Department of Mathematics

Chair of Mathematical Modeling of Biological Systems

Technische Universität München

Master's Thesis in Bioinformatics

Single-cell analysis of cancer drug response using computer vision and learning algorithms on time-lapse microtrench data

Pandu Raharja

Department of Mathematics

Chair of Mathematical Modeling of Biological Systems

Technische Universität München

Master's Thesis in Bioinformatics

Single-cell analysis of cancer drug response using computer vision and learning algorithms on time-lapse microtrench data

Wirkungsanalyse von Krebsmedikamenten in Einzeller Auflösung durch die Anwendung von Computer-Vision- und Machine-Learning-Algorithmen auf Microtrench-Videoaufnahme

Author:	Pandu Raharja
Supervisor:	Prof. Dr. Fabian Theis, Dr. Carsten Marr
Advisor:	Prof. Dr. Fabian Theis
	Prof. Dr. Dmitrij Frishman
Submitted:	15.10.2017

Abstract

Quantitative measurement of cancer drug response is esential to objectively gauge the efficacy of cancer drugs. So far, there has been no method to track and quantitatively measure single-cell response of of cancer drug treatment. A novel pipeline is presented in this thesis. First, a quasi-high-throughput method to track cells and quantitatively analyze single-cell response to drugs. We investigate the response of model cancer cell lineagues, MOLM and Jurkat, to known anti-cancer drugs Vincristine and Doxorubicine. While the method enabled relatively easy and quasi-high-throughput analysis of cancer treatment *in vitro*, our pipeline could also be adapted in varios contexts involving single-cell analysis with reasonable amount of modifications necessary.

Chapter 1

Introduction

Lorem ipsum

Chapter 2

Background

Lorem ipsum dolor si amet

Chapter 3

Methods

3.1 Laplacian of Gaussian (LoG) Cell Recognition

3.2 Image Encoding

Consider whether image encoding shoud contain Lena's picture instead.

3.3 Shift Correction

Consider following picture:

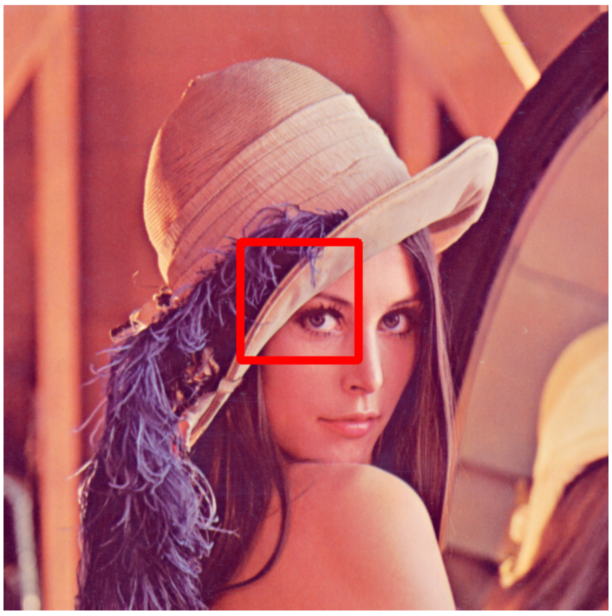


Figure 3.1: Lenna

There are many encodings that could be used to internally store this picture. Many such encodings utilized this so-called Red-Green-Blue encodings (RGB). RGB encoding represents the color of a pixel as a combination of red, green and blue color. This encoding is known to various spectra of human visible color and useful enough for most use cases (citation). To give representation on how the encoding works, the RGB encoding of some part of above picture is shown in Figure 3.2. For an image of size $m \times n$ pixels, the RGB encoding is thus a 4-dimensional matrix of dimension $m \times n \times 3$. For time-lapsed images accordingly, the RGB encoding of the video of length T is a 5-dimensional matrix of shape $t \times m \times n \times 3$.

Now, consider a case in which an image shifts. No rotation of camera is assumed, hence there are only two degree of freedoms (vertical and horizontal). Thus, a shift could be defined as a vector movement $\vec{v}$ of all points $x_{i,j} \in M_{t_i}$ in the time-lapse from time t_i to t_{i+1}. Given two degrees of freedom and discreteness of the problem due to pixel representation, the task is reduced to finding difference in x- and y-axis (δ_x and δ_y), so that the difference of transormed pixels at t_i and the pixels at t_{i+1}, i.e.:

$$argmin_{\delta_x,\delta_y}\{d(M_x, M_y)\}$$

For distance function d, all-channels absolute difference function is used, which is defined as:

$$d(M, N) = \sum_{c \in \{R,B,G\}} \sum_x \sum_y \|M_{c,x,y} - N_{c,x,y}\|$$

Since some pixels are lost from the field of view during the view, only a subset of both pictures are used to determine the distance, preferably those around the center point. The search for (δ_x, δ_y) pair is then implemented as grid search along x- and y-axis. An example of the search grid is shown in Figure 3.3. In the example, the point that returns the minimum distance was marked with thick black dot and is returned after every grid-search call as inferred shift.

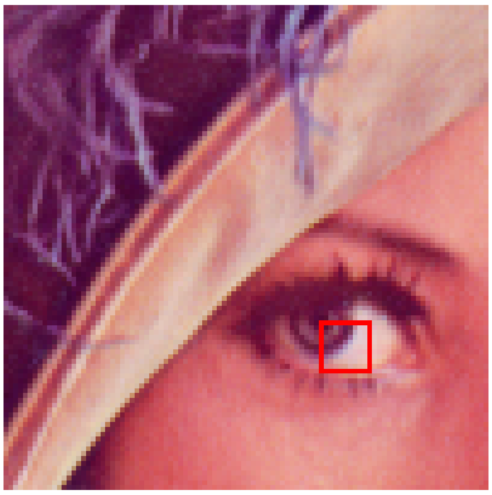
Since the time-lapsed data consists mainly of grayscale image, the RGB encoding could be the directly transformed to grayscale encoding. Using the transformed method also speeds up the calculation process since the distance function only computes the difference of grayscale channel's values:

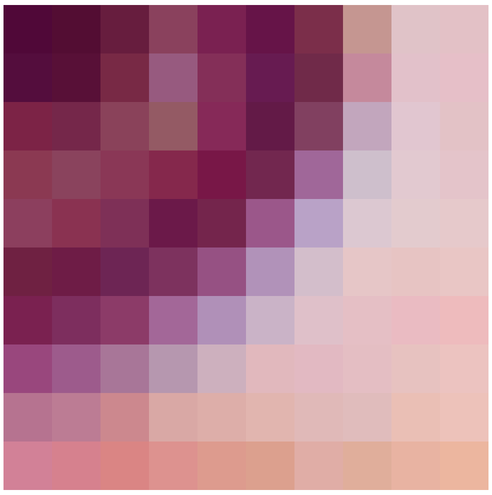
$$d(M, N) = \sum_x \sum_y \|M^{gray}_{c,x,y} - N^{gray}_{x,y}\|$$

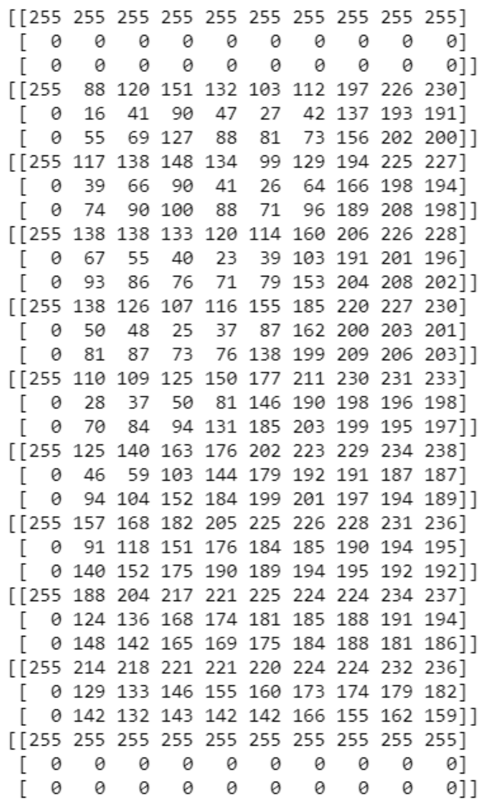
Due to lost pixels around the margin of before and after pictures, only the overlapping part of both slides are included after the correction. Thus, for an inferred shift of (δ_x, δ_y), the new dimension of the pictures is then $(m - \delta_x) \times (n - \delta_y)$. This change would then propagation to the other time-lapse images to maintain consistency of the images.

Ideally, the frame correction should be done for each position to reduce the track dropout rate caused by image shifts. This is however computationally very expensive and, as could be seen in Figure 3.4, not really necessary since the biggest shift indeed only happens right before and after the treatment, as it was expected during the experiment setting.

The algorithm for shift inference could be seen in Appendix A.


(a) 1a


(b) 1b



```
[[255 255 255 255 255 255 255 255 255 255]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]]
[[255  88 120 151 132 103 112 197 226 230]
 [  0  16  41  90  47  27  42 137 193 191]
 [  0  55  69 127  88  81  73 156 202 200]]
[[255 117 138 148 134  99 129 194 225 227]
 [  0  39  66  90  41  26  64 166 198 194]
 [  0  74  90 100  88  71  96 189 208 198]]
[[255 138 138 133 120 114 160 206 226 228]
 [  0  67  55  40  23  39 103 191 201 196]
 [  0  93  86  76  71  79 153 204 208 202]]
[[255 138 126 107 116 155 185 220 227 230]
 [  0  50  48  25  37  87 162 200 203 201]
 [  0  81  87  73  76 138 199 209 206 203]]
[[255 110 109 125 150 177 211 230 231 233]
 [  0  28  37  50  81 146 190 198 196 198]
 [  0  70  84  94 131 185 203 199 195 197]]
[[255 125 140 163 176 202 223 229 234 238]
 [  0  46  59 103 144 179 192 191 187 187]
 [  0  94 104 152 184 199 201 197 194 189]]
[[255 157 168 182 205 225 226 228 231 236]
 [  0  91 118 151 176 184 185 190 194 195]
 [  0 140 152 175 190 189 194 195 192 192]]
[[255 188 204 217 221 225 224 224 234 237]
 [  0 124 136 168 174 181 185 188 191 194]
 [  0 148 142 165 169 175 184 188 181 186]]
[[255 214 218 221 221 220 224 224 232 236]
 [  0 129 133 146 155 160 173 174 179 182]
 [  0 142 132 143 142 142 166 155 162 159]]
[[255 255 255 255 255 255 255 255 255 255]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]]
```

(c) 1b

Figure 3.2: Figure 3.2a shows the content of red marked region in Figure **??**. Figure3.2b shows the zoomed part around Lena's right eye and matrix represented in Figure 3.2c shows the RGB representation of the eye.

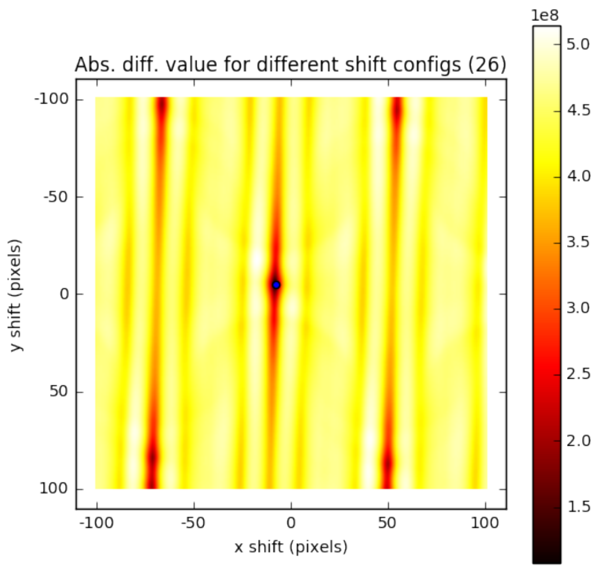


Figure 3.3: Search grid shift for Position 26. The search was conducted for shift between the last time point before and the first time point after the drugs treatment. The minimum is marked with thick black dot. The shift was inferred to be 8 pixels upwards and 5 pixels leftwards. Notice the repeating pattern of relatively favorable configurations after approximately 50 horizontal and 100 vertical pixels caused by lattice nature of the slits.

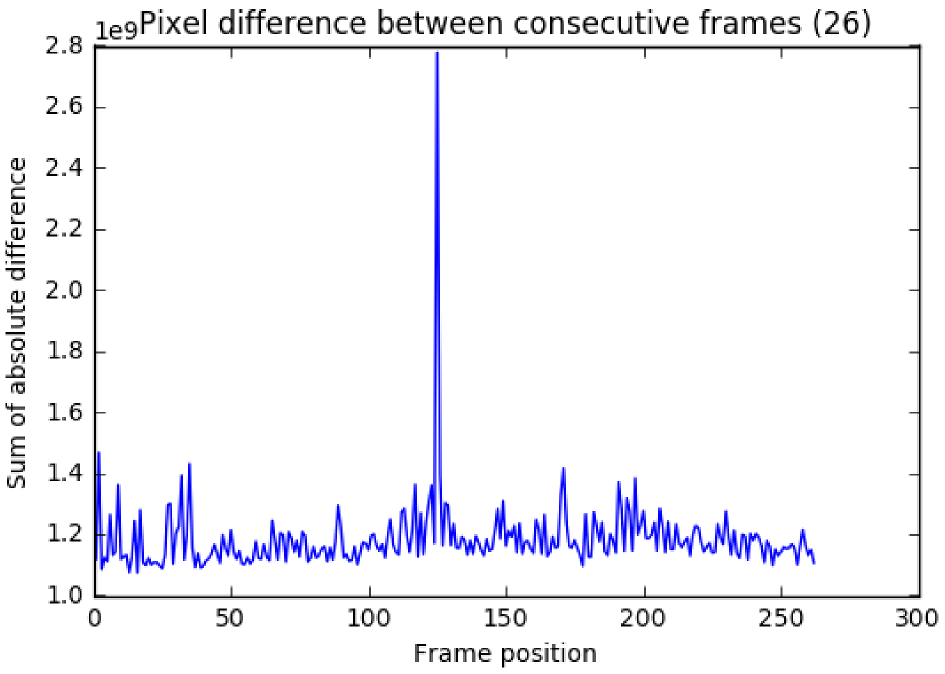


Figure 3.4: Sum of absolute differences of consecutive stacks for Position 26.

Chapter 4

Pipeline

Lorem ipsum

Chapter 5

Quantitative Analysis

Lorem ipsum

Chapter 6

Summary and Outlook

Lorem ipsum

Appendices

Appendix A

Algorithms

Shift Inference

```
def infer_shift(last_slide, first_slide, search_space=(200, 200)):

  if (search_space[0] % 2 != 0) or (search_space[1] % 2 != 0):
    print("Search spaces have to be even!")
    return None
  else:

    ## calculate absolute difference for various shifts
    x1 = search_space[0]
    x2 = search_space[0]
    y1 = search_space[1]
    y2 = search_space[1]
    mid = f1.shape[0] // 2, f1.shape[1] // 2

    ## results storage
    absdiffs = np.zeros((search_space[0] + 1, search_space[1] + 1))

    ## last slide before treatment
    f1sub = f1[(mid[0] - x1):(mid[0] + x2), (mid[1] - y1):(mid[1] + y2)]

    ## search space
    xdiff1 = -int(search_space[0] / 2)
    xdiff2 = int(search_space[0] / 2) + 1
    ydiff1 = -int(search_space[1] / 2)
    ydiff2 = int(search_space[1] / 2) + 1

    for xdiff in range(xdiff1, xdiff2):
      for ydiff in range(ydiff1, ydiff2):

        ## calculate absolute difference for shift
        f2sub = f2[(mid[0] - x1 + xdiff):(mid[0] + x2 + xdiff),
                   (mid[1] - y1 + ydiff):(mid[1] + y2 + ydiff)]
    x = xdiff + int(search_space[0] / 2)
    y = ydiff + int(search_space[1] / 2)
    absdiff_xy = np.sum(cv2.absdiff(f1sub, f2sub).ravel())
    absdiffs[x][y] = absdiff_xy


## calculate shift based on calibration data
x = np.argmin(absdiffs) // absdiffs.shape[0]
y = np.argmin(absdiffs) % absdiffs.shape[0]

"""
True shift is the opposite of coordinate encoded
in absdiff

Let X2 the second picture and X1 the first picture.
If the sub-picture of first slide of X2 centered
at (c1 + s1, c2 + s2) fits the most with the sub-picture
of the last slide of X1 centered at (c1, c2)
then the pictures shift by (-s1, -s2) upon treatment
"""
diff = -(x - search_space[0] / 2), -(y - search_space[1] / 2)

return diff
```