PC4199: HONORS PROJECT IN PHYSICS

Colour based Motif in Butterfly Scales to characterise genetic variations.

By : YEO Zhen Yuan
Matric No. : A0108535H

Supervisor : Asst/P Duane LOH*
*National University of Singapore, Department of Physics
Date : April 3, 2017

Abstract

The evolution of colours in living things have long been a huge area of interest in fields like evolution biology. In particular, structural colours may have important applications in material science. However, the formation of such thin and fine structures was not well understood.

The method of analysing easily obtainable optical microscope images was explored in this report. A Bayesian approach to segment a Bicyclus butterfly scale was used and have shown to be very successful. With that image segmentation, the motifs, or patterned spatial distribution, of colours could be studied with a soft clustering algorithm and proven to have some persistent features.

This series of development may lead to a new and practical way of characterising the diverse colours of different butterflies.

Acknowledgements

The author would like to dedicate this section to appreciate his supervisor Duane. Duane have spent many fruitful hours with the author to explore the new territories in this field for this project. Despite many setbacks, bugs and pivots, he managed to guide and motivate the author to, finally, complete the year long project.

Special thanks to Antónia Monteiro for the collaboration as well as Anupama Prakash for the beautiful butterfly images.

Contents

1 Introduction

1.1 Overview

Colours in living things, for a very long time, have been a topic of interest to biologists and philosophers. In the last few centuries, scientists worked hard to identify the mechanisms to produce, as well as understand the purpose of, those colours [1, 2, 3]. While most of the scientific studies were focused on the evolutionary aspects, some of them can have huge implications in materials science [2].

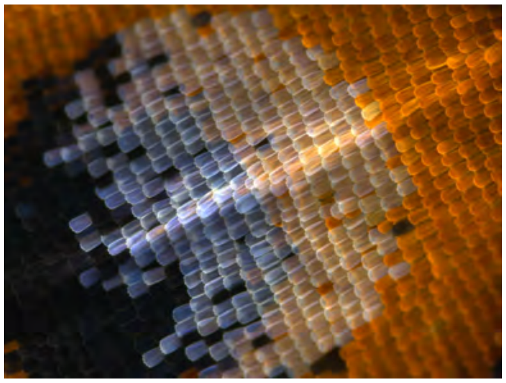


Figure 1: A section of a butterfly wing. Image taken by Anupama Prakash.

In general, colours in living things can be created by selective absorption of light by pigments (pigment colour), by interference scattering of light from highly structured tissues (structural colour) and by a combination of these elements.

1.2 Butterfly scale

The brilliant blue colours of a butterfly scale was thought to be a result from inference effects due to the fine structures on the scale, in contrast to the absorptive colours (brown) resulting from pigmentation. In particular, the blue colours on the Bicyclus butterflies was found out to be produced by thin film interference at the base as illustrated in Figure 2 [4] .

These fine structures were believed to be created from a single cell

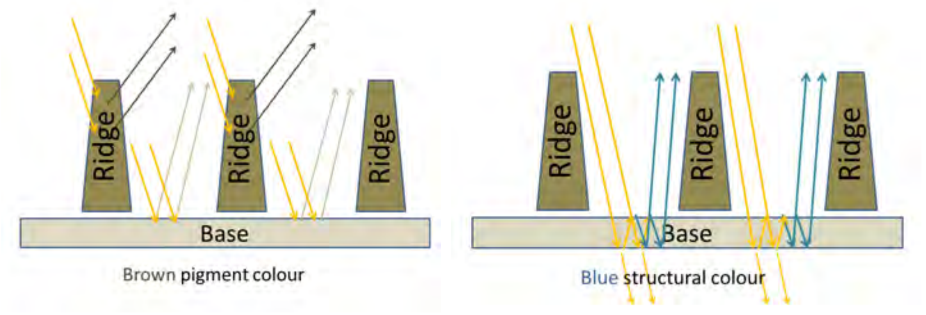


Figure 2: Model of a butterfly scale showing the distribution of (LEFT) pigment colour and (RIGHT) structural colour. For a Bicyclus butterfly, the structural colour is mainly produced at the base.

when a caterpillar metamorphoses into a butterfly [5]. Within a butterfly, the genetic information codes for the variations in the proteins and enzymes which ultimately result in the structural and pigment colours. Furthermore, between neighbouring cells, the chemical signals were sufficient to trigger the cells to differentiate and to look different from each other, for example, during the formation of eye spots, neighbouring scales communicate each other to form the different coloured rings [6].

Within a species, the gender differentiation can determine the differences in these structures and thus result in the different dominant colour for each gender. Equipped with a better understanding of these structures for the different gender within a species, one can possibly obtain better insights on how gender differentiation relates to the structural colours.

1.3 The Problem

In order to study the structures which produce such structural colours, a Transmission Electron Microscope (TEM) will be needed to resolve the fea-

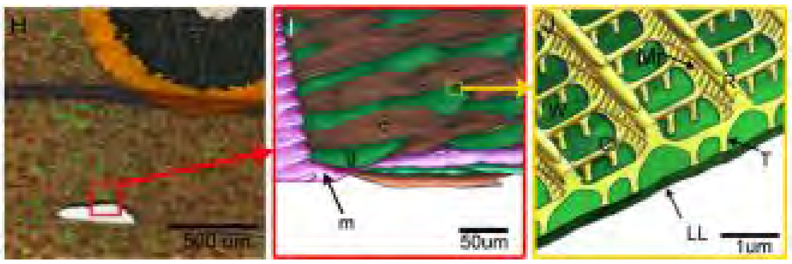


Figure 3: The close up image of a typical butterfly cover scale shows the intricate features at about 1 micron scale [7]. Such fine features can only be resolved by an equipment with a high resolution.

tures on a butterfly scale. However, in using this technique, at best, one can only observe the 3D reconstruction, without any colour information. With TEM, it is not possible to recover the colour of the structural colour, which are the colours observable to other butterflies.

The other issue of studying structural colours using TEM is that it is expensive and slow. This makes studying the changes in the structures, from genetic variations, a very costly and tedious task. At a reasonable resolution, it would cost a researcher an order of about a thousand dollar (SGD) to study a small section of the butterfly wing.

The next best method for studying 3D structures is to use 3D X-ray ptychography with the capacity to image 3D structures at a high resolution of 16nm [8, 9]. However, similar to TEM, any information of the colour is lost. Thus, studying the variations in the formation of structural colours may require other techniques.

1.4 The Solution

A novel approach to this problem is to apply image processing techniques as well as statistical clustering techniques on high-resolution optical images of butterfly scales. Optical microscopes are commonly found in most imaging laboratories and obtaining a high-resolution optical image is not prohibitively expensive. Furthermore, optical microscope images retain the colour information which may be useful to study their chemical origins.

However, the typical laboratory-grade optical microscope have a resolution of about 1 μm and is unable to resolve the small features which leads to the structural colour. Instead, it is able to capture the regions which produce such colours as shown in Figure 4.

Motifs refer to a pattern and for this report, the pattern or spatial distribution of colours was investigated. Using this method, characterising the diverse colours of butterflies may become a relatively cheap and fast process and this may result in better insights on the different and diverse mechanisms for such colours. The techniques used were described in the following sections.

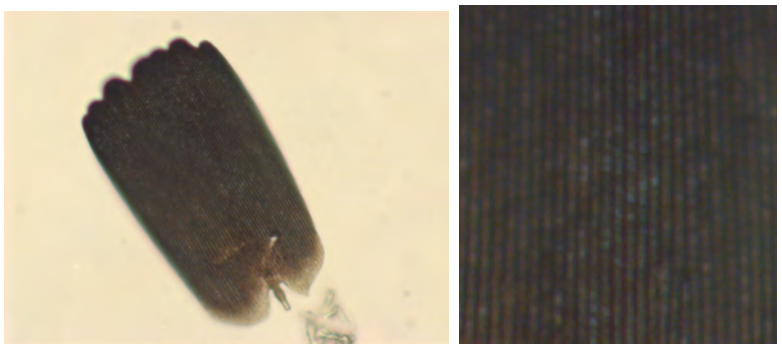


Figure 4: Cover scale Female pansy butterfly (left). Close up of the cover scale (right). Image taken by Anupama Prakash.

2 Methodology

Image processing techniques as well as clustering techniques was applied on an optical image of a butterfly scale to quantitatively study the structural features on them. For the optical image to be of quantitative use, the image was first segmented, then restructured and finally fed into a clustering algorithm. Section 2.1 explains the process of image segmentation while Section 2.2 describes the process of restructuring and re-labelling the image so that quantification is possible. Section 3.2 is focused on using the restructured image to obtain quantifiable information which could be used for further analysis.

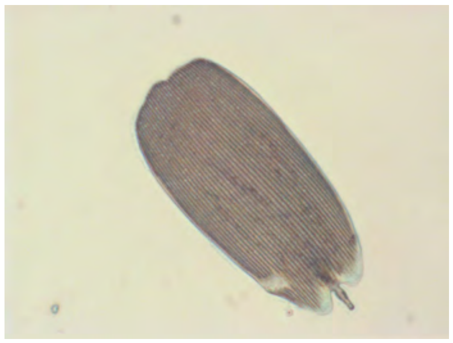


Figure 5: Optical microscope image of a Bicyclus butterfly scale.

2.1 Image Segmentation

2.1.1 RGB-space clustering

With a colour image of the butterfly scale, the RGB values of each pixel was obtained and plotted as points on a 3 dimensional scatter graph where each axis represents the individual colour channels as shown in Figure 6 (LEFT).

Observing the number of "groups" or clusters, one can identify how many major colours are present on the image. The number of points in a cluster will indicate how often those colours appear. The spread of the cluster can tell us how much do the colours deviate from its centre and finally, the spread between clusters can tell us about the “inter-mixing” between two major colour groups.

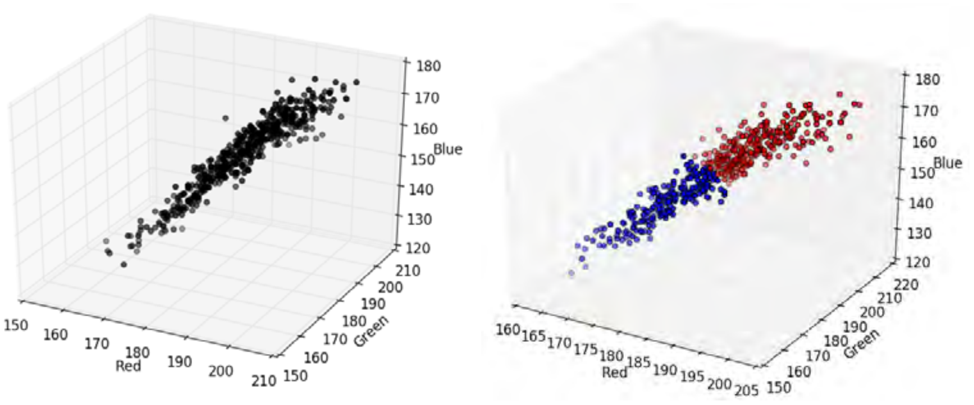


Figure 6: (LEFT) Scatter plot of the butterfly scale shows the distribution of RGB values for each pixel. Instead of distinct colors, a spread or smearing of colours can be observed. (RIGHT) The Gaussian mixture model clustering of the RGB plot. The red points were classified as the valleys and the green points were classified as the ridges.

2.1.2 Gaussian mixture model

A Gaussian mixture model was used to classify the points into two clusters, one for points from the ridges and another for points from the valleys. The idea used here is that the ridges and the valleys of the scale would have distinctively different RGB values. For the case of this butterfly scale, the blue values would be more intense for the valleys since they were mainly responsible for the blue colour as shown in Figure 6 (RIGHT). With the scatter points in the RGB-space, the Gaussian mixture model will be able to determine, for each pixel, the probability of it being in each cluster.

With this clustering, it is now possible to determine the probability of a point, of a particular RGB value, to be classified as either a ridge or a valley. Mapping the probabilities back to the original scale, it is observed that the clustering does indeed correspond to the actual ridges and valleys, as shown in Figure 7. It is with this reorganisation of the image that one can possibly quantify the image.

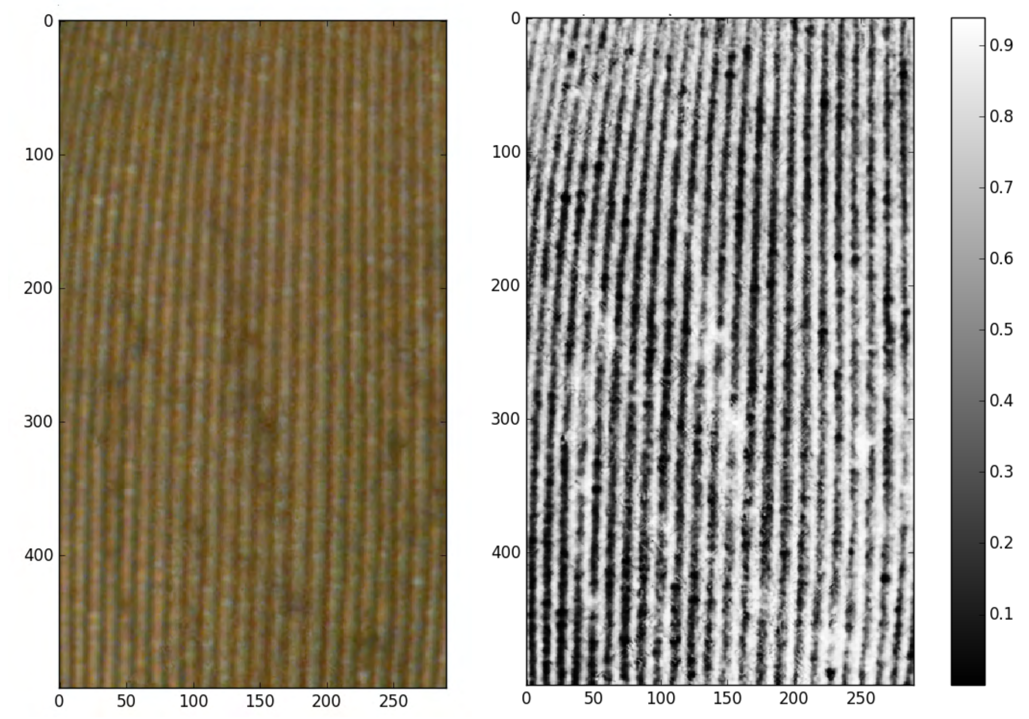


Figure 7: (LEFT) Close up of a male cover scale. (RIGHT) The probability map of a scale, where a value closer to 1 represents a high likelihood of being in the ridge and a value closer to 0 represents a high likelihood of being in the valley.

2.1.3 Gabor filter

With the probability map, a gabor filter was used to further modify the map so that the image can be properly segmented. A gabor filter is a filter commonly used for edge detection in computer vision applications. In this case, it was utilised to segment the ridges from the valleys in the probability map. It serves the purpose of better refinement of the classification between the ridges and valleys since there were regions where it was incorrectly classified by the Gaussian mixture model classification.

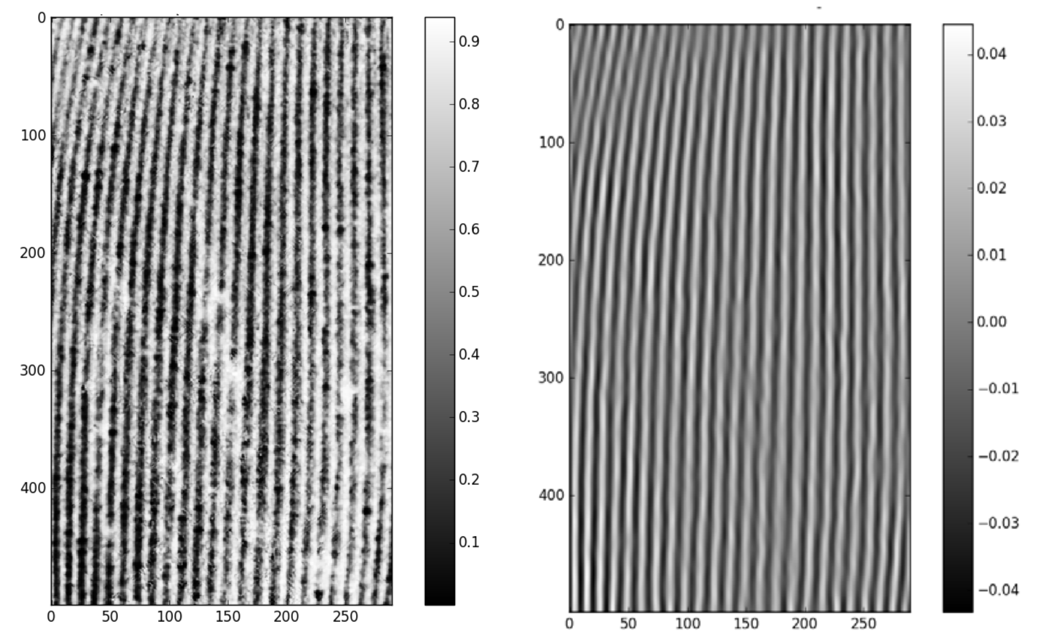


Figure 8: (LEFT) The probability map of a male cover scale. (RIGHT) The real component of the gabor filter with a frequency of 0.1. The frequency of 0.1 was chosen to suit the interval of the repeating linear pattern, which in this case, is visually identifiable as the valleys or ridges.

2.2 Reshaping the scale

With the proper image segmentation of the ridges and the valleys, one will be able to restructure the way he/she view the scale and finally to obtain new insights and findings. A restructured view of the scale will possibly provide a quantitative way of looking at the scale that could be used to be compared with other scales.

2.2.1 Fitting with parameters

After thresholding and labelling individual ridge and valleys as shown in Figure 9. The coordinates of ridges and valleys were, separately, fit to a 4th order polynomial and a line was plotted for each of them in Figure 9. A 4th order polynomial was chosen because it was sufficient to accurately fit the shape of the ridges and valleys, while not compromising on computation speed.

With this 4th order polynomials, it is possible to retrieve the pixel values at positions relative to the line. This can be done by substituting the "x" values onto the polynomial equations to retrieve the "y" values. This greatly simplifies the process of retrieving the relevant coordinates on the scale.

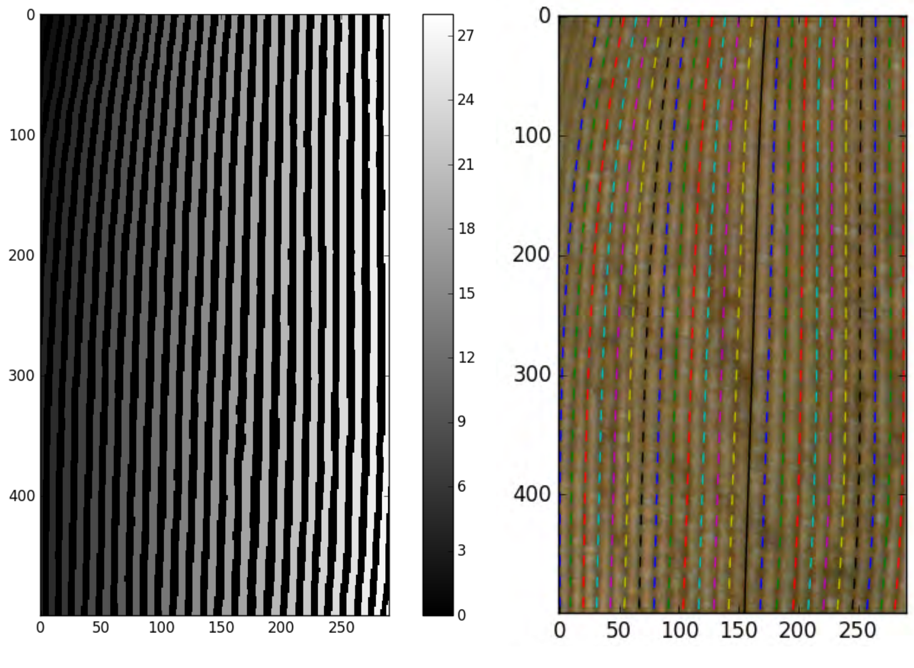


Figure 9: (LEFT) After threshold was applied and the ridges labelled with an index. (RIGHT) Fitted equation for the ridges was plotted against the original image of the cover scale to compare the quality of the fit.

2.2.2 Realigning the scale

From the polynomial parametrisation, it is now easier to obtain the "x" and "y" coordinates of the valleys or ridges on the scale. This implies that it is possible to also obtain the "x" values that are perpendicular to line, using the line that is normal to the polynomial. This is shown in Figure 10.

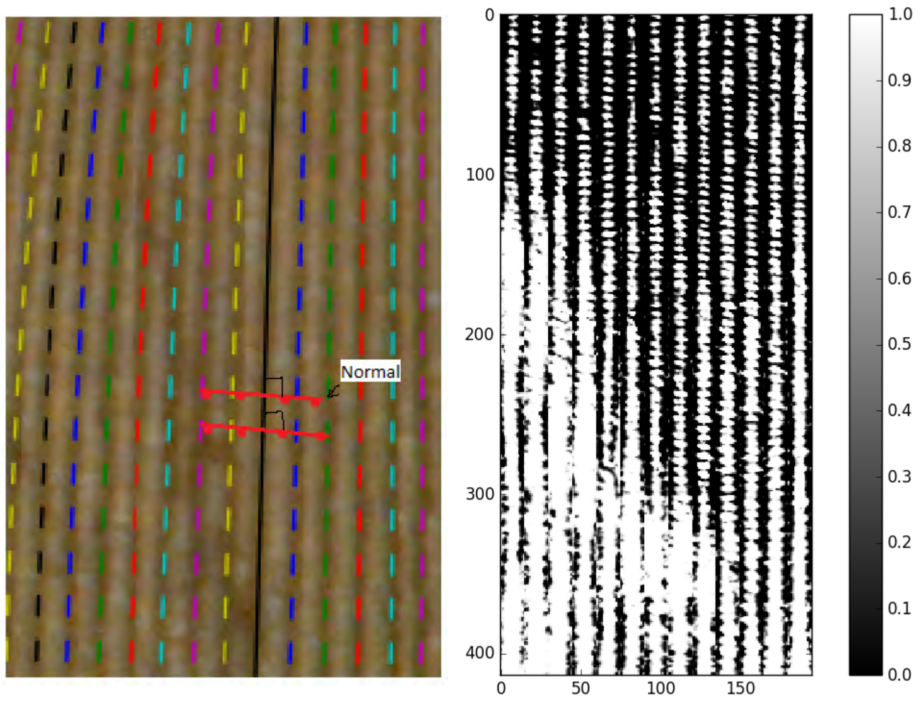


Figure 10: (LEFT) Normal lines were drawn perpendicular to the polynomials. A representative coordinate that is perpendicular to the polynomial was chosen to illustrate the perpendicular positions. (RIGHT) The probability map of a straightened scale where the valleys were straightened such that the centre of the valleys were at the centre of each column. Note the periodicity of the occurrence of the white "spots".

The straightened scale unlocks new possibilities in quantifying the diverse nature of butterfly scales. This straightening utilises the regularities between the ridges and valleys, ultimately providing some avenues for quantitative comparisons between different butterfly scales.

2.2.3 Box-plot of probability

With the fitted polynomials, the points of ridges were retrieved from the probability map and one can observe quantitatively how the average ridge looks like along the length of the scale as shown in Figure 12. This could be used to quantify the differences in structures between different scales from different butterflies.

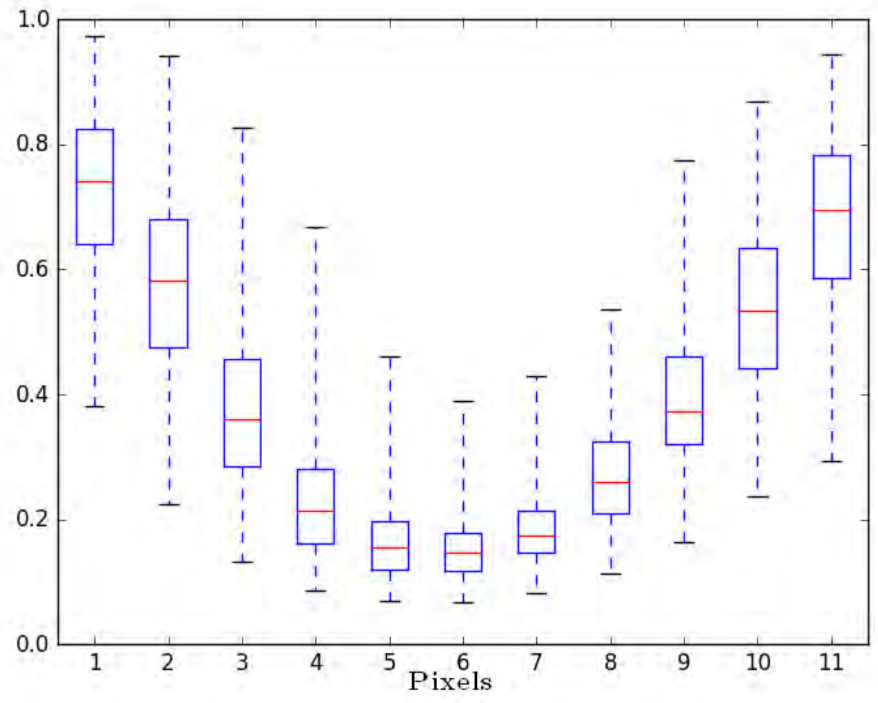


Figure 11: Statistical box-plot of a particular ridge. For every pixel along the ridge, the probability of the pixels to be classified as the ridge were tallied as a function of the distance to the central pixel. This box-plot shows the variations of the average ridge when looking at the neighbouring pixels.

2.3 Soft clustering for motifs

Another method to quantitatively analyse a butterfly scale is to look at the persistent motifs of the shape of the valleys. This is interesting to consider because it may serve as an indicator on the number of hidden variables for the formation of such structures as described in Section 1.

This section contains the description of the soft clustering algorithm which used to obtain the motifs. Section 2.3.2 explains the scoring function used to measure the difference between two images which is a crucial component for the algorithm to function.

2.3.1 Clustering algorithm

```
Input: images ($I$), classes ($C$)
Output: classes ($C^{'}$), weights ($\rho^{'}$)
Initialization;
for each $k$ in $I_k$ do
    for each $n$ in $C_n$ do
        for each $r$ in roll do
            $S_{k,n,r} = a\frac{\exp[\frac{-(I_{k,r}-I_n)^2}{\Delta}]}{\Delta}$
        end
    end
    Normalise the score;
    for each $n$ in $C_i$ do
        for each $r$ in roll do
            $C^{'}_n \mathrel{+}= I_{k,r} \cdot S_{k,n,r}$
            $\rho^{'}_n \mathrel{+}= S_{k,n,r}$
            $\Delta \mathrel{+}= S_{n,r} \cdot \overline{(C_n - I_{n,r})^2}$
        end
    end
end
```

Algorithm 1: Soft clustering algorithm

2.3.2 Scoring function

While there are many types of scoring functions, the suitability of the function depends on the use case and affects the rate of convergence.

$$S_{k,n,r} = a\frac{\exp[\frac{-(I_{k,r}-I_n)^2}{\Delta}]}{\Delta}, \tag{1}$$

where a is a normalising constant such that $\sum S_{k,n,r} = n(I) * n(C) * n(roll)$. This scoring function describes the differences between the classes and images by a Gaussian distribution. The sensitivity of the scoring function is controlled by the value of Δ in the exponent, a smaller value of Δ will cause score to be closer to zero while a larger value will cause the score to be closer to one. Hence, choosing an ideal value for Δ is important so as to not under-fit and over-fit the classes. The algorithm includes a section where it automatically fine tunes this parameter so that an optimal value of Δ can be obtained.

3 Discussion

The next two sections addresses the capacity of the methods described in Section 2 to quantify the features on different butterfly scales by comparing across butterflies of different genders in a particular species. Section 3.3 analyses the possible issues with reproducibility that was not considered. Section 3.4 discusses the extensions of this technique and how it could be extended to future studies.

3.1 Box-plot of probability

The image segmentation techniques as discussed in Section 2 was applied on male and female Bicyclus butterfly scales and showed promising results.

Observing the differences of the box plots between the scale from differ-

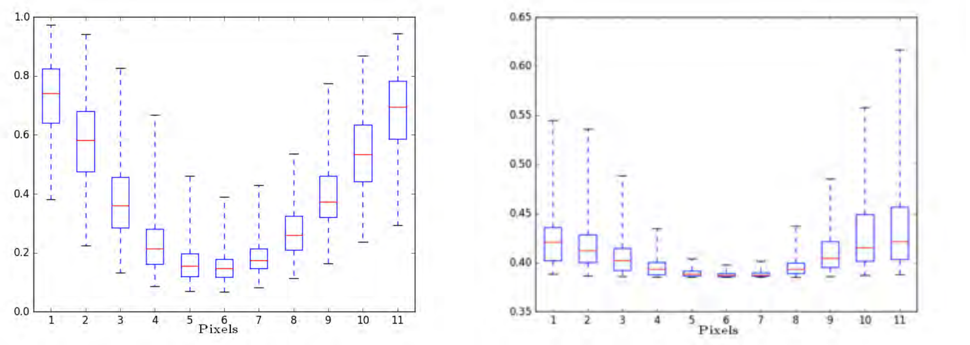


Figure 12: Box plot of an average ridge of (LEFT) male and (RIGHT) female Bicyclus butterfly scale. The deviation at the peak of the ridge is small and becomes larger when it is not at the peak. The difference in distribution were caused by difference in appearance of the butterfly scales and this features were likely to be caused by gender differentiation.

ent gender of the Bicyclus butterfly, one can conclude that there is indeed a statistical difference in the average shape of their ridges. Namely, the male ridge on the male Bicyclus butterfly varies more than the female variety.

Moreover, with the probability map of the scale, it is possible to reconstruct the 3D structure of the scale as shown in Figure 13. This was done with the assumption that the thickness of the scale is directly related to the probability that it is part of ridge. However, the limitation in this assumption is that the regions with zero probability would mean that the scale would have zero thickness. While this method produces a visually pleasing image, it may not correspond well with the actual ridge heights since the assumption of the relationship between probability and height may not hold true.

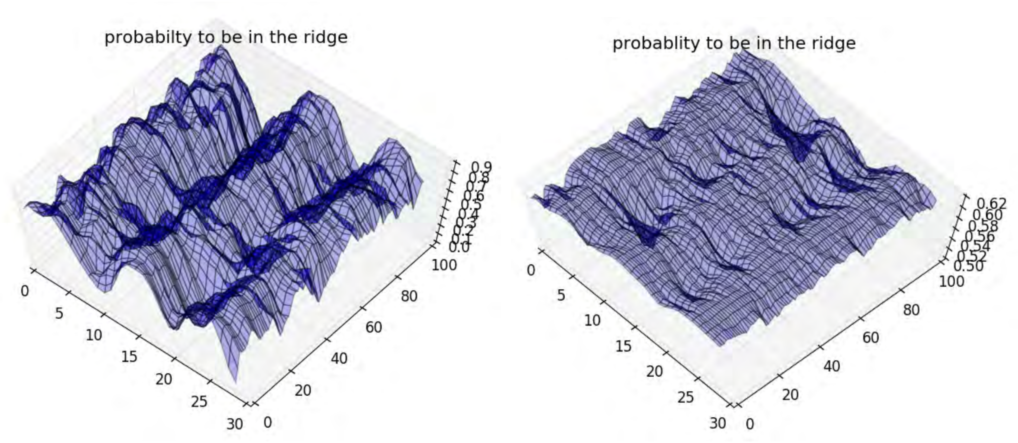


Figure 13: 3D surface plot of (LEFT) male and (RIGHT) female Bicyclus butterfly scale. The probability was used as the height in the z-axis.

3.2 Motifs on a butterfly scale

The soft clustering algorithm as described in Section 2 was applied to a butterfly scale in order to obtain the motifs. Recall that the purpose of studying the motifs is to attempt to answer the question of hidden variables, or at least provide insights to that issue.

From the different attempts as shown in Figure 14, it can be observed that the motifs were not consistent with each other despite being from the same scale. This clustering algorithm was further applied to the red channel of the

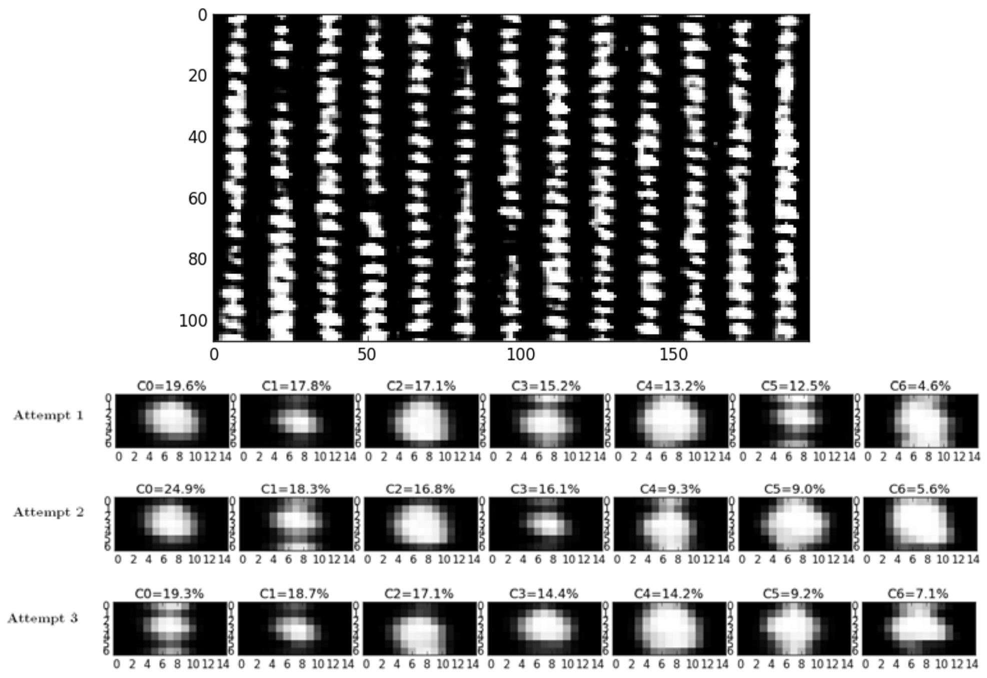


Figure 14: (TOP) Straightened probability map of the butterfly scale. (BOTTOM) Multiple attempts of different initialisation of the soft clustering algorithm. C_n refers to the n^{th} motif.

same section of the same butterfly scale and the motifs are shown in Figure 15 and showed similar results.

This results were unexpected since one would expect the motifs to appear the same for an identical image. To gain a better understanding, a fictional butterfly scale with known motifs was generated and the algorithm was applied, its results were reported in Appendix A. From the simulation, it was noted that the soft clustering algorithm was unable to obtain the same motifs if there are too many different motifs present on the image. This is likely to be the case since the valleys on the scale does look very different from each other.

While this may be a big hint that there may be too many hidden variables, there are still room for development to be able to provide a better

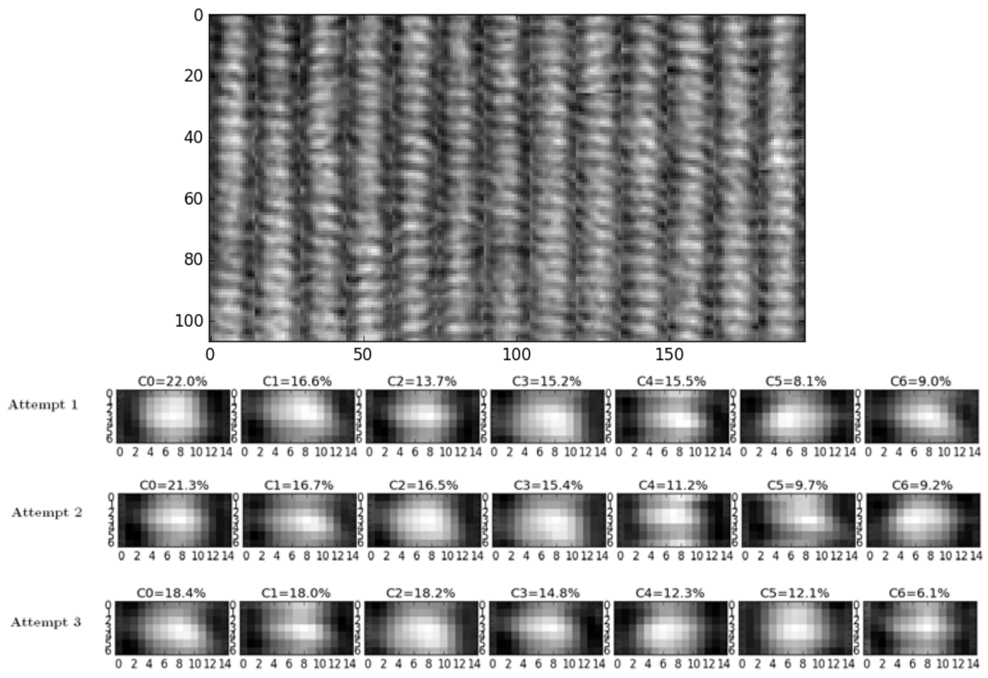


Figure 15: (TOP) Straightened red channel values of the butterfly scale. (BOTTOM) Multiple attempts of different initialisation of the soft clustering algorithm. C_n refers to the n^{th} motif.

gauge of the number of motifs. For example, one could naively do a pair wise comparison between motifs in a single attempt as well as between different attempts. The idea here is that a truly persistent motif would appear in most, if not all, of the attempts and it could provide a rough estimate on the number of motifs present.

3.3 Limitations and considerations

While being affordable and accessible, imaging butterfly scales with optical microscope images does have a multitude of limitations as well as factors to consider. In the development of a technique for a high throughput screening of butterfly scales, it is impartial that most, if not all, of these factors must be accounted for.

3.3.1 Type of microscopy

The butterfly scales would look different under different types of microscope and this will definitely affect the comparison studies. Consider the most common set-ups available in the common laboratory, namely, transmission microscopy and reflectance microscopy.

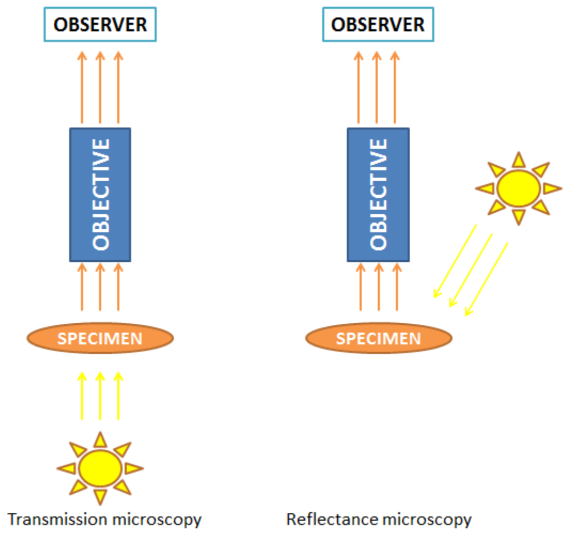


Figure 16: (LEFT) Schematic of light rays in transmission microscopy. (RIGHT) Schematic of light rays in reflectance microscopy.

Transmission microscopy is where the microscope illumination passes through the sample, followed by the microscope objective and finally to the viewer. Reflectance microscopy is where the microscope illumination reflects onto the surface of the sample and passes through the microscope objective and finally to the viewer as shown in Figure 16.

This is an important consideration when collecting optical microscopy images since butterfly scales have different optical properties under these lighting conditions. Most of the structural colour will show up in the reflectance microscopy images while both pigment and structural colour show up from the transmission microscopy.

A possible way of improving the way of fingerprinting of the structures is to compare the differences between the image from reflectance microscopy and the image from transmission microscopy. By comparing the two images, of the same butterfly scale, we can have a better contrast for the structural colours and thus providing a better fingerprint of the butterfly scale for its structural colour. This is provided that the butterfly scale images could be superimposed onto each other to obtain the differences.

3.3.2 Colour balance

Ideally, the optical image of the butterfly scale would look the same when using different optical microscopes. However, this is not usually true because of difference in the image capturing device on the microscope as well as the background illumination.

The difference in the illumination will cause the colours on the butterfly scale to appear significantly different when captured by a camera. For example, when using a yellow sodium lamp as compared to a mercury vapour lamp, the image will appear to be more yellow or more blue respectively. This will cause the butterfly scale to look discoloured and thus skewing the results from the colour fingerprinting.

Colour correction will be needed to ensure that the optical images were compared with the same conditions. A possible implementation is to image the butterfly scale beside a standard colour chart. The colour chart could be used to calibrate any camera and any lighting such that quantitative comparison studies can be done.

3.3.3 Refractive index of medium

The medium in which the butterfly scales were immersed in will also alter the colours captured by the optical microscope. For example, if a transmission microscope was used to capture butterfly scale images, immersing the butterfly scale in a solution with similar refractive index will reduce the contrast between ridges and valleys. This is because less light will be reflected and hence more light will be transmitted, reducing contrast on the images. This is evident in Figure 17.

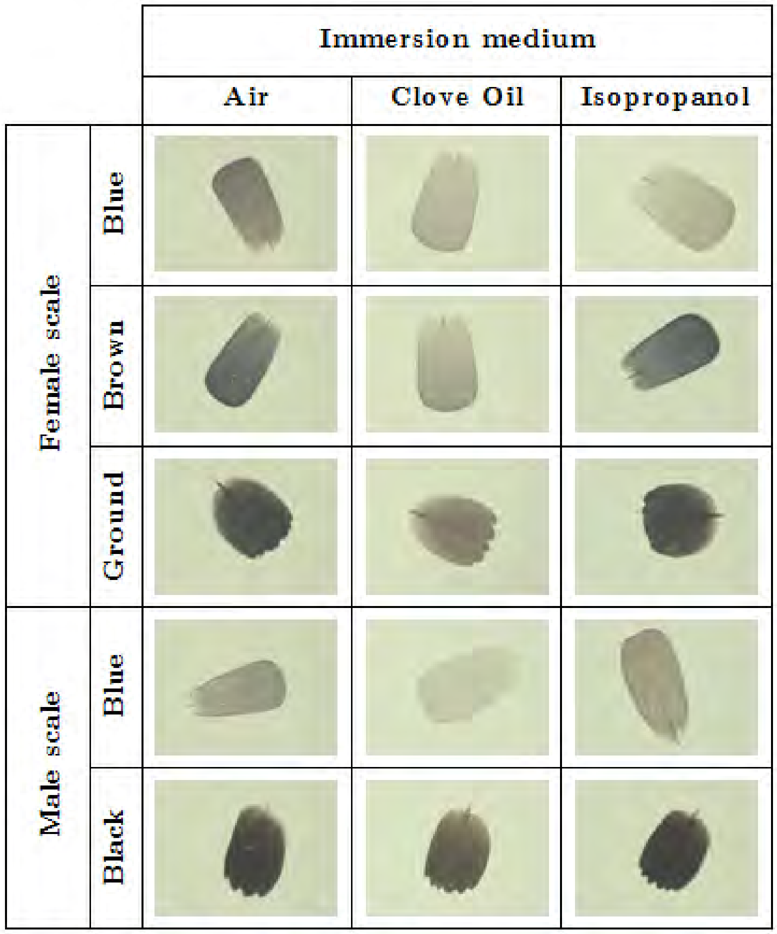


Figure 17: Transmission microscopy images of Bicyclus butterfly scales immersed in various mediums. The scales show a larger contrast in air as compared to clove oil. The refractive index of clove oil is similar to that of the butterfly scale. Images taken by Anupama Prakash.

3.4 Future works

The future works can be broadly categorised into two categories, namely, development phase and application phase.

In the development phase, a deeper understanding of obtaining the motifs is needed. As suggested by Section 3.2, obtaining an ideal number of motifs may not be a trivial implementation. Furthermore, it is vital to be able to correlate this persistence in motifs with known and controlled changes in the butterfly scale. As a next step in the development phase, for example, one can look at motifs of butterfly scales where a particular gene was not expressed or purposely suppressed and compare to that of a wild type butterfly scale.

In the application phase, one has to consider the the myriad of factors that will affect the quality of the optical image. As suggested in 3.3, a standard chart could be used to calibrate different microscope and different lighting conditions. However, colour correction schemes are non-trivial and must be customised to suit the purpose of imaging butterfly scales, since the scales are usually and translucent may oversaturate the camera.

As a further extension to the project, it is possible to consider other types of motifs on a butterfly. For example, one can look at the motif of colours on a butterfly scale across the whole wing of a single butterfly. This process eliminates the need for having genetic variations and instead considers chemical signals between scales as a potential source of hidden variables. With these type of studies, one would be able to know how many type of colours there are on a butterfly. Using this knowledge, one can infer the number of mechanisms the butterfly has to produce such colours. This is an exciting approach because there must be an finite number of ways a butterfly can produce colours, since there are only a limited number of genes. However, the risk may be that the finite list of ways may be too large of a list to provide useful insights.

4 Conclusion

Overall, the study of structural colours in butterfly scales using microscope images was discussed. Gaussian mixture model have shown spectacular results as a general implementation of segmenting an optical image of the butterfly scale. This led to the possibility of identifying and quantifying the features on the scales. Statistical comparison using box-plots of probability maps could be used as a method to compare between scales.

The use of motifs to characterise similar features on a butterfly scale was implemented and, despite being preliminary and developmental, have shown some promising results. The methods discussed could have a big impact on the study of the mechanisms of genetic variations on structural colour because of its high throughput capabilities.

References

[1] W.Owen McMillan, Antnia Monteiro, and Durrell D. Kapan. Development and evolution on the wing. *Trends in Ecology and Evolution*, 17(3):125–133, 2002.

[2] S Gorb. *Functional Surfaces in Biology: little structures with big Effects*. Springer, 1 edition, 2009.

[3] Andrew Richard Parker. 515 million years of structural colour. *Journal of Optics A: Pure and Applied Optics*, 2(6):R15–R28, 2000.

[4] M. A. Giraldo and D. G. Stavenga. Brilliant iridescence of morpho butterfly wing scales is due to both a thin film lower lamina and a multilayered upper lamina. *Journal of Comparative Physiology A*, 202(5):381–388, 2016.

[5] D. G. Stavenga, H. L. Leertouwer, and B. D. Wilts. Coloration principles of nymphaline butterflies - thin films, melanin, ommochromes and wing scale stacking. *Journal of Experimental Biology*, 217(12):2171–2180, 2014.

[6] Antnia Monteiro. Origin, development, and evolution of butterfly eyespots. *Annual Review of Entomology*, 60(1):253–271, 2015.

[7] B. R. Wasik, S. F. Liew, D. A. Lilien, A. J. Dinwiddie, H. Noh, H. Cao, and A. Monteiro. Artificial selection for structural color on butterfly wings and comparison with natural evolution. *Proceedings of the National Academy of Sciences*, 111(33):12109–12114, 2014.

[8] P. Thibault, M. Dierolf, A. Menzel, O. Bunk, C. David, and F. Pfeiffer. High-resolution scanning x-ray diffraction microscopy. *Science*, 321(5887):379–382, 2008.

[9] M. Holler, A. Diaz, M. Guizar-Sicairos, P. Karvinen, Elina Frm, Emma Hrknen, Mikko Ritala, A. Menzel, J. Raabe, and O. Bunk. X-ray ptychographic computed tomography at 16 nm isotropic 3d resolution. *Scientific Reports*, 4, 2014.

A Appendix - Simulation

A.1 Simulation of a butterfly scale

A fictional butterfly scale was prepared in the following sequence of steps. First, 4 unique classes of valleys were drawn and each multiplied to a Gaussian blur as shown in Figure 18 and finally added with a 10% normally distributed noise as shown in Figure 19 (LEFT). Next, they were stitched together in random sequences to form a segmented section of a fictional butterfly scale.

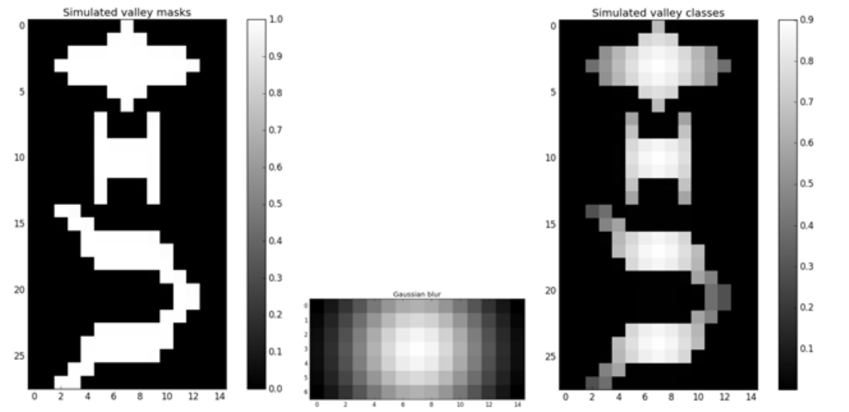


Figure 18: (LEFT) Mask for the valleys. (CENTRE) Gaussian blur array. (RIGHT) Classes of valleys.

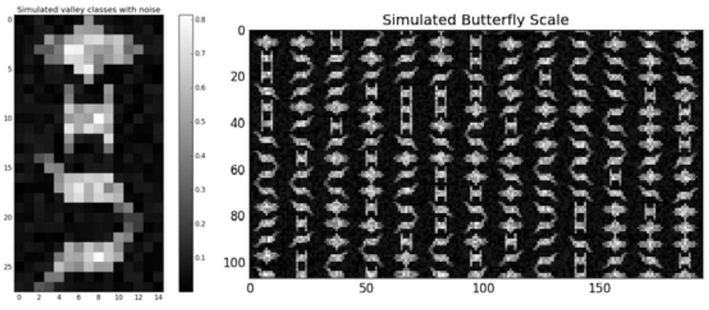


Figure 19: (LEFT) Classes for valleys with 10% noise. (RIGHT) The simulated butterfly scale.

A.2 Proof of concept

The simulated scale was put into the soft clustering algorithm and the number of classes was varied to observe the quality of the clustering when there was an excess of classes or when there was a lack thereof.

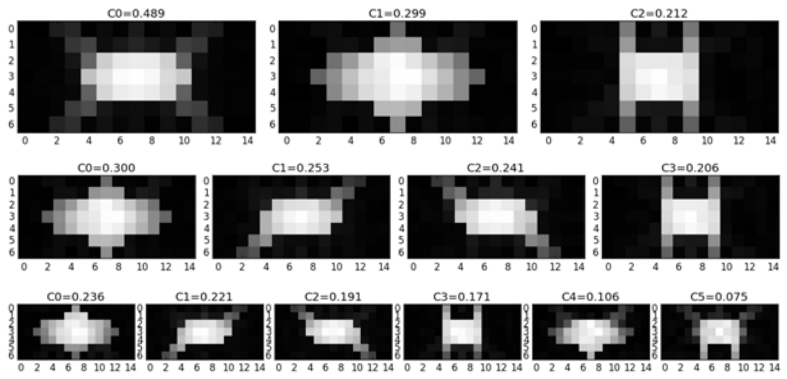


Figure 20: Top to bottom: 3, 4 and 6 classes were used in 40 iterations of the soft clustering algorithm. $Cn = 0.489$ *indicates the fraction of the valleys (48.9%) that were labelled as class 1.*

With reference to Figure 20, it can be observed that if an exact amount (4) of classes was provided to the soft clustering algorithm, the clustering algorithm was able to accurately retrieve the 4 simulated classes of valleys. However, if too few or too much classes were provided, one or some of the retrieved class will have a non-unique superposition of multiple classes. Additionally, when too many classes were provided to the soft clustering algorithm, the excess classes will have a smaller fraction of valleys to be labelled in that class.

B Appendix - Soft clustering package

The package for image segmentation and soft clustering was written in python.

B.1 Usage

The code was run for multiple images and the result is shown in Figure 21.

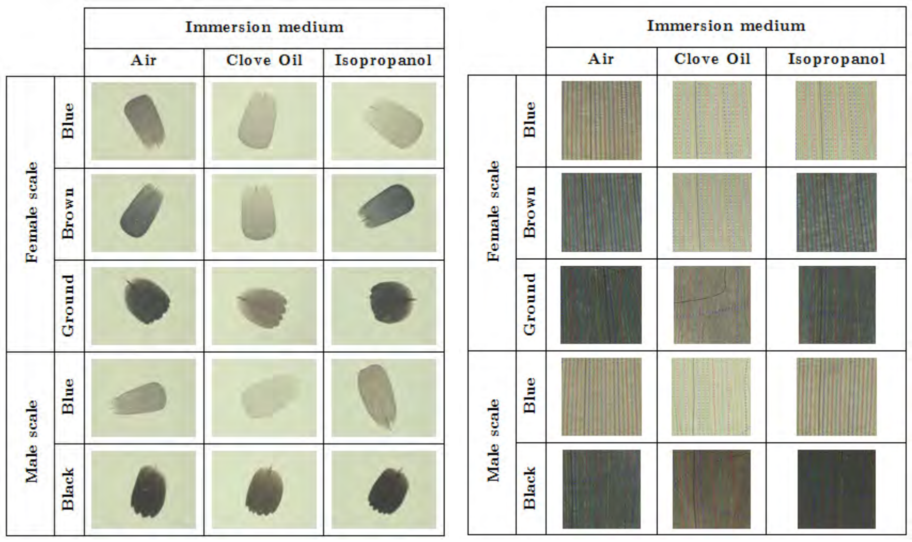
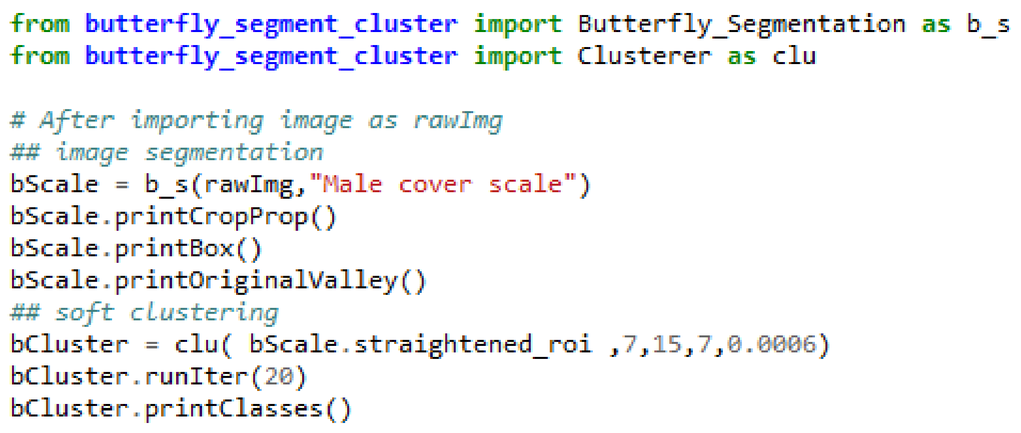


Figure 21: Code applied onto microscope images of Bicyclus butterfly scales. Images taken by Anupama Prakash.



```python
from butterfly_segment_cluster import Butterfly_Segmentation as b_s
from butterfly_segment_cluster import Clusterer as clu

# After importing image as rawImg
## image segmentation
bScale = b_s(rawImg,"Male cover scale")
bScale.printCropProp()
bScale.printBox()
bScale.printOriginalValley()
## soft clustering
bCluster = clu( bScale.straightened_roi ,7,15,7,0.0006)
bCluster.runIter(20)
bCluster.printClasses()
```

Figure 22: Code snippet for the usage of the package.

B.2 Package

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 20 17:54:02 2017
This package contains two classes: Clusterer and Butterfly_Segmentation

    -Butterfly_Segmentation segments the butterfly scale image.
    -Clusterer implements the soft clustering algorithm on the segmented image.

@author: yeo zhen yuan
"""

from skimage.filters import gabor
from scipy import ndimage as ndi
from skimage import feature
from matplotlib import pyplot as plt
import numpy as np
from sklearn import mixture
from sklearn import decomposition
from scipy.ndimage.interpolation import rotate
from skimage.measure import label
#%%
class Butterfly_Segmentation:
    """
    Segments an optical image of a butterfly scale.
    Applies gaussian mixture model to seperate the ridges from the valleys.
    Reshapes the valleys into a straight line.

    Print functions:
        showRawImg()
        printCropProp()
        printBox()
        printStraightenedValley()
        printOriginalValley()

    """
    def __init__(self,rawImg,name):
        self.name = name
        self.rawImg = rawImg
        #printCropProp variables
        self.rot_plots = None
        self.boxpoints = None
        self.plotrect0 = None
        self.plotrect1 = None
        self.angle = None

        self.croppedImg = None
        self.probmapridge = None
        self.probmapvalley = None
        #print box variables
        self.newbinarised = None
        self.croppedImg = None
        self.ridge_label = None
        self.valley_label = None
        self.scale_eqn = None
        self.scale_yp = None
        self.ridgedatapoints = None
        #printStraighten variables
        self.straightened_roi = None
        self.relabeled_w3 = None
        #process the image
        self.process_image()
    #%% public functions
    def showRawImg(self):
        """
        Displays the input image.
        """
        self.__showRe(self.rawImg,self.name)
    def printCropProp(self):
        """
        Prints the probability map.
        """
        figusize = 20
        figu = plt.figure(figsize=(figusize, figusize))
        afigu=figu.add_subplot(2,3,1)
        imgplot = plt.imshow(self.rawImg, interpolation='none')
        afigu.set_title(self.name)
        afigu=figu.add_subplot(2,3,2)
        afigu.set_title("rotated+crop")
        for each_point in self.rot_plots:
            plt.scatter(each_point[0],each_point[1],c=each_point[2])
        plt.scatter(self.boxpoints[::,1],self.boxpoints[::,0],color="g")
        plt.plot( self.plotrect0,self.plotrect1, 'g--')
        plt.imshow(rotate(self.rawImg,self.angle,reshape=False))
        afigu=figu.add_subplot(2,3,3)
        afigu.set_title("cropped")
        imgplot = plt.imshow(self.croppedImg, interpolation='none')
        #showRe(croppedImg,"cropped img") #3
        afigu=figu.add_subplot(2,3,4)
        afigu.set_title("ProbValley")
        imgplot = plt.imshow(self.probmapvalley,cmap='gray',
                             interpolation='none')
        plt.colorbar(imgplot)
        #showRe(probmapvalley,"prob valley") #4
        afigu=figu.add_subplot(2,3,5)
        afigu.set_title("ProbRidge")
        imgplot = plt.imshow(self.probmapridge,cmap='gray',
                             interpolation='none')
        plt.colorbar(imgplot)
        #showRe(probmapridge,"prob ridge") #5
        plt.subplots_adjust(wspace=0.1,hspace=0.1)
        plt.show()
    def printBox(self):
        """
        Prints the box plot for a particular ridge.
        """
        figusize = 20
        figu = plt.figure(figsize=(figusize, figusize))
        plt.subplots_adjust(wspace=0.1,hspace=0.2)
        afigu=figu.add_subplot(2,3,1)
        afigu.set_title("freq=0.085")
        plt.imshow(np.real(self.newbinarised),
                   interpolation='none',cmap='gray')
        afigu=figu.add_subplot(2,3,2)
        afigu.set_title("cropped")
        plt.imshow(np.real(self.croppedImg),
                   interpolation='none',cmap='gray')
        afigu=figu.add_subplot(2,3,3)
        afigu.set_title("ridgesnonzero")
        plt.imshow(np.real(self.ridge_label),
                   interpolation='none',cmap='gray')
        afigu=figu.add_subplot(2,3,4)
        afigu.set_title("valleysnonzero")
        plt.imshow(np.real(self.valley_label),
                   interpolation='none',cmap='gray')
        afigu=figu.add_subplot(2,3,5)
        for idx,eqn in enumerate(self.scale_eqn):
            if idx != 6 :
                plt.plot(  eqn(self.scale_yp) ,self.scale_yp,"--")
            else:
                plt.plot(  eqn(self.scale_yp) ,self.scale_yp,"-")
        afigu.set_title("blk=eqn6=ridge")
        plt.imshow(self.croppedImg,interpolation="none")
        """
        Boxplot functionality.
        """
        afigu=figu.add_subplot(2,3,6)
        plt.boxplot(self.ridgedatapoints, whis='range')
        plt.ylim((0,1))
        afigu.set_title('box plot of multiple ridges for'+str(self.name))
    def printStraightenedValley(self):
        c_label = np.expand_dims(self.relabeled_w3,axis=1)/2
        straightenedValley = np.append(self.straightened_roi,c_label,axis=1)
        self.__showRe(straightenedValley,"Straightened Valley")
    def printOriginalValley(self):
        self.__showRe(self.original_roi,"Straightened Valleys, width = "+
                      str(self.width))
    def process_image(self):
        print("Processing image...")
        #%% crop and rotate
        print("Cropping and rotating...")
        edges1 = feature.canny(self.rawImg.mean(axis=2), sigma=90)
```

```python
        filled_edges = ndi.binary_fill_holes(edges1)
        coords = np.where(filled_edges==1)
        points = np.concatenate(([coords[1]],[coords[0]]),axis=0).T
        #%% pca on the scatter
        pca = decomposition.PCA(n_components=2)
        pca.fit(points)
        pcaScale0 = pca.transform(points)
        #%% the 4 important points
        value1 = np.where(pcaScale0[::,0]==np.min(pcaScale0[::,0]))
        value2 = np.where(pcaScale0[::,0]==np.max(pcaScale0[::,0]))
        value3 = np.where(pcaScale0[::,1]==np.max(pcaScale0[::,1]))
        value4 = np.where(pcaScale0[::,1]==np.min(pcaScale0[::,1]))
        #%% from 4 points rotate
        a=points[value1[0]][0]
        b = points[value2[0]][0]
        angle = np.rad2deg(np.arctan2( a[-1] - b[1],a[0] - b[0])+np.pi/2)

        self.rot_plots = [[points[value1[0],0][0],points[value1[0],1][0],"r"],
                          [points[value2[0],0][0],points[value2[0],1][0],"r"],
                          [points[value3[0],0][0],points[value3[0],1][0],"r"],
                          [points[value4[0],0][0],points[value4[0],1][0],"r"]]
        boxpoints = np.array([[]])
        for value11 in [value1,value2,value3,value4]:
            newpoint = self.__rotate_point(points[value11[0],1],
                                           points[value11[0],0],
                                            angle,
                                            self.rawImg.shape[0]/2,
                                            self.rawImg.shape[1]/2)
            if boxpoints.shape == 0 :
                boxpoints = newpoint
            else:
                boxpoints = np.append(boxpoints ,newpoint)
        boxpoints= boxpoints.reshape(-1,2)
        self.boxpoints = boxpoints
        xwidth = np.abs(boxpoints[0,0]-boxpoints[1,0])//5
        ywidth = np.abs(boxpoints[2,1]-boxpoints[3,1])//5
        if boxpoints[2,1]>boxpoints[3,1]:
            y2 , y1 = boxpoints[2,1],boxpoints[3,1]
        else:
            y2,y1 = boxpoints[3,1],boxpoints[2,1]
        x1 = boxpoints[0,0]
        x2 = boxpoints[1,0]
        self.plotrect0 = [y1+ywidth,y2-ywidth,y2-ywidth,y1+ywidth,y1+ywidth]
        self.plotrect1 = [x2-xwidth, x2-xwidth,x1+xwidth,x1+xwidth,x2-xwidth]
        self.angle = angle
        #%% from 4 points rotated, crop and then continue
        rotatedImg = rotate(self.rawImg,angle,reshape=False)
        croppedImg = rotatedImg[int(boxpoints[0,0]+xwidth):
                                int(boxpoints[1,0]-xwidth),
                                int(y1+ywidth):
                                int(y2-ywidth)]
        #%% gaussian mixture model to get the seperation
        print("Applying gaussian mixture model clustering...")
        g = mixture.GMM(n_components=2)
        coords5 = croppedImg.reshape(-1,3)
        #%% add pcascale to coords
        g.fit(coords5)
        w=g.predict_proba(np.array(coords5,dtype="int64"))
        #%% finding the correct class for valleys or ridges
        c_shape = croppedImg.shape[:2]
        prob_class_0 = w[:,0].reshape(c_shape)
        prob_class_1 = w[:,1].reshape(c_shape)
        class0x = np.where(prob_class_0==np.max(prob_class_0))[0][0]
        class0y = np.where(prob_class_0==np.max(prob_class_0))[1][0]
        class1x = np.where(prob_class_1==np.max(prob_class_1))[0][0]
        class1y = np.where(prob_class_1==np.max(prob_class_1))[1][0]
        class0_brightness = np.sum(croppedImg[class0x,class0y])
        class1_brightness = np.sum(croppedImg[class1x,class1y])
        #compares between the brightest points to identify ridge from valleys
        if class0_brightness < class1_brightness:
            probmapvalley = prob_class_1
            probmapridge = prob_class_0
        else:
            probmapvalley = prob_class_0
            probmapridge = prob_class_1
        self.croppedImg = croppedImg
        self.probmapvalley = probmapvalley
        self.probmapridge = probmapridge
        #%% doing analysis on the probmap
        print("Running gabor filter...")
        for freq in [0.085]:
            filt_real, filt_imag = gabor(probmapridge, frequency=freq)
        newbinarised = filt_real>0
        valley_label = label(1-newbinarised)
        ridge_label = label(newbinarised)
        self.newbinarised = newbinarised
        self.valley_label = valley_label
        self.ridge_label = ridge_label
        #%% fitting eqns to ridges and valleys
        print("Fitting eqns to ridges and valleys...")
        eqn_ridge=()
        for i in range(1,np.max(ridge_label)):
            coords = np.where(ridge_label == i)
            coef= np.polyfit(coords[0], coords[1], 4)
            p3 = np.poly1d(coef)
            if  len(coords[0]) >200 and len(coords[0]) <5000:
                eqn_ridge = p3 if eqn_ridge.count ==0 else eqn_ridge + (p3,)

        eqn_valley=()
        for i in range(1,np.max(valley_label)):
            coords = np.where(valley_label == i)
            coef= np.polyfit(coords[0], coords[1], 4)
            p3 = np.poly1d(coef)
            if  len(coords[0]) >200 and len(coords[0]) <5000:
                eqn_valley = p3 if eqn_valley.count ==0  else eqn_valley +(p3,)
        #%### drawing lines on the original image
        yp = np.linspace(0,croppedImg.shape[0]-1,croppedImg.shape[0])
        self.scale_eqn = eqn_ridge
        self.scale_yp = yp
        #%### observing one particular ridge over the length of the scale
        xp = np.linspace(10,croppedImg.shape[0]-21,croppedImg.shape[0]-30)
        widthh = 5
        one_ridge = np.array([])
        for ridge in [4,5,6]:
            temp = self.__imcurvecorrected(probmapridge,xp,
                                           eqn_ridge[ridge],widthh)
            one_ridge = temp if len(one_ridge)==0 else np.append(temp,
                                                                  one_ridge,
                                                                               axis=0)
        r_pts = np.array([])
        for row in range(len(one_ridge)):
            roww = [one_ridge[row]]
            r_pts = roww if len(r_pts)==0 else np.append(r_pts,roww,axis=0)
        #%% ###
        self.ridgedatapoints = r_pts
        # cropping to the top half of the probability map
        crop_height1 = 0
        crop_height2 = 107
        yp=np.arange(crop_height1,crop_height2)#probmapvalley.shape[0])
        for wid in [7]:
            highestpoints1 = self.__imcurvecorrected(probmapvalley,
                                                     yp,eqn_valley[3],wid)
            highestpointsShow = np.copy(highestpoints1)
            for vnumber in range(4,len(eqn_valley)-3):
                temp = self.__imcurvecorrected(probmapvalley,yp,
                                               eqn_valley[vnumber],wid)
                highestpoints1 = np.append(highestpoints1,temp,axis=0)
                highestpointsShow = np.append(highestpointsShow,temp,axis=1)
        straightened_roi = np.copy(highestpoints1)
        original_roi = np.copy(highestpointsShow)
        #%% clustering by gausion
        g3 = mixture.GMM(n_components=3)
        feature_set = straightened_roi #croppedImg.reshape(-1,3)
        pca3 = decomposition.PCA(n_components=15)
        pca3.fit(feature_set)
        pcaScale3 = pca3.transform(feature_set)
        g3.fit(pcaScale3)
        w3=g3.predict_proba(pcaScale3)
        relabeled_w3 = np.argmax(w3,axis=1)
        ## saving straighted ridge
        self.straightened_roi = straightened_roi
        self.original_roi = original_roi
        self.width = wid
        self.relabeled_w3 = relabeled_w3
    #%% helper methods
    def __saveRe(self,array,title):
        """
        Prints and saves real part of an array, with title as string
        """
        width = 10
        height = 10
        plt.cla
        plt.figure(figsize=(width, height))
        imgplot = plt.imshow(np.real(array),cmap='gray', interpolation='none')
        plt.colorbar(imgplot)
        plt.title(str(title))
        plt.savefig(str(title)+".png")
    def __showRe(self,*arg):
        """
        Prints real part of an array, with title as string
        """
        array = arg[0]
        width = 10
        height = 10
        plt.cla
        plt.figure(figsize=(width, height))
        imgplot = plt.imshow(np.real(array),cmap='gray', interpolation='none')
        plt.colorbar(imgplot) #, interpolation='none',aspect='auto'
        if len(arg) == 2:
            plt.title(str(arg[1]))
    def __between(self,p1,ratio,p2):
        return 1.0*p1*(1-ratio)+1.0*p2*ratio
    def __bilinear(self,im,row,col): #row, col
        """
        Returns the value of the interpolated pixels.
        """
        # input the row and col of the interested region
        a, b =im.shape[:2] #row col
        row,col = np.clip([row,col],[0,0],[a-2,b-2])
        col1, row1 = int(col), int(row)
        col2, row2 = col1+1, row1+1
        p1 = self.__between(im[row1,col1],row-row1,im[row2,col1])
        p2 = self.__between(im[row1,col2],row-row1,im[row2,col2])
        return self.__between(p1,col-col1,p2)
    def __rotate_point(self,x1,y1,angle,xcentre,ycentre):
        """
        Returns rotated points based on angle and centre of rotation.
        """
        centeredx = x1 - xcentre
        centeredy = y1 - ycentre
        s = np.sin(angle/180*np.pi)
        c = np.cos(angle/180*np.pi)
        newx1 = centeredx*c - centeredy *s +xcentre
        newy1 = centeredx*s + centeredy *c +ycentre
        return [newx1,newy1]

    def __imcurvecorrected(self,im,rows,eqn,width):
        """
        Returns values of interpolated pixel values in im based on eqn.
        """
        pixels = np.array([])
        for height in rows:
            normgrad = np.polyder(eqn)(height)
            inv_norm_grad = 1. / normgrad
            n =1.0/np.sqrt(1.+ inv_norm_grad**2)
            (c, r) = (inv_norm_grad*n, -1.*n)
            row = np.array([])
            for wid in range(-width,width+1):
                i_polated = self.__bilinear(im,height+r*wid,eqn(height)+c*wid)
                row = np.append(row,np.array([i_polated]),axis=None )
            pixels = row if len(pixels)==0 else np.append(pixels,row,axis=0)
        if len(im.shape)==3:
            return pixels.reshape(len(rows),width*2+1,3)
        else:
            return pixels.reshape(len(rows),width*2+1)


class Clusterer:
    """
    Given a straighted valley, locates clusters of persistent motifs.

    Print functions:
        printClasses()
    """
    def __init__(self, im,window_height,window_width,num_of_class,curr_delta):
        self.im = im
        self.window_height = window_height # 7
        self.window_width = window_width # 15
        self.num_of_class = num_of_class # 7
        self.iter_count = 0
        #initalise the zeros / random arrays
        self.set_of_classes = np.random.rand(self.num_of_class,
                                             self.window_height,
                                             self.window_width)
        self.new_set_of_classes = self.set_of_classes * 0
        self.roll_for_each_class = np.zeros(self.set_of_classes.shape[0])
        self.count_addition = np.zeros(self.num_of_class)
        self.weights = np.zeros(self.num_of_class)
        self.curr_delta = curr_delta
        self.new_delta = 0.
#%% helper classes
    def __center_class(self,s_set_of_classes):
        s_new_set_of_classes=np.zeros(s_set_of_classes.shape)
        for class_index in np.arange(s_set_of_classes.shape[0]):
            amax = s_set_of_classes[class_index].sum(axis = 1).argmax()
            unrolled = np.roll(s_set_of_classes[class_index],
                               -amax +(len(s_set_of_classes[class_index]))//2,
                                axis=0)
            s_new_set_of_classes[class_index]= unrolled
        return np.copy(s_new_set_of_classes)
    def __reorder_classes(self):
        """
        Reorder the classes so that the max is in front.
        Reorder the count_addition correspondingly.
        """
        set_of_classes,count_addition=(self.set_of_classes,self.count_addition)
        new_count = np.sort(count_addition)
        new_set_of_class = set_of_classes[count_addition.argsort()]
        self.set_ofclasses = np.copy(new_set_of_class[::-1])
        self.count_addition = np.copy(new_count[::-1])
#%% proper run iter
    def runIter(self,num):
        for times in np.arange(num):
            #initialise:
            self.weights = np.zeros(self.num_of_class)
            self.new_set_of_classes = np.copy(self.set_of_classes * 0)
            #scoring function:
            num_windows = self.im.shape[0]//self.window_height-1
            for k in np.arange(num_windows):
                score = np.zeros((len(self.set_of_classes),self.window_height))
                window_arr = self.im[int(k*self.window_height):
                                     int((k+1)*self.window_height)]
                for n , class_arr in enumerate(self.set_of_classes):
                    for r in np.arange(window_arr.shape[0]):
                        rolled_window = np.roll(window_arr,r,axis=0)
                        var=np.mean(np.abs(class_arr-rolled_window)**2)
                        #print (var)
                        score[n,r] = np.exp(-var/self.curr_delta)
            #normalising the score:
                score/= score.sum()
            #updating clusters
                window_arr = self.im[int(k*self.window_height):
                                     int((k+1)*self.window_height)]
                for n , class_arr in enumerate(self.set_of_classes):
                    for r in np.arange(window_arr.shape[0]):
                        rolled_window = np.roll(window_arr,r,axis=0)
                        self.new_set_of_classes[n] += score[n,r]*rolled_window
                        self.weights[n] += score[n,r]
                        average = np.mean(np.abs(class_arr-rolled_window)**2)
                        self.new_delta += score[n,r]*average
            #scaling updated clusters with weights
            self.curr_delta = self.new_delta/num_windows/n/r
            self.new_delta = 0.
            print ("Current delta = " + str(self.curr_delta))
            for n , class_arr in enumerate(self.set_of_classes):
                if self.weights[n] > 0:
                    self.new_set_of_classes[n] /= self.weights[n]
            #setting up for next iteration
            print("Iteration "+str(self.iter_count+1))
            self.set_of_classes = np.copy(self.new_set_of_classes)
            self.count_addition = np.copy(self.weights)
            #recenter classes every 10 iteration.
            if (self.iter_count%10 == 0):
                self.set_of_classes = self.__center_class(self.set_of_classes)
                self.__reorder_classes()
            self.iter_count += 1
    #%% print classes
    def printClasses(self):
        figusize = 20
        figu = plt.figure(figsize=(figusize, figusize))
        for n in np.arange(self.set_of_classes.shape[0]):
            afigu=figu.add_subplot(1,self.set_of_classes.shape[0],n+1)
            percent = 100.0*self.count_addition[n]/self.count_addition.sum()
            afigu.set_title("C"+str(n)+"=" +"%0.1lf"%(percent)+"%")
            afigu.imshow(np.real(self.set_of_classes[n]),cmap='gray',
                         interpolation='none')
        plt.suptitle("run = "+str(self.iter_count))
        plt.subplots_adjust(wspace=0.1,hspace=0.1)
        plt.show()
```