Cluster CT Scans Tumors using Deep Learning Features

Yuval Ishay 206327553, Dor Lotan 205431158, Michal Shuvi 313450363

1. Introduction

Current medical science recognizes several types of tumors: Lymphoma, Carcinoma, Sarcoma, etc. These are used for both diagnosis and treatment of cancer patients. However, it could be the case that these categories are sub-optimal in this regard, and there possibly exists a superior categorization. This is Prof. Ilan Tsarfaty's working hypothesis.

For the purpose of finding this superior categorization, we have joined Tsarfaty's research team, with the goal of using deep learning tools to further their research. Our task, essentially, is clustering of CT image scans of lab mice afflicted with tumors, provided by Tsarfaty's lab, in order to reveal novel features of the tumors.

Moreover, such a categorization should correlate to specific Phenotypes of the mice. The lab mice are grouped into nine genetic lineages, and if our hypothesis is correct then the clustering will result in two or three distinct clusters. Hence, we use a clustering algorithm without an a-priori number of clusters as an input (such as k-means).

Currently, such scans are analyzed not with ML tools, but with classical image analysis. There are several features that radiologists use in their analysis and research, i.e Radiomic Features. These features, if treated as a feature space, would also allow for clustering of our images. These radiomic features can be extracted using the **PyRadiomics** library. However, this would lead to a clustering similar to the classical clustering we are attempting to avoid. Hence, we would need a new feature space.

To arrive at such an unknown feature space, we present three **convolutional neural networks**, each with a different task. That approach can lead to varied clusters and to several conclusions regarding them. Each network is trained on the data, and embeddings of the scans are extracted and flattened from one of its hidden layers as which we use for the purposes of clustering. This clustering will be evaluated using domain-specific metrics; Kaplan Meier analysis and QTL. Kaplan Meier creates a survival over time function for each group (Kaplan Meier Estimator), and our goal is to have distinct survival functions for our clusters. Additionally, we will analyze our results using QTL. That is, measure the genetic similarity between mice of the same clusters.

2. Dataset

The Data that was given by Tsarfaty’s lab contains 28 mice CT scans with tumors that have varied genetic backgrounds - specifically, 9 genetic lines. The data was divided into around 23k

CT-Scans DICOM files and 6k Masks MHA files. Each DICOM file represents a “slice”, for our purpose - a 2D scan image, and so concatenating these files in a specific order yields a 3D image. Almost all of the mice had 812 or 890 DICOM files.

Each mask matches a scan and specifies the location of the tumor in it, by containing the value 255 in a pixel with a tumor and 0 otherwise. Scans without any tumor did not have matching masks, meaning overall 6k of the 23k CT-Scans contained tumors.

DICOM files are images saved in the Digital Imaging and Communications in Medicine (DICOM) format. Each one contains an image from a medical scan but may also include identification data for patients. This format, however, is different from ordinary image files (for example PNG) which are often required when using deep neural networks, and specifically CNN. This was our first challenge.

In DICOM format, the pixel values are in units called Hounsfield, in the range between -1000 and 3000. In this format, images cannot be displayed, and many image processing operations may fail. Therefore a conversion to a positive 1-byte integer (between 0 and 255) was necessary. In the beginning, we decided to normalize the values and multiply by 255; a simple and common method. In our case, the images are divided into patches before being passed to a network, and therefore additional normalization is required, at the patch’s level. The conversion returned images that lacked contrast between their pixel values, and consequently yielded very problematic patches. We still tried to use this data, in several networks and in several patch sizes, but without succession. That is when we understood that we must find a better conversion, mainly regarding contrast.

The next try was to use SimpleITK’s built-in function, IntensityWindowing, which functions as a linear intensity transformation between ranges and therefore is supposed to increase the contrast between values. The result was significantly improved but was still a bit lacking. We then found “Aspose.Imaging for .NET”, a powerful API, that is capable of processing both commonly used formats and some special formats including DICOM. Using this API, we were able to make a much better conversion, and therefore this conversion was eventually chosen.

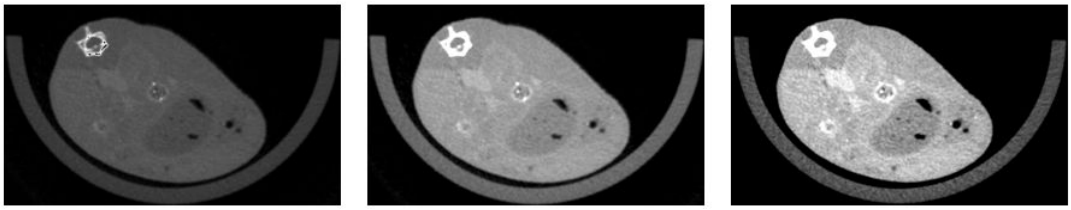

Figure 1: Tested conversions, first to final (left to right)

The second issue we had to deal with was the different sizes of the CT scans, in order to cluster these different sized images according to the same set of patches. Therefore, we resized our images and masks, and their current size is now the same, 300X300 pixels. We used a function

that downsamples the image preserving its coordinate system properties. It is done by, among other things, using a transformation that maps points from the old coordinate system to the new coordinate system, and interpolator - a method for obtaining the intensity values at arbitrary points in the coordinate system from the values of the points defined by the image. From our tests, the damage in terms of network results was relatively minor. Still, in case the new proportion would somehow damage the professional medical analysis that lies ahead, our code allows the user to do the same process again without resizing.

Using this preprocessed data, we eventually created two main datasets:

1. 2D patches: around 16k 50X50 tensors. Each tensor was extracted from a PNG file (representing a single scan) and was paired with another 50X50 tensor containing the matching mask. 65% of the data used for training and the rest for testing.
2. 3D patches: around 5k 30X30X30 tensors. Each tensor is a concatenation of 30 30X30 tensors (each one extracted from a PNG file) and is paired with another 30X30X30 tensor containing the matching mask. 65% of the data used for training and the rest for testing.

At first, we used 30X30 tensors for 2D patches as well, but the results were not as good as 50X50 patches, probably due to a lack of information.

The motivation for using patches of a scan slice instead of the whole scan slice is medical; A tumor has a shape but also a homogenous pattern inside it which can be examined in the scan. We would like to focus the model on the tumor pattern and not only the shape, so that the model will find interesting features of it. Using the whole scan slice as an input to the model could have caused the model to learn too general features of a tumor; color, shape, etc; which is not what Ilan's lab is aiming for. In addition, from a technical perspective, there is not enough data of that kind.

The next issue we were facing was "tumor leftovers" mainly at the edges of "healthy" patches. We had to make sure that enough deep information about tumors will be passed to the networks, so we could not, for example, choose to create a dataset in which half of the patches contain a tumor and half not. We needed to consider the percentage of the tumor in a patch. Therefore we created our datasets in a way that half of the patches contain at least a specific percentage and half not. The specific percentages we chose were 30% for the 2D networks, and 20% for the 3D network (simply since there is not enough content for 30% as well).

Our last main decision regarding the datasets was the way to split the data into train and test. Our plan was to split them according to the mice's identity, meaning all the patches of a specific mouse will be a part of either the train or the test data. However, after reconsidering it, we understood that these networks are just a part of the final purpose of this project - the clustering. The clustering needs to include all the mice. Sticking to the plan means giving up on some mice - since we can not train a network with some data that we also plan to test (the clustering is based

on the networks). That is why we chose eventually to split the data across the mice - meaning patches of a certain mouse will be found in both train (65%) and test (35%) data. In the 2D dataset, since close slices are very similar to each other, for every mouse - the patches in the train data and test data are from disjoint ranges.

3. Methods

In this section we present three methods, each method leads to a different feature space and to different clusters. The main motivation for this is to provide varied options to be examined in Ilan's lab so that at least one of them, hopefully, will lead to significant and novel results.

3.1 Classification

One task that suits the mice dataset is classification of patches: a patch that contains a tumor will be labeled as 1, and 0 otherwise. As described in the dataset section, a threshold was set to determine the patch label. Due to the size of the patch, the model lacks spatial information, which means it treats each patch independently. This fact makes the task more difficult and might force the model to identify different tumor patterns instead of a general shape of a tumor.

3.1.1 2D Classification

The first model is a two-dimensional classifier with a label threshold of 30%. I.e. patch will be considered as a tumor if it has more than 30% tumor pixels. The model architecture was inspired by Tiny ImageNet (Yao & Miller, 2015) due to the fact that the size of the dataset images and the goal of the network is similar to ours. The network consists of 12 layers, all conv layers are with kernel size of 2x2:

- Conv layer with 16 filters with stride of size 2.
- Leaky ReLU layer with negative slope of 0.2
- Conv layer with 32 filters with stride of size 1.
- Batch normalization layer
- Leaky ReLU
- Max pool layer
- Conv layer with 64 filters with stride of size 1.
- Leaky ReLU
- Max pool layer
- 3 fully connected layers: $3136 \rightarrow 1500 \rightarrow 750 \rightarrow 2$

Binary cross-entropy loss was used after running softmax on the net results. Without max-pooling layers the model suffered from early overfitting, adding those layers improved training. Additional experiments were training deeper networks or adding more filters but those did not improve results in terms of test accuracy and test loss.

We trained the model for 50 epochs but only the best result was saved from epoch to epoch. i.e. in practice the number of epochs can be smaller. The model weights were initialized as $W_{ij} = N(0,\ 0.2)$. We experimented with Xavier initialization as well, but that did not improve convergence time nor accuracy results. We chose Adam optimizer (betas are (0.99, 0.999)) and L2 regularization with $\lambda = 1 \bullet 10^{-5}$ to encourage the model to generalize better. Furthermore, Learning rate scheduler was used, the learning rate starts at 0.0002 for 10 epochs and decays to zeros in the remaining 40 epochs.

After training the model for 50 epochs, the best test accuracy was 86.53% within 12 epochs, then the model starts overfitting. The accuracy is pretty high and that made us think about a possibility that the model could have learned some kind of heuristic feature, i.e. a simple rule that is good enough in the classification case; for example, the color of the tumor in comparison to the color of the mouse.

The embedding vector was extracted from the second fully-connected layer (fc2), with a size of 750. We decided on this layer to represent the patch, because it seems to contain the needed information for the classification task and yet the embedding vector is not too large, i.e. there are less unneeded features. One main drawback of that technique is that the spatial information inside the patch is lost.

3.1.2 3D Classification

The second model we could think of is a classification of 3D patches. As mentioned in section 2, a CT scan is 3D and each DICOM image is a "slice" in a 3D image. It means that more information about the tumor is embodied in the third dimension too. We would like to create a model that uses the additional information, this model may discover different and interesting features that can only be learned thanks to the third dimension.

Model architecture is similar to the 2D classifier model architecture except that every 2D layer becomes 3D: convolution layer 2D → 3D, Max pool layer 2D → 3D and batch normalization 2D → 3D. Furthermore, L2 regularization was dropped and fully connected layers have different sizes: 4096 → 2045 → 1024 → 2. Initialization and training are identical to the 2D classifier. This model reached test accuracy of 89.22% after 13 epochs and embedding vectors were extracted from the second fully connected layer with size 1024.

3.2 Segmentation

Another task is **segmentation**: For each patch, we can have our network generate a mask, such that each pixel of the mask is 1 if it belongs to a tumor, and 0 otherwise. Since these masks already exist in our dataset, we could take advantage of that data to train a segmentation network.

Also, this is a much more difficult task than classification: while classification learns to recognize the presence or lack of a tumor, segmentation involves finding its location as well. Hence, we can assume a segmentation network could learn a much more robust representation.

The segmentation network's architecture is a smaller version of Unet (Ronneberger, Fischer and Brox 2015) because of the size of the data instances. This consists of four downsampling layers, followed by four upsampling layers which use several outputs, thus creating a "U-shape". In addition, we used less filters than the original implementation due to the patch size; we used the following growing filter sizes: 16, 32, 64 and 128. The downsampling sequence can be viewed as an encoder, and the upsampling sequence is similar to a decoder, although it cannot be applied to the direct output of the decoder. For our clustering purposes, that encoder is the vital part, as its final layer provides us with embeddings.

To train and evaluate the network, we use a loss function and accuracy measure which are specific for segmentation. For evaluation, we use the Jaccard Index, or Intersection over Union metric. That is, given a target mask A and predicted mask B, their similarity will be measured as:

$$Jaccard(A,B) = |A \cap B|/|A \cup B|$$

To calculate the Jaccard index, we also round the pixel values of the predicted mask: all values above 0.5 are treated as 1, and the rest as 0. Hence, this measure is non-differentiable and cannot be used as a loss function for training. For a loss function, we use the Dice Coefficient (a.k.a Sørensen index), which is similar to Jaccard Index, however, it is calculated slightly differently. For a target mask A and predicted mask B, the Dice Coefficient is defined:

$$Dice(A,B) = 1 - \frac{2(AB)}{(A^2+B^2)}$$

Note that for A=B this is minimized to 0, yet unlike the Jaccard Index it is differentiable. Both of these measures have the advantage of being based on the areas where the predicted and target masks are 1, and hence are not biased by the potentially large area where both the target and prediction are 0 (True Negative). A loss function based both on True Positive and True Negative would have given us a much lower loss value in the initial stages, somewhat undeservedly. The Dice Coefficient, then, is a more precise measure of our goal.

Training the model was quite similar to the training of the 3D classification tasks, except for the number of epochs, which was 100 in this case, and the learning rate which was static (at 0.0002) at the first 20 epochs and then started to decay to zero in the last 80 epochs.

The best test accuracy was 53% at epoch 67. For reference, a naive baseline using random masks on the test dataset yields 20.7% accuracy on average. This task is harder than classification because every pixel needs to be classified as tumor or no, so small visible errors can cause a higher loss value. furthermore, the tumors were segmented manually, and hence there might be inaccuracies or artifacts within our target masks. In figure 2 we can observe a few examples of

the segmentation model results. The last example is very interesting since the tumor border is not clear-cut like in the other results, still, the model separates the patch into two different areas.

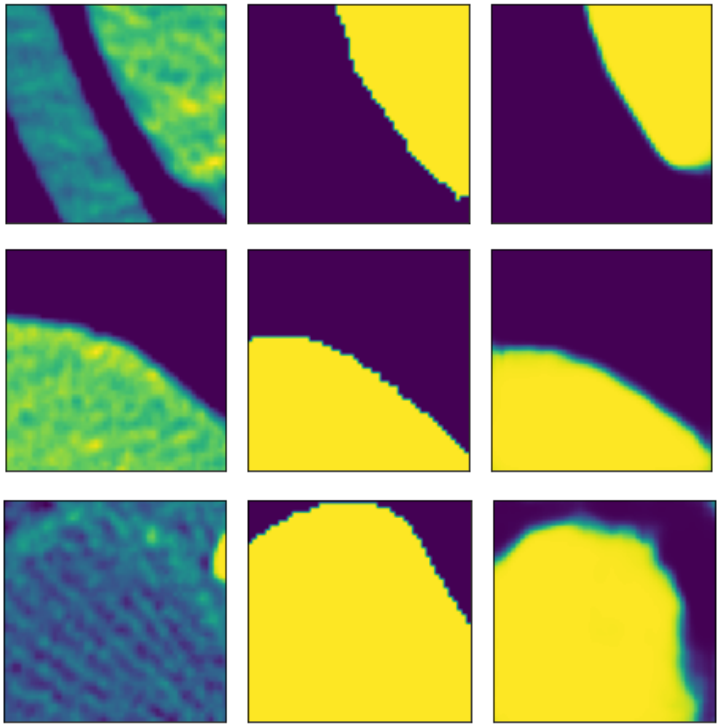

Figure 2: Examples from the segmentation results.
Left to right - CT scan, Ground Truth and Predicted mask.

4. Clustering and Dimensionality Reduction

Clustering was performed on four datasets, or more precisely four encodings of the same dataset: using the 2D and 3D classification nets, and the segmentation net. These could be very different from each other, and hence we need some pre-processing of our data, and a clustering method that is as general as possible.

To ignore the problem of scale, we use Cosine Distances instead of a Euclidean metric, as well as standardization. Standardization is a pre-processing step in which we subtract the mean of the data μ and divide by its standard deviation σ: $x' = \frac{(x-\mu)}{\sigma}$

This is done for each feature of the data. Cosine distance between two vectors u, v is defined as:

$$CosDist(u, v) = 1 - \frac{uv}{|u||v|} = 1 - cos(\theta)$$

Where θ is the angle between the vectors. This ignores the different scales, and allows us to cluster the samples based not on **distance**, but on distribution **similarity**. The clustering algorithm we use is Density Based Spatial Clustering of Applications with Noise, or DBSCAN (Ester, Kriegel, Sander, Xu, 1996), with different parameters for each encoding. DBSCAN iterates over each sample in the dataset, and gets its neighbors that are ε distance away. If there

are more than MinSamples such neighbors, then these are defined as a cluster which is then expanded in the same manner. See appendix for pseudo-code.

In other words, we check every point for being a "Core" point, and if so we use it to expand its' cluster further. There are three advantages to this algorithm. It:

1. Does not assume how many clusters we end up with (unlike EM or k-means).
2. Allows us to label some of the points as Outliers - points which do not fall into any cluster.
3. Does not assume a "shape" for the clusters, the way k-means assumes round clusters and EM assumes some distribution (e.g. Multi Gaussian).

These advantages allow us to cluster our four different encodings with the same algorithm, as it generalizes well to different forms of data. However, there are some disadvantages:

1. Both parameters, ε and MinSamples, must be set manually. MinSamples has a recommended value of $2d$ where d is the dimension of the data, but for ε this proves more difficult, as these values are handpicked. See the table below.
2. Hard to evaluate, as most evaluation metrics are based on distances and density of convex clusters.
3. Nondeterministic: clustering depends on scan order.

The disadvantages are mostly technical, and are far outweighed by DBSCAN's advantages for our purposes. The difficulty in evaluation, specifically, is solved by our domain-specific evaluation using KM and QTL.

2D classification	3D classification	Segmentation
ε = 0.2745 MinSamples = 100	ε = 0.15 MinSamples = 40	ε = 0.39479 MinSamples = 100
ε = 0.2 MinSamples = 100	ε = 0.10133 MinSamples = 20	ε = 0.38 MinSamples = 100
ε = 0.1 MinSamples = 50	ε = 0.1 MinSamples = 20	ε = 0.2 MinSamples = 50

Table 1: different clustering parameters for each model.

Another task is a reduction in dimensionality. This is mainly for visualization purposes, but can be done prior to clustering as well (for example, to have all feature spaces share the same dimension). We could reduce the dimension simply by using PCA, but this implies a strong

hypothesis: that the data has some degree of linearity. This linearity would. Again, be dominated by the large magnitude features we dealt with while clustering. Since PCA is not strictly reliant on distance, cosine distance is not directly applicable.

To solve this problem, we reduce the dimension of the data not with PCA, but with Feature Agglomeration. First, we transpose the dataset: from having n samples in m dimensions, we have m samples in n dimensions. In other words, our new "samples" are the features of the original dataset. Next, we cluster the new dataset, using Hierarchical Clustering with cosine distances. The idea here is to find clusters of features that have similar distributions. Finally, we average each cluster to create a set of centroids. When we transpose the data back, we are again left with n samples, but in less dimensions. Specifically, the new dimension of the reduced data is the amount of clusters we found. We reduce the dimension to MinSamples divided by two.

Feature Agglomeration has a low degree of information loss: since the features that are clustered together behave in a similar way, this has the effect of uniting the two most correlated features whenever two are clustered together, as the clustering algorithm behind this is hierarchical. Hence, we can be confident that each merge is well justified, and that our resulting reduced data is an appropriate low dimensional fit. Currently, we use it to fit our data into 50 dimensions prior to clustering, to allow uniform dimensionality across the data. We then reduced that data down to two dimensions with PCA for visualization.

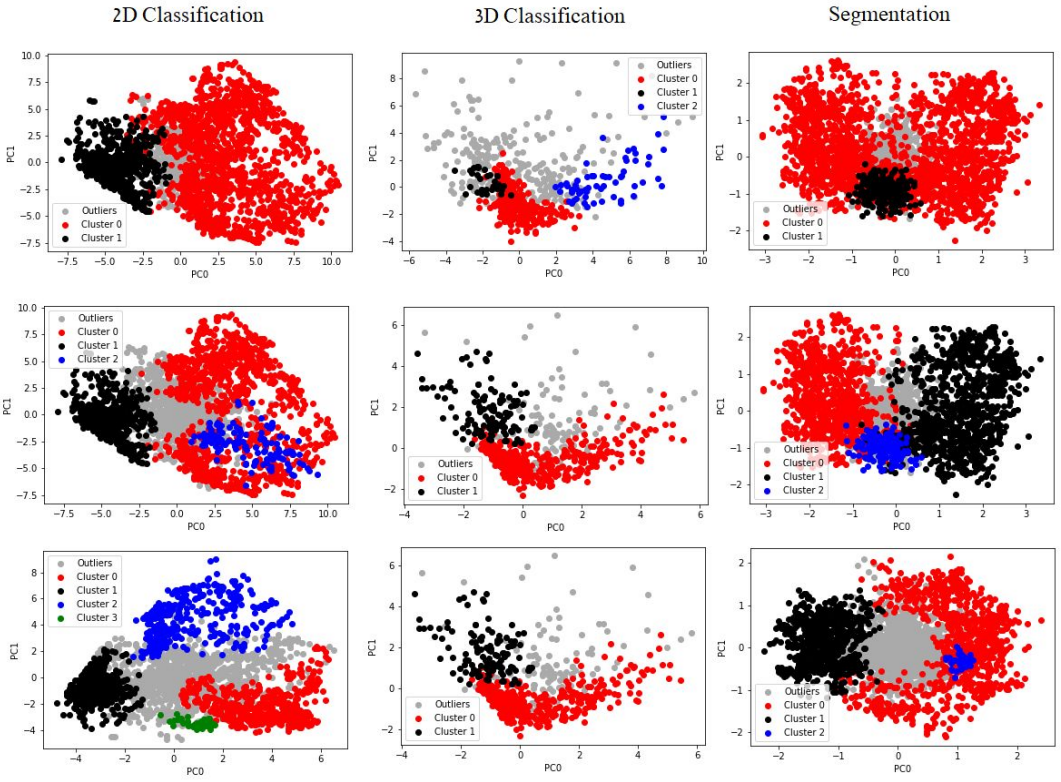

Figure 3: 2D visualization of different clustering output, clustering parameters respectively to table 1.

As is shown in figure 3, different values of ε and MinSamples lead to different amounts of clusters. Since we use a non-euclidean distance metric, these clusters may seem counter-intuitive.

These clusters do contain differing amounts of noise. DBSCAN makes it difficult to evaluate these clusters by their own geometry: most clustering evaluation metrics are based either on ground truth or on the density of the clusters. Here, we have no clear ground truth to use, and the non-convexity of DBSCAN clusters leads to inability to use density based methods.

It was unclear whether a specific clustering is successful or not. We saw, for example, that in one of the clusterings, that was based on the 2D classification network, the tumor patches were divided into a cluster with around 50% tumor mean and a cluster with 95% tumor mean. There was a possibility that one of our networks, especially the 2D classification, learned some irrelevant heuristic. However, From our conversations with Prof. Tsarfaty, this might be expected; the inner and outer parts of a tumor look very different. Hence, our distribution of tumor patches contains at least **two different sub-distributions**: edges and cores (and possibly more). The networks, in turn, must classify both as tumors. This seems to be the cause of the two emerging clusters.

We, therefore, needed some visual way to observe our results. Printing the patches, according to clusters, by itself was not enough, since it was out of the context of the whole slice; and it was also hard to compare patches from different clusters that were also taken from the same slice. Therefore, in consultation with Tsarfaty's lab for their future radiomic analysis, we added a feature that marks the tumor patches of a slice according to the clusters. Our ability to professionally analyze the results is of course partial, but still, we can carefully say that we understood some principles, which can be seen in figure 4.

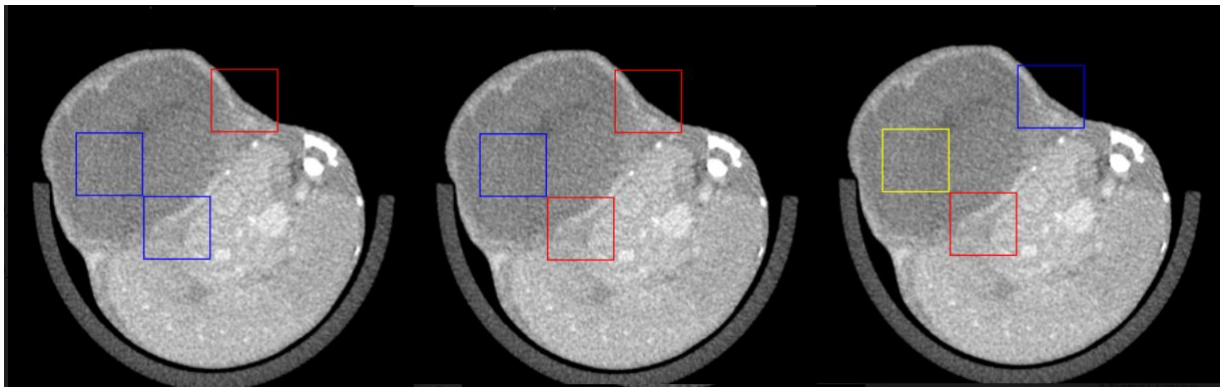

Figure 4: Marked tumor patches of some slice; different colours represent different clusters
From left to right: classification 2d, segmentation with 2 clusters, segmentation with 3 clusters

Our edges-cores hypothesis seems to be incorrect for classification at first glance. However, note the three types of tumor patches seen here: tumor core (yellow in the rightmost image), inner edge (red), and outer edge (blue). By inner and outer we mean that the outside of the tumor is within the mouse or outside of it. For classification, this distinction seems to be more relevant: both inner parts are blue, while the outer part is red. For segmentation, however, the core-edge distinction is more significant.

These observations of course can't replace a real evaluation of the clusters, and like mentioned, since we have no way to evaluate those clusters in the current state, we need to use domain-specific evaluation metrics, which for our case is KM analysis.

5. Evaluation

In this section, we attempt to evaluate our results using domain-specific metrics; KM and QTL. We present these metrics, their implications for our results, and further conclusions.

5.1 Kaplan Meier

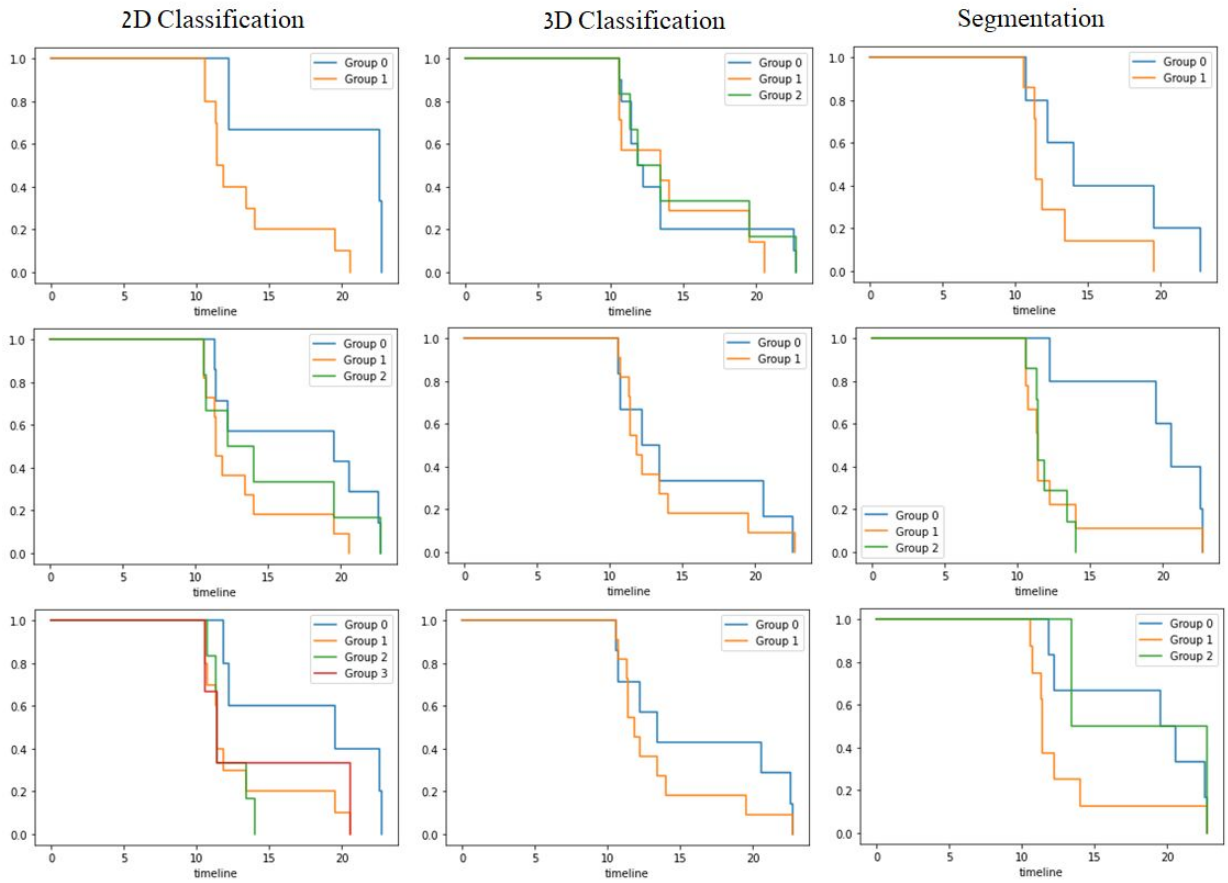

Figure 5: KM graph for each clustering result, clustering parameters respectively to table 1.

The Kaplan-Meier Estimator allows us to estimate a survivability function for members of a given group, given their death time. In other words, a function from time to survival probability. If these groups are sufficiently different from each other, then these graphs should be further apart. For our purposes, mice can belong to multiple groups at once. The KM plots for our clusterings can be viewed in figure 5.

For many of the clusterings, we obtain graphs that are very close to each other, such as Classification 3D no. 1 and 2. However, most of the segmentation plots as well as classification 2D no. 1 outline two groups where one has a higher rate of survival than the other (especially in no. 2). Classification 2D no.2 and 3 divide the mice into three such groups, where one is a median between the two extremes.

For each graph, we measure significance with p-value, and aim for lower than 0.05. The clusterings that lead to p-values smaller than 0.05 are:

1. 2D classification with eps=0.1, min_samples=50, when comparing group 1 to the other groups, the p-value is 0.039.
2. Segmentation with eps=0.38 and min_samples=100, group 0 yields a p-value of 0.008, and group 1 yields 0.047. This result is our most significant result, and the KM plot reflects this fact, as group 0 is completely separated and furthest away from the other two groups.
3. Segmentation with eps=0.2, min_samples 50 gives us a clustering where group 0 yields a p-value of 0.013.

The edge-core hypothesis could be the cause of this; The edges are described by Prof. Tsarfaty as the active area of the tumor. Hence, a mouse with more patches belonging to this group is different from a mouse with mostly inactive patches. These two mice, in turn, would have different survival rates and might, themselves, belong to genetic lines which are prone to one type of a tumor or the other. That last part is Prof. Tsarfaty's primary research, for which our neural clustering tool was built. It is also important to note that the amount of mice we are working with is rather low, hence there should not yet be high confidence in these results, and more rigorous testing is needed.

5.2 QTL Analysis

QTL (Quantitative Trait Locus) is a measurement of genetic similarity between members of a given group. For each gene, we measure how many members have it out of the entire group. Our hypothesis supposes that there are clusterings which are predictable genetically: there exist some specific genes which correlate highly to them.

For some of our clusterings, there were specific chromosomes that were significant in each cluster. For example, segmentation with eps=0.2 and min_samples=50 resulted in chromosome number 16 being dominant in group B and 12 in group C (Group A consists of outliers).

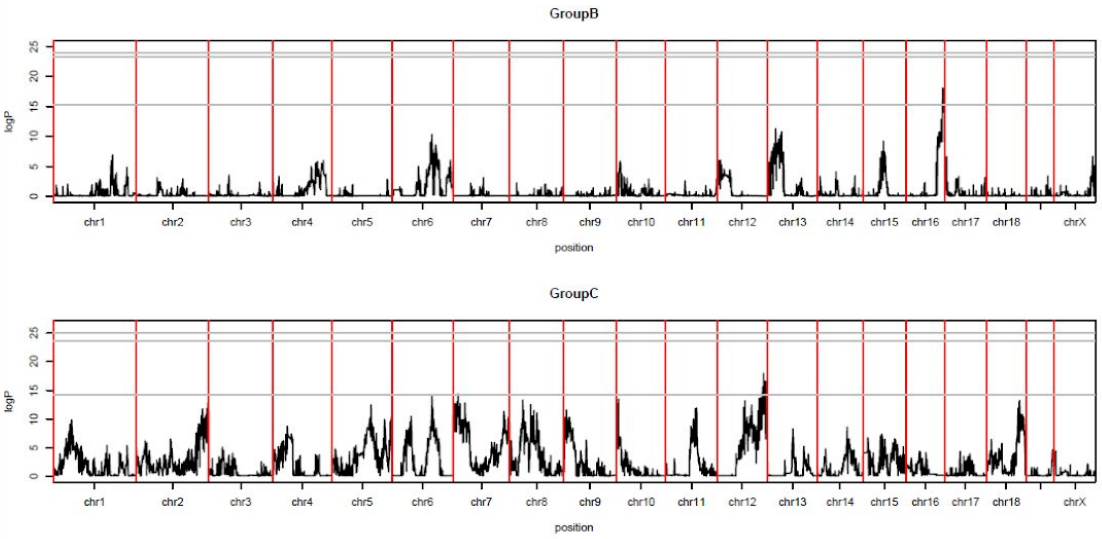

Figure 6: QTL mapping for segmentation, eps=0.2, min_samples=50. The horizontal axis contains different chromosomes, the vertical axis is the log of significance. The grey lines are, from bottom to top, 50%, 90% and 95%.

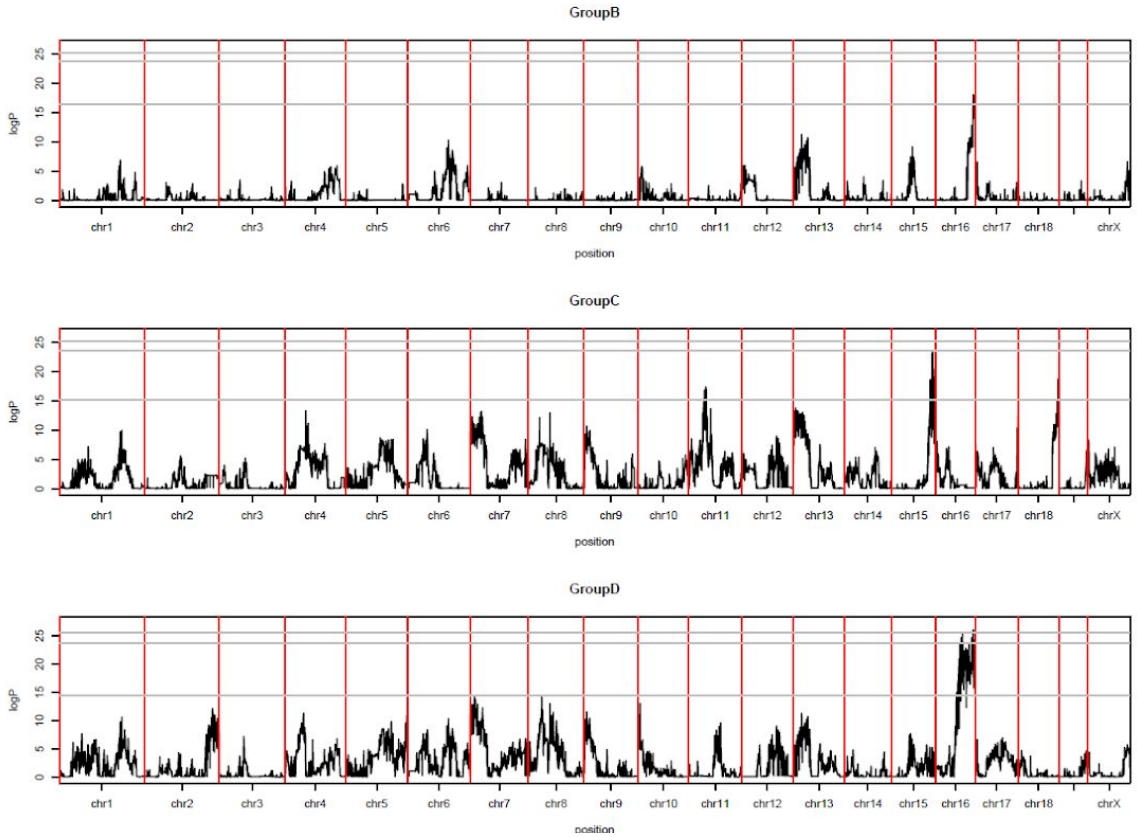

Figure 7: QTL mapping for 2D classification, eps=0.1, min_samples=50.

In figure 7 a similar result can be viewed, it was produced with 2D classification with eps=0.1 and min_samples=50. This time, chromosome 16 being significant in groups B and D and chromosome 15 significant in group C. Group C corresponds to group 1 of the KM analysis, which yielded a low p-value. This is the group where chromosome 15 is highly significant, around 90%.

Our most significant KM result also shows significance of chromosome 16 in group B (group 0 in KM):

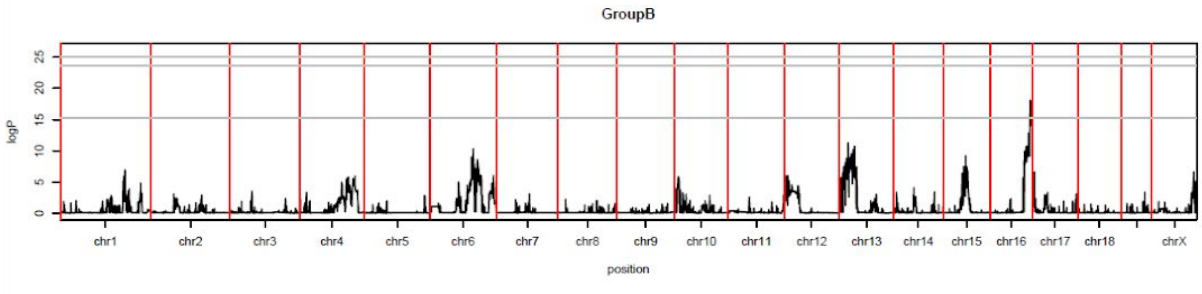

Figure 8: QTL mapping for segmentation, eps=0.38, min_samples=100.

As can be seen from the QTL analysis in figures 6-8, chromosomes 12, 15, and 16 are the ones found to be most significant in the different clusterings. These clusterings, when analyzed using KM (fig. 5) show a correlation between these chromosomes and survivability. Specifically, chromosome 16 seems to correlate with higher survival rates over time.

These results will serve as a basis for future research from Prof. Tsarfaty, and there may be new results. Also, there is a need for further lab research on our result, for which we provided the ML based tools.

6. References

1. Olaf Ronneberger, Philipp Fischer and Thomas Brox 2015, *U-Net: Convolutional Networks for Biomedical Image Segmentation*

2. Leon Yao and John Miller. *Tiny imagenet classification with convolutional neural networks*. CS 231N, 2015.

3. Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). *A density-based algorithm for discovering clusters in large spatial databases with noise*.

7. Appendix

A. DBSCAN pseudo code:

```
C := 0 //Cluster index
For each sample S in data:
     If Cluster(S) ≠ Undefined then continue //Already assigned
     Neighbors N := GetNeighbors(S, eps) // Get a set of samples
that are eps away from S
     If |N| < min_samples then:
          Cluster(S)   :=   Outlier   //S   doesn't   have   enough
neighbors to be in a cluster
          Continue
     C := C + 1 //Start new cluster
     Cluster(S) := C
     Expanders E := N / {S}
     For each sample S' in E:
          If Cluster(S') ≠ Undefined then continue
          Cluster(S') := C //Add S' to current cluster
          Neighbors N' := GetNeighbors(S', eps)// Get a set of
samples that are eps away from S'
          If |N'| ≥ min_samples then:
               E := E ∪ N' //Add N' to the expanding cluster
```