University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

Making RetinaNets Context-Aware Using Temporal Sub-Networks For Surgical Action Detection In Laparoscopic Images Of Robot-Assisted Prostatectomies

by

Muhammad Uzair Abid

September 2021

Supervisor: Dr. Adam Prugel-Bennett
Second Examiner: Dr. Srinandan Dasmahapatra

A dissertation submitted in partial fulfilment of the degree
of MSc. Artificial Intelligence

University of Southampton

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Master of Science in Artificial Intelligence

by Muhammad Uzair Abid

Robot-assisted surgeries have become desirable in minimally invasive procedures such as radical prostatectomies due to their reduced risks and high success rate. Granting the ability to detect surgical actions in laparoscopic footage can enable the robot to give critical alerts to the surgical team based on out-of-sequence surgical actions and an anomalous human tissue state. Modern object detectors strive by learning spatial image features in the real world where the environment has sufficient variability between object classes. However, much of the laparoscopic surgical scene consists of similar-looking human tissue with aberrations present from blood, motion blur, and surgical smoke leading to low variance in image features between classes. We present a recurrent model, *TempRetinaNet*, that extends the RetinaNet object detector to learn both spatial image features and temporal features. We also provide a semi-supervised approach to learn surgical phases in a multi-task manner to make the model bundle actions together across time. We train our model using hours of footage and solve the computational overhead involved by implementing truncated backpropagation through time. Our results show that under the computational budget available, TempRetinaNet has a better predictive performance than RetinaNet, when both models are trained with unshuffled data, as the latter fails to learn at all. However, currently, TempRetinaNet does not outperform RetinaNet when the latter is trained with shuffled data and falls short by a mean average precision (mAP) of $\sim$ **15 points**. This work provides a detailed analysis of the problems posed by the dataset; the challenges in training such a model across time, a comparison of several model variations, testing conditions, and next steps to improve training.

Acknowledgements

The author acknowledges the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have built upon the following open-source repositories:
https://github.com/Viveksbawa/SARAS-ESAD-baseline
https://github.com/ndrplz/ConvLSTM_pytorch

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work involves human endoscopic images and has been ethically approved under submission ID: 64671

Contents

List of Figures

List of Tables

Nomenclature

MIS	Minimally Invasive Surgery
R-MIS	Robot-Assisted Minimally Invasive Surgery
SSC	Surgeon Side Console
OR	Operating Room
RARP	Robot-Assisted Radical Prostatectomies
CNN	Convolutional Neural Network
HOG	Histogram of Oriented Gradients
SIFT	Scale Invariant Feature Transform
SSD	Single-Shot Detector
YOLO	You Only Look Once
R-CNN	Region-Based Convolutional Neural Network
SVM	Support Vector Machine
RoI	Region of Interest
RPN	Region Proposal Network
NMS	Non-Maximum Suppression
IoU	Intersection-over-Union
ESAD	Endoscopic Surgical Action Detection
FPN	Feature Pyramid Network
FCN	Fully Convolutional Network
ReLU	Rectified Linear Unit
K	Number of classes
A	Number of anchors
P3, P4...P7	Outputs of FPN at all five layers
FL	Focal Loss
BCE	Binary Cross Entropy loss
α	Balancing factor of binary classes
y	Predicted class
$\hat{y}$	True class
λ	Focusing parameter in FL
LSTM	Long Short Term Memory
ConvLSTM	Convolutional Long Short Term Memory
HHMM	Hierarchical Hidden Markov Model

OvR	One Versus Rest
GRU	Gated Recurrent Unit
BP	Back Propagation
BPTT	Back Propagation Through Time
TBPTT	Truncated Back Propagation Through Time
GPU	Graphic Processing Unit
Conv2d	Two Dimensional Convolutional Layer
3D	Three Dimensional
N	Batch Size
C	Number of channels
H	Height
W	Width
T	Timesteps
N*	Total Batch Size = N $\times$ T
ERNN	Elman Recurrent Neural Network
θ	Weight parameter
CEC	Constant Error Carousel
b	Bias parameter
i_t	Input gate at timestep t
f_t	Forget gate at timestep t
o_t	Output gate at timestep t
$\tilde{\mathcal{C}}_t$	Cell state input activation tensor at timestep t
$\mathcal{C}_t$	Cell state at timestep t
$\mathcal{H}_t$	Hidden State at timestep t
$\mathcal{X}_t$	Input (to the ConvLSTM) at timestep t
$\sigma()$	Sigmoid activation function
$tanh()$	Hyperbolic tangent activation function
W	Weight matrix (or filter matrix)
TempRetinaNet or TRN	Temporal Retina Net
RN	RetinaNet
Cls	Classification
Reg	Regression
P	Number of surgical phases
k1	Number of forward passes to be done before TBPTT
k2	Number of iterations to roll-back to in TBPTT
l_i	Total loss at iteration i
δl_i	Gradient of l_i
$\mathcal{S}$	Set of stored states at previous iterations
$\mathcal{T}_i$	Tuple containing initial state and next state
∇s	Stored gradient information of state s
mAP	Mean Average Precision

AP	Average Precision
β	Momentum parameter in stochastic gradient descent
SGD	Stochastic Gradient Descent
SGDM	SGD with Momentum
R, G, B	Red, Green, Blue channels of image
sfl	Shuffled data
hicon	High-contrast data

Chapter 1

Introduction

1.1 Background

The earliest evidence of surgeries dates back to several thousands of years. Procedures performed back then were trepanations consisting of holes manually drilled into the human skill with aspirations to cure migraines, trauma, and other neurological diseases. From the 19^{th} century to the 20^{th} century, a better understanding of human anatomy, medicine, and technology led to the practice of more advanced procedures such as heart surgery, kidney transplant, and liver transplant, etc.

In the 1970s, the first laparoscopic surgery was performed proving the benefits of minimally invasive surgeries (MIS). MIS consists of a small incision made instead of a large one leading to quicker recovery and reduced surgical risks. Finally, in the 2000s, robot-assisted MIS (R-MIS) was incorporated in surgical theatres providing improvements in surgical precision, reduced human error, and greater operating room efficiency.

R-MIS is a system with two main modules interacting remotely. The patient-side module (SI) consists of robotic arms equipped with tools that perform surgical procedures on the patient. The surgeon side console (SSC) consists of a screen and controllers that enable the surgeon to make precise movements of the robotic arms. While such a system has been proven to pose much lower health risks and yield a high success rate, it serves merely as a remote controller for the surgeon to interact with the patient. This leaves room to make the robot assistant smarter.

By providing the ability to perceive the endoscopic surgical scene, an intelligent robot assistant will be able to provide relevant guidance to the operating team as well as warn of any future complication based on an anomalous human tissue state, excessive blood flow, or an unexpected surgical action that seems out-of-order in the overall procedure. By providing relevant assistance, external complications such as staff-scheduling, operating room (OR) resource management, and safety can also be improved. Furthermore, it can

also be built upon to construct a metric for measuring the surgical skill of the surgeon, leading to improvements in the training of surgical residents as well as their evaluation.

Surgical scene perception is a large set of tasks that can include detecting and localizing surgical actions, recognizing surgical phase, recognizing tool presence, etc. The work provided in this dissertation focuses on surgical action detection and localization and phase recognition in robot-assisted radical prostatectomies (RARP). It first presents a detailed overview of some of the recent best-performing object detectors with some connections made to our final proposed approach. Then, some of the related work in surgical scene perception tasks have been provided. This is followed by a short study of the SARAS-ESAD dataset [Singh Bawa et al. (2021)] used in this work and finally the proposed work. The remainder of the dissertation provides experimental results and ablation studies to argue the motivation and efficacy of the various design choices. The work tackles challenges about the extraction of spatial and temporal information that is robust to variability of the surgeon's skill, the patient's anatomy as well as occlusions in the endoscopic scene caused by blood, smoke, lens blur and motion blur. As a result, this project will serve as a step towards a wider provision of intelligence to the robot assistant.

1.2 Object Detection in Deep Learning

To understand some of the current works and our proposed approach to the surgical action detection problem, it is relevant to first gain an understanding of modern object detection techniques using deep learning.

The problem of object detection is a natural progression of the traditional image classification task. While the latter only seeks to find the existence of a certain class in an image, the former seeks to accurately bound the desired object using bounding boxes by predicting the coordinates of the boxes. Object detection is usually followed by object recognition/classification where the class of the specific object is inferred.

The convolutional neural network (CNN) has become the de facto deep learning technique for working on image-related feature extraction tasks over the past two decades. This biologically-inspired model works on receptive fields similar to that in the visual cortex of the human visual system by producing feature maps from repeated convolutions across receptive fields of the image using filters. These feature maps are usually passed through further layers of convolutions until introduced to a standard dense layer network where the class of the image is predicted. This approach reached massive stardom once back-propagation was introduced for images in [LeCun et al. (1989)] as it meant the weights of these filters could be learned based on the loss of the classifications.

CNNs have been adapted to object detection and recognition tasks where the problem is now a multi-task one consisting of regression of bounding box coordinates and classification of boxed objects. Due to the ability of these deep networks to extract more robust features than manually engineered features such as HOG [Dalal and Triggs (2005)] or SIFT [Lowe (2004)], CNNs have become the primary feature extractors or the "backbone[1]" of the best object detection models. Some highly known models in this realm are one-stage detectors such as SSD [Liu et al. (2015)], YOLO [Redmon et al. (2015)], RetinaNet [Lin et al. (2017)] and two-stage detectors such as the revered Faster R-CNN [Ren et al. (2017)].

1.2.1 Two-stage detectors

Two-stage detectors are so-called because their training process can be broken down into two broad steps: object proposal generation and detection. Object proposals are a set of regions found as a preprocessing step from the image in which objects are likely to exist. By focusing the training on only these regions, we save up time and space as every region of arbitrary size isn't checked for the existence of an object. There are several object proposal generation algorithms proposed, of which selective search is highly known [Uijlings et al. (2013)]. The second stage of two-stage detectors consists of feature extraction using CNNs and some mechanism of fine-tuning feature maps for localization and classification of objects. The functioning of the two-stage detectors mentioned here is relevant to the proposed solution and aren't an exhaustive list of the latest best-performing object detectors.

1.2.1.1 R-CNN

One of the earliest CNN detectors in this category was R-CNN [Girshick et al. (2014)] that distributed the training process into 4 steps: object proposal generation using selective search, independently fine-tuning a CNN feature extractor using a softmax classifier head, using CNN features to train SVM classifiers and finally training bounding-box regressors. Features extracted from the trained CNN had to be written to disk to be later used for SVM classification and bounding-box regression making it time and space-inefficient.

1.2.1.2 Fast R-CNN

Problems posed by time and space complexity were improved by Fast R-CNN [Girshick (2015)] by combining the last three steps in R-CNN using a regions-of-interest (RoI) pooling layer and multi-task loss. Here, the entire image is convoluted using the backbone,

[1]Backbone: A term typically meaning a deep CNN used in a model for feature extraction from an image.

and regions on the feature maps corresponding to the object proposals are extracted using fixed-size pooling layers. The resultant feature maps are then passed through dense layers, the outputs of which are fed into a softmax classifier and a bounding-box regressor. Smooth L1 Loss (a relaxed version of the standard L1-loss) is summed over each regression variable (see Equation 1.1). Variables x, y correspond to the top-left box coordinate and w, h represent the width and height of the box.

$$L_{reg}(\hat{t}, t) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i - \hat{t_i}) \tag{1.1}$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{1.2}$$

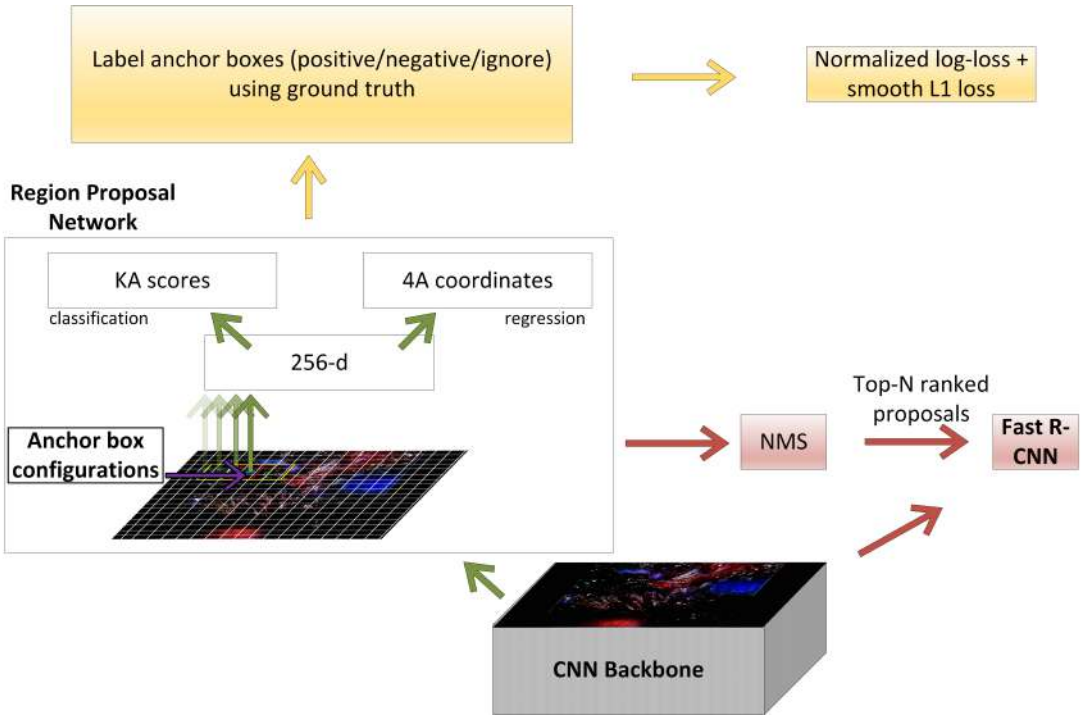
The training is done end-to-end excluding the region proposal generation.

1.2.1.3 Faster R-CNN

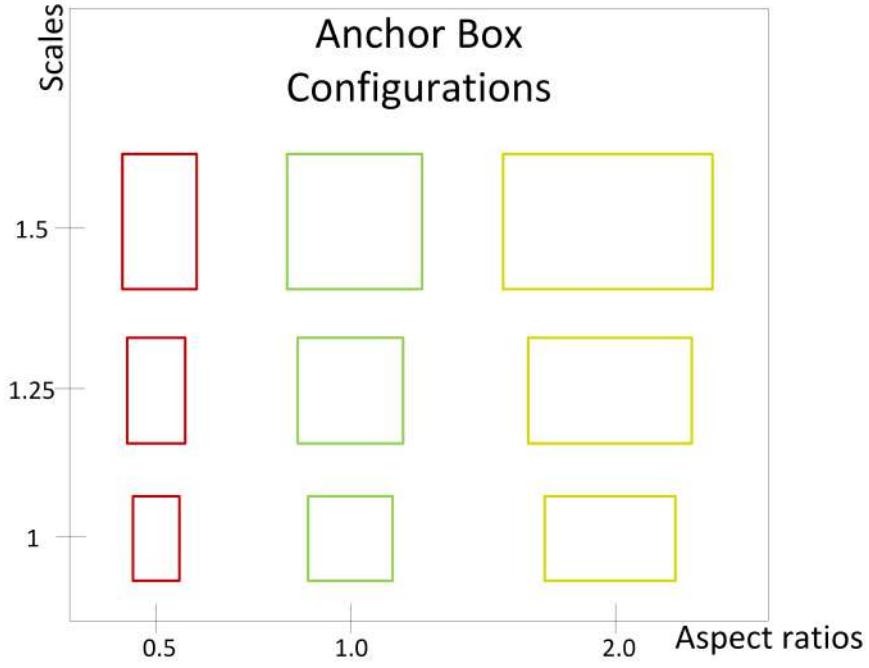
Finally, Faster R-CNNs [Ren et al. (2017)] use a region proposal network (RPN) to learn object proposals from the backbone feature maps which are then fed into the RoI pooling layer and later layers of a standard Fast R-CNN. The RPN learns object proposals by sliding a window of a fixed stride across the image where at each point anchors of a set of scales and aspect ratios are generated. The anchor box scale and aspect ratios are manually set and later rescaled based on the size of the featurized input image. The window corresponds to a shallow convolutional network where the output heads are a classification layer of size W x H x KA and a regression layer of size W x H x 4A. Here, A corresponds to the total number of default anchor configurations (9 configurations in the original work) and K corresponds to the number of classes. The anchors of different sizes serve to detect objects at different scales.

Figure 1.1 shows the structure of the RPN in a Faster R-CNN. The dark green arrows follow the sequence of region proposal generation. The yellow arrows represent the flow active during the training process of the RPN. The red arrows represent the sequence when region proposals are used for object detection using the Fast R-CNN model. As a high number of region proposals may cover the same ground truth, non-maximum suppression (NMS) with a threshold of 0.7 is used on their classification scores resulting in a significant reduction in proposal regions. Finally, the top-N ranked proposals are used for object detection.

The classification loss is often a simple cross-entropy loss and the regression loss is the smooth L1 loss [Fu et al. (2019)]. Each output plane of the RPN is only considered for



(a) Faster R-CNN



(b) Anchor Box Configurations

FIGURE 1.1: The Faster R-CNN architecture uses a common CNN backbone for region proposal generation in an RPN and object detection in Fast R-CNN. The shallow RPN network is passed on every coordinate of the featurized image map. Anchor boxes that cross the image boundary are ignored. The regions are then labeled with positive/negative for each class to depict foreground/background and ignored if they fall in the middle of the negative and positive threshold.

regression if the corresponding anchor has sufficient overlap with a nearby ground-truth box called the intersection-over-union threshold (IoU) or if a plane has the highest IoU with a ground-truth-box in a set of planes despite having an IoU lower than the threshold (set at 0.7 by the authors). This also holds true for classification with the addition that if the anchor has an IoU lower than a lower threshold (set at 0.3 by the authors) then the anchor is classified as the background class. All other anchors are ignored from the loss. The RPN is trained independently of the Fast R-CNN inside the Faster R-CNN architecture despite having a shared backbone. Thus, algorithms have been suggested by the authors to fine-tune the backbone during independent training.

Faster R-CNN with a strong backbone network has been shown to have the highest accuracy with benchmark datasets (PASCAL VOC 2007 [Everingham et al. (2007)], PASCAL VOC 2012 [Everingham et al. (2012)], Microsoft COCO [Lin et al. (2015)]) and are fastest of the two-stage detectors (17 fps with ZF net backbone [Zeiler and Fergus (2013)] and 5 fps with VGG-16 [Simonyan and Zisserman (2015)] as compared to 0.5 fps using selective search in place of RPN).

The surgical action detection architecture later proposed in this dissertation, albeit being a one-stage detector, shares components with the Faster R-CNN architecture (anchors generation and loss computation) and thus makes an understanding of RPNs in Faster R-CNN essential to an extent.

1.2.2 One-stage detectors

One-stage detectors save up on the time of bounding box detection by having a unified pipeline of region of interest/default box generation and box localization and classification. This provision allows them to detect objects at frame rates magnitudes higher than two-stage detectors making them suitable for real-time detection. Despite formerly having a lower accuracy than two-stage detectors, incremental updates to state-of-the-art one-stage detectors have given benchmark performance on standard object detection tasks.

1.2.2.1 YOLO

The YOLO [Redmon et al. (2015)] object detector consists of a deep CNN backbone and prediction heads that directly perform bounding box regression and classification. Object proposal generation is replaced by dividing the image into a fixed $S \times S$ grid where each grid cell consists of B bounding boxes. For each bounding box, the model regresses center coordinates, width, and height and computes confidence scores for objectness of those bounding boxes and K class probabilities. The authors use $B = 2$ on PASCAL VOC as compared to Faster R-CNN that uses 300 proposals. This limitation and a

unified network for localization and classification account for marginally faster testing times (mAP=63.4, FPS=45 on YOLO as compared to mAP=62.1, FPS=18 on ZF Faster R-CNN on VOC 2007). With a VGG-16 backbone, YOLO performs its best with mAP=66.4 but a lower FPS of 21. While more inaccurate, this is faster than VGG-16 Faster R-CNN (mAP=73.2, FPS=7).

1.2.2.2 YOLO9000 (v2)

The other two versions of YOLO are incremental updates, incorporating several network optimization techniques and some novel training methods. YOLO9000(v2) [Redmon and Farhadi (2016)], the first update to YOLO, uses anchor boxes for regression and classification similar to Faster R-CNN instead of directly regressing box variables. Anchor box prior width and height and grid cell height and width are used to predict bounding box coordinates. However, instead of manually setting anchor box scale and aspect ratios, k-mean clustering is used on the training set ground truths to determine the best anchor box priors with a distance measure determined by the IoU of the box and centroid. YOLO9000 also incorporates a number of optimizations such as batch normalization, pre-training at a higher resolution for better learning of box detection, incorporating a passthrough layer that outputs low resolution features from an earlier layer for incorporating fine-grained features, and multi-scale training for robustness at multiple input sizes. Multi-scale training works by resizing input images at every iteration and also changing the network size. Coupled with a new classification model, Darknet-19, and utilizing hierarchical classification using labels from WordNet [Miller et al. (1990)], YOLO9000 achieves a mAP of 78.6 on VOC2007.

1.2.2.3 YOLOv3

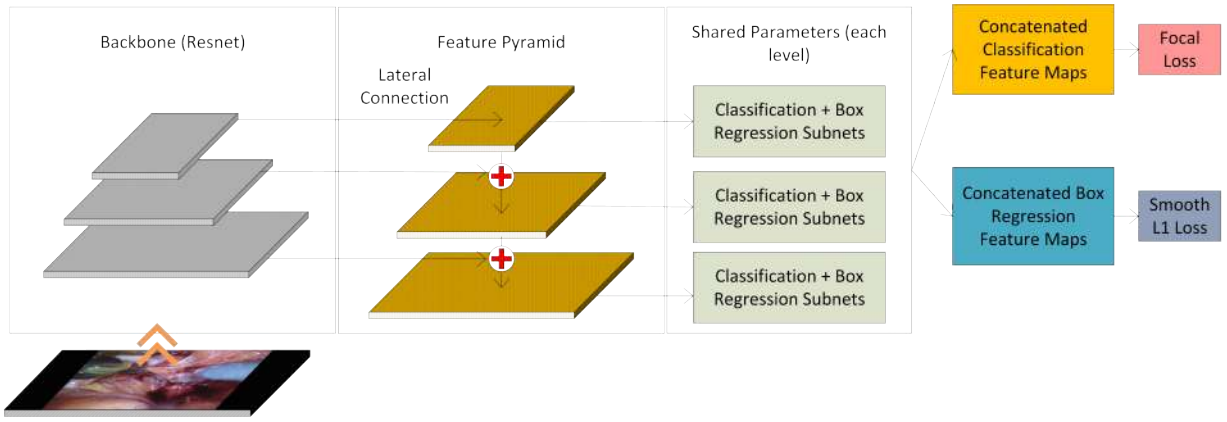
YOLOv3 offers minor adjustments such as using individual logistic classifiers instead of a softmax, multi-scale learning using an approach similar to feature pyramid networks (FPN) [Lin et al. (2016)] and a deeper CNN network, Darknet-53. YOLOv3 is fast and more accurate than SSD [Liu et al. (2015)] but less accurate than the deeper RetinaNet variants (ResNet-101 backbone) on the COCO dataset. It is still 3.8 times faster than RetinaNet thus making it the most suitable real-time one-stage detector.

1.2.2.4 RetinaNet

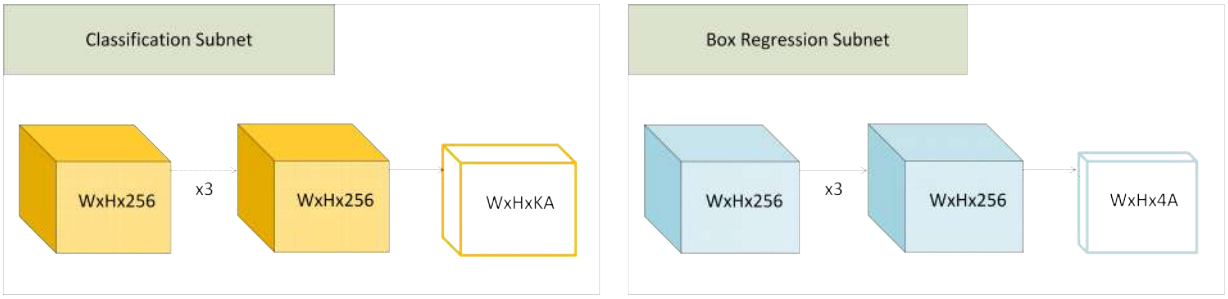
RetinaNets [Lin et al. (2017)] use FPNs and perform box prediction and classification using anchor boxes with manually set priors and a novel classification loss to cater to the heavy class imbalance in most detection tasks called the focal loss. They are among the most accurate one-stage detectors surpassing the best two-stage methods on most

tasks. The proposed solution is also a RetinaNet with hyper-parameters adapted to the SARAS-ESAD dataset [Singh Bawa et al. (2021)] and architectural modifications incorporating convolutional long short term memory (ConvLSTMs) [Shi et al. (2015)] to introduce a temporal context into the classification and box detection heads. Further temporal cues are also provided by a subnet that predicts the surgical phase as part of the multi-task loss.

A RetinaNet can be broken down into a deep CNN backbone that uses an FPN; classification, and box detection heads that concatenate multi-layer features from the FPN into box anchor features; and finally smooth L1 loss for box localization and a modified cross-entropy loss called the focal loss for classification.



(a) RetinaNet



(b) Classification and box regression subnets

FIGURE 1.2: RetinaNet architecture with an arbitrary ResNet backbone. Classification and box regression subnets consist of inputs coming in from the ResNet-FPN backbone. They are fully convolutional and output a tensor of channel size = KA for classification where K is the number of classes and A is the number of anchors and channel size = 4A for box regression corresponding to top-left x coordinate, top-left y coordinate, width and height of each offset anchor.

Feature Pyramid Network (FPN) Feature pyramid networks [Lin et al. (2016)] are a deep learning implementation of feature pyramids. Pyramid methods have been an essential tool to traditional object detection in computer vision [Adelson et al. (1984)] that largely consisted of doing object detection on multiple scales of the input image.

Multi-scale learning of such sort ensures robustness in the detection of objects of multiple sizes. Instead of doing forward passes on multiple input image sizes, FPNs utilize the natural pyramidal feature hierarchy of a deep ConvNet backbone. Each layer of the backbone carries features of varying spatial resolution with the top-most smaller resolution maps having semantically stronger features. While a technique such as SSD [Liu et al. (2015)] directly uses a set of layers from the backbone in a fully convolutional manner for prediction, FPNs build a top-down pathway of FCN layers with lateral connections from backbone layers forming a network. Prediction heads are then attached to these layers. This ensures that all levels of the pyramid are semantically strong including the coarse-grain featured high-resolution maps.

The original FPN Lin et al. (2016) uses final layers at the last 4 stages of the ResNet backbone: conv2, conv3, conv4, conv5 as lateral connections {C2, C3, C4, C5} feeding into pyramid layers {P2, P3, P4, P5} by first going through a 1x1 convolution operation followed by an addition operation with the upsampled pyramid layer above and finally a 3x3 convolution to reduce the aliasing caused by upsampling. The purpose of the 1x1 convolution is to get a fixed channel size; 256 in the original paper. Upsampling is done via interpolation using the nearest neighbor. However, in pursuit of speed, while maintaining accuracy, RetinaNet only uses lateral connections {C3, C4, C5} to create {P3, P4, P5}. P6 and P7 are additionally created where a 2-stride 3x3 convolution on C5 generates P6 and a 2-stride 3x3 convolution on P6 generates P7. Figure 1.3 gives an overview of the ResNet-FPN structure in RetinaNet.

Classification Subnet Outputs from each layer of the ResNet-FPN are input into a shallow FCN consisting of 4 convolutional layers where the number of filters is equal to the input channels, i.e., 256; See Figure 1.2(b). Each 3x3 convolution is followed by a ReLU activation. A classification head is appended at the end of the subnet via a 3x3 convolution with the number of filters = KA where K is the total number of classes and A is the number of anchor boxes. This corresponds to the confidence in the existence of each class at each anchor box. Sigmoidal activations are then applied to this head. The parameters of this subnet are shared among all levels of the ResNet-FPN.

Box Regression Subnet The box regression subnet is responsible for regressing the value corresponding to the offset between the default anchor box position and any nearby ground-truth box at each spatial location. The FCN subnet is the same in design as the classification subnet except for a linear box regression head appended at the end of channel size = 4A; See Figure 1.2(b). This corresponds to the offset in values of the top-left x and y coordinate and the width and height of each box with the nearest ground-truth box.

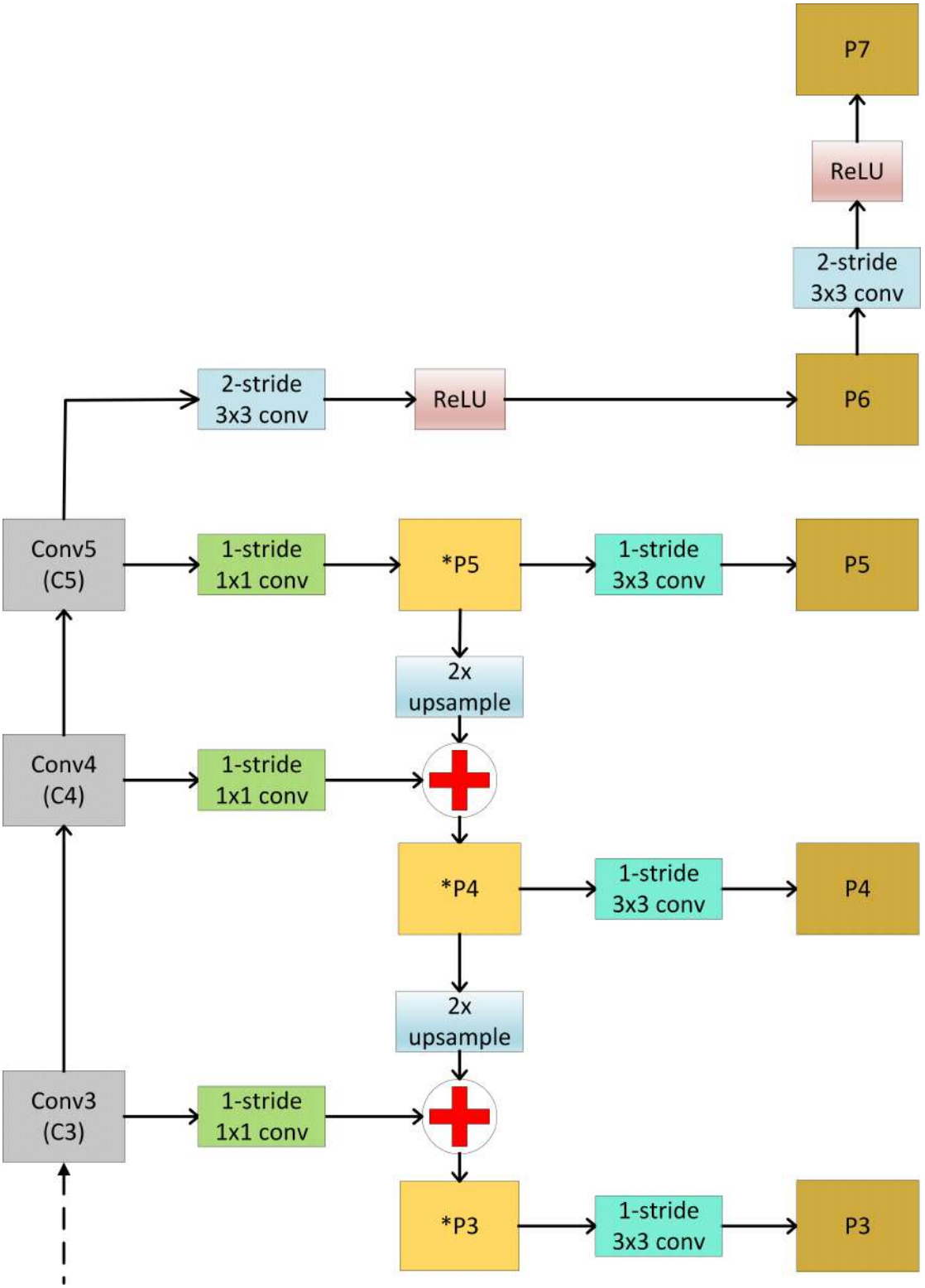


FIGURE 1.3: ResNet-FPN backbone in RetinaNet.

Anchor Boxes Anchor box generation and ground-truth matching and labeling are done similarly as in an RPN in a Faster R-CNN (see Section 1.2.1.3 and Figure 1.1(b)). However, using an FPN means that we have a set of features of decreasing size at each layer for which anchor boxes need to be generated. The steps for anchor box generation in RetinaNet is as follows (note that this is the same for anchor box generation in RPN with the exception that there is a reference size for each FPN output layer as opposed to there being only one output layer of the backbone feeding into the RPN):

1. Set anchor box priors (common aspect ratios, common scales, reference sizes[2], strides[3].)
2. For each reference size:
   (a) Rescale common scales by reference size.
   (b) Generate reference anchor box center coordinates, width, height for each aspect ratio/common scale configuration.
3. For each FPN output grid size (P3 to P7):
   (a) Get stride for grid size.
   (b) Calculate center coordinates of sliding window on grid at each stride.
   (c) Generate anchor box coordinates, width, height using reference box parameters at each sliding window center coordinate.

All the anchors at each layer are concatenated and labeled as positive, negative, and ignored based on a negative and positive threshold in the same manner as Faster R-CNN during training. The anchor labeling enables generating a mask of positive examples from the predicted locations for calculating smooth L1 loss (see Equation 1.1). Another mask is generated consisting of confidence scores (of classification) of both positive and negative examples (excluding the ignored examples) for focal loss calculation. During inference, non-maximum suppression (NMS) is performed to filter out the best-predicted boxes similar to how the best-proposed regions are filtered out from the RPN during the object detection stage in Faster R-CNN.

Focal Loss (FL) Previously, one-stage detectors had been less accurate than the best two-stage detectors. RetinaNet authors [Lin et al. (2017)] argued that such a discrepancy was due to the dominant majority of the background class in training in densely sampled object locations in one-stage detectors. This is opposed to the sparse object

[2]Each layer of the FPN output will have a distinct scaling factor where smaller FPN output layers will have a larger reference scaling factor. The authors set it from 32^2 to 512^2 for layers P3 to P7. This factor is further resized by the common scales shared across each layer.

[3]Anchor boxes are generated at each sliding window position. The stride determines the distance between sliding windows. Smaller FPN output layers have larger strides.

propositions in two-stage detectors. Thus they proposed a novel loss for classification that down-weights the loss of the high probability class (easy examples), enabling the low-probability foreground (hard examples) to have an increased impact in training. When a large number of easy negatives have a smaller loss, their computed gradients no longer overwhelm training. An important observation is that focal loss does not up-weight hard examples, thus avoiding outliers from impacting training as well. This loss has been shown to outperform previous attempts to deal with such class imbalance in one-stage detectors such as bootstrapping or hard exampling mining [Felzenszwalb et al. (2010), Shrivastava et al. (2016)].

The focal loss is a modification of the binary cross entropy loss (BCE) where a weighting factor $(1-p)^{\lambda}$ is multiplied to the binary cross entropy loss where $p \in [0,1]$ is the estimated probability of class $y = 1$ calculated by applying sigmoid activation $\sigma(s)$ to network output s. When $p >> 0.5$ and $\lambda >> 0$ the loss is down-weighted more strictly. An additional normalized balancing factor α is also used to specifically down-weight or up-weight positive/negative examples apriori. Focal loss for binary classification is given in Equation (1.3).

$$FL(\hat{y}, y) = -\boldsymbol{\alpha^*(\hat{y})} \cdot (1 - \mathbf{p(\hat{y}, y)})^{\lambda} \cdot \mathrm{BCE}(\hat{y}, y) \tag{1.3}$$

where

$$\boldsymbol{\alpha^*(\hat{y})} = \alpha \cdot \hat{y} + (1-\alpha) \cdot (1-\hat{y}) \tag{1.4}$$

$$\mathbf{p(\hat{y}, y)} = \hat{y} \cdot y + (1-\hat{y}) \cdot (1-y) \tag{1.5}$$

$$y = \sigma(s) \tag{1.6}$$

$$\mathrm{BCE}(\hat{y}, y) = \hat{y} \cdot \log(y) + (1-\hat{y}) \cdot \log(1-y) \tag{1.7}$$

For multi-class classification, Equation (1.3) can either be modified to use softmax activations and setting a balancing factor α for each class; or used directly with a One-Vs-Rest (OvR) solution. For our proposed solution, we use the latter approach.

1.3 Our Contributions

First, we extend the capability of a RetinaNet to capture spatiotemporal features by using a temporal subnet consisting of convolutional blocks and ConvLSTM blocks [Shi et al. (2015)] inside a shallow network. The objective is to show that providing self-learned temporal cues to a surgical action detection model can outperform RetinaNet, a spatial-only detector, when unshuffled and class-imbalanced data is presented to both models.

Second, we boost the depth and width of the temporal subnet to test the efficacy of learning better spatiotemporal features with a deeper temporal subnet. This consists of increasing the number of Conv-ConvLSTM layers as well as increasing the number of consecutive ConvLSTM blocks. The latter is motivated by the observation that in a standard long short term memory (LSTM) architecture [Hochreiter and Schmidhuber (1997)], stacked LSTMs perform better [Cui et al. (2020), Moniz and Krueger (2018), Graves et al. (2007)] due the network's capability to capture complex input temporal patterns.

Third, we implement truncated backpropagation through time (TBPTT) [Sutskever (2013)] to learn from long sequences of surgical footage frames; a task that would otherwise exceed the memory capacity when done with conventional backpropagation through time as the former does not require storing the entire sequence in a mini-batch. We explain the challenges faced while implementing TBPTT in Pytorch[4] 1.7 [Paszke et al. (2019)], and how we overcame those challenges. We also provide an algorithm listing the steps to implement TBPTT using a similar machine learning framework as Pytorch 1.7.

Fourth, we make the model more context-aware by introducing an additional phase recognition task. This is done by manually observing phases in surgical actions across timesteps. Each ground truth is then annotated with its ordinal phase. To learn phase features, a network of configurable depth consisting of dense layers is appended to the classification temporal subnet. Cross-entropy loss for the predicted phases is added to the existing classification and box regression loss.

We try to justify our reasoning for each improvement with experimental results and compare the predictive performance of all model designs.

[4] A python library used as a framework to create deep-learning models.

1.4 Related Work

There are several recognition tasks researchers have worked on for the surgical domain using several types of data. One of the earliest tasks worked on is the surgical phase[5] recognition task. Before the widespread adoption of CNNs as robust visual feature extractors, tool usage signals [Padoy et al. (2012), Forestier et al. (2013), Blum et al. (2010)] and surgical action triplets *(surgical tool, surgical action, target tissue)* [Katic et al. (2014), Forestier et al. (2015)] were being used to predict surgical phase. In a different context, some work was put into tool presence detection [Bouget et al. (2015)], tracking [Kranzfelder et al. (2013)] and pose estimation [Allan et al. (2015), Wang and Song (2020)]. [van Amsterdam et al. (2020)] address the surgical action recognition problem by creating granular surgical progress units consisting of surgical actions where each unit may occur more than once and two units may have overlapping actions. Then surgical action and progress are recognized in a multi-task manner using a network with bidirectional LSTMs with kinematics[6] features.

However, some of these approaches require specialist hardware or data that is difficult and more time-consuming to annotate. Due to the higher availability of visual surgical data in most ORs as well the existence of visual data in our dataset, the methods discussed will focus on recognition tasks using image data.

1.4.1 Surgical Phase Recognition

The surgical workflow follows a high-level sequential order, thus capturing temporal information has been one of the forefronts in surgical recognition tasks. [Twinanda et al. (2016)] propose *EndoNet*, an extension of AlexNet [Krizhevsky et al. (2012)], to perform tool presence recognition and phase recognition in a multi-task manner on a cholecystectomy (laparoscopic) dataset. This is achieved by appending a dense layer for tool presence recognition to the AlexNet and then both confidence scores of each tool presence and the forwarded image features (along with some handcrafted features) are input to a linear OvR SVM. Finally, a two-level Hierarchical Hidden Markov Model (HHMM) [Padoy et al. (2009)] is applied to the phase confidence scores obtained from the SVM. The inclusion of the HHMM has been shown to provide a ∼17-point boost in accuracy in addition to the performance gains brought by the rest of the model as compared to AlexNet. The temporal constraints imposed by the HHMM in phase estimation bring top-notch results however it does not form an integral part of EndoNet and thus the feature extraction process of EndoNet does not take temporal cues into account. On

[5] The surgical phase refers to a meaningful segment of work consisting of a set of surgical actions from the duration of a certain surgical procedure.

[6] Data captured by the robot-assistant in real-time that tracks the movement of patient side and surgeon side manipulators. Metrics for movement include cartesian positions, orientations, velocities, etc.

the other hand, our approach uses ConvLSTMs to learn temporal information from the sequential dataset and provides an architecture for optimized and efficient end-to-end learning.

Many approaches have been proposed that use some kind of a recurrent neural network structure with a CNN backbone to make temporal feature extraction a part of the network. CNN-LSTM/CNN-GRU (gated recurrent unit) structures have been of wide use for both laparoscopic surgical phase-detection tasks as well action detection in general to learn temporal context. Typically, optimizer steps in recurrent networks are not taken until the complete sequence has been input into the network. As a result the gradient of the loss w.r.t. the parameters of the model at time t = 1 will be dependent on the gradients at time t = T in a manner referred to as back-propagation through time (BPTT). This poses a problem for laparoscopic surgeries because each procedure may last for at least an hour, producing a large number of video frames. The gradient computation graphs of these video frames cannot be stored in standard modern GPUs all at once and thus some sort of workaround is done to address this issue. [Bodenstedt et al. (2017)] use a GRU with a CNN backbone to predict phase by training in an end-to-end manner on laparoscopic video sub-sequences. By passing the GRU's hidden state in-between sub-sequences, temporal connections can be maintained between batches. [Twinanda (2017)] optimizes on complete video sequences but makes up for the space complexity by training the CNN and LSTM in a two-step manner. However, it has been shown in previous works that end-to-end training yields the best performance on recurrent CNN architectures [Hajj et al. (2018)] for surgical video tasks.

[Yengera et al. (2018)] propose to do end-to-end training on the model *EndoN2N* using an approximated BPTT in addition to some preliminary fine-tuning done before the end-to-end training. The EndoN2N model consists of a CaffeNet [Jia et al. (2014)] backbone which is first fine-tuned separately for surgical phase prediction. The phase prediction output layer is then replaced with an LSTM and a new phase predictor head to form the final EndoN2N model. An additional surgical progress prediction (percentage of surgery completed) and remaining surgery duration (RSD) pretraining multi-task is also proposed for CNN-LSTM architectures like EndoN2N which has been shown to get higher accuracy than simple pretraining tasks such as ImageNet. The approximated BPTT is done by dividing the surgical footage into sub-sequences. A backward pass is done over each sub-sequence and the parameter update step is done once all sub-sequences have been forward/backward passed. With a similar architecture used for EndoLSTM, the end-of-end training of *EndoN2N* has been shown to have a ∼4-point boost in F1-score than the two-step approach of EndoLSTM. Since our proposed approach tackles a similar challenge using long laparoscopic footage albeit for the more challenging task of surgical action detection, we will use a similar approximated BPTT technique with some modifications made to the optimization step called the truncated backpropagation through time (TBPTT).

1.4.2 Surgical Action Recognition

Surgical action recognition is a relaxed version of the surgical action detection problem where confidence scores of surgical actions only need to be output per frame without the need for localization. [Zia et al. (2018)] provide a comparison of solutions for gesture recognition in robot-assisted radical prostatectomies (RARP); the surgical procedure we are tackling as well; consisting of off-the-shelf deep CNN models, a slightly modified Inception V3 [Szegedy et al. (2015)] network proposed as the model *RP-Net* and 3D convolutional networks. 3D CNNs [Ji et al. (2013)] use 5D data, i.e., Batch (N) $\times$ Timesteps (T) $\times$ Channels (C) $\times$ Height (H) $\times$ Width (W); in contrast to 4D data in standard 2D CNNs, i.e., N $\times$ C $\times$ H $\times$ W; where each input in a mini-batch consists of a stack of consecutive video frames. By using 4D filters (in contrast to 3D filters), 3D CNNs can capture spatiotemporal features from the dataset. The authors show that both the RP-Net and off-the-shelf 2D CNN models perform better than the 3D CNN. While our proposed approach also uses 5D data, we use a slightly different approach in the ResNet backbone called a *time-distributed* layer which flattens timesteps and batch size to one dimension and performs standard 2D convolutions.

[Funke et al. (2019)] use an inflated ResNet-18 (effectively a 3D ResNet-18) for surgical action recognition on the *JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS)* dataset [Gao et al. (2014)]. The dataset consists of R-MIS data (video, kinematics, etc) from elementary surgical tasks (suturing, knot-tying, needle-passing) done on a bench-top model. The 3D ResNet-18 outputs surgical actions for each stacked video frame and penalizes the loss of more recent frames higher than the older frames in the stack using a weighted cross-entropy loss. The total loss is the sum of the weighted loss of each video frame in the stack. While our RARP surgical action detection problem is similar, it is vastly more challenging as we use live RARP footage which frequently consists of occlusions brought by blood, smoke, and motion blur. It also contains more surgical actions (21 actions) than those captured in the JIGSAWS dataset (16 actions).

1.4.3 Recurrent Neural Networks

All recurrent neural network architectures are used to benefit from the temporal context created by past inputs to predict the next output. The Elman recurrent neural network (ERNN) [Elman (1990)] was among the first deep networks to store an activated (non-linear) hidden state for every timestep and use it to calculate the hidden state and output for the next timestep. The recursive function invocation involving hidden states from the previous timestep to calculate the output of the current timestep creates directed cycles in the computation graph and thus sets up the basis of backpropagation through time (BPTT).

1.4.3.1 Long Short Term Memory (LSTM)

LSTMs quickly overran ERNNs due to their robustness against the gradient vanishing problem [Pascanu et al. (2013)]. The gradient vanishing problem emerges due to the application of the chain rule to find the gradient of the loss w.r.t weight parameters and the flow of gradients of hidden state $h_{t'}$ w.r.t h_t to the first timestep. The flow of gradients means that the weight parameter θ is exponentiated multiple times and depending on if $\theta < 1$ or $\theta > 1$, the gradients will vanish or explode.

LSTMs solve the vanishing gradient problem by using a long-term memory called the cell state which is used to create the new hidden state (short term memory), for the current timestep. At each timestep, the input and hidden state determine what components are removed from the cell state and what new components are added. This removal and addition mechanism to the cell state is incorporated using multiple dense networks acting as gates. Such a mechanism allows for the avoidance of exponentiating θ while computing gradients and thus (to an extent) circumvents the gradient vanishing problem.

1.4.3.2 Convolutional Long Short Term Memory (ConvLSTM)

A ConvLSTM [Shi et al. (2015)] has convolutional layers in place of dense networks acting as gates. It shares the same number of gating functions but extends the capability of the LSTM to input 3D data (channels, height, width) across timesteps. The original ConvLSTM more explicitly extends the cell state, hidden state, and gate equations of [Graves (2014)] which builds upon a version of LSTM known as peephole-LSTM [Gers et al. (2003)]. The peephole-LSTM brings two innovations to the original LSTM: it uses connections from the constant error carousel (CEC), or the cell state, in the gate equations in place of the hidden state; and it omits the hidden state input to the cell input activation vector. [Gers et al. (2003)] argue that granting the gates access to the CEC in this manner allows the network to take the cell state into account even when the output gate is closed. As an advantage, the model seems to be able to learn time intervals in events spaced out by long time lags.

However, we would like to highlight that for simplicity we have only converted the original LSTM [Hochreiter and Schmidhuber (1997)] equations and mechanisms to their convolutional counterparts for our proposed solution. The equations for the corresponding input gate i_t, forget gate f_t, output gate o_t, cell input activation tensor $\tilde{\mathcal{C}}_t$, cell state $\mathcal{C}_t$ and hidden state $\mathcal{H}_t$ using input $\mathcal{X}_t$ are given in Equation (1.8). The $*$ operator represents a convolution operation and $\circ$ represents the Hadamard product.

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + \mathbf{b_i}) \\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + \mathbf{b_f}) \\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + \mathbf{b_o}) \\
\tilde{\mathcal{C}_t} &= \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + \mathbf{b_g}) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tilde{\mathcal{C}_t} \\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}
\tag{1.8}
$$

The need for convolutional layers in LSTMs stems from the fact that if an image feature map is flattened for input into a standard LSTM, the space complexity and computation costs will be too large for any modern system; whereas convolutional layers allow for parameter sharing thus solving the space issue and modern GPU implementations allow for fast computation.

It is important to note that part of our work is similar to [Li et al. (2018)] in that they use convolutional LSTMs in the classification and regression subnets of a RetinaNet as well. However, they fail to report how they use the outputs P3 to P7 of the FPN (see Section 1.2.2.4) into the classification and regression subnets. It is unknown if parameters are shared in the subnets for each FPN layer and consequently if the ConvLSTM hidden state is somehow passed between the FPN outputs of different sizes. Thus to the best of our knowledge, the work presented here is novel and contains sufficient explanation for reproduction as well.

Chapter 2

SARAS-Endoscopic Surgical Action Detection (ESAD) Dataset

The SARAS-ESAD dataset [Singh Bawa et al. (2021)] consists of monocular video footage of four RARP procedures recorded using the da Vinci Xi robotic system[1] at 1920×1080 resolution and 30 fps. In different parts of the operation, two lenses are used of angles 0° and 30° to ensure visibility.

The camera is passed through a trocar over the umbilicus from which live video is streamed and stored for an endoscopic view of the abdomen. Trocars are also used to pass in other surgical instruments which are controlled by the surgeon using the SSC where natural wrist movements are used to control the robotic arms. The primary objective of the operation is the resection of the prostate gland in patients with prostate cancer. On average each RARP procedure lasts 3 to 4 hours.

2.1 Specification

The dataset is divided into a training set consisting of RARP footage from two procedures, a validation set, and a test set. It consists of full video frames from each procedure and annotation is done of surgical actions using bounding boxes.

[1]The da Vinci Xi robot system is a robot assistant developed by Intuitive Surgical Inc. for R-MIS procedures.

2.1.1 Annotation

Both the names of the surgical actions and the drawing of bounding boxes have been decided and done by medical experts. The dataset consists of 21 different surgical actions. The actual annotation is done using Microsoft's open-source annotation tool called the *Virtual object Tagging Tool (VoTT)*. Each ground truth consists of two components: an integer representing the index of a specific surgical action, and a four-element set consisting of the center x-y coordinate and the width and height of the bounding box. Each video frame may consist of none, singular, or multiple labeled surgical actions. Furthermore, one frame may also have overlapping bounding boxes.

The overarching annotation methodology followed is that a frame is only annotated with a bounding box if the surgical tool was close enough to the appropriate organ to highlight an activity of interest. This saves up the learning model from focusing too much on learning the presence of surgical tools, enables the model to effectively differentiate between foreground and background, and creates an opportunity for the model to learn higher-level events of interest.

2.1.2 Size

Both the surgical procedures in the training set make up 22601 annotated frames consisting of a total of 28055 surgical actions. The validation data contains 4574 annotated frames with 7133 actions. The test set contains 6223 annotated frames and 11565 actions.

2.2 Analysis

To get a better view of some of the possible challenges of the surgical action detection task, it is worthwhile to gain some insights from the dataset. In the subsequent sections, we will give challenges posed by the dataset and some insight into the class distribution of surgical actions over time.

2.2.1 Class Imbalance

As evident in Section 2.1.2, each surgical footage in the dataset consists of a large number of frames and in turn have a large number of ground truths. However, there is also a significant amount of class imbalance. This is because given a certain surgery, certain fundamental tasks are performed multiple times while others are auxiliary actions that are performed as their need arises in the surgery based on the state of the tissue. In

addition to this, some actions are highly specialized and performed on a specific tissue at a certain stage in the surgery.

Figure 2.3 illustrates the variability in class representation in the training set. We can see in train set 1, *PullingTissue* and *CuttingTissue* dominate. These are fundamental actions in any surgery and are expected to be performed multiple times. *SuckingBlood* is an auxiliary action that is performed when needed based on the tissue state and has the third highest representation in the train set. Most of the other actions are specialized actions performed on specific tissue, e.g., *BladderAnastomosis* is a highly specialized surgically demanding task where the bladder neck is reconnected to the urethra and has an appreciable class representation. However, most of the specialized actions such as those involving the vasa deferentia have low representation.

Some of the class distribution trends are carried over to train set 2 where *PullingTissue*, *CuttingTissue*, *SuckingBlood* and *BladderAnastomosis* have high representation. However, some actions that were not performed at all in train set 1 are performed in train set 2, namely, *ClippingSeminalVesicle*, *CuttingSeminalVesicle* where the latter has a very high representation in train set 2 and the former has a very low representation. Furthermore, we can see in Figure 2.4 that class support in the validation set is similar to train set 1 but has all action classes present. *PullingBladderNeck* has higher support in the validation set than the entire training set combined which means the classification is likely to be problematic.

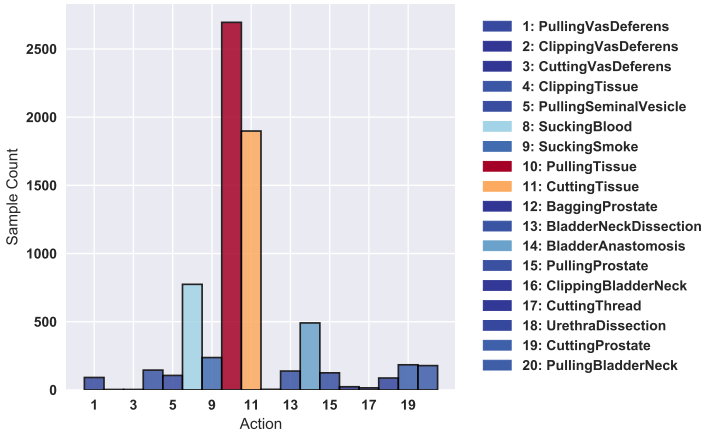


FIGURE 2.1: Train set 1

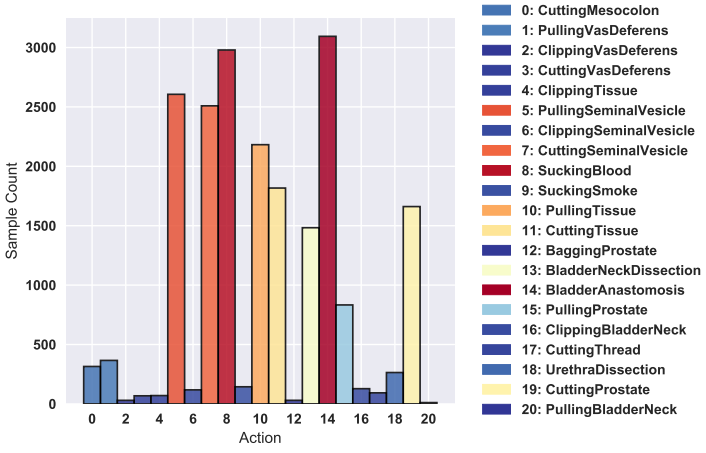


FIGURE 2.2: Train set 2

FIGURE 2.3: Class participation in the both surgical footage in the training set. Fundamental surgical maneuvers like "PullingTissue" and "CuttingTissue" have a high participation in both the sets. While primarily two actions dominate train set 1, train set 2 has higher variability in class representation.

From the imbalance in class support, we can deduce that unless an action has outstanding features, most of the lowly support actions will be difficult to detect.

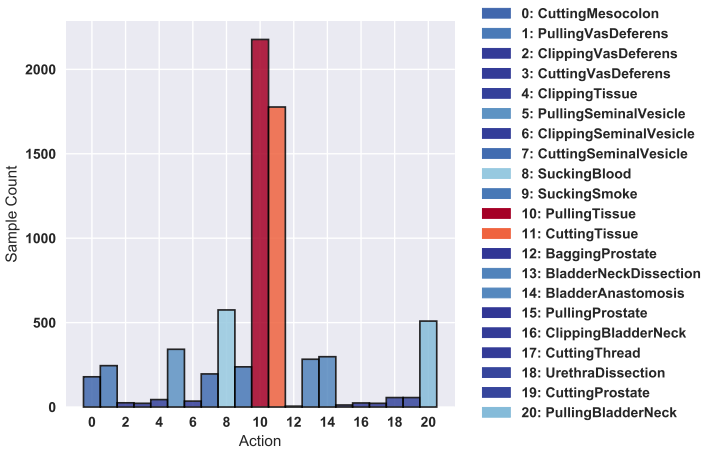


FIGURE 2.4: Class participation in the validation set. Class support is similar to train set 1 in Figure 2.3.

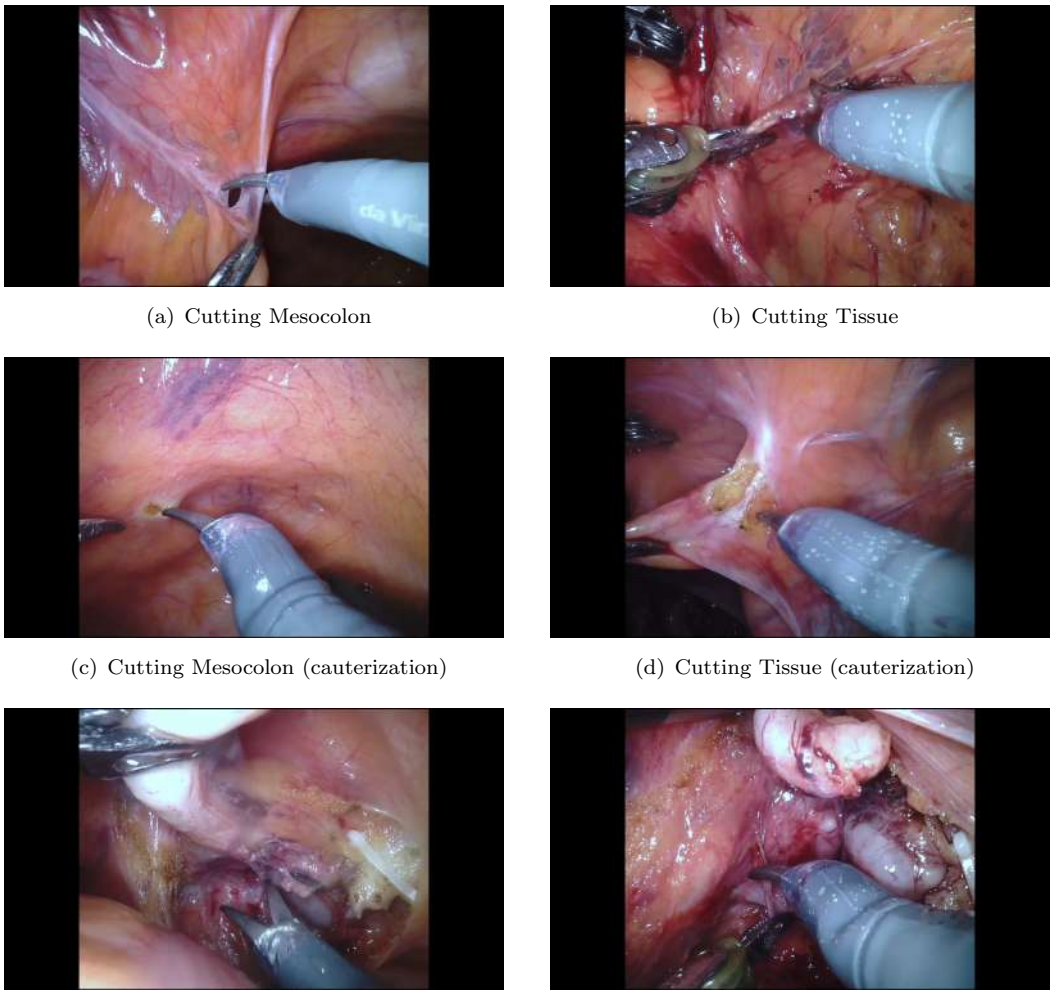
2.2.2 Poor Inter-Class Distinction

One of the biggest challenges in visual surgical recognition tasks is posed by the similarity in color, shape, and structure of the many tissue and organs targeted. Additionally, the dissecting tool in our dataset can be used on different kinds of tissue (general and specialized) and is also used as a cautery where dissection is done using heat transferred through the tool (thus producing surgical smoke). Thus, the presence of this tool does not necessarily imply only one surgical action. These challenges mean that with insufficient examples, the CNN backbone alone may not be able to extract distinguishing features for each class.

Figure 2.5 shows the similarity in physical features of named and unnamed tissue. Taking into account the corresponding sub-figures in the left and right column, it is difficult to identify a distinguishing feature between the named and unnamed tissue. Further, as previously stated, equally problematic is that the same surgical tool is used for three different actions as well (cutting tissue with and without cauterization count as one action). Thus the model will benefit very little from learning tool presence alone as the majority class will be classified with greater confidence only. This means that relying purely on spatial features for surgical action classification will require a complex convolutional model.

2.2.3 Aberrations

In the different parts of the procedure, the image may get aberrated due to the presence of blood, motion blur or surgical smoke as shown in Figure 2.6. Excessive blood may make it further difficult to discern which particular tissue is being targeted. Surgical smoke caused by cauterization may completely occlude the image. However, the presence of such occlusions are usually followed by surgical actions to counteract their effect such



(a) Cutting Mesocolon

(b) Cutting Tissue

(c) Cutting Mesocolon (cauterization)

(d) Cutting Tissue (cauterization)

(e) Cutting Seminal Vesicle

(f) Cutting Tissue

FIGURE 2.5: Grid showing a poor inter-class distinction between the left and right columns. Overall, it is difficult to discern if the cutting operation is being performed on a named tissue or a general tissue. We can also see that the same dissecting tool is used for cauterization as well.

as *SuckingBlood* and *SuckingSmoke*, limiting their impact in training. With that said, the *SuckingBlood* action is usually only performed when there is a pool of blood involved and thus bloody tissue alone is sometimes left uncleaned. Such aberrations pose further problems for the spatial feature extractor.

2.2.4 Variability in Patient Anatomy

While all human beings usually share the same organs, their appearance can be largely different based on many factors including fatty tissue. This is evident in the dataset as well because the amount of yellow fatty tissue present is not homogenous in the three different surgical footage. Furthermore, different patient anatomies mean that some procedures may take longer to reach target organs and may face different aberrations

(a) Blood (b) Motion Blur (c) Smoke

FIGURE 2.6: Three different kinds of aberrations found in the dataset. While blood and smoke are often caused by specific actions such as dissecting and cauterizing operations, motion blur is caused by the movement of the camera and cannot be easily associated with a certain preceding surgical action.

along the way. This is also the reason behind the differences in class support between the surgical footage in the dataset.

2.2.5 Class Distribution Over Time

While the exact order of actions may vary across surgical operations of the same procedure done on different patients; all do follow a similar higher-level order. The surgery consists of several objectives to be achieved and the way to achieve those objectives may be governed by a set of actions frequently occurring together across surgical operations.

Similarly, RARP procedures also have such higher-level objectives. Without the opinion of a medical expert, it will be harmful to declare those objectives in this report. Regardless, it is worthwhile to study the distribution of surgical actions over time to deduce our own set of objectives to gain a better view of the dataset. The end-use of this extensive study will be made apparent in the proposed solution when such "artificial" surgical phases are introduced in training.

Figure 2.9 and Figure 2.10 show class distribution over time in the training and validation set respectively. We can gain an appreciation over the variability in the length of procedure caused by variability in patient anatomy and other incidental causes. Train set 2 takes the highest amount of time, yet all three procedures roughly contain the same set of actions occurring together. This similarity is due to the same higher-level objectives and the nearness of particular organ tissue with each other.

This pattern creates a significant advantage for our highly imbalanced and aberrated learning problem. By incorporating training with temporal cues that group a set of surgical actions together over time, we can increase the confidence of the model in predicting a minority class that occurs alongside a majority class. Building on top of this, we can also increase the impact of minority classes in loss calculation by understanding that a class that should not occur in a certain phase should be punished more severely. In the proposed solution, such temporal cues will be made use of using a ConvLSTM model that learns these action groupings. In addition to this, we will use a multi-task

surgical phase recognition approach in conjunction with surgical action detection to severely punish incorrect classifications of out-of-phase surgical actions.

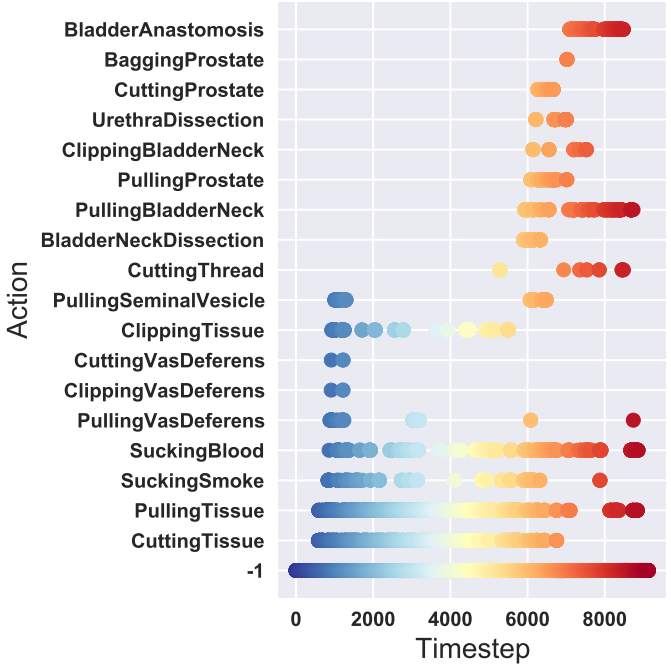


FIGURE 2.7: Train set 1

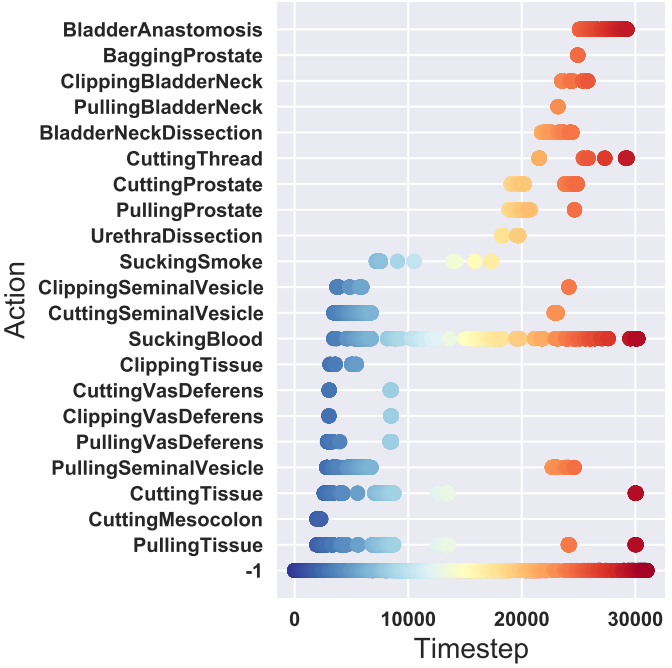


FIGURE 2.8: Train set 2

FIGURE 2.9: Scatter plot showing class distribution over time in the training set. We can see most of the frames contain no actions (plot corresponding to -1). We can also make out clusters from a set of actions that tend to occur together such as those involving the vasa deferentia or those involving the bladder and prostate.

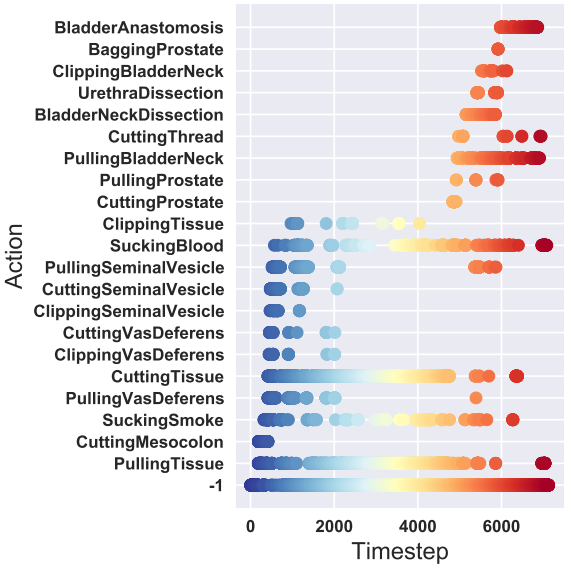


FIGURE 2.10: Scatter plot showing class distribution over time in the validation set. We can roughly observe the same set of actions occurring together as those in the training set.

Chapter 3

Proposed Solution - Temporal RetinaNet

As highlighted in Chapter 2 the problems posed by the dataset introduce difficulties to the learning process. CNN models relying on spatial data alone are less likely to learn sufficient distinguishing features for classification. To this effect, we propose to use a set of temporal subnets in place of the classification and box regression subnets in the RetinaNet (see Section 1.2.2.4). We call this modification to the RetinaNet, *TempRetinaNet*. The set of subnets consists of a temporal subnet for each FPN output layer P3 to P7. There is an individual set of temporal subnets for classification and box regression each. The code has been made publicly available[1].

3.0.1 Acknowledgements

We built upon the basic RetinaNet with backpropagation implementation by Bawa[2]. We heavily modified the data-loading pipeline to load full homogeneous mini-batches (more on this later), the RetinaNet architecture itself, the loss calculation, and introduced truncated backpropagation through time. We also improved the ConvLSTM implementation of Palazzi[3] to store state across mini-batches.

3.1 Architecture

Though the model appears intuitive at first glance, a number of design decisions have been made on purpose to aid learning of endoscopic image sequences using a special

[1] https://github.com/uzborg950/Surgical-Action-Detection-Using-Deep-Learning-Impl
[2] https://github.com/Viveksbawa/SARAS-ESAD-baseline
[3] https://github.com/ndrplz/ConvLSTM_pytorch

recurrent network and adhere to the computational budget. Figure 3.1 details the temporal subnets used in TempRetinaNet. Outputs from the FPN of a standard RetinaNet are passed through temporal subnets maintaining the channel size at 256 and finally reducing the channel size to KA or 4A at the head to store confidence scores or box regression coordinates at each anchor box. Each temporal layer consists of a sequence of ConvLSTM and Conv2d pairs with batch normalization and ReLU activation applied after every Conv2d layer. "Same" padding is applied at all blocks with single strides. This is because depending on the input size, the smallest spatial resolution output of the FPN may deteriorate learning of spatiotemporal features with further downscaling, thus all convolutions have been kept to maintain output size.

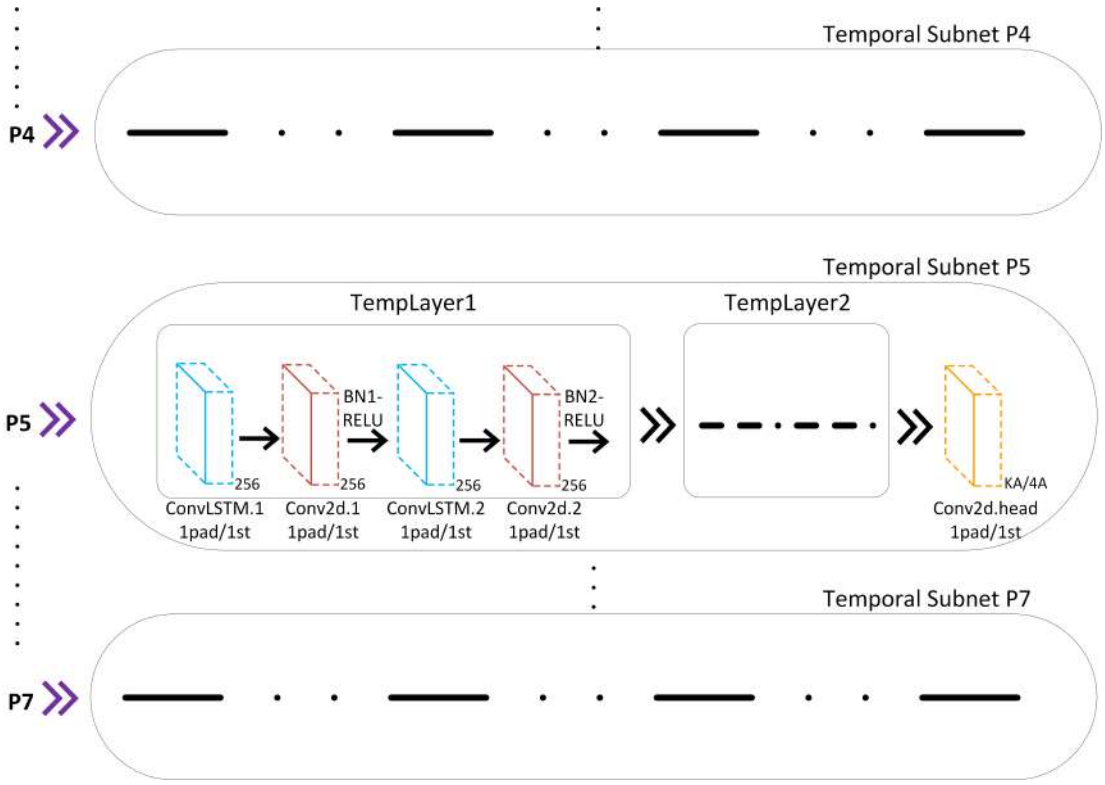


FIGURE 3.1: The FPN outputs P3 to P7 are passed through temporal subnets to learn from temporal cues at each scale. The spatial size of convolutions is kept the same throughout while the depth is reduced at the head to reflect confidence, KA, (for classification) or box coordinates, 4A, (for box regression) at each anchor point.

The number of temporal layers are configurable and can be increased to better learn spatiotemporal associations but due to computational and space limitations, we have only been able to experiment with two layers. We would also like to argue that the model would have benefited further with stacked ConvLSTM blocks as originally suggested in [Shi et al. (2015)], however the additional space requirements meant that we could not effectively run our truncated backpropagation through time (more on this later) with an appreciable sequence length. Thus single ConvLSTM blocks have been used.

Temporal subnet outputs P3 to P7 are then concatenated for anchor box matching, focal loss and smooth L1-loss calculation just like RetinaNet.

While the temporal subnet is largely motivated by the associations between sequence of surgical actions and time primarily, we have also used a temporal subnet for box regression. In our tests, we found that using the regular box regression subnet with the classification temporal subnet yielded slightly worse localization results, thus we kept the box regression temporal subnet as well. We hypothesize that this might be because of two reasons: the temporal subnet makes the model deeper and thus the increased non-linearities allow the model to learn more complex spatial relationships, and most consecutive actions are performed near to each other in spatial distance thus the temporal subnet introduces a starting reference point at each timestep.

Finally, due to space limitations, we only experimented with ResNet-18. However, testing the vanilla RetinaNet with larger ResNet backbones showed increased classification and localization performance. Thus we expect a better performance of TempRetinaNet with a deeper ResNet backbone as well.

3.1.1 State Management

State management is an important aspect of any recurrent neural architecture. In a ConvLSTM, state at timestep t is maintained via cell state $\mathcal{C}_t$ and the hidden state $\mathcal{H}_t$ where the latter is the output of the ConvLSTM at the previous timestep (see Equation (1.8)). At each timestep, $\mathcal{C}_t$ is updated using input $\mathcal{X}_t$, the previous hidden state $\mathcal{H}_{t-1}$ and the previous cell state $\mathcal{C}_{t-1}$. The new hidden state/output $\mathcal{H}_t$ is then found using the output gate, o_t and cell state $\mathcal{C}_t$.

In a standard LSTM implementation, the entire sequence is stored in one mini-batch. This means at each timestep, the state from the previous timestep is passed forward until the entire mini-batch has been passed to the LSTM. Then, the state is wiped and a tensor of zeros is commonly used as the initial state for the next mini-batch. However, this strategy is infeasible for our dataset as one sequence effectively counts as the entire surgical footage and this cannot be stored in a GPU as a mini-batch. Thus we pass the states at the last timestep of the previous mini-batch to the first timestep of the next mini-batch. These states are then reset once one particular footage has been fully passed.

To gain an appreciation of the space costs of these states, we have tallied the total number of states at a given timestep in Table 3.1.

As the state in each temporal subnet is a different spatial resolution, we could not use the same temporal subnet for all the FPN outputs in a way similar to the parameter sharing done in the classification and regression subnets in RetinaNet. All of this contributes to the added space complexity of the architecture.

Component	Quantity	Cumulative States
ConvLSTM1	1	2 ($\mathcal{H}_t$ and $\mathcal{C}_t$)
ConvLSTM2	1	$2 + 2 = 4$
Temporal Layers	2	$2 \times 4 = 8$
Temporal Subnets	5	$5 \times 8 = 40$

TABLE 3.1: Number of (spatial) states maintained at each timestep.

3.1.2 Time Distributed Layers

Using a recurrent neural architecture means that we run the backpropagation through time (truncated) once a set number of frame sequences in across mini-batches have been passed through the model. This means we need to ensure that the same weight and bias parameters in the ResNet-18 backbone are applied to the input sequences. As a conventional ResNet expects individual spatial inputs only, we must modify the way we feed our temporal slices of input. We do this by flattening the temporal dimension and the batch size dimension. In fact, this is done for all such convolutional or dense layers (when applicable) in the network when temporal input is fed into a block that conventionally expects static input.

While this is a trivial aspect, it is necessary to gain an overall understanding of how 2D convolutional layers are modified to use temporal inputs without the need for 3D convolutional layers. This was preferred over using 3D convolutions because the purpose of this particular work is to gauge the efficacy of ConvLSTMs in capturing spatiotemporal cues in laparoscopic footage in various settings. Thus the ResNet backbone was to be reused as much as possible without the need for further enhancement. Hence, the time distributed layer[4] approach was used.

3.1.3 Loss Calculation

We concatenate outputs from all timesteps in a mini-batch of the temporal subnets and create one big flattened batch for loss calculation. Thus in one mini-batch, the loss is being calculated for each timestep and all scales (of the FPN). The same losses are used as mentioned in Section 1.2.2.4 (Equation (1.2) and Equation (1.3)) for a conventional RetinaNet. We found weighing parameter $\alpha = 0.25$ and focusing parameter $\lambda = 2.0$ to work best in the focal loss for our training configuration.

[4]The word is taken from the concept of time-distributed layers in Keras. Our implementation is effectively just a wrapper for partially flattening the temporal input.

3.1.4 Multi-Task Surgical Phase Recognition

In Section 2.2 we presented a number of challenges posed by the SARAS-ESAD dataset that suggested that in many scenarios the model may favor the majority class. While the ConvLSTM architecture is expected to learn long-distance relationships in order of actions, the strong imbalance in classes, presence of occlusions, and the unequal distribution of classes over time may still prove cumbersome for the model to learn such temporal cues. For these reasons, we propose to extend TempRetinaNet to predict the surgical phase as well in a multi-task manner. In this way, we introduce stronger temporal cues that force the model into grouping a set of surgical actions together and basing confidence in classification on the likelihood of the existence of a certain surgical action in a surgical phase.

3.1.4.1 Surgical Phase Annotation

While the SARAS-ESAD dataset does not have ground-truths for the surgical phase included, we showed a similarity in the higher-level order of surgical actions across the three surgical footage in Section 2.2.5. This means that even without the presence of a medical expert, we can create our surgical phases employing observation. While ensuring a good compromise in granularity, we identified four surgical phases in each surgical footage.

The methodology followed was to observe when work on a particular named tissue was started and ended. All actions occurring during that time were considered to belong to the same phase. Some actions start and end before actions on another named tissue had ended. E.g., surgical actions on the seminal vesicle start and end before actions on the vas deferens had ended. In all such cases, such actions are said to belong to the same phase as well. We also ensure that same phase actions occur without a significant gap between each other. The gap was also determined empirically by observing for how long tissues were worked upon in particular surgical footage. Figure 3.2 illustrates how surgical phases were annotated in the SARAS-ESAD dataset.

3.1.4.2 Phase Model

The extension to the TempRetinaNet to make it recognize phase as well consists of a two-layered dense network that takes concatenated feature outputs from the classification subnet. The motivation here lies in the fact that the feature outputs at this point consist of confidence scores for each action class at each anchor box, and such scores can help decide the current phase of the procedure. At the same time, mistakes made in the predicted phase will modify learnable parameters in the classification temporal subnet to boost confidence in those actions that lie in the true phase.

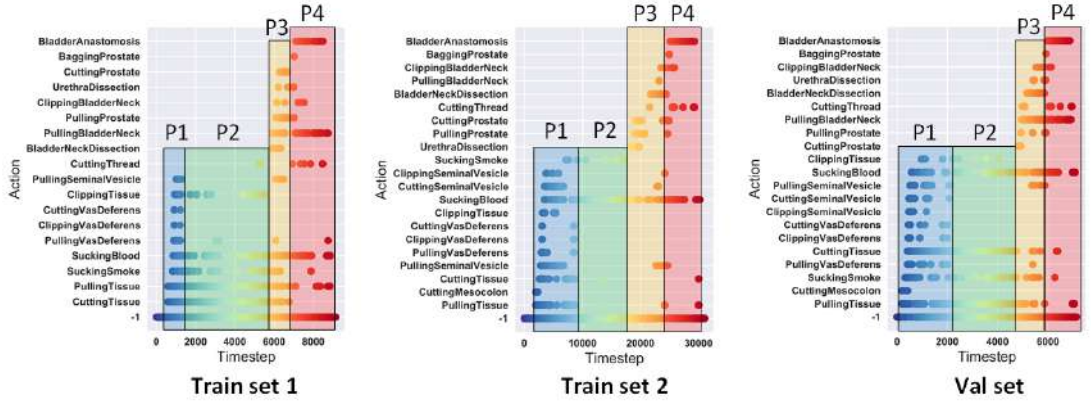


FIGURE 3.2: Surgical phases are annotated empirically by marking the starting and ending timestep when a particular named tissue is worked on. For e.g., Phase *P1* encapsulates surgical actions performed on the vas deferens and seminal vesicle; *P2* serves as an intermediary phase between *P1* and *P3*; *P3* encapsulates surgical actions performed on and around the bladder before anastomosis; *P4* encapsulates all surgical actions after bladder anastomosis till the end of surgery.

Figure 3.9 shows the changes made to TempRetinaNet to incorporate surgical phase prediction. The feature maps from each classification temporal subnet are concatenated and fed into a dense network containing a configurable number of hidden layers. We tested with a depth of 2 and 4 and found the deeper dense network to fit the training data better as expected. The quantity N* is the product of the batch size and number of timesteps in one temporal slice; K is the number of action classes; P is the number of phases (four in our case). Batch normalization is used to prevent numerical overflow and ReLU activation is used to introduce non-linearity. Cross-entropy loss is used to find the error in the confidence in the predicted phase. The loss is then added to the cumulative loss consisting of classification loss and box regression loss. The backpropagation step is then run on this final cumulative loss.

3.2 Training

A number of training approaches were tested to train the TempRetinaNet model. Challenges ranged from how to perform backpropagation through time when the entire surgical footage cannot be passed at once in one mini-batch; how to deal with datasets of unequal sizes; what to do with incomplete mini-batches and classical problems such as how to adapt the learning rate. A number of design decisions for such challenges are given here.

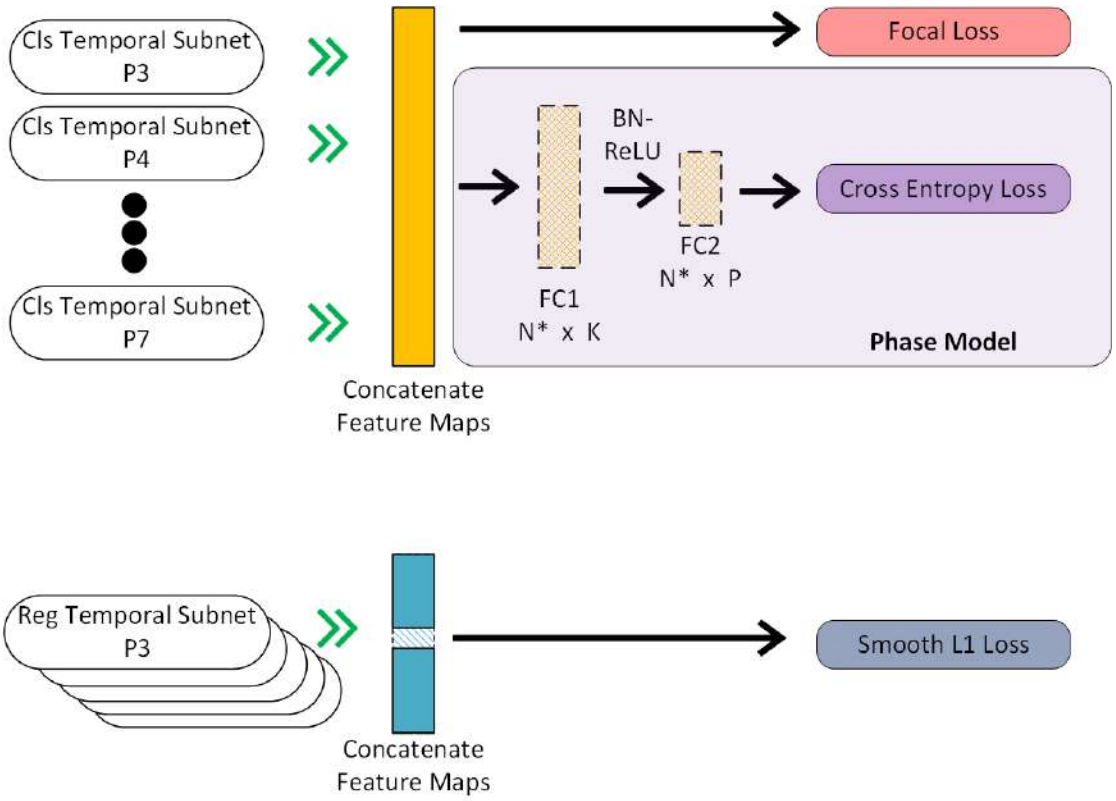


FIGURE 3.3: The phase model is a small extension to the TempRetinaNet architecture. Concatenated feature outputs from the classification temporal subnet are fed into a dense network responsible for predicting phase. The depth of this dense network is configurable. The batch size and timesteps are flattened to a quantity N*. The first layer consists of K=22 hidden units while the output layer consists of P=4 output units.

3.2.1 Truncated Back Propagation Through Time (TBPTT)

Truncated backpropagation through time (TBPTT) [Sutskever (2013)] is a solution to the problem of having high volume temporal inputs that cannot be stored in modern computational hardware. Conventional BPTT unrolls the computation graph of a recurrent model up to the first timestep and uses gradient information from the next timestep in the previous timestep. This means that to find gradients at the first timestep, the model will first need to pass the very last timestep while keeping the entire computation graph in memory. For a dataset with around 5000 frames minimum, this is infeasible and in our dataset, the longest footage is around 14200 frames counting only those frames with non-empty ground truths.

Concept Truncated backpropagation through time limits how far back the computation graph is unrolled and only finds gradients up to a set number of timesteps, k2. Before it performs the backpropagation step, it first performs a set number of forward passes, k1, on mini-batches to speed up training. In theory, the larger the k2, the more accurate the truncated backpropagation through time due to a greater knowledge of the

history but a very large k2 can cause vanishing gradients. In this manner, gradients of k2 states w.r.t the parameters are accumulated after every k1 timesteps. The parameter optimization step could be performed once all mini-batches of surgical footage have been forward-passed or after every k1 timesteps. The former approach is likely to yield more stability in learning but will be highly time-consuming. With this addition, the total number of states maintained in TempRetinaNet at each timestep per set of temporal nets increases to k2 $\times$ 40 in Table 3.1.

To illustrate the idea of TBPTT, Figure 3.4 shows the difference between BPTT and TBPTT. Here, k1 and k2 have different values than our experiments to illustrate a possible configuration of our implementation. In BPTT, all input sequences S are independent of each other while in TBPTT all input sequences shown belong to the same footage and are in order. In the figure, L_s represents the mean loss of the mini-batch s. Note that only the loss after k1 timesteps will be backpropagated.

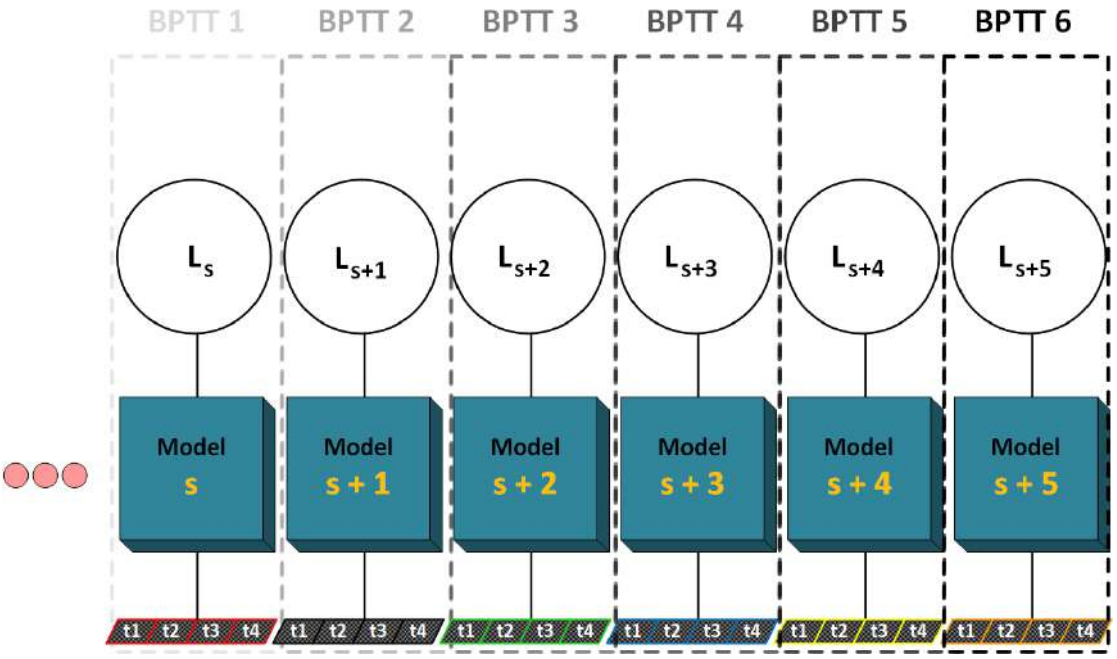
3.2.1.1 Implementation

We implemented TBPTT using PyTorch 1.7, thus the implementation takes into consideration how PyTorch maintains the computation graph of tensors. Algorithm 1 shows the TBPTT algorithm for one epoch. States (of the temporal subnets) are reset with zeros at the start of each surgical footage. The technical challenge is to isolate the computation graph of a state so that in the next timestep when this state is used as input, a new computation graph is generated. This is because we want to keep the existing computation graph untouched to be later used while backpropagating at a given timestep using gradients at the next timestep. Following the chain rule, we want to pass the gradients of the states at the next timestep to the gradient calculation step at the previous timestep.

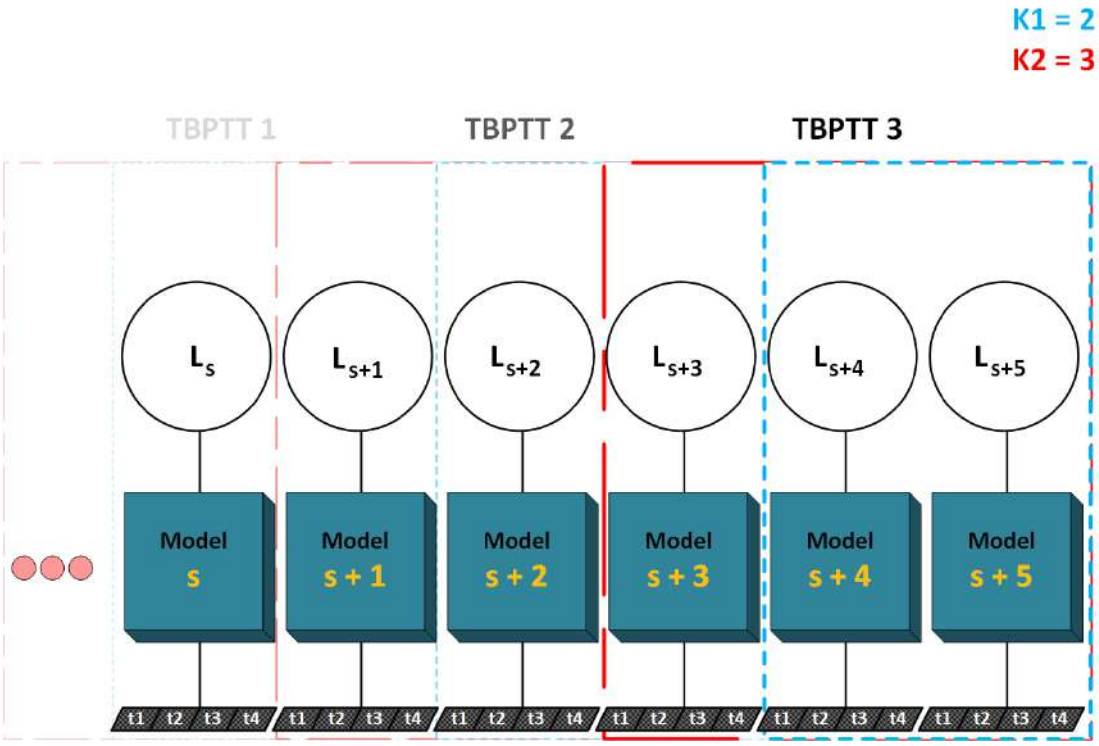
Thus, a copy of the state at the previous timestep is made without copying its computation graph as well (line 7 in Algorithm 1). This copy is then used at the current timestep's forward pass and a new computation graph is generated for the copied state. This process is repeated for k1 forward passes and then the loss is backpropagated, generating gradients for the new state and the copied state. The gradient of the copied state is then used as a seed while backpropagating back to the first state in $\mathcal{S}$.

Gradients of the loss and states w.r.t weight parameters are accumulated until the entire surgical footage is forward-passed. The optimization step is run at the end, after which the states are again reset.

Multiple Backpropagations Per Iteration Note in Algorithm 1 line 13 and line 21, backpropagation is run multiple times. The first backpropagation step flows the gradient w.r.t model weights of the current loss l_i backwards on the computation graph



(a) Backpropagation through time



(b) Truncated backpropagation through time

FIGURE 3.4: Backpropagation through time uses independent input sequences S and in a standard setting, each mini-batch forward pass is followed by a backpropagation step and an optimization step. While in truncated backpropagation through time, the state is passed between mini-batches till k1 forward-passes. Then, backpropagation is run for the previous k2 timesteps.

and uses a seed gradient $\frac{\delta l_i}{\delta l_i} = 1$. This is the standard *machine learning* backpropagation step. The remaining backpropagation steps flow the gradient of the next iteration to the previous state, i.e., it uses the gradient at the next iteration as a seed. This process is repeated back to the first k2th state.

Problem We set k1=1 and k2=5, which were the best parameters accommodated by our testing environment. This makes TBPTT very slow to train because an optimization step is not run until the entire surgical footage consisting of around 4600 to 14200 frames is passed. This takes a long time ($\sim$ 2 hours per optimization step) when a small total batch size (N*) is used, which in our case is 4. For our budget, this is infeasible.

We mentioned previously that to speed up training, the optimization step can be run after every k1 timesteps. However, PyTorch 1.7 makes doing this complicated. PyTorch 1.7 does its optimization step of the weight tensors in-place which means that the computation graph of the states in $\mathcal{S}$ becomes invalid after each weight update. We would need to somehow keep model weight values consistent with that of the computation graph of the states.

Improvement As mentioned in the previous section, we need to update the computation graph of the states so that it is consistent with the current model weight values after every optimization step. We solve this problem by repeating past k2 forward-passes and replacing the existing state set $\mathcal{S}$ with the new set formed by the forward passes. By doing this, the computation graph of the new states stores the current model weight values. This process is repeated after each optimization step.

Trade-off From an implementation point of view, we bear additional space complexity as we store the past k2 images and ground truths to repeat forward passes later. With our budget, we were able to accommodate this improvement by lowering the value of k2 from 5 to 3. With stronger hardware, we would like to experiment with large values of k2 in the 100s as well as a large total batch size N*. We also bear additional time complexity as now each iteration contains k2 + 1 forward passes. In our experiments, we saw that it took roughly 4 seconds per iteration and 5.2 hours per epoch as compared to in BPTT or BP where it took roughly 1.5 hours per epoch.

3.2.2 Data Loading

Using a model that keeps the state for each surgical footage passed means that we need to ensure that all frames in one mini-batch belong to the same surgical footage. This needs to be addressed explicitly as the total batch size, N*, may not fully divide the surgical footage to ensure this homogeneity in the mini-batches. To address this, we

Algorithm 1 Truncated backpropagation through time for one epoch.

Require: $\mathcal{F}$ mini-batches of all surgical footage

Initialize timestep counter i $\leftarrow 0$
Initialize state s_0 with zeros
Push tuple $(\emptyset, s_0)$ to list $\mathcal{S}$ ▷ Stores "previous ($\emptyset$)" and "current (s_0)" states.
while $\mathcal{F}$ contains mini-batches **do**
    $f_i \leftarrow$ get i$^{\text{th}}$ mini-batch from $\mathcal{F}$
    $\mathcal{T}_{i-1} \leftarrow$ get last state tuple in $\mathcal{S}$
    $s_{i-1} \leftarrow$ get second element of $\mathcal{T}_{i-1}$ **without** previous computation graph
    loss (l_i), new state $(s_i) \leftarrow$ **Forward-pass**(f_i, s_{i-1})
    $\mathcal{T}_i \leftarrow$ create tuple (s_{i-1}, s_i) ▷ The computation graph of s_{i-1} only contains functions from the current timestep.
    Push $\mathcal{T}_i$ to $\mathcal{S}$
    Keep the latest $k2$ state tuples in $\mathcal{S}$, remove rest
    if current timestep i divides $k1$ **then**
        Backpropagate loss l_i with seed gradient = 1
        $j \leftarrow k2 - 1$
        while $j > 0$ **do**
            if tuple at index $j-1$ exists in $\mathcal{S}$ **then**
                $\mathcal{T}_{j-1} \leftarrow$ get tuple before the tuple at index j in $\mathcal{S}$
                $s_{j-1} \leftarrow$ get second element of $\mathcal{T}_{j-1}$ **with** computation graph
                $\mathcal{T}_j \leftarrow$ get tuple at index j in $\mathcal{S}$
                $\nabla s \leftarrow$ get gradient of first element in $\mathcal{T}_j$
                Backpropagate data in s_{j-1} with seed gradient = ∇s ▷ uses the gradient at the next timestep to calculate gradients at the previous timestep.
                Decrement j
            end if
        end while
        if One surgical footage completely passed **then**
            Optimize weights
            Empty $\mathcal{S}$
            Initialize state s_0 with zeros
            Push tuple $(\emptyset, s_0)$ to list $\mathcal{S}$
        end if
    end if
end while

repeat the last frame and ground truths to fill out the final mini-batch for particular footage. Due to the variable size of each surgical footage, the number of frames repeated tends to be different. In practice, this does not result in many repeated frames as the N* is kept small (4 in our experiments) due to space constraints. However, even for larger N* the repeated frames are bound to have a limited effect on training due to the much larger size of the surgical footage in comparison.

The frames are then resized according to the set row size configuration. We experimented with shifting the mean of the frames by RGB values [0.485, 0.456, 0.406] and standard deviation by [0.229, 0.224, 0.225] for a more contrasted image and got better results than using zero mean and unit standard deviation. However, the shifted mean and

standard deviation would completely blacken out certain important parts of the frame leading to limited learning, thus we made a compromise and used zero mean and unit standard deviation for our more complicated TempRetinaNet models. Finally, for each mini-batch, the batch dimension N* is reshaped into a batch N containing timesteps T. This creates the sequential dataset expected by the time distributed layers and temporal subnets.

For all our experiments, we only load frames that have ground-truths associated with them. As future work, it is worth training on full frame sequences and figuring out a way how to distribute ground-truths of nearby frames to unannotated frames.

3.2.3 Jaccard Threshold/Intersection-Over-Union Threshold (IoU)

As highlighted in Section 1.2.2.4, the positive and negative IoU thresholds determine if the spatial region occupied by each anchor box should be scrutinized for nearness to a ground-truth box during training by labeling them as foreground, background, or ignored depending on the value of their IoU with the ground-truth boxes.

We experimented with a number of (negative, positive) thresholds, namely, (0.4, 0.5), (0.4, 0.6) , (0.3, 0.7), (0.2, 0.8), and found that all sets of thresholds other than (0.4, 0.6) and (0.4, 0.5) failed to learn and resulted in numerical overflow in error calculation. Experiments on RetinaNet showed a slight improvement in mean average precision (mAP) with (0.4, 0.6) thus we used these thresholds for all of our other experiments.

3.2.4 Initialization

The weight parameters of all hidden layers in the temporal subnets and phase model involving ReLU activations are initialized with the He-initialization [He et al. (2015)] while weights of the output heads are initialized normally. All bias terms are initialized with zeros. The ResNet backbone is initialized using ImageNet pre-trained parameters and batch normalization units in the ResNet backbone are frozen and do not participate in learning. The bias term of the classification subnet output head in RetinaNet and TempRetinaNet is initialized by $b = \log((1 - \pi)/\pi)$ where π is the confidence that every anchor at the start of training covers the foreground. We use π=0.01, same as the original RetinaNet [Lin et al. (2017)].

3.2.5 Optimization

We use stochastic gradient descent with momentum (SGDM) with L2 regularization and weight decay parameter set to $1e-4$. Momentum parameter β is set to 0.9. In our

experiments, we drop the learning rate by 10 every 10 epochs for all backpropagation techniques used (BP, BPTT, TBPTT).

3.3 Evaluation Criteria

3.3.1 Average Precision (AP)

For most object detection tasks, the average precision is used as the default evaluation metric, where precision (P) = True Positives (TP)/(True Positives + False Positives (FP). In general cases, positives and negatives are used for binary classification where the term true positive means that a data input that belonged to class A was correctly classified as belonging to A by the model. If the data point belonged to class B but was incorrectly classified as belonging to A by the model, then this would be called a false positive. To find average precision (AP) the recall (R) is also found where R = True Positives/(True Positives + False Negatives (FN). In this case, the instance where a data point belonging to A was classified as B is termed a false negative. Thus the recall is the true positive rate. To find the average precision, we find the area under the curve plotting precision against the recall.

3.3.2 IoU Thresholds

During the evaluation, the IoU thresholds determine how much overlap is required between a prediction and the ground-truth class bounding box for the prediction to be considered a true positive. This is following the same metric used in [Singh Bawa et al. (2021)]. The motivation is that with increasing thresholds, evaluation becomes stricter and this gives an estimation of how *precise* the detector is in its predictions. The thresholds used are {0.1, 0.3, 0.5}. The AP of each class at these thresholds, AP_{10}, AP_{30} and AP_{50}, are then averaged to yield the mean average precision (mAP).

3.4 Experiments

We compared different configurations of the RetinaNet and TempRetinaNet for our experiments to test for the effectiveness of capturing temporal information of different lengths. We also experimented with a binary classifier (BC) added to the RetinaNet pipeline in addition to the existing OvR classifier responsible for predicting the existence of any label. Not shown is the experiment where we trained a RetinaNet without shuffling the data to show that without temporal context, a RetinaNet cannot learn with an imbalanced presentation of data classes. This is because the classification loss of the

RetinaNet was overflowing for the same learning rate and thus the lack of convergence led to us not including its results.

Table 3.2 shows some preliminary experimentation on the conventional RetinaNet (RN). Each experiment is run for 15 epochs and a batch size of 16. It shows how shuffling the data and using different IoU thresholds affect the mAP on the test set. We see that using unshuffled data with the basic spatial object detector is not able to learn due to massive class imbalance over time. The highest mAP configuration consists of IoU thresholds{0.4, 0.6} and is thus further used in the TempRetinaNet experiments.

Model	$\text{Th}_{-\text{ve}}$	$\text{Th}_{+\text{ve}}$	**mAP**
RN (sfl, w/ BC)	0.4	0.6	22.38±0.05
RN (sfl, w/o BC)	0.4	0.6	-
RN (sfl, w/ BC)	0.4	0.5	21.85±0.05
RN (sfl, w/o BC)	0.4	0.6	22.04±0.05
RN (w/ BC)	0.4	0.5	-
RN (w/o BC)	0.4	0.5	-

TABLE 3.2: Performance of RetinaNet with or without binary classifier (BC) and shuffled (sfl) or unshuffled data. The entries labeled '-' represent those models that failed to converge under the same testing conditions. The $\text{Th}_{-\text{ve}}$ and $\text{Th}_{+\text{ve}}$ values represent the negative and positive IoU thresholds. Errors are found by taking the average of the standard errors of AP_{10}, AP_{30} and AP_{50}.

3.4.1 Setup

The primary objective is to compare the efficacy of using a TempRetinaNet (TRN) model that keeps state while passing particular surgical footage with that of a model that does not. Secondary objectives include making the model perform its best through data augmentations and backpropagation techniques.

State vs No-State To keep the experiments fair and fulfill the primary objective, the same TRN model is used but with state management turned on and off. When state management is off, there is no benefit to using TBPTT as the state is reset after each mini-batch. In this case, a new state is generated every time while having no memory of the past. This means we can use standard BPTT in such experiments. When the state is kept, TBPTT is used while passing states between mini-batches.

Contrasted vs Raw Data augmentation techniques have often been shown to help to learn. One of these techniques includes contrasting the image. For such experiments (hicon), we normalize the data and shift the mean of the [R,G,B] values by [0.485, 0.456, 0.406] and the standard deviation by [0.229, 0.224, 0.225]. To compare, we also use the

raw frames from the surgical footage and apply simple zero-mean and unit standard deviation normalization.

Shuffled vs Non-Shuffled We saw in Table 3.2 that using unshuffled data with such a model that does not keep state such as RetinaNet causes an overflow of loss and a failure in learning. Now we have a scenario where we want to run BPTT without keeping state between mini-batches, i.e., we want to keep state for only the length of one mini-batch. As such a mini-batch is quite small, spatiotemporal features learned turned out to be quite limited and thus the loss would not converge over a large number of epochs. Thus for the experiments where we use BPTT with TRN, we shuffle the data as well. In this case, the state maintained inside a mini-batch does not contain anything useful as the data inside the mini-batch is shuffled as well. However, the TRN model benefits from its deeper architecture nonetheless when compared to RetinaNet, leading us to expect some improvements over RetinaNet over the same testing conditions.

We run all experiments for 50 epochs. For experiments that do not use TBPTT, we use a batch size of 16. For TBPTT experiments we use a total batch size (N*) of 4, consisting of N=2 and T=2. Hyper-parameters k1 and k2 are set equal to 1 and 3 respectively. An input row size of 200 is used for all experiments. Generally, early stopping is used for all experiments at the epoch where the mAP on the validation set is the highest. Table 3.3 shows the results of all mentioned experiments with TempRetinaNet (TRN).

Model	CStacks	PDepth	AP_{10}	AP_{30}	AP_{50}	**mAP**
TRN (hicon, sfl, BPTT)	1	0	34.52±0.05	26.03±0.04	14.07±0.03	24.87±0.04
TRN (hicon, sfl, BPTT)	2	0	30.55±0.05	22.76±0.05	14.76±0.04	22.69±0.05
TRN (sfl, BPTT)	1	0	31.20±0.05	18.35±0.05	5.16±0.02	18.24±0.04
TRN (TBPTT)	1	0	5.35±0.02	3.40±0.02	1.78±8.79e^{-3}	3.51±0.02
TRN (TBPTT)	1	4	3.86±0.01	2.17±0.01	0.89±5.08e^{-3}	2.31±8.33e^{-3}

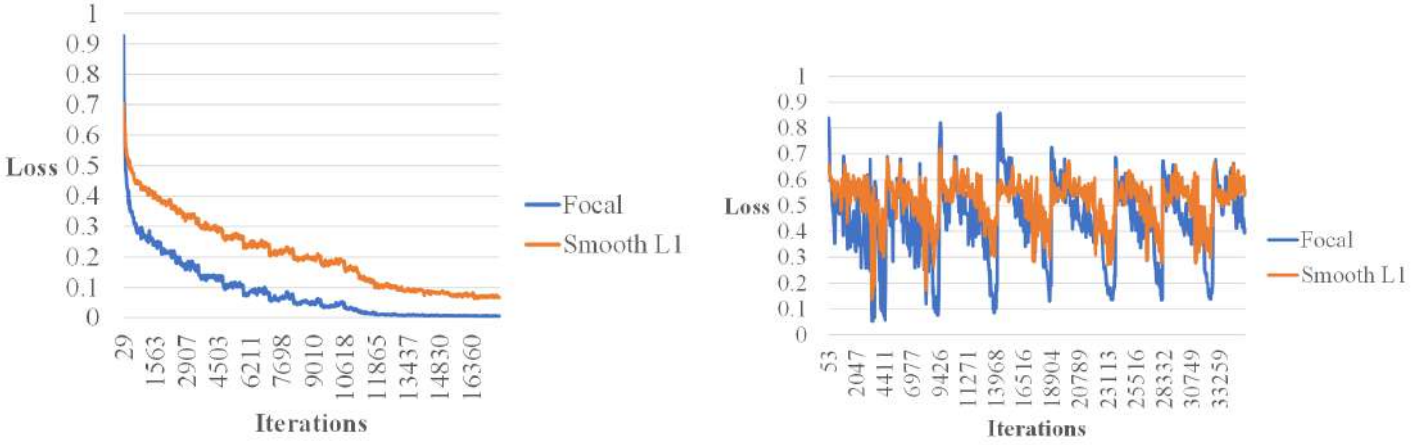
TABLE 3.3: Average precision (AP) at IoU thresholds 0.1, 0.3, and 0.5 along with their standard errors calculated across the precision of all classes. TempRetinaNet (TRN) models include those with high-contrast (hicon), shuffled (sfl) batches and those that use BPTT and TBPTT. In experiments with BPTT with sfl, the state is reset at every step. The column **CStacks** refers to the number of ConvLSTMs stacked per temporal layer and the column **PDepth** refers to the depth of the phase model where a PDepth of 0 signifies that the phase model is not used.

3.5 Evaluation

Shuffled Vs Random Data Experiments showed that smooth learning and an appreciable predictive performance were guaranteed when the dataset was shuffled. This was as expected as with an imbalanced dataset unevenly spread across time, data shuffling presented a greater variety of examples between batches and ensured that learning

was not overly influenced by one action class at a certain point in time. Despite being ∼15 points lower in mAP, using TBPTT with unshuffled data led to some learning while using BP in RetinaNet or BPTT in TempRetinaNet with unshuffled data led to exploding classification losses. This hints at the fact that while our TempRetinaNet model is not perfect, it can learn temporal associations in addition to the spatial cues in the data that prevent it from being influenced too much by the majority action class.

Figure 3.5 shows a comparison of the training loss over iterations for the TempRetinaNet model that uses shuffled data and the model that uses unshuffled data. The converging loss in Figure 3.5 (a) shows that the model is valid for learning spatial features. The regularly oscillating loss in Figure 3.5 (b) shows that despite the presentation of imbalanced classes over time, the model prevents itself from being too overwhelmed by the majority class gradients. However, currently, the TempRetinaNet model does not come close to the benchmark spatial detector using TBPTT and requires further work.



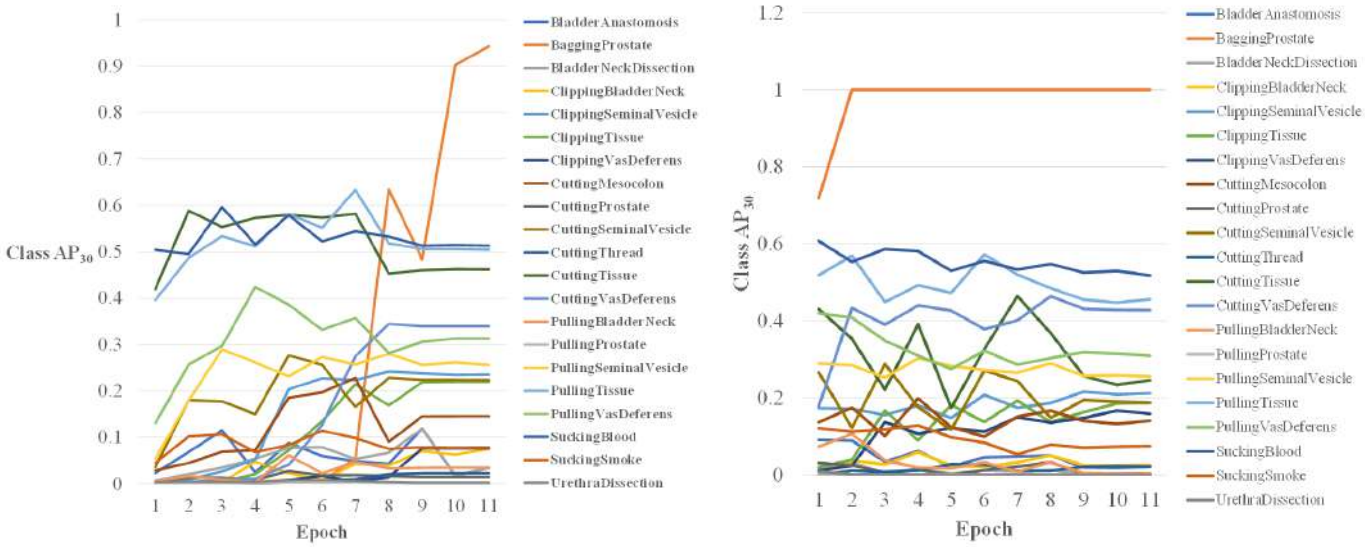
(a) Training Loss TempRetinaNet (sfl, BPTT)

(b) Training Loss TempRetinaNet (TBPTT)

FIGURE 3.5: The training loss converges when BPTT is used in TempRetinaNet with shuffled data. However, when TBPTT is used with sequential data, the loss oscillates regularly across epochs. While this suggests that the model is not able to fully learn spatiotemporal features, it also shows that learning stays stable and that losses do not explode.

Class APs We observed that when we trained our TempRetinaNet with shuffled data, action classes that had the highest AP were usually the same as those that had the highest AP in the RetinaNet model. In general, while the higher AP action classes included mostly the majority classes, the minority class *BaggingProstate* had the highest AP. A closer inspection of the training data revealed that the bag used while bagging the prostate had a very distinctive color and shape which might be why our models were able to capture better spatial features for the class. Figure 3.6 shows the class AP at IoU=0.30 for RetinaNet and TempRetinaNet with shuffled data. We can see some overlap of the higher scoring classes such as *PullingTissue*, *SuckingBlood* and *BaggingProstate*. This

supports the hypothesis that the TempRetinaNet with shuffled data effectively acts as a deeper spatial detector but because no real temporal associations are learned, some noise is introduced in learning class features. Thus, we do not see the same order of highest scoring classes in both models.



(a) RetinaNet class AP_{30} over epochs

(b) TempRetinaNet class AP_{30} over epochs

FIGURE 3.6: Class AP at IoU=0.30 over epochs of the validation set in RetinaNet and TempRetinaNet (sfl, BPTT). We see that while most of the class actions above the ∼0.4 mark are majority classes, namely, *PullingTissue*, *SuckingBlood* etc., the class AP of the specialized action class *BaggingProstate* rises to a perfect 1.0 for both the models.

Color-Altered Images For the case with data consisting of raw frames, we saw that TempRetinaNet had a mAP score ∼4.14 points lower than the RetinaNet but ∼2.49 points higher when high-contrast. In older experiments, we tested high-contrast images with RetinaNet as well and saw a higher mAP. In closer inspection we saw that contrasting caused parts of the image to blacken out, leaving the model to learn a more limited feature space across images. We believe that despite positive results, such a solution should be used in conjunction with data augmentation techniques during training rather than be the sole data available as for a temporal model, it can be difficult to get a better context of the surgical scene with parts of the image blackened out. For a spatial model, using this in conjunction with other data augmentation techniques has the plausibility of making the model more robust.

Stacked ConvLSTMs Stacked (bidirectional) LSTMs are mostly used as an industry-standard due to their ability to learn from both the past and future. In our experiments due to space limitations, we were only able to test stacked ConvLSTMs with BPTT. We found that it performed slightly worse than its non-stacked counterpart. While we used

shuffled data (non-sequential) for these experiments, we wanted to test if there would be any impact of making the temporal layer deeper with more recurrent units. We found that multiple recurrent blocks do not have a positive impact on performance when data is random most likely because no meaningful temporal associations are learned.

Surgical Phase Effectiveness TempRetinaNet with phase recognition in Table 3.3 showed a poor prediction performance. The phase information did not seem to provide the temporal cues as expected. A deeper analysis of the phase training loss over time showed why this was the case. Figure 3.7 shows the cross-entropy loss over iterations. As one surgical phase can occur for a large number of timesteps (Figure 3.2), the TempRetinaModel overfits classification features to the particular phase it is currently iterating. Thus upon the presentation of a new phase, the cross-entropy loss spikes back up, and soon the model starts overfitting to that phase. Gradually as the learning rate decreases, the model stops overfitting the classification features to a particular phase and the height of the phase loss spikes reduce. However, the model is still not able to distinguish one phase from another with high confidence, thus the loss tapers off at a certain point.

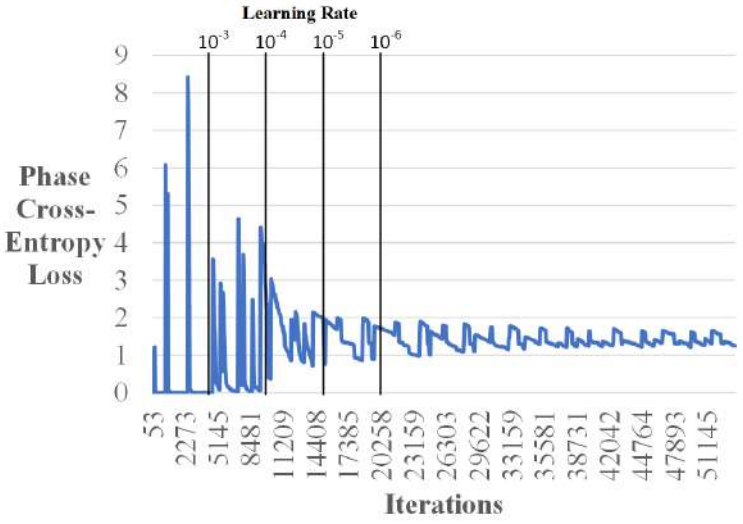


FIGURE 3.7: The phase loss oscillates with varying amplitudes in an identifiable pattern over time. At the start of training, the phase network is quick to associate classification features to a phase as one phase occurs for a set of timesteps. As soon as a new phase comes up, the loss spikes up until that phase is learned as well. With decreasing learning rates over time, the spikes become shorter but the model is still not able to learn distinctive classification features that properly distinguish one surgical phase from another.

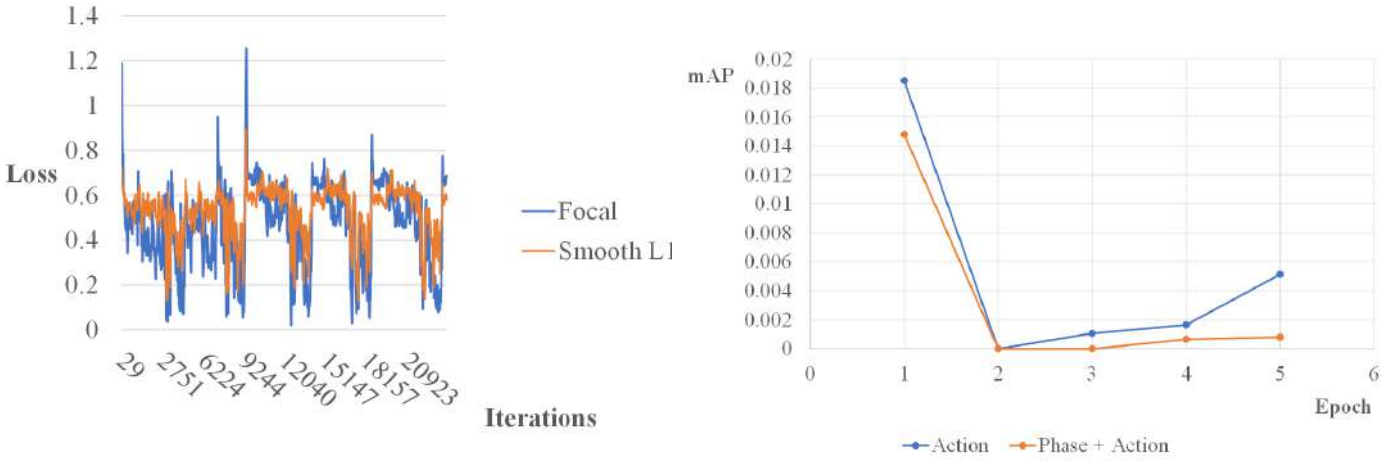
We conducted this experiment with a hidden layer depth of 4 for the phase model. We did this after observing a similar pattern for the model with depth 2. As future work, we would like to experiment with an even deeper phase model to learn more complex features. More importantly, we would like to keep the learning rate of the phase model

at a lower magnitude than the rest of the model to prevent the overbearing influence of the current phase while training.

3.6 Ablation

3.6.1 Shallower Temporal Layer

We showed previously in Figure 3.1 that each temporal layer consists of a sequence of ConvLSTM-Conv2d pairs. We experimented with several ConvLSTM-Conv2d pairs per temporal layer and saw their effect on learning for the basic action detection task as well as the phase recognition task. As our computation budget would not allow us to use a larger number of pairs, we compared pairs of sizes one and two, the latter being the one in TempRetinaNet originally proposed. We saw that the model learned very slowly with a single pair and that the phase model failed to learn altogether after 5 epochs with an exploding classification loss. This might be because the features captured by the classification subnet might not be complex enough for the phase model to find patterns in, resulting in a bad performance of both the phase model and the classification model. We can see the single-pair model performance in Figure 3.8. We refused to train for longer due to the very slow learning overall and time constraints.



(a) Training loss over iterations

(b) mAP over epochs

FIGURE 3.8: Left: training loss trend in surgical action detection model with single ConvLSTM-Conv2d pair per temporal layer. Not shown is the training loss trend for the phase recognition multi-task that had exploding losses after epoch 5. Right: comparison of the performance of single pair models on the surgical action detection task and the phase recognition multi-task for 5 epochs. Though both models have a low mAP, the phase model performs worse and shows signs of slower learning.

3.6.2 Rapid Learning Rate Reduction

Experiments (RetinaNet and TempRetinaNet) conducted with BPTT and BP all had the same learning rate schedule consisting of a drop of 10 every 10 epochs starting with a value of 0.01. We saw that doing this resulted in a sharp decrease in the training loss at every such point. However, we saw in Figure 3.8 that for every epoch, the loss seemed to oscillate regularly, pointing to a lack of effective learning. To test the contribution of the learning rate to the learning, we changed our schedule to dropping the learning rate by 10 every epoch. We did this experiment on the single-pair-temporal layer TempRetinaNet in the surgical action detection task. As the single-pair experiment granted us a greater space budget, we tested with N*=8 thus the TBTT algorithm had a long record of the past.

Comparing with 3.8, the rapidly reducing learning rate and greater batch size boosted the prediction performance of the model by a significant amount. While this was not in the proportions of those models using BP and BPTT, our experiments show that complex temporal models such as ours that use TBPTT on challenging imbalanced sequential datasets can be better trained by these methods. From the perspective of TBPTT, going from N*=4 to N*=8 means that the model keeps the state of double the timesteps in the past, thus the temporal model can create better long-range associations. As future work, experiments need to be conducted with larger values of hyperparameters N*, k1 and k2 and secondly, the learning rate drop should be made more dynamic according to the rate in the drop of training loss at each epoch.

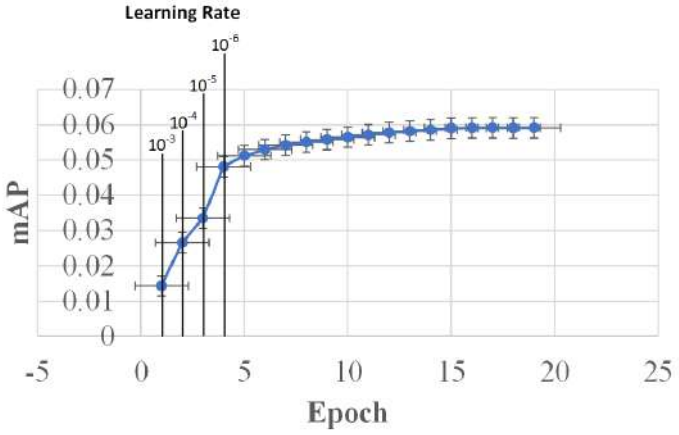


FIGURE 3.9: mAP over epochs of the validation set. The total batch size N* is kept at 8. In the first three epochs, the learning rate is dropped by 10 each time. We can see a clear improvement in the predictive performance of the single-pair TempRetinaNet model with TBPTT over Figure 3.8 where the learning rate is dropped by 10 every 10 epochs.

Chapter 4

Conclusions

In this work, we provided limitations to one of the best performing object detectors, namely RetinaNet for the surgical action detection tasks in laparoscopic footage. We stated why the challenges of class imbalance over time and aberration in the dataset pose a problem to any *pure* spatial detector. Subsequently, we then provided extensions to the RetinaNet architecture in the form of the TempRetinaNet model as well as a better-suited backpropagation method called the truncated backpropagation through time to overcome the challenges. Our motivations were to capture long-range temporal associations between surgical action classes in hours of surgical footage in addition to spatial features. Finally, we presented an additional multi-task semi-supervised learning method for the model to obtain further temporal cues from surgical phases.

Our experiments showed that under our limited testing conditions, RetinaNet performed better than TempRetinaNet. We were also able to show that the models learned better spatial features with high-contrast images. While the TempRetinaNet model had a predictive performance significantly lower than the RetinaNet model, it outperformed RetinaNet for unshuffled data. We found that RetinaNet would not learn at all with unshuffled data due to the severe class imbalance and thus would get overwhelmed by the consecutive presentation of the majority class early on in training. However, TempRetinaNet would preserve temporal associations for classification over time. We observed that the model trained faster with rapidly decreasing learning rates as well at each epoch.

4.1 Future Works

Peephole ConvLSTM We stated in Section 1.4.3.2 that we used a simplification of the ConvLSTM so it better resembled the conventional LSTM equations. However, we saw that the training loss would not converge and would start oscillating at regular

intervals. While many factors in our testing conditions could account for this, it is worth testing the original Peephole-LSTM design of the ConvLSTM as it provides the model with the capability to learn from long time intervals between events. This has the potential to further mitigate the problems of learning imbalanced classes over time.

Larger K1 and K2 The ability of the TempRetinaNet model to learn temporal cues is majorly governed by the TBPTT parameters k1 and k2. With a larger k1, gradients can be accumulated over a larger number of iterations before a backpropagation step is run. This makes training significantly faster and can allow for training over more epochs. A larger k2 means that at each iteration, a long history of the past is stored in the states, and thus backpropagation through time unrolls to a greater number of timesteps. This means that the model can learn class associations from longer time intervals. Both the parameters k1 and k2 have an impact on the computational budget and thus the computational budget would have to be improved as well.

Deeper TempRetinaNet The failure of TempRetinaNet to fit the data may also be due to a shallow temporal subnet. It is worth experimenting with a greater number of stacked (bidirectional) ConvLSTMs as well as a greater number of temporal layers. Stacked ConvLSTMs will allow for learning temporal cues from the past and future in a given mini-batch and more temporal layers will introduce more non-linearity to the learning process.

Phase Focal Loss We saw that surgical phase cross-entropy loss is highly overwhelmed by the consecutive presentation of the same phase early on in the training. This can be mitigated by using focal loss in place of the cross-entropy loss as the focal loss down-weights well-classified examples. The focusing parameter λ can be tweaked to reduce the influence of a well-classified example and observe which parameter delivers the most stable learning of phase, devoid of spikes in the loss.

Dynamic Learning Rate Schedule In TempRetinaNet, the training loss would oscillate with the same range of amplitude if the learning rate was not dropped after each epoch. To make this more effective, a dynamic schedule should be experimented with to drop the learning rate based on the error across the previous epoch. Furthermore, the problem of the phase model being overwhelmed by the consistent presentation of the same phase can also be improved upon by lowering the learning rate of the phase model from the start. This way, the phase model can have an independent schedule to drop the learning rate.

Data Augmentation We saw that using high-contrast images made learning easier for the model and resulted in greater predictive performance. We also stated that using such modified images only limited the quality of features learned by the model. With this motivation and the fact that our training set is only limited to two surgical procedures, it is worth experimenting with data augmentation techniques to increase the size of the training set and make the model more robust. Learning better spatial features can ultimately aid the model in making better temporal associations between classes and thus further improve learning with sequential data.

Bibliography

E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. 1984, Pyramid methods in image processing. *RCA Engineer*, 29(6):33–41, 1984.

Max Allan, Ping-Lin Chang, Sebastien Ourselin, David Hawkes, Ashwin Sridhar, John Kelly, and Danail Stoyanov. Image based surgical instrument pose estimation with multi-class labelling and optical flow. 10 2015. ISBN 978-3-319-24552-2.

Tobias Blum, Hubertus Feussner, and Nassir Navab. Modeling and segmentation of surgical workflow from laparoscopic video. volume 13, pages 400–7, 09 2010. ISBN 978-3-642-15710-3.

Sebastian Bodenstedt, Martin Wagner, Darko Katić, Patrick Mietkowski, Benjamin Mayer, Hannes Kenngott, Beat Müller-Stich, Rüdiger Dillmann, and Stefanie Speidel. Unsupervised temporal context learning using convolutional neural networks for laparoscopic workflow analysis, 2017.

David Bouget, Rodrigo Benenson, Mohamed Omran, Laurent Riffaud, Bernt Schiele, and Pierre Jannin. Detecting surgical tools by modelling local appearance and global shape. *IEEE Transactions on Medical Imaging*, 34(12):2603–2617, 2015.

Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Stacked bidirectional and unidirectional lstm recurrent neural network for forecasting network-wide traffic state with missing values, 2020.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:886–893, 2005.

Jeffrey Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 03 1990.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

Pedro F. Felzenszwalb, Ross B. Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2241–2248, 2010.

Germain Forestier, Florent Lalys, Laurent Riffaud, D. Louis Collins, Jurgen Meixensberger, Shafik N. Wassef, Thomas Neumuth, Benoit Goulet, and Pierre Jannin. Multi-site study of surgical practice in neurosurgery based on surgical process models. *Journal of Biomedical Informatics*, 46(5):822–829, 2013. ISSN 1532-0464.

Germain Forestier, Laurent Riffaud, and Pierre Jannin. Automatic phase prediction from low-level surgical activities. *International Journal of Computer Assisted Radiology and Surgery*, 10(6):833–841, April 2015.

Cheng-Yang Fu, Mykhailo Shvets, and Alexander C. Berg. Retinamask: Learning to predict masks improves state-of-the-art single-shot detection for free, 2019.

Isabel Funke, Sebastian Bodenstedt, Florian Oehme, Felix Von Bechtolsheim, Jürgen Weitz, and Stefanie Speidel. Using 3D Convolutional Neural Networks to Learn Spatiotemporal Features for Automatic Surgical Gesture Recognition in Video. Technical report, 2019.

Yixin Gao, S. Vedula, C. Reiley, N. Ahmidi, Balakrishnan Varadarajan, Henry C. Lin, Lingling Tao, L. Zappella, B. Béjar, D. Yuh, C. C. G. Chen, R. Vidal, S. Khudanpur, and Gregory Hager. Jhu-isi gesture and skill assessment working set (jigsaws) : A surgical activity dataset for human motion modeling. 2014.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *J. Mach. Learn. Res.*, 3(null):115–143, March 2003. ISSN 1532-4435.

Ross Girshick. Fast R-CNN. 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

Alex Graves. Generating sequences with recurrent neural networks, 2014.

Alex Graves, Santiago Fernandez, and Juergen Schmidhuber. Multi-dimensional recurrent neural networks, 2007.

Hassan Al Hajj, Mathieu Lamard, Pierre-Henri Conze, Béatrice Cochener, and Gwenolé Quellec. Monitoring tool usage in surgery videos using boosted convolutional and recurrent neural networks, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Shuiwang Ji, W. Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:221–231, 2013.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding, 2014.

Darko Katic, Anna-Laura Wekerle, Fabian Gärtner, Hannes Kenngott, Beat Peter Müller-Stich, Rüdiger Dillmann, and Stefanie Speidel. Knowledge-driven formalization of laparoscopic surgeries for rule-based intraoperative context-aware assistance. In Danail Stoyanov, D. Louis Collins, Ichiro Sakuma, Purang Abolmaesumi, and Pierre Jannin, editors, *Information Processing in Computer-Assisted Interventions - 5th International Conference, IPCAI 2014, Fukuoka, Japan, June 28, 2014. Proceedings*, volume 8498 of *Lecture Notes in Computer Science*, pages 158–167. Springer, 2014.

Michael Kranzfelder, Armin Schneider, Adam Fiolka, Elena Schwan, Sonja Gillen, Dirk Wilhelm, Rebecca Schirren, Silvano Reiser, Brian Jensen, and Hubertus Feussner. Real-time instrument detection in minimally invasive surgery using radiofrequency identification technology. *The Journal of surgical research*, 185, 07 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

Xiaobo Li, Haohua Zhao, and Liqing Zhang. *Recurrent retinaNet: A video object detection model based on focal loss*, volume 11304 LNCS. Springer International Publishing, 2018. ISBN 9783030042110.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. Technical report, 2016.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. Technical report, 2017.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox Detector. Technical report, 2015.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004. ISSN 0920-5691.

G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4): 235–244, 1990.

Joel Ruben Antony Moniz and David Krueger. Nested lstms, 2018.

Nicolas Padoy, Tobias Blum, Seyed-Ahmad Ahmadi, Hubertus Feussner, Marie-Odile Berger, and Nassir Navab. Statistical modeling and recognition of surgical workflow. *Medical Image Analysis*, 16(3):632–641, 2012. ISSN 1361-8415. Computer Assisted Interventions.

Nicolas Padoy, Diana Mateus, Daniel Weinland, Marie-Odile Berger, and Nassir Navab. Workflow Monitoring based on 3D Motion Features. In *Workshop on Video-Oriented Object and Event Classification in Conjunction with ICCV 2009*, 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), pages 585–592, Kyoto, Japan, September 2009. IEEE. IBM Best Paper Award ; ISBN : 978-1-4244-4442-7.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 2015.

Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, jun 2017. ISSN 01628828.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. *CoRR*, abs/1604.03540, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Vivek Singh Bawa, Gurkirt Singh, Francis Kaping'a, Inna Skarga-Bandurova, Elettra Oleari, Alice Leporini, Carmela Landolfo, Pengfei Zhao, Xi Xiang, Gongning Luo, Kuanquan Wang, Liangzhi Li, Bowen Wang, Shang Zhao, Li Li, Armando Stabile, Francesco Setti, Riccardo Muradore, and Fabio Cuzzolin. The SARAS Endoscopic Surgeon Action Detection (ESAD) dataset: Challenges and methods. Technical report, 2021.

Ilya Sutskever. *Training Recurrent Neural Networks*. PhD thesis, CAN, 2013. AAINS22066.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

Andru P. Twinanda, Sherif Shehata, Didier Mutter, Jacques Marescaux, Michel de Mathelin, and Nicolas Padoy. EndoNet: A Deep Architecture for Recognition Tasks on Laparoscopic Videos. *IEEE Transactions on Medical Imaging*, 36(1):86–97, feb 2016.

Andru Putra Twinanda. Vision-based approaches for surgical activity recognition using laparoscopic and rbgd videos. (approches basées vision pour la reconnaissance d'activités chirurgicales à partir de vidéos laparoscopiques et multi-vues rgbd). 2017.

J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

Beatrice van Amsterdam, Matthew J Clarkson, and Danail Stoyanov. Multi-Task Recurrent Neural Network for Surgical Gesture Recognition Multi-Task Recurrent Neural Network for Surgical Gesture Recognition and Progress Prediction. Technical report, 2020.

Yu-Shin Wang and Kai-Tai Song. Image-based pose estimation and tracking of surgical instruments in minimally invasive surgery. In *2020 International Automatic Control Conference (CACS)*, pages 1–6, 2020.

Gaurav Yengera, Jacques Marescaux, and Nicolas Padoy. Less is More: Surgical Phase Recognition with Less Annotations through Self-Supervised Pre-training of CNN-LSTM Networks. Technical report, 2018.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

Aneeq Zia, Andrew Hung, Irfan Essa, and Anthony Jarc. Surgical Activity Recognition in Robot-Assisted Radical Prostatectomy using Deep Learning. Technical report, 2018.