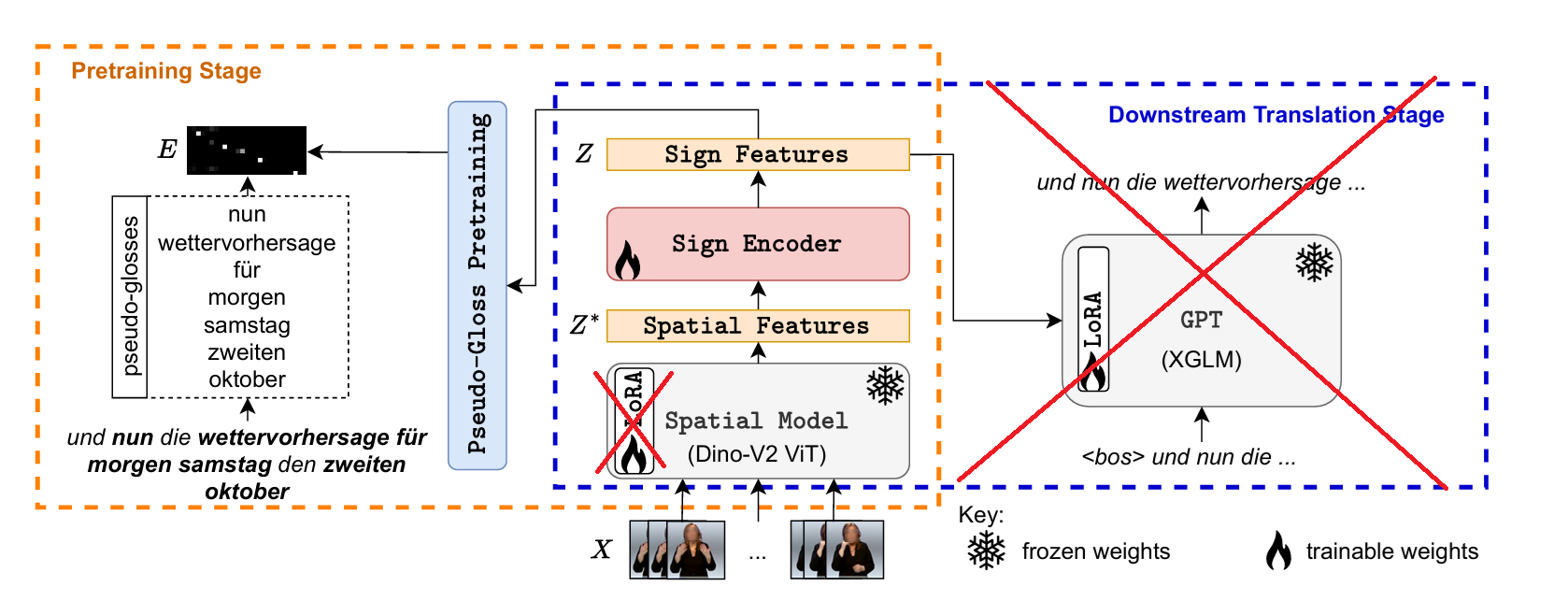
**BBM480   
PROGRESS REPORT**

Tamer Yeşildağ

Eray Taymaz

**1-Model Architecture**



1- Input Layer

* This layer is used for linear projection to hidden size
  + self.input\_projection = nn.Linear(input\_dim, hidden\_size)
  + input\_dim is set to 384 which is embedding size of DinoV2’s output
  + hidden\_size has 2 different values, 512 and 300
  + 512 is used if model has a linear layer at the end of forwarding
  + 300 is desired output dimensions because it is the dimensions of prototypes, in order to simplify encoder model the linear layer at the end of forwarding is discarded and hidden\_size is set to 300

2- Positional Encoding

* Uses custom sinusoidal positional encodings to inject temporal information.
  + self.positional\_encoding = PositionalEncoding(size=hidden\_size)

3- Embedding Dropout

* Regularization with dropout applied after adding positional encodings.
  + self.emb\_dropout = nn.Dropout(p=emb\_dropout)

4- Transformer Encoder

* Based on PyTorch’s nn.TransformerEncoder
  + num\_layer
  + Each layer has:
    - hidden\_size
    - ff\_size
    - num\_heads
    - dropout
    - layer\_norm\_eps = 1e-6
    - batch\_first = True
    - norm\_first = True

5- Output Projection

* Projects the final hidden states to a 300-dimensional space
  + self.ouput\_projection = nn.Linear(hidden\_size, 300)

7- Forward Pass

def forward(self, src, src\_key\_padding\_mask, src\_mask=None):

x = self.input\_projection(src)

x = self.positional\_encoding(x)

x = self.emb\_dropout(x)

x = self.transformer\_encoder(x, src\_key\_padding\_mask=src\_key\_padding\_mask)

x = self.ouput\_projection(x)

return x

**Pre-Training Module Architecture**

1- Encoder

* self.encoder = SignTransformerEncoder()
* Takes input: [batch\_size, T, 384]
* Outputs: [batch\_size, T, 300]

2- Learnable Prototypes

* self.prototypes = nn.Parameter(prototypes)
* Shape: [300, num\_prototypes]
* Represents pseudo-glosses as learnable 300-dimensional vectors.

3- Temperature Scalars

* self.tau\_t = nn.Parameter(torch.tensor(0.1))
* self.tau\_u = nn.Parameter(torch.tensor(0.1))
* Learnable temperature parameters used in softmax operations

4- Forward Pass

1. encoded\_features = self.encoder(sign\_features, masks)
2. encoded\_features\_norm = F.normalize(encoded\_features, dim=-1) # along 300-dim
3. prototypes\_norm = F.normalize(self.prototypes, dim=0) # across 300-dim
4. similarity = torch.einsum("btd,du->btu", encoded\_features\_norm, prototypes\_norm)
   1. Cosine similarity between encoded frame features and prototype vectors.
5. temporal\_prob = F.softmax(similarity / self.tau\_t, dim=1) # Over time
6. prototype\_prob = F.softmax(similarity / self.tau\_u, dim=2) # Over prototypes
7. localization\_matrix = temporal\_prob \* prototype\_prob
   1. Highlights which frame-prototype pairs are most important.
8. loss = F.binary\_cross\_entropy(aggregated\_probs, labels)

**2-Hyperparameter Details**

Model was tested with different combinations of hyperparameters including the change of hidden size by implementing a linear layer for output projection. These both experiments used schedulers.

The scheduler parameters:

* max\_lr=3e-4
* epochs=num\_epochs
* steps\_per\_epoch=len(train\_dataloader)
* anneal\_strategy="cos"
* pct\_start=0.05
* div\_factor=100
* final\_div\_factor=100

Base parameters given by Sign2GPT.

This graph belongs to our experiment with Sign2GPT’s parameters which is:

* ff\_size=2048
* layer=4
* dropout=0.2
* num\_heads=8
* hidden\_size=512
* total\_trainable\_parameter=13684282 (including linear layer for output projection)

****

Figure 1: Loss Graph of Training With Sign2GPT papers

This experiment uses different scheduler settings. Parameters:

* ff\_size=1024
* layer=3
* dropout=0.1
* num\_heads=8
* hidden\_size=300
* total\_trainable\_parameter=3773174



Figure 2: Training with custom parameters

This experiment does not use a scheduler, it only uses constant learning rate(0.00004) and Adam optimizer. Parameters:

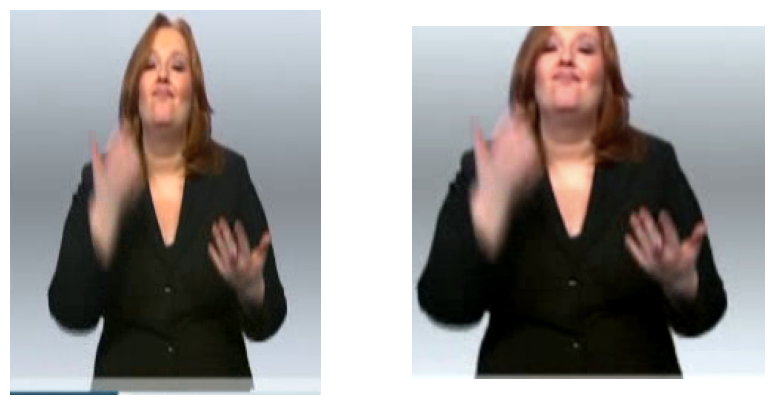
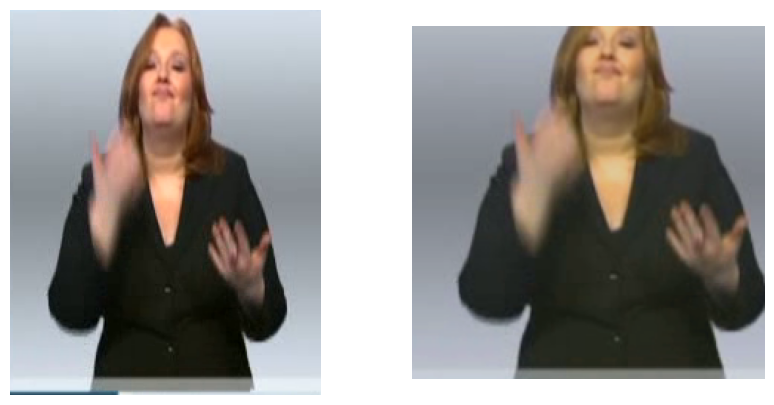
* ff\_size=2048
* layer=3
* dropout=0.1
* num\_heads=6
* hidden\_size=300
* total\_trainable\_parameter=5619446

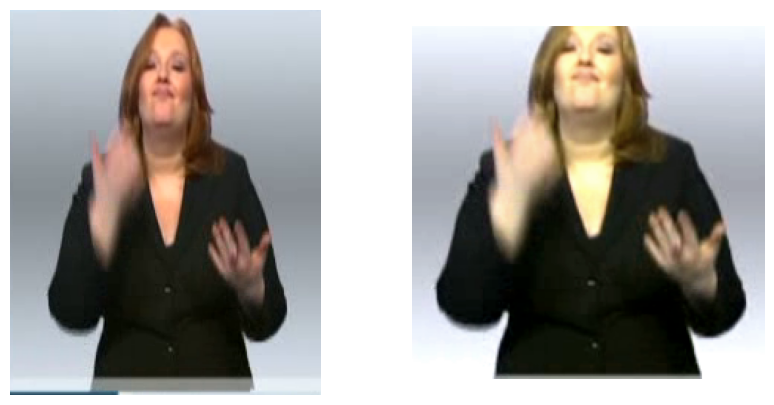


Figure 3: Training with custom parameters

**3-Augmentation Details**

Previously we were creating videos with different frame rates due to our wrong sampling method. Now, we are omitting every 2 frames like the reference paper but it didn’t cause a significant difference in the results. In the augmented train dataset, random color jitter and random cropping have been applied to each frame. Each frame has 3 different variations. Thus, our total train dataset size increased from 7096 to 21288. We didn’t do flipping and rotating. Left is the original image, right is the processed image.





**4-Metrics:**

Here are the best metrics we have obtained before and after data augmentation. This suggests a problem in our augmentation technique but we couldn’t find the problem. Frames look ok. In both models we have the same hyperparemeters: (ff\_size=1024, hidden\_size=300, layer=3, dropout=0.1, num\_heads=6

total\_trainable\_parameter=3773174)

| **Model** | **Precision** | **Recall** | **F1** |
| --- | --- | --- | --- |
| Reference Paper | 0.52 | 0.39 | 0.44 |
| Before Augmentation | 0.2440 | 0.2748 | 0.2585 |
| After Augmentation | 0.2038 | 0.1420 | 0.1674 |

**Loss Graphs:**

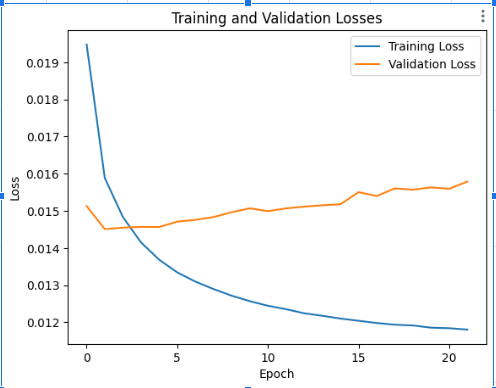
****

Figure 4: Best Model Before Augmentation

(metrics are calculated with an early-stopped model)



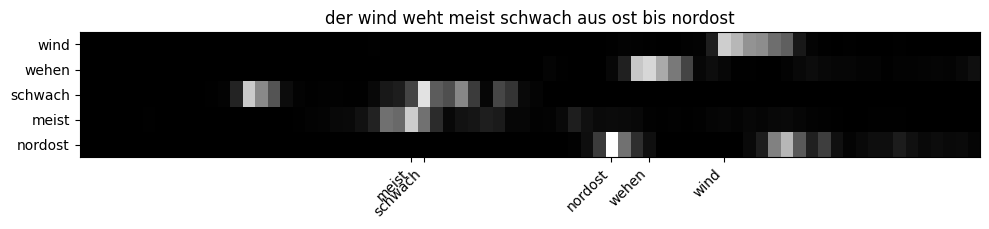
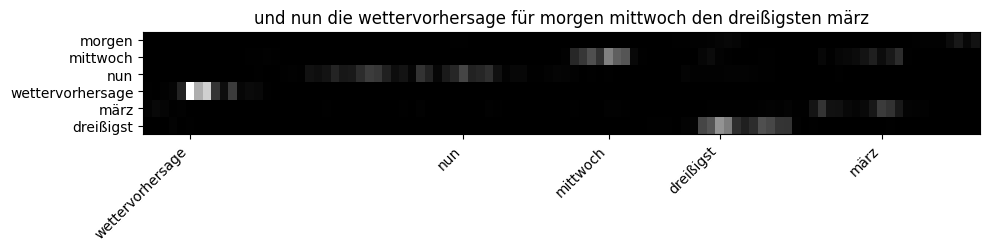
Figure 5: Best Model after data augmentation

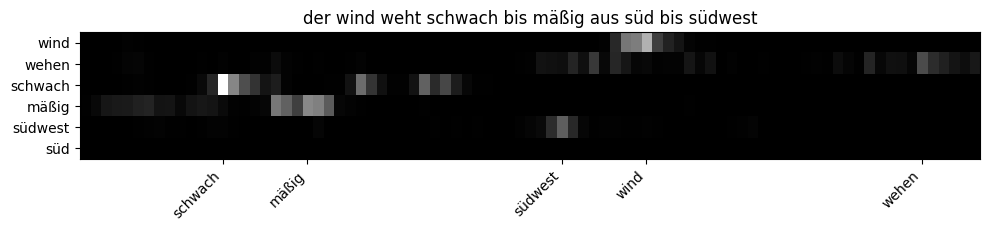
**5-Visualization**

**a-Visualization Using Best Model Before Augmentation:**

ff\_size=1024, hidden\_size=300, layer=3, dropout=0.1, num\_heads=6

total\_trainable\_parameter=3773174





**b-Visualization Using Best Model After Augmentation:**

ff\_size=1024, hidden\_size=300, layer=3, dropout= 0.1, num\_heads= 6

total\_trainable\_parameter=3773174

