Surgical Gesture Recognition Using Multi-Encoder Based Architecture

Eyal Finkelshtein and **Ido Terner**
Industrial Engineering and Management Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel
{eyal.f, ido.terner}@campus.technion.ac.il

1 Introduction

Gesture recognition is a type of perceptual computing user interface that allows computers to capture and interpret human gestures as commands. The general definition of gesture recognition is the ability of a computer to understand gestures and execute commands based on those gestures. Automatically recognizing surgical gestures is a crucial step towards a thorough understanding of the surgical skill. Possible areas of application include automatic skill assessment, intra-operative monitoring of critical surgical steps, and semi-automation of surgical tasks.

Solutions that rely only on raw video and do not require additional sensor hardware are especially attractive as they can be implemented at a low cost in many scenarios. However, surgical gesture recognition based only on video is a challenging problem that requires effective means to extract both visual and temporal information from the video. In this project, we propose to use a multi encoder-single decoder architecture based on LSTM to solve the gesture recognition task, using combined kinematic and video input data. We evaluate our work on the APAS dataset.

The model proposed in this paper achieves promising results, compared to the benchmark model given to us.

2 Dataset

We trained our model on the APAS dataset. The dataset includes kinematic data (X, Y and Z positions and 3 Euler angles for 6 sensors) and RGB video frames at a resolution of 640X480. The frames are labeled by one of six possible gestures (needle passing, pull the suture, instrument tie, lay the knot, cut the suture, and no gesture), and also by the tools held in each hand (needle driver, forceps, scissors and no tool in hand). The lists of gestures and tools are given in Table 1.

The kinematic data was used as given - feature vectors of size 36 per frame. The frames were embedded to feature vectors of size 512, using pre-trained ResNet-34 architecture. For the embedding of the frames we used img2vec[1], which is a library that uses pre-trained CNN models such as ResNet, VGG-16, AlexNet, etc., and uses the output of the average pooling layer as a feature vector for the given image. The frames of each video were saved as a single tensor of size (S, F), where S is the number of frames in the video and F=512 is the number of features extracted from each frame. As part of the

[1]https://github.com/christiansafka/img2vec

features extraction, the frames where resized to 224X224, the image values were scaled to the range $[0, 1]$, and then the image channels were normalized with expectations vector $\mu = (0.485, 0.456, 0.406)$ and standard deviations vector $\sigma = (0.229, 0.224, 0.225)$.

Tag	Name
G0	No Gesture
G1	Needle Passing
G2	Pull The Suture
G3	Instrument Tie
G4	Lay The Knot
G5	Cut the suture

(a) Gesture vocabulary

Tag	Name
T0	No Tool In Hand
T1	Needle Driver
T2	Forceps
T3	Scissors

(b) Tool usage vocabulary

Table 1: Cocabularies of the gestures (a) and tool usage (b).

3 Architecture

The data processed as described in Section 2, is sent to our proposed model, which combines several types of input and includes several LSTM units to encode these inputs and exploit the time dependencies. This model is based on the architecture presented in [1]. In this paper, the authors used three input types (video, kinematics and system events), as well as LSTM with attention, and StiseNet [2] for fine-grained surgical states. Since we don't have time granularity hierarchy, we decided to use a simpler model, and leverage the advantages of combining several data types.

To exploit the dependencies that present in the data between different times, as the data is taken from real life surgical procedures, we implemented an LSTM encoder to embed the correlation among the kinematic feature vectors x^{kin} of size 36 into a latent representation $h_t^{kin} = LSTM(h_{t-1}^{kin}, x_t^{kin})$. The latent representation at time t therefore includes information from prior time steps. Similarly, after feature extraction from the video frames, using the output of the average pooling layer of a pre-trained ResNet-34 architecture, the video feature vectors x^{vid} of size 512 were sent to a second LSTM encoder which produced the embedding $h_t^{vid} = LSTM(h_{t-1}^{vid}, x_t^{vid})$. The latent representations, or "features", of the kinematic data and video data, are concatenated at each time t to form a feature vector $H_t \in R^{n_{kin}+n_{vid}}$. The concatenated feature vector is then sent to an LSTM decoder followed by a linear layer, which gives us the classification prediction to different gestures. The full architecture is presented in Figure 1.

4 Results

In this section, we provide a detailed experiment analysis of our model on the APAS dataset. We will begin by shortly describing the metrics we used to evaluate our model, and then describing the training and evaluation process.

4.1 Metrics

In this project, we evaluated the model using two types of metrics - frame-wise and segmentation. To do that, we used six different metrics: accuracy, and F1-macro for the frame-wise evaluation, and Edit, F1@10, F1@25, and F1@50 for the segmentation evaluation.

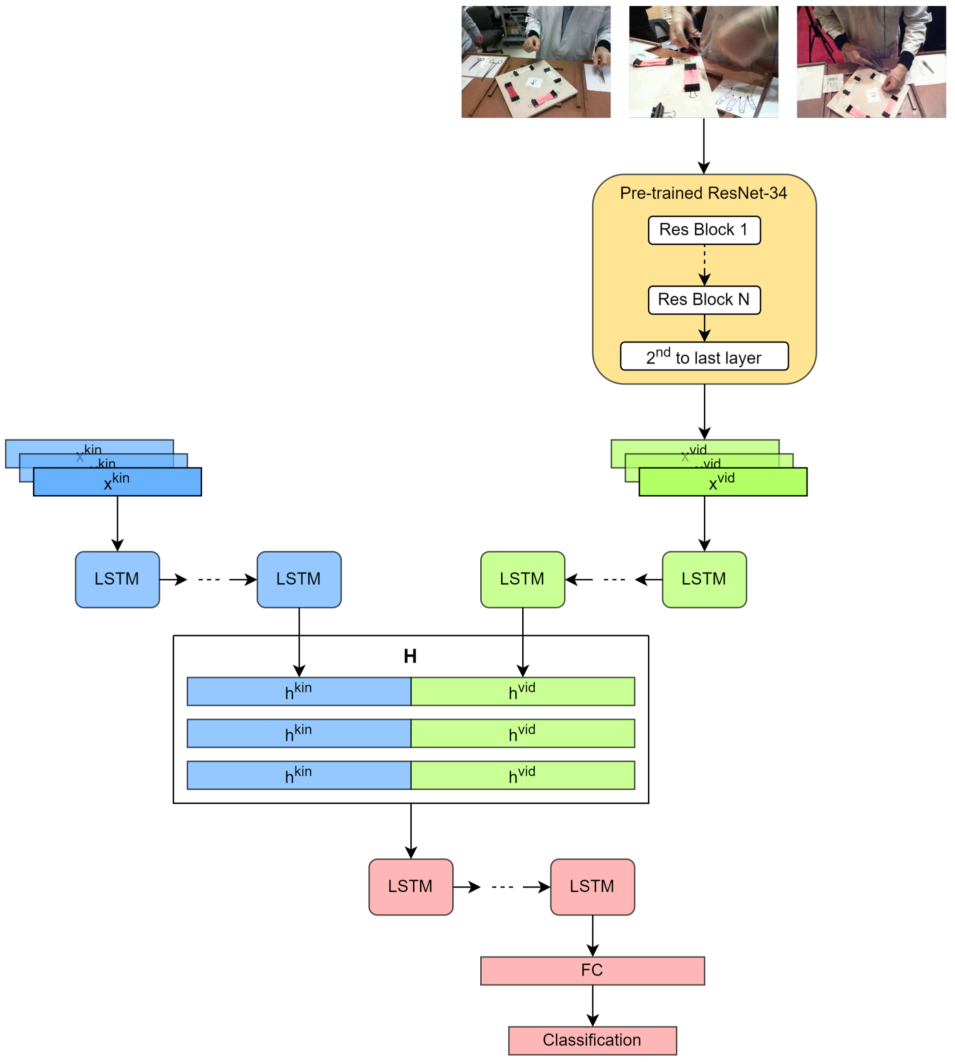


Figure 1: The architecture of the proposed model. The kinematic features are used as is, and passed through an LSTM encoder (in blue). The surgical video frames are passed through a pre-trained ResNet-34 model to extract latent space features (in yellow), which then sent to a second LSTM encoder (in green). The kinematic and video features are concatenated and passed through an LSTM decoder, followed by a fully connected layer, which produces the gesture classification (in pink).

4.2 Experiment Evaluation and Comparison

As a first step, to determine whether there is an advantage in using video inputs from two angles (side and top), we performed an evaluation for models using each one of the video angles separately, and both of them together on a single split. The comparison was made using the mean over the metrics accuracy, F1-macro, edit, F1@10, F1@25 and F1@50, to consider all of them together. As can be seen in Figure 2, there are very small differences between the combinations of input videos. We chose to proceed only with the video input from the side angle, as it has achieved slightly better results, and a shorter training and evaluation time, compared to using both video angles.

The model was trained on 5 different splits of the data, using 5-fold cross validation, and the results of the metrics described earlier were recorded. For each fold, the model was trained for 60 epochs from scratch with a batch size of 4. For optimization, we used Adam optimizer with a learning rate of 1.47×10^{-3}. The training loss and accuracy throughout the training process are shown in Figure 3,

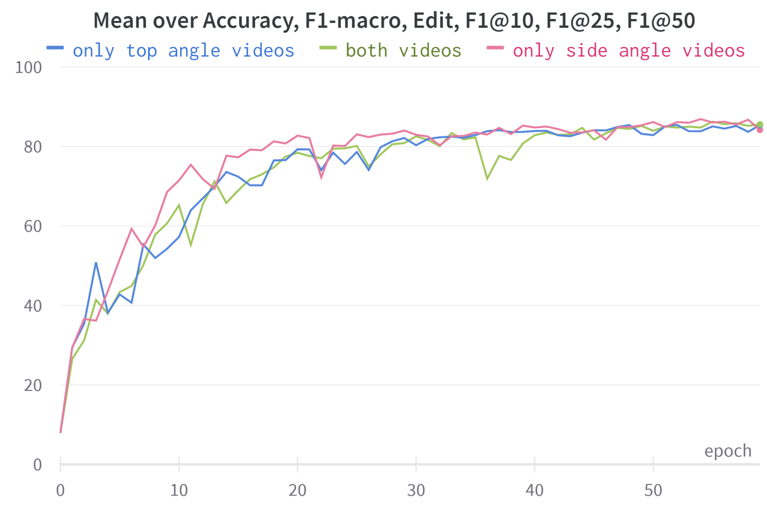
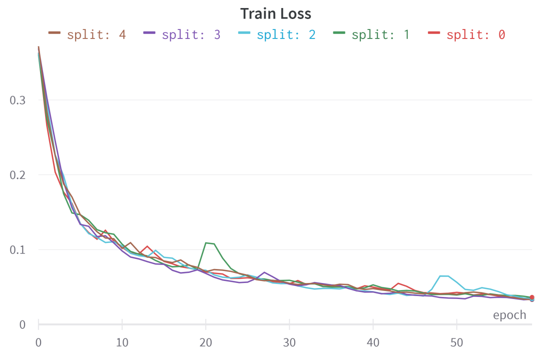


Figure 2: The mean over all metrics used in this paper to evaluate the models, for inputs of a top angle video, a side and video, and videos from both angles.

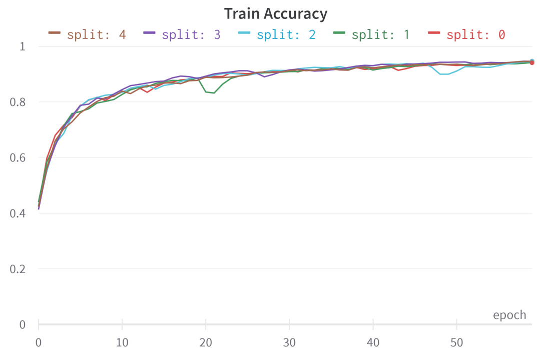
and the evaluation metrics, which where collected after each training epoch are shown in Figure 4. Other hyper-parameters used in this paper:

- Kinematic LSTM encoder: number of layers - 4, hidden dimension - 123.
- Video LSTM encoder: number of layers - 1, hidden dimension - 215.
- LSTM Decoder: number of layers - 1, hidden dimension - 153.

The hyper-parameters were determined by a sweep run with Bayesian search strategy, optimizing over the mean of all the metrics used in this paper.



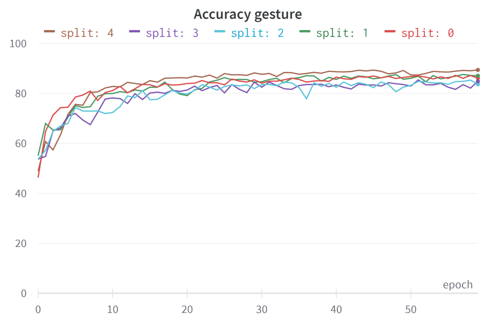
(a)



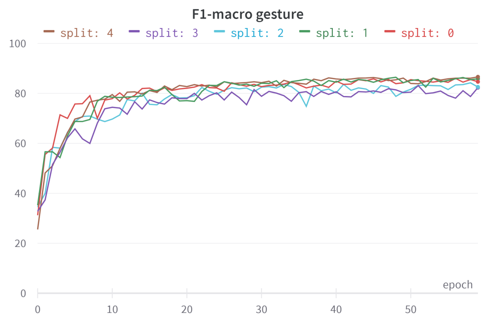
(b)

Figure 3: Train loss (a) and train accuracy (b).

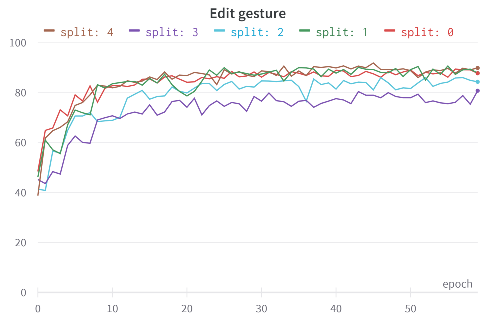
The evaluation results of the model for the best scoring epoch of each split are demonstrated in Table 2, and the evaluation results of the benchmark model given to us in this project are presented in Table 3. Our model outperformed the benchmark model in every metric, by 3.3-16.5% (averaged over all splits), and achieved an average accuracy of 86.771%.



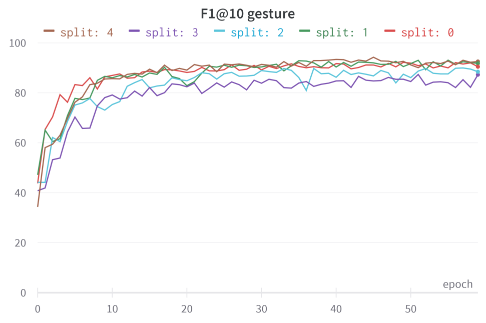
(a)



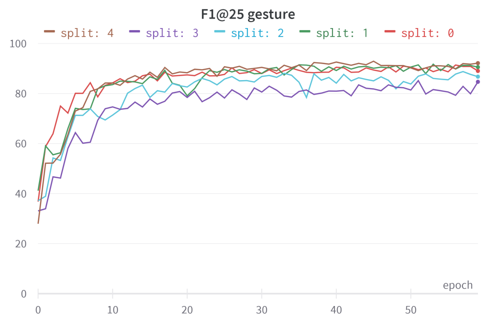
(b)



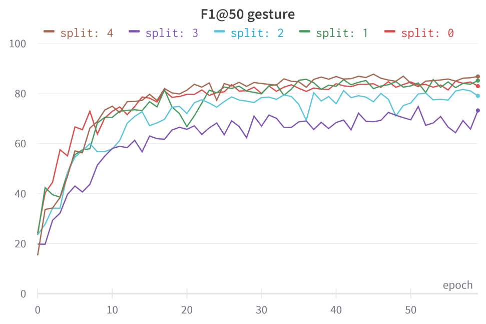
(c)



(d)



(e)



(f)

Figure 4: All metrics used to evaluate the model performance: accuracy (a), F1-macro (b), Edit (c), F1@10 (d), F1@25 (e), F1@50 (f).

5 Discussion

This work proposes an architecture for solving the gesture recognition task based on the encoder-decoder architecture and utilizes both kinematic and video data. We strongly believe that utilizing both types of data is the key to achieving great results on this task. In addition, we provided an extensive experimental analysis of our work on a surgical gesture recognition dataset and showed promising results. Given more time, there is no doubt the hyper-parameters can be further optimized to achieve even better results.

6 Failed Experiments

In this project, we tried several architectures and ideas for solving the gesture recognition task which didn't work.

Split	**Accuracy**	**F1-macro**	**Edit**	**F1@10**	**F1@25**	**F1@50**
1	87.227	86.014	89.448	92.131	91.402	84.940
2	87.089	85.690	89.898	92.743	91.355	85.699
3	84.810	83.535	86.019	89.865	87.768	80.979
4	85.398	83.130	79.376	87.382	85.158	74.819
5	89.332	86.340	91.869	94.278	92.916	87.738
Average	86.771	84.942	87.322	91.280	89.720	82.835

Table 2: The performance of the suggested model for each split and their average. All metrics are given in %.

Split	**Accuracy**	**F1-macro**	**Edit**	**F1@10**	**F1@25**	**F1@50**
1	79.249	75.618	87.008	87.461	83.49	65.517
2	83.828	80.04	85.022	88.282	86.6	74.199
3	78.332	74.869	81.28	84.365	81.015	68.426
4	78.264	72.915	80.161	83.065	75.991	57.449
5	81.465	75.989	86.651	87.155	82.448	66.308
Average	80.227	75.886	84.024	86.065	81.908	66.379

Table 3: The performance of the benchmark model for each split and their average. All metrics are given in %.

Our first try was using only the kinematic data with a 1D convolution after the LSTM encoder. We thought that using the convolution layers could help us find a correlation of areas in the feature vectors extracted from the LSTM encoder, which could help the model learn and generalize better, but unfortunately, it didn't quite work.

Another idea combined the kinematic data and the videos. We tried to use multiple encoders, one for the kinematic data, and one for the raw frames. The process of the kinematic data was the same as presented in this paper. For the raw frames, we first used a simple convolutional neural network to extract a feature vector for each frame, which was later used as input for the LSTM encoder. Both feature vectors, after the LSTM encoders, were concatenated and sent to a linear decoder. This idea didn't work due to lack of memory in the GPU, even though we used a batch size of 1, and resized the images to a resolution of 96X96 with one channel (grayscale). This probably happened because of the additional gradients of the convolutional network that had to be saved in the memory as well. This brought as to use a pre-trained model.

We also tried to use a transformer based encoder, but got pretty bad results, probably due to the lack of a sufficient amount of data.

References

[1] Yidan Qin, Max Allan, Joel W. Burdick, and Mahdi Azizian. Autonomous hierarchical surgical state estimation during robot-assisted surgery through deep neural networks. *IEEE Robotics and Automation Letters*, 6(4):6220–6227, 2021.

[2] Yidan Qin, Max Allan, Yisong Yue, Joel W. Burdick, and Mahdi Azizian. Learning invariant representation of tasks for robust surgical state estimation. *IEEE Robotics and Automation Letters*, 6(2):3208–3215, 2021.