Model_Training

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.

After Modeling.ipynb, we have following result: 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.

Then we are using full dataset to train the transfomer model again, and save the model weight to model.h5, also the mean and standard divation for lip, hand, and pose will be save to numpy file for loading later:

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

```
                                611.8/611.8
kB 8.6 MB/s eta 0:00:00
```

[ ]:
```
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
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(
```

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

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

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

[ ]:
```
X_train = np.load(f'{googledrive_dir}/X.npy')
y_train = np.load(f'{googledrive_dir}/y.npy')
NON_EMPTY_FRAME_IDXS_TRAIN = np.load(f'{googledrive_dir}/NON_EMPTY_FRAME_IDXS.
 ↪npy')
```

[ ]:
```
# 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}')
```

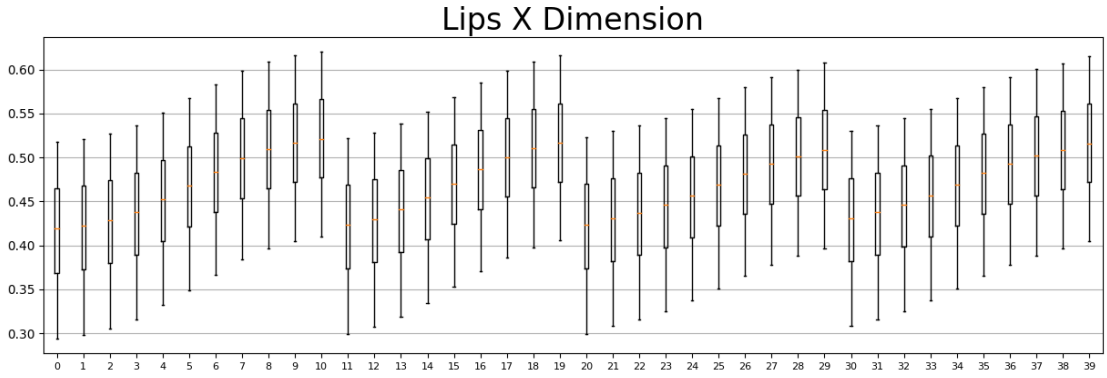
```
# HAND_IDXS: 21, N_COLS: 66
```

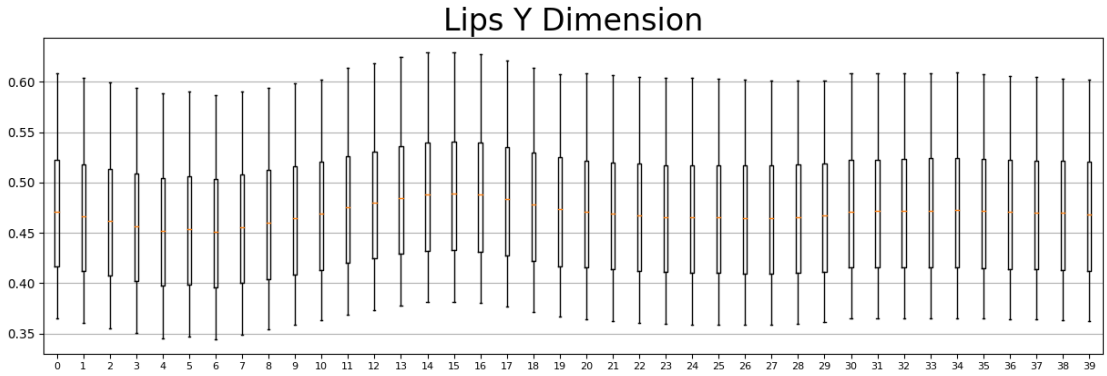
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
 ↪gislr-tf-data-processing-transformer-training
LIPS_START = 0
LEFT_HAND_START = LIPS_IDXS.size
RIGHT_HAND_START = LEFT_HAND_START + LEFT_HAND_IDXS.size
POSE_START = RIGHT_HAND_START + RIGHT_HAND_IDXS.size

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

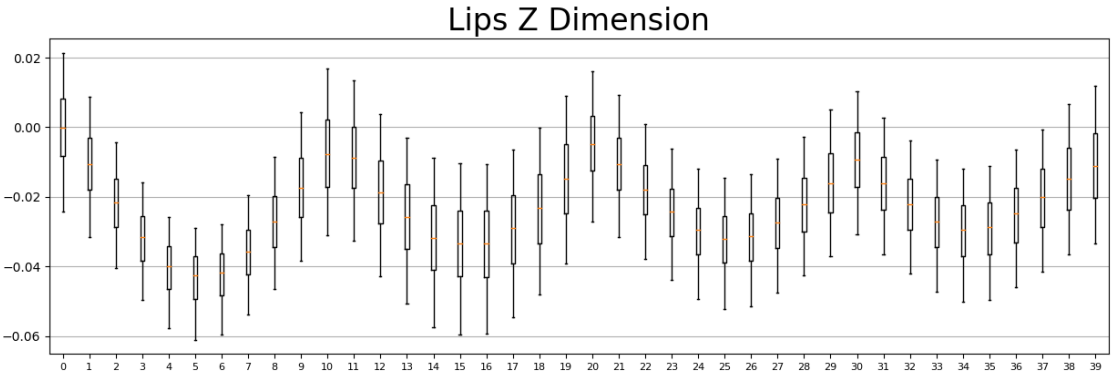
```
LIPS_START: 0, LEFT_HAND_START: 40, RIGHT_HAND_START: 61, POSE_START: 61
```

[ ]:
```
# 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]
```



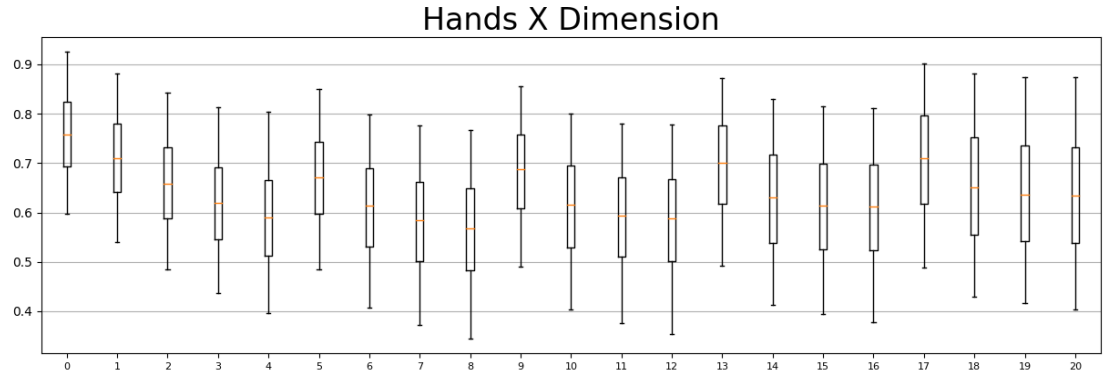


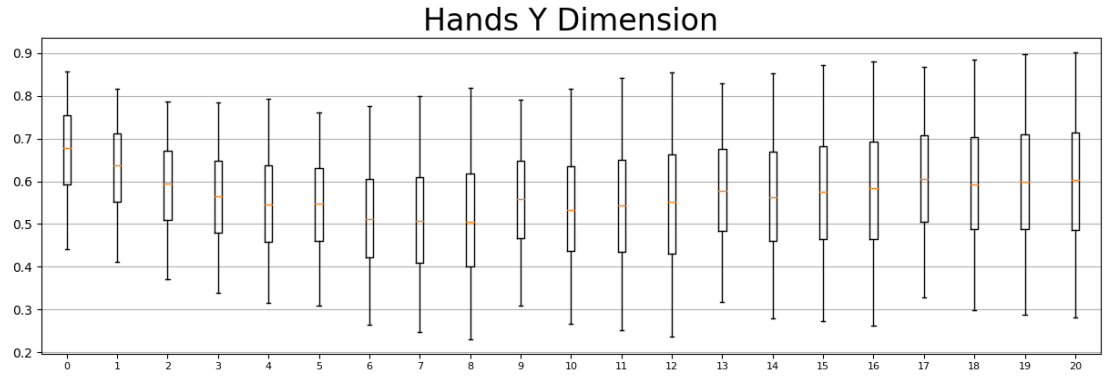


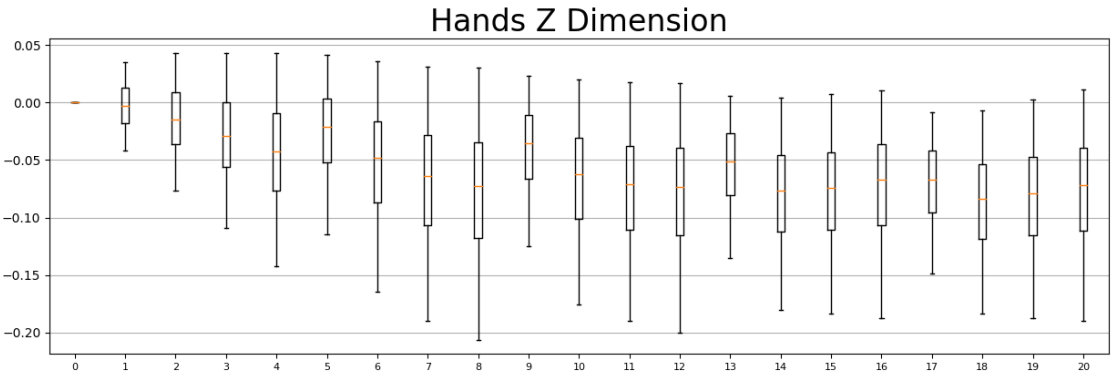
[ ]:
```
# Code From https://www.kaggle.com/code/markwijkhuizen/
  ↪gislr-tf-data-processing-transformer-training
def get_left_right_hand_mean_std():
    # LEFT HAND
    LEFT_HANDS_MEAN_X = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)
    LEFT_HANDS_MEAN_Y = np.zeros([LEFT_HAND_IDXS.size], dtype=np.float32)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



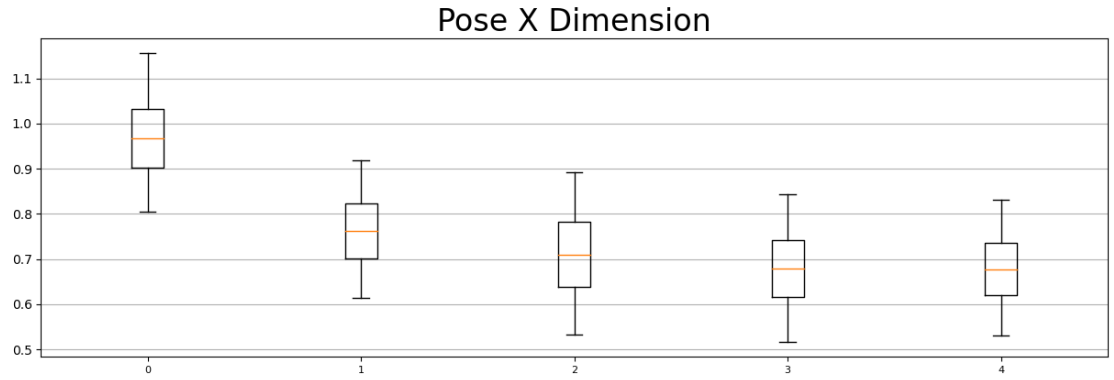


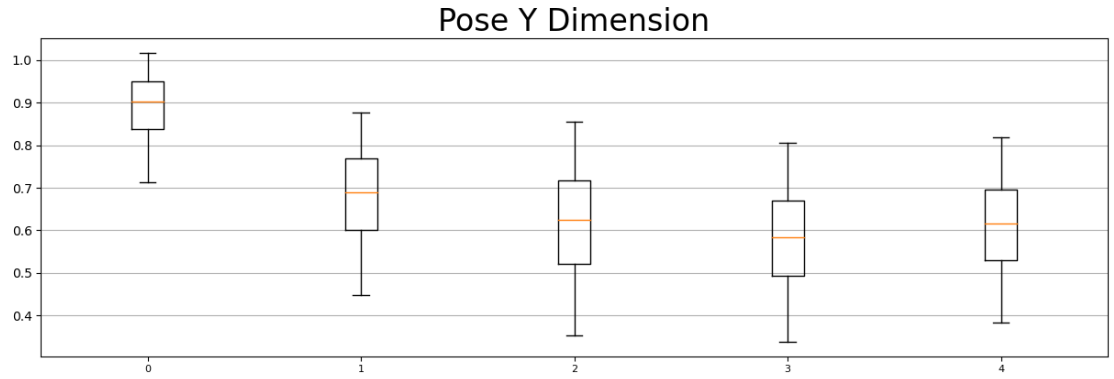


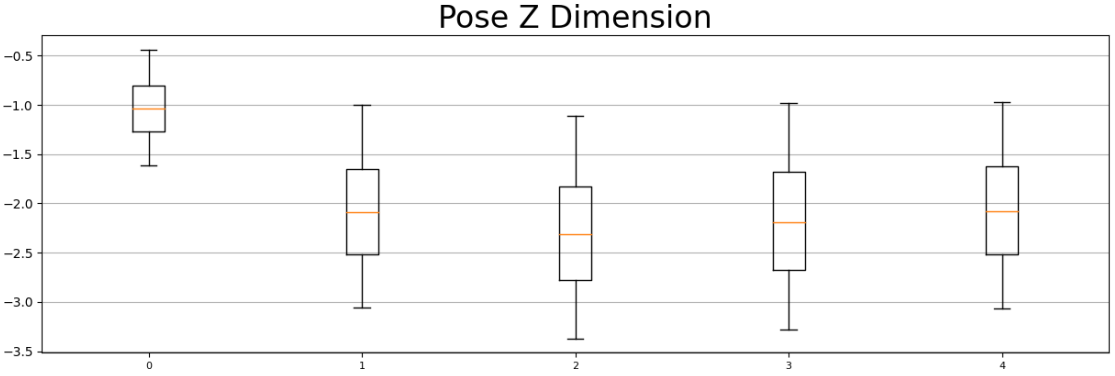
[ ]:
```python
# Code From https://www.kaggle.com/code/markwijkhuizen/
  ↪gislr-tf-data-processing-transformer-training
def get_pose_mean_std():
    # POSE
    POSE_MEAN_X = np.zeros([POSE_IDXS.size], dtype=np.float32)
    POSE_MEAN_Y = np.zeros([POSE_IDXS.size], dtype=np.float32)
    POSE_STD_X = np.zeros([POSE_IDXS.size],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







[ ]:
```
LIPS_STD
```

[ ]:
```
array([[0.06913371, 0.0749447 ],
       [0.06869563, 0.07505549],
       [0.06836782, 0.07519231],
       [0.06812248, 0.07535139],
       [0.06790033, 0.07548924],
```

```
       [0.06740177, 0.07569747],
       [0.06690664, 0.07540744],
       [0.06630632, 0.0751383 ],
       [0.06579747, 0.07488327],
       [0.06536146, 0.07465134],
       [0.06490868, 0.07446797],
       [0.0686974 , 0.07510085],
       [0.06826583, 0.07542164],
       [0.06773155, 0.07593741],
       [0.06713565, 0.07637213],
       [0.06652237, 0.07647745],
       [0.06590941, 0.07628901],
       [0.06545945, 0.07577442],
       [0.06515191, 0.07514609],
       [0.06499287, 0.0747105 ],
       [0.06892715, 0.0749959 ],
       [0.06840112, 0.07515378],
       [0.0680631 , 0.07531367],
       [0.06770497, 0.07549456],
       [0.0673436 , 0.07565831],
       [0.06693598, 0.07573235],
       [0.06640731, 0.07556409],
       [0.06595178, 0.07529225],
       [0.06554479, 0.07501528],
       [0.06529062, 0.07478278],
       [0.0683827 , 0.07500488],
       [0.0680342 , 0.07509418],
       [0.06765123, 0.07527245],
       [0.06726003, 0.07546542],
       [0.06684113, 0.07556838],
       [0.06633036, 0.07539269],
       [0.06588241, 0.07511063],
       [0.06549872, 0.07483311],
       [0.06523801, 0.07465854],
       [0.06498756, 0.07455516]], dtype=float32)
```

[ ]:
```
np.save(os.path.join(googledrive_dir, 'LIPS_MEAN.npy'), LIPS_MEAN)
np.save(os.path.join(googledrive_dir, 'LIPS_STD.npy'), LIPS_STD)
np.save(os.path.join(googledrive_dir, 'LEFT_HANDS_MEAN.npy'), LEFT_HANDS_MEAN)
np.save(os.path.join(googledrive_dir, 'LEFT_HANDS_STD.npy'), LEFT_HANDS_STD)
np.save(os.path.join(googledrive_dir, 'POSE_MEAN.npy'), POSE_MEAN)
np.save(os.path.join(googledrive_dir, 'POSE_STD.npy'), POSE_STD)
```

[ ]:
```
# 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
```

[ ]:
```
# 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),
            )
```

[ ]:
```
# 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
```

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

[ ]:
```python
# 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
94/94 [==============================] - 42s 288ms/step - loss: 4.2173 - acc:
0.3037 - top_5_acc: 0.5661 - top_10_acc: 0.6701
Epoch 2/100
94/94 [==============================] - 27s 289ms/step - loss: 3.2221 - acc:
0.6053 - top_5_acc: 0.8466 - top_10_acc: 0.8972
Epoch 3/100
94/94 [==============================] - 27s 289ms/step - loss: 2.9747 - acc:
0.6914 - top_5_acc: 0.8920 - top_10_acc: 0.9267
Epoch 4/100
94/94 [==============================] - 27s 289ms/step - loss: 2.8269 - acc:
0.7456 - top_5_acc: 0.9165 - top_10_acc: 0.9434
Epoch 5/100
94/94 [==============================] - 27s 290ms/step - loss: 2.7447 - acc:
0.7771 - top_5_acc: 0.9294 - top_10_acc: 0.9516
Epoch 6/100
94/94 [==============================] - 27s 289ms/step - loss: 2.6789 - acc:
0.8014 - top_5_acc: 0.9391 - top_10_acc: 0.9584
Epoch 7/100
94/94 [==============================] - 27s 289ms/step - loss: 2.6201 - acc:
0.8235 - top_5_acc: 0.9478 - top_10_acc: 0.9649
Epoch 8/100
94/94 [==============================] - 27s 289ms/step - loss: 2.5751 - acc:
0.8415 - top_5_acc: 0.9536 - top_10_acc: 0.9686
Epoch 9/100
94/94 [==============================] - 27s 290ms/step - loss: 2.5478 - acc:
0.8520 - top_5_acc: 0.9576 - top_10_acc: 0.9714
Epoch 10/100
94/94 [==============================] - 27s 290ms/step - loss: 2.5127 - acc:
0.8654 - top_5_acc: 0.9631 - top_10_acc: 0.9758
Epoch 11/100
94/94 [==============================] - 27s 290ms/step - loss: 2.4809 - acc:
0.8769 - top_5_acc: 0.9666 - top_10_acc: 0.9783
Epoch 12/100
94/94 [==============================] - 27s 290ms/step - loss: 2.4417 - acc:
0.8904 - top_5_acc: 0.9726 - top_10_acc: 0.9819
Epoch 13/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.4155 - acc:
0.9008 - top_5_acc: 0.9757 - top_10_acc: 0.9844
Epoch 14/100
94/94 [==============================] - 27s 289ms/step - loss: 2.3976 - acc:
0.9080 - top_5_acc: 0.9781 - top_10_acc: 0.9854
Epoch 15/100
94/94 [==============================] - 27s 289ms/step - loss: 2.3756 - acc:
0.9167 - top_5_acc: 0.9802 - top_10_acc: 0.9877
Epoch 16/100
94/94 [==============================] - 27s 289ms/step - loss: 2.3633 - acc:
0.9186 - top_5_acc: 0.9829 - top_10_acc: 0.9890
Epoch 17/100
94/94 [==============================] - 27s 289ms/step - loss: 2.3410 - acc:
0.9281 - top_5_acc: 0.9855 - top_10_acc: 0.9910
Epoch 18/100
94/94 [==============================] - 27s 289ms/step - loss: 2.3249 - acc:
0.9336 - top_5_acc: 0.9872 - top_10_acc: 0.9918
Epoch 19/100
94/94 [==============================] - 27s 290ms/step - loss: 2.3132 - acc:
0.9374 - top_5_acc: 0.9881 - top_10_acc: 0.9925
Epoch 20/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2979 - acc:
0.9426 - top_5_acc: 0.9896 - top_10_acc: 0.9935
Epoch 21/100
94/94 [==============================] - 27s 290ms/step - loss: 2.3075 - acc:
0.9397 - top_5_acc: 0.9897 - top_10_acc: 0.9936
Epoch 22/100
94/94 [==============================] - 27s 290ms/step - loss: 2.3025 - acc:
0.9417 - top_5_acc: 0.9903 - top_10_acc: 0.9942
Epoch 23/100
94/94 [==============================] - 27s 290ms/step - loss: 2.3011 - acc:
0.9415 - top_5_acc: 0.9900 - top_10_acc: 0.9943
Epoch 24/100
94/94 [==============================] - 27s 290ms/step - loss: 2.2876 - acc:
0.9467 - top_5_acc: 0.9918 - top_10_acc: 0.9953
Epoch 25/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2829 - acc:
0.9479 - top_5_acc: 0.9916 - top_10_acc: 0.9953
Epoch 26/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2687 - acc:
0.9527 - top_5_acc: 0.9929 - top_10_acc: 0.9959
Epoch 27/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2690 - acc:
0.9524 - top_5_acc: 0.9933 - top_10_acc: 0.9964
Epoch 28/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2601 - acc:
0.9558 - top_5_acc: 0.9935 - top_10_acc: 0.9964
Epoch 29/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.2520 - acc:
0.9573 - top_5_acc: 0.9946 - top_10_acc: 0.9972
Epoch 30/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2379 - acc:
0.9621 - top_5_acc: 0.9955 - top_10_acc: 0.9977
Epoch 31/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2370 - acc:
0.9622 - top_5_acc: 0.9955 - top_10_acc: 0.9979
Epoch 32/100
94/94 [==============================] - 27s 290ms/step - loss: 2.2318 - acc:
0.9652 - top_5_acc: 0.9961 - top_10_acc: 0.9980
Epoch 33/100
94/94 [==============================] - 27s 290ms/step - loss: 2.2305 - acc:
0.9648 - top_5_acc: 0.9962 - top_10_acc: 0.9982
Epoch 34/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2247 - acc:
0.9660 - top_5_acc: 0.9964 - top_10_acc: 0.9982
Epoch 35/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2205 - acc:
0.9678 - top_5_acc: 0.9966 - top_10_acc: 0.9984
Epoch 36/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2179 - acc:
0.9688 - top_5_acc: 0.9971 - top_10_acc: 0.9987
Epoch 37/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2055 - acc:
0.9722 - top_5_acc: 0.9974 - top_10_acc: 0.9987
Epoch 38/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2027 - acc:
0.9727 - top_5_acc: 0.9977 - top_10_acc: 0.9989
Epoch 39/100
94/94 [==============================] - 27s 289ms/step - loss: 2.2105 - acc:
0.9707 - top_5_acc: 0.9973 - top_10_acc: 0.9988
Epoch 40/100
94/94 [==============================] - 27s 290ms/step - loss: 2.2054 - acc:
0.9711 - top_5_acc: 0.9976 - top_10_acc: 0.9989
Epoch 41/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1922 - acc:
0.9750 - top_5_acc: 0.9982 - top_10_acc: 0.9993
Epoch 42/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1982 - acc:
0.9738 - top_5_acc: 0.9981 - top_10_acc: 0.9992
Epoch 43/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1949 - acc:
0.9749 - top_5_acc: 0.9982 - top_10_acc: 0.9992
Epoch 44/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1857 - acc:
0.9764 - top_5_acc: 0.9984 - top_10_acc: 0.9994
Epoch 45/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.1912 - acc:
0.9758 - top_5_acc: 0.9983 - top_10_acc: 0.9993
Epoch 46/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1903 - acc:
0.9758 - top_5_acc: 0.9984 - top_10_acc: 0.9993
Epoch 47/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1784 - acc:
0.9786 - top_5_acc: 0.9987 - top_10_acc: 0.9996
Epoch 48/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1704 - acc:
0.9811 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 49/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1703 - acc:
0.9803 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 50/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1741 - acc:
0.9802 - top_5_acc: 0.9988 - top_10_acc: 0.9996
Epoch 51/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1773 - acc:
0.9787 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 52/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1895 - acc:
0.9758 - top_5_acc: 0.9983 - top_10_acc: 0.9994
Epoch 53/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1776 - acc:
0.9786 - top_5_acc: 0.9989 - top_10_acc: 0.9995
Epoch 54/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1705 - acc:
0.9809 - top_5_acc: 0.9989 - top_10_acc: 0.9996
Epoch 55/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1683 - acc:
0.9813 - top_5_acc: 0.9990 - top_10_acc: 0.9997
Epoch 56/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1692 - acc:
0.9815 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 57/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1559 - acc:
0.9844 - top_5_acc: 0.9994 - top_10_acc: 0.9997
Epoch 58/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1526 - acc:
0.9845 - top_5_acc: 0.9994 - top_10_acc: 0.9997
Epoch 59/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1511 - acc:
0.9845 - top_5_acc: 0.9993 - top_10_acc: 0.9997
Epoch 60/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1535 - acc:
0.9843 - top_5_acc: 0.9993 - top_10_acc: 0.9997
Epoch 61/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.1590 - acc:
0.9825 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 62/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1572 - acc:
0.9833 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 63/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1666 - acc:
0.9821 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 64/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1678 - acc:
0.9811 - top_5_acc: 0.9991 - top_10_acc: 0.9997
Epoch 65/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1628 - acc:
0.9815 - top_5_acc: 0.9993 - top_10_acc: 0.9997
Epoch 66/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1541 - acc:
0.9847 - top_5_acc: 0.9993 - top_10_acc: 0.9998
Epoch 67/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1525 - acc:
0.9843 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 68/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1575 - acc:
0.9831 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 69/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1523 - acc:
0.9844 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 70/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1410 - acc:
0.9874 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 71/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1442 - acc:
0.9857 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 72/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1475 - acc:
0.9847 - top_5_acc: 0.9993 - top_10_acc: 0.9998
Epoch 73/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1403 - acc:
0.9864 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 74/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1412 - acc:
0.9866 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 75/100
94/94 [==============================] - 27s 288ms/step - loss: 2.1335 - acc:
0.9878 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 76/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1281 - acc:
0.9895 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 77/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.1360 - acc:
0.9873 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 78/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1532 - acc:
0.9840 - top_5_acc: 0.9995 - top_10_acc: 0.9999
Epoch 79/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1564 - acc:
0.9834 - top_5_acc: 0.9993 - top_10_acc: 0.9998
Epoch 80/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1567 - acc:
0.9831 - top_5_acc: 0.9992 - top_10_acc: 0.9997
Epoch 81/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1417 - acc:
0.9863 - top_5_acc: 0.9994 - top_10_acc: 0.9998
Epoch 82/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1382 - acc:
0.9867 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 83/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1365 - acc:
0.9868 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 84/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1304 - acc:
0.9885 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 85/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1263 - acc:
0.9893 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 86/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1250 - acc:
0.9895 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 87/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1359 - acc:
0.9866 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 88/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1266 - acc:
0.9887 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 89/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1296 - acc:
0.9888 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 90/100
94/94 [==============================] - 27s 290ms/step - loss: 2.1210 - acc:
0.9897 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 91/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1182 - acc:
0.9900 - top_5_acc: 0.9998 - top_10_acc: 0.9999
Epoch 92/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1162 - acc:
0.9908 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 93/100
```

```
94/94 [==============================] - 27s 289ms/step - loss: 2.1208 - acc:
0.9899 - top_5_acc: 0.9997 - top_10_acc: 0.9999
Epoch 94/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1252 - acc:
0.9887 - top_5_acc: 0.9996 - top_10_acc: 0.9998
Epoch 95/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1257 - acc:
0.9892 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 96/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1282 - acc:
0.9881 - top_5_acc: 0.9994 - top_10_acc: 0.9997
Epoch 97/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1302 - acc:
0.9875 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 98/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1324 - acc:
0.9881 - top_5_acc: 0.9995 - top_10_acc: 0.9998
Epoch 99/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1339 - acc:
0.9868 - top_5_acc: 0.9996 - top_10_acc: 0.9999
Epoch 100/100
94/94 [==============================] - 27s 289ms/step - loss: 2.1351 - acc:
0.9865 - top_5_acc: 0.9996 - top_10_acc: 0.9999
```

[ ]:
```
model.save_weights('/content/drive/MyDrive/Colab Notebooks/Data/asl-signs/
↪transformerModel')
```

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