Technische Hochschule
Brandenburg
University of
Applied Sciences
Fachbereich
Informatik und Medien

Effect of color space on deep learning algorithm for surgical image classification

Author: Julien Schwerin
Matriculation number: 20200541
Submitted on: 29 March 2021

A Bachelor's Thesis submitted in partial fulfilment of the requirements for the degree of BSc (Computer Engineering)

1st referee: Dipl.-Inform. Ingo Boersch
2nd referee: Prof. Dr.-Ing. Jochen Heinsohn
Supervisor: Dr.rer.nat. Florian Aspart

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

Effect of color space on deep learning algorithm for surgical image classification

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 29.03.2021

Julien Schwerin

Zusammenfassung

Die videoassistierte minimalinvasive Chirurgie ist dank diverser Vorteile immer dann, wenn es möglich ist die erste Wahl im Operationssaal. Durch die Verwendung einer Kamera und filigraner Werkzeuge entstehen daraus viele intraoperative Vorteile für den Patienten. Zusätzlich lassen sich riesige Datenmengen in Form von Bildern und Videos erzeugen. Die gewonnenen Daten sind sowohl für die Lehre als auch für die Evaluation von Operationsverläufen sehr Wertvoll. Außerdem können auch für das trainieren von Deep Learning Modellen verwendet werden. Unterstützung in der intraoperativen Entscheidungsfindung, Risikoeinschätzungen oder Beurteilungen der Performanz des Operateurs sind nur einige von unzähligen daraus resultierenden Möglichkeiten.

Deep Learning Algorithmen arbeiten mit Bildern die meist aus drei Farbkanälen aufgebaut sind. Ob die Farbräume, die den Kanaelen zu Grunde liegen einen Einfluss auf die Performanz eines modernen Deep Learning Modells zur Klassifizierung haben ist Teil dieser Untersuchung.

Da Rot-Grün-Blau aktuell der Standardfarbraum im Deep Learning ist wird er in diesem Versuch als Vergleichswert verwendet. Der zweite Farbraum ist Hue-Saturation-Value.

Trainiert und evaluiert wird auf dem öffentlichen verfügbaren Cholec80 Datensatz. Dieser enthält gelabelte Operationsvideos von insgesamt 80 laparoskopischen Cholecystektomien. Im Rahmen der Arbeit werden einfache Modelle in verschiedenen Farbräumen trainiert um einzelne Werkzeuge zu klassifizieren. Im Anschluss werden die erzielten Ergebnisse verglichen.

Im Laufe der Untersuchung zeigte sich, dass die Farbcodierung einen Einfluss auf die Performanz hat. Trotzdem erzielte Rot-Grün-Blau die besten Ergebnisse. Gemessen an der Tatsache, dass dieser auch der meistgenutzte Farbraum ist, ist das Ergebnis zufriedenstellend.

Abstract

Thanks to various advantages, video-assisted minimally invasive surgery is the first choice in the operating room whenever possible. The use of a camera and delicate tools results in many intraoperative advantages for the patient. In addition, huge amounts of data can be generated in the form of images and videos. The data obtained is very valuable for teaching as well as for the evaluation of surgical procedures. Moreover, it can also be used for training deep learning models. Support in intraoperative decision making, risk assessments or evaluations of the surgeon's performance are just a few of countless resulting possibilities.

Deep Learning algorithms work with images that are mostly composed of three color channels. Whether the color spaces underlying the channels have an influence on the performance of a modern Deep Learning model for classification is part of this investigation.

Since Red Green Blue is currently the standard color space in Deep Learning it is used as a benchmark in this experiment. The second color space is Hue-Saturation-Value.

Training and evaluation is performed on the publicly available Cholec80 dataset. This contains labeled surgical videos of a total of 80 laparoscopic cholecystectomies.In the context of the work, simple models are trained in different color spaces to classify individual tools. Subsequently, the obtained results are compared.

During the course of the study, color encoding was shown to have an impact on performance. Nevertheless, Red Green Blue achieved the best results. Measured against the fact that this is also the most used color space, the result is satisfying.

Acknowledgements

First of all I feel great gratitude for Florians support in the preparation and execution of this work. His wealth of experience technical know-how took and patience me a long way in preparing this investigation. Also his insightful feedback brought this thesis to a higher level.

Secondly I would like to thank Dipl.-Inf. Ingo Boersch for his constructive feedback and the time he took to answer all my questions. I am thankful to both Mr. Boersch and Prof. Dr. Heinsohn that they have accepted the topic for this paper and will review it.

Thirdly, I am very grateful to my team at Caresyntax for allowing me the time to write the bachelor thesis.That there is always someone to deal with my questions and that we manage to have online data science social events despite home office and lockdowns.

A very special thank you goes from the bottom of my heart to my future wife Josi. Without her encouragement, her support and her beeing my most important wingwoman I would definetly not be able to write acknowledgements right now.

Lastly I would also like to thank the THB and all its employees. The study and all the experiences that come with it are something I would not want to miss.

Contents

List of Figures

List of Tables

Nomenclature

acc	accuracy
AUC	Area Under the Curve
AUROC	AREA Under Receiving Operator Characteristics
AVG	average
bn	batch normalization
CC BY	Creative common attribution license
CNN	convolutional neural network
CS	color space
fc	fully connected layer
FPR	False Positive Rate
HSV	Hue Saturation Value
HSVrgb	Hue Saturation Value but normalized as if it was RGB.
LR	learning rate
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ROC	Receiving Operator Characteristic
TPR	True Positive Rate
val	validation

1 Introduction

In many areas of everyday surgery, video-assisted minimally invasive surgery is no longer just an alternative to open surgery, but the standard. Instead of one large incision, several small ones are made at tactically appropriate locations. These much smaller wounds inflicted on the body heal faster and better. The camera used in minimally invasive surgery has many other advantages. With the help of machine vision and learning, it is possible to assist the surgeon postoperatively and even intraoperatively. For example, convolutional neural networks (CNN) can be used to assess which phase of surgery the surgeon is currently in, how safe it is to make an incision in the currently visible tissue, or how experienced the surgeon is as such. Postoperatively, the acquired images can be used to further train existing models or to evaluate the course of an operation and thereby teach students or inexperienced surgeons. By doing this machine vision proves to be of a big importance a safe and future-oriented surgery.

In order to take advantage of all of the above and countless other benefits, it is important that the CNN can work as effectively as possible with the image data from the surgeries. There are lots of performance influencing factors for machine learning like preprocessing and normalization of given images. In this context, the encoding of the image, e.g. the color space, could impact the classification. Whether and how big this impact is shall be investigated in this thesis.

1.1 Motivation

When thinking about optical perception, colors play a major role in addition to shapes, distances and sizes. It doesn't matter whether the field of vision is that of an animal, a human being or a machine. Therefore, it makes sense to investigate their influence on the accuracy of models for object classification.

In everyday life, when taking a photo, while watching a film or working on the computer, there are different color spaces which are used to represent different things. Red Green Blue (RGB) is the most commonly used color space in the field of machine vision or deep

learning. Although Hue Saturation Value (HSV) is preferable because it separates chromaticity from luminance (1). In addition it was shown that HSV color space perfoms better when it comes to finding surgical tools (2). State of the art classifiers take images color encoded in RGB into training without any transformations (3). Images obtained from videos of surgical procedures are particularly striking for the frequency of red colors. Thus, the color red can be found in them in all imaginable nuances. From the delicate pink of the colon to the strong red of the blood and the brownish red of the liver (cf. figure 1.1). Of course there are also other colors in the abdomen. However, colors like blue or black usually appear only in the form of instruments or sutures within the body.

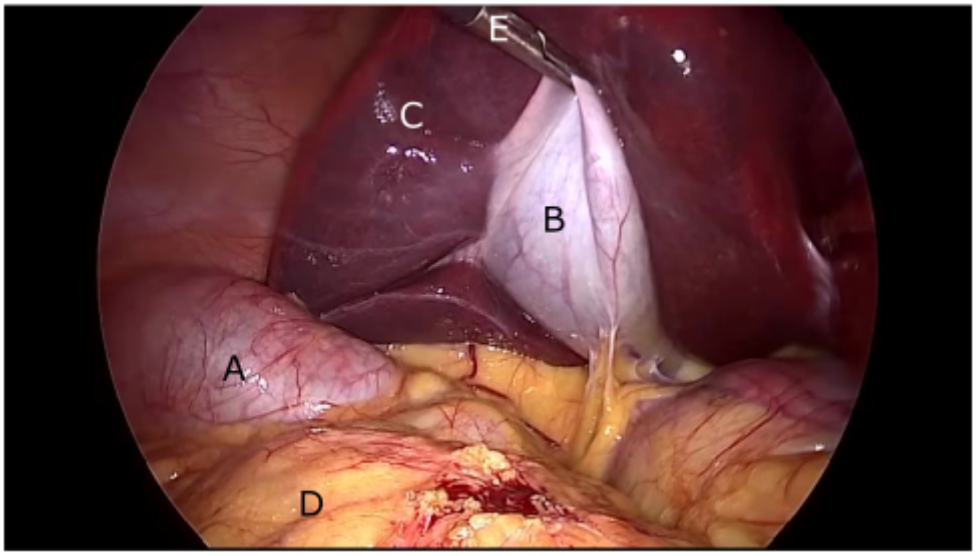

Figure 1.1: **Overview of the abdomen** showing the colon **(A)**, the gallbladder **(B)**, the liver **(C)**, the omentum majus (a structure of fat and connective tissue) **(D)** and a tool (grasper) **(E)**.

Considering that HSV achieved better results than RGB when using only traditional computer vision techniques (1). Therefore, in the context of this work it shall be investigated whether it is possible to make the tools more visible to the classifier with the help of color encoding. In addition, whether the normalization, adapted to the color space, has further influence.

1.2 Scope

The aim of this work is to investigate a potential performance improvement of a classifier for surgical instruments. Two possible color spaces are considered for this purpose. RGB as current standard space and HSV as alternative. Furthermore, the datasets are preprocessed differently depending on their color spaces.

Afterwards, the performance of the different classifiers is examined with respect to their AUROC values. Thereby, the color spaces and the different preprocessing steps are also discussed.

2 Background

The Cholec80 dataset (4) is used as a benchmark for the instrument presence detection. The classifiers are trained and evaluated on this dataset. For comparison and evaluation of the calssification performance the AUROC metric is used.

2.1 Deep Learning

Deep Learning is a special method of data processing and belongs to machine learning. This method uses artificial neural networks (ANN) to process large amounts of data. (5) For this work, a convolutional neural network (CNN) was written. A CNN belongs to the ANNs. The special thing about them is that they have one or more layers to perform a convolution of the given images. When an image passes the convolutional layers, the model recognizes coherent pixels. When an image passes through an ANN of a deep learning model, the model recognizes similar pixels in the first layer. Assembles them into features as it passes through the following layers. In this way a model learns which shapes or structures can be used to classify the given image. (6) An CNN consists of many processing units, the neurons, which are arranged in layers and connected linearly or non-linearly. Each has an input layer, several hidden layers and an output layer. Between those layers the data gets processed by different operations. Figure 2.1 shows a simple example of an ANN containing

- an input layer with three neurons
- two hidden layers with each eight neurons
- an output layer with one neuron

The number of layers and neurons can theoretically be expanded infinitely.

2.2 Cholec80 dataset

This dataset consists of 80 videos showing laparoscopic cholecystectomies performed by 13 different surgeons. A cholecystectomy is a professional term meaning the removal of the gallbladder. Figure 1.1 and 3.1 show some frame taken from this dataset. Laparoscopic

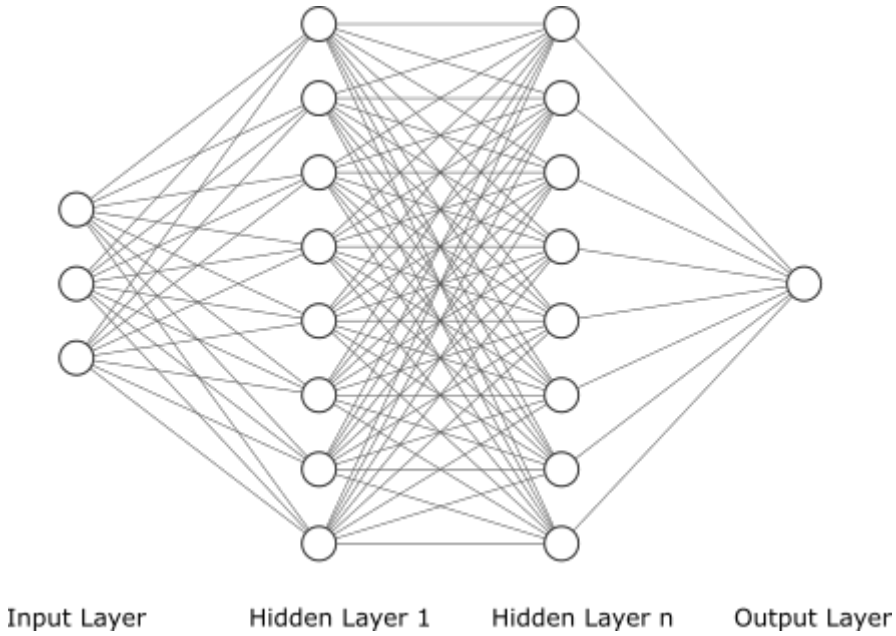

Figure 2.1: **Simple exemplary representation of an artificial neuronal network.** It contains an input layer, two hidden layers and an output layer.
(This figure was created using http://alexlenail.me/NN-SVG/index.html)

means that instead of a large incision several smaller ones are done. In those smaller incisions an optic connected to a camera and the tools need to perform the procedure are inserted to the abdomen. The creators of this dataset have also given tool presence labels set at 1 frame per second. A tool is considered present if at least half of its tip is visible in the frame. Furthermore it contains labels for the phases of the procedure done by a senior surgeon. This dataset contains eight different tools which are grasper, hook, clipper, irrigator, scissors, specimen bag, bipolar forceps and cotton swab. Beneath those labels it also contains a labeling of the surgery phases at 25 fps. (7)

2.3 Color Spaces

A color space describes the range of all displayable colors in a color model. Depending on where colors are represented or perceived, different color spaces are needed. It is possible to convert between the different color spaces. For example, the opsines of the human eye perceive colors in the color spectrum Red Green Blue (RGB). Images on monitors are displayed according to the same principle. In order to be able to print those images in accurate colors, they must first be converted into the color space Cyan Magenta Yellow Black (CMYK).

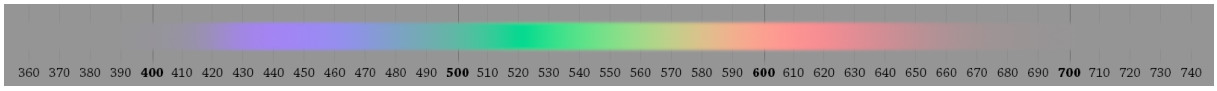

Figure 2.3: **Cutout of the spectrum of light.** The numbers on at the bottom show the wavelengthes of the light. The color bar above those numbers shows the colors that the rods in the human eye sense.
(Adapted from wikimedia commons, Authour: Spigget, File:Rendered Spectrum.png, CC BY-SA 3.0))

Red Green Blue

RGB is an additive color space based on color perception by the rods in the human eye. These are able to perceive light with wavelengths from about 380nm to about 750nm (Figure 2.3). Beeing an additive colorspace means that all colors belonging to this space can be formed by adding the three primary colors. This color space can be well represented as a cube in a three-dimensional coordinate system as shown in figure 2.2. Each axis represents the degree of saturation of a different color. In the coordinate origin is the color black and in the opposite corner is white.

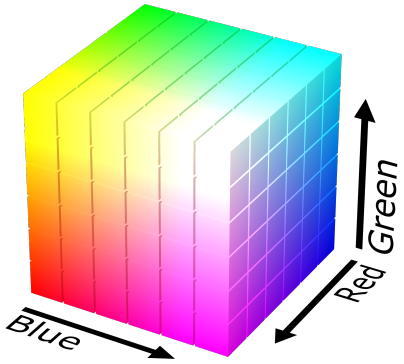

Figure 2.2: **RGB colorspace displayed as cube.** This example shows the three color channels red, green and blue in a three dimensional coordinate system as well as the additive behaviour of the color space.
(Adapted from wikimedia commons, Authour: Datumizer, File:RGB color solid cube.png, CC BY-SA 4.0)

Hue Saturation Value

The color space Hue Saturation Value (HSV) is formed with the three mentioned channels. The special feature of this color space is that the Hue channel is circular. Unlike RGB, HSV is therefore represented more as a cone (Shown in Figure 2.4) or cylinder. An advantage of this color space is that a color is determined only by an angle in the hue channel. The saturation channel gives information about how pure a color is. The value channel in turn

shows the darkness level of the color.
From this structure of the color space results the advantage that coherent areas can be recognized better as such independent of any better or worse illuminated areas. Such variations in illumination are more likely to affect the saturation and value channels.

The color spaces RGB and HSV can be linear transformed into another. Python has got different frameworks performing this. In this thesis cv2.cvtCOLOR() is used.

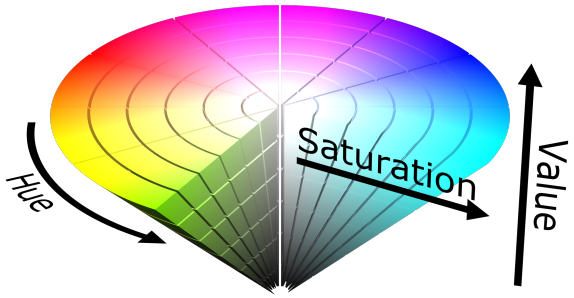

Figure 2.4: **HSV color space displayed as cone.** This example cone shows the circular structure of the hue channel.It also displays the value and the saturation channel.
(Adapted from wikimedia commons, Author: Datumizer, File:HSV color solid cone.png, CC BY-SA 4.0)

2.4 Evaluation

2.4.1 Confusion Matrix

A confusion matrix like the one in figure 2.5 evaluates the performance of a classifying model. Therefore it compares the ground truth labels with the predicted ones. It is a square matrix whose size depends on the number of possible classes. A binary classification as done in this thesis results in 2x2 matrices, Three possible classes would result in 3x3 matrices and so on.

The rows of a confusion matrix represent the actual labels while the columns represent the predicted ones. Therefore, in the example matrix shown each row and column got a zero or a one as their index. TN in the first cell stands for true negative. This means, that the classifier predicted zero and that came out to be true as the ground truth label said the same. TP on the other hand means true positive. The classifier predicted one and that prediction was correct. FP means false positive and FN false negative. In both cases the prediction the classifier did, whether it has been one or zero, was found to be false.

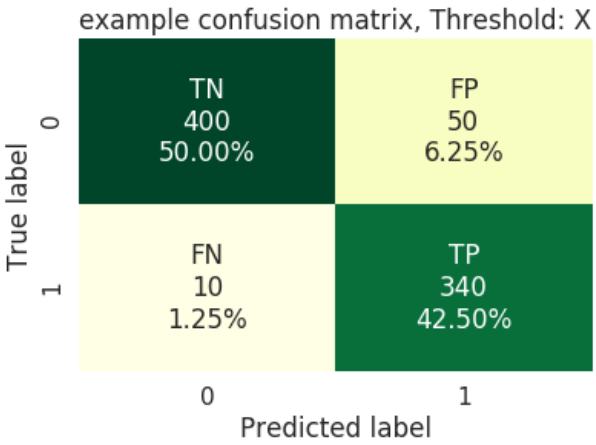

Figure 2.5: **Example confusion matrix** containing arbitrarily values for explanatory and illustrative reasons.

The numbers underneath those four abbreviations tell the total amount of predictions belonging to each category and their share in percent. Usually the color of a cell gets darker when the number of predicitions belonging to that cell rises. Therefore, simply spoken, the aim is to get the diagonal of the matrix as dark as possible by reaching lots of TP and TN scores.

2.4.2 Receiver Operating Characteristic (ROC)

The Receiver Operating Characteristic Curve is a metric that allows the evaluation of binary classification problems. It plots the True Positive Rate or Sensitivity ($\frac{TruePositive}{TruePositive\ +\ FalseNegative}$) on the Y-axis against the False Positive Rate ($1\ -\ \frac{TrueNegative}{TrueNegative+FalsePositive}$) on the X-axis (Figure 2.6). This is done on different thresholds for the y score.

The y score in this thesis is the probability that an image belongs to a class. It has a minimum value of zero, which means the classifier thinks the image does probably not belong to the class and a maximum value of 1 which means the classifier is absolutely sure the image belongs to the class we are interested in.

Figure 2.6 shows a simple and explaining example of a ROC curve. The red dashed line marks a classifier that is not able to classify. An ROC curve showing a graph close to that line is probably just randomly guessing the classes. The more the graph moves to the left and the top the better it is at classifying. Therefore the blue point marks a perfect classifier.

By doing this the ROC helps by facilitating the decision-making process regarding a

threshold value.

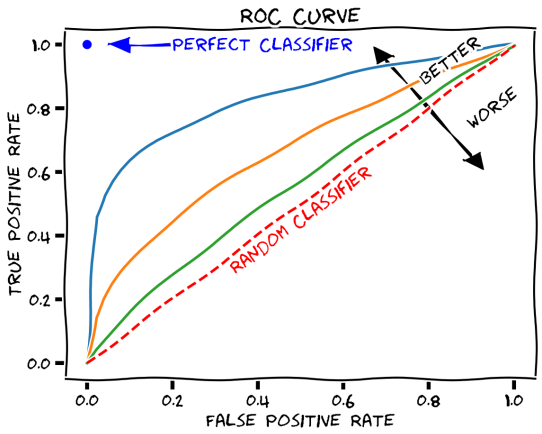

Figure 2.6: **Example ROC curve** showing properties of an ROC curve giving examples of their interpretation.
(Adapted from wikimedia commons, File:Roc-draft-xkcd-style.svg, CC 0)

2.4.3 AUROC curve

Auroc is the abrevation for Area Under the Receiver Operating Characteristic. This metric calculates the area under the curve of ROC. It outputs a scalar as the result of the calculation. Its value range is between zero and one. A classifier that only makes correct classifications achieves an AUROC value of one. This perfect classifier is market with the blue dot in figure 2.6. A score close to zero on points out, that the classifier guesses the wrong way around. It would, for instance classify a present tool as absence and an absence tool as present. A classifier randomly guessing the the classes (red dashed line in figure 2.6) would reach an AUROC score of around 0.5.

Summarized the AUROC curve evaluates the performance of a classifier independent of the threshold and returns a scalar between zero and one. Thereby it shows whether and how accurately a model is able to distinguish between classes.

2.5 normalization

Normalizing images in a dataset is an important step in preprocessing (8). After a dataset of images is normalized, the values of the pixels should be centered around zero and have a standard deviation of one. This is achieved by subtracting the mean value and then dividing the standard deviation:

$$value_{normalized} = \frac{value_{unnormalized} - mean_{all\_values}}{std_{all\_values}} \tag{1}$$

Normalization helps by bringing different values of datasets to one scale to make them compareable. Furthermore, since neural networks execute the dot product for both the forward and backpropagation, non-normalized pixel values can become extremely large. Normalizing the pixel values of images of a dataset correctly can reduce the computation time significantly. The normalized images are normalized with respect to their pixel values. For the human viewer, however, the resulting images appear unnatural in their coloring like shown in Figure 2.7:

- **(A) unnormalized image** in the RGB color space. It only got resized.
- **(B) normalized image** in the RGB color space which got also resized. It looks over- and underlit at the same time.

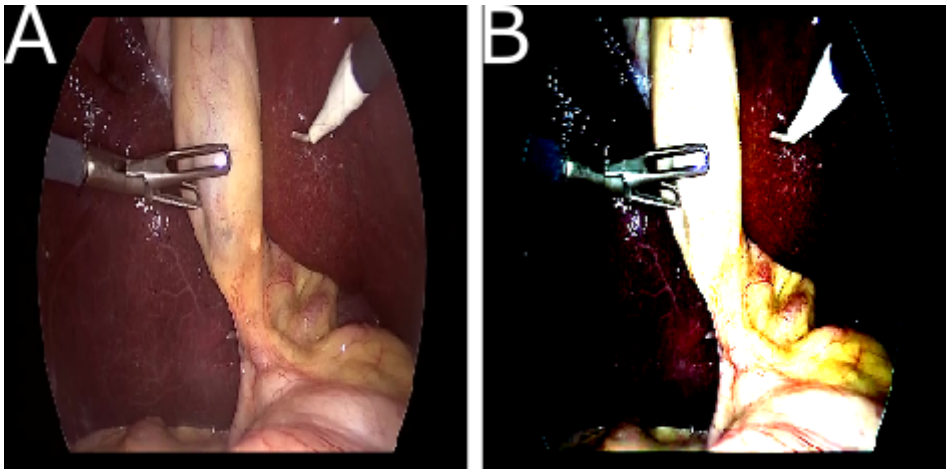

Figure 2.7: **Comparison of normalized vs non-normalized image. (A)** Non normalized but resized image, **(B)** Normalized and resized image

3 Experiments

In this experiments the influence of preprocessing and color encoding of the datasets on image classification models in the context of a surgery shall be investigated. Therefore a model shall classify the presence of a surgical tool in laprascopic frame. Regarding the color encoding the two color spaces HSV and RGB have been chosen. Furthermore there will be two versions of the HSV color space differing from each other in the way they get normalized.

The training, validating and testing of those classifiers will be done on the Cholec80 dataset. Cholec80 is a public dataset containing 80 videos of laparoscopic cholecystectomies and the related labels for tool presence. Once the tools to be classified wether they are in the image or not have been selected, one dataset for each tool gets created. To obtain best possible learning results each dataset gets balanced on the belonging tool.
A simple model, normalization functions and dataset classes were written as part of the experiment preparation.

3.1 Description

The 80 videos of the Cholec80 dataset are randomly divided into training, validation, test set. To avoid data leakage, only whole videos are assigned to the respective sets.The training set consists of 51, the validation set of 13 and the test set of 16 videos.

In total, models are to be trained for four different tools (figure 3.1):

- **(A) Clipper:** The clipper is an instrument that can apply polymer or metal clips, depending on the model. In laparoscopic cholecystectomy, the clips it applies are used to close the cystic artery and the cystic duct. The model used in the Cholec80 dataset uses metal clips.
- **(B) Scissors:** Scissors are used in this operation for adhesiolysis, i.e. surgical loosening of adhesions, and cutting of the artery and ductus cysticus. In contrast to open surgery, scissors are rarely used for tissue dissection in this procedure.

- **(C) Grasper:** The Grasper is an instrument that is similar in function to the forceps used in open surgery. This means that it is used, for example, to grasp and hold organs, tissue, sutures or the specimen bag.
- **(D) Hook:** The hook is a preparation instrument which, with the help of a pedal, can apply monopolar current to separate tissue and coagulate it at the same time. In laparoscopic cholecystectomy, the hook is used for adhesiolysis, preparation of calots triangle, dislodging the gallbladder from the gallbladder bed, and subsequent hemostasis. Hemostasis means the stopping of bleeding.

Since the individual tools are represented differently in the videos depending on what they are used for, this can lead to a strong imbalance. The grasper and the hook are almost balanced in terms of their percentage of presence and absence and have each over 100k of a total of 184498 frames labeled present. That means about 55% presence versus 45% absence.

The clipper on the other hand has a total of about 5.9k frames and the scissors only 3.2k frames. This means, that a classifier, which classifies the clipper always as absent, would get an accuracy of about 96.8% and with constant absence of the scissors even about 98.2%. (c.f. table 3.1)

To counteract this, the dataframes are balanced by omitting frames. A separate dataset is created for each of the instruments whose classification is to be trained, consisting of training, validation and test set. The dimensions of these data frames for each instrument can be seen in the table 3.2.

	grasper		hook		clipper		scissors	
	present	absent	present	absent	present	absent	present	absent
total amount	102588	81910	103106	81392	5986	178512	3254	181244
share (rounded)	55.6%	44.4%	55.9%	44.1%	3.2%	96.8%	1.8%	98.2%

Table 3.1: **Tool distribution over the entire Cholec80 dataset** in absolute numbers and percentages.

In addition, models are trained for each tool in different color spaces. For one trial the frames are read in RGB and normalized. In the next experiment they are read in HSV color space but normalized as if they were read in RGB color space. In the third, there are those that are read in HSV color space and normalized accordingly. This means in this case that the Hue channel is normalized according to its circular nature.

For each tool and color space, models are trained with three different learning rates on a Nvidia GeForce RTX 2080 TI. In total, there are 36 training runs that take between eight

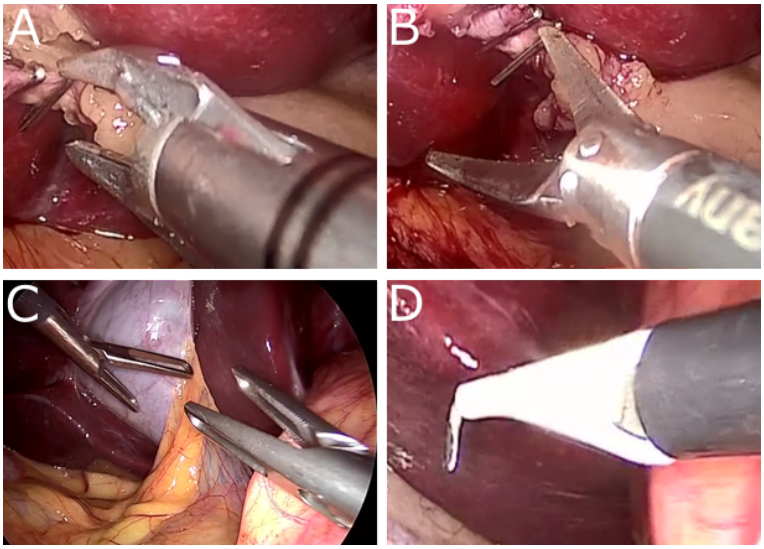

Figure 3.1: **Tools whose classification is to be trained by the models.** Those four images show the tools that where chosen for the classifiers to be trained on. **A** is a clipper, **B** is a scissors, **C** are two graspers and **D** is a hook. All of those images are taken from the videos of the Cholec80 dataset.

minutes for the small datasets (Clipper and Scissors) and up to seven hours for the larger datasets (Grasper and Hook).

	Grasper dataset	Hook dataset	Clipper dataset	Scissors dataset
total amount of frames	163820	162784	11972	6508
train set (share in %)	63.63	65.10	56.23	62.41
validation set (share in %)	16.90	15.74	20.13	14.00
test set (share in %)	19.47	19.16	23.64	23.59

Table 3.2: **Relative size of each dataset.** Includes total number of frames and percentage distribution.

3.2 Implementation

3.2.1 Model

The model used in this thesis was written using the PyTorch Lighning framework.
For this experiment, a CBR tiny was constructed as described by Raghu et al (9) and shown in Figure 3.2. It contains 4 convolutional layers and one fully connected (fc) layer. The

convolutional layers use a 5x5 kernel with a step size of 1 and 0 padding. This is followed by a rectified linear unit (ReLU) as activation function. A ReLU replaces every negative number with a zero. Afterwards a max pooling with a 3x3 spatial window and a step size of 2. Max pooling reduces the size of the given image depending on its window and step size. Each batch is normalized before passing to the next layer. Now it surpasses the fc layer which returns logits that afterwards get converted into a probability using a softmax. Finally the cross entropy loss gets calculated on those probabiities.

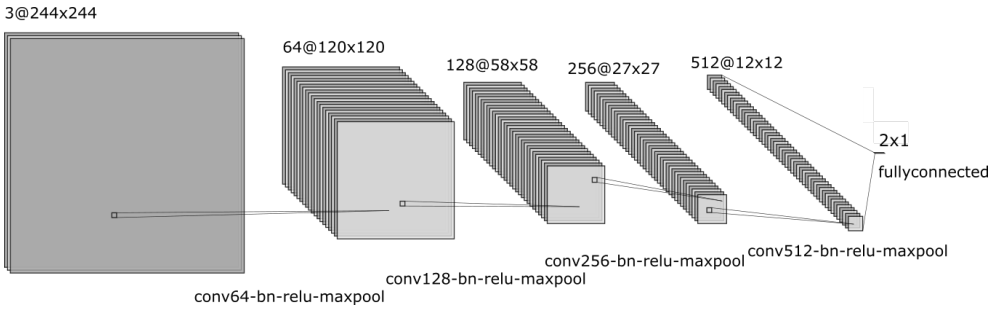

Figure 3.2: **Architecture of the model.** Simplified presentation showing the amount and size of channels on each layer on top. The text on the bottom gives an explanation on the operations being done in that particular layer.
(This figure was created using http://alexlenail.me/NN-SVG/index.html)

The number in front of the @ sign indicates the number of channels in this layer of the network, while the two numbers behind it indicate the width and height. Under the connectors of the different layers, the operations performed from one layer to the next are indicated. On the far right is the last layer, fc, which shows the output of the network.

3.2.2 Normalization

Non circular channels

The normalization of the images was done on each color channel separately. For this purpose a function was written which takes the dataset and the color space as parameters. Within the function a DataLoader, taken from torch.utils.data, was used for the calculation of the mean values and standard deviations for each channel. The DataLoader was assigned the dataset, a batch size of 100 and 4 workers as parameters.

For the calculation of the mean value (μ) the values of all pixels need to be added and divided by the total number of pixels (N) in this channel.

$$\mu = \frac{\sum \mathrm{px}}{\mathrm{N}} \tag{1}$$

For the calculation of the standard deviation (σ) the mean value needs to be substracted from each pixel. Afterwards these are squared and added together. When the number of

pixels in the channel are divided, the variance (σ^2) is obtained. Subsequently taking the square root gives the standard deviation.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{2}$$

This is how the mean value and standard deviation is calculated for all channels of the color spaces RGB and HSVrgb.

TO investigate the influence of the normalization itself another HSV dataset was created and normalized as if it was not circular. This one was given the designation HSVrgb.

Circular channels

For the first channel of the color space HSV a different procedure had to be chosen because of its circular structure. Instead of starting with the calculation of a mean value, a mean angle ($\mu angle$) is calculated. For this purpose the pixel values were transformed into complex numbers with the help of NumPy in order to calculate a mean vector.

$$\mu\text{angle} = \text{angle}\left(\frac{\sum_k^N e^{2i\pi\text{channel}}}{N}\right) \tag{3}$$

This mean vector was then used to center the pixel values of the circular channel around 0.

$$\text{centered circular channel} = angle(e^{2i\pi channel - i\mu angle}) \tag{4}$$

After centering this channel can be normalized as if it was a non circular channel.

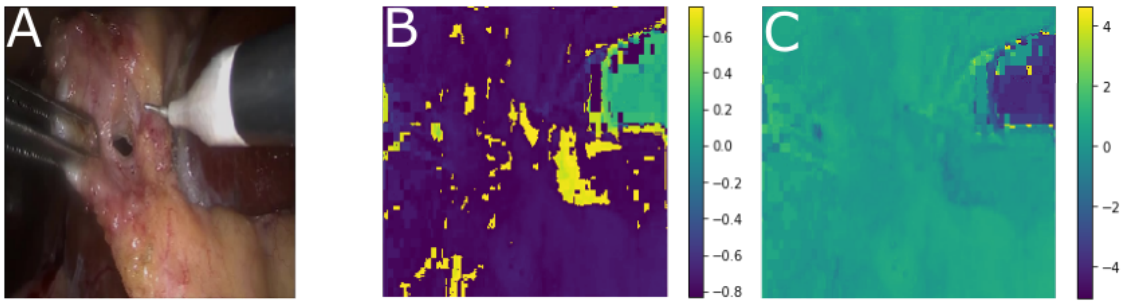

Figure 3.3: **Differences between normalization procedures on the H channel.** All three pictures are resized. (A) unnormalzed image, (B) image of hue channel normalized as if it was not circular, (C) image of hue channel normalized circular. The Colorbar beneath image (B) and (C) shows the the pixel values of the images.

The differences resulting out of the differing normalization procedures can be seen in Figure 3.3. The trained eye immediately recognizes grasper, hook, gallbladder and liver in image A. This one contains the original Image which got just resized. Images B and C, on the other

hand, make it much more difficult, if not impossible, to recognize something.
Image B shows the result of channel H that was normalized ignoring its circular nature. It consists mostly of yellow and purple pixels.
The last one, image C is the result of a normalization of the H channel adapted to the circular structure.

3.2.3 Training

For the training the Pytorch Lightning Framework was used. During the training, the accuracy, the loss and the AUC ROC per batch were determined and stored for the evaluation.
The average loss per validation epoch was used as termination criterion. For this, the early stop callback from the Pytorch Lightning framework was used with a patience of 30. The patience specifies how many epochs a loss value may increase before the training is terminated. The value of 30 means that after the lowest measured loss value the training will continue for another 30 epochs to see if the value can be undercut. Thus, no maximum number of training epochs is set in advance.

3.3 results

In a direct comparison of the learning rates during training, the model trained on data sets in the RGB color space shows better values in the area of validation accuracy, validation loss and AUROC than the other two models. In the small data sets of scissors and clipper, this advantage is more pronounced than in the larger data sets. The model trained in the HSV color space is again more successful than the HSVrgb model in almost all learning rates. The gap to the RGB models is smaller than that to the HSVrgb models.

3.3.1 results of best models

Table 3.3 shows the AUROC and accuracy values obtained when the best models were applied to the test sets. Considering the the cells with the bold values one sees the the highest scores achieved for each tool. The RGB classifier has got the best scores for three out of the four tools. Those three are scissors, clipper and grasper. On the hook dataset the HSV classifier achieved the highest score.

Unsurprisingly, classification models trained on much smaller datasets achieve lower AUROC and accuracy values. Especially when considering the two small datasets, the RGB classifier stands out against the HSV and the HSVrgb classifier. (cf. 3.3) For example, the RGB AUROC value for the scissors datasets with its 6508 images, is 0.7771. This value is significantly better than those achieved by the other two models. HSVrgb came to 0.7054

and HSV even only to 0.6565. It was the same for the accuracy. The HSV model achieved an accuracy of 0.6987 on the smallest dataset and thus a higher value than HSVrgb with an accuracy value of 0.6396. The HSV classifier performed significantly worse on this dataset. This reached a value of 0.5880.

Looking at the larger dataset hook with its 162784 images, the achieved values of the classifiers are much closer together. All three classifiers achieve an AUROC score of more than 0.98 on this dataset. Also in terms of accuracy the best values of the experiment are achieved with 0.9398 (HSVrgb), 0.9468 (RGB) and 0.9505 (HSV). The HSV classifier is the one with the highest values in this case.

The grasper, which with 163820 images is a dataset of comparable size to the hook dataset, nevertheless achieves significantly worse values. The best performing classifier is again the RGB classifier with an AUROC of 0.8384 and an accuracy of 0.7579. Just below is HSV with an AUROC of 0.8275 and an accuracy of 0.7483. The HSVrgb classifier with 0.7688 AUROC and 0.6963 accuracy is much worse. (cf. table 3.3)

		AUROC			Accuracy		
	amount of frames	HSV	RGB	HSVrgb	HSV	RGB	HSVrgb
scissors	6508	0.6565	**0.7771**	0.7054	0.5880	**0.6987**	0.6396
clipper	11972	0.7584	**0.8212**	0.8050	0.6641	**0.7460**	0.6469
hook	162784	**0.9855**	0.9835	0.9827	**0.9505**	0.9468	0.9398
grasper	163820	0.8275	**0.8384**	0.7688	0.7483	**0.7579**	0.6963

Table 3.3: **Results of the tested models on each of the four tools.** Sorted first by AUROCand Accuracy. Afterwards by the color spaces. Total amount of frames gives the number of frames each dataset contains in total. The bold written scores show the best results for a AUROC and accuracy for each tool.

The confusion matrices shown in figure 3.5, figure 3.7, figure 3.6 and figure 3.4 and evaluate the performance of the different classifiers. Each confusion matrix displays the result related to one color space and one tool. The color spaces are ordered from left to right. Beginning with HSV, followed by RGB and HSVrgb. The datasets are ordered top down. In the first row are the scissors datasets followed by the clipper, the grasper and the hook datasets.

The y score threshold those matrices got calculated on is 0.5.

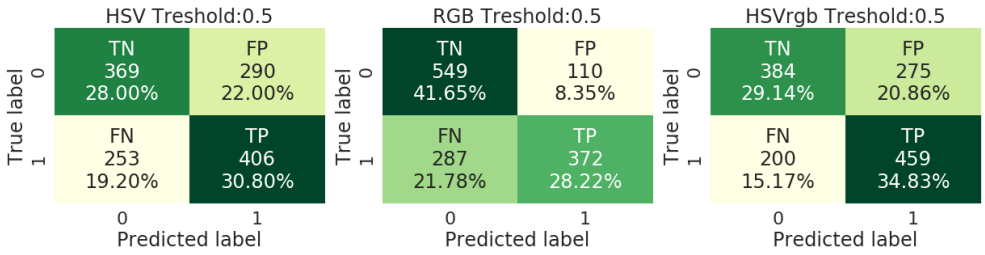

Figure 3.4: **Confusion matrices: scissors.** Comparing the ground truth with the predicted labels of the three scissors dataset classifiers. From left to right are the color spaces HSV, RGB and HSVrgb. The threshold for these matrices is 0.5.

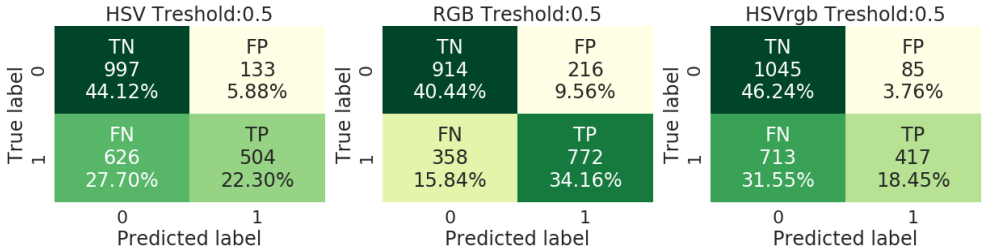

Figure 3.5: **Confusion matrices: clipper.** Comparing the ground truth with the predicted labels of the three clipper dataset classifiers. From left to right are the color spaces HSV, RGB and HSVrgb. The threshold for these matrices is 0.5.

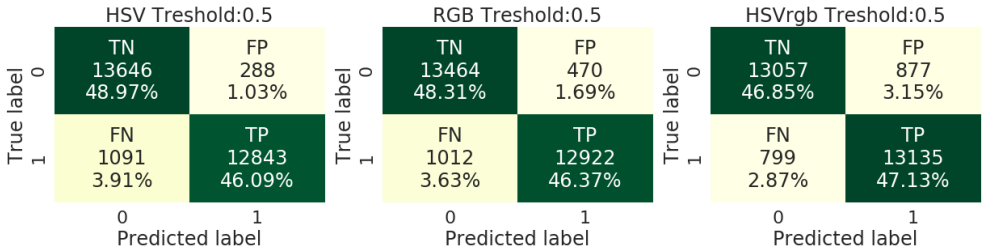

Figure 3.6: **Confusion matrices: hook.** Comparing the ground truth with the predicted labels of the three hook datasets classifiers. From left to right are the color spaces HSV, RGB and HSVrgb. The threshold for these matrices is 0.5.

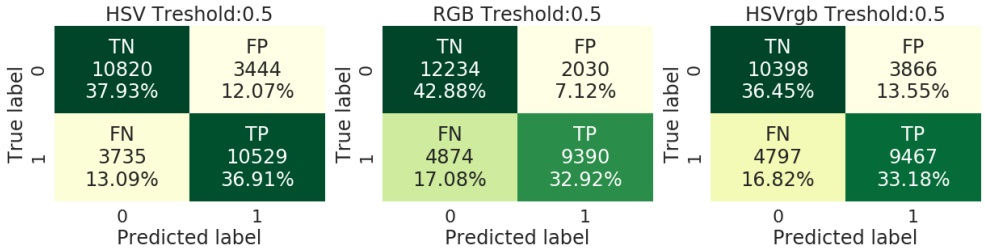

Figure 3.7: **Confusion matrices: grasper.** Comparing the ground truth with the predicted labels of the three grasper dataset classifiers. From left to right are the color spaces HSV, RGB and HSVrgb. The threshold for these matrices is 0.5.

3.3.2 Example frames of false classified frames

Besides the actual performance of a classification model, the evaluation of other factors is also of great importance. Checking upon images with wrong predictions resulting in false positives or negatives can be a first step. By doing this one can see possible reasons for the false classification. There might be, for instance, something on the image that distracts the classifier. Another possible reason would be wrong labels at some of the images resulting in false positives or negatives. Therefore it is quite useful to have at least a look at those false positives and negatives with high confidence scores on the side of the classifier as those are the ones with the highest loss.

Examples of most confident False Positives

Following are some examples of the false positive classified frames with the highest confidence per tool and color space. Each of the figures shows three results of each color space for one tool. From left to right are shown RGB, HSV and HSVrgb. One can find the ground truth label followed by the predicted y score of the classifier on top of each image shown in the figures.

When looking at the figures, it is important to remember that they make up only a tiny amount of the datasets. Also, if in a figure all images have wrong labels or in a figure none of the models has made a correct prediction, this does not mean that the whole dataset is correct.

Figure 3.8 shows some of the FP predictions achieved by each classifier on the scissors dataset. Different tools can be seen on the images. The ground truth labels appear to be correct on all images occurring in this figure.

The FP predicted clippers in figure 3.9 again show different tools. None of them is the clipper. Also in this figure all ground truth labels are assigned correctly. It is noticeable that often the shafts of the tools are in the picture.

The predictions made for the hook and shown in figure 3.10 seem to be correct contrary to the ground truth labels. There is a hook on each of the images.However, often only the end of the tool can be seen.
Similar to figure 3.10, labels in figure 3.11 seem to be set incorrectly. The tip of the grasper cannot be seen in its entirety in any of the images.

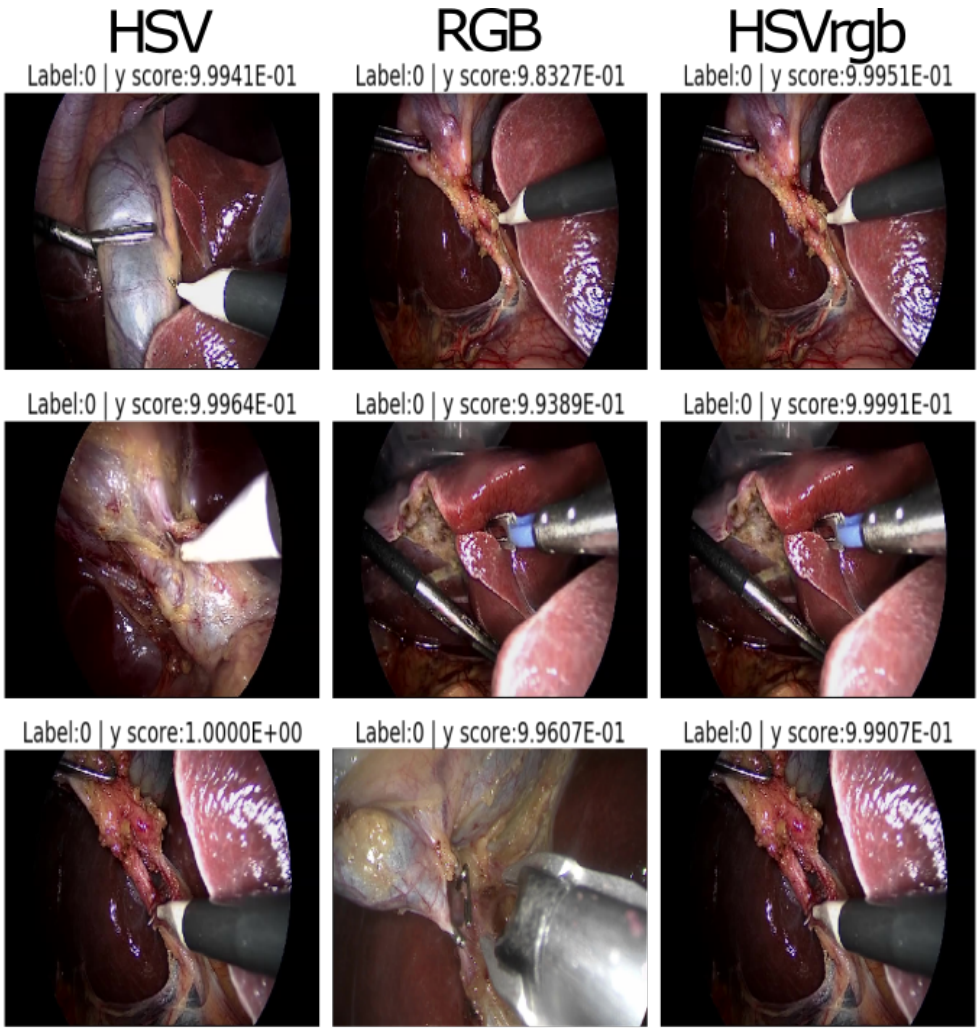

Figure 3.8: **Scissors: FP examples for each color space.**

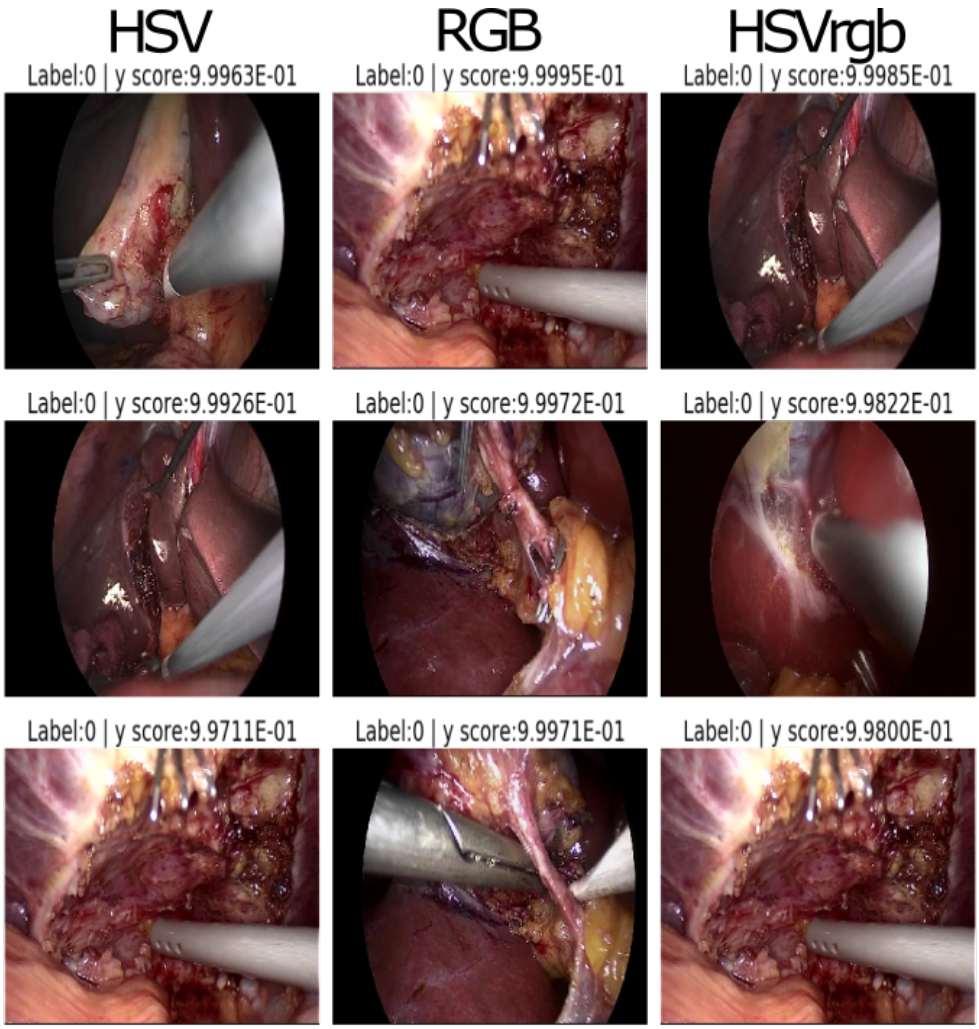

Figure 3.9: **Clipper: FP examples for each color space.**

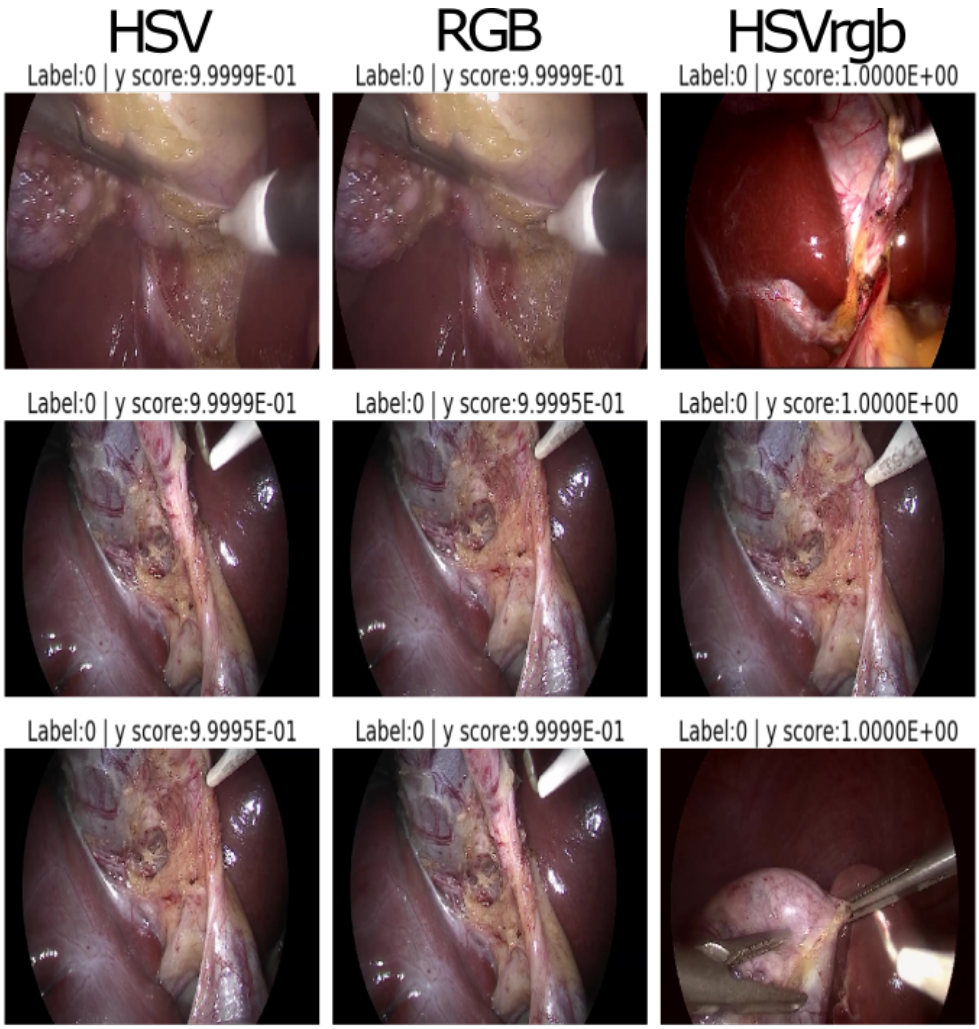

Figure 3.10: **Hook: FP examples for each color space.**

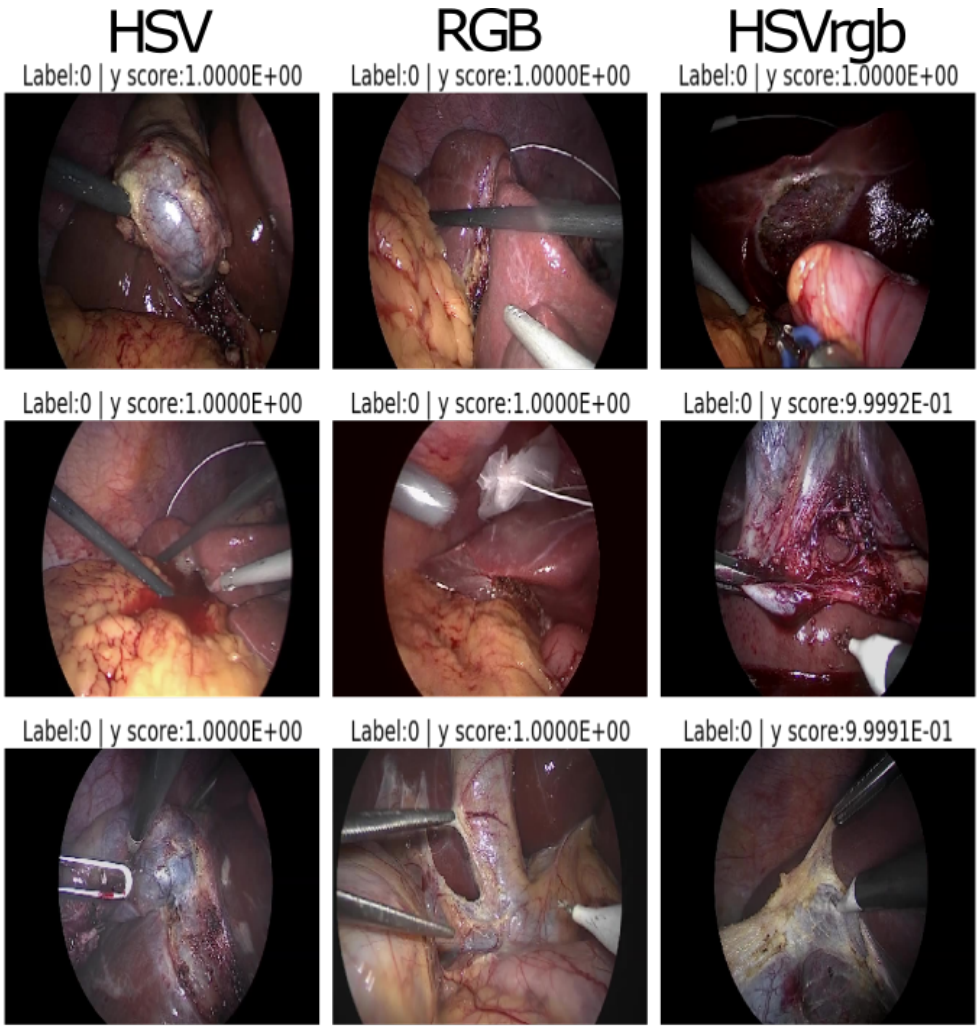

Figure 3.11: **Grasper: FP examples for each color space.**

Examples of most confident False Negatives

The following figures show examples of false negative predicted images. Those are again the ones with the highest confidence. The images inside these figures are arranged the same way the false positives are.
The examples shown in Figure 3.12 reveal scissors to a greater or lesser degree in each of the images. The labels seem to be correct contrary to the predictions. In some cases, much of the shaft of the scissors is visible (row 1 of HSV and RGB column) and in others the tip (row 2 RGB column) or all but the tip is obscured by tissue.

The FN examples shown in Figure 3.13 all appear to contain at least part of the clipper. In many of the images, either the tip or all but the tip is obscured or out of frame.

The hook in figure 3.14 is present in all of the images. In some cases it is covered by tissue. In others, only the front part of the tip is visible.

In some of the FN examples of the grasper in Figure 3.15 a grasper is visible. In other cases images only contain a fraction of the tool.

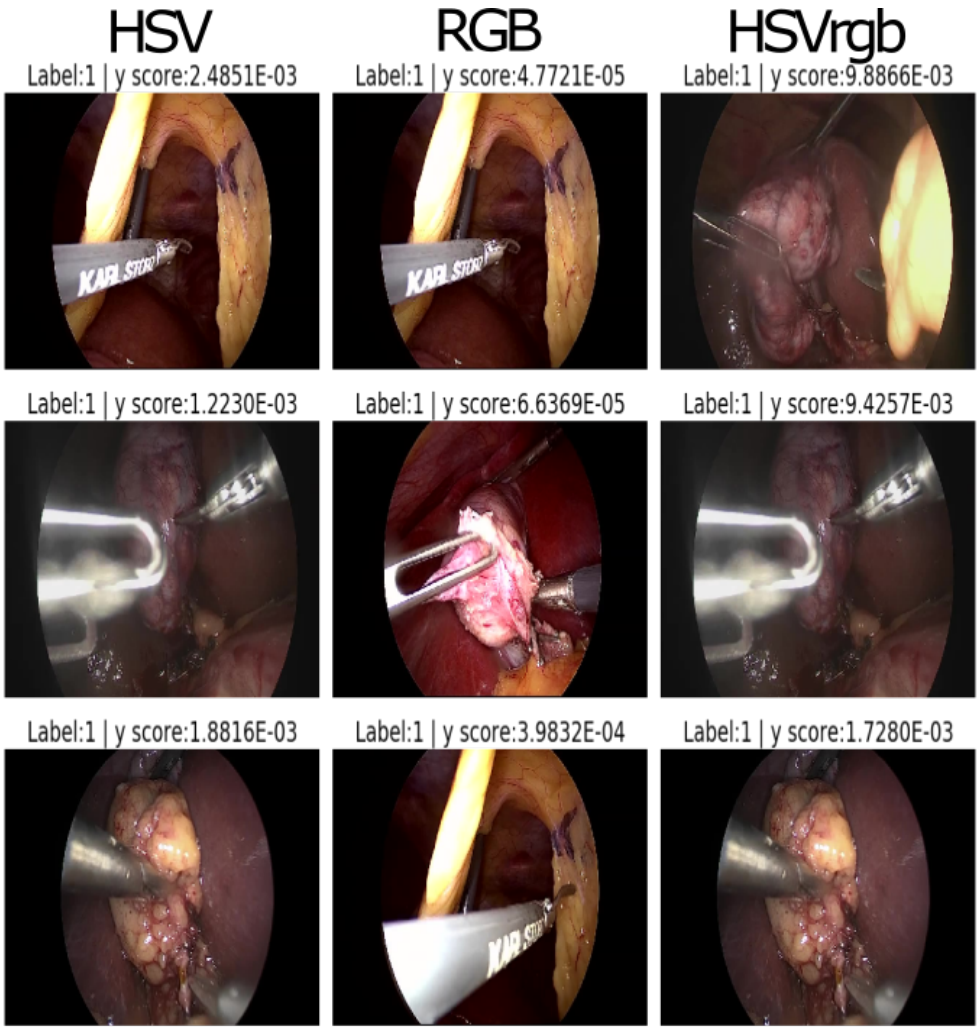

Figure 3.12: **Scissors: FN examples for each color space.**

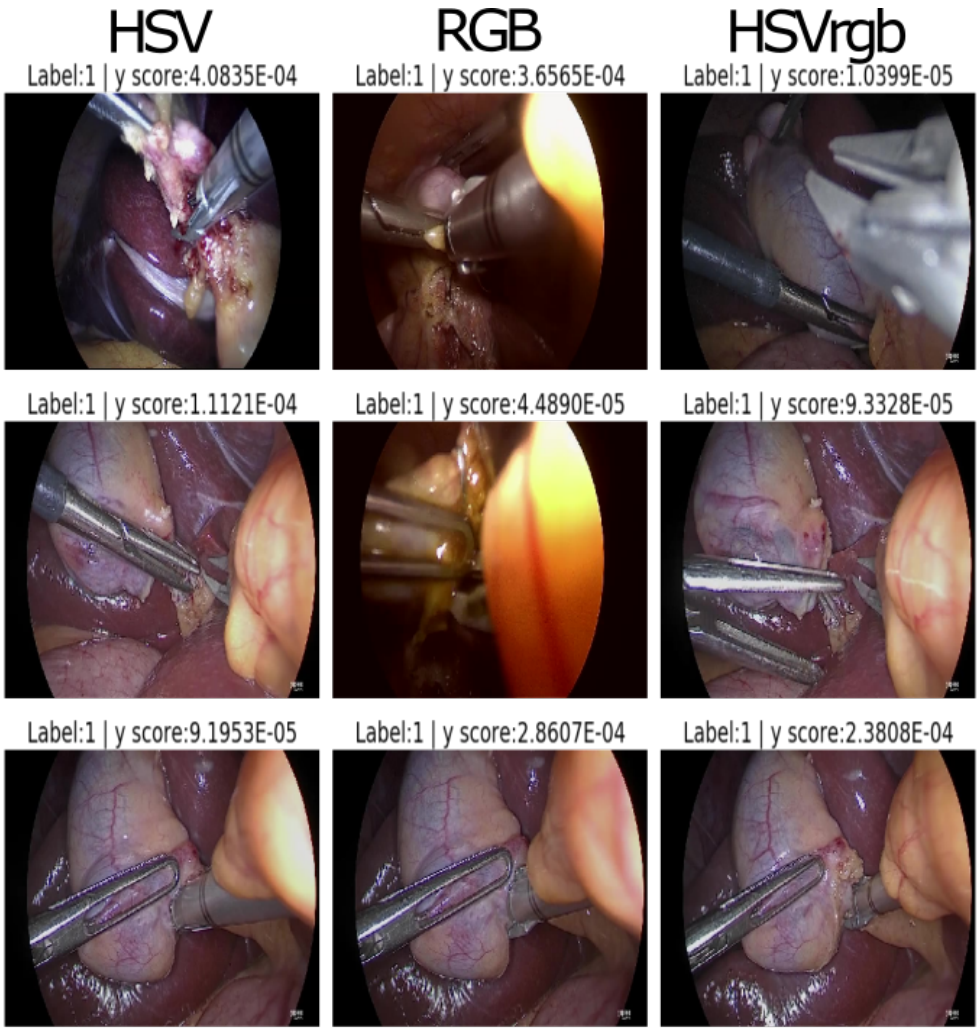

Figure 3.13: **Clipper: FN examples for each color space.**

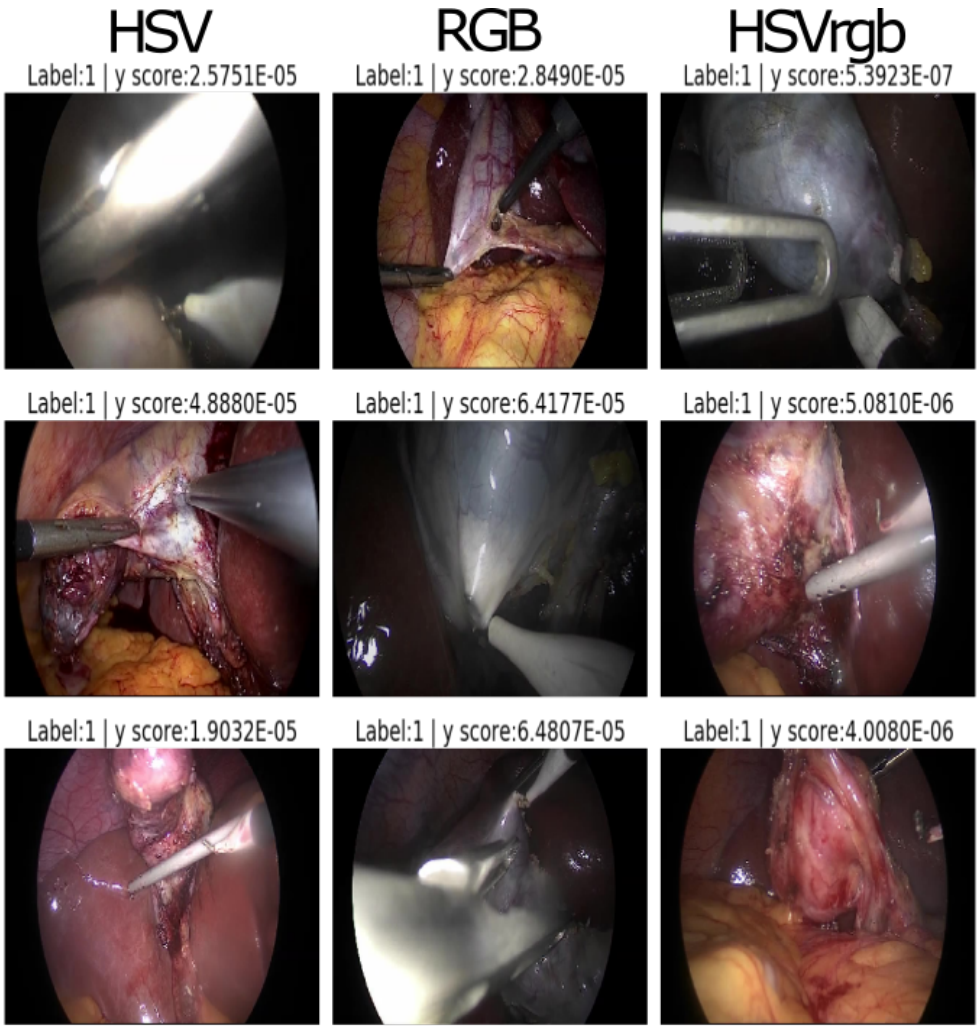

Figure 3.14: **Hook: FN examples for each color space.**

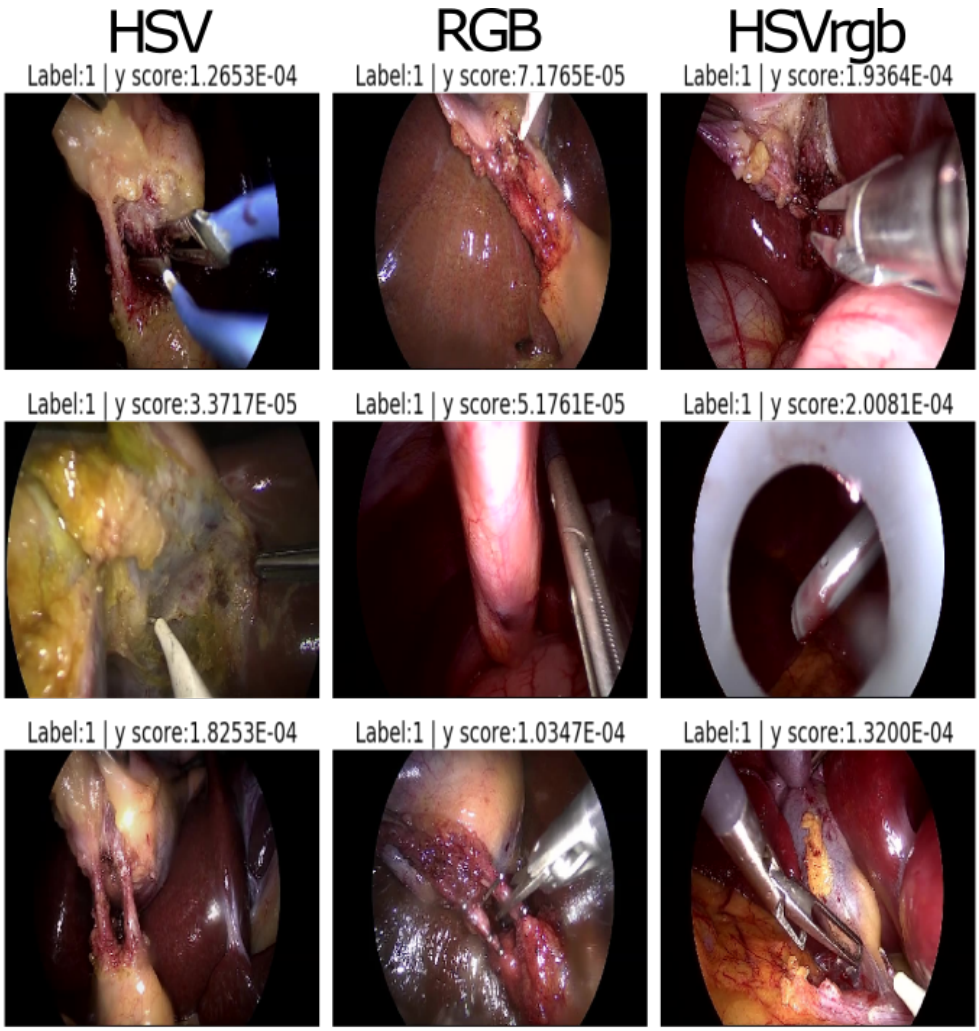

Figure 3.15: **Grasper: FN examples for each color space.**

3.4 Interpretation

Comparing the results of the model testings one notices that the current standard color space for deep learning, RGB, does in deed perform better than its two challengers. Achieving higher scores regarding accuracy and AUROC on three of four tools it clearly stands out. It seems that the RGB model does work more robust even on smaller datasets as its lead is the biggest for those. Comparing RGB and HSV regarding the two bigger datasets they achieve almost similar results. The HSVrgb which the hue channel being normalized like it was not circular does perform better than HSV regarding the small datasets. Though it is not as good as RGB. When it comes to the hook dataset it has similar AUROC scores as the other two classifiers but stands a bit back regarding the accuracy. Interestingly it performs much worse than HSV and RGB on the grasper dataset.

When it comes to the two larger datasets the HSV model can definitely catch up with the

RGB model. Regarding the second largest dataset, the hook, all three classifiers achieve similar high scores.
One possible reason for the much better performance of all models on the hook dataset could be that the hook as such is easier to classify than the other tools. Unlike the other three tools, this one has white rather than silver at the tip. Furthermore, unlike the hook, the other three tools have two tips when open which can be seen in figure 3.1

Regarding the differing performance of the three classifiers on the two small datasets an option could be that possible false set labels as shown in 3.10 do have a much stronger impact on datasets with a much smaller pool of images. Even a few predictions classified as false positive or false negative increase the loss of the model clearly and thereby decrease the accuracy. One solution approach would be to iterate trough the false positive and false negative classified images with high prediction values. While having a look at those images one can check for possible reason of the false classification. If the ground truth has some false set labels those might be corrected. Also other possible reasons for errors can be investigated that way.

The figures showing FPs and FNs in some cases have the exactly same images for the different color spaces. Examples of those can for instance be found in figure 3.8 row 1 and 2 of the RGB and HSVrgb columns or row 1 of the HSV and RGB column and row 3 of the HSV and HSVrgb column of figure 3.12. This fact is quite interesting as it shows that the three classifiers seem to stumble, at least partly, over the same images.

Some of the FP and FN classified predictions raise questions about the definition of the labels. A tool is labeled as present when at least half of its tip is visible. (7) Where the tip starts or ends is not mentioned. For example, figure 3.10 shows the hook in different orientations with different amounts of the tip visible. At least some of these are wrong according to my understanding.

In the image in the 2nd row of the HSVrgb classifier in in figure 3.15, only a silver reflection around the upper edge of the trocar is visible from the grasper. The tip of the clipper in the 3rd row of the HSV classifier in figure 3.13 is completely covered by tissue. Both tools are still visible according to ground truth.

The smaller the dataset on which the models are to be trained, the more important it is thatthe data of the dataset is well cleaned.

4 Discussion

Exploring the possibilities of increasing the prediction accuracy of classifying models used in the operating theatre and thereby increase the safety of patients is an extremely important and necessary. Therefore within this thesis the influence of color spaces on deep learning models for image classification was investigated. Several models have been trained on different tools and using different color encoding and normalizations.

The normalization of the HSV coded images proved to be difficult. Since the Hue channel is circular, normalization as one would do for each channel of the RGB images does not produce the same satisfactory results. One of the reasons for this is the combination of the color distribution of the surgical images and the Hue channel. The images are all very red-heavy. Red is again exactly the color where the circle of the Hue channel is split. Thus, both at 0 and 360 degrees is red.

To investigate the influence of this factor, two datasets were created for the HSV color space. One of them was normalized as if it were three linear channels. This was given the designation HSVrgb. The other dataset was normalized considering the circular structure of the Hue channel. It was this normalization that was to prove challenging. Unlike for RGB, there are no standard libraries for image normalization in this color space. For the normalization of this channel complex numbers were used. The implementation of this normalization function took a relatively long time.

During the training it became apparent how important balanced datasets are for the actual learning success of a classifier. When training the tools scissors or clippers on an unbalanced dataset, accuracy values of well over 90% were achieved (cf. table 3.1). However, because of the small number of these two tools in relation to the total number of images in the datasets, it was found that stubbornly predicting them as absent was sufficient to produce such high values. Thus, the model did not necessarily learn the classification of the images. Therefore, for each of the four selected tools, a separate dataset was created, balanced with respect to the tool.

By balancing the datasets the achieved results are not representative for normal surgical image classification. Though this was needed to compare the results and performance of the different color spaces. That was also the reason to choose different tools. Nevertheless the whole setup is extremely specific. Therefore changing the surgical department or even the procedure might light to completely different results.

4.1 Conclusion

In this work, the influence of color spaces on image classifiers was investigated for a very specific use case. It was found that color spaces do have an impact and performance can be influenced by the choice of color space and its normalization. Furthermore, it was found that the RGB color space performs better than the HSV color space, which is fair since it is the most commonly used. Especially for small datasets, this color space seems to be more robust.

The fact that despite the special use case the best results were achieved by the RGB model shows that RGB is efficient as a standard color space for Deep Learning models.

However, the results of this experiment have no general validity due to their specific orientation. It may be that changing the surgical procedure will produce different results. Also the use of other color spaces could bring further interesting results.

4.2 Outlook

Overall, the investigation of the effect of a color space on a deep learning algorithm for classifying surgical images worked as intended. Even if the results are not what was hoped for. When switching the color space in preprocessing it is of importance to normalize accordingly. The classification model using HSVrgb can neither keep up with RGB on the smaller datasets nor with RGB and HSV on the larger ones.

The classifier using HSV could not stand its ground against the one using RGB regarding small datasets with a few thousand images. When it comes to the larger datasets with more than 150k images the HSV classifier is able to keep up with the RGB classifier. It can even achieve better results for one the two large datasets. Though its lead is not of any significance regarding accuracy and AUROC score. One can not tell if a bigger classification model bigger datasets or an adapted normalization function would increase the gap. Whether or HSV or RGB.

The tight time frame made it impossible for me to investigate different model architectures

including a lot more complex ones. Considering the influence the dataset size has on the HSV model regarding its performance it would be quite interesting to investigate this behavior on much larger sets with a variation of different models being trained on various different color spaces. The RGB color space seems to be more robust against small datasets. Also the normalization process, especially of the hue channel and its correlation to the saturation channel deserve a further investigation.

Bibliography

[1] Stefanie Speidel, Michael Delles, Carsten Gutt, and Rüdiger Dillmann. Tracking of instruments in minimally invasive surgery for surgical skill analysis. pages 148–155, 2006. doi: 10.1007/11812715_19.

[2] David Bouget, Max Allan, Danail Stoyanov, and Pierre Jannin. Vision-based and marker-less surgical tool detection and tracking: a review of the literature. *Medical Image Analysis*, 35:633–654, jan 2017. doi: 10.1016/j.media.2016.09.003.

[3] Shreyank N Gowda and Chun Yuan. Colornet: Investigating the importance of color spaces for image classification. *arXiv*, 2019.

[4] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. de Mathelin, and N. Padoy. Endonet: A deep architecture for recognition tasks on laparoscopic videos. *IEEE Transactions on Medical Imaging*, 36(1):86–97, 2017. doi: 10.1109/TMI.2016.2593957.

[5] Johanna Ronsdorf. Microsoft erklärt: Was ist deep learning? definition & funktionen von dl, 2020. URL `https://news.microsoft.com/de-de/microsoft-erklaert-was-ist-deep-learning-definition-funktionen-von-dl/`.

[6] Nicola Jones. Computer science: The learning machines, 2014. URL `https://www.nature.com/news/computer-science-the-learning-machines-1.14481`.

[7] A.P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. de Mathelin, and N. Padoy. Cholec80 dataset, 2016. URL `http://camma.u-strasbg.fr/datasets`.

[8] Sylvain Gugger Jeremy Howard. *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media, Inc., 2020.

[9] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. *arXiv*, 2019.

A1 Video distribution

A1.1 Train set

video01	video22	video40	video56	video70
video02	video24	video41	video57	video72
video04	video25	video42	video58	video73
video08	video27	video44	video59	video74
video10	video28	video45	video60	video75
video13	video31	video46	video63	video76
video14	video33	video47	video66	video77
video16	video35	video48	video67	video78
video17	video36	video51	video68	video79
video18	video38	video54	video69	video80
video21				

Table A1.1: **Videos that are part of the test set.**

A1.2 Validation set

video05	video11	video26	video53	video61
video07	video12	video49	video55	video65
video09	video15	video50		

Table A1.2: **Videos that are part of the validation set.**

A1.3 Test set

video03	video23	video32	video39	video62
video06	video29	video34	video43	video64
video19	video30	video37	video52	video71
video20				

Table A1.3: **Videos that are part of the test set.**

A2 Source code

A2.1 Modules

Listing A2.1: Imports

```
1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as F
4  import torch.optim as optim
5  import torchvision
6  import matplotlib.pyplot as plt
7  import pandas as pd
8  import re
9  import cv2
10 import albumentations
11 import albumentations.pytorch
12 import time
13 import numpy as np
14 import pytorch_lightning as pl
15 import neptune
16 import time
17 from tqdm import tqdm
18 from pytorch_lightning.callbacks import ModelCheckpoint
19 from pytorch_lightning.metrics.functional import auroc
20 from pytorch_lightning.metrics.functional import accuracy
21 from pathlib import Path
22 from torch.utils.data import Dataset
23 from torch.utils.data import DataLoader
24 from torch.utils.data import random_split
25 from pytorch_lightning.callbacks.early_stopping import EarlyStopping
```

Listing A2.2: Simple Dataset

```
1  class SimpleDataset(Dataset):
2      '''
3      Creates Dataset without any normalisation procedure beeing done.
4      This one is needed to calculate normalization values.
```

```
    '''
    def __init__(self, df, cs='RGB'):
        '''
        :param cs: Colorspace in which the images shall be loaded. Default value is 'RGB'.

        :return: Dataset of wanted color space, unnormalized.
        '''
        self.img_path = ('/workspace/shared_data/cholec80/frames/'+df.video.astype(str)+'/'
                         +df.video.astype(str)+df.frame.apply(lambda f: f'_{f:06d}.jpg')).to_numpy()
        self.label = df.label.to_numpy().astype(int)
        self.resize_fn = albumentations.Resize(244,244)
        self.colorspace = cs

    def _resize_img(self, X):
        X = self.resize_fn(image = X)['image']
        return X

    def __getitem__(self, index):
        X = cv2.imread(self.img_path[index])
        if self.colorspace == 'RGB':
            X = cv2.cvtColor(X, cv2.COLOR_BGR2RGB)
        else:
            X = cv2.cvtColor(X, cv2.COLOR_BGR2HSV)
        X = self._resize_img(X)
        X = albumentations.pytorch.ToTensorV2()(image = X)['image']
        X = X.float()
        if self.colorspace == 'RGB':
            X = torch.div(X, 255)
        else:
            X[0] /= 179
            X[1:] /= 255
        y = self.label[index]
        return X, y

    def __len__(self):
        return len(self.label)
```

Listing A2.3: Image dataset

```
class ImgDataset(Dataset):
    '''
    Creates image dataset out of given csv.
    '''
    def __init__(self, df, norm_vals, cs='RGB', normalize=True, rgb_like = False):
        '''
        :param df: csv that shall be converted to image dataset
        :param cs: Colorspace in which the images shall be loaded. Default value is 'RGB'.
```

```
        :param normalize: True if dataset shall be normalized. Default value is True.

        :return: Dataset, normalized on given colorspace if wanted.
        '''
        self.img_path = ('/workspace/shared_data/cholec80/frames/'+df.video.astype(str)+'/'
                         +df.video.astype(str)+df.frame.apply(lambda f: f'_{f:06d}.jpg')).to_numpy()
        self.label = df.label.to_numpy().astype(int)
        self.resize_fn = albumentations.Resize(244,244)
        self.colorspace = cs
        self.normalize = normalize
        self.means, self.stds = norm_vals
        self.rgb_like = rgb_like

        # normalize on given colorspace if normalize parameter is set to True.
        if normalize:
            if self.colorspace == 'RGB':
                self.normalize_fn = self.normalization_RGB_like_fn(self.means, self.stds)
            elif self.colorspace == 'HSV':
                if rgb_like:
                    self.normalize_fn = self.normalization_RGB_like_fn(self.means, self.stds)
                else:
                    self.normalize_fn = self.normalization_H_circular_fn(self.means, self.stds)
            else:
                print('Wrong input. Default colorspace used.')
        else:
            self.normalize_fn = albumentations.Normalize([0, 0, 0], [1, 1, 1])
        print(f'__init__ \n - normalized : {self.normalize}\n - colorspace : {self.colorspace}\n -
            rgb_like : {rgb_like}')

    def normalization_RGB_like_fn(self, means, stds):
        def fn(image):
            image = torch.div(image, 255.)
            image[0,:,:] = (image[0,:,:] - means[0])/stds[0]
            image[1,:,:] = (image[1,:,:] - means[1])/stds[1]
            image[2,:,:] = (image[2,:,:] - means[2])/stds[2]
            return {'image':image}
        return fn

    def normalization_H_circular_fn(self, means, stds):
        def fn(image):
            image[0,:,:] = torch.div(image[0],179.)
            image[1:,:,:] =torch.div(image[1:],255.)
            H_centered = np.exp((2 * 1.j * np.pi * image[0,:,:].numpy() - 1j * means[0].numpy()))
            H_centered = np.angle(H_centered)
            image[0,:,:] = H_centered/stds[0]
            image[1,:,:] = (image[1,:,:] - means[1])/stds[1]
            image[2,:,:] = (image[2,:,:] - means[2])/stds[2]
```

```
            return {'image':image}
        return fn

    def __call__(self, cs):
        self.colorspace = cs
        print(f'__call__ with {self.colorspace}')

    def _normalize_img(self, X):
        X = self.normalize_fn(image = X)['image']
        return X

    def _resize_img(self, X):
        X = self.resize_fn(image = X)['image']
        return X

    def __getitem__(self, index):
        X = cv2.imread(self.img_path[index])
        X = self._resize_img(X)
        if self.colorspace == 'RGB':
            X = cv2.cvtColor(X, cv2.COLOR_BGR2RGB)
        else:
            X = cv2.cvtColor(X, cv2.COLOR_BGR2HSV)
        X = albumentations.pytorch.ToTensorV2()(image = X)['image'].type(torch.float)
        X = self._normalize_img(X)
        y = self.label[index]
        return X, y

    def __len__(self):
        return len(self.label)
```

Listing A2.4: Balanced dataset creation

```
def create_balanced_dataframes(unbalanced_df, id_list=[], tool='clipper', span=[0,0],
    small_df_sample_size=0):
    '''
    Returns a version of given dataframe which is balanced regarding the given tool (which should be one
        of the columns).
    It might be chosen to create a smaller dataset out of the balanced. If no argument regarding that is
        given the
    mentioned dataframe gets returned.

    :param unbalanced_df: Dataframe that shall be taken a balanced sample from
    :param id_list: List of video ids from that the dataframes shall be created.
    :param tool: Tool on which the dataframe shall be balanced on. The tool has to be one of the columns
        of the dataframe.
    :param span: Range of dataframe that shall be created
    :param small_df_sample_size: Desired size of smaller dataframe. No smaller Dataframe gets created if
```

```
        0. Default is 0.
12
13      :return: either one or two dataframes depending on given parameters
14      '''
15      df = pd.DataFrame()
16      start, end = span
17
18      # filter unique video names to work with whole videos while balancing dataset.
19      video_ids = id_list if id_list != [] else unbalanced_df.video.unique()
20
21
22      if start < end:
23          for i in range(start, end):
24              df = df.append(unbalanced_df[unbalanced_df['video'] == video_ids[i]])
25          #checks if there is more ones or zeros in column with tools name
26          more_less=(1,0) if (df[tool].values == 1).sum() > (df[tool].values == 0).sum() else (0,1)
27          bal_df = df[df[tool] == more_less[1]]
28          bal_df = bal_df.append(df[df[tool] == more_less[0]].sample(n=len(bal_df)))
29          # dataset gets shuffeld via .sample(frac=1) to avoid batches including ones XOR zeros
30          bal_df = bal_df.sample(frac=1)
31      else:
32          for i in range(len(df)):
33              df = df.append(unbalanced_df[unbalanced_df['video'] == video_ids[i]])
34          #checks if there is more ones or zeros in column with tools name
35          more_less=(1,0) if (df[tool].values == 1).sum() > (df[tool].values == 0).sum() else (0,1)
36          bal_df = df[df[tool] == more_less[1]]
37          bal_df = bal_df.append(df[df[tool] == more_less[0]].sample(n=len(bal_df)))
38          # dataset gets shuffeld via .sample(frac=1) to avoid batches including ones XOR zeros
39          bal_df = bal_df.sample(frac=1)
40
41      bal_df['label'] = bal_df[tool]
42
43      if small_df_sample_size != 0:
44          bal_small_df = pd.DataFrame()
45          bal_small_df = df[df[tool] == 1].sample(n = small_df_sample_size)
46          bal_small_df = bal_small_df.append(df[df[tool] == 0].sample(n = small_df_sample_size))
47          # dataset gets shuffeld via .sample(frac=1) to avoid batches including ones XOR zeros
48          bal_small_df = bal_small_df.sample(frac=1)
49          return bal_df, bal_small_df
50      else:
51          return bal_df
```

Listing A2.5: Normalization value calculation

```
1   def calc_normalisation_values(dataset, rgb_like=True, batch_size=100, num_workers=4):
2       '''
3       calculates mean and standard difference for each picture channel of given dataset.
```

```
4
5       :param dataset: Dataset on which normalisation values shall be calculated.
6       :param rgb_like: False if first channel shall be normalized circular. Default is True.
7       :param batch_size: Batch size for DataLoader. Default is 100.
8       :param num_workers: Num of workers for DataLoader. Default is 4.
9
10      :return: Dictionary containing mean and std for each channel.
11      '''
12
13      loader = DataLoader(dataset, batch_size=batch_size, num_workers=num_workers)
14
15      if rgb_like:
16          num_of_px = len(dataset) * 244 * 244
17          sum_chan0 = sum_chan1 = sum_chan2 = 0.
18          for imgs, labels in loader:
19                  sum_chan0 += imgs[:,0,:,:].sum()
20                  sum_chan1 += imgs[:,1,:,:].sum()
21                  sum_chan2 += imgs[:,2,:,:].sum()
22          mean_r = sum_chan0 / num_of_px
23          mean_g = sum_chan1 / num_of_px
24          mean_b = sum_chan2 / num_of_px
25
26          sum_squared_err0 = sum_squared_err1 = sum_squared_err2 = 0.
27          for imgs, labels in loader:
28              sum_squared_err0 += (imgs[:,0,:,:]-mean_r).pow(2).sum()
29              sum_squared_err1 += (imgs[:,1,:,:]-mean_g).pow(2).sum()
30              sum_squared_err2 += (imgs[:,2,:,:]-mean_b).pow(2).sum()
31          std_r = torch.sqrt(sum_squared_err0 / num_of_px)
32          std_g = torch.sqrt(sum_squared_err1 / num_of_px)
33          std_b = torch.sqrt(sum_squared_err2 / num_of_px)
34
35          return [mean_r, mean_g, mean_b],[std_r, std_g, std_b]
36
37      #calculating values for the first channel 'H' looks different due to 'H' beeing circular.
38      else:
39          num_of_px = len(dataset) * 244 * 244
40          sum_h = 0.j
41          sum_s = sum_v = 0.
42
43          for imgs, labels in loader:
44              sum_h += (np.exp(2 * 1j * np.pi * imgs[:,0,:,:].numpy())/num_of_px).sum()
45              sum_s += imgs[:,1,:,:].sum()
46              sum_v += imgs[:,2,:,:].sum()
47          H_mean_angle = np.angle(sum_h)
48          mean_s = (sum_s / num_of_px)
49          mean_v = (sum_v / num_of_px)
50
```

```
        sum_squared_h = sum_squared_s = sum_squared_v = 0.
        for imgs, labels in loader:
                H_centered = np.exp((2 * 1.j * np.pi * imgs[:,0,:,:].numpy() - 1.j * H_mean_angle))
                H_centered = np.angle(H_centered)
                sum_squared_h += np.power(H_centered,2).sum()
                sum_squared_s += (imgs[:,1,:,:]-mean_s).pow(2).sum()
                sum_squared_v += (imgs[:,2,:,:]-mean_v).pow(2).sum()
        std_h = torch.tensor(np.sqrt(sum_squared_h / num_of_px))
        std_s = np.sqrt(sum_squared_s / num_of_px)
        std_v = np.sqrt(sum_squared_v / num_of_px)

        return [[torch.tensor(H_mean_angle), mean_s, mean_v], [std_h, std_s, std_v]]
```

Listing A2.6: Normalized dataset creation

```
def norm_dataset_creation(tool, cs, rgb_like, batch_size=300, num_workers=4):
    '''
    This function creates three image datasets according to the given tool. All three get normalized on
    the normalization values calculated on the train dataframe. Plots Histogram of flattend first channel
    and mean of rows and columns of first channel (of one batch).

    :param tool: Tool the datasets shall be balanced on.
    :param cs: Wanted color space.
    :param rgb_like: False if normalization on first channel shall be circular which is needed for HSV
                     normalization.

    :return: Dataset class for each of three given dataframes.
    '''
    train_path = f'/workspace/app/resources/{tool}/train_df.csv'
    val_path = f'/workspace/app/resources/{tool}/val_df.csv'
    test_path = f'/workspace/app/resources/{tool}/test_df.csv'

    unnorm_ds = SimpleDataset(df=pd.read_csv(train_path), cs=cs)
    norm_vals = calc_normalisation_values(dataset=unnorm_ds, rgb_like=rgb_like)

    print('train')
    train_ds = ImgDataset(df=pd.read_csv(train_path), cs=cs, norm_vals=norm_vals, normalize=True,
        rgb_like=rgb_like)
    print('val')
    val_ds = ImgDataset(df=pd.read_csv(val_path), cs=cs, norm_vals=norm_vals, normalize=True,
        rgb_like=rgb_like)
    print('test')
    test_ds = ImgDataset(df=pd.read_csv(test_path), cs=cs, norm_vals=norm_vals, normalize=True,
        rgb_like=rgb_like)

    trainloader=DataLoader(train_ds, batch_size=300, num_workers=4, shuffle=True)
    for imgs, labels in trainloader:
```

```
            break
        plt.figure(figsize=[12,6])
        plt.subplot(1,2,1)
        plt.hist(imgs[:,0,:,:].flatten(), bins=20)
        plt.title(f'train␣|␣cs:{cs}␣|␣rgb_like:{rgb_like}')
        plt.subplot(1,2,2)
        plt.hist(imgs[:,0,:,:].mean(axis=-1).mean(axis=-1), bins=20)
        plt.title(f'train␣|␣mean␣row␣&␣col␣|␣cs:{cs}␣|␣rgb_like:{rgb_like}')

        return train_ds, val_ds, test_ds
```

Listing A2.7: The model

```
class lightningTinyCBR(pl.LightningModule):

    def __init__(self, train_set, val_set, test_set, learning_rate, batch_size=64, num_workers=4):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, 5)
        self.conv1_bn = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 5)
        self.conv2_bn = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, 5)
        self.conv3_bn = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, 5)
        self.conv4_bn = nn.BatchNorm2d(512)

        self.pool = nn.MaxPool2d(3, stride=2)
        self.fc1 = nn.Linear(51200, 2)
        self.sm = nn.Softmax(dim = 1)

        self.loss = nn.CrossEntropyLoss()

        self.accuracy = pl.metrics.Accuracy()

        self.batch_size = batch_size
        self.num_workers = num_workers
        self.lr = learning_rate
        self.train_set = train_set
        self.val_set = val_set
        self.test_set = test_set

    def forward(self, X):
        X = self.conv1_bn(self.pool(F.relu(self.conv1(X))))
        X = self.conv2_bn(self.pool(F.relu(self.conv2(X))))
        X = self.conv3_bn(self.pool(F.relu(self.conv3(X))))
        X = self.conv4_bn(self.pool(F.relu(self.conv4(X))))
        X = torch.flatten(X, start_dim = 1)
```

```
35          X = self.fc1(X)
36          return X
37
38      def configure_optimizers(self):
39          return optim.Adam(self.parameters(), lr=(self.lr))
40
41      def training_step(self, batch, batch_idx):
42          inputs, labels = batch
43          outputs = self(inputs)
44          y_score = self.sm(outputs)[:,1]
45          _, preds = torch.max(outputs, dim=1)
46          train_loss = self.loss(outputs, labels)
47          train_step_log = {'train_step_acc': self.accuracy(preds, labels), 'train_step_loss': train_loss}
48          return {'log': train_step_log,
49                  'loss': train_loss,
50                  'preds': preds,
51                  'labels': labels,
52                  'y_score': y_score}
53
54      def training_epoch_end(self, training_step_outputs):
55          train_epoch_log = {'avg_train_loss': torch.stack([out['loss'] for out in training_step_outputs]).
                mean()}
56          return {'log': train_epoch_log}
57
58      def validation_step(self, batch, batch_idx):
59          val_step_dict = self.training_step(batch, batch_idx)
60          return {'val_loss': val_step_dict['loss'],
61                  'val_preds': val_step_dict['preds'],
62                  'val_labels': val_step_dict['labels'],
63                  'y_score': val_step_dict['y_score']}
64
65      def validation_epoch_end(self, val_step_outputs):
66          preds = torch.stack([x['val_preds'] for x in val_step_outputs if x['val_preds'].shape[0] == self.
                batch_size]).flatten()
67          labels = torch.stack([x['val_labels'] for x in val_step_outputs if x['val_preds'].shape[0] == self.
                batch_size]).flatten()
68          y_score = torch.stack([x['y_score'] for x in val_step_outputs if x['y_score'].shape[0] == self.
                batch_size]).flatten()
69          avg_val_loss = torch.stack([x['val_loss'] for x in val_step_outputs]).mean()
70          roc_auc = pl.metrics.functional.auroc(pred=y_score, target=labels)
71          avg_val_acc = self.accuracy(preds, labels)
72          val_epoch_log = {'avg_val_loss': avg_val_loss, 'avg_val_acc': avg_val_acc, 'val_roc_auc' :
                roc_auc}
73
74          return {'log': val_epoch_log}
75
76      def test_step(self, batch, batch_idx):
```

```
77          test_step_dict = self.training_step(batch, batch_idx)
78          return {'test_loss': test_step_dict['loss'],
79                  'test_preds': test_step_dict['preds'],
80                  'test_labels': test_step_dict['labels'],
81                  'y_score': test_step_dict['y_score']}
82
83      def test_epoch_end(self, test_step_outputs):
84          preds = torch.cat([x['test_preds'] for x in test_step_outputs]).flatten()
85          labels = torch.cat([x['test_labels'] for x in test_step_outputs]).flatten()
86          y_score = torch.cat([x['y_score'] for x in test_step_outputs]).flatten()
87          avg_test_loss = torch.stack([x['test_loss'] for x in test_step_outputs]).mean()
88          roc_auc = pl.metrics.functional.auroc(pred=y_score, target=labels)
89          avg_test_acc = self.accuracy(preds, labels)
90          test_epoch_log = {'avg_test_loss': avg_test_loss, 'avg_test_acc': avg_test_acc, 'test_roc_auc'
                : roc_auc}
91          self.y_score_test = y_score.tolist()
92          self.ground_truth = labels.tolist()
93          return {'log' : test_epoch_log}
94
95
96      def train_dataloader(self):
97          return DataLoader(self.train_set, batch_size=self.batch_size, num_workers=self.num_workers,
                shuffle=True)
98
99      def val_dataloader(self):
100         return DataLoader(self.val_set, batch_size=self.batch_size, num_workers=self.num_workers)
101
102     def test_dataloader(self):
103         return DataLoader(self.test_set, batch_size=self.batch_size, num_workers=self.num_workers)
```

A2.2 Training

Listing A2.8: Imports

```
1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as F
4  import torch.optim as optim
5  import torchvision
6  import matplotlib.pyplot as plt
7  import pandas as pd
8  import re
9  import cv2
10 import albumentations
11 import albumentations.pytorch
12 import time
```

```
import numpy as np
import pytorch_lightning as pl
import neptune
import time
from tqdm import tqdm
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.metrics.functional import auroc
from pytorch_lightning.metrics.functional import accuracy
from pathlib import Path
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
from pytorch_lightning import Trainer
from pytorch_lightning.loggers import NeptuneLogger
from pytorch_lightning.callbacks import LearningRateMonitor

import ba_modules as bam
from cxdata.cholec80.utils import load_cholec80_as_dataframe

%config Completer.use_jedi = False
```

Listing A2.9: Imports

```
tools = ['grasper', 'scissors', 'hook', 'clipper']
for tool in tools:
    # load Dataset (25fps)
    data = load_cholec80_as_dataframe(path)

    # reduce to 1 fps, drop nan
    data1fps = data[data['frame'] % 25 == 0]
    data1fps = data1fps.dropna(subset=['grasper'])
    data1fps.reset_index(drop = True, inplace = True)

    # shuffled list of all videos
    all_videos = data1fps.video.unique()
    np.random.shuffle(all_videos)

    train_df = bam.create_balanced_dataframes(data1fps,
                                              id_list=all_videos,
                                              tool=tool,
                                              span=[0, 51])
    val_df = bam.create_balanced_dataframes(data1fps,
                                            id_list=all_videos,
                                            tool=tool,
                                            span=[51, 64])
    test_df = bam.create_balanced_dataframes(data1fps,
```

```
                                                id_list=all_videos,
                                                tool=tool,
                                                span=[64, 80])

    train_df.to_csv(f'/workspace/app/resources/{tool}/train_df.csv', index=False)
    val_df.to_csv(f'/workspace/app/resources/{tool}/val_df.csv', index=False)
    test_df.to_csv(f'/workspace/app/resources/{tool}/test_df.csv', index=False)
```

Listing A2.10: Dataset creation

```
#scissors
sciss_train_HSV, sciss_val_HSV, sciss_test_HSV = bam.norm_dataset_creation('scissors', cs='HSV',
    rgb_like=False)
sciss_train_HSVrgb, sciss_val_HSVrgb, sciss_test_HSVrgb = bam.norm_dataset_creation('scissors', cs='
    HSV', rgb_like=True)
sciss_train_RGB, sciss_val_RGB, sciss_test_RGB = bam.norm_dataset_creation('scissors', cs='RGB',
    rgb_like=True)

#clipper
clip_train_HSV, clip_val_HSV, clip_test_HSV = bam.norm_dataset_creation('clipper', cs='HSV',
    rgb_like=False)
clip_train_HSVrgb, clip_val_HSVrgb, clip_test_HSVrgb = bam.norm_dataset_creation('clipper', cs='
    HSV', rgb_like=True)
clip_train_RGB, clip_val_RGB, clip_test_RGB = bam.norm_dataset_creation('clipper', cs='RGB',
    rgb_like=True)

#grasper
grasp_train_HSV, grasp_val_HSV, grasp_test_HSV = bam.norm_dataset_creation('grasper', cs='HSV',
    rgb_like=False)
grasp_train_HSVrgb, grasp_val_HSVrgb, grasp_test_HSVrgb = bam.norm_dataset_creation('grasper', cs
    ='HSV', rgb_like=True)
grasp_train_RGB, grasp_val_RGB, grasp_test_RGB = bam.norm_dataset_creation('grasper', cs='RGB',
    rgb_like=True)

#hook
hook_train_HSV, hook_val_HSV, hook_test_HSV = bam.norm_dataset_creation('hook', cs='HSV',
    rgb_like=False)
hook_train_HSVrgb, hook_val_HSVrgb, hook_test_HSVrgb = bam.norm_dataset_creation('hook', cs='
    HSV', rgb_like=True)
hook_train_RGB, hook_val_RGB, hook_test_RGB = bam.norm_dataset_creation('hook', cs='RGB',
    rgb_like=True)
```

Listing A2.11: The training

```
datasetdic = {'clipperHSV':[clip_train_HSV,clip_val_HSV,clip_test_HSV],
              'clipperHSVrgb':[clip_train_HSVrgb,clip_val_HSVrgb,clip_test_HSVrgb],
              'clipperRGB':[clip_train_RGB,clip_val_RGB,clip_test_RGB],
              'scissorsHSV':[sciss_train_HSV,sciss_val_HSV,sciss_test_HSV],
```

```
                'scissorsHSVrgb':[sciss_train_HSVrgb,sciss_val_HSVrgb,sciss_test_HSVrgb],
                'scissorsRGB':[sciss_train_RGB,sciss_val_RGB,sciss_test_RGB],
                'grasperHSV':[grasp_train_HSV,grasp_val_HSV,grasp_test_HSV],
                'grasperHSVrgb':[grasp_train_HSVrgb,grasp_val_HSVrgb,grasp_test_HSVrgb],
                'grasperRGB':[grasp_train_RGB,grasp_val_RGB,grasp_test_RGB],
                'hookHSV':[hook_train_HSV,hook_val_HSV,hook_test_HSV],
                'hookHSVrgb':[hook_train_HSVrgb,hook_val_HSVrgb,hook_test_HSVrgb],
                'hookRGB':[hook_train_RGB,hook_val_RGB,hook_test_RGB]}

tools = ['scissors', 'clipper', 'grasper', 'hook']
lrs = [8e-05, 1e-4, 1e-3]
css = ['HSV', 'RGB', 'HSVrgb']

for tool in tools:
    for lr in lrs:
        for cs in css:
            neptune_logger = NeptuneLogger(api_key='<own␣neptune␣key',
                                           project_name='<own␣project␣name>',
                                           experiment_name='<own␣experiment␣name>', # Optional,
                                           tags=[<list of own tags>] # Optional,
                                          )

            checkpoint_callback = ModelCheckpoint(monitor='avg_val_loss',
                                                  verbose=True,
                                                  dirpath='/workspace/app/notebooks/checks/z',
                                                  filename=f'{neptune_logger.experiment.id}-{cs}-{
                                                      tool}''-{val_roc_auc:.2f}-{avg_val_loss:.2f}')

            lr_monitor = LearningRateMonitor(logging_interval='epoch')

            train, val, test = datasetdic[f'{tool}{cs}']

            model = bam.lightningTinyCBR(train_set=train,val_set=val,test_set=test, batch_size=64,
                learning_rate=lr)

            trainer = pl.Trainer(callbacks=[EarlyStopping(monitor='avg_val_loss', patience = 30),
                checkpoint_callback, lr_monitor],
                                 gpus=1,
                                 logger=neptune_logger,
                                 auto_lr_find=False)

            trainer.fit(model)

            del(model)
            del(trainer)
```

A2.3 Testing

Listing A2.12: Imports

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import matplotlib.pyplot as plt
import pandas as pd
import re
import cv2
import albumentations
import albumentations.pytorch
import time
import numpy as np
import pytorch_lightning as pl
import neptune
import time
from tqdm import tqdm
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.metrics.functional import auroc
from pytorch_lightning.metrics.functional import accuracy
from pathlib import Path
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
from pytorch_lightning import Trainer
from pytorch_lightning.loggers import NeptuneLogger
from pytorch_lightning.callbacks import LearningRateMonitor

import ba_modules as bam
from cxdata.cholec80.utils import load_cholec80_as_dataframe

%config Completer.use_jedi = False
```

Listing A2.13: Hyperparameters

```
# 'scissors', 'clipper', 'grasper', 'hook'
tool = 'grasper'

# 'HSV','RGB', for 'HSVrgb' choose colorspace = 'HSV' and rgb_like = <True> to normalize it non
    circular
colorspace = 'HSV'

```

```
# False for circular normalization, True for non circular normalization
rgb_like = True

checkpoint = <path to desired checkpoint>
```

Listing A2.14: Preparation

```
train_set,val_set,test_set = bam.norm_dataset_creation(tool=tool, cs=colorspace, rgb_like=rgb_like)

neptune_logger = NeptuneLogger(
    api_key='<own key>',
    project_name='<own project name>',
    experiment_name=f'<own experiment name>', # Optional,
    tags=['<list of tags>'] # Optional,
)

model = bam.lightningTinyCBR(train_set,val_set,test_set)

trainer = pl.Trainer(gpus=1, logger=neptune_logger)

model = model.load_from_checkpoint(f'{checkpoint}', train_set=train_set, val_set=val_set, test_set=
    test_set, learning_rate=1e-4)
```

Listing A2.15: Testing

```
trainer.test(model)
```