Classification of Lacroscopic Surgical Phases

Sam Fieldman
M.S Computer Science
Columbia University
SF3043@columbia.edu

Trevor Gordon
M.S Electrical Engineering
Columbia University
tjg2148@columbia.edu

Abstract—**Automatic surgical phase recognition is a challenging task with the potential to to increase patient safety, reduce surgical errors and optimize the communication in the operating room. Surgical phase recognition can provide physicians with early warnings in cases of deviations and anomalies, as well as providing crucial information for archiving, educational and post-operative patient-monitoring purposes. We first attempt to address this issue by posing the problem as an image recognition task, and find that remarkably high performance is possible even without utilizing temporal features captured by sequence models such as recurrent neural networks. Our approach focuses on fine tuning pre-trained convolutional neural networks (CNNs) to perform surgical phase classification on a set of lacroscopic surgery videos. To further improve model performance and generalization, we evaluate the use of data-augmentation, balanced classes sampling, alternate classification loss-functions, and label-smoothing. Finally, to incorporate the temporal nature of the problem, we follow the approach of Czempiel et al. (2020) by utilizing Temporal Convolutional Network (Lea et al. (2016)) in conjunction with a pre-trained ResNet50 as a visual feature extractor. With the approaches outlined above we were able to achieve a test loss of 79.5%.**

I. Introduction

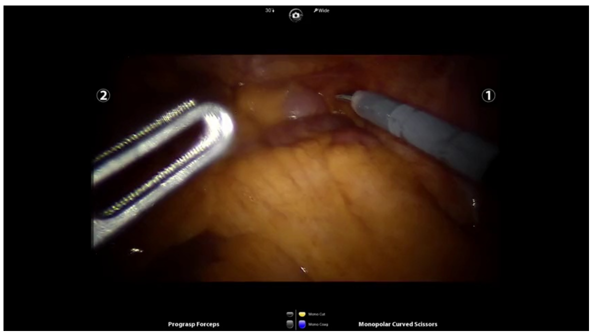
The goal of this project is to develop a surgical phase recognition model which is both fast and accurate. We propose two primary methods for approaching this task. The first of which is similar to that taken in Primus et al. (2016), where the phase recognition problem is treated simply as an image classification task. This setting does not utilize temporal features, but makes up for this loss by being much more memory and computation efficient compared to approaches such as Recurrent Neural Networks or Hidden Markov Models. Our second approach was to incorporate the temporal features by using Temporal Convolutional Networks (Lea et al. (2016)). The primary benefit of using a TCN instead of other sequence models such as RNNs or Transformers, is that they fast and parallel computation along with a low memory footprint during training. This makes them a good candidate when training an RNN model may be prohibitively expensive or prevent them from being feasible in an online OR setting.

Surgery Phase	Data Frequency
Adhesiolysis	0.14%
Blurry	0.042%
Catheter Insertion	1.35%
Mesh Placement	12.91%
Mesh Positioning	0.54%
Out of Body	0.71%
Peritoneal Closure	26.06%
Peritoneal Scoring	5.04%
Positioning Suture	6.78%
Preperioneal Dissection	13.34%
Primary Hernia Repair	0.53%
Reduction of Hernia	23.96%
Stationary Idle	1.02%
Transitionary Idle	7.53%

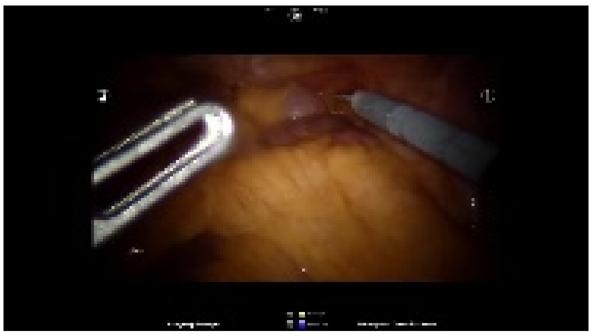
Table I: Class Occurrence Rates

The primary challenge faced when developing our model was dealing with the large amount of available training data. While only 70 videos were made available to us, each video was on average 51 minutes long, with a frame rate of 1 frame per second. If we consider each frame as a single training sample, the we have a total of approximately 214,200 images to train on. Further, each frame has a resolution of 420X796, and therefore contains approximately twice the average number of pixels of an image from ImageNet. This imposes a large computational cost when trying to train larger networks, and as mentioned above, would make training models such as LSTMs or GRU very time consuming and computationally expensive.

A further challenge was introduced by the distribution of the class labels. As shown in table I, some surgery phases only account for as low 0.05% of the total labeled data, while others account for as much as 25% of the data. Because of this imbalance, maximizing a loss function such as Categorical Cross Entropy without accounting for the class imbalances can potentially lead to poor performance on these



(a) Full Resolution



(b) Low Resolution

relatively unobserved classes. Given that the true end goal of this model is to accurately classify all classes equally well, either a sampling strategy of modified loss function will be necessary to properly address this.

II. Architecture

For the classification task, we utilize a pre-trained model, and then either fine-tune the final layer as well as a new classification layer for our specific classes, or simply leave all the model weights frozen and simply replace the fully connected layers. Using a pre-trained ResNet18 model from pyTorch Paszke et al. (2019) as a feature extractor, and training a single feed-forward layer on its features, we are able to achieve a test accuracy of approximately 75% while also achieving almost perfect accuracy on the training data. Interestingly, switching to a deeper model such as ResNet50, did not lead to any noticeable change in the test or training accuracy, not did fine-tuning the final convolutional blocks of either model. This is a very surprising result given that the features and images that these models were trained on are likely vastly different from the surgical data set. A potentially interesting avenue of further work would be to perform unsupervised fine-tuning using by treating the ResNet as an encoder in an AutoEncoder set up.

Building on using a pretrained model as a feature extractor, we further utilize if as the backbone of a Temporal Convolutional Network (TCN), as done in the paper Czempiel et al. (2020). A similar approach is also taken in works such as Namazi et al. (2018) and Jin et al. (2021), where convolutional networks are used as feature extractors, and RNNs serve to extract the temporal information. The primary difference between these two approaches is in the computational requirements for training and evaluating these models. As discussed in Bai et al. (2018), while RNNs are often seen as the default approach for sequence modelling, TCNs are often able to achieve comparable performance, while also exploiting parallelism for faster training and inference. As mentioned above, this makes it more amenable when computation is at a premium, or when the model is to be used in a online setting. While we began to explore this approach, we found that our initial approach performed significantly worse on the test set when compared to the simple fine-tuned ResNet approach (50% test accuracy compared to 77% for ResNet18). However, given the positive results of Czempiel et al. (2020), and a manageable training time of approximately 12-15 minutes per epoch, we believe that this would be a worthwhile approach to explore further.

III. Training and Evaluation

A. *Pre-processing*

To address the bottle neck presented by the size and quantity of the training data, we first preprocess the mp4 files into individual jpg files identified by their frame number. However, at full resolution, training on the full data set for 1 epoch would take between half an hour to an hour when fine-tuning ResNet models. To address this, we down-sample the images to a resolution of 120X200, decreasing the pixel density by more than a magnitude of 10. A snapshot of a video at each resolution is provided in figure **??** showing the qualitative difference between the images. Not only were we still able to achieve competitive performance with this lower resolution image, we were also able to see a similar order of magnitude speed up in model training, with epochs for fine tuning ResNet18 taking only 5-6 minutes.

Along with this, we include a data-augmentation pipeline to help increase generalization to the validation and test sets. The data-augmentation pipeline is composed of a Random Horizontal flip, A Random Color Jitter in the form of changes to image brightness, contrast, and saturation, and a Gaussian Blur with a kernel size of 3 and a standard deviation value randomly selected in the range of 0.1 to 2. Each of these augmentations occur with independent probability 0.25. Finally, a mean-zero Gaussian noise is applied to the image.

B. Data Imbalance

Due to the issue of class imbalances note above, we utilize weighted random sampling to over sample the data from the less frequent classes. However, rather than using a weighting strategy that sets all of the sampling frequencies to be equal, we experiment with a linear weighting between the underlying frequencies and the ideal balanced frequencies. The formula for the class frequencies used in sampling mini-batches is then

$$\text{adj. freq.}_{\text{class } i} = w * \frac{1}{14} + (1 - w) * \text{obs. freq.}_{\text{class } i}.$$

Since the least frequent class occurs almost 500x less frequently than the most frequent class, an equal sampling strategy would lead us to very frequently retraining on the same handful of images, which could both cause us to over fit to that subset of the data, while also performing poorly on the remaining data samples. We compare the validation statistics of the ResNets fine-tuned using the true underlying class frequencies, the 'ideal' even frequencies, and the average value of the two in **??**.

One other approach considered is to directly maximize the criteria of interest, namely the Macro-F1 score. The primary problem in doing this is that precision, recall, and F1-Scores are all non-differential functions. However, we can use a soft Macro F1 score as the training objective, which is computed identically to the regular Macro F1, but removes all calls to argmax. Note that since most optimization packages try to minimize the metric of interest, so we must instead optimize the value $(1 - \text{Soft-F1})$. Again, while we began working on this avenue of investigation, we were unable to successfully utilize this metric as our loss function, but believe that it is worth pursuing further.

C. Post-processing

Predictions are carried out for a single image frame at a time. A drawback to this is that adjacent frame classifications are completely independent. We know that adjacent frames have a high likelihood to share the same class. To improve our predictions, we employ a rolling window like Stauder et al. (2016) most common class to smooth our output predictions. In our final predictions we use a sliding window of 9 which was chosen by observing our training data set. Below we compare the predicted output classes before and after smoothing. In can be seen from the training example that it is unlikely for single frames to be different than their neighbours. In our original predicted output it can be seen that some frames show scattered predictions from their neighbours. However, after using a sliding window our predictions look much cleaner.

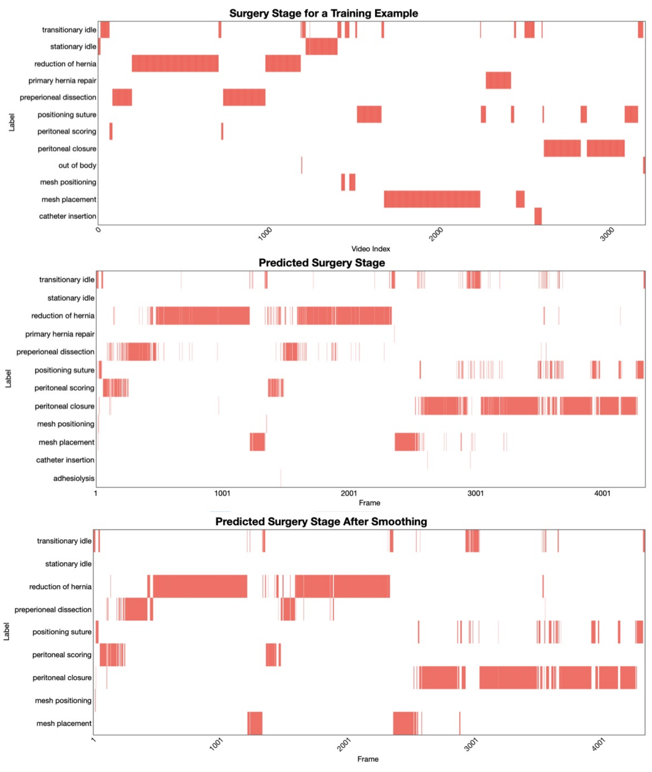


Figure 2: The surgery stages over time. The top graph shows a training example. The middle graph shows predicted labeling before smoothing. The bottom graph shows labeling after smoothing

IV. Discussion

V. Model Specifications

The final model weights are stored https://drive.google.com/file/d/1mW8SFB0NVxLbbKCtC–W5G-F4i1Qla4i/view?usp=sharing . The model is able to predict at a rate of 750 FPS using a Tesla A100 GPU. Our final validation accuracy reaches 95 percent. Our memory model is comparable to the pytorch resnet80 model.

References

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.

Tobias Czempiel, Magdalini Paschali, Matthias Keicher, Walter Simson, Hubertus Feussner, Seong Tae Kim, and Nassir Navab. TeCNO: Surgical phase recognition with multi-stage temporal convolutional networks. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*. 2020.

Yueming Jin, Yonghao Long, Cheng Chen, Zixu Zhao, Qi Dou, and Pheng-Ann Heng. Temporal memory relation network for workflow recognition from surgical video, 2021.

Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. 2016.

Babak Namazi, Ganesh Sankaranarayanan, and Venkat Devarajan. Automatic detection of surgical phases in laparoscopic videos. 07 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

Manfred Primus, Klaus Schoeffmann, and Laszlo Böszörmenyi. Temporal segmentation of laparoscopic videos into surgical phases. pages 1–6, 06 2016. doi: 10.1109/CBMI.2016.7500249.

Ralf Stauder, Daniel Ostler, Michael Kranzfelder, Sebastian Koller, Hubertus Feußner, and Nassir Navab. The tum lapchole dataset for the m2cai 2016 workflow challenge, 2016. URL https://arxiv.org/abs/1610.09278.