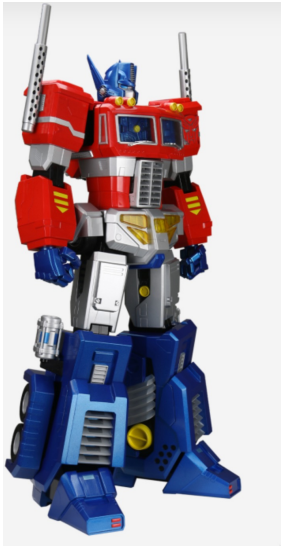
Arrhythmia classification using Transformers

Natalia Sokolova



Problem definition

The **ECG dataset** is composed of two collections of heartbeat signals derived from two famous datasets in heartbeat classification. The signals correspond to electrocardiogram (ECG) shapes of heartbeats for the normal case and the cases affected by different arrhythmias and myocardial infarction.

We should classify the signal to **5 categories**:

0 - Normal beat (N)

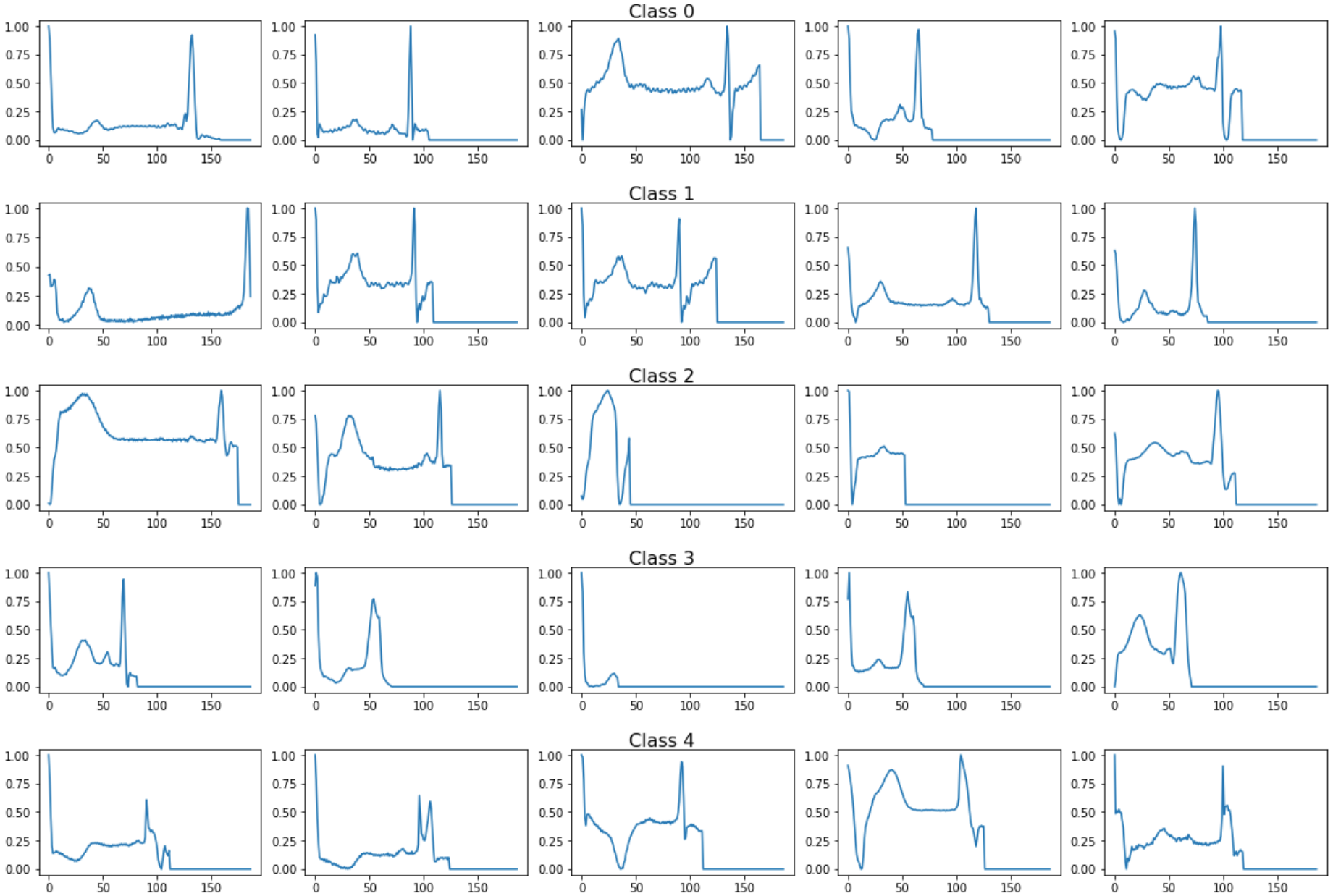
1 - Supraventricular premature beat (S)

2 - Premature ventricular contraction (V)

3 - Fusion of ventricular and normal beat (F)

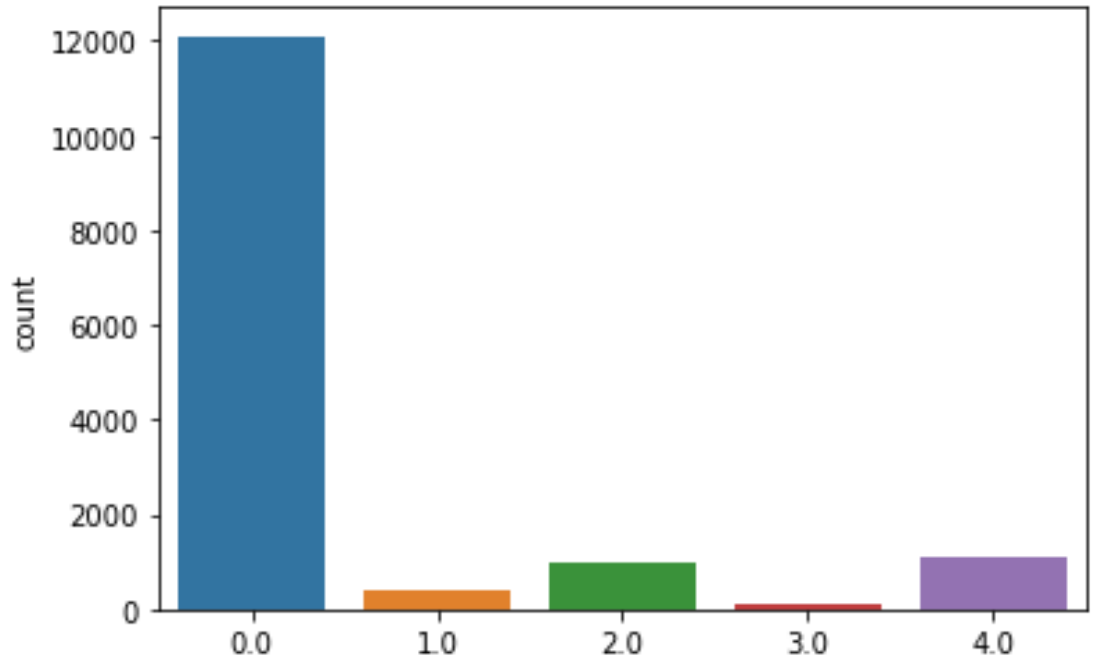
4 - Unclassifiable beat (Q)

Let’s look at ECG signals



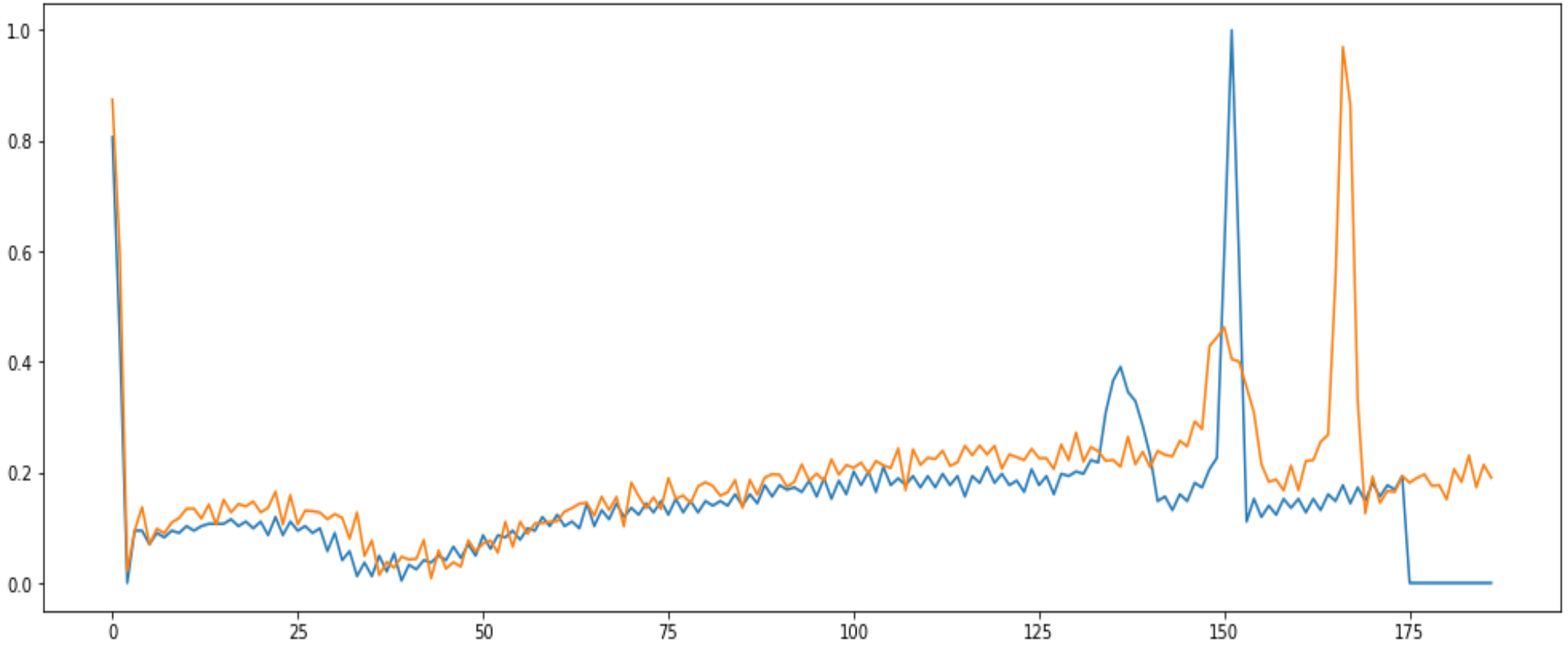
Issue 1: Classes are unbalanced

```
Classes Counts = {0.0: 12102, 1.0: 378, 2.0: 993, 3.0: 103, 4.0: 1091}
Number of classes = 5
```



Solution 1: Augmentation

- Stretching, Amplifying and adding Gaussian Noise with probability = 0.5 each.
- Undersampling for 0 class to 6500 samples per class.
- Oversampling for other classes to 6500 samples per class.



Solution 2: Oversampling using SMOTE

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

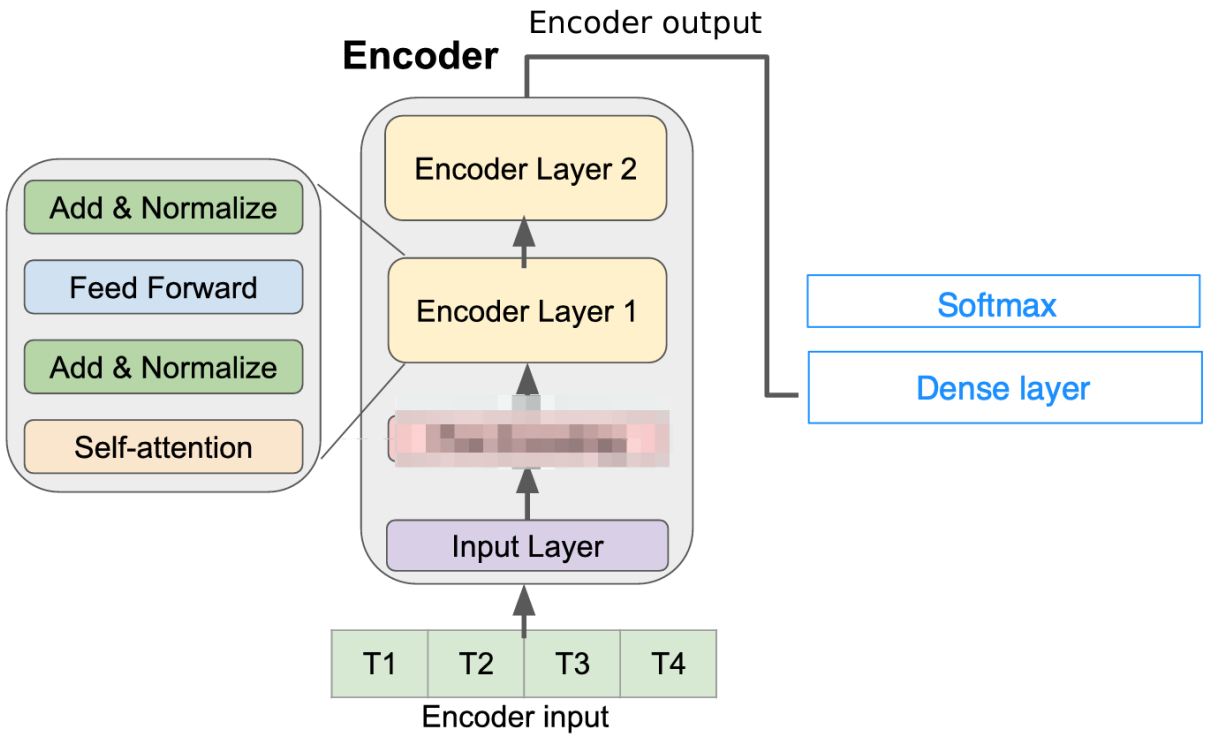
Applied Oversampling for Minority classes: 12102 samples per class

Takeaways: SMOTE augmentation gives better results

Transformer model

Modifications:

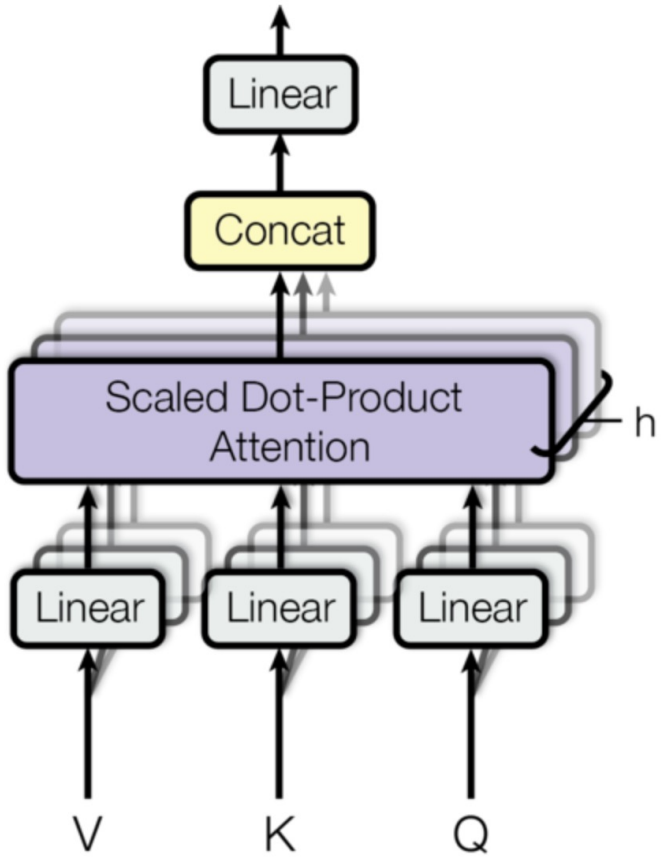
- Decoder block removed
- No Positional encoding
- Feed Forward: 2 Dense layers



Multi-head attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

```python
#### ATTENTION
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = scores.softmax(dim=-1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```



Results for Transformer vs CNN so far

F1 score for Test dataset (private part):

- Transformer: 0.837
- CNN: 0.895

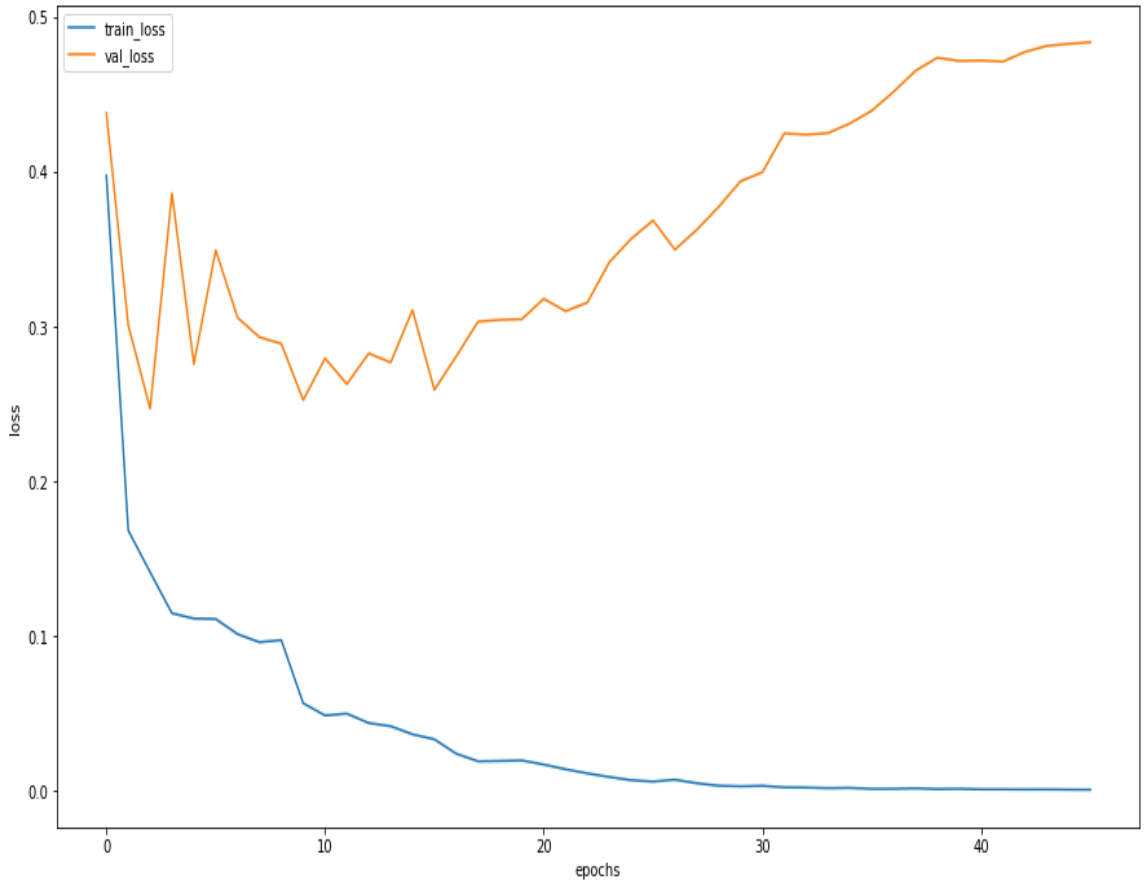
Applied techniques:

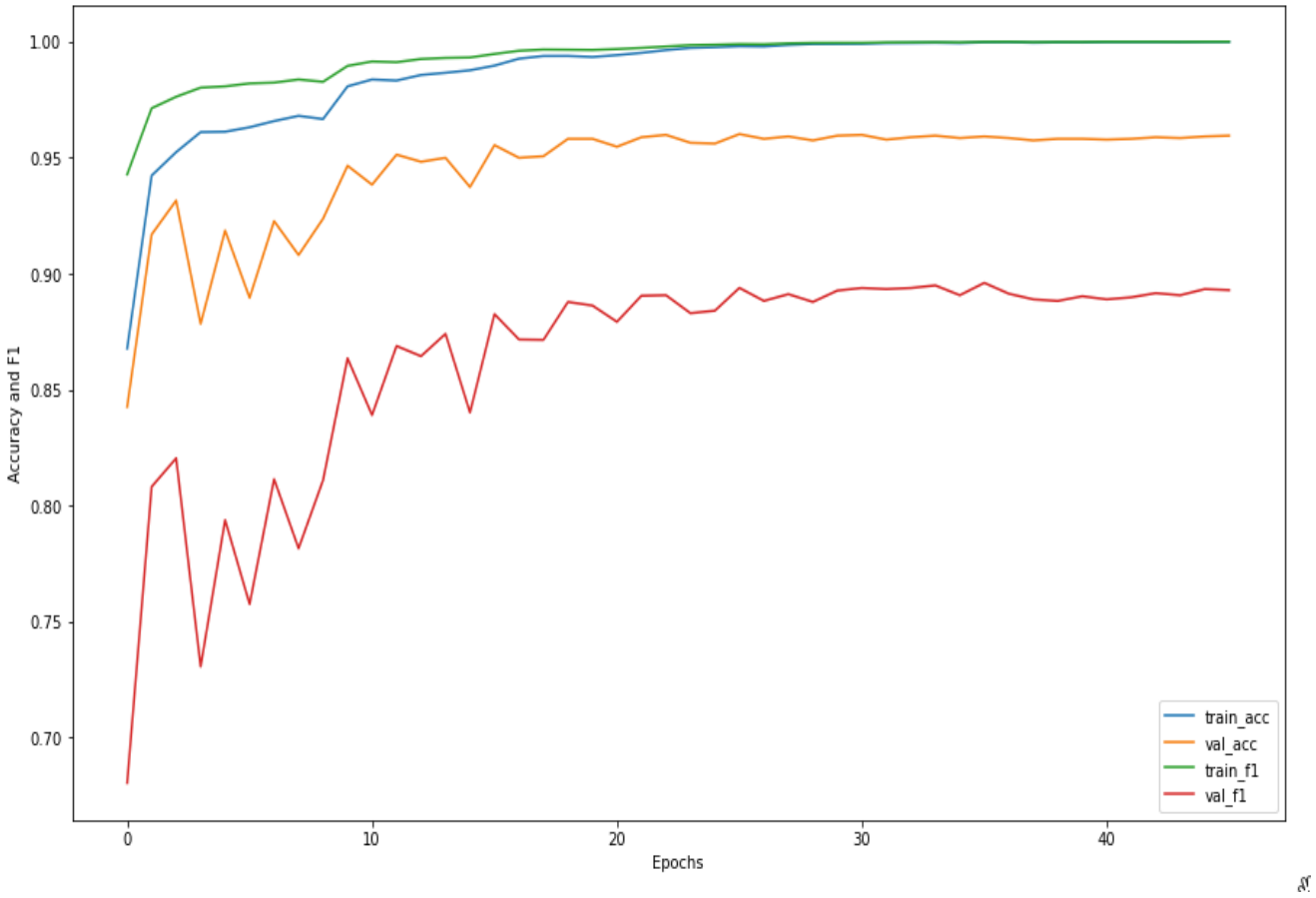
- Layers initialization using Xavier initialization
- Adjusting Dropout parameters
- Learning rate scheduler: Reduce LR on Plateau
- Early stopping
- Adjusting Number of Encoders
- Adjusting Number of Attention Heads
- Toying with Feed Forward dimensions

Transformer model parameters:

```
BATCH_SIZE = 128
DROPOUT = 0.3
LR = 5e-03
N_EPOCHS = 100
HUM_HEADS = 5
NUM_ENCODERS = 5 #were 6 before
DIM_FF = 128

model_t = Transformer(INPUT_SIZE, N_classes, HUM_HEADS, NUM_ENCODERS, d_ff = DIM_FF, dropout = DROPOUT)
init_parameters(model_t)

model_t = model_t.to(DEVICE)

optimizer = Adam(model_t.parameters(), lr=LR)
criterion = torch.nn.CrossEntropyLoss()
scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.5, patience=5, threshold=1e-05, verbose = 1)
early_stopping = EarlyStopping(patience = 10, verbose = 1, mode = 'max')
```

Plotting the training process





What’s next?

Fight overfitting by:

- Simplifying Transformer model
- Using full dataset
- Signal preprocessing (STFT)

Libraries and services

References

- Data Transformer Models for Time series forecasting: https://arxiv.org/abs/2001.08317
- Article: http://nlp.seas.harvard.edu/annotated-transformer/
- Kaggle: https://www.kaggle.com/code/cyrillecervantes/arrhythmia-detection-using-transformer/notebook
- My Notebook: https://www.kaggle.com/code/megazotya/ecg-transformer/notebook

- You can reach me via Telegram @megazotya, Email , Linked In

Thank you