Surgical Data Science - project report

Mitchell Butovsky and Tom Bekor

In this project we dealt with the task of gesture recognition using both kinematic and visual data from the APAS dataset.

1. The dataset

The dataset is a collection of both kinematic data and visual data collected during medical procedure simulations performed by multiple participants. Each participant was recorded performing 4 simulations. In each such simulation, the participant was recorded from the top and from the side using a 30 FPS, 640x480 camera. In addition, kinematic data was collected using 6 electromagnetic sensors the participants wore, each measuring the absolute location, the euler angles and the rotation matrix with respect to a calibrated reference point. Finally, the labels for the tools and gestures at each timestep were supplied. The possible tools are: “no tool”(T0), “needle driver”(T1), “forceps”(T2) and “scissors”(T3). The possible gestures are “no gesture”, “needle passing”, “pull the suture”, “instrument tie”, “lay the knot”, “cut the suture”. All the data/labels were collected/sampled to a rate of 30Hz.

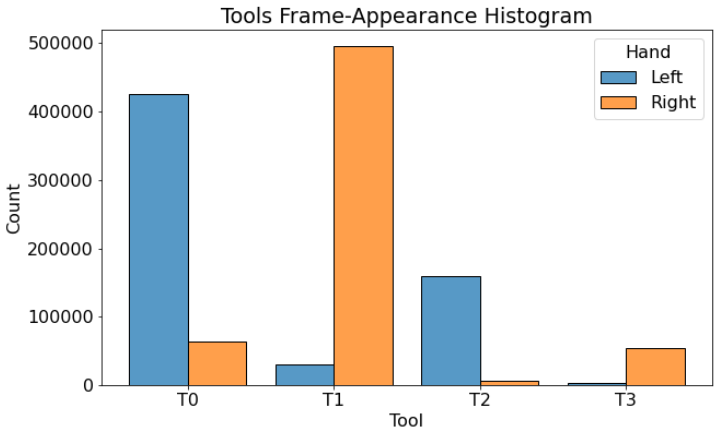

Figure 1: Tool distribution in the dataset

As can be seen, there is a great imbalance in the tool distribution, which we discuss later in the “Training” section.

2. The model

As a baseline model for this project, we were given a simple recurrent neural network which uses the kinematic features as inputs.

We introduce improvements to this model, as well build a model that leverages the visual data.

2.a. Our visual model (2InToolOut): The initial idea we had for using the visual data was to extract feature maps from the side and top videos, frame by frame, using a convolutional neural network architecture (CNN) and then add them as features to the RNN model. The problem was however, that in this project we were supplied with a very small GPU that had only 8GB memory. This problem was so severe that we didn't manage to train such a model even when we down-sized each frame very heavily. Hence, we came to the conclusion that we have to build a model which considers every frame separately. However, in this case we came to a realization that it is substantially easier to predict the tools in such a setting (in which the model "sees" just a frame) rather than predicting the gestures directly. Following this reasoning, we built a model which tries to predict the tools in both hands using the visual data. We figured out that a good and efficient way to do it is to build a model which receives two inputs - a frame from the side and from the top of the same time step in the experiment and uses the learned representations to generate predictions for the tool both in the right and the left hand. The model would extract features from both of the frames using a convolutional neural network and then concatenate the features and generate both predictions using corresponding Linear layers for each of the missions. In order to enforce the model to use the features from both of the frames we have added an additional dropout layer before the final linear layers.

Figure 2: 2InToolsOut model inputs/outputs illustration

In order to achieve a good performance, we decided to use a pre-trained model from PyTorch's pre-trained CNN models. As we have memory constraints we chose EfficientNet-B0 which was well suited for exactly this setting. EfficientNet architecture is based on MB inverted blocks (which are some sort of inverted residual blocks).

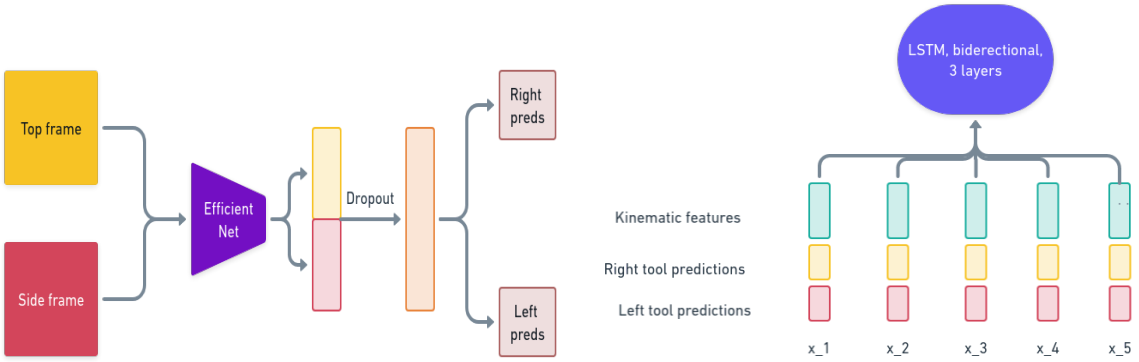

Figure 3: Our model architectures

We used the images in full resolution (640x480x3). The learned representation is of size 1280 (for each frame).

2.b. Improving the RNN model

In order to improve the RNN we introduced some changes. First, we changed the normalization method to standard normalization. Then, we have added an over-segmentation loss term (as in the MS-TCN architecture):

$$\mathcal{L}_{T-MSE} = \frac{1}{TC}\sum_{t,c}\tilde{\Delta}_{t,c}^2, \tag{8}$$

$$\tilde{\Delta}_{t,c} = \begin{cases} \Delta_{t,c} & : \Delta_{t,c} \leq \tau \\ \tau & : otherwise \end{cases}, \tag{9}$$

$$\Delta_{t,c} = |log\ y_{t,c} - log\ y_{t-1,c}|, \tag{10}$$

$$\mathcal{L}_s = \mathcal{L}_{cls} + \lambda\mathcal{L}_{T-MSE}, \tag{12}$$

Finally, we have added the “reduce LR on plateau” learning rate scheduler to the architecture. To conclude, our sequential model is an LSTM architecture, which takes as input sequences of the kinematic features concatenated with the tool predictions generated from the aforementioned model.

2.c. Training: The models were trained and evaluated in a 5-fold manner. The data was under-sampled to 5Hz using (sample_rate=6).

In order to deal with the tool heavy imbalance we performed an under sampling method in which we under sampled according to the minority class with respect to both hands and then united the resulting subsets.

The tool classifier was trained with dropout = 0.3, batches of size 4 using cross entropy loss (the sum of CE for both hands), and Adam optimizer with the default parameters and no scheduler for one epoch.

The sequential model (bidirectional LSTM, 3 layers, hidden dim = 64) was trained with the aforementioned over-segmentation loss with Adam optimizer (with learning rate = 0.003), dropout=0.4, reduce LR on plateau, with 0.5 factor and 5 patience for 40 epochs.

3. Results:

In this section we provide a comparison between the models we tested.

In all of the models below we used standard norm, scheduler and OS loss.

Model	Accuracy	F1-macro	F1@50	F1@25	F1@10	Edit
Baseline	79.14	74.81	64.85	80.09	84.06	80.67
Tools Predictions only	69.60	64.23	39.65	55.47	64.89	54.57
Kin + Velocity + Acceleration	82.79	78.60	71.21	85.64	89.70	87.80
Kin + Velocity + Acceleration + Frames Representations	78.96	74.43	68.62	80.22	83.22	78.89
Kin + Tools Predictions	**84.01**	**80.29**	**73.42**	**87.73**	**91.25**	**89.06**

The frame representations are the concatenated learned representations of both the top and side frames.