Lab Handout:

You will be provided H&E stained samples of blood from different animals (Bird, Frog, Fish)
H&E staining (Hematoxylin and eosin staining) is one of the principal tissue stains in histology. Hematoxylin stains the cell nucleus in dark purplish blue, while eosin stains the cell cytoplasm in pink.
Here you will take images of different blood samples, measure the nuclear areas and shape and compare these between the different animals statistically. You will then compile the data into a report.
Importantly, the aim of the practical is not to learn about the biology, but rather use this as an example for microscopy imaging, image processing and image analysis and in this sense, this could be easily replaced with other samples that contain different coloured or contrast areas.

Steps:

A- Imaging

1. To analyse the data properly you will first need to identify the pixel size. You will take images in 20x magnification. Similar to the previous semester you will start first with an image of the micrometer. The slides have a circular area in the middle and an etched ruler at its centre. The subdivisions are spaced at 10µm each. You can save and open this image later using Fiji (https://fiji.sc) to measure the pixel size. For calibration with ImageJ See Appendix 1.

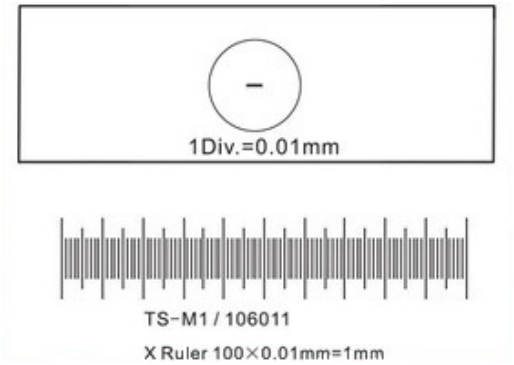


Figure 1: layout of stage micrometre

2. Take several images at 20x magnification from each sample:

    a. Bird blood
    b. Fish blood
    c. Frog blood

    make sure you change the file names (or make notes of which file corresponds to which sample!); every student should take one image for each sample! Make sure the illumination and focus are set up so that you can clearly distinguish the nucleus from the cytoplasm.

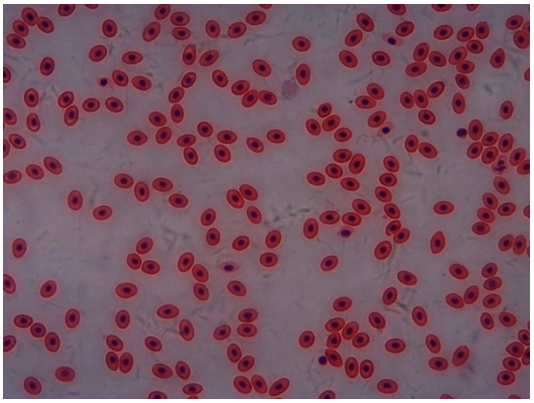


Figure 2: H&E staining of blood sample

3. Save images on your usb drives. Rename the files so that you can recapitulate the source, magnification and repeat and date. E.g. '220127_BirdBlood_20x_1.jpg'; This is good practice and it's best to get used to this early on. Avoid spaces in the name as this can cause issues with downstream processing.

B- Image processing and Image analysis (*code is displayed in italics* and *variables to be modified are displayed in red; make note of the final values you are using for each image; they will be put into the report*)

1. As mentioned in the lecture there are various common tools for image processing and analysis. Here you will use python, since you will have already become familiar with this language. Other options would include ImageJ/Fiji, Cell Profiler, Matlab; each with advantages and disadvantages.
2. An example code for python can be found in Appendix 2. This code makes use of various libraries, which you need to install first, especially
    - NumPy
    - Plotly
    - Matplotlib
    - Pandas
    - scikit-image
3. after calling the libraries and functions you will need to go through a sequence of steps:
    i. change the path to where you have the images saved
    ii. open the image
    iii. basic image processing – convert to greyscale, gaussian filtering

    img=rgb2gray(image)
    img=gaussian(img, sigma=1)

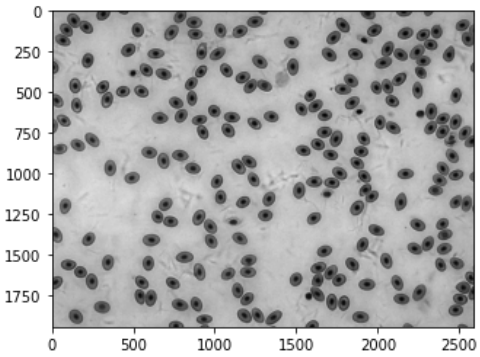


Figure 3: Cells after conversion to grey scale and gaussian filtering

iv. thresholding and binary functions

- Define threshold:
  threshold = threshold_yen(img, block_size)
  here you have different options for the thresholding algorithm. See here: https://scikit-image.org/docs/stable/api/skimage.filters.html; make sure you call the appropriate filters in the beginning); – the aim is here to only identify the nuclei, not the whole cells.

- mask the image with the calculated threshold; i.e. you remove everything above or below the threshold value; you will likely need to include a multiplication factor:
  *mask = img < threshold*factor*
  Whereby the factor will need to be optimized for each image
- Remove small objects (e.g. background staining, etc.) using
  mask = morphology.remove_small_objects(mask, minimumsize)
- fill holes in objects using:
  mask = morphology.remove_small_holes(mask, wholesize)

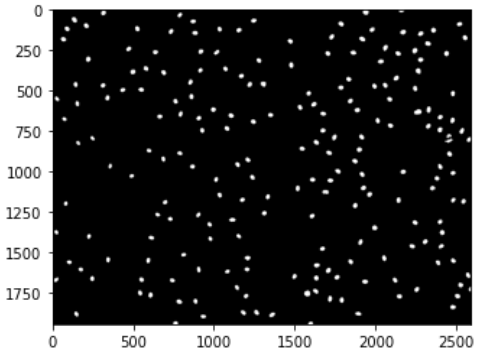


Figure 4: Thresholded image of nuclei

v. Measure objects area and shape data

```
labels = measure.label(mask)
props1 = regionprops_table(labels,
    properties=('area','perimeter','eccentricity','solidity'))
dataframe = pd.DataFrame(props1)
```

vi. convert array to csv and save file

```
dataframe.to_csv(imagename+'nuclear_shapedata.csv',
    encoding='utf-8', index=False)
```

vii. The thresholding algorithm and some of the factors need optimisation; one way to do this is to generate an interactive (html) image that has the data displayed when hovering over the particular objects; this will allow you to identify how many objects were successfully recovered, and what is the typical size of the nuclei or artifacts you want to filter out. This can be done with the following code (don't worry if you don't understand everything at this point):

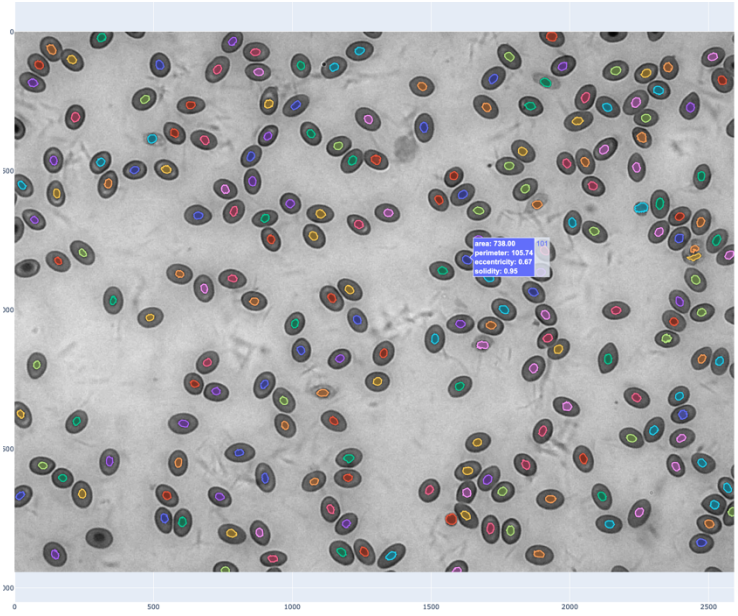


Figure 5: Screenshot of interactive window with displayed data for a particular object

```
fig = px.imshow(img, binary_string=True)
fig.update_traces(hoverinfo='skip') # hover is only for label info

props = measure.regionprops(labels, img)
properties = ['area','perimeter','eccentricity','solidity']

for index in range(1, labels.max()):
   label_i = props[index].label
   contour = measure.find_contours(labels == label_i, 0.5)[0]
   y, x = contour.T
   hoverinfo = ''
   for prop_name in properties:
      hoverinfo += f'<b>{prop_name}: {getattr(props[index], prop_name):.2f}</b><br>'
   fig.add_trace(go.Scatter(
      x=x, y=y, name=label_i,
      mode='lines', fill='toself', showlegend=False,
      hovertemplate=hoverinfo, hoveron='points+fills'))

#show interactive figure and save to html (in specified folder)
plotly.io.show(fig)
fig.write_html(imagename+"nuclei.html")
```

C- Data analysis

1. The output of the python program will be:
    - A html file that you can use to optimize the settings in an iterative process
    - A csv file for each image that you analysed, containing data for the area, perimeter, eccentricity, and solidity for each nucleus.
2. For analysis you will read the csv files, combine the data for each animal; **convert the area to µm^2** and **perimeter to µm** using the calibration values from the micrometre (appendix 1)

3. Prepare a **box and whisker plot** using the boxplot function from 'matplotlib.pyplot'; see appendix 3)

4. Calculate mean, median, standard deviation, and count(n-number) for each image and each parameter/species
5. statistical analysis (ANOVA) using the scipy.stats library and the f_oneway function and post-hoc test for multiple comparisons using the scikit_posthocs library and the posthoc_dunn function, using p_adjust = 'sidak' to adjust the p-values for multiple comparison. Please note that post-hoc tests are not easily performed using excel, so this is a clear advantage for the analysis using python. These are an important part of the statistical testing, as these will tell you which pairs of comparisons are significantly different or not. (see appendix 3)

D- **Assembly of the report**

To assemble the report include the following sections:

1. Introduction (a brief description of the experiment in no more than 150W)
2. Methods;
    a. list in a tabular form the file names and the final parameters you used in python for the thresholding algorithm and the factors
3. Results; include here
    a. the raw images that were used for the analysis
    b. the overlay images from the html files
    c. box plots for area, perimeter, eccentricity, and solidity
    6. a table with mean, median, standard deviation, and count(n-number) for each parameter/species, plus ANOVA and post-hoc testing results

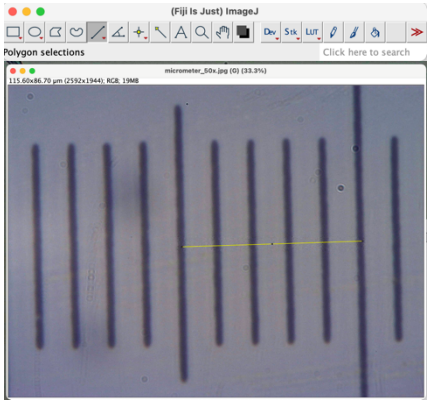
4. Conclusions (no more than 100W)

Marking Scheme:

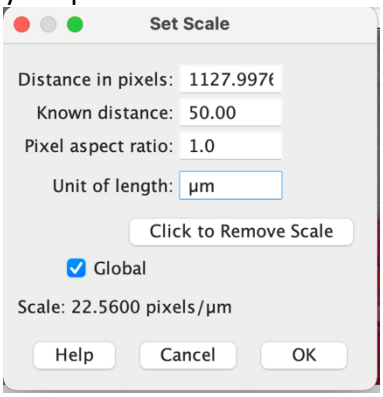
Introduction 25; Methods 25; Results 25; Conclusion 25;
for each 0-7 marks: No effort seen (e.g. only few nuclei detected; data improperly logged; graphs improperly plotted; etc.) unstructured, repetitive
8-17 marks: Introduction and Conclusions generally good, but too short or too long, or misses emphasis; Data in graph correct but incomplete format (units, decimals, etc);
18-25 marks: Effective in delivering the message, good effort in optimizing the image analysis; Good graph, decimals, units and excellent interpretation of the data in the conclusions.

Appendix 1 – Calibration with ImageJ

1. Install (if necessary; https://fiji.sc) and open Fiji
2. Drag and drop the image onto ImageJ (or open via File>open)
3. Use line function to draw a line between known number of subdivisions (on this example 5x10 = 50µm)



4. use tools > set scale to calibrate the images. Keep the distance in pixel (this will be populated from the line that you made); add the known distance (here 50) and unit of length (here µm). select global to keep this for all images you open. This will also tell you the scale in pixels / µm. take a note of this, as this will be used to convert your pixel measurements from python.



Appendix 2: Example code for Python

```
#import libraries
import os #for changing path, etc
import imageio #for reading image files
import matplotlib.pyplot as plt #for displaying figure/image data
import pandas as pd #for data handling/analysis
import plotly #for interactive plots
import plotly.express as px #for displaying data on overlay
import plotly.graph_objects as go #for intearctive plots
import scipy.stats as ss # for statistical testing
import scikit_posthocs as sp # for posthoc testing
#import functions
from skimage import measure, morphology
from skimage.color import rgb2gray
from skimage.filters import (gaussian, threshold_yen)
from skimage.measure import regionprops_table


#Define Path and change directory
path = '/Users/folder address'
os.chdir(path)
imagename = filename' #change here the file name
#read image - change image name here!
image = imageio.imread(imagename+'.jpg')
#image processing - convert to greyscale and perform gaussian filtering
img = rgb2gray(image)
img= gaussian(img, sigma=1)
#Show image
plt.imshow(img, cmap='gray')

# Binary image, post-process the binary mask and compute labels; change parameters here as needed
block_size = 51 # this is the size of the window for adaptive thresholding
threshold = threshold_yen(img, block_size) #change here the algorithm
mask = img < threshold*factor #change here the factor
#especially change here the parameters below for erosion and removing objects; comment out if not
needed
#mask = morphology.erosion(mask,square(1))
mask = morphology.remove_small_objects(mask, 400) #start with low value, then look at interactive
picture to see what is nuclear size, then iterate
mask = morphology.remove_small_holes(mask, 5000)
plt.imshow(mask, cmap='gray')


#Measure objects area and shape data and save to array props1
labels = measure.label(mask)
props1 = regionprops_table(labels, properties=('area','perimeter','eccentricity','solidity'))
dataframe = pd.DataFrame(props1)


#convert array to csv and save to "shapedata" in path specified above
dataframe.to_csv(imagename+'nuclear_shapedata.csv', encoding='utf-8', index=False)

#generate interactive image to display data on cells when hovering over the cells
fig = px.imshow(img, binary_string=True,  width=1400, height=1000)
```

```
fig.update_traces(hoverinfo='skip') # hover is only for label info

props = measure.regionprops(labels, img)
properties = ['area','perimeter','eccentricity','solidity']

for index in range(1, labels.max()):
   label_i = props[index].label
   contour = measure.find_contours(labels == label_i, 0.5)[0]
   y, x = contour.T
   hoverinfo = ''
   for prop_name in properties:
      hoverinfo += f'<b>{prop_name}: {getattr(props[index], prop_name):.2f}</b><br>'
   fig.add_trace(go.Scatter(
      x=x, y=y, name=label_i,
      mode='lines', fill='toself', showlegend=False,
      hovertemplate=hoverinfo, hoveron='points+fills'))

#show interactive figure and save to html (in specified folder)
plotly.io.show(fig)
fig.write_html(imagename+"nuclei.html")
```

Appendix 3: Boxplot and statistics
For boxplot:

If not already done above…(see appendix 2) import libraries:

- *import matplotlib.pyplot as plt* # for boxplot
- *import pandas as pd* # for data handling
- *import scipy.stats as ss #* for ANOVA
- *import scikit_posthocs as sp* #for post-hoc testing

- set path (as above)
- set filenames to the csv files, e.g.
  filename1 = 'HE_stain_Blood_Fish_20xnuclear_shapedata.csv' #change here the file name, etc.
- read files and convert to pandas data frames
  df1 = pd.read_csv(filename1) #etc for the other files
- Concatenate the data frames for each species (i.e. all fish, all bird, etc)
  allFish=pd.concat([df1,df2])
- Extract and combine the measurements for each into a list
  Areas=[allFish["area"],allFrog["area"],allBird["area"]]
  Eccentricities=… # repeat the same for perimeter, eccentricity, and solidity
- Multiply Areas and perimeters by pixel conversion factors
  A*reas_µm2 = [value * factor for value in Areas] #change "factor" to conversion factor; same for perimeter*
- Print out basic statistics:
  for df in Eccentricities
  print(df.mean(),df.median(),df.std(),df.count())
  # repeat the same for the other measurements
- Prepare boxplots

  plt.boxplot(Eccentricitys,
  labels= ['Fish', 'Frog', 'Bird'],
  showmeans= True)

- Perform ANOVA
  ss.stats.f_oneway(allFish ["area"],allFrog["area"],allBird["area"]) # etc for the other measurements
  #comment: write this into a variable and use print for output
- Perform Posthoc test
  sp.posthoc_dunn(Areas, p_adjust = 'sidak') #etc. for the other measurements
  #comment: see above