Query By Video for Surgical Activities

Felix Yu, Gianluca Camargo Silva Croso
Mentors: Tae Soo Kim, Haider Ali, Gregory Hager, Austin Reiter and Swaroop Vedula

I. Key Points

How can we devise a way to encode a database of surgical phase clips such that given a query phase clip, we can search the database for videos of similar behavior?
53 cataract surgery videos were hand-segmented into 386 phase clips and encoded using a series of neural networks. We then queried 136 surgical phase clips using a nearest neighbor framework and found that we correctly classified 71.1% of these queries for its phase.
Our encoding framework is able to discern between clips of different phases with reasonable accuracy, and forms a base for querying a database for similar surgical behavior.

***Abstract*—Currently, providing surgeons feedback on procedures manually is both time-intensive, and requires expert knowledge. At least partially automating this procedure would be very beneficial to surgeons-in-training. One of the first steps towards using video data to produce feedback is to be able to encode encode spatial and temporal information of a clip. We would expect that such an encoding would be able to at least guarantee that clips of the same phase within the surgery have more similar encodings than those of different phases. Here, we propose a pipeline to create such an encoding, and evaluate its accuracy in surgical phase classification based for cataract surgery clips. The classification is based on nearest neighbors within the encoding feature space. The pipeline consists of a convolutional neural network meant to extract spatial features followed by a recurrent neural network, which also uses tool annotations, to extract temporal information. With this pipeline, we attain a phase classification accuracy of 71.1% in a testing dataset of videos that were not used at any step of the pipeline training or validation.**

II. Introduction

Currently, providing feedback for surgeons manually after an operation is both costly and difficult. This is due to the large time commitment required by those giving feedback, as well as the scarcity of experts with enough experience to have accurate and useful comments. Therefore, one of the goals of analyzing surgical workflows is to automate the process of providing meaningful feedback post-operation to the surgeons, allowing for both training of novice surgeons and improving or diversifying the skill of expert surgeons. However, this goal has constraints. Because the methods developed for delivering post-operational feedback should be executable for many, if not all, hospitals, the data used to provide the commentary needs to be easily obtained. Therefore, instead of using information that is measured by special devices that do not exist in most hospital settings in order to perform surgical workflow analysis, we target our analysis to use videos of the surgery, as well as tool label annotations. In our case, tool label annotations are defined to be information detailing which tools are present in each timestep of the videos.
Using videos presents a variety of challenges. Because there is no standard protocol on how a video of the surgery should be recorded, there is high variability between surgical videos in terms of video quality, the overlay used, the angle in which the surgery is filmed, lighting conditions, etc. This is why we have also decided to use tool label information as well. Because the tools used in specific surgeries stay relatively consistent for different doctors and hospitals, we are able to introduce a factor of consistency that allows us to guide our encodings and account for the variability in the videos.
The question then follows: given this information for an operation, what processing methods should be applied in order to automatically output novel feedback to the surgeon? In order to begin solving this, we would need existing commentary to base our feedback off of. Therefore, a database of these videos must first be obtained, and expert feedback should be generated manually to serve as a ground truth for possible commentary. It is also difficult to provide feedback on the entire surgery in one shot. Because a surgery comprises of multiple phases, segmentation of the video into its respective phases should be done in order to separate out the various different activities the surgeon does within the operation. Feedback can then be produced for each of these phase clips, and makes our problem much more feasible. Next, there needs to be a way to tell if two clips are "similar" to each other in terms of which phase is currently being performed and the method of execution. This is so that we can build a database of commentary that resides within a feature space based on the encoding. Once a new surgery takes place, the video can be segmented. Finally, by taking the resulting query clips and finding where it resides within this feature space, we can look at nearby commentary through a nearest neighbors method and construct novel feedback for the specific phase.
This paper focuses on third sub-problem, namely creating a method of encoding phase clips into a vector in some feature space to allow for nearest neighbor querying. Because our method aims to generate feedback towards specific phases in a surgery, our work is specific to the type of surgery performed. We analyze cataract surgical videos, in which surgeons replace the damaged lens of an eye with an artificial one. We assume ground truth segmentation of the videos are available, as well as information on which tools are

Felix Yu and Gianluca Camargo Silva Croso are Undergraduate Students at the Johns Hopkins University.
T. Kim, H. Ali, G. Hager and S. Vedula are with the Johns Hopkins University.
Manuscript from May 10th, 2018.

currently in the clips at each time step of the clip itself. Fig 8 in the supplementary figures section of the appendix lists all 10 possible phases that may occur during the cataract surgery, and maps an integer between 1 and 10 to these phases. For simplicity sake, we will refer to the phases based on the numbers from here on out. Our method adapts two existing neural network frameworks, SqueezeNet and LSTM networks [1], [2], to capture spatial information found within the videos, as well as temporal information of both the videos and tool information. After the networks encode this information into feature vectors, we use a nearest neighbors method with a Euclidean distance metric to query for similar clips in the database. We show by doing so, we are able to discern which phase each query clip belongs to with reasonable accuracy.

III. Related Work

There has been a significant amount of recent interest in various applications of automated surgical analysis, particularly in the computer vision community. Studies have different specific focuses, from phase classification to tool detection, but they all aim to improve automatic interpretation of surgical data.
Recent work by Lea *et al.* [4] demonstrates the possibility of automatic segmentation and phase recognition in a dataset consisting of endoscopic cholecystectomy videos Convolutional Neural Networks to extract an encoding of spatial and temporal features, and also fusing tool information to improve phase classification.
The 2017 Cataract Grand challenge [12] resulted in several methods for automatic tool annotation for cataract surgery, and work by Kim *et al.* [11] indicated that such annotations can also be obtained reliably through crowdsourcing, which means they other methods attempting video based analysis should be able to rely on tool annotations as auxiliary data.
Additionally, recent Work by Gao *et al.* [8] has demonstrated the effectiveness of a query-by-example approach using kinematic data, and work by Zhao *et al.* [5] suggests methods for fusing spatial, temporal and different kinds of data in a pipeline consisting of more than one traditional neural network model.

IV. Methods

In this section we describe the methods that we used to build our pipeline. We describe the concepts of Convolutional and Recurrent Neural Networks as well as different Loss functions we used.

A. Convolutional Neural Network

Convolutional Neural Networks are extremely popular in the computer vision domain currently. As a result, there is an abundance of existing architectures designed to encode the spatial information found in an image. This network architecture is based on convolutional filters to learn spatial features on a progressively more general scale. To learn a large amount of generalizable features, it is common to stack multiple layers of convolutional filters and max-pooling operations when designing a model.

B. Recurrent Neural Network

Recurrent Neural Networks are network architectures well suited for dealing with sequential information of varied length. For this reason, they are often used to encode time-based information. The work of Zhao *et al.* [5] describes at length how RNNs work as well as several variations of possible RNN models. RNNs have a hidden state h at each time step that is determined by the current input and the previous hidden state $h1$. The following equation describes the activation of a vanilla RNN unit as shown in [5].

$$h_t = \sigma\left(\mathbf{W}\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}\right)$$
$$y_t = \sigma\left(\mathbf{V}h_t\right)$$

Here σ is a non-linear activation function, usually the standard logistic function or hyperbolic tangent. x_t is the input at time t. y_t is the output at that same time step, and $\mathbf{W}$ and $\mathbf{V}$ are the network parameters. This allows the network to retain information from the entire sequence of inputs. This architecture, however, is ineffective for long time step sequences due to the vanishing gradient problem. An improvement to the vanilla RNN unit was the development of LSTM (Long Short-Term Memory) units. This architecture uses three gated cells to retain more information about the different hidden states and properly propagate errors through hundreds of thousands of time steps.

C. Cross-Entropy Loss

Cross-Entropy is one of the most commonly used loss functions for Neural Network classification tasks. It increases as the probability assigned to each classification label diverges from the true label. Formally, this loss is defined as

$$L_{CE} = \sum_{i=1}^{10} b_i p(y_i) \qquad p(y_i) = \frac{1}{Z}\exp\left\{l_i\right\}$$

Where b_i is a binary variable that takes on values either 0 or 1, 0 if the class of $\mathbf{I}_t$ is not i, and 1 otherwise. Z is the normalizing factor that makes sure the probabilities sum up to 1.

D. Triplet Loss

Triplet loss was introduced by Schoff *et al.* for the purpose of face recognition and clustering [3]. The triplet loss function takes in a triplet which consists of the features of three different exemplars (normalized to be unit vectors), known as the anchor, the positive, and the negative (denoted $\vec{f_a}$, $\vec{f_p}$, and $\vec{f_n}$. The anchor and the positive encodings belong to the same class, while the anchor and the negative belong to different classes. Formally, the triplet loss is then computed to be

$$L_T = \left[||f_a - f_p||^2 - ||f_a - f_n||^2 + \alpha\right]_+$$

As can be seen, this loss is positive if the Euclidean distance between the anchor and the positive is not smaller by the distance between the anchor and the negative by a value of at least α. Therefore, this loss tries to reduce distance between feature vectors of the same class, while increasing distance between feature vectors of different classes.

V. Experiment

In this section we describe the complete pipeline, illustrated in Figure 1. Our method can be broken down into two three steps. First, we use a spatial convolutional neural network in order to analyze each frame of a video clip. The goal of this network is to capture information such as the tools that are present in each frame of the clip, as well as the state and appearance of the eye. Next, we use a recurrent neural network in order to process this information along with tool presence information across the temporal domain. This produces a singular encoding that represents the entire clip. Finally, we set up our database of surgical clips and use a nearest neighbor approach to classify new surgical query clips. Implementation details and model specifics can be found in the Appendix.

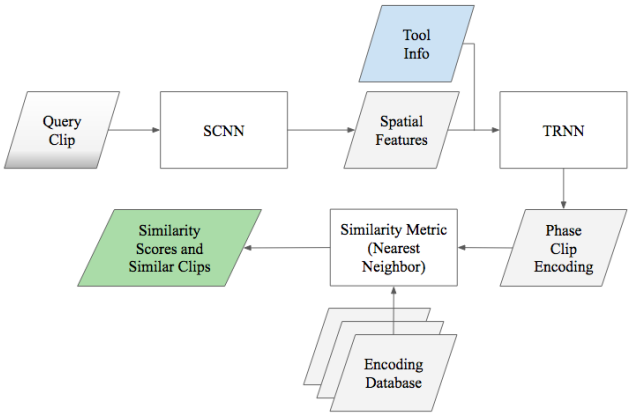

Fig. 1. The flowchart for the pipeline. A query clip video will have the spatial and temporal features encoded through two neural networks, and then the database will be queried for similar clips.

A. Spatial Convolutional Neural Network (SCNN)

Our problem for this portion of the network is phrased as follows: Given a clip of a surgical phase, how many of the frames can we successfully classify as the correct phase, independent of the classification of any other frame?

Let $\mathbf{P} = p_1, p_2, \ldots, p_{10}$ be the 10 phases that can occur during cataract surgery. The legend detailing which p_i corresponds with each phase can be found in the appendix. Furthermore, let $\mathbf{I}_t$ denote the RGB image at timestep $t \in \mathbf{Z}$, $[1, T]$ for some phase clip $\mathbf{V}$. Because $\mathbf{V}$ is a video of a singular phase in the cataract surgery, we can assume that for all timesteps t, the phase of the image at that timestep is p_i, where i corresponds to one of the possible phases found in cataract surgery. We train a neural network that tries to correctly classify as many of the RGB images as possible across all of our training videos. In order to to do this, we decided to base our architecture off of the SqueezeNet architecture [1], with the primary motivator stemming from the low amount of parameters the network has. This makes training much faster, as well as gives us a simpler architecture for the simpler problem of a 10 way classification. The SqueezeNet architecture is built off of fire modules, which is described in the original paper as convolutions with 1x1 filters to squeeze the inputs, followed by convolutions with 1x1 and 3x3 filters to expand it. SqueezeNet puts the input image through eight of these fire modules, while also taking a global average pooling after the fourth and eighth modules. By the end of this, the image has gone through a series of transformations and is now represented by matrix $\mathbf{H}_t$, with dimensions $13 \times 13 \times 512$. From there, one final convolutional layer occurs, transforming the dimensions to $13 \times 13 \times 10$, and this is passed through a global max pooling across the first and second dimensions to achieve a $1 \times 1 \times 10$ vector, $\vec{l}_t$. This vector is then used for phase classification of the image. We train the neural network for phase classification using cross entropy loss.

Although we need the phase classification portion of the network in order to properly train our model, we are more interested in the feature vector encoding given to the image before this classification occurs. We obtain this feature vector encoding by taking $\mathbf{H}_t$ and running a global max pooling across the 1st and 2nd dimensions in order to collapse $\mathbf{H}_t$ into a vector $\vec{s}_t$. We take $\vec{s}_t$ to be the feature encoding output of the network, given an image $\mathbf{I}_t$. Figure 2 gives the overview of the SCNN process.

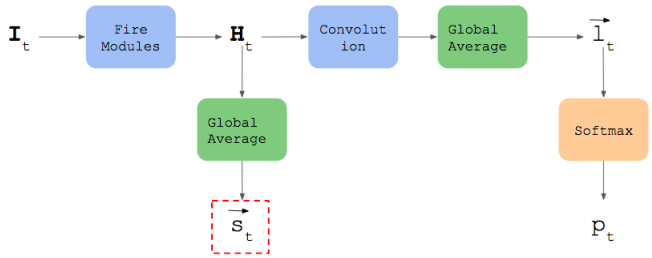

Fig. 2. The structure of the SCNN. The image at each timestep, $\mathbf{I}_t$, is fed into this network to get the corresponding $\vec{s}_t$ For training purposes, we also need to predict the phase classification of the image, which is done by generating $\vec{l}_t$ to predict p.

B. Temporal Recurrent Neural Network (TRNN)

All images in video $\mathbf{V}$ are fed through the SCNN, and we end up with a matrix representation $\mathbf{S}$ with dimensions $T \times 512$. Each row t represents the encoding outputted by the SCNN at timestep t. Furthermore, we bring in the tool label information. There are 16 unique tools used in cataract surgery, and thus the tool label information is formatted as a $T \times 16$ dimension matrix $\mathbf{A}$, and each element in the matrix $T_{ij} \in \{0, 1\}$ represents whether that specific tool j is present within the current frame i. We concatenate $\mathbf{S}$ with $\mathbf{A}$ to form the input into the TRNN, denoted as $\mathbf{C}$.

The problem the TRNN tries to solve is the following: How can we capture information across the temporal domain while simultaneously making sure encodings that come from clips of the same phase are similar, while clips of different phases are dissimilar?

To solve the first part of the problem, we introduce a basic recurrent neural network architecture with LSTM nodes [2]. Each timestep in C is fed into the LSTM layer (as a vector c_t), and all outputs are saved. Afterwards, we take the average of these outputs in order to gain a summary of the temporal

information over the entirety of the video. We do this rather than just take the last output of the LSTM, which theoretically contains information over all frames of the video, because frames near the beginning are very underrepresented in the final output of the LSTM. This produces a single vector of length 512, which we then reduce down to a feature vector with length 128 through a fully connected layer. This vector is then normalized to a unit length 1, which becomes our final video encoding, called $\vec{f}$. Figure 3 give a basic overview of how this vector $\vec{f}$ is generated.

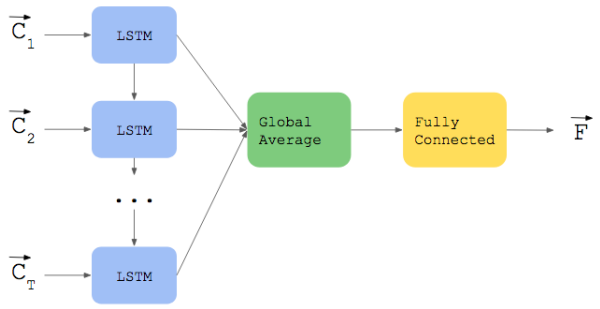

Fig. 3. The structure of the RNN. We first individually feed each timestep into the LSTM layer, and then take the average of the outputs across all timesteps. This average is then fed into the fully connected layer, giving us our output feature encoding $\vec{f}$.

To solve the second part of the problem, we use the loss function known as triplet loss [3] in order to increase the Euclidean distance in feature encodings of clips of different phases. This gives us an encoding for our phase clips which meets our requirements, without performing explicit phase classification during this training step.

C. Nearest Neighbor Querying

Now that we have a method of encoding phase clips V into feature vectors $\vec{f}$ through the SCNN and TRNN, we can construct a database of feature encodings that correspond to the phase clips that were used during the training of the neural networks. Each of the 382 training clip is passed through the SCNN/TRNN pipeline and encoded. These encodings make up our database to be queried. Afterwards, given a query clip, we can pass it through the pipeline as well to encode it, and look through the database to find similar clips through a nearest neighbors approach using Euclidean distance as a similarity measurement.

VI. Dataset

The training and evaluation of our pipeline was done on a novel dataset generated by those involved with the project. 66 entire cataract surgery videos were taken from 1 expert and multiple trainees at the Wilmer Eye Institute, and ground-truth phase segmentation annotations were generated by members of the project. These annotations were provided in text files, and came in two separate formats. We were also given information to simulate tool tip annotation information, since these could not be labeled in time for the completion of this project. For each phase, we were told the probability of each tool appearing, as well as what fraction of the total time the tool would appear for if it shows up. This information can be found within the supplementary figures of the appendix. Of the 66 whole videos, 45 of them were used to train our machine learning pipeline and also became our cataract surgery database. 8 of them were used to validate the accuracy of our models during the training procedure. Finally, the remaining 13 videos were used as a test dataset, in which we passed the clips generated from these videos through the already trained pipeline and queried the database for clips that are ”similar”.

Given the videos and annotations, we did the following pre-processing. From the ground truth segmentation, we broke up the whole surgery videos into phase clips, and sorted the clips based on which phase they belonged to. On average, each video yielded 9 phase clips, giving us a total of 382 phase clips for training, 71 phase clips for validating, and 136 phase clips for our final testing. The full breakdown for how many clips are in each of the phases can be found in figure 8 of the appendix.

At the same time, we also simulated tool tip annotations $\mathbf{A}$. For each clip, we look at the phase that the clip belongs to, and generate the annotations as follows.

1) For each tool, Check to see if the tool deterministically appears. If it does, check to see for what fraction of the duration it will be in the clip (call this fraction d).
2) If the tool does not deterministically appear, randomly decide whether the tool will appear for the specific clip or not with a fixed probability.
3) If the tool does show up, randomly sample the duration d to be between 0.1 and 0.4.
4) Choose a random start point within the clip, making sure that this start point is no later than d away from the end of the clip.
5) Set matrix $\mathbf{A}$ to contain the tool from the start point for the length of the duration d.

VII. Results

We first visualize our database consisting of training encodings on a 2D plane, as seen in figure 4. This is done using Principle Component Analysis to reduce the dimensions of the encodings for each phase clip from 128 to 2, and then plotting these top two principle components. This allows us to qualitatively visualize the effectiveness of using triplet loss in order to separate phase encodings that belong to different classes, and cluster together phase encodings that belong to the same class.

Furthermore, we quantitatively evaluate our pipeline by analyzing the classification accuracy on the test dataset, which is the dataset that the model did not see during the training procedure. We define the following method of classification. Given our database of phase clip encodings, we can send our test dataset through the SCNN/TRNN pipeline in order to encode them as well, and ”query” the database through the nearest neighbors method. We then classify each test clip to be the same class as the clip that is closest to it in the database.

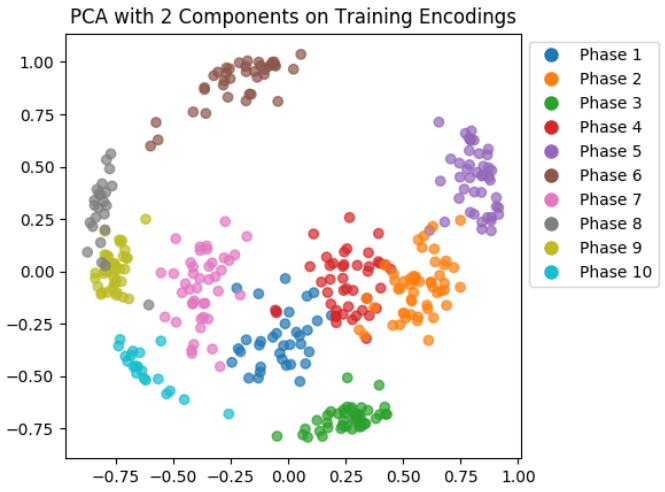

Fig. 4. Visualization of our database features generated through the pipeline that was trained on both video clips and tool annotations, colored according to the phases. Distinct clusters can be seen, although there are regions of overlap.

	Precision Video	Recall Video	Precision Video + Tool	Recall Video + Tool
Phase 1	0.000	0.000	1.000	0.364
Phase 2	0.727	0.615	1.000	0.769
Phase 3	0.538	0.538	0.733	0.846
Phase 4	0.444	0.154	0.917	0.846
Phase 5	0.417	0.769	0.706	0.923
Phase 6	0.229	0.786	0.387	0.857
Phase 7	0.154	0.133	0.789	1.000
Phase 8	0.083	0.067	0.667	0.133
Phase 9	1.00	0.278	0.688	0.611
Phase 10	0.667	0.200	1.000	0.800

Fig. 5. Table that shows precision and recall across the various phases. The first two columns correspond to the model trained solely on phase video clips, while the second two columns correspond to the model trained on both phase video clips and simulated label annotations. Both precision and recall improve across all phases when tool labels are introduced.

In other words, given that our query clip is q, and and $p(i)$ denotes the phase of query clip i,

$$p(q) = p(\text{argmin}_i ||\vec{f_q} - \vec{f_i}||^2)$$

We look at overall accuracy of our pipeline, as well as the precision and recall for each individual phase. Furthermore, we also look at the confusion matrix generated to see which phases are most commonly getting mixed up.

Aside from training our pipeline on both videos and tool annotations, we also trained our pipeline to classify phase using only video information to quantify the usefulness of including tool annotations within our training procedure as well. In the end, we obtained a classification accuracy of 0.356 using only video information, while our accuracy increases to 0.711 when using both videos and tool annotations. Figures 5, 6 and 7 summarize the results.

VIII. Discussion

By looking at Figure 4, we can see that our pipeline which uses triplet loss creates distinct clusters for each phase. Some phases, such as phase 1 (side incision) and phase 4 (hydrodissection), seem to have small regions of overlap. However, distinct areas are still shown.

By looking at figures 5, 6, and 7, we can see that we achieve our best results from using both video and tool label information. Accuracy, precision, and recall improve for each phase with this addition, with the accuracy of the model trained on both tools and videos to reach 71.1%, while the accuracy of the model trained using only videos reached 35.6%. Furthermore, there were noticeable jumps in both precision and recall in phase 4, going from 44.4% precision and 15.4% recall to 91.7% and 84.6% recall. Similar improvements can be observed for phases 7 and 10 as well. This is to be expected, since tool annotation information is highly separable based on phase. As a matter of fact, if we were to train a model purely on tool annotation information,

Fig. 6. Confusion matrix of the classification of cataract phase across the 10 different phases, trained using only video information. We achieve an accuracy of 0.356, significantly lower than the classification trained on both video information and tool annotations.

our classification accuracy would rise even further. However, the ultimate goal is to create an encoding that captures spatio-temporal information. By using tool label information only to steer the encodings towards proper classification rather than replace the spatio-temporal information, we improve our ability to cluster together the encodings based on phase while maintaining the information we desire within the encodings themselves.

In our confusion matrix, it can be seen that the majority of incorrect classifications come from phase 1 (side incision)

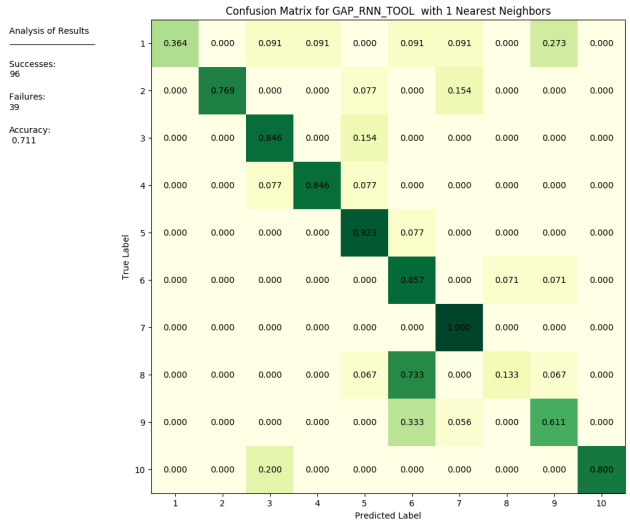

Fig. 7. Confusion matrix of the classification of cataract phase across the 10 different phases, trained using both video information and tool information. Compared to training with only video information, our classification accuracy is much higher, at 0.711. Note that most of our mistakes come from phases 1 and 8.

and phase 8 (OVD removal), while accuracy in other phases remain high (above 60% accuracy for all other phases). Phase 1's misclassification is spread out through a variety of different classes. This may be due to the brevity of clips for the phase itself, which does not give our model enough information to create an sufficient encoding. Regardless, the side incision is one of the more trivial portions of the surgery, and is not one subject to heavy inspection. Phase 8 is heavily classified as phase 6. This classification error has multiple explanations: The most prominent can be seen by observing the information used to simulate tool annotations, found in figure 9. As can be seen there, there is high chance that the tool annotations that are generated for these two phases are identical. This can lead to misclassification. Another reason why phase 6 is preferred is due to the larger number of training samples for phase 6. We hope that by obtaining more training samples across all phases, which will help define our clusters better, and obtain real tool annotations, which will reduce the issue of having identical annotations between phases, the accuracy will increase.

IX. Conclusion

In short, we have shown that including tool annotations is useful in phase classification and creating a meaningful encoding of videos. Furthermore, we achieved reasonable accuracy in phase classification using our method, and showed that our current encoding pipeline of SCNN/TRNN trained using triplet loss successfully creates separable regions for each of the classes. Finally, we identified shortcomings within our model, and proposed that having more training data as well as real tool annotations may help rectify these issues.

Acknowledgment

The authors would like to thank the entire Cataract Group for their support and mentorship, including Tae Soo Kim, Dr. Haider Ali, Dr. Austin Reiter, Dr. Swaroop Vedula, Xinyi Chen and Anand Malpani.

We would also like to thank Dr. Gregory Hager and the students in his lab for their assistance and feedback, particularly Robert DiPietro and Michael Peven.

The data in this study was collected with support from the Wilmer Eye Institute Pooled Professor's fund awarded to Dr. Shameema Sikder.

Finally, we would like to thank the instructor and Teaching Assistant of the Advanced Computer Integrated Surgery course at Johns Hopkins University, Dr. Russel Taylor, Ehsan Azimi.

References

[1] F. Iandola, S. Song, M. Moskewicz, K. Ashraf, W. Dally J., K. Keutzer, *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size*, ICLR Conference, 2017.

[2] S. Hochreiter, J. Schmidhuber, *Long Short-term Memory*, Neural Computation 9(8):1735-1780, 1997

[3] F. Schoff, D. Kalenchenko, J. Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clustering*, CVPR Conference, 2015.

[4] C Lea., J. H. Choi, A. Reiter and G. D. Hager, *Surgical Phase Recognition: from Instrumented ORs to Hospitals Around the World*, M2CAI workshop, 2016.

[5] R. Zhao, H. Ali and P. van der Smargt, *Two-Stream RNN/CNN for Action Recognition in 3D Videos*, 2017.

[6] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, *Learning Spatiotemporal Features with 3D Convolutional Networks.*, 2015 IEEE International Conference on Computer Vision (ICCV), 2015. doi:10.1109/iccv.2015.510.

[7] S. Chopra, R. Hadsell, and Y. Lecun. *Learning a Similarity Metric Discriminatively, with Application to Face Verification.* 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05), 2005. doi:10.1109/cvpr.2005.202.

[8] Y. Gao, S. S. Vedula, G. I. Lee, M. R. Lee, S. Khudanpur, and G. Hager. *Query-by-example surgical activity detection.* International Journal of Computer Assisted Radiology and Surgery 11, no. 6 (April 12, 2016): 987-96. doi:10.1007/s11548-016-1386-3.

[9] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition.* 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. doi:10.1109/cvpr.2016.90

[10] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. *Temporal Convolutional Networks for Action Segmentation and Detection.* 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. doi:10.1109/cvpr.2017.113.

[11] T. S. Kim, A. Malpani, S. S. Vedula, A. Reiter, S. Sikder, *Advancing Surgeon Training through Data Science: Pilot Study of Cataract Surgical Tool Annotation through Crowd Sourcing*, Association for Cataract and Refractive Surgery Annual Meeting. April 13-17, 2018.

[12] CATARACTS: Challenge on Automatic Tool Annotation for cataRACT surgery, 2017, *https://cataracts.grand-challenge.org*

APPENDIX A
SUPPLEMENTARY FIGURES

	Activity of Phase	Train #	Val #	Test #
Phase 1	Side Incision	35	9	12
Phase 2	Main Incision	51	9	13
Phase 3	Capsulorhexis	43	8	13
Phase 4	Hydrodissection	43	8	13
Phase 5	Phacoemulsification	44	8	13
Phase 6	Cortical Removal	38	7	14
Phase 7	Lens Insertion	43	7	15
Phase 8	OVD Removal	24	8	15
Phase 9	Corneal Hydration	41	7	18
Phase 10	Suture Incision	20	0	10

Fig. 8. Information on each phase of cataract surgery. The 2nd column provides a legend for the action that is performed during each numbered phase, and the following three columns give the number of clips that are in each of the three datasets.

	Definitely Used, Full Phase	Definitely used, part of phase	May be used (random duration between 0.1 and 0.4)
Phase 1	1		
Phase 2	2		4 (0.1)
Phase 3		6 (0.2), 7 (0.8)	3 (0.1), 15 (0.4)
Phase 4	8		
Phase 5	9		15 (0.3)
Phase 6	10		
Phase 7	11	12 (0.4)	
Phase 8	10		3 (0.1)
Phase 9	15	14 (0.2)	
Phase 10	16, 4		

Fig. 9. Information given to simulate which tools appear will appear in each phase of the cataract surgery. The tools are numbered 1 through 16. In the second column, the fraction within the parentheses mark the duration that the tool will appear for, while in the third column, the fraction denotes the probability that the tool maybe used. Empty spaces mean that no tools apply under the specific context.

APPENDIX B
IMPLEMENTATION DETAILS

All code was written with Python, using the CV2 and skvideo.io packages for video processing, scikit-learn for nearest neighbors, and PyTorch for neural network implementations.

A. SCNN Model

The SCNN, once again, is based off of the SqueezeNet architecture [1]. SqueezeNet consists of multiple fire modules. Each module contains a series of s_1 1×1 convolutional filters to "squeeze" the information followed by a set of e_1 1×1 and e_3 3×3 filters to expand. In the standard architecture, there are 8 fire modules. The number of filters in each of the modules is given by the table:

SqueezeNet is implemented as follows:

	s_1	e_1	e_3
Module 1	16	64	64
Module 2	16	64	64
Module 3	32	128	128
Module 4	32	128	128
Module 5	48	192	192
Module 6	48	192	192
Module 7	64	256	256
Module 8	64	256	256

Fig. 10. Information on the number of filters in each fire module. Note that in the SqueezeNet paper, these modules are labeled "fire2" through "fire9".

1) A convolutional layer with 96 7×7 filters
2) fire2 through fire4
3) Max pooling
4) fire5 through fire8
5) Max pooling
6) fire 9
7) Convolution with n filters, where n is the number of classes. In our case, n is 10.
8) Global Average Pooling
9) Softmax.

We take PyTorch's implementation of SqueezeNet from their pretrainedmodels package, already pre-trained on ImageNet. We then replaced the last convolutional layer with one that has 10 classes rather than 1000, and fine-tuned the model to classify phase on cataract surgery images. The output dimension of the feature vector of SqueezeNet is 512.

B. TRNN

The TRNN is relatively simple. Because the output dimension of SqueezeNet is 512 and the dimensions of the tool annotations are 16, the TRNN takes in inputs of size 528, over some time-series T. the TRNN is structured as such:

1) An LSTM layer that outputs a 528 dimensional vector. The input at each time-step is fed into this layer and saved.
2) A global average pooling across time-steps.

3) A fully connected layer that outputs a feature vector of length 128.
4) A vector normalization that turns the output vector into unit length. This is done for triplet loss.

This model is implemented in PyTorch, and then trained from scratch by us.

C. Nearest Neighbors

The Nearest Neighbors querying is implemented through sklearn.neighbors. A Nearest Neighbors object is fit to the database of training video encodings with a Euclidean distance metric. Query encodings are then read in, and the nearest neighbors are grabbed from this fitted object.

APPENDIX C
CIS II COURSE REQUIREMENTS

This section includes final details for the evaluation of this project in the context of the Computer Integrated Surgery II course.

A. Deliverables

The deliverables of this project were updated twice during the semester in agreement with our mentors in order to make this project more realistic and adapt to an unexpectedly low accuracy of the spatial CNN in early versions of the project. The final deliverables as agreed upon by our mentors were the following:

- Minimum
  1) Design Document documenting the code and model description
  2) Create a working, well documented pipeline to generate video descriptors given a surgical clip of an activity.
  3) develop a similarity metric that can discriminate between clips of same and different activities.
- Expected
  1) Validate our model by analyzing similarity scores activity clips in our dataset with target accuracy 30% on single-frame extractor, 60% after including temporal features
  2) Submit a paper-style report documenting our findings that could be used as first draft of a manuscript for publication
- Maximum
  1) Use tool tip annotations (simulated if real are not available) to improve classification accuracy

All of the deliverables above were achieved successfully.

B. Distribution of work

The SCNN based on SqueezeNet was provided by our mentor Tae Soo Kim. All other aspects were implemented equally and together by Felix Yu and Gianluca Croso. Documentation of code was mostly done by Gianluca, while Felix focused more on writing the report.

C. Plan modifications

1) Deliverable changes: Our initial deliverables included updating the pipeline to classify surgeon skill and even possibly creating a method to rank different surgeries in terms of skill. The inclusion of these reflected an overly optimistic belief that phase classification would not be too difficult with the pipeline we had in mind.

After a few meetings with Dr. Haider Ali, who specializes in computer vision and later became one of the mentors of this project, we realized that it was an unrealistic goal. He believed that a better approach was to train for phase classification exclusively with expert surgeons in order to decrease the intra-class variability of the examples, and then implement fine-tuning using associative loss to classify the novice surgeons. This would be a more complex form of phase classification, but the encoding would hopefully intrinsically capture some of the differences between novices and experts thus making skill classification easier as future work. We updated our deliverables accordingly, removing skill classification and introducing associative loss as our maximum deliverable.

As we worked on the project, most of our attempts at the spatial feature extractor (the first part of the pipeline) were unexpectedly unsuccessful (we will describe these attempts in the next subsection), which severely delayed our progress. After we eventually found a reasonably good architecture with the help of our mentor Tae Soo Kim, and implemented the temporal encoder, we were getting clearly non-random results, but the accuracy was still insufficient. By then we had realized our dataset was very difficult to classify, partly due to the lack of standard in filming, which was causing a large intra-class variation far beyond what we would have expected between experts and novices. That led to the of using auxiliary data in the form tool labels, which should be helpful in identifying phases, and to the final update to our deliverables, which substituted the implementation of associative loss for the implementation and fusion of tool label data. As we did not have them readily available and did not have the time to obtain it, we simulated that data based on the knowledge of our mentors for which tools should appear in which phases, for which duration and with which probability.

2) Unsuccessful approaches: We had many unsuccessful attempts at designing the spacial feature extractor. Thankfully, the input data for the temporal extractor was significantly simpler (a matrix of feature vectors instead of a video) and therefore our design for that step was significantly more successful. The first model we used for the spatial feature extractor was called C3D [6]. It is a deep network that includes many layers of 3D convolutions to extract spatiotemporal information over short clips. We trained it using Triplet Loss. It is a very complex model that we believed would be able to go beyond learning just the spatial features but also capture some movement information if trained over 1 second segments of the clips. The results were unfortunately only barely better than random. We thought the loss function might be the problem, and attempted training the model with the more commonly used Cross-Entropy Loss, with no improvement. We then attempted training on individual frames using ResNet [9], a

complex but widely used and generally very successful model for image interpretation, also without significant success using either loss function. Our final unsuccessful attempt happened when we were exploring the possibility of converting the data to optical flow, and training a network that fused a ResNet with an LSTM layer over one second clips of optical flow data. We converted the data and trained the model but that approach also failed to produce adequate accuracy. However that happened around the same time as we were successful with SqueezeNet, at which point we moved on to the temporal encoder.

D. Challenges and Lessons Learned

As mentioned in the plan modifications, we initially underestimated the difficulty of the phase classification problem in this dataset. Our initial plan to use C3D as the spatial feature extractor did not work well, and we spend several more weeks than originally planed trying to improve the accuracy of the first part of the pipeline. The model later suggested by our mentor Tae Soo Kim, SqueezeNet, was much simpler than our initial model, but obtained significantly better results.
This taught us two important lessons. First, that it is not always a good idea to use a very complicated model from the start. Simpler models are easier to implement and faster to train, and they can give us insight on what is the right kind of complexity to introduce in order to obtain better results. Second, we should not have spent so much time trying to improve an incomplete pipeline. Had we completed the temporal encoder before spending as much time trying to improve the spatial feature extractor, we might have obtained valuable insight on the shortcomings of the pipeline as a whole.
A third important lesson was learned when we presented partial results to Dr. Hager's lab in early April. We had not given enough attention to the inherent difficulties of the dataset early on. The way the videos were filmed was not standardized, which created large differences in clips that should be of the same class and inherent similarities in videos that were of different classes but part of the same surgery. It would have been a good idea to test our pipeline on a simpler dataset in order to evaluate its strengths and weaknesses before attempting such a difficult task.

E. Future work

We consider that, at the end of the course, the following are the immediate steps to be taken with this project:

- Review manuscript to submit for publication either in PlosOne or JAMA Open.
- Work will be continued by cataract group with sparse involvement by Felix and Gianluca.
- Future work to improve results and usefulness of project includes training model on larger dataset, investigating skill related encodings, and obtaining more fine-grain tool annotations.

F. Code and Documentation

A zip archive containing the all the code for this project is available in the project Wiki page along with a complete user's manual / documentation guide detailing how to use each and every file in the final version of the pipeline. The code itself of the files used in the final version of the pipeline is also well documented with every function including a description along with expected inputs and outputs. Both the code and the documentation guide are uploaded in the "Reports and presentations" section of the wiki under the item "Project code & documentation" right after this final report.