Modeling

December 11, 2023

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.

[ ]:
```
!pip install tensorflow-addons
```

```
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.23.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (611 kB)
                           611.8/611.8

kB 10.8 MB/s eta 0:00:00
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow-addons) (23.2)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow-addons)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.23.0 typeguard-2.13.3
```

[ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sn

from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split, GroupShuffleSplit

import glob
import sys
import os
import math
import gc
import sys
```

```python
import sklearn
import scipy
```

```
/usr/local/lib/python3.10/dist-
packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of
life in May 2024.
Please modify downstream libraries to take dependencies from other repositories
in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
```

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

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

[ ]:
```python
googledrive_dir = '/content/drive/MyDrive/Colab Notebooks/Data/asl-signs/'
```

[ ]:
```python
X_train = np.load(f'{googledrive_dir}/X_train.npy')
y_train = np.load(f'{googledrive_dir}/y_train.npy')
NON_EMPTY_FRAME_IDXS_TRAIN = np.load(f'{googledrive_dir}/
 ↪NON_EMPTY_FRAME_IDXS_TRAIN.npy')
X_val = np.load(f'{googledrive_dir}/X_val.npy')
y_val = np.load(f'{googledrive_dir}/y_val.npy')
NON_EMPTY_FRAME_IDXS_VAL = np.load(f'{googledrive_dir}/NON_EMPTY_FRAME_IDXS_VAL.
 ↪npy')
```

[ ]:
```python
X_train.shape
```

[ ]: (80229, 64, 66, 3)

[ ]:
```python
y_train.shape
```

[ ]: (80229,)

[ ]:
```python
X_train.dtype
```

[ ]: dtype('float32')

[ ]:
```python
y_train.dtype
```

[ ]: dtype('int32')

[ ]:
```python
NON_EMPTY_FRAME_IDXS_TRAIN.shape
```

[ ]: (80229, 64)

[ ]:
```
display(pd.Series(y_train).value_counts().to_frame('Class Count').
 ↪iloc[[0,1,2,3,4, -5,-4,-3,-2,-1]])
```

	Class Count
135	358
136	352
60	351
194	351
67	348
170	277
249	267
56	266
21	263
231	255

[ ]:
```
y_train
```

[ ]: array([25, 232, 48, …, 86, 188, 105], dtype=int32)

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
 ↪gislr-tf-data-processing-transformer-training
# Epsilon value for layer normalisation
LAYER_NORM_EPS = 1e-6

# Dense layer units for landmarks
LIPS_UNITS = 384
HANDS_UNITS = 384
POSE_UNITS = 384
# final embedding and transformer embedding size
UNITS = 512

# Transformer
NUM_BLOCKS = 2
MLP_RATIO = 2

# Dropout
EMBEDDING_DROPOUT = 0.00
MLP_DROPOUT_RATIO = 0.30
CLASSIFIER_DROPOUT_RATIO = 0.10

# Initiailizers
INIT_HE_UNIFORM = tf.keras.initializers.he_uniform
INIT_GLOROT_UNIFORM = tf.keras.initializers.glorot_uniform
INIT_ZEROS = tf.keras.initializers.constant(0.0)
# Activations
GELU = tf.keras.activations.gelu
```

```
print(f'UNITS: {UNITS}')
```

```
UNITS: 512
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
# If True, processing data from scratch
# If False, loads preprocessed data
PREPROCESS_DATA = False
TRAIN_MODEL = True
# True: use 10% of participants as validation set
# False: use all data for training -> gives better LB result
USE_VAL = False

N_ROWS = 543
N_DIMS = 3
DIM_NAMES = ['x', 'y', 'z']
SEED = 42
NUM_CLASSES = 250
IS_INTERACTIVE = True
VERBOSE = 1 if IS_INTERACTIVE else 2

INPUT_SIZE = 64

BATCH_ALL_SIGNS_N = 4
BATCH_SIZE = 256
N_EPOCHS = 100
LR_MAX = 1e-3
N_WARMUP_EPOCHS = 0
WD_RATIO = 0.05
MASK_VAL = 4237
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
USE_TYPES = ['left_hand', 'pose', 'right_hand']
START_IDX = 468
LIPS_IDXS0 = np.array([
        61, 185, 40, 39, 37, 0, 267, 269, 270, 409,
        291, 146, 91, 181, 84, 17, 314, 405, 321, 375,
        78, 191, 80, 81, 82, 13, 312, 311, 310, 415,
        95, 88, 178, 87, 14, 317, 402, 318, 324, 308,
    ])
# Landmark indices in original data
LEFT_HAND_IDXS0 = np.arange(468,489)
RIGHT_HAND_IDXS0 = np.arange(522,543)
LEFT_POSE_IDXS0 = np.array([502, 504, 506, 508, 510])
```

```
RIGHT_POSE_IDXS0 = np.array([503, 505, 507, 509, 511])
LANDMARK_IDXS_LEFT_DOMINANT0 = np.concatenate((LIPS_IDXS0, LEFT_HAND_IDXS0,␣
↪LEFT_POSE_IDXS0))
LANDMARK_IDXS_RIGHT_DOMINANT0 = np.concatenate((LIPS_IDXS0, RIGHT_HAND_IDXS0,␣
↪RIGHT_POSE_IDXS0))
HAND_IDXS0 = np.concatenate((LEFT_HAND_IDXS0, RIGHT_HAND_IDXS0), axis=0)
N_COLS = LANDMARK_IDXS_LEFT_DOMINANT0.size
# Landmark indices in processed data
LIPS_IDXS = np.argwhere(np.isin(LANDMARK_IDXS_LEFT_DOMINANT0, LIPS_IDXS0)).
↪squeeze()
LEFT_HAND_IDXS = np.argwhere(np.isin(LANDMARK_IDXS_LEFT_DOMINANT0,␣
↪LEFT_HAND_IDXS0)).squeeze()
RIGHT_HAND_IDXS = np.argwhere(np.isin(LANDMARK_IDXS_LEFT_DOMINANT0,␣
↪RIGHT_HAND_IDXS0)).squeeze()
HAND_IDXS = np.argwhere(np.isin(LANDMARK_IDXS_LEFT_DOMINANT0, HAND_IDXS0)).
↪squeeze()
POSE_IDXS = np.argwhere(np.isin(LANDMARK_IDXS_LEFT_DOMINANT0, LEFT_POSE_IDXS0)).
↪squeeze()

print(f'# HAND_IDXS: {len(HAND_IDXS)}, N_COLS: {N_COLS}')
```

```
# HAND_IDXS: 21, N_COLS: 66
```

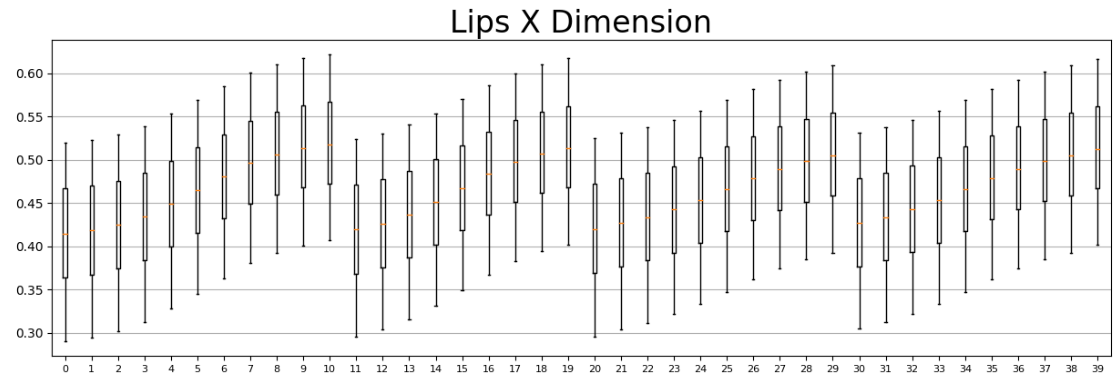
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
LIPS_START = 0
LEFT_HAND_START = LIPS_IDXS.size
RIGHT_HAND_START = LEFT_HAND_START + LEFT_HAND_IDXS.size
POSE_START = RIGHT_HAND_START + RIGHT_HAND_IDXS.size

print(f'LIPS_START: {LIPS_START}, LEFT_HAND_START: {LEFT_HAND_START},␣
↪RIGHT_HAND_START: {RIGHT_HAND_START}, POSE_START: {POSE_START}')
```

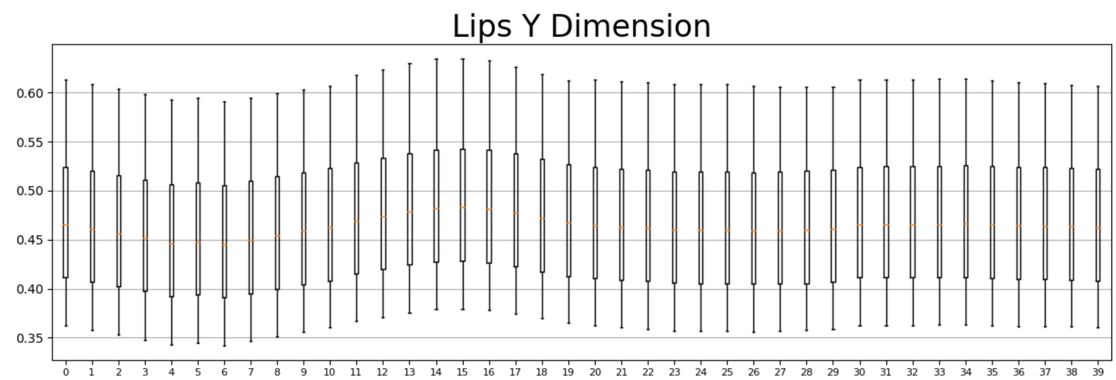
```
LIPS_START: 0, LEFT_HAND_START: 40, RIGHT_HAND_START: 61, POSE_START: 61
```

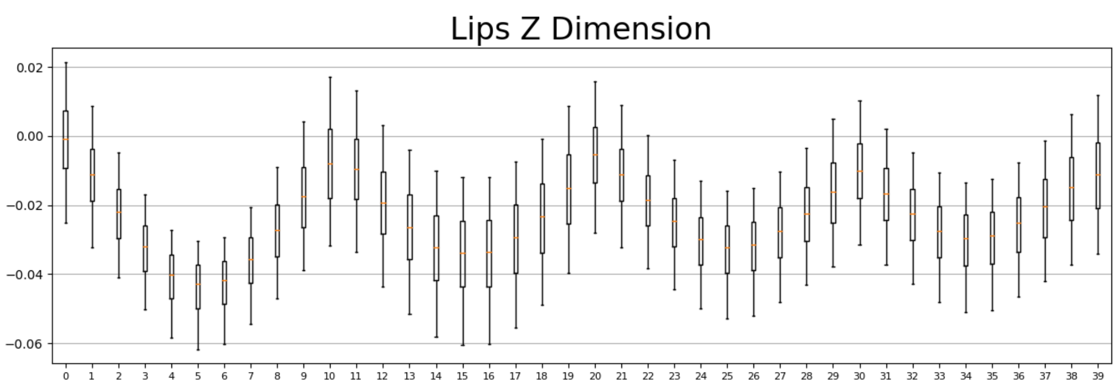
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
def get_lips_mean_std():
    # LIPS
    LIPS_MEAN_X = np.zeros([LIPS_IDXS.size], dtype=np.float32)
    LIPS_MEAN_Y = np.zeros([LIPS_IDXS.size], dtype=np.float32)
    LIPS_STD_X = np.zeros([LIPS_IDXS.size], dtype=np.float32)
    LIPS_STD_Y = np.zeros([LIPS_IDXS.size], dtype=np.float32)

    fig, axes = plt.subplots(3, 1, figsize=(15, N_DIMS*6))
```

```
    for col, ll in enumerate(tqdm( np.transpose(X_train[:,:,LIPS_IDXS],␣
↪[2,3,0,1]).reshape([LIPS_IDXS.size, N_DIMS, -1]) )):
        for dim, l in enumerate(ll):
            v = l[np.nonzero(l)]
            if dim == 0: # X
                LIPS_MEAN_X[col] = v.mean()
                LIPS_STD_X[col] = v.std()
            if dim == 1: # Y
                LIPS_MEAN_Y[col] = v.mean()
                LIPS_STD_Y[col] = v.std()

            axes[dim].boxplot(v, notch=False, showfliers=False,␣
↪positions=[col], whis=[5,95])

    for ax, dim_name in zip(axes, DIM_NAMES):
        ax.set_title(f'Lips {dim_name.upper()} Dimension', size=24)
        ax.tick_params(axis='x', labelsize=8)
        ax.grid(axis='y')

    plt.subplots_adjust(hspace=0.50)
    plt.show()

    LIPS_MEAN = np.array([LIPS_MEAN_X, LIPS_MEAN_Y]).T
    LIPS_STD = np.array([LIPS_STD_X, LIPS_STD_Y]).T

    return LIPS_MEAN, LIPS_STD

LIPS_MEAN, LIPS_STD = get_lips_mean_std()
```

```
 0%|          | 0/40 [00:00<?, ?it/s]
```



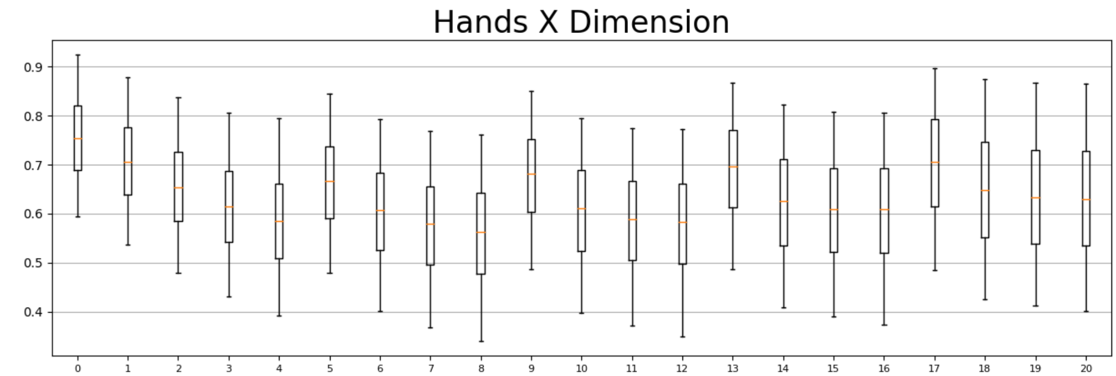


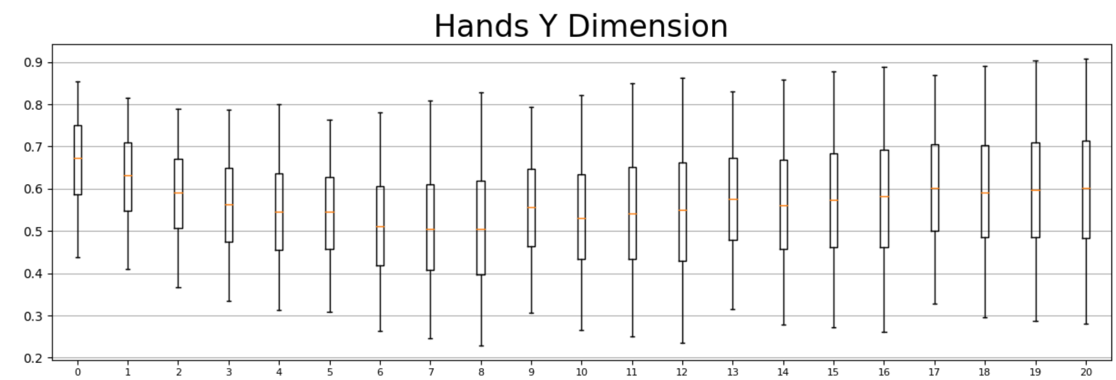


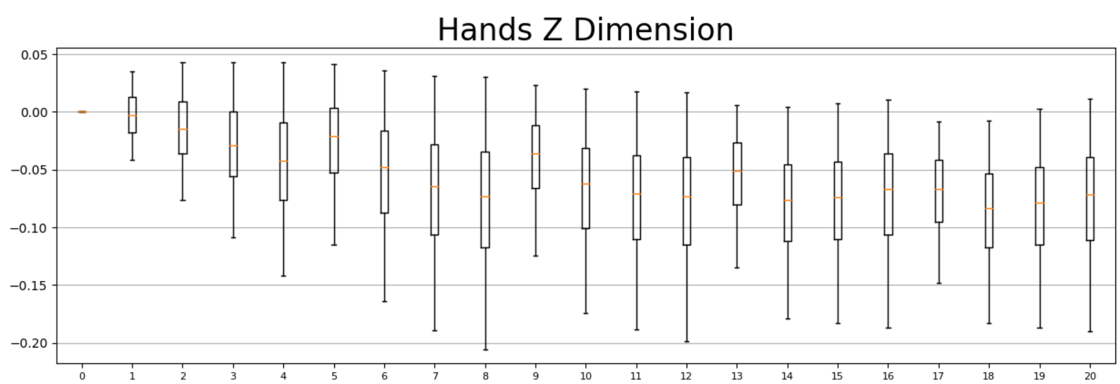
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
def get_left_right_hand_mean_std():
    # LEFT HAND
    LEFT_HANDS_MEAN_X = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)
    LEFT_HANDS_MEAN_Y = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)
    LEFT_HANDS_STD_X = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)
```

```
    LEFT_HANDS_STD_Y = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)

    fig, axes = plt.subplots(3, 1, figsize=(15, N_DIMS*6))

    for col, ll in enumerate(tqdm( np.transpose(X_train[:,:,LEFT_HAND_IDXS],␣
 ↪[2,3,0,1]).reshape([LEFT_HAND_IDXS.size, N_DIMS, -1]) )):
        for dim, l in enumerate(ll):
            v = l[np.nonzero(l)]
            if dim == 0: # X
                LEFT_HANDS_MEAN_X[col] = v.mean()
                LEFT_HANDS_STD_X[col] = v.std()
            if dim == 1: # Y
                LEFT_HANDS_MEAN_Y[col] = v.mean()
                LEFT_HANDS_STD_Y[col] = v.std()
            # Plot
            axes[dim].boxplot(v, notch=False, showfliers=False,␣
 ↪positions=[col], whis=[5,95])

    for ax, dim_name in zip(axes, DIM_NAMES):
        ax.set_title(f'Hands {dim_name.upper()} Dimension', size=24)
        ax.tick_params(axis='x', labelsize=8)
        ax.grid(axis='y')

    plt.subplots_adjust(hspace=0.50)
    plt.show()

    LEFT_HANDS_MEAN = np.array([LEFT_HANDS_MEAN_X, LEFT_HANDS_MEAN_Y]).T
    LEFT_HANDS_STD = np.array([LEFT_HANDS_STD_X, LEFT_HANDS_STD_Y]).T

    return LEFT_HANDS_MEAN, LEFT_HANDS_STD

LEFT_HANDS_MEAN, LEFT_HANDS_STD = get_left_right_hand_mean_std()
```

```
  0%|          | 0/21 [00:00<?, ?it/s]
```



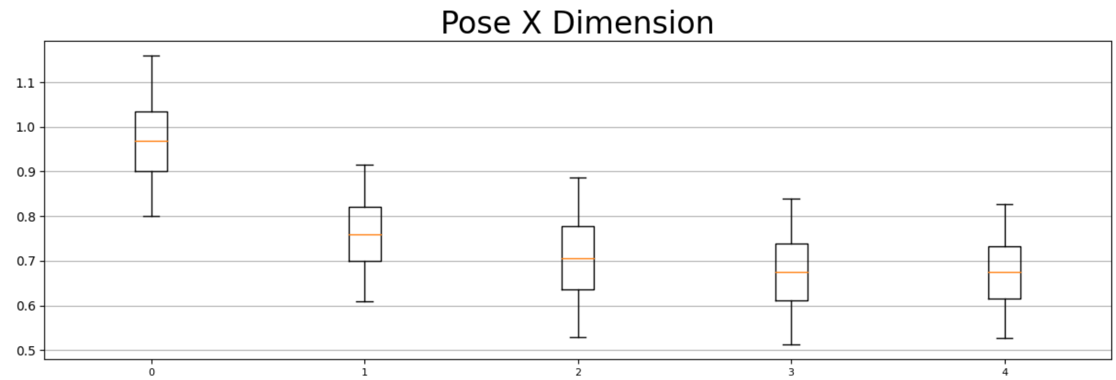


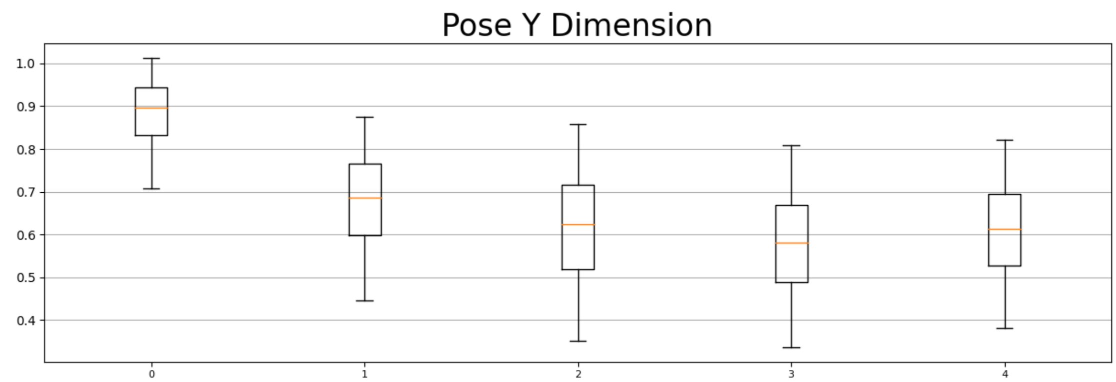


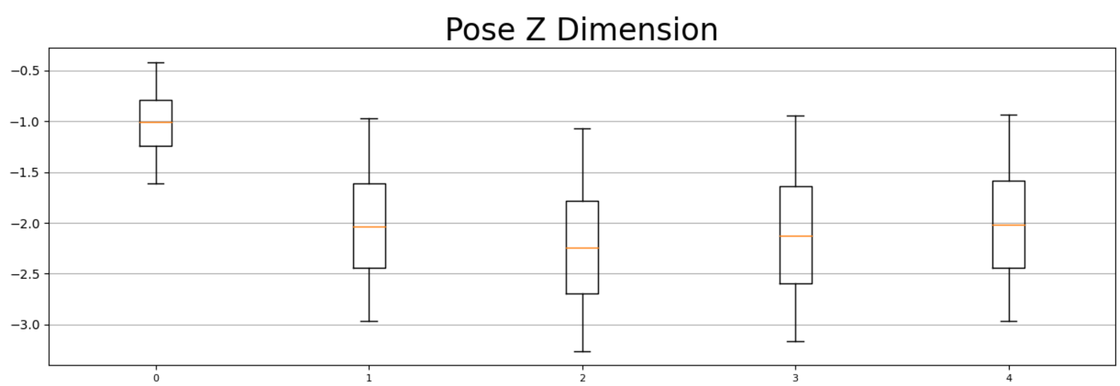
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
 ↪gislr-tf-data-processing-transformer-training
def get_pose_mean_std():
    # POSE
    POSE_MEAN_X = np.zeros([POSE_IDXS.size], dtype=np.float32)
    POSE_MEAN_Y = np.zeros([POSE_IDXS.size], dtype=np.float32)
    POSE_STD_X = np.zeros([POSE_IDXS.size], dtype=np.float32)
```

```
    POSE_STD_Y = np.zeros([POSE_IDXS.size], dtype=np.float32)

    fig, axes = plt.subplots(3, 1, figsize=(15, N_DIMS*6))

    for col, ll in enumerate(tqdm( np.transpose(X_train[:,:,POSE_IDXS],␣
 ↪[2,3,0,1]).reshape([POSE_IDXS.size, N_DIMS, -1]) )):
        for dim, l in enumerate(ll):
            v = l[np.nonzero(l)]
            if dim == 0: # X
                POSE_MEAN_X[col] = v.mean()
                POSE_STD_X[col] = v.std()
            if dim == 1: # Y
                POSE_MEAN_Y[col] = v.mean()
                POSE_STD_Y[col] = v.std()

            axes[dim].boxplot(v, notch=False, showfliers=False,␣
 ↪positions=[col], whis=[5,95])

    for ax, dim_name in zip(axes, DIM_NAMES):
        ax.set_title(f'Pose {dim_name.upper()} Dimension', size=24)
        ax.tick_params(axis='x', labelsize=8)
        ax.grid(axis='y')

    plt.subplots_adjust(hspace=0.50)
    plt.show()

    POSE_MEAN = np.array([POSE_MEAN_X, POSE_MEAN_Y]).T
    POSE_STD = np.array([POSE_STD_X, POSE_STD_Y]).T

    return POSE_MEAN, POSE_STD

POSE_MEAN, POSE_STD = get_pose_mean_std()
```

```
  0%|          | 0/5 [00:00<?, ?it/s]
```







The attention mechanism is a key component in Transformer models, enabling the model to focus on different parts of the input sequence for each step of the output sequence. Attention enables the model to concentrate selectively on various segments of the input sequence for making predictions, rather than interpreting the entire sequence as a uniform-length vector. This feature has been crucial in the triumph of the transformer model, sparking extensive subsequent research and the development of numerous new models (Kumar, A. 2023). Wijkhuizen, M.'s custom transformer model deployed attention_mask in the Scaled Dot-Product function in a different way compared to the classic transformer model in that this mask is applied in the Softmax step to selectively ignore or pay less attention to certain parts of the input, such as padding or irrelevant frames in a video sequence. Also, the Softmax layer was used instead of the Softmax function. The attention mechanism allows the model to focus on different parts of the input sequence dynamically, which is crucial for tasks like ASL recognition. In ASL, the importance of different landmarks can vary significantly across different signs. The multi-head attention mechanism is particularly well-suited to capture these varied dependencies.

```
[ ]: # Code From https://www.kaggle.com/code/markwijkhuizen/
      ↪gislr-tf-data-processing-transformer-training
     # based on: https://stackoverflow.com/questions/67342988/
      ↪verifying-the-implementation-of-multihead-attention-in-transformer
     # replaced softmax with softmax layer to support masked softmax
     def scaled_dot_product(q,k,v, softmax, attention_mask):
         #calculates Q . K(transpose)
         qkt = tf.matmul(q,k,transpose_b=True)
         #caculates scaling factor
         dk = tf.math.sqrt(tf.cast(q.shape[-1],dtype=tf.float32))
         scaled_qkt = qkt/dk
         softmax = softmax(scaled_qkt, mask=attention_mask)

         z = tf.matmul(softmax,v)
         #shape: (m,Tx,depth), same shape as q,k,v
         return z

     class MultiHeadAttention(tf.keras.layers.Layer):
         def __init__(self,d_model,num_of_heads):
             super(MultiHeadAttention,self).__init__()
             self.d_model = d_model
             self.num_of_heads = num_of_heads
             self.depth = d_model//num_of_heads
             self.wq = [tf.keras.layers.Dense(self.depth) for i in␣
      ↪range(num_of_heads)]
             self.wk = [tf.keras.layers.Dense(self.depth) for i in␣
      ↪range(num_of_heads)]
             self.wv = [tf.keras.layers.Dense(self.depth) for i in␣
      ↪range(num_of_heads)]
             self.wo = tf.keras.layers.Dense(d_model)
             self.softmax = tf.keras.layers.Softmax()

         def call(self,x, attention_mask):

             multi_attn = []
             for i in range(self.num_of_heads):
                 Q = self.wq[i](x)
                 K = self.wk[i](x)
                 V = self.wv[i](x)
```

```
            multi_attn.append(scaled_dot_product(Q,K,V, self.softmax,␣
↪attention_mask))

        multi_head = tf.concat(multi_attn,axis=-1)
        multi_head_attention = self.wo(multi_head)
        return multi_head_attention
```

Wijkhuizen, M.'s transformer model approach is a custom-modified transformer model similar similar the classic transformer model. There are custom changes to fit the ASL recognition tasks. This custom transformer only used a classic transformer encoder part for classification tasks, and it is the normal approach to only use the transformer encoder or decoder part alone. Besides the architecture part, there are two major customizations that Wijkhuizen, M. created that are different from the classic transformer model; one is the attention mechanism, and the other one is the embedding layer.

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
# Full Transformer
class Transformer(tf.keras.Model):
    def __init__(self, num_blocks):
        super(Transformer, self).__init__(name='transformer')
        self.num_blocks = num_blocks

    def build(self, input_shape):
        self.ln_1s = []
        self.mhas = []
        self.ln_2s = []
        self.mlps = []
        # Make Transformer Blocks
        for i in range(self.num_blocks):
            # Multi Head Attention
            self.mhas.append(MultiHeadAttention(UNITS, 8))
            # Multi Layer Perception
            self.mlps.append(tf.keras.Sequential([
                tf.keras.layers.Dense(UNITS * MLP_RATIO, activation=GELU,␣
↪kernel_initializer=INIT_GLOROT_UNIFORM),
                tf.keras.layers.Dropout(MLP_DROPOUT_RATIO),
                tf.keras.layers.Dense(UNITS,␣
↪kernel_initializer=INIT_HE_UNIFORM),
            ]))

    def call(self, x, attention_mask):
        # Iterate input over transformer blocks
        for mha, mlp in zip(self.mhas, self.mlps):
            x = x + mha(x, attention_mask)
            x = x + mlp(x)
```

```
        return x
```

```
[ ]: # Code From https://www.kaggle.com/code/markwijkhuizen/
      ↪gislr-tf-data-processing-transformer-training
class LandmarkEmbedding(tf.keras.Model):
    def __init__(self, units, name):
        super(LandmarkEmbedding, self).__init__(name=f'{name}_embedding')
        self.units = units

    def build(self, input_shape):
        # Embedding for missing landmark in frame, initizlied with zeros
        self.empty_embedding = self.add_weight(
            name=f'{self.name}_empty_embedding',
            shape=[self.units],
            initializer=INIT_ZEROS,
        )
        # Embedding
        self.dense = tf.keras.Sequential([
            tf.keras.layers.Dense(self.units, name=f'{self.name}_dense_1',␣
 ↪use_bias=False, kernel_initializer=INIT_GLOROT_UNIFORM),
            tf.keras.layers.Activation(GELU),
            tf.keras.layers.Dense(self.units, name=f'{self.name}_dense_2',␣
 ↪use_bias=False, kernel_initializer=INIT_HE_UNIFORM),
        ], name=f'{self.name}_dense')

    def call(self, x):
        return tf.where(
                # Checks whether landmark is missing in frame
                tf.reduce_sum(x, axis=2, keepdims=True) == 0,
                # If so, the empty embedding is used
                self.empty_embedding,
                # Otherwise the landmark data is embedded
                self.dense(x),
            )
```

Another customized part is the embedding layer. In the context of Transformer models, positional embeddings are crucial for providing information about the order or position of the elements in the input sequence (Huang, Z. et al., 2020). Compared to the classic transformer model, this ASL transformer model approach has an additional LandmarkEmbedding class that the Embedding class uses to deal with the embedding of individual landmarks. Each landmark type was embedded separately from lips, hand, and pose. The Embedding class is still handling the positional embedding.

```
[ ]: # Code From https://www.kaggle.com/code/markwijkhuizen/
      ↪gislr-tf-data-processing-transformer-training
class Embedding(tf.keras.Model):
    def __init__(self):
```

```
        super(Embedding, self).__init__()

    def get_diffs(self, l):
        S = l.shape[2]
        other = tf.expand_dims(l, 3)
        other = tf.repeat(other, S, axis=3)
        other = tf.transpose(other, [0,1,3,2])
        diffs = tf.expand_dims(l, 3) - other
        diffs = tf.reshape(diffs, [-1, INPUT_SIZE, S*S])
        return diffs

    def build(self, input_shape):
        # Positional Embedding, initialized with zeros
        self.positional_embedding = tf.keras.layers.Embedding(INPUT_SIZE+1,␣
↪UNITS, embeddings_initializer=INIT_ZEROS)
        # Embedding layer for Landmarks
        self.lips_embedding = LandmarkEmbedding(LIPS_UNITS, 'lips')
        self.left_hand_embedding = LandmarkEmbedding(HANDS_UNITS, 'left_hand')
        self.pose_embedding = LandmarkEmbedding(POSE_UNITS, 'pose')
        # Landmark Weights
        self.landmark_weights = tf.Variable(tf.zeros([3], dtype=tf.float32),␣
↪name='landmark_weights')
        # Fully Connected Layers for combined landmarks
        self.fc = tf.keras.Sequential([
            tf.keras.layers.Dense(UNITS, name='fully_connected_1',␣
↪use_bias=False, kernel_initializer=INIT_GLOROT_UNIFORM),
            tf.keras.layers.Activation(GELU),
            tf.keras.layers.Dense(UNITS, name='fully_connected_2',␣
↪use_bias=False, kernel_initializer=INIT_HE_UNIFORM),
        ], name='fc')


    def call(self, lips0, left_hand0, pose0, non_empty_frame_idxs,␣
↪training=False):
        # Lips
        lips_embedding = self.lips_embedding(lips0)
        # Left Hand
        left_hand_embedding = self.left_hand_embedding(left_hand0)
        # Pose
        pose_embedding = self.pose_embedding(pose0)
        # Merge Embeddings of all landmarks with mean pooling
        x = tf.stack((
            lips_embedding, left_hand_embedding, pose_embedding,
        ), axis=3)
        x = x * tf.nn.softmax(self.landmark_weights)
        x = tf.reduce_sum(x, axis=3)
        # Fully Connected Layers
```

```
        x = self.fc(x)
        # Add Positional Embedding
        max_frame_idxs = tf.clip_by_value(
                tf.reduce_max(non_empty_frame_idxs, axis=1, keepdims=True),
                1,
                np.PINF,
            )
        normalised_non_empty_frame_idxs = tf.where(
            tf.math.equal(non_empty_frame_idxs, -1.0),
            INPUT_SIZE,
            tf.cast(
                non_empty_frame_idxs / max_frame_idxs * INPUT_SIZE,
                tf.int32,
            ),
        )
        x = x + self.positional_embedding(normalised_non_empty_frame_idxs)

        return x
```

[ ]:
```
# source:: https://stackoverflow.com/questions/60689185/
↪label-smoothing-for-sparse-categorical-crossentropy
def scce_with_ls(y_true, y_pred):
    # One Hot Encode Sparsely Encoded Target Sign
    y_true = tf.cast(y_true, tf.int32)
    y_true = tf.one_hot(y_true, NUM_CLASSES, axis=1)
    y_true = tf.squeeze(y_true, axis=2)
    # Categorical Crossentropy with native label smoothing support
    return tf.keras.losses.categorical_crossentropy(y_true, y_pred,␣
↪label_smoothing=0.25)
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
def get_model():
    # Inputs
    frames = tf.keras.layers.Input([INPUT_SIZE, N_COLS, N_DIMS], dtype=tf.
↪float32, name='frames')
    non_empty_frame_idxs = tf.keras.layers.Input([INPUT_SIZE], dtype=tf.
↪float32, name='non_empty_frame_idxs')
    # Padding Mask
    mask0 = tf.cast(tf.math.not_equal(non_empty_frame_idxs, -1), tf.float32)
    mask0 = tf.expand_dims(mask0, axis=2)
    # Random Frame Masking
    mask = tf.where(
        (tf.random.uniform(tf.shape(mask0)) > 0.25) & tf.math.not_equal(mask0,␣
↪0.0),
        1.0,
        0.0,
```

```
    )
    # Correct Samples Which are all masked now...
    mask = tf.where(
        tf.math.equal(tf.reduce_sum(mask, axis=[1,2], keepdims=True), 0.0),
        mask0,
        mask,
    )


    """
        left_hand: 468:489
        pose: 489:522
        right_hand: 522:543
    """
    x = frames
    x = tf.slice(x, [0,0,0,0], [-1,INPUT_SIZE, N_COLS, 2])
    # LIPS
    lips = tf.slice(x, [0,0,LIPS_START,0], [-1,INPUT_SIZE, 40, 2])
    lips = tf.where(
            tf.math.equal(lips, 0.0),
            0.0,
            (lips - LIPS_MEAN) / LIPS_STD,
        )
    # LEFT HAND
    left_hand = tf.slice(x, [0,0,40,0], [-1,INPUT_SIZE, 21, 2])
    left_hand = tf.where(
            tf.math.equal(left_hand, 0.0),
            0.0,
            (left_hand - LEFT_HANDS_MEAN) / LEFT_HANDS_STD,
        )
    # POSE
    pose = tf.slice(x, [0,0,61,0], [-1,INPUT_SIZE, 5, 2])
    pose = tf.where(
            tf.math.equal(pose, 0.0),
            0.0,
            (pose - POSE_MEAN) / POSE_STD,
        )

    # Flatten
    lips = tf.reshape(lips, [-1, INPUT_SIZE, 40*2])
    left_hand = tf.reshape(left_hand, [-1, INPUT_SIZE, 21*2])
    pose = tf.reshape(pose, [-1, INPUT_SIZE, 5*2])

    # Embedding
    x = Embedding()(lips, left_hand, pose, non_empty_frame_idxs)

    # Encoder Transformer Blocks
```

```
    x = Transformer(NUM_BLOCKS)(x, mask)

    # Pooling
    x = tf.reduce_sum(x * mask, axis=1) / tf.reduce_sum(mask, axis=1)
    # Classifier Dropout
    x = tf.keras.layers.Dropout(CLASSIFIER_DROPOUT_RATIO)(x)
    # Classification Layer
    x = tf.keras.layers.Dense(NUM_CLASSES, activation=tf.keras.activations.
 ↪softmax, kernel_initializer=INIT_GLOROT_UNIFORM)(x)

    outputs = x

    # Create Tensorflow Model
    model = tf.keras.models.Model(inputs=[frames, non_empty_frame_idxs],␣
 ↪outputs=outputs)

    # Sparse Categorical Cross Entropy With Label Smoothing
    loss = scce_with_ls

    # Adam Optimizer with weight decay
    optimizer = tfa.optimizers.AdamW(learning_rate=1e-3, weight_decay=1e-5,␣
 ↪clipnorm=1.0)

    # TopK Metrics
    metrics = [
        tf.keras.metrics.SparseCategoricalAccuracy(name='acc'),
        tf.keras.metrics.SparseTopKCategoricalAccuracy(k=5, name='top_5_acc'),
        tf.keras.metrics.SparseTopKCategoricalAccuracy(k=10, name='top_10_acc'),
    ]

    model.compile(loss=loss, optimizer=optimizer, metrics=metrics)

    return model
```

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

model = get_model()
```

[ ]:
```
# Plot model summary
model.summary(expand_nested=True)
```

```
Model: "model"
________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #     Connected to
```

```
==================================================================================
==================
 non_empty_frame_idxs (Inpu  [(None, 64)]                0          []
 tLayer)

 tf.math.not_equal (TFOpLam  (None, 64)                  0
['non_empty_frame_idxs[0][0]']
 bda)

 tf.cast (TFOpLambda)        (None, 64)                  0
['tf.math.not_equal[0][0]']

 tf.expand_dims (TFOpLambda  (None, 64, 1)               0
['tf.cast[0][0]']
 )

 frames (InputLayer)         [(None, 64, 66, 3)]         0          []

 tf.compat.v1.shape (TFOpLa  (3,)                        0
['tf.expand_dims[0][0]']
 mbda)

 tf.slice (TFOpLambda)       (None, 64, 66, 2)           0
['frames[0][0]']

 tf.random.uniform (TFOpLam  (None, 64, 1)               0
['tf.compat.v1.shape[0][0]']
 bda)

 tf.slice_1 (TFOpLambda)     (None, 64, 40, 2)           0
['tf.slice[0][0]']

 tf.slice_2 (TFOpLambda)     (None, 64, 21, 2)           0
['tf.slice[0][0]']

 tf.slice_3 (TFOpLambda)     (None, 64, 5, 2)            0
['tf.slice[0][0]']

 tf.math.greater (TFOpLambd  (None, 64, 1)               0
['tf.random.uniform[0][0]']
 a)

 tf.math.not_equal_1 (TFOpL  (None, 64, 1)               0
['tf.expand_dims[0][0]']
 ambda)

 tf.math.subtract (TFOpLamb  (None, 64, 40, 2)           0
['tf.slice_1[0][0]']
```

```
 da)

 tf.math.subtract_1 (TFOpLa  (None, 64, 21, 2)            0
['tf.slice_2[0][0]']
 mbda)

 tf.math.subtract_2 (TFOpLa  (None, 64, 5, 2)             0
['tf.slice_3[0][0]']
 mbda)

 tf.math.logical_and (TFOpL  (None, 64, 1)                0
['tf.math.greater[0][0]',
 ambda)
'tf.math.not_equal_1[0][0]']

 tf.math.equal_1 (TFOpLambd  (None, 64, 40, 2)            0
['tf.slice_1[0][0]']
 a)

 tf.math.truediv (TFOpLambd  (None, 64, 40, 2)            0
['tf.math.subtract[0][0]']
 a)

 tf.math.equal_2 (TFOpLambd  (None, 64, 21, 2)            0
['tf.slice_2[0][0]']
 a)

 tf.math.truediv_1 (TFOpLam  (None, 64, 21, 2)            0
['tf.math.subtract_1[0][0]']
 bda)

 tf.math.equal_3 (TFOpLambd  (None, 64, 5, 2)             0
['tf.slice_3[0][0]']
 a)

 tf.math.truediv_2 (TFOpLam  (None, 64, 5, 2)             0
['tf.math.subtract_2[0][0]']
 bda)

 tf.where (TFOpLambda)       (None, 64, 1)                0
['tf.math.logical_and[0][0]']

 tf.where_2 (TFOpLambda)     (None, 64, 40, 2)            0
['tf.math.equal_1[0][0]',
'tf.math.truediv[0][0]']

 tf.where_3 (TFOpLambda)     (None, 64, 21, 2)            0
['tf.math.equal_2[0][0]',
```

```
'tf.math.truediv_1[0][0]']

 tf.where_4 (TFOpLambda)     (None, 64, 5, 2)             0
['tf.math.equal_3[0][0]',
'tf.math.truediv_2[0][0]']

 tf.math.reduce_sum (TFOpLa  (None, 1, 1)                 0
['tf.where[0][0]']
 mbda)

 tf.reshape (TFOpLambda)     (None, 64, 80)               0
['tf.where_2[0][0]']

 tf.reshape_1 (TFOpLambda)   (None, 64, 42)               0
['tf.where_3[0][0]']

 tf.reshape_2 (TFOpLambda)   (None, 64, 10)               0
['tf.where_4[0][0]']

 tf.math.equal (TFOpLambda)  (None, 1, 1)                 0
['tf.math.reduce_sum[0][0]']

 embedding (Embedding)       (None, 64, 512)              986243
['tf.reshape[0][0]',
'tf.reshape_1[0][0]',
'tf.reshape_2[0][0]',
'non_empty_frame_idxs[0][0]']
|-------------------------------------------------------------------------------
-----------------|
| embedding (Embedding)      multiple                     33280     []
|
|
|
| lips_embedding (LandmarkE  multiple                     178560    []
|
| mbedding)
|
||------------------------------------------------------------------------------
----------------||
|| lips_embedding_dense (Se  (None, 64, 384)              178176    []
||
|| quential)
||
|||-----------------------------------------------------------------------------
---------------|||
||| lips_embedding_dense_1   (None, 64, 384)              30720     []
|||
||| (Dense)
```

```
|||
|||
|||
||| activation_1 (Activatio  (None, 64, 384)              0          []
|||
||| n)
|||
|||
|||
||| lips_embedding_dense_2   (None, 64, 384)              147456     []
|||
||| (Dense)
|||
||----------------------------------------------------------------------------
----------------||
|-----------------------------------------------------------------------------
-----------------|
| left_hand_embedding (Land  multiple                     163968     []
|
| markEmbedding)
|
||----------------------------------------------------------------------------
----------------||
|| left_hand_embedding_dens  (None, 64, 384)              163584     []
||
|| e (Sequential)
||
|||---------------------------------------------------------------------------
---------------|||
||| left_hand_embedding_den  (None, 64, 384)              16128      []
|||
||| se_1 (Dense)
|||
|||
|||
||| activation_2 (Activatio  (None, 64, 384)              0          []
|||
||| n)
|||
|||
|||
||| left_hand_embedding_den  (None, 64, 384)              147456     []
|||
||| se_2 (Dense)
|||
||----------------------------------------------------------------------------
----------------||
|-----------------------------------------------------------------------------
```

```
-----------------|
| pose_embedding (LandmarkE  multiple                    151680    []
|
| mbedding)
|
||------------------------------------------------------------------------------
---------------||
|| pose_embedding_dense (Se  (None, 64, 384)             151296    []
||
|| quential)
||
|||-----------------------------------------------------------------------------
--------------|||
||| pose_embedding_dense_1   (None, 64, 384)             3840      []
|||
||| (Dense)
|||
|||
|||
||| activation_3 (Activatio  (None, 64, 384)             0         []
|||
||| n)
|||
|||
|||
||| pose_embedding_dense_2   (None, 64, 384)             147456    []
|||
||| (Dense)
|||
||------------------------------------------------------------------------------
---------------||
|-------------------------------------------------------------------------------
-----------------|
| fc (Sequential)            (None, 64, 512)             458752    []
|
||------------------------------------------------------------------------------
---------------||
|| fully_connected_1 (Dense  (None, 64, 512)             196608    []
||
|| )
||
||
||
|| activation (Activation)   (None, 64, 512)             0         []
||
||
||
|| fully_connected_2 (Dense  (None, 64, 512)             262144    []
```

```
||
|| )
||
|-------------------------------------------------------------------------------
-----------------|
--------------------------------------------------------------------------------
------------------
 tf.where_1 (TFOpLambda)     (None, 64, 1)                0
['tf.math.equal[0][0]',
'tf.expand_dims[0][0]',
'tf.where[0][0]']

 transformer (Transformer)   (None, 64, 512)              4201472
['embedding[0][0]',
'tf.where_1[0][0]']
|-------------------------------------------------------------------------------
-----------------|
| multi_head_attention (Mul  multiple                     1050624   []
|
| tiHeadAttention)
|
|
|
| multi_head_attention_1 (M  multiple                     1050624   []
|
| ultiHeadAttention)
|
|
|
| sequential (Sequential)    (None, 64, 512)              1050112   []
|
||------------------------------------------------------------------------------
----------------||
|| dense_25 (Dense)          (None, 64, 1024)             525312    []
||
||
||
|| dropout (Dropout)         (None, 64, 1024)             0         []
||
||
||
|| dense_26 (Dense)          (None, 64, 512)              524800    []
||
|-------------------------------------------------------------------------------
-----------------|
| sequential_1 (Sequential)  (None, 64, 512)              1050112   []
|
||------------------------------------------------------------------------------
```

```
----------------||
|| dense_52 (Dense)          (None, 64, 1024)             525312    []
||
||
||
|| dropout_1 (Dropout)       (None, 64, 1024)             0         []
||
||
||
|| dense_53 (Dense)          (None, 64, 512)              524800    []
||
|--------------------------------------------------------------------------------
-----------------|
---------------------------------------------------------------------------------
------------------
 tf.math.multiply (TFOpLamb  (None, 64, 512)              0
['transformer[0][0]',
 da)
'tf.where_1[0][0]']

 tf.math.reduce_sum_1 (TFOp  (None, 512)                  0
['tf.math.multiply[0][0]']
 Lambda)

 tf.math.reduce_sum_2 (TFOp  (None, 1)                    0
['tf.where_1[0][0]']
 Lambda)

 tf.math.truediv_3 (TFOpLam  (None, 512)                  0
['tf.math.reduce_sum_1[0][0]',
 bda)
'tf.math.reduce_sum_2[0][0]']

 dropout (Dropout)           (None, 512)                  0
['tf.math.truediv_3[0][0]']

 dense (Dense)               (None, 250)                  128250
['dropout[0][0]']

=================================================================================
==================
Total params: 5315965 (20.28 MB)
Trainable params: 5315965 (20.28 MB)
Non-trainable params: 0 (0.00 Byte)
---------------------------------------------------------------------------------
------------------
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
def get_train_batch_all_signs(X, y, NON_EMPTY_FRAME_IDXS, n=BATCH_ALL_SIGNS_N):
    # Arrays to store batch in
    X_batch = np.zeros([NUM_CLASSES*n, INPUT_SIZE, N_COLS, N_DIMS], dtype=np.
↪float32)
    y_batch = np.arange(0, NUM_CLASSES, step=1/n, dtype=np.float32).astype(np.
↪int64)
    non_empty_frame_idxs_batch = np.zeros([NUM_CLASSES*n, INPUT_SIZE], dtype=np.
↪float32)

    # Dictionary mapping ordinally encoded sign to corresponding sample indices
    CLASS2IDXS = {}
    for i in range(NUM_CLASSES):
        CLASS2IDXS[i] = np.argwhere(y == i).squeeze().astype(np.int32)

    while True:
        # Fill batch arrays
        for i in range(NUM_CLASSES):
            idxs = np.random.choice(CLASS2IDXS[i], n)
            X_batch[i*n:(i+1)*n] = X[idxs]
            non_empty_frame_idxs_batch[i*n:(i+1)*n] = NON_EMPTY_FRAME_IDXS[idxs]

        yield { 'frames': X_batch, 'non_empty_frame_idxs':␣
↪non_empty_frame_idxs_batch }, y_batch
```

[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
↪gislr-tf-data-processing-transformer-training
# Actual Training
history = model.fit(
            x=get_train_batch_all_signs(X_train, y_train,␣
↪NON_EMPTY_FRAME_IDXS_TRAIN),
            steps_per_epoch=len(X_train) // (NUM_CLASSES * BATCH_ALL_SIGNS_N),
            epochs=N_EPOCHS,
            batch_size=BATCH_SIZE,
        )
```

```
Epoch 1/100
80/80 [==============================] - 34s 213ms/step - loss: 4.3454 - acc:
0.2691 - top_5_acc: 0.5258 - top_10_acc: 0.6316
Epoch 2/100
80/80 [==============================] - 17s 212ms/step - loss: 3.3600 - acc:
0.5616 - top_5_acc: 0.8160 - top_10_acc: 0.8755
Epoch 3/100
80/80 [==============================] - 17s 213ms/step - loss: 3.0562 - acc:
0.6665 - top_5_acc: 0.8765 - top_10_acc: 0.9160
Epoch 4/100
```

```
80/80 [==============================] - 17s 212ms/step - loss: 2.9035 - acc:
0.7227 - top_5_acc: 0.9030 - top_10_acc: 0.9325
Epoch 5/100
80/80 [==============================] - 17s 212ms/step - loss: 2.8028 - acc:
0.7592 - top_5_acc: 0.9164 - top_10_acc: 0.9422
Epoch 6/100
80/80 [==============================] - 17s 212ms/step - loss: 2.7248 - acc:
0.7879 - top_5_acc: 0.9294 - top_10_acc: 0.9517
Epoch 7/100
80/80 [==============================] - 17s 212ms/step - loss: 2.6656 - acc:
0.8100 - top_5_acc: 0.9388 - top_10_acc: 0.9578
Epoch 8/100
80/80 [==============================] - 17s 213ms/step - loss: 2.6164 - acc:
0.8279 - top_5_acc: 0.9471 - top_10_acc: 0.9636
Epoch 9/100
80/80 [==============================] - 17s 212ms/step - loss: 2.5814 - acc:
0.8413 - top_5_acc: 0.9530 - top_10_acc: 0.9684
Epoch 10/100
80/80 [==============================] - 17s 213ms/step - loss: 2.5529 - acc:
0.8528 - top_5_acc: 0.9582 - top_10_acc: 0.9721
Epoch 11/100
80/80 [==============================] - 17s 212ms/step - loss: 2.5209 - acc:
0.8652 - top_5_acc: 0.9637 - top_10_acc: 0.9753
Epoch 12/100
80/80 [==============================] - 17s 212ms/step - loss: 2.4854 - acc:
0.8784 - top_5_acc: 0.9682 - top_10_acc: 0.9788
Epoch 13/100
80/80 [==============================] - 17s 212ms/step - loss: 2.4557 - acc:
0.8899 - top_5_acc: 0.9715 - top_10_acc: 0.9814
Epoch 14/100
80/80 [==============================] - 17s 212ms/step - loss: 2.4267 - acc:
0.9001 - top_5_acc: 0.9750 - top_10_acc: 0.9841
Epoch 15/100
80/80 [==============================] - 17s 212ms/step - loss: 2.4016 - acc:
0.9093 - top_5_acc: 0.9784 - top_10_acc: 0.9862
Epoch 16/100
80/80 [==============================] - 17s 213ms/step - loss: 2.3756 - acc:
0.9175 - top_5_acc: 0.9812 - top_10_acc: 0.9879
Epoch 17/100
80/80 [==============================] - 17s 213ms/step - loss: 2.3534 - acc:
0.9265 - top_5_acc: 0.9839 - top_10_acc: 0.9898
Epoch 18/100
80/80 [==============================] - 17s 212ms/step - loss: 2.3355 - acc:
0.9319 - top_5_acc: 0.9859 - top_10_acc: 0.9910
Epoch 19/100
80/80 [==============================] - 17s 213ms/step - loss: 2.3125 - acc:
0.9420 - top_5_acc: 0.9882 - top_10_acc: 0.9927
Epoch 20/100
```

```
80/80 [==============================] - 17s 212ms/step - loss: 2.3044 - acc:
0.9432 - top_5_acc: 0.9892 - top_10_acc: 0.9937
Epoch 21/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2896 - acc:
0.9491 - top_5_acc: 0.9905 - top_10_acc: 0.9948
Epoch 22/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2773 - acc:
0.9524 - top_5_acc: 0.9914 - top_10_acc: 0.9948
Epoch 23/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2823 - acc:
0.9523 - top_5_acc: 0.9921 - top_10_acc: 0.9953
Epoch 24/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2719 - acc:
0.9547 - top_5_acc: 0.9926 - top_10_acc: 0.9959
Epoch 25/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2757 - acc:
0.9541 - top_5_acc: 0.9927 - top_10_acc: 0.9959
Epoch 26/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2646 - acc:
0.9575 - top_5_acc: 0.9941 - top_10_acc: 0.9966
Epoch 27/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2591 - acc:
0.9590 - top_5_acc: 0.9944 - top_10_acc: 0.9970
Epoch 28/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2620 - acc:
0.9582 - top_5_acc: 0.9940 - top_10_acc: 0.9967
Epoch 29/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2553 - acc:
0.9603 - top_5_acc: 0.9952 - top_10_acc: 0.9973
Epoch 30/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2412 - acc:
0.9642 - top_5_acc: 0.9955 - top_10_acc: 0.9980
Epoch 31/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2417 - acc:
0.9638 - top_5_acc: 0.9961 - top_10_acc: 0.9982
Epoch 32/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2300 - acc:
0.9687 - top_5_acc: 0.9965 - top_10_acc: 0.9983
Epoch 33/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2245 - acc:
0.9690 - top_5_acc: 0.9968 - top_10_acc: 0.9985
Epoch 34/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2157 - acc:
0.9719 - top_5_acc: 0.9974 - top_10_acc: 0.9989
Epoch 35/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2132 - acc:
0.9727 - top_5_acc: 0.9979 - top_10_acc: 0.9990
Epoch 36/100
```

```
80/80 [==============================] - 17s 212ms/step - loss: 2.2120 - acc:
0.9730 - top_5_acc: 0.9980 - top_10_acc: 0.9991
Epoch 37/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2113 - acc:
0.9730 - top_5_acc: 0.9976 - top_10_acc: 0.9991
Epoch 38/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1991 - acc:
0.9768 - top_5_acc: 0.9981 - top_10_acc: 0.9993
Epoch 39/100
80/80 [==============================] - 17s 212ms/step - loss: 2.2091 - acc:
0.9746 - top_5_acc: 0.9982 - top_10_acc: 0.9993
Epoch 40/100
80/80 [==============================] - 17s 213ms/step - loss: 2.2064 - acc:
0.9745 - top_5_acc: 0.9981 - top_10_acc: 0.9993
Epoch 41/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1985 - acc:
0.9768 - top_5_acc: 0.9986 - top_10_acc: 0.9995
Epoch 42/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1908 - acc:
0.9786 - top_5_acc: 0.9984 - top_10_acc: 0.9994
Epoch 43/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1864 - acc:
0.9797 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 44/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1894 - acc:
0.9793 - top_5_acc: 0.9987 - top_10_acc: 0.9995
Epoch 45/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1851 - acc:
0.9804 - top_5_acc: 0.9987 - top_10_acc: 0.9995
Epoch 46/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1856 - acc:
0.9799 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 47/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1821 - acc:
0.9809 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 48/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1774 - acc:
0.9813 - top_5_acc: 0.9990 - top_10_acc: 0.9995
Epoch 49/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1757 - acc:
0.9830 - top_5_acc: 0.9990 - top_10_acc: 0.9996
Epoch 50/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1845 - acc:
0.9802 - top_5_acc: 0.9990 - top_10_acc: 0.9995
Epoch 51/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1786 - acc:
0.9818 - top_5_acc: 0.9993 - top_10_acc: 0.9998
Epoch 52/100
```

```
80/80 [==============================] - 17s 212ms/step - loss: 2.1759 - acc:
0.9815 - top_5_acc: 0.9990 - top_10_acc: 0.9996
Epoch 53/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1649 - acc:
0.9839 - top_5_acc: 0.9991 - top_10_acc: 0.9996
Epoch 54/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1734 - acc:
0.9827 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 55/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1685 - acc:
0.9835 - top_5_acc: 0.9991 - top_10_acc: 0.9996
Epoch 56/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1836 - acc:
0.9807 - top_5_acc: 0.9991 - top_10_acc: 0.9997
Epoch 57/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1731 - acc:
0.9830 - top_5_acc: 0.9992 - top_10_acc: 0.9998
Epoch 58/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1624 - acc:
0.9852 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 59/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1498 - acc:
0.9869 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 60/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1421 - acc:
0.9886 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 61/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1522 - acc:
0.9870 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 62/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1498 - acc:
0.9872 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 63/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1590 - acc:
0.9853 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 64/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1571 - acc:
0.9859 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 65/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1620 - acc:
0.9841 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 66/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1584 - acc:
0.9857 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 67/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1479 - acc:
0.9876 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 68/100
```

```
80/80 [==============================] - 17s 213ms/step - loss: 2.1402 - acc:
0.9890 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 69/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1360 - acc:
0.9895 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 70/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1389 - acc:
0.9892 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 71/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1612 - acc:
0.9847 - top_5_acc: 0.9993 - top_10_acc: 0.9997
Epoch 72/100
80/80 [==============================] - 17s 214ms/step - loss: 2.1525 - acc:
0.9873 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 73/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1473 - acc:
0.9869 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 74/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1393 - acc:
0.9890 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 75/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1378 - acc:
0.9889 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 76/100
80/80 [==============================] - 17s 214ms/step - loss: 2.1309 - acc:
0.9899 - top_5_acc: 0.9998 - top_10_acc: 0.9999
Epoch 77/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1278 - acc:
0.9903 - top_5_acc: 0.9998 - top_10_acc: 1.0000
Epoch 78/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1444 - acc:
0.9880 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 79/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1399 - acc:
0.9887 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 80/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1372 - acc:
0.9897 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 81/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1375 - acc:
0.9893 - top_5_acc: 0.9998 - top_10_acc: 0.9999
Epoch 82/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1417 - acc:
0.9879 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 83/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1404 - acc:
0.9891 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 84/100
```

```
80/80 [==============================] - 17s 212ms/step - loss: 2.1333 - acc:
0.9894 - top_5_acc: 0.9998 - top_10_acc: 0.9999
Epoch 85/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1256 - acc:
0.9911 - top_5_acc: 0.9997 - top_10_acc: 1.0000
Epoch 86/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1319 - acc:
0.9896 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 87/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1290 - acc:
0.9902 - top_5_acc: 0.9998 - top_10_acc: 1.0000
Epoch 88/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1353 - acc:
0.9889 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 89/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1451 - acc:
0.9873 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 90/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1329 - acc:
0.9899 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 91/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1321 - acc:
0.9898 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 92/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1384 - acc:
0.9879 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 93/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1364 - acc:
0.9889 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 94/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1353 - acc:
0.9893 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 95/100
80/80 [==============================] - 17s 212ms/step - loss: 2.1267 - acc:
0.9903 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 96/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1231 - acc:
0.9912 - top_5_acc: 0.9997 - top_10_acc: 0.9998
Epoch 97/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1176 - acc:
0.9926 - top_5_acc: 0.9998 - top_10_acc: 0.9999
Epoch 98/100
80/80 [==============================] - 17s 214ms/step - loss: 2.1271 - acc:
0.9907 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 99/100
80/80 [==============================] - 17s 213ms/step - loss: 2.1428 - acc:
0.9881 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 100/100
```

```
80/80 [==============================] - 17s 213ms/step - loss: 2.1362 - acc:
0.9885 - top_5_acc: 0.9996 - top_10_acc: 0.9999
```

[ ]:
```python
y_val_pred = model.predict({ 'frames': X_val, 'non_empty_frame_idxs':␣
↪NON_EMPTY_FRAME_IDXS_VAL }, verbose=2).argmax(axis=1)
```

```
446/446 - 6s - 6s/epoch - 12ms/step
```

[ ]:
```python
y_val_pred.shape
```

[ ]:
```
(14248,)
```

[ ]:
```python
y_val_pred
```

[ ]:
```
array([ 47,  30,  54, …, 145, 142, 238])
```

[ ]:
```python
y_val
```

[ ]:
```
array([ 47,  30,  54, …,  79, 142, 238], dtype=int32)
```

[ ]:
```python
import json

signmap_sub_dir = 'sign_to_prediction_index_map.json'
signmap_full_file_path = os.path.join(googledrive_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)
```

[ ]:
```python
sign_to_index
```

[ ]:
```
{'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,
'grandpa': 100,
'grass': 101,
'green': 102,
'gum': 103,
'hair': 104,
'happy': 105,
'hat': 106,
'hate': 107,
'have': 108,
'haveto': 109,
'head': 110,
'hear': 111,
'helicopter': 112,
```

```
'hello': 113,
'hen': 114,
'hesheit': 115,
'hide': 116,
'high': 117,
'home': 118,
'horse': 119,
'hot': 120,
'hungry': 121,
'icecream': 122,
'if': 123,
'into': 124,
'jacket': 125,
'jeans': 126,
'jump': 127,
'kiss': 128,
'kitty': 129,
'lamp': 130,
'later': 131,
'like': 132,
'lion': 133,
'lips': 134,
'listen': 135,
'look': 136,
'loud': 137,
'mad': 138,
'make': 139,
'man': 140,
'many': 141,
'milk': 142,
'minemy': 143,
'mitten': 144,
'mom': 145,
'moon': 146,
'morning': 147,
'mouse': 148,
'mouth': 149,
'nap': 150,
'napkin': 151,
'night': 152,
'no': 153,
'noisy': 154,
'nose': 155,
'not': 156,
'now': 157,
'nuts': 158,
'old': 159,
```

```
'on': 160,
'open': 161,
'orange': 162,
'outside': 163,
'owie': 164,
'owl': 165,
'pajamas': 166,
'pen': 167,
'pencil': 168,
'penny': 169,
'person': 170,
'pig': 171,
'pizza': 172,
'please': 173,
'police': 174,
'pool': 175,
'potty': 176,
'pretend': 177,
'pretty': 178,
'puppy': 179,
'puzzle': 180,
'quiet': 181,
'radio': 182,
'rain': 183,
'read': 184,
'red': 185,
'refrigerator': 186,
'ride': 187,
'room': 188,
'sad': 189,
'same': 190,
'say': 191,
'scissors': 192,
'see': 193,
'shhh': 194,
'shirt': 195,
'shoe': 196,
'shower': 197,
'sick': 198,
'sleep': 199,
'sleepy': 200,
'smile': 201,
'snack': 202,
'snow': 203,
'stairs': 204,
'stay': 205,
'sticky': 206,
```

```
'store': 207,
'story': 208,
'stuck': 209,
'sun': 210,
'table': 211,
'talk': 212,
'taste': 213,
'thankyou': 214,
'that': 215,
'there': 216,
'think': 217,
'thirsty': 218,
'tiger': 219,
'time': 220,
'tomorrow': 221,
'tongue': 222,
'tooth': 223,
'toothbrush': 224,
'touch': 225,
'toy': 226,
'tree': 227,
'uncle': 228,
'underwear': 229,
'up': 230,
'vacuum': 231,
'wait': 232,
'wake': 233,
'water': 234,
'wet': 235,
'weus': 236,
'where': 237,
'white': 238,
'who': 239,
'why': 240,
'will': 241,
'wolf': 242,
'yellow': 243,
'yes': 244,
'yesterday': 245,
'yourself': 246,
'yucky': 247,
'zebra': 248,
'zipper': 249}
```

[ ]:
```
inverted_mapping = {v: k for k, v in sign_to_index.items()}
```

```
[ ]: # Convert mapping to list
class_names = [inverted_mapping[i] for i in sorted(inverted_mapping)]
```

```
[ ]: from sklearn.metrics import classification_report

print(classification_report(y_val, y_val_pred, target_names=class_names))
```

```
              precision    recall  f1-score   support

          TV       0.76      0.95      0.84        60
       after       0.39      0.37      0.38        59
    airplane       0.98      1.00      0.99        57
         all       0.78      0.64      0.70        59
   alligator       0.56      0.77      0.65        61
      animal       0.63      0.54      0.58        48
     another       0.61      0.69      0.65        51
         any       0.56      0.76      0.65        58
       apple       0.94      0.89      0.91        53
         arm       0.78      0.39      0.52        54
        aunt       0.72      0.65      0.69        55
       awake       0.44      0.26      0.33        57
    backyard       0.76      0.59      0.67        59
         bad       0.80      0.67      0.73        60
     balloon       0.70      0.79      0.75        63
        bath       0.65      0.96      0.78        54
     because       0.88      0.83      0.85        59
         bed       0.62      0.96      0.76        55
     bedroom       0.71      0.61      0.65        61
         bee       0.76      0.63      0.69        60
      before       0.73      0.61      0.67        54
      beside       0.64      0.49      0.55        47
      better       0.90      0.77      0.83        57
        bird       0.73      0.97      0.83        58
       black       0.93      0.88      0.90        58
        blow       0.75      0.88      0.81        59
        blue       0.83      0.71      0.76        55
        boat       0.68      0.81      0.74        53
        book       0.71      0.88      0.78        56
         boy       0.92      0.91      0.92        54
     brother       0.84      0.88      0.86        60
       brown       0.84      0.98      0.90        58
         bug       0.67      0.90      0.77        62
         bye       0.66      0.75      0.70        60
 callonphone       1.00      0.72      0.84        57
         can       0.52      0.57      0.54        54
         car       0.79      0.63      0.70        54
      carrot       0.96      0.45      0.62        55
         cat       0.54      0.66      0.59        67
```

cereal	0.79	0.54	0.64	56
chair	0.68	0.85	0.75	52
cheek	0.90	0.74	0.81	61
child	0.76	0.54	0.63	52
chin	0.75	0.78	0.76	58
chocolate	0.83	0.91	0.87	58
clean	0.43	0.87	0.57	53
close	0.76	0.38	0.51	65
closet	0.88	0.57	0.69	61
cloud	0.74	0.75	0.75	61
clown	0.87	0.87	0.87	61
cow	0.92	0.95	0.93	60
cowboy	0.84	0.83	0.83	58
cry	0.87	0.84	0.86	57
cut	0.60	0.79	0.68	57
cute	0.95	0.92	0.93	59
dad	0.81	0.98	0.89	57
dance	0.84	0.70	0.76	46
dirty	0.95	0.75	0.84	53
dog	0.71	0.57	0.63	56
doll	0.98	0.90	0.94	61
donkey	0.88	0.97	0.92	59
down	0.74	0.81	0.78	43
drawer	0.66	0.67	0.67	58
drink	0.90	0.98	0.94	57
drop	0.78	0.43	0.55	49
dry	0.79	0.71	0.75	59
dryer	0.84	0.77	0.80	60
duck	0.64	0.84	0.73	57
ear	0.97	0.48	0.64	60
elephant	0.68	0.58	0.62	59
empty	0.69	0.62	0.65	58
every	0.47	0.48	0.48	50
eye	0.82	0.78	0.80	54
face	0.54	0.72	0.61	53
fall	0.86	0.62	0.72	58
farm	0.87	0.67	0.76	58
fast	0.45	0.27	0.34	51
feet	0.88	0.63	0.74	60
find	0.31	0.56	0.40	59
fine	0.52	0.89	0.66	53
finger	0.72	0.57	0.64	60
finish	0.69	0.78	0.73	60
fireman	0.96	0.83	0.89	60
first	0.85	0.78	0.82	60
fish	0.59	0.71	0.65	56
flag	0.75	0.97	0.84	58
flower	0.93	0.88	0.90	57

food	0.90	0.96	0.93	57
for	0.96	0.95	0.96	57
frenchfries	0.56	0.98	0.71	60
frog	0.98	0.91	0.94	54
garbage	0.97	0.58	0.72	50
gift	0.58	0.84	0.69	58
giraffe	0.91	0.59	0.72	54
girl	0.55	0.42	0.48	52
give	0.59	0.25	0.35	53
glasswindow	0.50	0.35	0.41	54
go	0.86	0.23	0.36	53
goose	0.60	0.05	0.09	62
grandma	0.58	0.75	0.66	57
grandpa	0.98	0.83	0.90	60
grass	0.46	0.47	0.46	60
green	0.79	0.91	0.85	55
gum	0.89	0.93	0.91	59
hair	0.83	0.60	0.70	58
happy	0.82	0.90	0.86	60
hat	0.80	0.81	0.80	43
hate	0.49	0.67	0.57	58
have	0.68	0.88	0.77	49
haveto	0.66	0.85	0.75	55
head	0.79	0.91	0.85	58
hear	0.57	0.56	0.56	68
helicopter	0.66	0.47	0.55	57
hello	0.62	0.79	0.69	57
hen	0.52	0.79	0.63	61
hesheit	0.63	0.77	0.69	60
hide	0.48	0.57	0.52	51
high	0.88	0.93	0.90	54
home	0.98	0.81	0.89	59
horse	0.90	1.00	0.94	60
hot	0.76	0.85	0.80	60
hungry	0.87	0.81	0.84	59
icecream	0.91	0.72	0.80	60
if	0.84	0.93	0.88	55
into	0.67	0.69	0.68	54
jacket	0.70	0.72	0.71	58
jeans	0.76	0.55	0.64	56
jump	0.79	0.63	0.70	59
kiss	0.71	0.63	0.67	59
kitty	0.34	0.26	0.30	65
lamp	0.71	0.59	0.64	61
later	0.59	0.78	0.67	58
like	0.66	0.92	0.77	64
lion	0.90	0.84	0.87	55
lips	0.47	0.58	0.52	60

listen	0.76	0.56	0.65	57
look	0.77	0.44	0.56	62
loud	0.77	0.49	0.60	61
mad	0.63	0.83	0.72	58
make	0.60	0.83	0.70	59
man	0.81	0.78	0.79	59
many	0.40	0.64	0.49	50
milk	0.70	0.78	0.74	58
minemy	0.70	0.97	0.81	60
mitten	0.94	0.48	0.64	62
mom	0.58	0.91	0.71	58
moon	0.93	0.95	0.94	59
morning	0.85	0.93	0.89	55
mouse	0.95	0.92	0.93	60
mouth	0.52	0.22	0.31	60
nap	0.59	0.15	0.24	65
napkin	0.73	0.45	0.56	60
night	0.79	0.58	0.67	45
no	0.68	0.85	0.76	61
noisy	0.92	0.41	0.57	56
nose	0.89	0.88	0.89	58
not	0.59	0.90	0.71	61
now	0.69	0.94	0.80	52
nuts	0.60	0.76	0.67	63
old	0.89	0.63	0.74	54
on	0.78	0.73	0.75	59
open	0.66	0.78	0.71	49
orange	0.94	0.79	0.86	58
outside	0.55	0.48	0.51	61
owie	0.63	0.30	0.40	57
owl	0.96	0.95	0.96	57
pajamas	0.87	0.74	0.80	54
pen	0.69	0.34	0.46	64
pencil	0.61	0.58	0.60	60
penny	0.61	0.46	0.53	54
person	0.36	0.40	0.38	35
pig	0.89	0.92	0.91	53
pizza	0.58	0.38	0.46	66
please	0.90	0.78	0.84	60
police	0.98	0.87	0.92	60
pool	0.90	0.48	0.63	56
potty	0.81	0.94	0.87	53
pretend	0.82	0.84	0.83	61
pretty	0.56	0.52	0.54	58
puppy	0.80	0.52	0.63	62
puzzle	0.68	0.78	0.73	50
quiet	0.70	0.66	0.68	61
radio	0.89	0.65	0.75	60

rain	0.87	0.65	0.74	60
read	0.82	0.75	0.78	53
red	0.90	0.74	0.81	58
refrigerator	0.88	0.53	0.66	55
ride	0.90	0.34	0.49	53
room	0.44	0.67	0.53	60
sad	0.77	0.92	0.84	60
same	0.81	0.80	0.80	59
say	0.38	0.64	0.48	55
scissors	0.60	0.73	0.66	60
see	0.62	0.98	0.76	60
shhh	0.84	0.95	0.89	60
shirt	0.88	0.93	0.90	54
shoe	0.92	0.90	0.91	60
shower	0.72	0.62	0.67	55
sick	0.87	0.70	0.77	56
sleep	0.37	0.67	0.48	57
sleepy	0.54	0.71	0.61	63
smile	0.73	0.73	0.73	56
snack	0.81	0.60	0.69	58
snow	0.94	0.84	0.89	58
stairs	0.83	0.63	0.72	60
stay	0.69	0.75	0.72	56
sticky	0.40	0.79	0.53	53
store	0.82	0.93	0.87	55
story	0.94	0.56	0.70	59
stuck	0.90	0.95	0.92	56
sun	0.62	0.50	0.55	58
table	0.71	0.92	0.80	51
talk	0.97	0.50	0.66	58
taste	0.73	0.91	0.81	57
thankyou	0.38	0.90	0.53	52
that	0.58	0.60	0.59	55
there	0.52	0.46	0.49	50
think	0.58	0.76	0.66	55
thirsty	0.93	0.86	0.89	49
tiger	0.89	0.91	0.90	56
time	0.82	0.84	0.83	50
tomorrow	0.87	0.49	0.63	55
tongue	0.49	0.81	0.61	58
tooth	0.51	0.62	0.56	53
toothbrush	0.67	0.67	0.67	57
touch	0.68	0.60	0.64	57
toy	0.98	0.80	0.88	55
tree	0.81	0.84	0.82	56
uncle	0.93	0.87	0.90	60
underwear	0.86	0.62	0.72	61
up	0.55	0.87	0.68	60

vacuum	0.45	0.48	0.47	52
wait	0.51	0.69	0.59	51
wake	0.41	0.64	0.50	58
water	0.81	0.95	0.87	62
wet	0.46	0.45	0.46	66
weus	0.69	0.88	0.78	57
where	0.80	0.93	0.86	56
white	0.88	0.73	0.80	60
who	0.86	0.80	0.83	60
why	0.82	0.71	0.76	59
will	0.87	0.59	0.70	56
wolf	0.71	0.61	0.66	57
yellow	0.80	0.83	0.81	58
yes	0.63	0.74	0.68	62
yesterday	0.78	0.65	0.71	60
yourself	0.88	0.75	0.81	57
yucky	0.44	0.34	0.38	56
zebra	0.73	0.87	0.79	61
zipper	0.75	0.66	0.70	32
accuracy			0.71	14248
macro avg	0.74	0.71	0.71	14248
weighted avg	0.74	0.71	0.71	14248

The overall ASL Transformer designed by Wijkhuizen, M. are shown in APPENDIX 1. After training and valuation, the Model performance is shown in APPENDIX 2. Wijkhuizen, M.'s transformer model has an overall 0.71 F1 score with a weighted precision of 0.74 and a weighted recall of 0.71. It has outperformed any other model types that we tried. Some ASL word predictions perform better than others; for example, airplane, apple, owl etc., have F1 scores higher than 0.90. and other words like kitty, yucky, and suffer under the F1 score lower than 0.40. However, most words' F1 scores are higher than 0.60, so we could use this model for a real-life application with some limitations.

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

Kumar, A. (2023, February 1). The transformer model and its applications: Understanding attention mechanism. Medium. Retrieved from https://medium.com/@mittal.atul06/the-transformer-model-and-its-applications-understanding-attention-mechanism-c37e6e3a76dc

Huang, Z., Liang, D., Xu, P., & Xiang, B. (2020). Improve Transformer Models with Better Relative Position Embeddings. arXiv preprint arXiv:2009.13658