POLITECNICO DI TORINO

Master's Degree
in Data Science and Engineering

Mesothelioma Classification Project

Name	**ID Number**
Davide Bussone	s276486
Pedro Antonio Hernandez Zamarron	s271129

Academic Year 2020-2021

CONTENTS

1 Building The Tiles 2
- 1.1 Manual Selection Of ROIs . . . 2
- 1.2 Dividing Regions Into Uniform Tiles . . . 3
- 1.3 Reading Regions . . . 4

2 Data Structure 6
- 2.1 Directories And Drive . . . 6
- 2.2 Dataset Construction . . . 6
- 2.3 Pickles . . . 7

3 Tile Network 9
- 3.1 Checking Pickles . . . 9
- 3.2 Data Labels . . . 10
  - 3.2.1 Label distribution . . . 10
- 3.3 The Tile Network . . . 11

4 Patient Predictions 13
- 4.1 The large_image Library . . . 13
- 4.2 Reducing the Number of Tiles . . . 13

5 Shallow Learning 16
- 5.1 Conclusion and Final Thoughts . . . 20

CHAPTER

1

BUILDING THE TILES

1.1 Manual Selection Of ROIs

The first step of this case study is to manually select the regions with patterns able to discriminate between an epitheliod and non-epithelioid mesothelioma. The program known as ASAP [1] (Automated Slide Analysis Platform) enables us to perform this task. It consists of two main components :

- IO libraries for reading and writing multi-resolution images. The processing libraries are wrapped in Python to more easily interact machine learning tools and frameworks.
- A viewer component for visualizing such images.

ASAP allows selecting regions of different shapes (dots, rectangles, polygons and spline) in an image. These regions can then be annotated. Such annotations are then stored into a XML file.To obtain a more uniform and tidier dataset, only rectangular annotations were used . Ten annotations are created for each chosen medical plate. While the total number of considered plates is equal to thirty.
We grouped plates in three distinct groups of ten from each of them we selected regions pertinent to the following classes:

- epithelioid patterns
- non-epithelioid patterns
- healthy tissue and parts of the plate where no tissue was present

As a further note, bifasic mesothelioma will be considered as a non-epithelioid, due to the lack of a sufficient number of sarcomatoid plates.

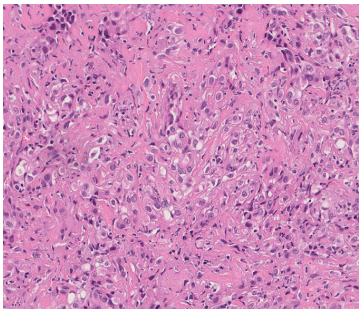


Figure 1.1. Epithelioid

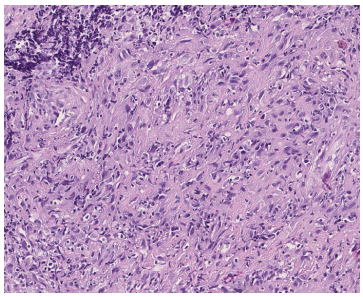


Figure 1.2. Non epithelioid

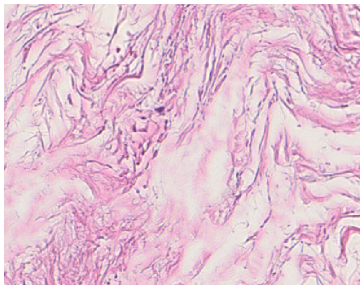


Figure 1.3. Healthy Tissue

1.2 Dividing Regions Into Uniform Tiles

As said in section 1.1, annotations are stored in an XML file, an extensible markup language file used to structure data for storage. It is composed of a tag and a text part. The tags unambiguously identify each region, and the text part within the tags describe the region in terms of Cartesian coordinates.

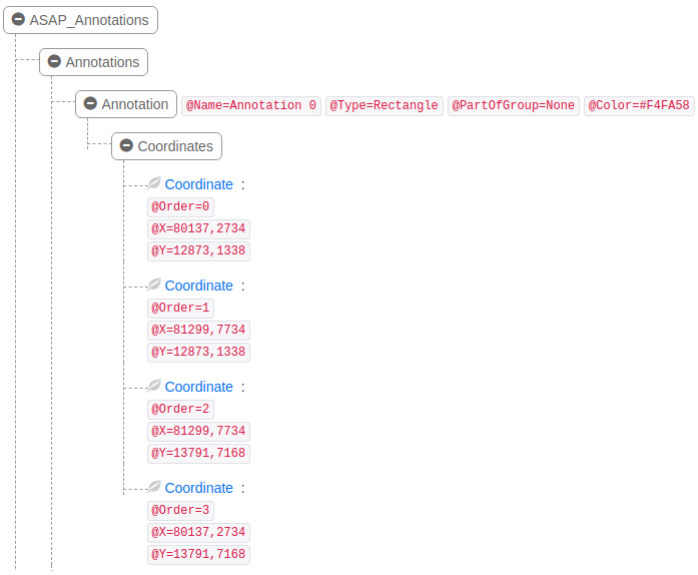


Figure 1.4. Structure of our XML annotations

In Python, the xml.etree.ElementTree (ET in short) module implements a simple and efficient API for parsing XML data. Since XML has a hierarchical format, the most natural way to represent it is with a tree. To achieve this goal ET has two classes:

- ElementTree represents the whole XML document as a tree
- Element represents a single node in this tree

Then, ET has a method that parses XML files and, through getroot(), to obtain the ASAP_Annotations in figure 1.4. Then, all the rectangles' positions will be used to compute height and width of each annotation.

The function getCoordinates(), inside *dataPreprocess.py*, parses the XML file and stores the coordinates of each tile in an internal data structure.

As an additional note, the aim is to find the largest rectangle within the original one, rounding down the coordinates when necessary, in order to have integer coordinates that can be read by the OpenSlide library.

1.3 Reading Regions

All the tile coordinates are now converted into images, in order to be learnt by a neural network. The file *dataPreprocess.py* offers the function getImageTiles().

```
getImageTiles(filenameImage,filenameXML,size=128)
```

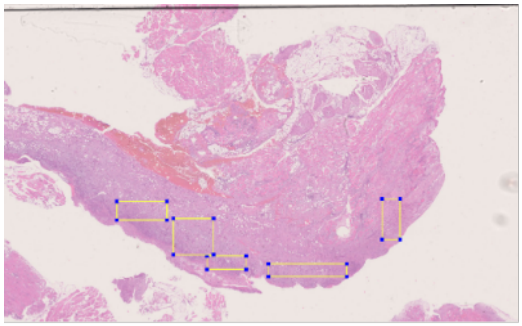
- filenameImage : is the name of the original plate.
- filenameXML : is the name of the xml file pertaining to the image
- size : refers to the desired size of the resulting tiles

The OpenSlide library [3], specialized in reading whole-side images, offers a method, called "read-region", suitable for this task.

```
read_region(location, level, size)
```

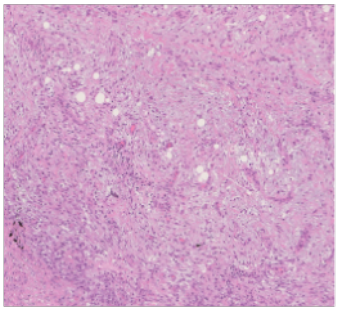
- location : (x, y) tuple giving the top left pixel in the image
- level : refers to the desired magnification of the image. Since we want to preserve the original resolution we call this with zero
- size : the desired size of the tile

get_coordinates() iterates over the region that it receives with a step equal to the size parameter. These coordinates are then passed to the read_region() function which generate tiles that will be fed into the neural network.

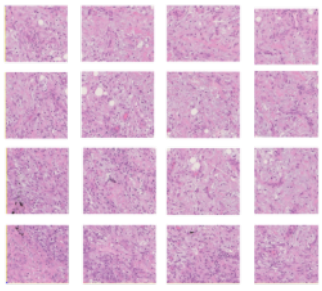


original image

get_coordinates()



single region



get_tiles()

CHAPTER

2

DATA STRUCTURE

2.1 Directories And Drive

Figure 2.1 portrays the organization of the project. First , a folder called "bioInfoImages" is created inside Google Drive. This environment will host the entire data set. All code is run on Google Colab, which allows to use a random GPU to speed the execution time. In addition, it enables the usage of a shared folder in the drive.

2.2 Dataset Construction

Subsequently the the data set is split into two parts. Each member of the group was assigned the task of labeling 15 images. 5 images of each class for each member of the team.
As a matter of fact there are 3 separate classes that we identified when selecting regions, epithelioid tissue, non epithelioid tissue and healthy tissue (and no tissue).
This last class was given the label "useless". We decided to include "third class" because the alternative would've been to merely calculate an average value for the pixels in a tile and filter tiles that are too white. In other words we are able to filter tiles that aren't white but are useless as far as classification is concerned.

Inside each user's directory there is a sub-directory pertaining to each class. To populate these directories with the tiles that will be the true input of the network, we make use of both the original images (in .tiff or .ndpi format) and their respective annotations. The "TIFdir" hosts the unaltered images. On the other hand the "XMLdir" contains the

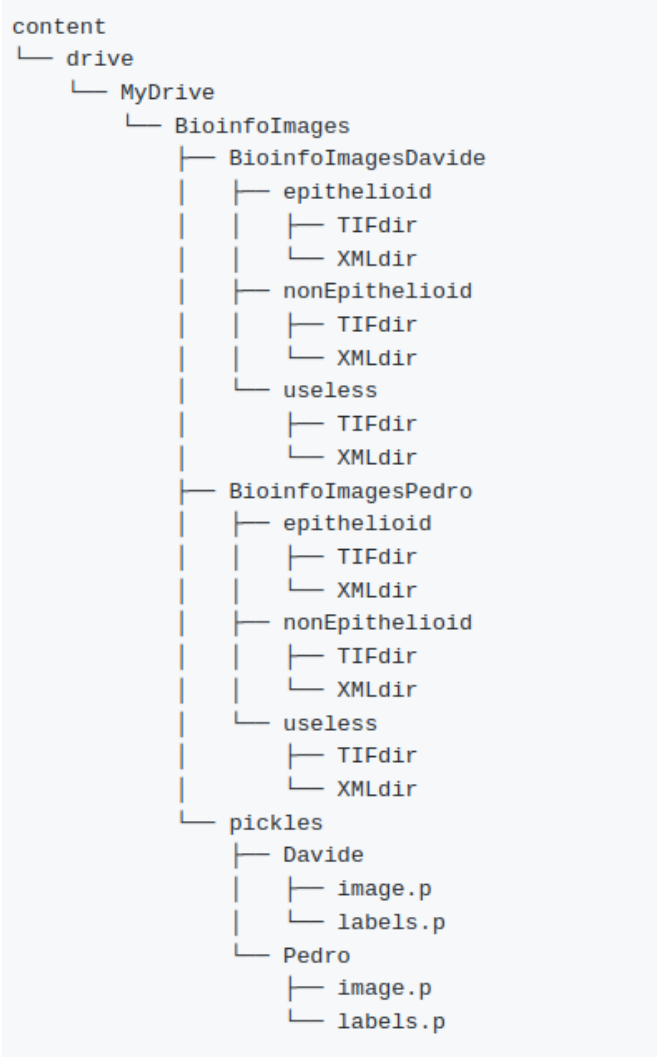


Figure 2.1. Structure of our dataset

annotations that we created through the selection of regions in the ASAP software.

The dataset can now be populated with the tiles, created thanks to the function "getImageTiles", defined in section 1.3. This function creates tiles of size 224x224 by taking as inputs the xml annotations and the original clinical plates. All the tiles' tensors and their respective labels are stored in two separate lists.

2.3 Pickles

Since the aforementioned process is computationally expensive, the tensors and the labels generated in the previous step are stored inside pickle files.

These pickle files are part of a Python module which serializes objects so they can be effortlessly saved and loaded into another program. In our case, we use it to access the tiles without having to call the function getImageTiles() each time we want to train the neural network.

CHAPTER

3

TILE NETWORK

3.1 Checking Pickles

Now that all the tiles have been saved inside pickle files, we can load them simply by calling the pickle.load() function. Furthermore the tensors inside the Davide and the Pedro pickle directories 2.1 need to be merged since they were generated separately in order to parallelize the workload and save time.

As a sanity check, the tensors are loaded into the Tensorboard library [4] and the first 25 tile images are printed onto the screen. Tensorboard is a library that supplies many useful tools for evaluating the performance of neural networks and ml algorithms.

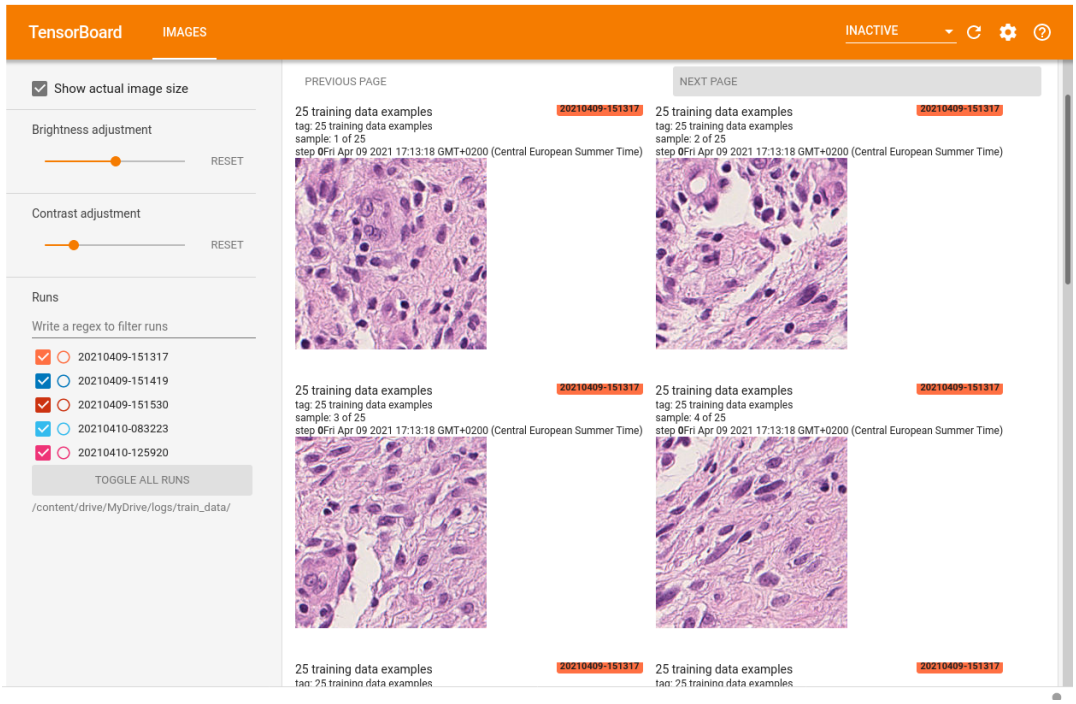


Figure 3.1. Tensorboard sanity check

Figure 3.1 shows images that our data preprocessing so far has been successful.

3.2 Data Labels

Hitherto labels are literal categories, such as "epithelioid" or "useless". As a consequence, they need to be converted into numerical data, otherwise the algorithm cannot work. The technique we have chosen is one-hot encoding. This creates a binary column for each class and returns a sparse matrix. By default, the encoder takes the categories based on the unique values in each feature. For instance, "Epithelioid" corresponds to (1,0,0), while "useless " to (0,0,1), since 1 represents this unique value.
We opted for one hot encoding instead of label encoding since the algorithm might erroneously infer some sort of sequentiality between the labels.

3.2.1 Label distribution

Figure 3.2 is a graphical representation of the class distribution within the tiles. The dataset isn't perfectly balanced, however the difference between the minority class and majority class isn't a cause for major concern. This degree of difference is acceptable in our estimation.

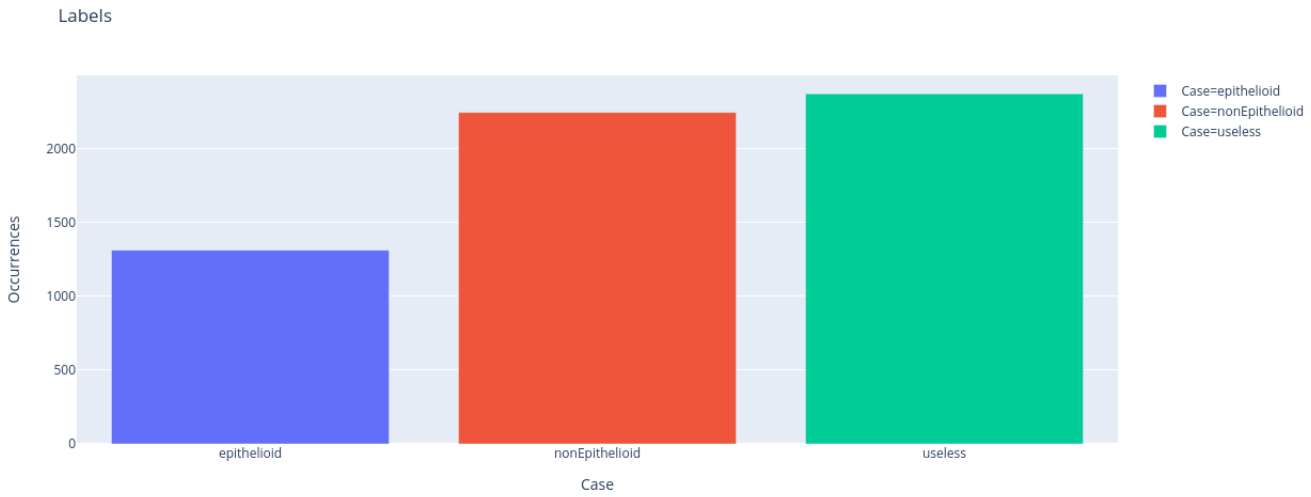


Figure 3.2. Distribution of data amongst classes

3.3 The Tile Network

So far, the labels and the tiles have been stored in separate data structures. We make use of the TensorFlow's Dataset [5] structure to merge these two elements.

Our data is fed into the from_tensor_slices() function and we must add an extra dimension on top of our data since this method removes the first dimension of the data while still keeping the other dimensions intact.

Then, the data-set's elements are shuffled with a buffer size of 5000. Each element is taken one by one and it can be swapped with any other element that is within a distance from its index lesser or equal to the buffer size. After, the dataset is split into a training and validation set. The last one represents a third of the entire dataset.

Graphs 3.3 and 3.4 represent the classes' occurrences. In addition, both the subsets have a batch size equal to 32.

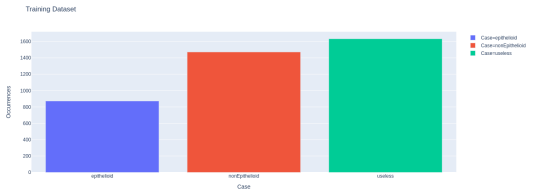


Figure 3.3. Training Classes Distribution

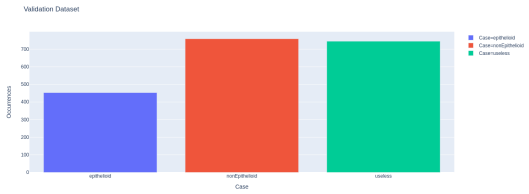


Figure 3.4. Validation Classes Distribution

Once the dataset is ready, it is possible to instantiate the model. The chosen architecture is a VGG-16, pretrained on Imagenet. The input to the first convolutional layer is of fixed size 224x224 RGB image, which is passed through a stock of convolutional layers, whose filters are of dimension 3x3. The spatial pooling is performed by 5 max-pooling layers, but not all convolutional layers are followed by a pooling. Three dense layers follow the stack of convolutional ones. The first one has 4096 channels, the second 1072. Both have "relu" as activation function. Finally, the output layer contains one channel for each class. The dropout is set to 0.2, while the selected optimizer is Adam, composed by a learning rate of 10^{-3}. Finally, the fit method can be invoked on the model. It requires as parameters the training and validation subsets and the number of epochs, set to 30. From plots 3.5 and 3.6 is possible to infer the evolution of the cross entropy loss and the accuracy metrics. On the validation set, the model manages to reach 97 % of accuracy. This represents an important achievement, since this model will be exploited to classify all the tiles of each patient, as proposed in section 4.

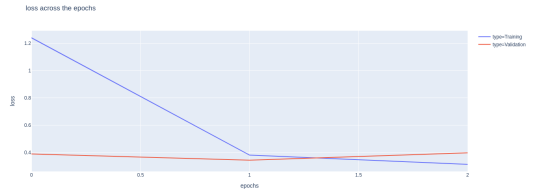


Figure 3.5. Cross-Entropy Loss Behaviour

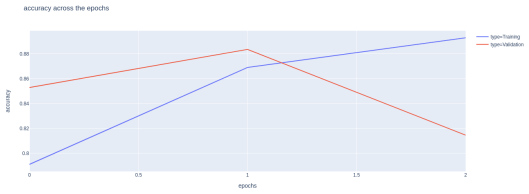


Figure 3.6. Accuracy Score Behaviour

CHAPTER

4

PATIENT PREDICTIONS

So far we've dealt with making predictions for single regions that we've have hand selected ourselves. In this next step have to predict each single tile for each one of the patients.

This presents some issues:

- Transforming each patient image into a set of tiles that can be fed into our network.
- The number of tiles in each plate is extremely high. Classifying each one of them is beyond the capabilities of the resources that we possess. Colab has a limit to the runtime that you can use each day, which is not enough for the large number of tiles by image.

4.1 The large_image Library

This specific library solves the first of our issues. As a matter of fact we exploit the function *tileIterator()*. tileIterator accepts the tile dimensions as a parameter, it builds a grid over the image where each square corresponds to a tile and subsequently it allows us to iterate over the constructed tiles.

4.2 Reducing the Number of Tiles

Reducing the number of tiles in order to overcome hardware constraints is a compromise that was necessary. As a matter of fact, with more resources the correct thing to would have been to make predictions for every tile in each patient, in order to obtain a more accurate diagnosis for the patient.

To reduce the impact of this necessary compromise used, a function from the Histomicstk library [2] *get_tissue_mask()*.

Get_tissue_mask takes the original plate and takes creates a mask over those parts of the image where tissue is present.

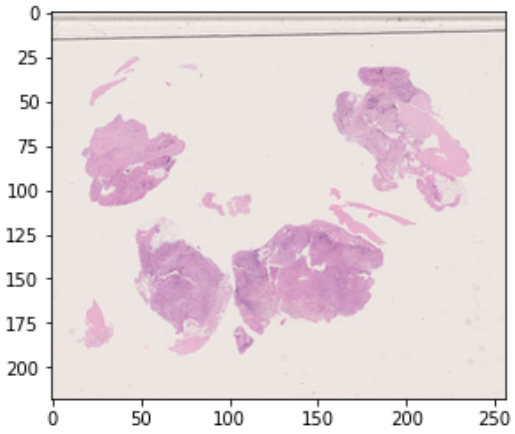


Figure 4.1. Original plate

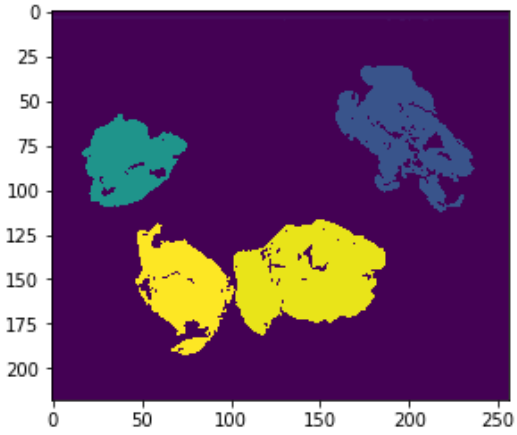


Figure 4.2. With a tissue detection mask

Depending on the particular parameters, such as n_tresholding_steps, you pass to this function the tissue detection function can be more or less sensitive to smaller areas of tissue. We used the default parameters since in our estimation they were good enough.

The output of such a function is two highly scaled-down versions of the image:

- In the first one each pixel is labeled with different integers. The number 0 corresponds to an area where no tissue was detected whereas other numbers represent a certain area of tissue. In 4.2 each color corresponds to a different number.
- The second one is similar, but only the largest tissue region is selected, like in 4.3.

The problem with selecting all of the tissue regions that were detected is that it is still too computationally intensive to make predictions for more than one hundred patients. We are aware of the shortcomings of our decision, however in the end we took the largest region of tissue for each patient and made our predictions upon such data.

Finally, two important steps remained:

1. Reshaping the tissue mask into a square shape that can be fed into the tileIterator() function
2. Resizing the mask into the original image of the plate.

The first problem was solved by simply taking the uppermost y coordinate and the leftmost x coordinate for the mask in order to find the top left corner of the tissue mask. This process was then repeated 3 more times in order to obtain the remaining corners.

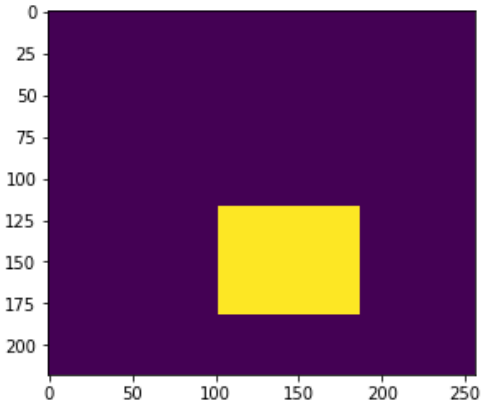


Figure 4.3. Largest mask in square shape

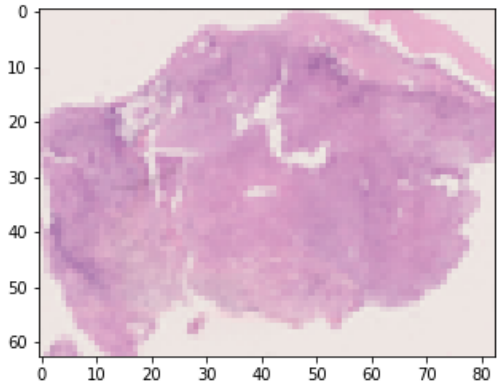


Figure 4.4. Mask applied on image

In order to compute the mask over the original image we took the coordinates obtained in the previous step and multiplied them by a coefficient "mult" obtained by dividing the original sides and dividing by the sides of the down-scaled image. The result was then rounded up in order to ensure that we were encompassing all of the tissue region.

CHAPTER

5

SHALLOW LEARNING

The result of the previous chapter is a datafame, displayed in figure 5.1, where each row corresponds to a patient, and the features are epithelioid, useless and non-epithelioid.

	PAZIENTE	epithelioid	non	useless	label
0	/content/drive/MyDrive/meso_san_luigi/M-1.ndpi	5391.789551	5746.389160	540.820557	0
1	/content/drive/MyDrive/meso_san_luigi/M-10.ndpi	2791.386475	2632.429199	166.183868	1
2	/content/drive/MyDrive/meso_san_luigi/M-44.ndpi	2039.272949	1157.257568	460.469604	1
3	/content/drive/MyDrive/meso_san_luigi/M-45.ndpi	13094.708984	5592.917480	1035.375977	1
4	/content/drive/MyDrive/meso_san_luigi/M-46.ndpi	7141.227539	5723.562988	510.207336	1
...	...	...	...	...	...
109	/content/drive/MyDrive/meso_san_luigi/M-89.ndpi	3419.622070	2874.378418	360.999146	1
110	/content/drive/MyDrive/meso_san_luigi/M-85.ndpi	36386.183594	21200.000000	5644.827148	1
111	/content/drive/MyDrive/meso_san_luigi/M-68.ndpi	31392.984375	25528.000000	2726.013184	1
112	/content/drive/MyDrive/meso_san_luigi/M-70.ndpi	57607.964844	18582.972656	3936.078369	1
113	/content/drive/MyDrive/meso_san_luigi/M-87.ndpi	45684.460938	68361.460938	5249.099609	0

Figure 5.1. Results of the previous step

Before using any shallow learning approaches for predicting a diagnosis the data-set was normalized using the StandardScaler class from the sklearn library. The way in which the datapoints occupy the scatter plot 5.2 is unchanged before this procedure. StandardScaler is useful for improving the performance of some of these classification algorithms.

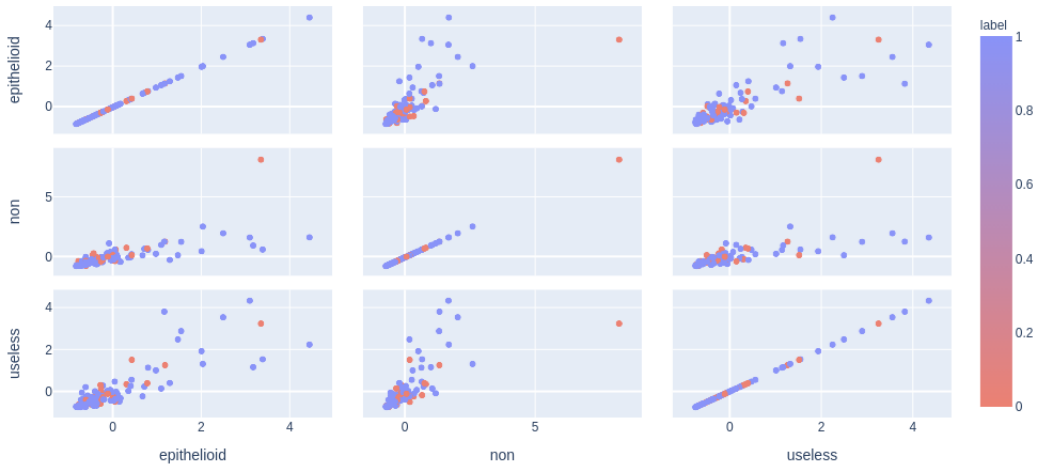


Figure 5.2. Normalized data

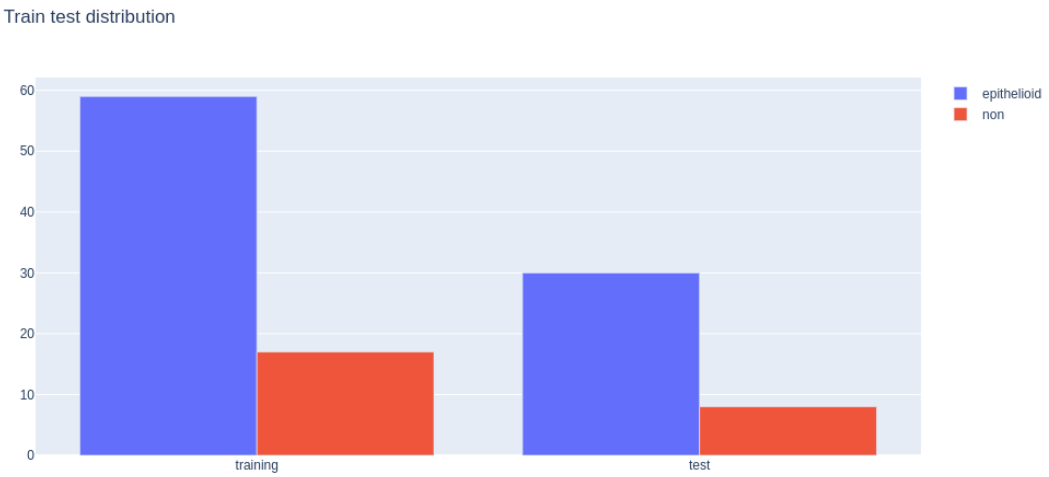


Figure 5.3. Train-Validation split distribution

The data set is split into a train and validation split. As figure 5.3 shows, there is a class imbalance problem in our data-set. Since most of the patients have an epithelioid mesothelioma diagnosis, there are very few non-epithelioid labels in our data.

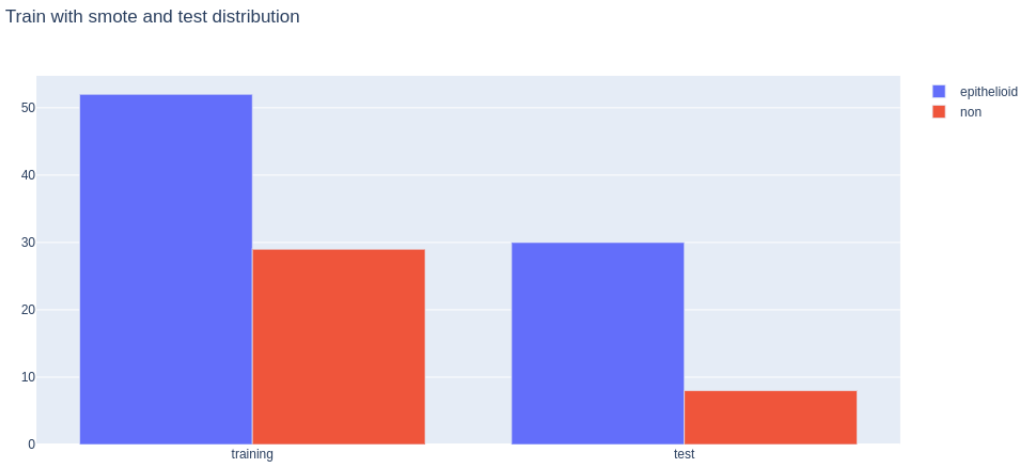


Figure 5.4. Train-Validation split distribution with Smote

To mitigate the negative impact that an imbalanced dataset would have in the classification of the minority class we use the SMOTE technique. This is a technique that over samples the minority labels by creating synthetic data through interpolation.
Furthermore we add a slight under-sampling of the majority class, since according to the original SMOTE paper [6], this significantly increases the performance of the algorithm. The new distribution is portrayed by graph 5.4.

As a baseline we decided to use the majority voting technique, from figure 5.5 it is possible to see the decision boundaries generated by this rule.

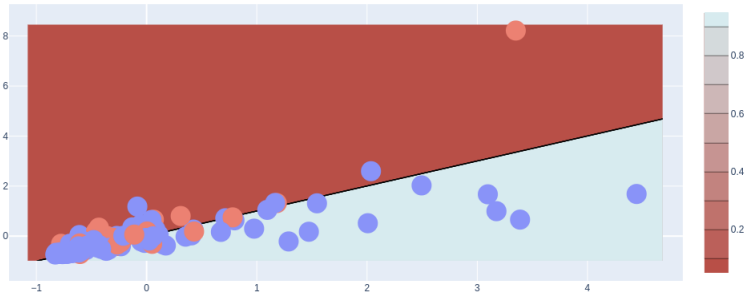


Figure 5.5. Majority Voting Decision Boundaries

After, we exploited other machine learning algorithms, like Support Vector Machines, K-nearest neighbors, Random Forest and Logistic regression. For all of them, the best hyper-parameters are selected by applying the grid search algorithm, with cross-validation and five splits. On the test set, we evaluated both the accuracy and the f1 score, which takes in account the precision and the recall. The confusion matrix 5.6 shows that the logistic regression algorithm does not predict any non-epithelioid, while K-nearest neighbors algorithms correctly classifies a part of them, as portrayed by figure 5.7. This problem might be caused by the small size of the data-set, since the non-epithelioid labels are rare.

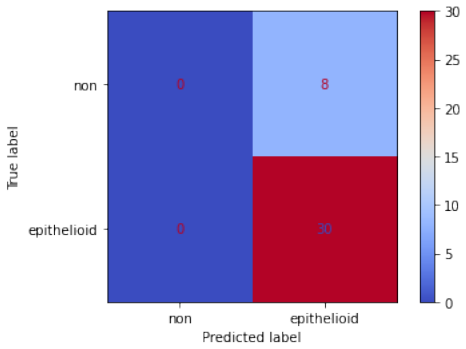


Figure 5.6. Logistic Regression

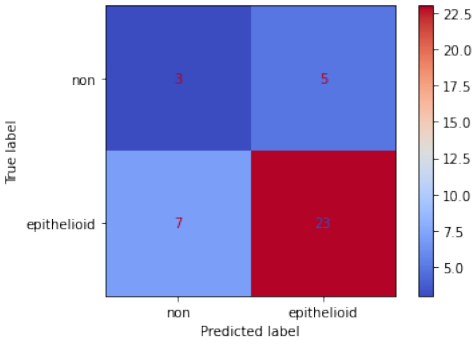


Figure 5.7. K-nearest neighbors

Finally, we made a comparison between all the algorithms tried. The histogram 5.8 takes into account the accuracy score, while the figure 5.9 shows the f1 score. As a further comment, logistic regression manages to have the best score despite the fact that it is only able to correctly predict the epithelioid labels.

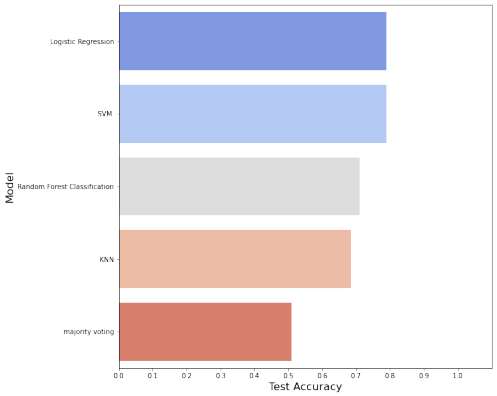


Figure 5.8. Accuracy for different algorithms

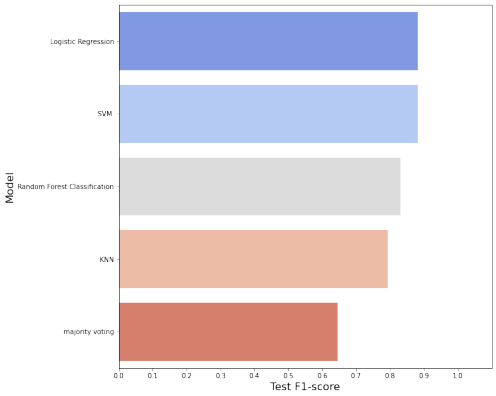


Figure 5.9. F1 score for different algorithms

5.1 Conclusion and Final Thoughts

In our opinion there are two main weak links in this project:

1. **Region selection**: The selection of the regions of interest is a task for a domain expert. To the best of our ability we tried to identify which were the patterns that best identify the type of mesothelioma, however our lack of expertise in this domain makes us prone to errors.

2. **Limited Resources**: Due to Colab's run-time constraints we had to perform the segmentation step. This is something that wouldn't have been done since a lot of tissue information is lost here.

BIBLIOGRAPHY

[1] Asap library. https://computationalpathologygroup.github.io/ASAP/.

[2] Histomicstk library. https://digitalslidearchive.github.io/HistomicsTK/index.html.

[3] Openslide library. https://openslide.org/.

[4] Tensorboard library. https://www.tensorflow.org/tensorboard.

[5] Tensorflow library. https://www.tensorflow.org/.

[6] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique.