Assignment 2 Report
Surgery Phase Recognition
Team HAAV

Antonin Vidon
MS Data Science
Columbia University
New York, NY
av3023@columbia.edu

Hugo Artigas
MS Business Analytics
Columbia University
New York, NY
ha2605@columbia.edu

***Abstract*—With the unquestionable success of Artificial Intelligence in object detection, recent years have seen a surge in research efforts to leverage its full potential in health care. In this context, state-of-the-art CNN based architectures have been developed to automatically recognize surgical phases in a supervised learning fashion. This report contains our work (team HAAV) on a dataset of 193 hernia surgery videos whose frames all belonged to one of 14 phases treated as labels.**

Due to the large amount of frames and our limited computing resources, we had to constitute our own 'tiny' dataset : each frame was resized and saved for fast and convenient loading at training time. We exploited pretrained CNN architectures such as ResNet and MobileNet to serve as backbone of our models. We completed our architectures with LSTM (MobileNetLSTM) or linear layers (MobileNetFC) to model temporal correlation between frames and eventually added stage of the operation as feature (MobileNetStage). In order to ensure higher performance, we processed the label predictions of our models with a smoothing function. Our best model achieved an accuracy of about 79.9% on the test set.

***Index Terms*—Surgical phase recognition, video classification, object detection, CNN, RNN, LSTM.**

I. Introduction

Surgical phase recognition has become an important matter in clinical care. Hospitals hope to use the algorithms to help surgeons during their interventions, e.g. by determining the remaining time of the current operation in order to optimize the use of their surgery rooms.
Recently, numerous Machine Learning frameworks have been used to try to crack this task [1]. In our setting, the objective is to predict the surgical phase of each frame of an hernia surgery video. We use accuracy and macro F1 score as evaluation metrics.

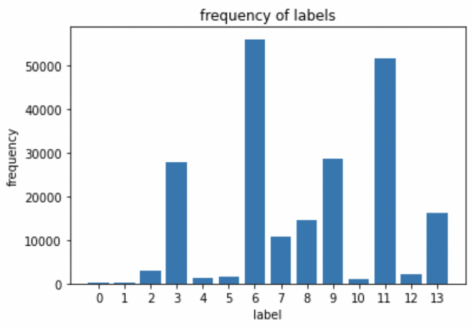
II. Data

A. Raw Data

The initial data is composed of 201 hernia surgery videos with various lengths (∼30min to ∼2h30) and frame dimensions. These videos were collected from surgeries conducted by 4 different surgeons and were shot at 1 frame per second (*fps*), although this value does not impact our study in any way. A subset of the raw data is made of 70 labelled training videos and 60 unlabelled testing videos. The labels were not directly provided but we were given the start and end time of each phase for every training video.

B. Preprocessing

In order not to have to load a whole video at training time, which would have caused the memory of our NVIDIA® T4 GPU to run out and would have slowed down model learning, we chose to save all frames of our videos separately. We opted for the well-known (224,224,3) dimension as resizing parameter in order not to have to build our own backbones from scratch. Indeed, a lower dimension would have caused the size of our features to drop down too quickly during the convolution process with a ResNet or MobileNet architecture. We created a dataframe that contains the label for each frame, and added as feature the number of frames of the video it belongs to. We also saved once and for all our training, validation (80% split) and testing dataframes that will be used to load the features by our custom PyTorch dataloaders.



adhesiolysis : 0
blurry : 1
catheter insertion : 2
mesh placement : 3
mesh positioning : 4
out of body : 5
peritoneal closure : 6
peritoneal scoring : 7
positioning suture : 8
preperioneal dissection :
primary hernia repair : 10
reduction of hernia : 11
stationary idle : 12
transitionary idle : 13

Fig. 1. Distribution of labels among video frames

The histogram of labels above reveals that some phases are largely predominant among frames such as *peritoneal closure* or *reduction of hernia*, making our data severely

imbalanced. With this in mind, it was important to keep an eye on the entropy of our prediction as our model might completely overlook the minority classes. A single look at the accuracy isn't completely relevant and other metrics such as the F1 score had to be used for a better picture of our model ability. This way, all classes would be given some importance with regards to the assessment of our results.

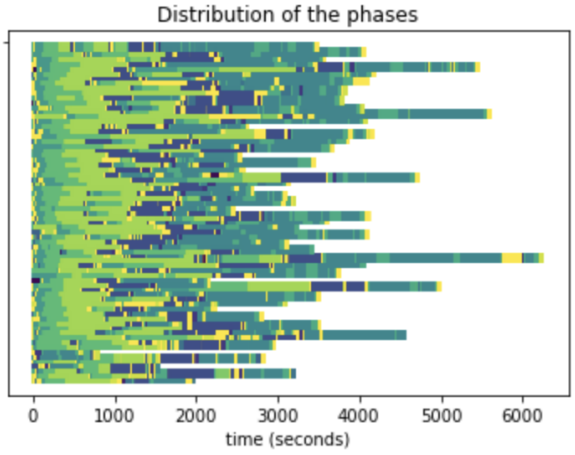


Fig. 2. The different phases over the videos

The histogram above depicts the temporal distribution of labels within training videos. Looking at the plot, it is obvious that the correlation between label and surgery progression is high, as labels seem to appear in the same order and for the same duration. This gave us the idea to consider the stage of the operation as input feature. By stage, we mean an idea of the time period of the operation from which the frame was extracted.

III. Models

Our method is inspired from several papers including [2]. Our architectures used either ResNet18 [3] or MobileNetV2 [4] as backbone to obtain highly relevant feature maps. We chose these architectures because of their high performance on common benchmarks and because of their relative light weight which would prevent too much overfitting on our data. We completed our backbones with various output layers, some of them modeling temporal correlation between successive frames or taking into account surgery progression.
All our code can be found here[1].

A. Our four main models : MobileNet, MobileNetStage, MobileNetLSTM and MobileNetFC 3

1) MobileNet: This architecture is the most elementary one. The output feature maps of our MobileNetV2 backbone is fed into a linear layer whose output size is the number of classes in our classification task.

[1]https://github.com/ecbme6040/e6691-2022spring-assign2-HAAV-av3023-ha2605

2) MobileNetStage: As we saw before **??**, knowing the time period from which the input frame was taken gives a good indication of its possible label. To exploit this idea, we fed to our network a one-hot-encoded vector of size 20 representing where each index stands for a 5% range the total video length. A 1 is placed for the range to which the frame belongs.

3) MobileNetLSTM: We observed that the labels of two consecutive frames are extremely highly correlated. In other words, if a frame belongs to label 0, there is a very high chance that the next one should also be labelled as such. With this in mind, instead of considering the surgery stage as input, we chose to process our data in batches of consecutive frames. To do so, an LSTM is added before the last linear layer in order and models the temporal correlation between those frames. The training of such a model is more complicated, as we need the number of frames for each video seen at training time to be a multiple of our chosen batch size. To overcome this difficulty, we padded each video with black frames in order to be able to process batches of size 64. At training time, the predictions for the black frames are discarded and not taken into account for loss computation. An evolution of this model was to replace the simple LSTM by a bidirectional-LSTM so as to access the information of past and future frames. It's interesting to notice that the use of a bidirectional-LSTM layer is only possible in the case of "offline" learning. This means that the surgery has already happened and we can use the "future" information for every frame. Thus the bi-LSTM network is not applicable in the case of an 'online' use during which the frames are fed to the network while the surgeon is working to predict for example the remaining time of the operation.

4) MobileNetFC: This model is very similar to *MobileNetLSTM*. However this time, instead of modeling temporal correlation between consecutive frames with an LSTM layer, we chose to channelize the feature maps of each frames of the batch and process all of them with a linear layer.

B. The label smoothing operation

In order to smooth the output labels of our models, we developed a smoothing function. This idea came from the observation that a lot of noisy labels appeared within our predictions, although we knew that only the start and end time of each phase were provided for evaluation of our models performance on test data. Hence, we chose to replace the label of a given frame if another label appeared at majority within a window of surrounding labels, typically of size 7, 11, 13 or 15.

IV. Training

We used random horizontal flipping for data augmentation. Some of our trainings were conducted with oversampling to overcome the issue of class imbalance, but led to longer tuning. The learning rates we chose were around .001 for finetuning

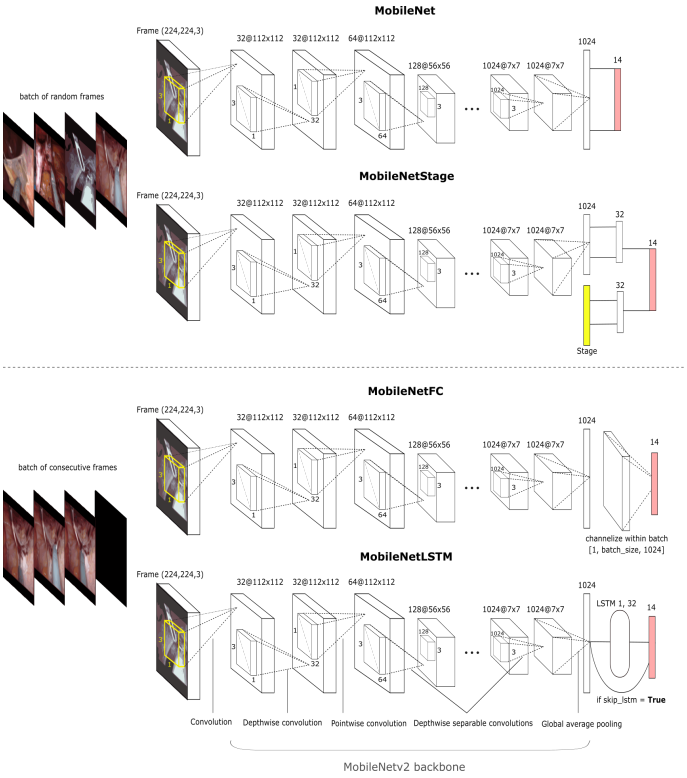


Fig. 3. The 4 different models

and .005 for output layers tuning, with exponential decrease throughout epochs. We processed our frames by batches of size 64 on NVIDIA® T4 GPU and 384 on NVIDIA® A100 GPU.

The first phase of our training was a fine-tuning of the backbone of the architecture. This is a very delicate phase as, although relatively light, the architectures we chose remained state-of-the-art complex models prone to overfitting. We chose to use pretrained versions of our backbones in order to speed-up the learning and used small learning rates. After this finetuning phase, we froze our backbones and focused on the training of the output layers of our models, so that they could better exploit the feature maps.

V. Results

At evaluation time, we observed that our backbone was quickly capturing the most important features of the frames. After 2 to 3 epochs with a learning rate close to .001, our evaluation accuracy was almost reaching 80%. This shows the efficiency of these state-of-the-art architectures. Since we completely shuffled the frames for backbone finetuning, we did not observe a steep decrease in prediction entropy and our model would predict labels from 12 out of the 14 classes on validation data, only discarding *adhesiolysis* and *blurry*, the two less frequent classes among frames.

The tuning of the output layer of *MobileNet* and *MobileNetStage* was effective, leading to an increase in accuracy of 1-2% vs. accuracy after backbone finetuning. On the other side, the tuning of the output layers of *MobileNetLSTM* and *MobileNetFC* was more tedious. Although we thought our idea to model correlation between consecutive frames was smart, we observed that training accuracy started decreasing when data came by batches of consecutive inputs. After feeding a batch with highly correlated labels and while doing the back propagation, our network somehow understands he has to converge to a constant prediction, equal to the label of the batch observed. Then, when the next batch is fed and the accuracy is computed over it (the accuracy is always computed before doing the back propagation), the result is a very low accuracy because the network tends to output the label of the previous batch which doesn't match the most frequent label of the new batch. This is why we observed very low training accuracy, decreasing very steeply at first. Doing the training with a data set not completely shuffled (i.e. by keeping the time order sequence) implies that, for each back propagation step, the weights of the output layers move in a direction orthogonal to the one it moved for the previous batch at the same learning rate, which is totally detrimental. Validation accuracy on the other side did not decrease as much as training accuracy due to the fact that backpropagation steps were somehow cancelling each other, but could only reaach 70% when the models were fully trained, which was very disappointing. Increasing the batch size could have been a solution but quickly becomes a memory issue with our limited GPU memory, already widely used to store our model weights.

Model	Accuracy	Macro F1 score
MobileNet	0.8037	0.5545
MobileNetStage	0.7988	0.5239
MobileNetLSTM	0.7011	0.3265
MobileNetFC	0.7002	0.3124

TABLE I
MODELS PERFORMANCE OVER VALIDATION SET WITHOUT OVERSAMPLING

In light of the performance of *MobileNetLSTM* and *MobileNetFC* at evaluation time, we decided to focus our ablation study on *MobileNet* and *MobileNetStage*. To be able to compare them, we ran the training of the models over 4 and 8 epochs 4 with similar learning rates (both for finetuning and output layers tuning), exponential decrease factor and batch sizes.

Looking at losses and accuracies evolution at training time, it's clear that *MobileNet* was faster to train and gives better validation results. Both the accuracy and the macro F1 score are 1% higher for *MobileNet* vs. *MobileNetStage*.

To solve the class imbalance issue, we tried to oversample the frames from the less frequent labels. The results are presented in the Figure 5. Adding these new frames tends to increase overfitting of our models and considerably slows

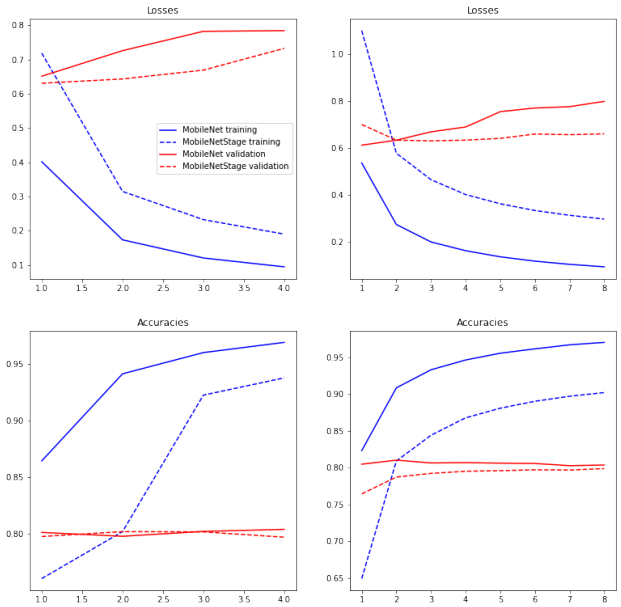


Fig. 4. *MobileNet* vs. *MobileNetStage*. On the left, the models are trained over 4 epochs and on the right over 8 epochs.

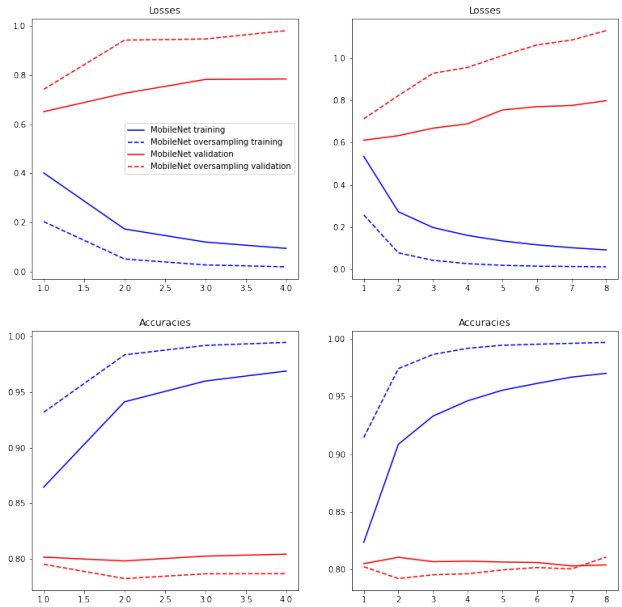


Fig. 5. Ablaation study of oversampling. On the left, the models are trained over 4 epochs and on the right over 8 epochs.

Model	Memory usage of the weights (in Mo)
MobileNet	9.1
MobileNetStage	9.3

TABLE II
TOTAL WEIGHT OF *MobileNet* AND *MobileNetStage*

down training time. The training accuracy increases faster and the loss is lower for the method with oversampling. Overall oversampling during training seems to slightly decrease accuracy. Although prediction has better entropy, the class imbalance is so severe that reaching class balance in the training set is simply unreasonable and leads to faster overfitting.

Model	Training time (min)
No oversampling	85
Oversampling	330

TABLE III
TRAINING TIME WITH AND WITHOUT OVERSAMPLING FOR 4 EPOCHS ON NVIDIA® T4 GPU WITH AND WITHOUT OVERSAMPLING FOR *MobileNet*

Finally, we observed that our smoothing function improved test accuracy by about 2%, with an optimal window size of 13.

VI. CONCLUSION

In order to solve this surgery phase recognition task, we first extracted relevant features from the input frames with a *MobileNetV2* pretrained backbone and then looked at correlation between time and labels, as well as correlation between

Window size	Test accuracy
No smoothing	0.77728
7	0.79557
9	0.79753
11	0.79853
13	0.79919
15	0.79611

TABLE IV
PERFORMANCE OF THE SMOOTHING FUNCTION FOR *MobileNet* PREDICTIONS

labels of consecutive frames. Overall, the most performing model turned out to be the simplest one among the four ones proposed. *MobileNet* with the smoothing output function performed with an accuracy of 0.799 over the test set (Kaggle Competition). Further work could be done to train an LSTM directly on the model prediction instead of incorporating it after the backbone architecture and compare its performance with the one of the smoothing function. A better understanding of time correlations would be required to build an architecture that properly captures the time feature.

REFERENCES

[1] Garrow, Carly R. BSc; Kowalewski, Karl-Friedrich MD; Li, Linhong BSc; Wagner, Martin MD; Schmidt, Mona W. MD; Engelhardt, Sandy PhD; Hashimoto, Daniel A. MD, MS; Kenngott, Hannes G. MD, MSc; Bodenstedt, Sebastian PhD,——; Speidel, Stefanie PhD; Müller-Stich, Beat P. MD, MBA; Nickel, Felix MD, MME " Machine Learning for Surgical Phase Recognition", Annals of Surgery: April 2021 - Volume 273 - Issue 4 - p 684-693, doi: 10.1097/SLA.0000000000004425.

[2] Gaurav Yengera, Didier Mutter, Jacques Marescaux and Nicolas Padoy, ”Less is More: Surgical Phase Recognition with Less Annotations through Self-Supervised Pre-training of CNN-LSTM Networks”, 2018, arXiv.org, https://arxiv.org/abs/1805.08569.

[3] He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” ArXiv:1512.03385 [Cs], Dec. 2015. arXiv.org, http://arxiv.org/abs/1512.03385.

[4] Sandler, Mark, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” ArXiv:1801.04381 [Cs], Mar. 2019. arXiv.org, http://arxiv.org/abs/1801.04381.