BIOMED Practical work:
BioImage Analysis using Icy

Carlos GRUSS (M2 IMA) & Niccolò ZOBOLI (Erasmus)

December 27, 2024

Contents

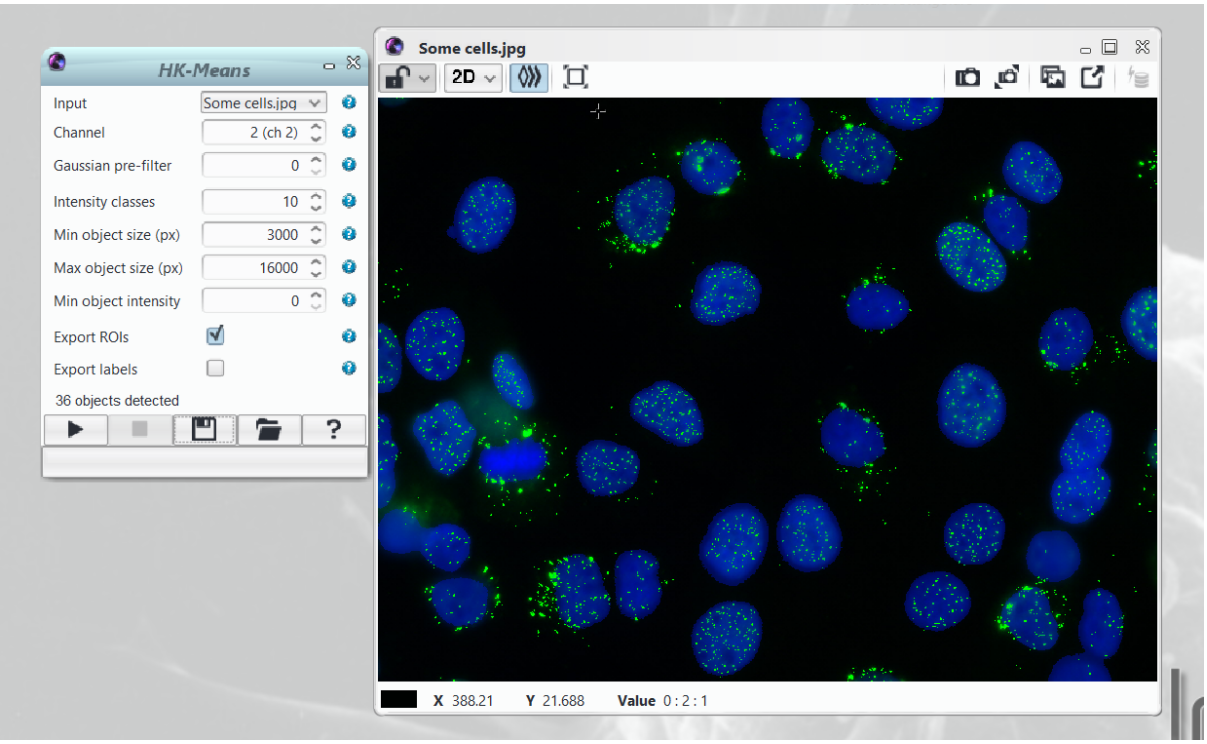
1 Detection and tracking of spots

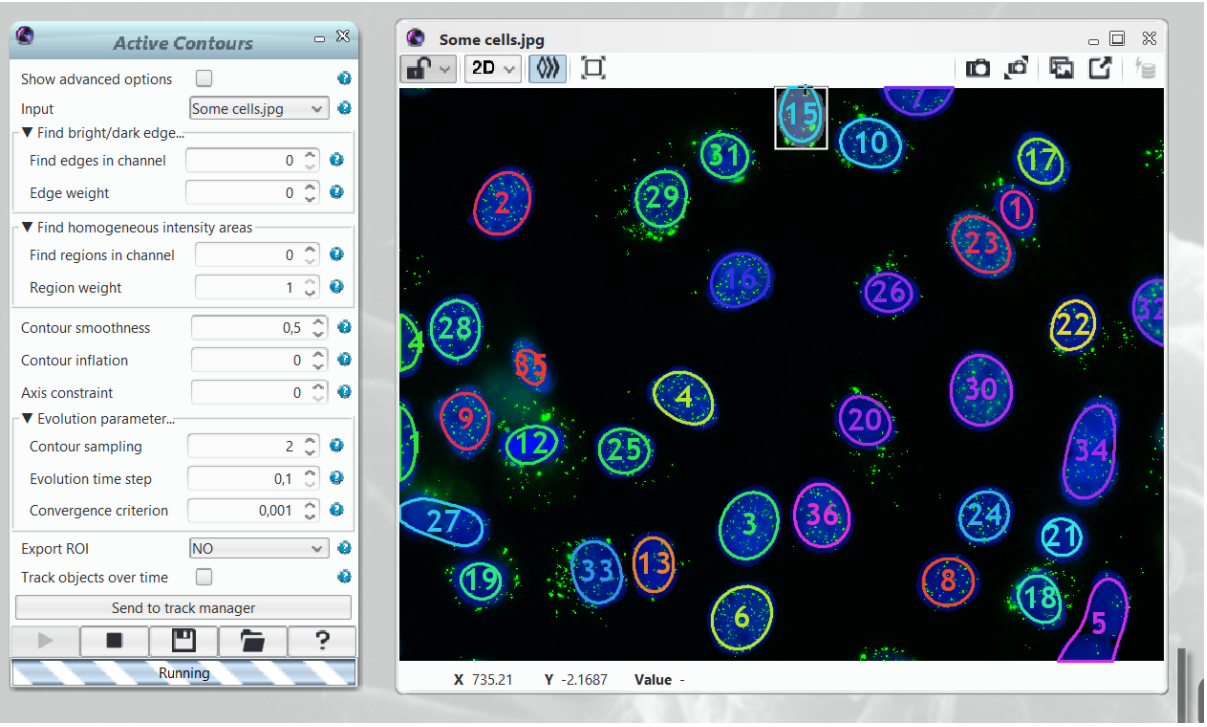
1.1 Compute the number of spots in each cell

Image: Some cells.jpg

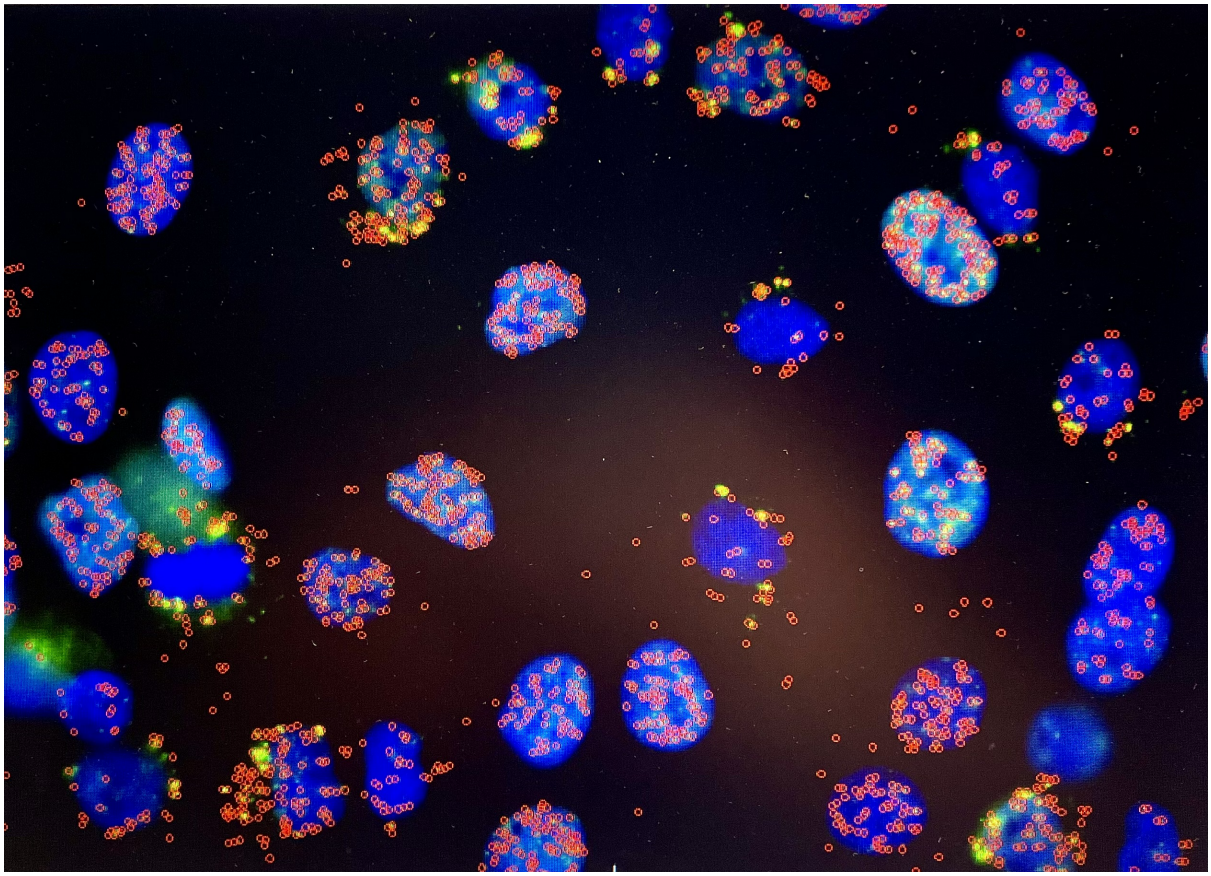
1. Detect the cells.
2. Detect the spots.
3. Create a protocol to automate the detection of spots and cells.

Detect the Cells – To identify the blue cells, the **HKmeans** algorithm was applied to the blue channel of the image (Channel 2). The minimum and maximum object sizes were set to **3000 px and 16000 px**, respectively. In this specific case, **36 cells** were detected, but some overlapping cell pairs were treated as a single cell by HKmeans. For conting in a better way the cells we decide to use the active contourn function and we discover 36 cells. However, for the purpose of this task, we proceed with the initial HKmeans result.

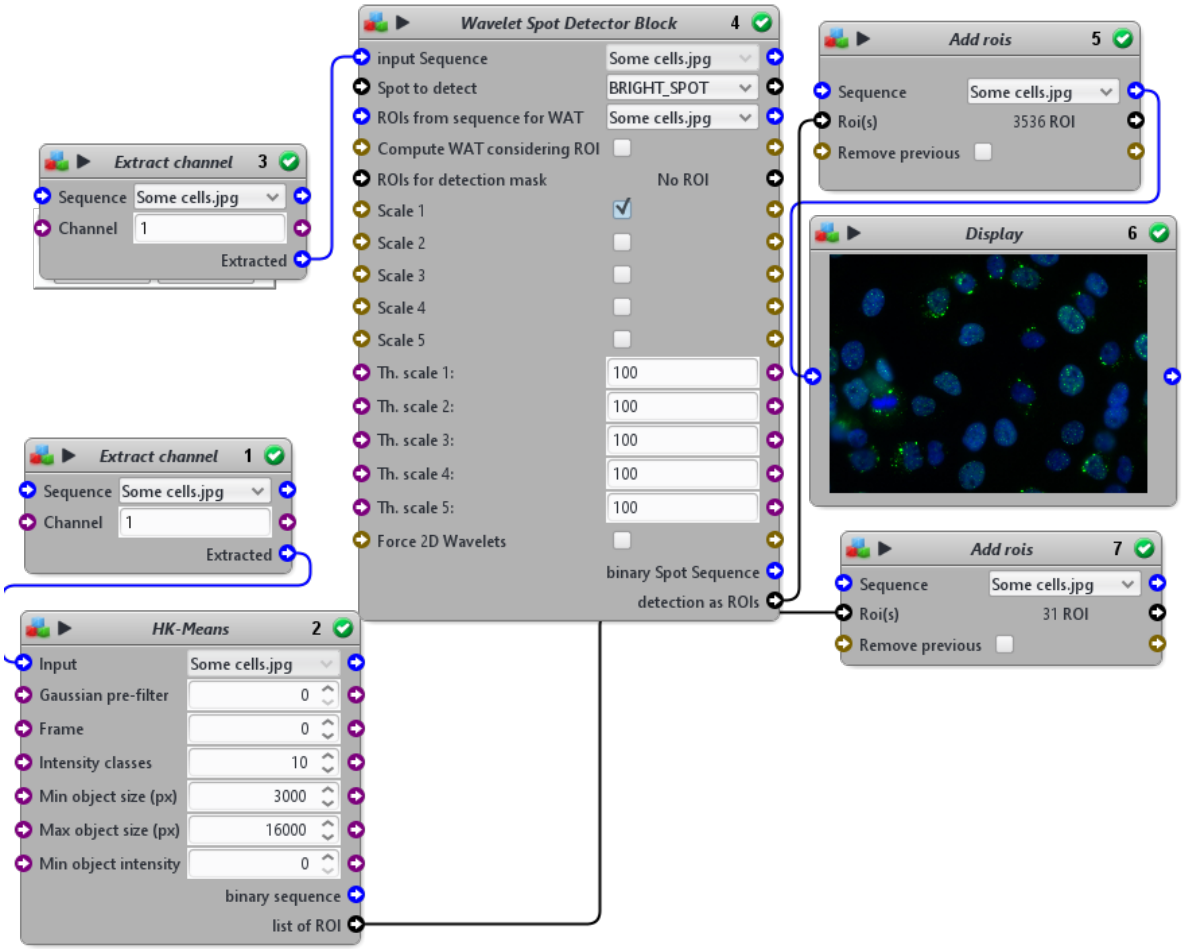




Detect the Spots – The **Spot Detector** tool was used to identify bright spots on a dark background within the green channel (Channel 1). With the appropriate configuration, **1436 spots** were detected across the cells. This method relies on selecting *"Detect bright spots over dark background"* to properly isolate spots of interest.



Protocol for Automated Detection – To streamline the process of detecting cells and spots, an automated protocol was developed by linking the HKmeans and Spot Detector blocks. This protocol simultaneously analyzes both the blue (cell detection) and green (spot detection) channels of the image with the parameters previously defined. The regions of interest (ROIs) are visualized on the same image, with cells displayed in blue and spots in green.



1.2 Detect and track spots

- Create a protocol to detect and track spots.
- Extract all possible information from these 2 sequences. Note that Tracking_FITC appears as a 3D data whereas it should be a temporal sequence. Therefore you should convert the sequence to a temporal sequence. For students that who have not a powerful machine, you should extract a smaller sequence.

Image: ParticleTracking02.tif – To track individual particle movements, we start by using Spot Detector to locate the particles in each frame. Spot Tracking is then used to track individual particles between frames and plot their trajectories.

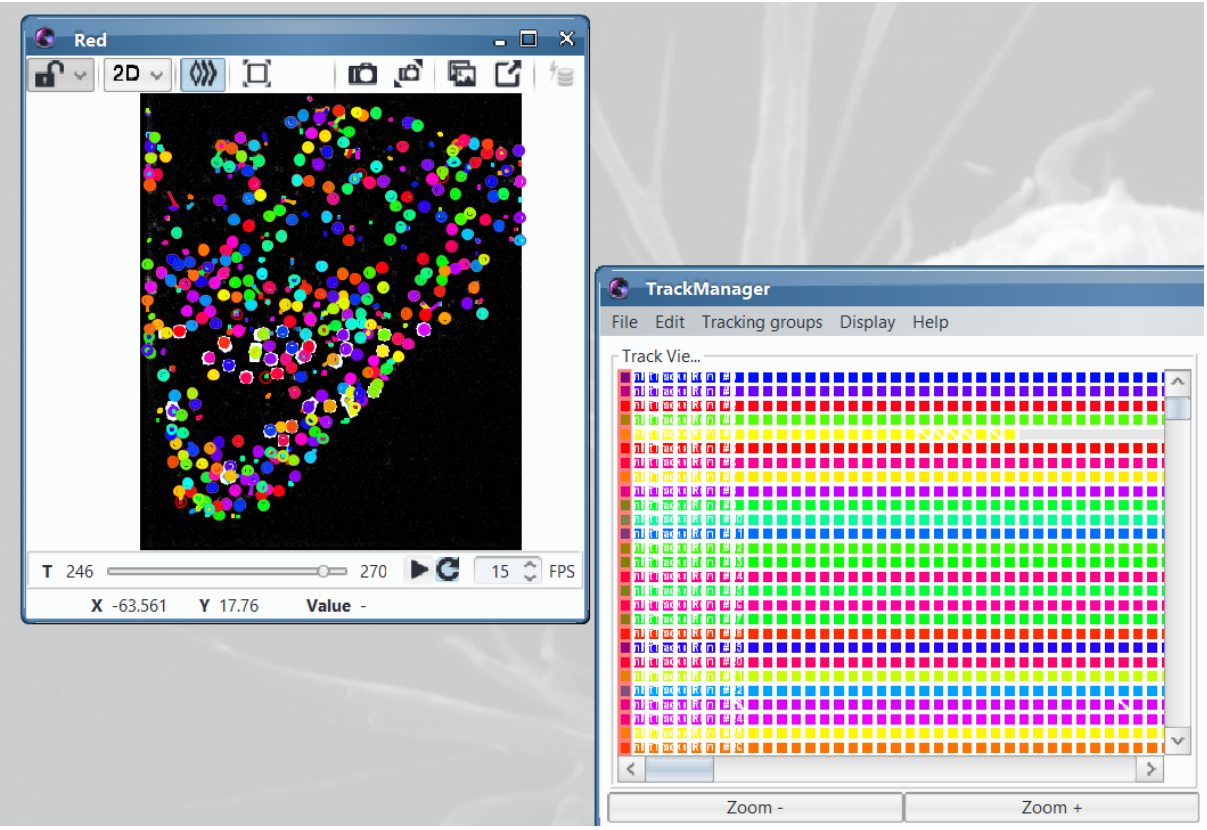
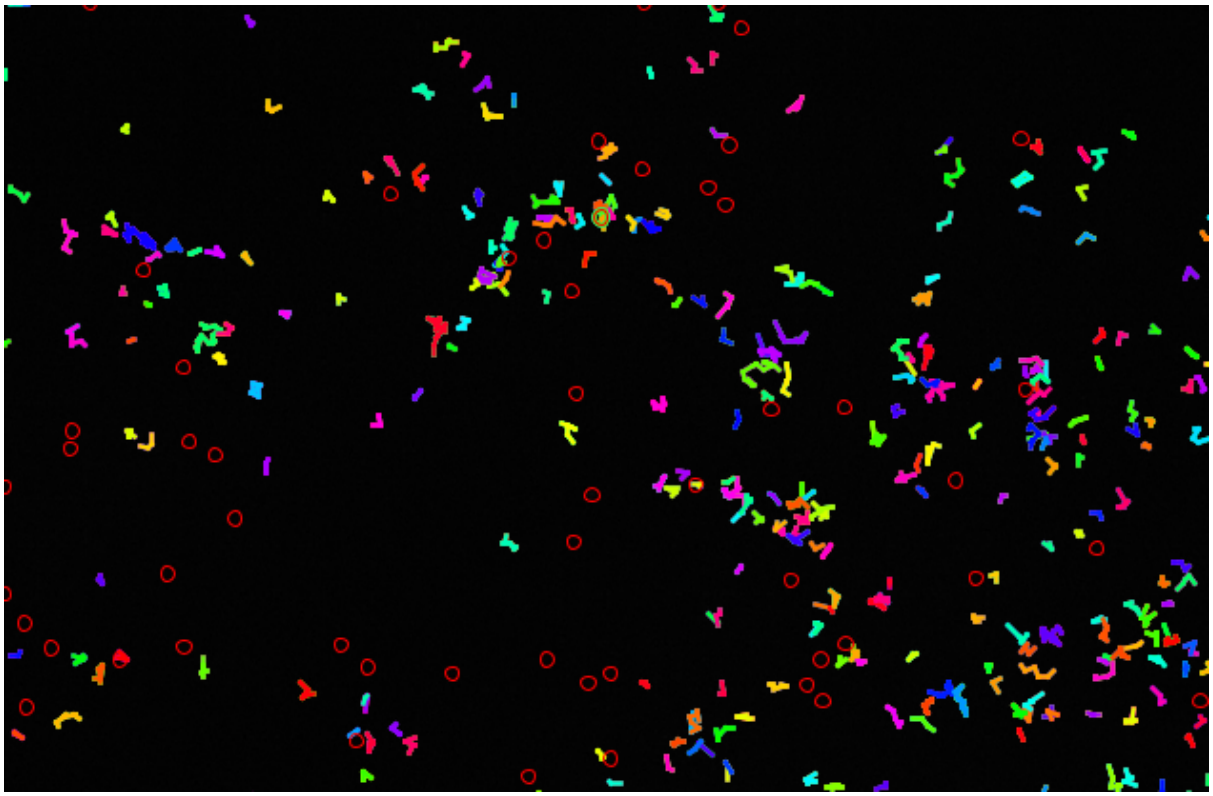


Image: Tracking_FITC.tif – The Tracking_FITC.tif file is a spatial sequence (3D). For the process to work properly, it is converted into a temporal sequence. We then start by applying Spot Detector and then Spot Tracking to obtain the results shown below. We also apply a filter to the duration of the trajectories. Only trajectories of more than 10 frames are taken into account (trajectories of less than 10 frames are considered to be noise) when calculating the average trajectory length.



Using the motion profiler we can also export the track information to analyze in Excel and we can estimate an average track length of $\approx 22\mu m$.

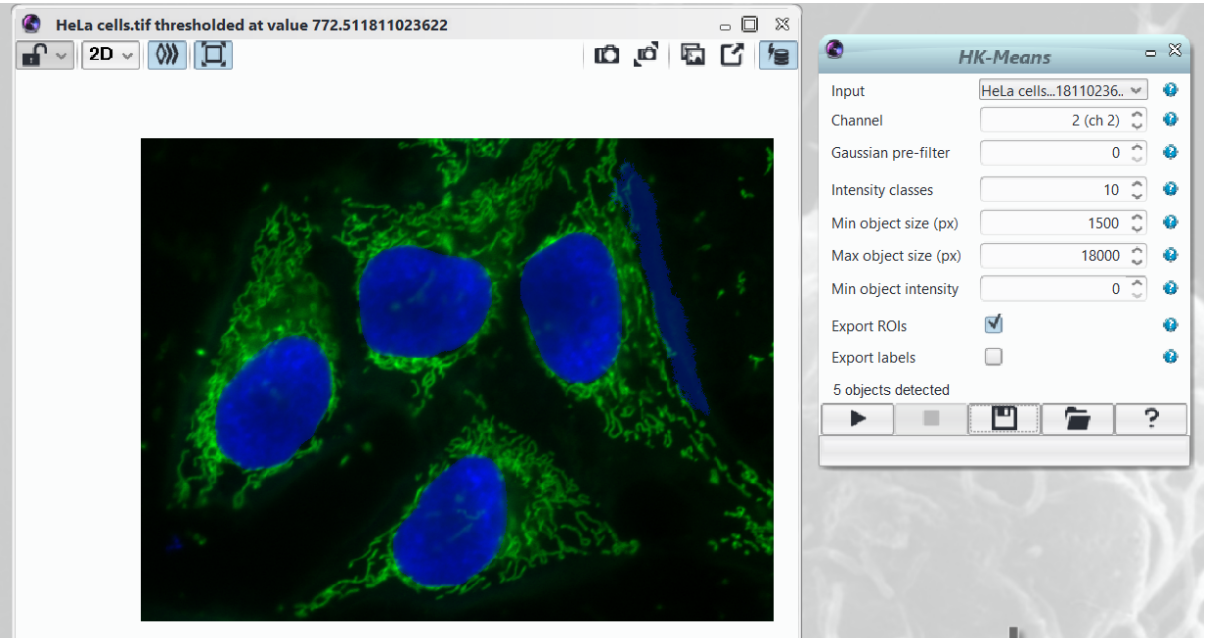
2 Segmentation and tracking

2.1 Segment and quantify spots in image

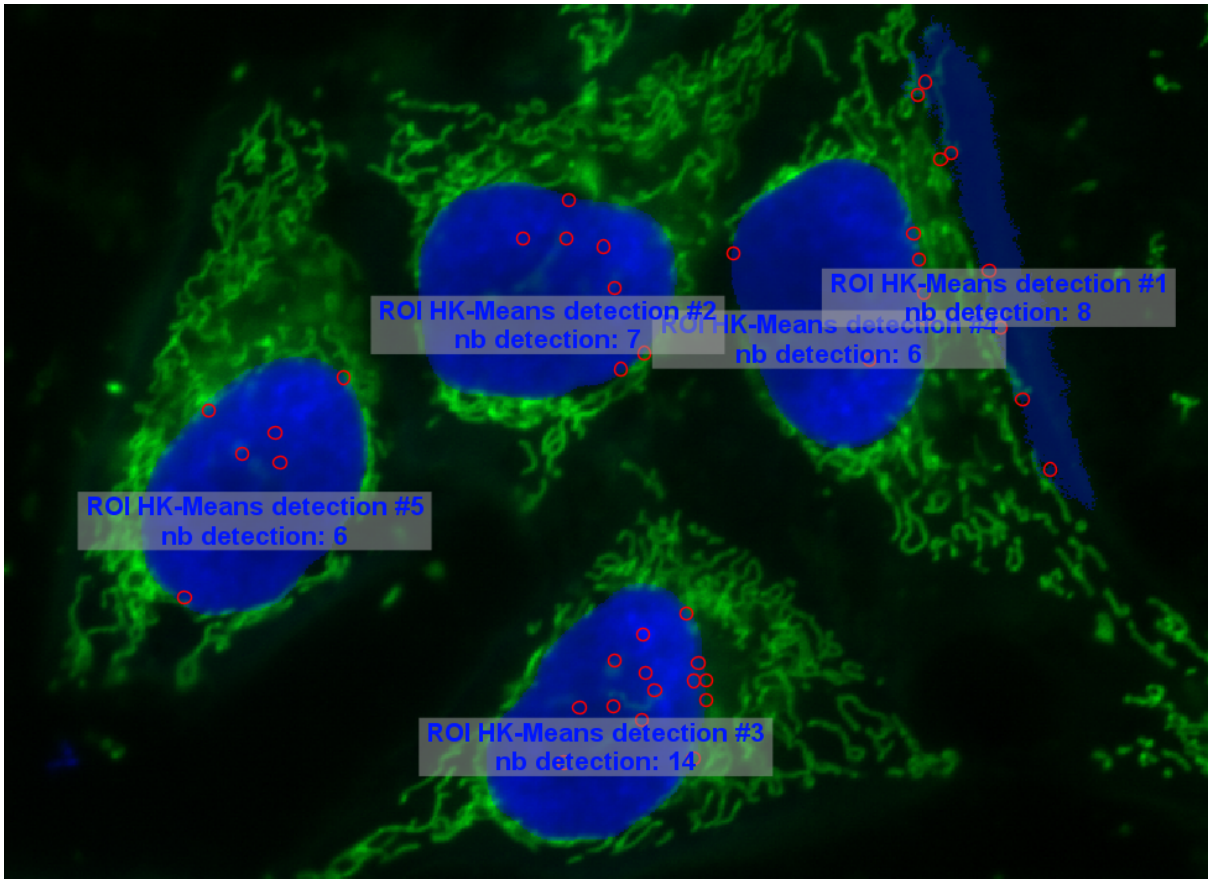
Image: HeLa_cells.tif

1. Segment the cells in this image.
2. Quantify spots inside each cell.

The cells are segmented using HKmeans on the blue channel (2) of the image with the same parameters as in the first part. The 5 cells are correctly segmented. We can see 4 rather round cells and an extended cell in the top right.



The Spot Detector is used to detect spots in the segmented cells. The ROIs corresponding to the cells (segmented areas) are given as input. As the spots have no apparent dominant colour, we chose to work on the average of the channels.



2.2 Segmentation and Tracking by CNN

Objective: Segment and track one of the sequences available at Cell Tracking Challenge. Proposed code:

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Paths for the dataset
TRAINING_DATA_PATH = '/Users/niccolozoboli/Desktop/DIC-C2DH-HeLa/01/'
TEST_DATA_PATH = '/Users/niccolozoboli/Desktop/DIC-C2DH-HeLa/02/'
OUTPUT_DIR = '/Users/niccolozoboli/Desktop/DIC-C2DH-HeLa/predictions/'

# Function to load images from a directory
def load_images_from_dir(directory, resize_shape=(256, 256)):
    images = []
    if not os.path.exists(directory):
        raise FileNotFoundError(f"Directory {directory} does not exist.")
    for file in sorted(os.listdir(directory)):
        file_path = os.path.join(directory, file)
        if os.path.isfile(file_path) and file.endswith('.tif'):
            img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
```

```
            if img is not None:
                img = cv2.resize(img, resize_shape)
                images.append(img)
            else:
                print(f"Warning: Unable to read {file_path}. Skipping.")
    return np.array(images)

# Load training and testing images
train_images = load_images_from_dir(TRAINING_DATA_PATH)
test_images = load_images_from_dir(TEST_DATA_PATH)

# Normalize the images
train_images = train_images / 255.0
test_images = test_images / 255.0

# Create masks for training (for testing purposes, masks are the same as the images)
train_masks = train_images

# Split training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(train_images, train_masks, \
test_size=0.2, random_state=42)

# Define the U-Net model
def unet_model(input_size=(256, 256, 1)):
    inputs = Input(input_size)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)
    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(p3)
    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
    u5 = UpSampling2D((2, 2))(c4)
    u5 = concatenate([u5, c3])
    c5 = Conv2D(256, (3, 3), activation='relu', padding='same')(u5)
    c5 = Conv2D(256, (3, 3), activation='relu', padding='same')(c5)
    u6 = UpSampling2D((2, 2))(c5)
    u6 = concatenate([u6, c2])
    c6 = Conv2D(128, (3, 3), activation='relu', padding='same')(u6)
    c6 = Conv2D(128, (3, 3), activation='relu', padding='same')(c6)
    u7 = UpSampling2D((2, 2))(c6)
    u7 = concatenate([u7, c1])
    c7 = Conv2D(64, (3, 3), activation='relu', padding='same')(u7)
    c7 = Conv2D(64, (3, 3), activation='relu', padding='same')(c7)
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c7)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', \
    metrics=['accuracy'])
    return model

# Create and train the U-Net model
model = unet_model()

# Train the model and save the training history
```

```
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), \
epochs=10, batch_size=1)

# Predict on test images
predictions = model.predict(test_images)

# Save predictions to the output directory
os.makedirs(OUTPUT_DIR, exist_ok=True)
for i, pred in enumerate(predictions):
    pred_img = (pred.squeeze() * 255).astype(np.uint8)
    cv2.imwrite(os.path.join(OUTPUT_DIR, f'prediction_{i}.png'), pred_img)

print(f"Predictions saved in {OUTPUT_DIR}")

# Plot training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```