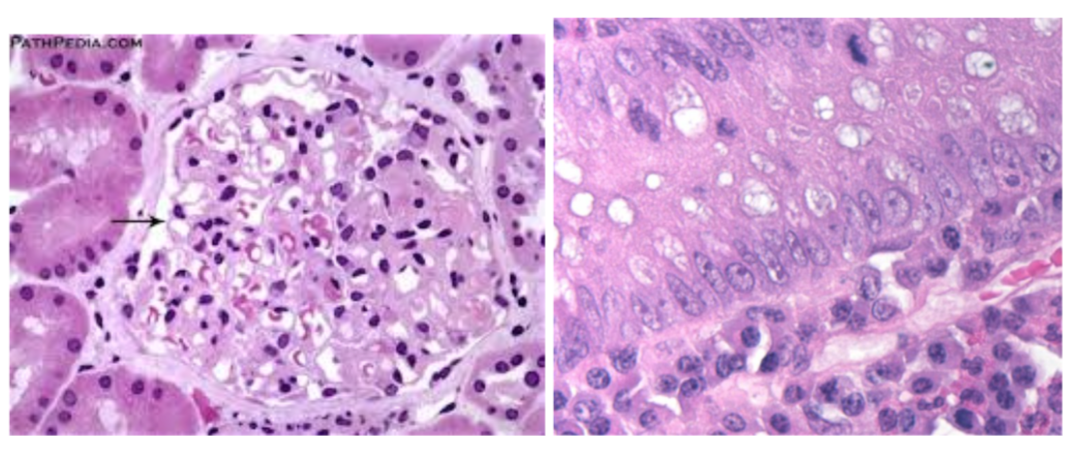
**Breast Cancer Pathology Project**

**Ron Tsibulsky Or Lichter**

## General Background:

In Professor Ilan Tzarfati’s lab, mice were split into groups, each group having specific genetic sequences that are suspected to be related to breast cancer development, and then bred. A group of offspring with the same genetic sequence (identified in their ancestors) is called a line. Tissue from the offspring was taken, stained in H&E, and classified by two professional pathologists as different types of breast cancer. Each image of the tumor is about 1.5 GB in size.

An example of a histopathologic image from google:



## Goal:

Given pathological images of the tumors our goal is to divide the mice into groups by clustering (each mouse might have a few tissue images) and then divide each line into clusters as well.

## Key Metrics:

QTL (Quantitative Trait Locus) Analysis - A statistical method that links two types of information—phenotypic data (trait measurements) and genotypic data (usually molecular markers)—in an attempt to explain the genetic basis of variation in complex traits. We used QTL analysis to measure the line clustering.

Kaplan-Meier estimate - A non-parametric statistic used to estimate the survival function from lifetime data. We used this metric to measure the individual mouse clustering.

### ***A deeper dive into QTL and how we would use it:***

QTL is a statistical method used to identify DNA snippets that are correlated with genetic traits (skin color, leaf size etc.). Ilan and his lab identified a few such snippets that might be correlated with the growth of breast cancer. The lines were breed in order to have some of these snippets. The goal of this project is to see if we can group these lines in a way that the QTL graph (the output is a graph) shows significant differences for the DNA snippets.

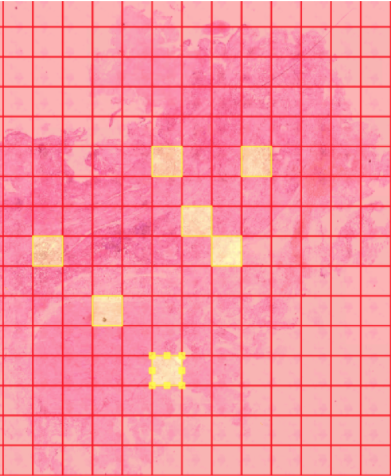
## First Benchmark

As explained above, we would like to cluster histopathological images of tumors from mice in order to try to split them into groups indicating their genome traits (i.e. having a specific genome snippet).

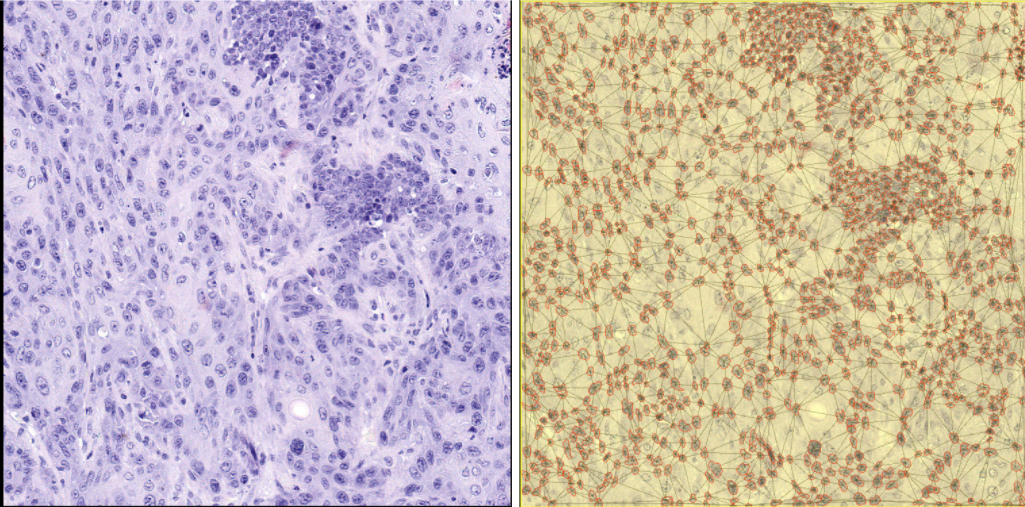
Our process for the first benchmark is a very naive approach using regular image processing tools and K-means.

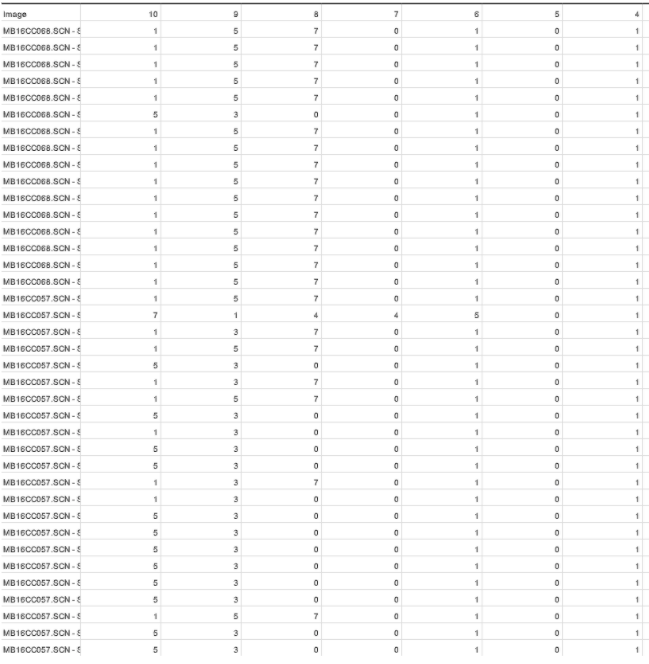
QuPath is a software used by pathologists to diagnose pathological images. We used a few scripts made by Prof. Tzarfati’s lab for the first steps:

1. **Split the images into smaller pieces:** A histopathological image is in a very high resolution and is very big in size and can reach up to 6 GB per image. In order to overcome this, Prof. Tzarfati’s lab has created a script that tiles the huge image into 2000x2000 pixel images (about 10 MB per image). Out of the tiled image, we chose and saved 15 random tiles, which by Prof. Tzarfati’s estimation is about the number of tiles to approximately portray the image.

1. **Identify and extract the nuclei from the image:** Another script the lab has created is a script that uses regular image processing tools in order to segment nuclei an image and extract details per nuclei such as x,y of the nucleus center, nucleus radius, and more. All in all, there are about 1400 features extracted, not all of which are important.



1. **Averaging the nuclei details per tiled image**
2. **K-Means:** We considered reducing the dimensions by applying PCA as well but 1400 features are no problem for k-means.
3. 
4. **Get the QTL analysis results:** As described above, QTL analysis will allow us to estimate how good our clustering is.. From our results, it can be easily seen that images from the same scan are largely clustered the same.

# Process

After getting familiar with the data, the process and having some ground results, we are diving deeper and trying some more advanced steps. In this step we decided to cluster each **nuclei** and not just each image. In order to do this, we had to handle large amounts of data (such that couldn't fit into RAM at once).

## Data Size

In the first step we explained about Qupath - the pathology tool that uses classical image analysis tools in order to extract features (around 1400) for each nuclei. So for each nuclei we have 1,400 parameters, in an image there are about 6-10 thousand nuclei (for rough calculations, let's say 8). We have about 45 images per slide and 70 slides. All in all 45,708,000 = 25,200,000 nuclei so our data set is 25M X 1400 which definitely can't fit all into memory.

## Scaling

One of the first steps in any ML algorithm is to scale the data. This makes the learning process a lot less susceptible to scale difference, for example if one feature is nuclei size which is in scale while another feature is nuclei blue color which is a number from 1-255, then a small difference in the hue of the nuclei might cause a big difference to the learning model. Here we first meet our data size problem. Some scaling algorithms such as RobustScaler (which uses quantiles in order to remove outliers before normalizing to standard distribution) need to have the whole data in memory in order to work properly. Luckily some of the more simple and familiar scaling algorithms keep a few parameters that can be calculated over the data iteratively and do not need the whole dataset in memory. We have used [MinMax scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html?highlight=minmax#sklearn.preprocessing.MinMaxScaler) and [standard scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html).

### **MinMax Scaler**

From its name it's not too hard to uderstand what it does. This scaler goes over all data (fit), keeps in memory the maximum and minimum of each parameter. After going over all data it goes over all data again (transform) and transforms the data between 0 and 1. Specifically this is the formula for the calculation:

You can find the trained scaler in our repository under **Pickled Data MinMax\_scaler.pickle**

### **Standard Scaler**

This scaler takes each feature of the data and transforms it to be normally distributed. The process is still pretty simple - over the first iteration of the data (fit) the scaler calculates the mean and the standard deviation. On the second run of the data (transform) it scales the data to be normally distributed:

You can find the trained scaler under **Pickled Data standard\_scaler.pickle**

## MiniBatch Kmeans

Most clustering algorithms need to have all the data in order to work. Kmeans, needs all the data in order to perform the iteration on centers and to calculate the distances, DBSCAN needs all data in order to count the number of data points in epsilon distance and so on... After some searching we found [MiniBatch Kmeans](https://scikit-learn.org/stable/modules/clustering.html" \l "mini-batch-kmeans). The algorithm assumes we sample iid samples from the dataset and performs Kmeans on them while keeping the center average of all sampled data. Using this algorithm, we managed to cluster the huge data.

## Aggregating the Data

After clustering the nuclei, we had to determine a clustering for each image. In order to do so, we decided on three methodologies:

1. **Majority Vote**: Clustering each image with a "majority vote" of the image's nuclei clusters. For examples, if the nuclei are clustered in groups 0-4, and for an image most of the nuclei are clustered to group 3, then the entire image is clustered to group 3. Specifically, it means the number of possible clusters for images is the same as number of possible clusters for nuclei. The code for this can be found in file **Scripts Aggregating Majority\_vote.py**
2. **Histogram**: Here we have calculated the percentage of nuclei of each cluster at each image. This way we have created for each image a k-dimensional vector, where k is the number of possible clusters for nuclei, so that feature i in this vector represents the percentage of nuclei (in that specific image) clustered to group i. On these k-dimensoinal vectors we have run another K-means process to cluster the images. In this method the number of possible clusters for images does not depend on the number of possible clusters for nuclei. For example, on the nuclei level we can cluster to 7 groups, and on the image level to 4 groups only. We have ran all variations for number of nuclei clusters between 2 to 10, with number of image clusters between 2 to 10 (81 variations at all). This code can be found **Scripts Aggregating Creat\_hitogram.py**
3. **Image average**: We have clustered images "straight forward" (using K-means) without clustering nuclei before hand, by calculating an average on all nuclei for each image (The average was calculated feature-wise, so that if Qpath gave us for each nucleus an ~1400 dimensional vector of features, we calculate an ~1400 dimensional average vector for the entire image).

## Metrics

Besides evaluating our clusters on QTL and Kaplan-Meir analyses, we have tested our results using some common clustering metrices. These metrices require ground-truth labeling, which we don't have, so we did the following, knowing that the results of these metrices should be taken with limited importance:

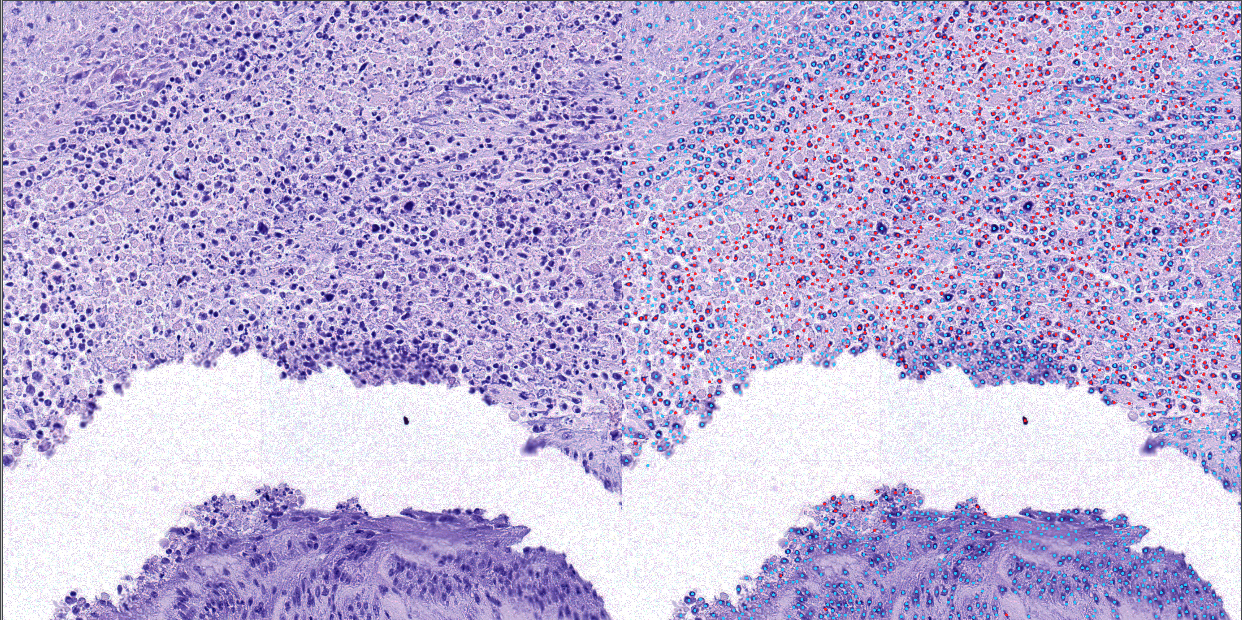
We have created a "ground-truth" labeling by looking at each mice line as a group, and clustering each image according to it's mouse's line. The metrices we have used are:

* NMI (Normalized Mutual Information)
* AMI (Adjusted Mutual Information)
* ARI (Adjusted Rand Index)
* Homogeneity
* Completeness
* V\_Measure (Which calculates an average of the above two metrices)
* Fowlkes Mallows

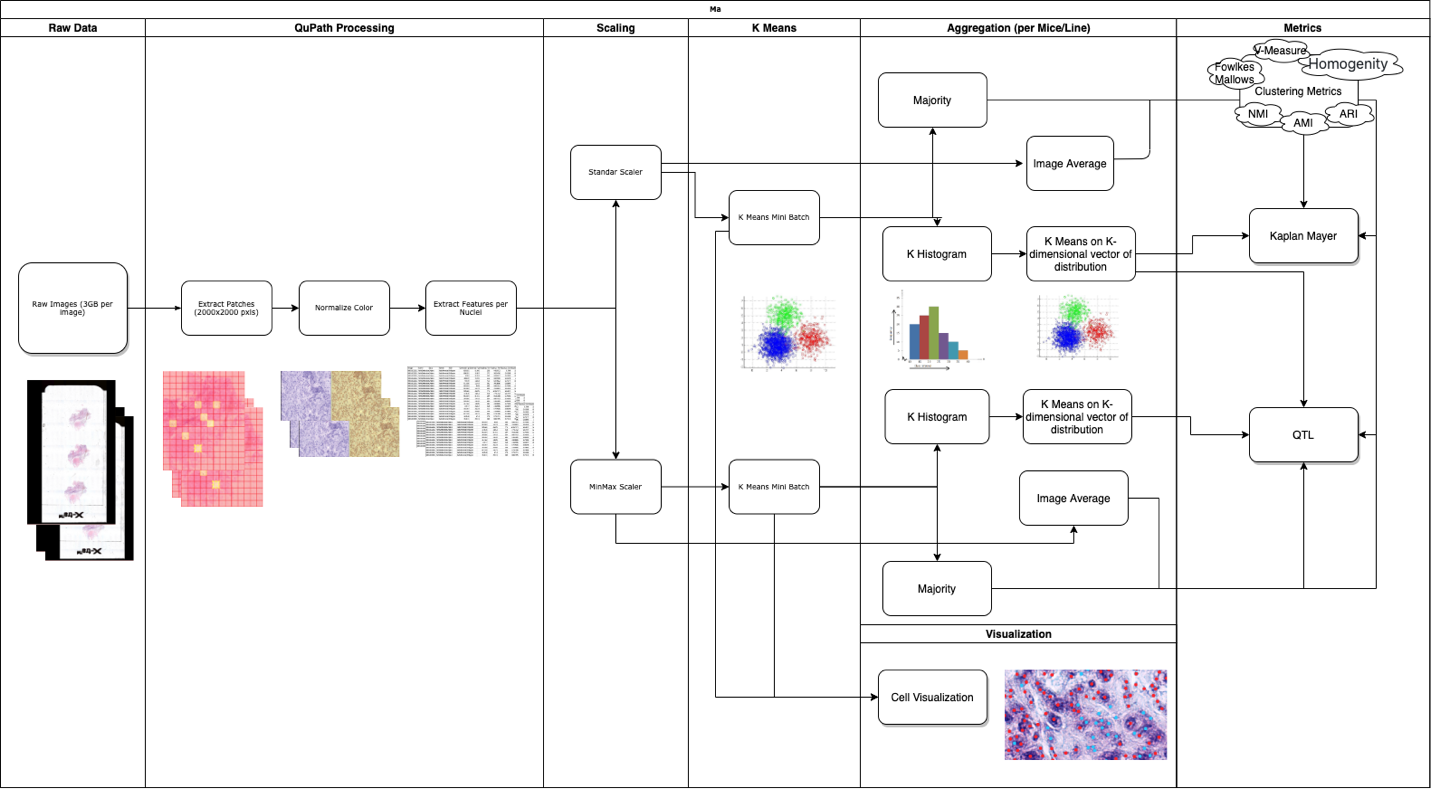
We are familiar also with metrices that do not require ground truth labels, such as Silhouette metric, clustered objects. Nevertheless, as it is not clear what would be a right distance function between the clustered images, and as calculating the distance of such a large dataset is computationally hard we have decided not to calculate these metrics.

## Visualizing the Data

One of the features extracted from QuPath is the (x,y) pixel center for each nuclei. We though - we have the center pixels, we have a cluster for each nuclei, why not try to visualize it?! So that's what we did. You can use the **Scripts Visaulization ImageChange2NucleiClasses.py** to download the data saved in my Google Drive, or refer it to a local path and get the following image:



## Process Flow



# Final Results

You can find our final results in the **Results** file in our repository. Just to give a few examples, the following is an example of a 3 cluster result that was scaled by standard scaler and histogramed into a 8-dimensional vector:

Chart, histogram

Description automatically generated

The P-Value you see in the image above is a Wilcoxon metric.

Also an example of the QTL results for a histogram of 8 dimensions and final 10 clusters:

A picture containing diagram

Description automatically generated

The results have been presented and submitted to Ilan’s lab and will be analyzed by them, as the have vast biological knowledge.

# Ideas for Next Steps

1. Feature Selection – As stated above QuPath outputs 1,400 features per nuclei. This might cause the “curse of dimensionality” meaning that the data points in the 1,400 dimensions will be sparse and far apart thus making it harder to cluster. Reducing the parameters might result in better clustering.
2. Dimensionality Reduction – As mentioned in the previous point, reducing the dimensions might lead to better clustering but classic PCA or other dimensionality reducing algorithms might cause for loss of variance which in this project we wanted to avoid. Nevertheless, we do recommend testing this approach.
3. Aggregating by mouse and not by patch – In this project we have clustered nuclei and the key metrics were meant for mouse and line clustering. In order to overcome this, we have done the intermediate clustering for the relevant resolution (this step is the “Aggregation” step). In this project we have aggregated up from nuclei resolution to patch resolution then for QTL have counted the patches to get line clusters and for Kaplan-Meier have given a majority score per mouse. Aggregating straight up to mouse or line without the intermediate patch step might give better results.
4. Replace K-means with a neural network – In the scope of this project we haven’t had the capacity to test this approach, but as Deep Learning advances it looks to be a good approach for this problem. Furthermore, using CNNs it would be possible to skip the QuPath stage and use the patches as raw input.