[df_landmark['frame'] == frame_number]

# Example: Drawing face landmarks
draw_mediapipe_landmarks(image, df_frame[df_frame['type'] == 'face'], 'face')

# Example: Drawing pose landmarks
draw_mediapipe_landmarks(image, df_frame[df_frame['type'] == 'pose'], 'pose')
```

```
# Example: Drawing left hand landmarks
#draw_mediapipe_landmarks(image, df_frame[df_frame['type'] == 'left_hand'],␣
↪'left_hand')

# Example: Drawing right hand landmarks
draw_mediapipe_landmarks(image, df_frame[df_frame['type'] == 'right_hand'],␣
↪'right_hand')
#draw_mediapipe_landmarks(image, df_frame[df_frame['type'] == 'left_hand'],␣
↪'left_hand')
# Display the image
cv2_imshow(image)
```

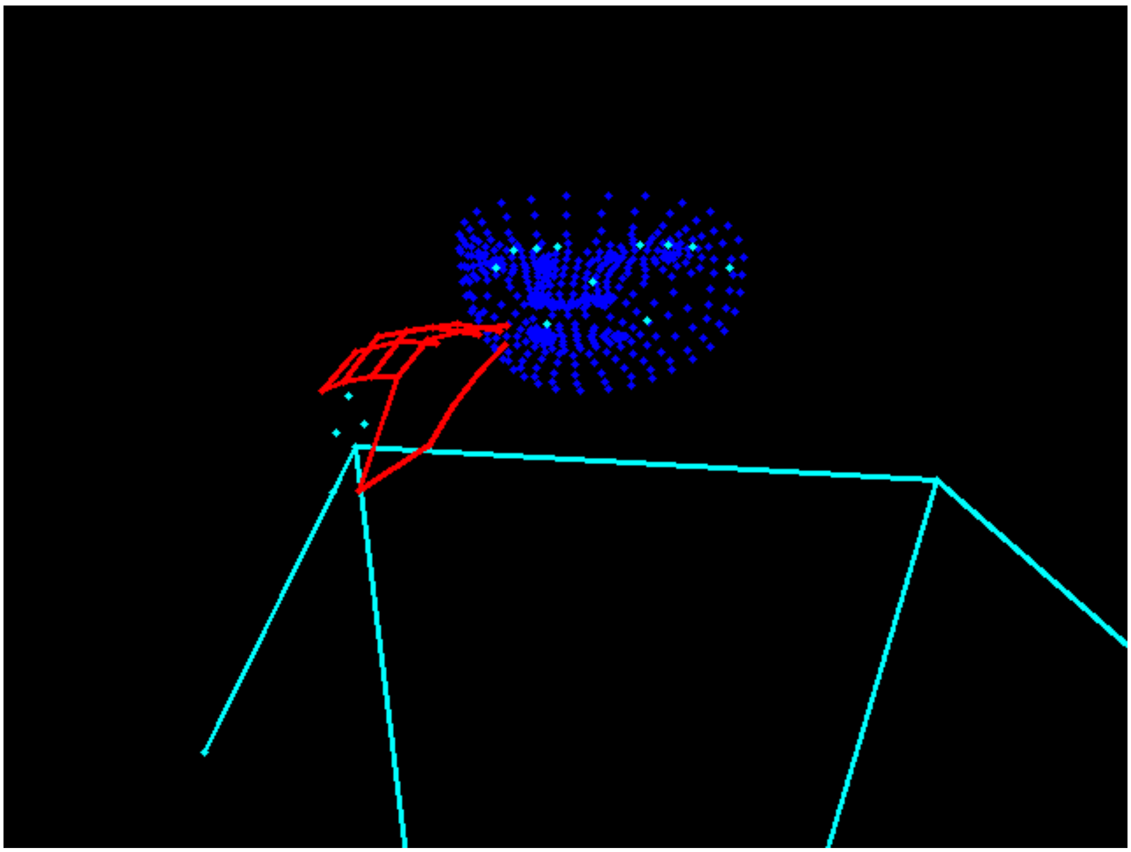


[ ]:
```
import cv2
import pandas as pd
import numpy as np
from google.colab.patches import cv2_imshow

# Load the landmark data
df_landmark = df_samplefile
```

```python
# Create a black image
image_height, image_width = 480, 640
image = np.zeros((image_height, image_width, 3), dtype=np.uint8)

# Define connections for face, pose, and hands
# Define the face connections here
#"""
FACE_CONNECTIONS = [
    # Face oval
    #*(list(zip(range(0, 151), range(1, 152))) + [(151, 0)]),

    # Eyebrows
    #*list(zip(range(152, 157), range(153, 158))), # Right eyebrow
    #*list(zip(range(158, 163), range(159, 164))), # Left eyebrow

    # Eyes
    #*list(zip(range(133, 141), range(134, 142))) + [(141, 133)], # Right eye
    #*list(zip(range(362, 370), range(363, 371))) + [(370, 362)], # Left eye

    # Lips (outer and inner)
    #*list(zip(range(61, 67), range(62, 68))) + [(67, 61)], # Outer top lip
    #*list(zip(range(146, 152), range(147, 153))) + [(152, 146)], # Outer bottom lip
    #*list(zip(range(78, 82), range(79, 83))) + [(82, 78)], # Inner top lip
    #*list(zip(range(87, 91), range(88, 92))) + [(91, 87)], # Inner bottom lip

    # Nose
    #*list(zip(range(234, 238), range(235, 239))), # Nose bridge
    #*list(zip(range(308, 314), range(309, 315)))  # Lower nose
]
#"""
#FACE_CONNECTIONS = []
# Define the pose connections here
POSE_CONNECTIONS = [
    # Torso
    (11, 12), (11, 23), (12, 24), (23, 24),

    # Arms
    (11, 13), (13, 15), (12, 14), (14, 16),

    # Legs
    (23, 25), (25, 27), (27, 31), (24, 26), (26, 28), (28, 32),

    # Shoulders to hips
    (11, 23), (12, 24)
]
```

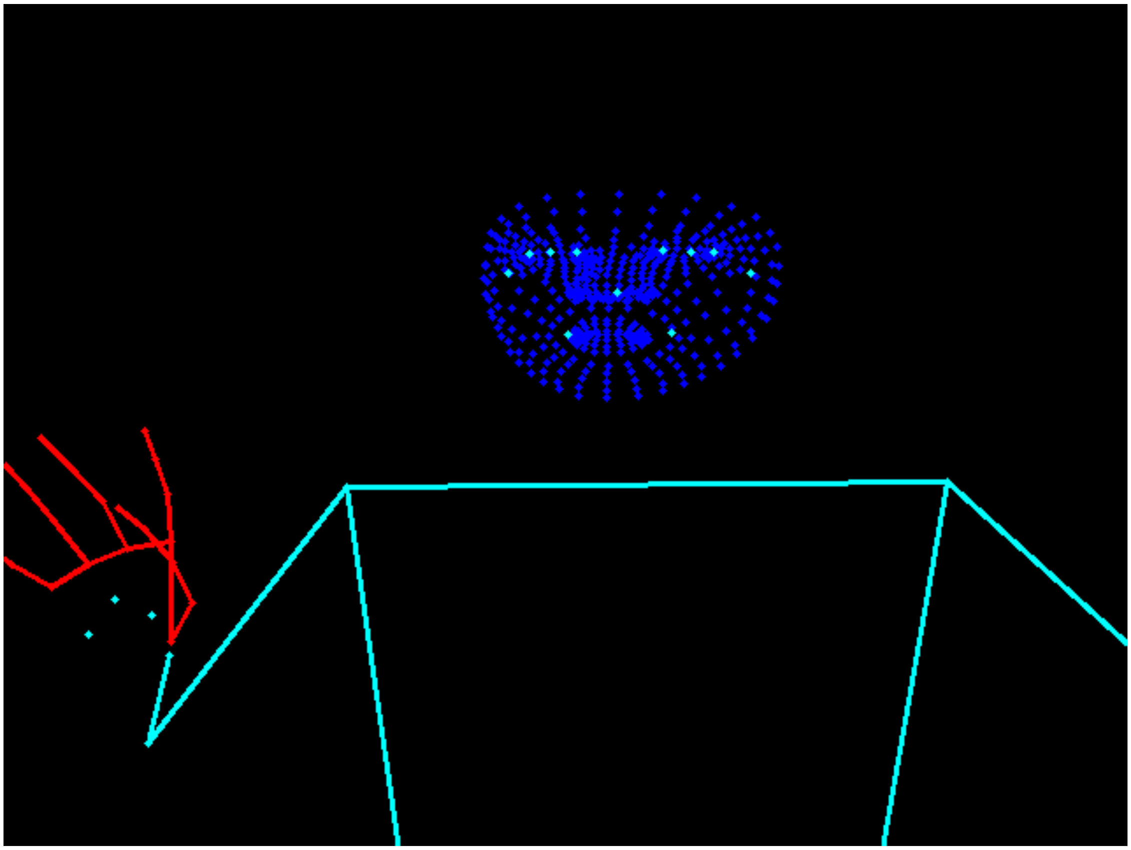
```python
# Hand connections based on MediaPipe hand landmark model
HAND_CONNECTIONS = [
    (0, 1), (1, 2), (2, 3), (3, 4),          # Thumb
    (0, 5), (5, 6), (6, 7), (7, 8),          # Index finger
    (5, 9), (9, 10), (10, 11), (11, 12),     # Middle finger
    (9, 13), (13, 14), (14, 15), (15, 16),   # Ring finger
    (13, 17), (17, 18), (18, 19), (19, 20)   # Little finger
]

def draw_landmarks(image, df):
    colors = {
        'face': (255, 0, 0),
        'left_hand': (0, 255, 0),
        'right_hand': (0, 0, 255),
        'pose': (255, 255, 0)
    }

    grouped = df.groupby('type')

    for group_name, group_df in grouped:
        connections = None
        if group_name == 'face':
            connections = FACE_CONNECTIONS
        elif group_name == 'pose':
            connections = POSE_CONNECTIONS
        elif group_name in ['left_hand', 'right_hand']:
            connections = HAND_CONNECTIONS

        if connections:
            for connection in connections:
                pt1 = group_df[group_df['landmark_index'] == connection[0]].
 ↪iloc[0]
                pt2 = group_df[group_df['landmark_index'] == connection[1]].
 ↪iloc[0]

                if not (pd.isna(pt1['x']) or pd.isna(pt1['y']) or pd.
 ↪isna(pt2['x']) or pd.isna(pt2['y'])):
                    x1, y1 = int(pt1['x'] * image_width), int(pt1['y'] *␣
 ↪image_height)
                    x2, y2 = int(pt2['x'] * image_width), int(pt2['y'] *␣
 ↪image_height)
                    cv2.line(image, (x1, y1), (x2, y2), colors[group_name], 2)

        # Draw landmarks
        for _, row in group_df.iterrows():
            if pd.isna(row['x']) or pd.isna(row['y']):
                continue
```

```
            x, y = int(row['x'] * image_width), int(row['y'] * image_height)
            color = colors.get(group_name, (255, 255, 255))
            cv2.circle(image, (x, y), 2, color, -1)

# Draw landmarks for a specific frame
frame_number = 20
df_frame = df_landmark[df_landmark['frame'] == frame_number]
draw_landmarks(image, df_frame)

# Display the image
cv2_imshow(image)
```



[ ]:
```
# Draw landmarks for a specific frame
image = np.zeros((image_height, image_width, 3), dtype=np.uint8)
frame_number = 42
df_frame = df_landmark[df_landmark['frame'] == frame_number]
draw_landmarks(image, df_frame)

# Display the image
```

```
cv2_imshow(image)
```



During our study of the data and research on the possible model solutions, there is one transformer model approach caught our eye. This transformer model approach was designed by Wijkhuizen, M., in the Kaggle competition (2023). Our project team decided to follow Wijkhuizen, M.'s approach to create a transformer model as one of the models to test for this project. Our goal with this approach is to get a better understanding of the transformer model since Wijkhuizen, M.'s approach is to build a transformer model from scratch and not fine-turn a base model.

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
  ↪gislr-tf-data-processing-transformer-training
import pandas as pd
import numpy as np
from tqdm import tqdm
SEED=1234
# Assuming df_metadata is already defined
N = min(1000, len(df_metadata))  # Sample size, adjust as needed
print(N)

# Arrays to store analysis results
N_UNIQUE_FRAMES = np.zeros(N, dtype=np.uint16)
```

```
N_MISSING_FRAMES = np.zeros(N, dtype=np.uint16)
MAX_FRAME = np.zeros(N, dtype=np.uint16)

# Sample a subset of the dataset for analysis
sampled_metadata = df_metadata.sample(N, random_state=SEED)

# Loop over the sampled metadata
for idx, (_, row) in enumerate(tqdm(sampled_metadata.iterrows(), total=N)):
    # Load the landmark data
    samplefile_dir = row['path']
    samplefile_full_file_path = os.path.join(main_dir, samplefile_dir)
    df_landmark = pd.read_parquet(samplefile_full_file_path)

    # Analysis of frames
    N_UNIQUE_FRAMES[idx] = df_landmark['frame'].nunique()
    N_MISSING_FRAMES[idx] = (df_landmark['frame'].max() - df_landmark['frame'].
 ↪min()) - df_landmark['frame'].nunique() + 1
    MAX_FRAME[idx] = df_landmark['frame'].max()

# Printing the first elements for inspection
print(N_UNIQUE_FRAMES[0], N_MISSING_FRAMES[0], MAX_FRAME[0])
```

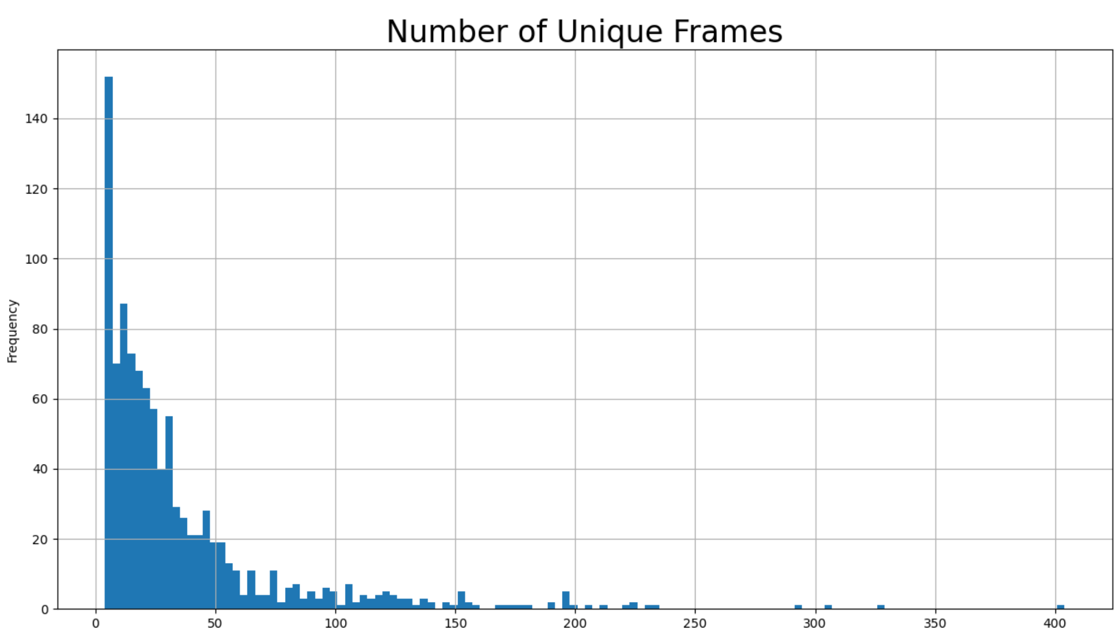
```
1000

100%|      | 1000/1000 [14:40<00:00,  1.14it/s]

109 0 148
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
 ↪gislr-tf-data-processing-transformer-training
import matplotlib.pyplot as plt
PERCENTILES = [0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99, 0.999]
# Number of unique frames in each video
display(pd.Series(N_UNIQUE_FRAMES).describe(percentiles=PERCENTILES).
 ↪to_frame('N_UNIQUE_FRAMES'))

plt.figure(figsize=(15,8))
plt.title('Number of Unique Frames', size=24)
pd.Series(N_UNIQUE_FRAMES).plot(kind='hist', bins=128)
plt.grid()
xlim = math.ceil(plt.xlim()[1])
plt.xlim(0, xlim)
plt.xticks(np.arange(0, xlim+25, 25))
plt.show()
```

```
       N_UNIQUE_FRAMES
count      1000.000000
```

```
mean          36.253000
std           42.776054
min            4.000000
1%             6.000000
5%             6.000000
25%           11.000000
50%           22.000000
75%           42.000000
95%          123.000000
99%          206.070000
99.9%        328.076000
max          404.000000
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-2709d42446f0> in <cell line: 10>()
      8 pd.Series(N_UNIQUE_FRAMES).plot(kind='hist', bins=128)
      9 plt.grid()
---> 10 xlim = math.ceil(plt.xlim()[1])
     11 plt.xlim(0, xlim)
     12 plt.xticks(np.arange(0, xlim+25, 25))

NameError: name 'math' is not defined
```



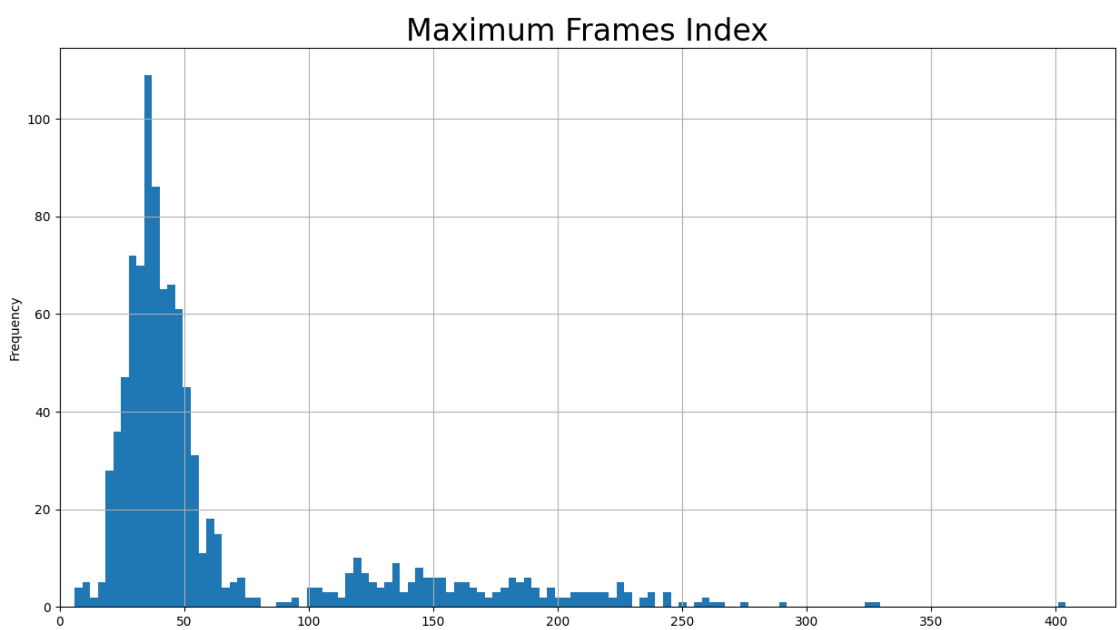
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
```

```python
import math
# Number of missing frames, consecutive frames with missing intermediate frame,␣
↪i.e. 1,2,4,5 -> 3 is missing
display(pd.Series(N_MISSING_FRAMES).describe(percentiles=PERCENTILES).
↪to_frame('N_MISSING_FRAMES'))

plt.figure(figsize=(15,8))
plt.title('Number of Missing Frames', size=24)
pd.Series(N_MISSING_FRAMES).plot(kind='hist', bins=128)
plt.grid()
plt.xlim(0, math.ceil(plt.xlim()[1]))
plt.show()
```

	N_MISSING_FRAMES
count	1000.0
mean	0.0
std	0.0
min	0.0
1%	0.0
5%	0.0
25%	0.0
50%	0.0
75%	0.0
95%	0.0
99%	0.0
99.9%	0.0
max	0.0

```
[ ]: # Code From https://www.kaggle.com/code/markwijkhuizen/
     ↪gislr-tf-data-processing-transformer-training
     # Maximum frame number
     display(pd.Series(MAX_FRAME).describe(percentiles=PERCENTILES).
      ↪to_frame('MAX_FRAME'))

     plt.figure(figsize=(15,8))
     plt.title('Maximum Frames Index', size=24)
     pd.Series(MAX_FRAME).plot(kind='hist', bins=128)
     plt.grid()
     plt.xlim(0, math.ceil(plt.xlim()[1]))
     plt.show()
```

	MAX_FRAME
count	1000.000000
mean	65.221000
std	57.220559
min	6.000000
1%	15.000000
5%	22.000000
25%	33.000000
50%	42.000000
75%	60.000000
95%	196.000000
99%	249.070000
99.9%	327.077000
max	404.000000



Reference:

Wijkhuizen, M. (2023, April 04). GISLR TF Data Processing & Transformer Training. Kaggle. https://www.kaggle.com/code/markwijkhuizen/gislr-tf-data-processing-transformer-training

Sign_Language_Detection_Project_Eyoha

December 11, 2023

#Sign Language Detection

##Abstract This project focuses on developing a deep learning model to accurately predict and classify hand gestures, representing words, and numbers in American Sign Language. With a primary aim to facilitate communication with deaf children, especially those born to hearing parents unfamiliar with ASL, the project leverages a comprehensive dataset comprising hand landmarks extracted from video frames. Utilizing advanced image processing techniques and machine learning algorithms, the model interprets hand positions, movements, and finger configurations to translate sign language into text. The goal of this project is to learn more aboout CV and deep learning techneques in the context of ASL while creating a an effective solution for people who use ASL.

##Introduction Background Communication barriers between deaf individuals and those unfamiliar with sign language pose significant challenges. Particularly, deaf children born to hearing parents often face communication gaps, as many parents do not initially know sign language. This project aims to bridge this gap by leveraging technology to translate American Sign Language into text, thus aiding parents, educators, and caregivers in learning and interacting more effectively with deaf children.

Objectives The primary objective of this project is to develop a deep learning-based model capable of accurately detecting and classifying ASL signs. The model will interpret hand gestures, including the position, movement, and orientation of hands and fingers, to translate these into corresponding textual representations.

Dataset and Methods The dataset for this project is sourced from a Kaggle competition, comprising landmark data extracted from videos using the MediaPipe holistic model. This data includes normalized spatial coordinates for hand landmarks, which are the critical features for model training. The project will employ various image processing techniques, such as edge detection and finger positioning analysis, alongside deep learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to interpret these signs. The model's performance will be evaluated using accuracy, precision, recall, and F1-score metrics, and the final model will be converted into TensorFlow Lite format for practical deployment.

Step 1: Setup Environment and Dependencies

[ ]:
```
!pip install tensorflow numpy pandas matplotlib scikit-learn opencv-python
```

[ ]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import pyarrow.parquet as pq
import cv2
import os
from sklearn.model_selection import train_test_split
```

Step 2: Data Acquisition and Loading

[ ]:
```python
from google.colab import files
files.upload()  # This will allow us to upload the kaggle.json file
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

[ ]:
```
{'kaggle.json':
 b'{"username":"yoha00","key":"009129d68ea830c0102186e43b0dd39f"}'}
```

[ ]:
```python
# setting up kaggle environment
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

[ ]:
```python
# now we use the kaggle API to download the asl dataset
!kaggle competitions download -c asl-signs
```

```
Downloading asl-signs.zip to /content
100% 37.4G/37.4G [21:21<00:00, 39.4MB/s]
100% 37.4G/37.4G [21:21<00:00, 31.3MB/s]
```

[ ]:
```python
# Unzip the downloaded file
!unzip asl-signs.zip
```

[ ]:
```python
#looking at the file structure
!ls
```

```
asl-signs.zip  sample_data                     train.csv
kaggle.json    sign_to_prediction_index_map.json  train_landmark_files
```

[ ]:
```python
# Taking a look at the train.csv
df = pd.read_csv('/content/train.csv')
df.head()
```

[ ]:
```
                                             path  participant_id  sequence_id  \
0  train_landmark_files/26734/1000035562.parquet           26734   1000035562
1  train_landmark_files/28656/1000106739.parquet           28656   1000106739
2   train_landmark_files/16069/100015657.parquet           16069    100015657
3  train_landmark_files/25571/1000210073.parquet           25571   1000210073
4  train_landmark_files/62590/1000240708.parquet           62590   1000240708

    sign
```

```
0   blow
1   wait
2  cloud
3   bird
4   owie
```

###Evaluation

Based on the structure above it looks like we have a number of participants in the data set who have each taken part in signing a word. Denoted by their participant ID.

We can also see that there are multiple signes in the data set that are denoted by their sign ID each corriponding to a participant_id and a parquet file for the landmark data.

[ ]:
```
# Taking a look at the sign_to_prediction_index_map.json file
df = pd.read_csv('/content/sign_to_prediction_index_map.json')
df.head()
```

[ ]:
```
Empty DataFrame
Columns: [{"TV": 0,  "after": 1,  "airplane": 2,  "all": 3,  "alligator": 4,
"animal": 5,  "another": 6,  "any": 7,  "apple": 8,  "arm": 9,  "aunt": 10,
"awake": 11,  "backyard": 12,  "bad": 13,  "balloon": 14,  "bath": 15,
"because": 16,  "bed": 17,  "bedroom": 18,  "bee": 19,  "before": 20,  "beside":
21,  "better": 22,  "bird": 23,  "black": 24,  "blow": 25,  "blue": 26,  "boat":
27,  "book": 28,  "boy": 29,  "brother": 30,  "brown": 31,  "bug": 32,  "bye":
33,  "callonphone": 34,  "can": 35,  "car": 36,  "carrot": 37,  "cat": 38,
"cereal": 39,  "chair": 40,  "cheek": 41,  "child": 42,  "chin": 43,
"chocolate": 44,  "clean": 45,  "close": 46,  "closet": 47,  "cloud": 48,
"clown": 49,  "cow": 50,  "cowboy": 51,  "cry": 52,  "cut": 53,  "cute": 54,
"dad": 55,  "dance": 56,  "dirty": 57,  "dog": 58,  "doll": 59,  "donkey": 60,
"down": 61,  "drawer": 62,  "drink": 63,  "drop": 64,  "dry": 65,  "dryer": 66,
"duck": 67,  "ear": 68,  "elephant": 69,  "empty": 70,  "every": 71,  "eye": 72,
"face": 73,  "fall": 74,  "farm": 75,  "fast": 76,  "feet": 77,  "find": 78,
"fine": 79,  "finger": 80,  "finish": 81,  "fireman": 82,  "first": 83,  "fish":
84,  "flag": 85,  "flower": 86,  "food": 87,  "for": 88,  "frenchfries": 89,
"frog": 90,  "garbage": 91,  "gift": 92,  "giraffe": 93,  "girl": 94,  "give":
95,  "glasswindow": 96,  "go": 97,  "goose": 98,  "grandma": 99, …]
Index: []

[0 rows x 250 columns]
```

[ ]:
```
#After unzipping we have a folder named train_landmark_files in our current␣
↪directory with the following files
!ls '/content/train_landmark_files'
```

```
16069   2044    25571   27610   29302   32319   36257   37779   49445   55372   62590
18796   22343   26734   28656   30680   34503   37055   4718    53618   61333
```

[ ]:
```
#Example of file count in one of our folders
!ls -l '/content/train_landmark_files/16069' | wc -l
!ls -l '/content/train_landmark_files/18796' | wc -l
```

```
4849
3503
```

[ ]:
```
#Based in vusila inspcetion it seems like all of the folders have the same data␣
↪type a Parquet file
#Let's look at one of the files
!file "/content/train_landmark_files/16069/100015657.parquet"
```

```
/content/train_landmark_files/16069/100015657.parquet: Apache Parquet
```

#Exploratory Data Analysis

For the Exploratory Data Analysis, we'll focus on understanding the train.csv file's contents and characteristics.

> Basic Descriptive Statistics: This includes counts, means, and other statistical measures that give a quick overview of the data.
>
> Missing Values Check: To ensure the integrity of the dataset, we will check for any missing values.
>
> Visualization of Sign Distribution: A visual representation (such as a histogram or bar chart) to show the distribution of different signs in the dataset.

Additionally, we'll analyze:

> The number of unique signs in the dataset.
>
> The number of unique participants.
>
> The distribution of the number of .parquet files (landmark files) per sign, focusing on the top 20 signs with the most files

[ ]:
```
# Lets start by creating our data frame

# Load the train.csv file into a DataFrame
train_csv_path = '/content/train.csv'
train_df = pd.read_csv(train_csv_path)
```

[ ]:
```
#Show basic Descriptive Statistics
basic_stats = train_df.describe(include='all')
basic_stats
```

[ ]:
```
                                                path  participant_id  \
count                                          94477    94477.000000
unique                                         94477             NaN
top     train_landmark_files/26734/1000035562.parquet             NaN
freq                                               1             NaN
mean                                             NaN    33678.632366
```

```
std                                                   NaN    16138.124387
min                                                   NaN     2044.000000
25%                                                   NaN    25571.000000
50%                                                   NaN    32319.000000
75%                                                   NaN    49445.000000
max                                                   NaN    62590.000000

         sequence_id    sign
count   9.447700e+04   94477
unique           NaN     250
top              NaN  listen
freq             NaN     415
mean    2.149377e+09     NaN
std     1.239239e+09     NaN
min     8.528200e+04     NaN
25%     1.078076e+09     NaN
50%     2.154240e+09     NaN
75%     3.218820e+09     NaN
max     4.294915e+09     NaN
```

[ ]:
```
# Number of Unique Signs in the data set
unique_signs_count = train_df['sign'].nunique()
unique_signs_count
```

[ ]: 250

[ ]:
```
#  Next we calculate the number of unique participants in the dataset.
unique_participants_count = train_df['participant_id'].nunique()
unique_participants_count
```

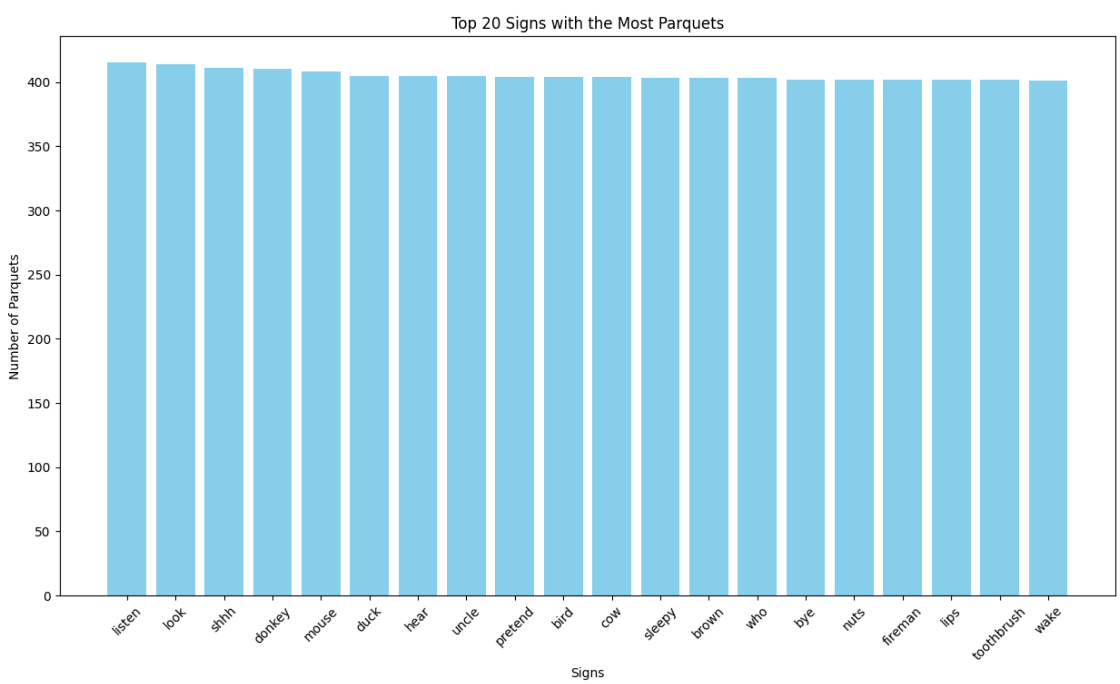
[ ]: 21

[ ]:
```
# To understand which signs have the most data points, we look at the␣
↪distribution of .parquet files per sign.
parquets_per_sign = train_df['sign'].value_counts().head(20)
parquets_per_sign
```

[ ]:
```
listen        415
look          414
shhh          411
donkey        410
mouse         408
duck          405
hear          405
uncle         405
pretend       404
bird          404
```

```
cow           404
sleepy        403
brown         403
who           403
bye           402
nuts          402
fireman       402
lips          402
toothbrush    402
wake          401
Name: sign, dtype: int64
```

[ ]:
```
# To better visualize this we plot the distribution the top 20 signs

# Preparing data for visualization
top_signs = parquets_per_sign.index.tolist()
top_signs_counts = parquets_per_sign.values.tolist()

# Visualization code
plt.figure(figsize=(15, 8))
plt.bar(top_signs, top_signs_counts, color='skyblue')
plt.xlabel('Signs')
plt.ylabel('Number of Parquets')
plt.title('Top 20 Signs with the Most Parquets')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```python
# Final lets check for missing values
missing_values = train_df.isnull().sum()
missing_values
```

[ ]:
```
path              0
participant_id    0
sequence_id       0
sign              0
dtype: int64
```

TODO add more analysis for EDA in relation to parquet files ##Data pre-processing

next step is data pre-processing. This phase involves preparing the .parquet files in the train_landmark_files folder, which contain the landmark data for the sign language gestures, for model training.

[ ]:
```python
# Reading the .praquet files

# Example of reading a single .parquet file
file_path = 'train_landmark_files/16069/100015657.parquet'  # Replace with␣
 ↪actual file path
parquet_data = pd.read_parquet(file_path)
parquet_data.head()
parquet_data.shape
```

[ ]:
```
(57015, 7)
```

[ ]:
```python
# Now lets read all of the .parquet files in the train_landmark_files folder
# We'll save the processed files in a log file incase we need to restart the␣
 ↪process
# We'll use the log file to skip files that have already been processed
# We'll save prgress of our process in a batch file every 2000 files

import os
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import datetime

def read_log(log_path):
    if os.path.exists(log_path):
        with open(log_path, 'r') as file:
            processed_files = file.read().splitlines()
        return set(processed_files)
    else:
        return set()
```

```
def update_log(log_path, file_name):
    with open(log_path, 'a') as file:
        file.write(file_name + '\n')

root_folder = '/content/train_landmark_files'
scaler = MinMaxScaler()
batch_size = 2000
log_path = 'processed_files_log.txt'
processed_files = read_log(log_path)

for subfolder in os.listdir(root_folder):
    subfolder_path = os.path.join(root_folder, subfolder)
    if os.path.isdir(subfolder_path):
        all_data = pd.DataFrame()
        file_count = 0

        for file in os.listdir(subfolder_path):
            if file in processed_files:
                continue  # Skip if the file is already processed

            file_path = os.path.join(subfolder_path, file)
            parquet_data = pd.read_parquet(file_path)
            data_of_interest = parquet_data[parquet_data['type'].
↪isin(['left_hand', 'right_hand', 'pose'])].copy()
            data_of_interest[['x', 'y', 'z']] = scaler.
↪fit_transform(data_of_interest[['x', 'y', 'z']])
            sequence_id = int(file.split('.')[0])
            data_of_interest['sequence_id'] = sequence_id
            all_data = pd.concat([all_data, data_of_interest],␣
↪ignore_index=True)

            update_log(log_path, file)  # Update the log
            file_count += 1
            if file_count >= batch_size:
                break

        batch_filename = f'batch_{len(processed_files)//batch_size}.csv'
        all_data.to_csv(batch_filename, index=False)
        print(f"Batch {len(processed_files)//batch_size} saved. Timestamp:␣
↪{datetime.datetime.now()}")
```

[ ]:
```
# prompt: how check csv file

import pandas as pd

# Read the CSV file
df = pd.read_csv('/content/batch_0 (2).csv')
```

```python
# Check the shape of the DataFrame
print('\n check the shape of the DataFrame:')
print(df.shape)

# Check the column names
print( '\n check the column names:')
print(df.columns)

# Check the data types of each column
print('\n check the data types of each column:')
print(df.dtypes)

# Check the missing values
print('\n check the missing values:')
print(df.isnull().sum())

# Check the unique values in each column
print('\n check the unique values in each column:')
for column in df.columns:
    print(df[column].unique())

# Check the distribution of each column
print('\n check the distribution of each column:')
for column in df.columns:
    print(df[column].value_counts())
```

```
 check the shape of the DataFrame:
(6854100, 8)

 check the column names:
Index(['frame', 'row_id', 'type', 'landmark_index', 'x', 'y', 'z',
       'sequence_id'],
      dtype='object')

 check the data types of each column:
frame                int64
row_id              object
type                object
landmark_index       int64
x                  float64
y                  float64
z                  float64
sequence_id          int64
dtype: object

 check the missing values:
```

```
frame                  0
row_id                 0
type                   0
landmark_index         0
x                2611371
y                2611371
z                2611371
sequence_id            0
dtype: int64

 check the unique values in each column:
[ 28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
  17  18  19  20  21  22  23  24  25  26  27  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368]
['28-left_hand-0' '28-left_hand-1' '28-left_hand-2' …
 '368-right_hand-18' '368-right_hand-19' '368-right_hand-20']
['left_hand' 'pose' 'right_hand']
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32]
[       nan 0.45774874 0.49702824 … 0.24177278 0.25391836 0.25731359]
[       nan 0.05357234 0.02111482 … 0.11386584 0.12666472 0.1371545 ]
[       nan 0.26440656 0.27806687 … 0.56036613 0.56105648 0.56260731]
[ 775017794 2374631589 1809269714 …  461538069 2403863845 3520591614]

 check the distribution of each column:
31     104100
32     104100
33     103725
34     103200
30     102900
```

```
        …
355        75
354        75
353        75
352        75
368        75
Name: frame, Length: 369, dtype: int64
32-left_hand-18      1388
31-right_hand-2      1388
31-right_hand-0      1388
31-pose-32           1388
31-pose-31           1388
                     …
357-pose-27             1
357-pose-26             1
357-pose-25             1
357-pose-24             1
368-right_hand-20       1
Name: row_id, Length: 27675, dtype: int64
pose          3015804
left_hand     1919148
right_hand    1919148
Name: type, dtype: int64
0     274164
11    274164
20    274164
19    274164
18    274164
17    274164
1     274164
15    274164
14    274164
13    274164
12    274164
16    274164
10    274164
8     274164
7     274164
6     274164
5     274164
4     274164
3     274164
2     274164
9     274164
27     91388
31     91388
30     91388
29     91388
```

```
28     91388
22     91388
26     91388
25     91388
24     91388
23     91388
21     91388
32     91388
Name: landmark_index, dtype: int64
0.000000    2000
1.000000    1939
1.000000      60
0.851977       2
0.341960       2
            …
0.445724       1
0.479908       1
0.502889       1
0.521004       1
0.257314       1
Name: x, Length: 4237907, dtype: int64
0.000000    2000
1.000000    1806
1.000000     145
1.000000      49
0.009188       3
            …
0.145814       1
0.163988       1
0.182917       1
0.147290       1
0.137155       1
Name: y, Length: 4237438, dtype: int64
0.000000    2000
1.000000    1996
1.000000       4
0.656015       2
0.768214       2
            …
0.568731       1
0.564664       1
0.557889       1
0.551387       1
0.562607       1
Name: z, Length: 4238532, dtype: int64
2235701764    21375
3896074830    18525
2047975791    18450
```

```
1478297125    18375
1863870271    18225
              …
3304912458      225
1868735223      150
1142420711      150
2189530106      150
617950465       150
Name: sequence_id, Length: 2000, dtype: int64
```

Data Shape and Columns:

The DataFrame has 6,854,100 rows and 8 columns. This is a large dataset, so efficient processing and memory management will be crucial.

Data Types: Most columns are of expected types (integers and floats). Ensure these types align with the intended use in the model.

Missing Values: Columns 'x', 'y', and 'z' have 2,611,371 missing values each. We'll handel these missing values by replacing them with 0's.

Unique Values and Distribution: The distribution of values in columns like 'frame', 'row_id', 'type', and 'landmark__index' indicates a wide range of data points. The 'type' column suggests data from three categories: 'left__hand', 'right__hand', and 'pose'.

The distribution of 'sequence__id' shows how many data points are available per sequence.

0.0.1 Feature Engineering

1. **Landmark Aggregation**: For each frame and each type (left hand, right hand, pose), we would like to create a feature vector that aggregateates the landmark data. This means creating a single feature vector per frame per type that encapsulates all the landmarks.
2. **Temporal Features**: Since this is time-series data (sequential data across frames), we'll be creating features that capture the temporal aspect, like the change in position of landmarks from one frame to the next.

Then we will reshape the data into a format that can be fed into the model.

Reshaping Data: We reshape the data into a suitable format. For sequence models like LSTM or GRU (common in handling time-series data), we need to structure the data into sequences.

[ ]:
```python
# Pull lables from train.csv

# Read the train.csv file
train_df = pd.read_csv('/content/train.csv')

# Create a dictionary mapping from sequence_id to sign label
sign_labels = dict(zip(train_df['sequence_id'], train_df['sign']))
```

[ ]:
```python
# Sample data for demonstration (replace with your actual DataFrame)
import pandas as pd
```

```python
import numpy as np

sequence_length = 20


def create_temporal_features(df):
    # Calculating differences in coordinates for each landmark
    df[['x_diff', 'y_diff', 'z_diff']] = df.groupby(['sequence_id', 'type', 'landmark_index'])[['x', 'y', 'z']].diff().fillna(0)
    temporal_features = []
    for (sequence_id, frame, typ), group in df.groupby(['sequence_id', 'frame', 'type']):
        sorted_group = group.sort_values(by='landmark_index')
        differences = sorted_group[['x_diff', 'y_diff', 'z_diff']].values.flatten()
        temporal_features.append((sequence_id, frame, typ, differences))
    return pd.DataFrame(temporal_features, columns=['sequence_id', 'frame', 'type', 'Features'])
temporal_data = create_temporal_features(df)

# Function to validate features
# This code will check if the feature vectors for a given frame and sequence_id in our dataset align with the expected structure.
def validate_features(temporal_data, sequence_id, frame, expected_features):
    frame_data = temporal_data[(temporal_data['sequence_id'] == sequence_id) & (temporal_data['frame'] == frame)]
    for _, row in frame_data.iterrows():
        features = row['Features']
        if len(features) != expected_features[row['type']]:
            return False, f"Mismatch in features for type {row['type']} at frame {frame} of sequence {sequence_id}"
    return True, "All features are correctly structured"

# Feature lengths for each type
expected_feature_lengths = {
    'left_hand': 63,  # 21 landmarks * 3 coordinates
    'right_hand': 63,
    'pose': 99  # 33 landmarks * 3 coordinates
}

# Validate for a particular frame and sequence_id
validation_result = validate_features(temporal_data, sequence_id=1, frame=0, expected_features=expected_feature_lengths)
print(validation_result)
```

```
(True, 'All features are correctly structured')
```

[ ]:
```python
from tensorflow.keras.preprocessing.sequence import pad_sequences

sequence_length = 20

def structure_data_for_lstm(temporal_data, sequence_length, sign_labels):
    sequences = []
    labels = []

    for sequence_id in temporal_data['sequence_id'].unique():
        sequence_data = temporal_data[temporal_data['sequence_id'] ==␣
↪sequence_id]
        label = sign_labels.get(sequence_id)
        if label is None:
            continue

        feature_vectors = []
        for frame in range(sequence_data['frame'].max() + 1):
            frame_data = sequence_data[sequence_data['frame'] == frame]
            feature_vector = np.zeros(225)  # Initialize with zeros
            # [Your code to create feature_vector for each type]
            for typ in ['left_hand', 'right_hand', 'pose']:
                type_data = frame_data[frame_data['type'] == typ]['Features']
                if not type_data.empty:
                    type_features = type_data.iloc[0]  # Assumes each type only␣
↪has one row per frame
                    # Check if type_features has the expected number of elements
                    expected_length = 63 if typ in ['left_hand', 'right_hand']␣
↪else 99  # 99 for pose
                    if len(type_features) == expected_length:
                        start_index = 0 if typ == 'left_hand' else (63 if typ␣
↪== 'right_hand' else 126)
                        feature_vector[start_index:start_index +␣
↪expected_length] = type_features
                    else:
                        # Handle the case where type_features is not as long as␣
↪expected
                        print(f"Warning: Missing data for {typ} in sequence␣
↪{sequence_id}, frame {frame}")

            feature_vectors.append(feature_vector)

        # Padding the sequence
        feature_vectors_padded = pad_sequences([feature_vectors],␣
↪maxlen=sequence_length, padding='post', dtype='float32')[0]

        sequences.append(feature_vectors_padded)
```

```
        labels.append(label)

    return np.array(sequences), np.array(labels)

sequences, labels = structure_data_for_lstm(temporal_data, sequence_length,
 ↪sign_labels)
```

0.1 Model Building

0.1.1 Number of Features

we have 21 unique landmarks for each hand and 33 landmarks for the pose. For each landmark, we have x, y, z coordinates. Thus, for each type (left hand, right hand, pose), you have 21 * 3 = 63 features for hands and 33 * 3 = 99 features for pose. If you're using all these features, n_features would be 63 (left hand) + 63 (right hand) + 99 (pose) = 225.

[ ]:
```
# we use TensorFlow and Keras to define the LSTM model.
# The model architecture can be simple to start with, and then we can expand or
 ↪modify it based on the model's performance.

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# We are using 225 features and we are classifying into 'n_classes' categories
n_features = 225
n_classes = len(np.unique(labels))  # Calculate the number of unique labels

# Define the sequence length (number of frames per sequence)


# Define the LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, n_features)),
    Dropout(0.2),
    LSTM(50),
    Dropout(0.2),
    Dense(n_classes, activation='softmax')  # Use 'softmax' for multi-class
 ↪classification
])

# Output the number of classes and model summary
print(f"Number of classes: {n_classes}")
model.summary()
```

```
Number of classes: 250
Model: "sequential_8"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 lstm_16 (LSTM)              (None, 20, 50)            55200

 dropout_16 (Dropout)        (None, 20, 50)            0

 lstm_17 (LSTM)              (None, 50)                20200

 dropout_17 (Dropout)        (None, 50)                0

 dense_10 (Dense)            (None, 250)               12750

=================================================================
Total params: 88150 (344.34 KB)
Trainable params: 88150 (344.34 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
# Compile the Model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# Data Preparation for Training



# Split the data into training and validation sets.
# The split is 80% for training and 20% for validation.
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(sequences, labels,␣
 ↪test_size=0.2, random_state=64)

# Check the shapes and types of the training and validation data
print(X_train.shape, X_train.dtype)
print(y_train.shape, y_train.dtype)
print(X_val.shape, X_val.dtype)
print(y_val.shape, y_val.dtype)
```

```
(1600, 20, 225) float32
(1600,) <U12
(400, 20, 225) float32
(400,) <U12
```

We can see from the data that the labels are encoded as integers but the features are strings. We need to convert them to integers.

[ ]:
```
from sklearn.preprocessing import LabelEncoder

# Create the label encoder
```

```
label_encoder = LabelEncoder()

# Fit the encoder to your labels (all labels in dataset)
label_encoder.fit(np.concatenate((y_train, y_val), axis=0))

# Transform the training and validation labels
y_train_encoded = label_encoder.transform(y_train)
y_val_encoded = label_encoder.transform(y_val)

# Check the shapes and types again
print(y_train_encoded.shape, y_train_encoded.dtype)
print(y_val_encoded.shape, y_val_encoded.dtype)
```

```
(1600,) int64
(400,) int64
```

[ ]:
```
# Model Training
history = model.fit(
    X_train, y_train_encoded,  # Use integer-encoded labels for training
    validation_data=(X_val, y_val_encoded),  # Use integer-encoded labels for␣
 ↪validation
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
50/50 [==============================] - 4s 24ms/step - loss: 5.5225 - accuracy:
0.0031 - val_loss: 5.5216 - val_accuracy: 0.0125
Epoch 2/10
50/50 [==============================] - 0s 8ms/step - loss: 5.5165 - accuracy:
0.0088 - val_loss: 5.5251 - val_accuracy: 0.0025
Epoch 3/10
50/50 [==============================] - 0s 8ms/step - loss: 5.5025 - accuracy:
0.0056 - val_loss: 5.5253 - val_accuracy: 0.0000e+00
Epoch 4/10
50/50 [==============================] - 0s 8ms/step - loss: 5.4840 - accuracy:
0.0075 - val_loss: 5.5361 - val_accuracy: 0.0125
Epoch 5/10
50/50 [==============================] - 0s 8ms/step - loss: 5.4560 - accuracy:
0.0075 - val_loss: 5.5195 - val_accuracy: 0.0075
Epoch 6/10
50/50 [==============================] - 0s 8ms/step - loss: 5.4141 - accuracy:
0.0081 - val_loss: 5.5085 - val_accuracy: 0.0200
Epoch 7/10
50/50 [==============================] - 0s 8ms/step - loss: 5.3680 - accuracy:
0.0119 - val_loss: 5.4855 - val_accuracy: 0.0150
Epoch 8/10
50/50 [==============================] - 0s 8ms/step - loss: 5.3079 - accuracy:
```

```
0.0169 - val_loss: 5.4679 - val_accuracy: 0.0100
Epoch 9/10
50/50 [==============================] - 0s 8ms/step - loss: 5.2490 - accuracy:
0.0219 - val_loss: 5.4706 - val_accuracy: 0.0075
Epoch 10/10
50/50 [==============================] - 0s 8ms/step - loss: 5.1830 - accuracy:
0.0244 - val_loss: 5.4722 - val_accuracy: 0.0200
```

Changing model architecture to Bidirectional LSTM to see if that improves performance

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.regularizers import l1_l2


n_features = 225
n_classes = len(np.unique(labels))

regularizer = l1_l2(l1=0.01, l2=0.01)  # Example regularization parameters,␣
 ↪these may need tuning


model = Sequential([
    Bidirectional(LSTM(200, return_sequences=True,␣
 ↪input_shape=(sequence_length, n_features))),
    Dropout(0.7),
    Bidirectional(LSTM(200)),
    Dropout(0.7),
    Dense(100, activation='relu', kernel_regularizer=regularizer),
    Dense(n_classes, activation='softmax')
])

# Using a custom learning rate
#optimizer = Adam(learning_rate=0.0001)

#model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

optimizer = RMSprop(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
sample_input = np.random.random((1, sequence_length, n_features))
model(sample_input)

model.summary()
```

```
Model: "sequential_23"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_30 (Bidirect  (1, 20, 400)              681600
 ional)

 dropout_45 (Dropout)        (1, 20, 400)              0

 bidirectional_31 (Bidirect  (1, 400)                  961600
 ional)

 dropout_46 (Dropout)        (1, 400)                  0

 dense_37 (Dense)            (1, 100)                  40100

 dense_38 (Dense)            (1, 250)                  25250

=================================================================
Total params: 1708550 (6.52 MB)
Trainable params: 1708550 (6.52 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:

```
model = Sequential([
    LSTM(100, input_shape=(sequence_length, n_features)),
    Dropout(0.5),
    Dense(n_classes, activation='softmax')
])

# Using a custom learning rate
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

#model(sample_input)

model.summary()
```

```
Model: "sequential_24"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_42 (LSTM)              (None, 100)               130400

 dropout_47 (Dropout)        (None, 100)               0

 dense_39 (Dense)            (None, 250)               25250
```

```
=================================================================
Total params: 155650 (608.01 KB)
Trainable params: 155650 (608.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
#Data preprocessing
from sklearn.preprocessing import StandardScaler

# Assuming 'sequences' is your data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).
↪reshape(X_train.shape)
X_val_scaled = scaler.transform(X_val.reshape(-1, X_val.shape[-1])).
↪reshape(X_val.shape)
```

[ ]:
```
#training procedure
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
↪restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',␣
↪save_best_only=True)


history = model.fit(
    X_train_scaled, y_train_encoded,
    validation_data=(X_val_scaled, y_val_encoded),
    epochs=50,  # Increased epochs
    batch_size=16,  # Adjusted batch size
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
100/100 [==============================] - 1s 6ms/step - loss: 0.2296 -
accuracy: 0.9737 - val_loss: 6.5063 - val_accuracy: 0.0375
Epoch 2/50
100/100 [==============================] - 1s 6ms/step - loss: 0.1969 -
accuracy: 0.9837 - val_loss: 6.5250 - val_accuracy: 0.0375
Epoch 3/50
100/100 [==============================] - 1s 5ms/step - loss: 0.2051 -
accuracy: 0.9725 - val_loss: 6.6032 - val_accuracy: 0.0500
Epoch 4/50
100/100 [==============================] - 1s 5ms/step - loss: 0.2072 -
accuracy: 0.9669 - val_loss: 6.5575 - val_accuracy: 0.0400
```

Based on the above, we can see that the training accuracy is increasing over epochs,

the validation accuracy is decreasing over epochs, and the validation loss is increasing over epochs which can be interprated as the following:

Increasing Training Accuracy: The model's training accuracy is increasing over epochs, which is a positive sign. It suggests that the model is learning from the training data.

Validation Accuracy Not Keeping Pace: However, the validation accuracy is much lower and doesn't increase at the same rate. This could be a sign of overfitting, where the model learns the training data too well, including its noise and outliers, but does not generalize well to new, unseen data.

Rising Validation Loss: The increasing validation loss further supports the possibility of overfitting.

To address these issues, we will consider adding the following steps:

Regularization: Implement dropout layers or L2 regularization to prevent overfitting. Data Augmentation: If possible, augment your data to introduce more variability and help the model generalize better.

Early Stopping: Implement early stopping to terminate training when the validation loss starts to increase, preventing overfitting.

Hyperparameter Tuning: Optimize Model Architecture and Tune LSTM Units Adjust the number of units in LSTM layers. Sometimes fewer units can help the model generalize better.

Layer Adjustments: Experiment with adding or removing layers to find a better architecture balance

[ ]:
```
# accounting for class imbalance

from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight('balanced', classes=np.
 ↪unique(y_train_encoded), y=y_train_encoded)
class_weight_dict = dict(enumerate(class_weights))

history = model.fit(
    X_train_scaled, y_train_encoded,
    validation_data=(X_val_scaled, y_val_encoded),
    epochs=50,
    batch_size=8,
    class_weight=class_weight_dict,
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
200/200 [==============================] - 7s 18ms/step - loss: 3.9584 -
accuracy: 0.1863 - val_loss: 5.3828 - val_accuracy: 0.0525
Epoch 2/50
200/200 [==============================] - 2s 11ms/step - loss: 3.3910 -
accuracy: 0.2262 - val_loss: 5.5687 - val_accuracy: 0.0450
```

```
Epoch 3/50
200/200 [==============================] - 2s 11ms/step - loss: 2.6470 -
accuracy: 0.3675 - val_loss: 5.7771 - val_accuracy: 0.0525
Epoch 4/50
200/200 [==============================] - 2s 12ms/step - loss: 1.9898 -
accuracy: 0.4919 - val_loss: 6.5177 - val_accuracy: 0.0700
```

more changes to see if it improves performance

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

# Assuming 'X_train' and 'X_val' are your training and validation sets
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).
↪reshape(X_train.shape)
X_val_scaled = scaler.transform(X_val.reshape(-1, X_val.shape[-1])).
↪reshape(X_val.shape)
```

[ ]:
```
from tensorflow.keras.layers import GRU

model = Sequential([
    Bidirectional(GRU(100, return_sequences=True, input_shape=(sequence_length,␣
↪n_features))),
    Dropout(0.3),
    Bidirectional(GRU(100)),
    Dropout(0.3),
    Dense(100, activation='relu'),
    Dense(n_classes, activation='softmax')
])

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
sample_input = np.random.random((1, sequence_length, n_features))
model(sample_input)

model.summary()
```

```
Model: "sequential_13"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_12 (Bidirect  (1, 20, 200)              196200
 ional)

 dropout_26 (Dropout)        (1, 20, 200)              0

 bidirectional_13 (Bidirect  (1, 200)                  181200
 ional)
```

```
 dropout_27 (Dropout)        (1, 200)                  0

 dense_18 (Dense)            (1, 100)                  20100

 dense_19 (Dense)            (1, 250)                  25250

=================================================================
Total params: 422750 (1.61 MB)
Trainable params: 422750 (1.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',␣
 ↪save_best_only=True)

# Add 'checkpoint' to the callbacks list in model.fit
history = model.fit(
    X_train_scaled, y_train_encoded,
    validation_data=(X_val_scaled, y_val_encoded),
    epochs=50,
    batch_size=64,
    callbacks=[early_stopping, checkpoint]  # Add 'checkpoint' here
)
```

```
Epoch 1/50
25/25 [==============================] - 8s 72ms/step - loss: 5.5545 - accuracy:
0.0031 - val_loss: 5.5231 - val_accuracy: 0.0050
Epoch 2/50
11/25 [============>…] - ETA: 0s - loss: 5.5209 - accuracy:
0.0043

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(

25/25 [==============================] - 0s 13ms/step - loss: 5.5398 - accuracy:
0.0031 - val_loss: 5.5309 - val_accuracy: 0.0000e+00
Epoch 3/50
25/25 [==============================] - 0s 13ms/step - loss: 5.5237 - accuracy:
0.0044 - val_loss: 5.5427 - val_accuracy: 0.0025
Epoch 4/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5232 - accuracy:
0.0044 - val_loss: 5.5409 - val_accuracy: 0.0050
Epoch 5/50
```

```
25/25 [==============================] - 0s 13ms/step - loss: 5.5200 - accuracy:
0.0056 - val_loss: 5.5386 - val_accuracy: 0.0050
Epoch 6/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5133 - accuracy:
0.0044 - val_loss: 5.5446 - val_accuracy: 0.0000e+00
Epoch 7/50
25/25 [==============================] - 0s 13ms/step - loss: 5.5175 - accuracy:
0.0044 - val_loss: 5.5502 - val_accuracy: 0.0025
Epoch 8/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5062 - accuracy:
0.0056 - val_loss: 5.5680 - val_accuracy: 0.0000e+00
Epoch 9/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5011 - accuracy:
0.0075 - val_loss: 5.5471 - val_accuracy: 0.0100
Epoch 10/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5115 - accuracy:
0.0050 - val_loss: 5.5499 - val_accuracy: 0.0100
Epoch 11/50
25/25 [==============================] - 0s 14ms/step - loss: 5.5037 - accuracy:
0.0050 - val_loss: 5.5581 - val_accuracy: 0.0100
```

At this point we can see that the mdoel is overfitting. If we had more time we would see if we could improve the model by applying more regularization, data augmentation, and hyperparameter tuning.

At this point we're going to try another more complex archetecture to see if that improves performance. In the next phase we'll expore transformer models with the same dataset.