

Report of IMA205 Challenge 2019

- Classify images as either melanoma or benign nevus

Chun Wu

I. Introduction

There has been a steady increase in the incidence of skin cancer worldwide, with a high rate of mortality. Early detection and segmentation of skin lesions are crucial for timely diagnosis and treatment, necessary to improve the survival rate of patients. The objective of this challenge is to classify images of skin lesions as either benign or melanoma. In order to do that, we will do a chain of work: (a) image segmentation to separate the lesion area from the background skin, (b) extraction of image features for classification purposes, and (c) final classification using statistical methods.

For the data, we have a data-set of 1000 RGB images of skin lesions with their relative segmentation. The training dataset contains 700 of those images and we only have the classification of the training dataset. While, we need to estimate or predict the correct class (benign or malign) for the rest 300 images in the test dataset.

I used Python 3 as programming language and imported panda, numpy, cv2, scikit-learn and etc.

The remainder of the report is organized as follows. Section II provides a description of the data and feature extraction. Section III describes some pre-processing and provides several classification algorithms I used and their corresponding results. And some conclusions are respectively drawn in Section IV.

II. Feature extraction

This part is almost the key of this challenge. First, these descriptive features attempt to quantify the asymmetry, the borders, and the colors of the lesions. These are among the anatomical attributes that dermatologists acknowledge are important for diagnosing melanomas. So, I decided to extract features based on the ABCD rule. The ABCD stands for Asymmetry, Border structure, Color variation and Diameter of lesion. It defines the basis for diagnosis of disease. I extracted Asymmetry, Border, Color and Diameter features along with lesion area, centroid, perimeter and some others.

After the data acquisition, I have observed that there are some segmented images where the lesion part goes beyond the image border (with "white" in the border). So, we

should clean this kind of image. I detected if there is any pixel no-null in the first/last row/column or if it is all black. Then cleaned training dataset has 613 elements, while the original one has 700 elements and the execution time is about 24s. After that, I resized the images to a same smaller size in order to reduce the cost of time by using resize function from scikit-image. So, this cleaned and resized dataset is the input of the feature extraction.

The main idea is to write a function which can implement the ABCD rule and extract these features. So, the input of the function is a `pd.DataFrame` to be filled by the output features and 3-d numpy array of each RGB image.

- i. Color variation (8 features: min_hue, max_hue, mean_hue, std_hue, min_int, max_int, mean_int, std_int)*

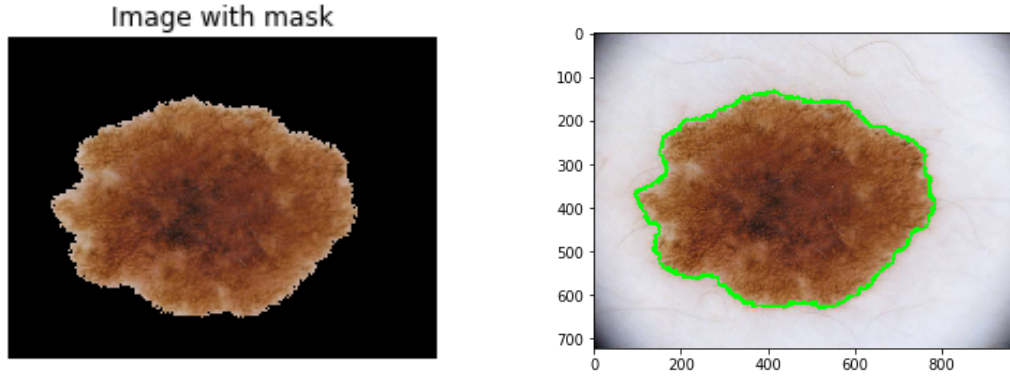
The emergence of color variation in the color is early sign of melanoma. The descriptors of color are mainly statistical parameters calculated from different color channels, like average value and standard deviation of the RGB or HSV color channel. Here color variance of the RGB image has been calculated using HSV channel.

OpenCV usually captures images and videos in 8-bit, unsigned integer, BGR format. In other words, captured images can be considered as 3 matrices, BLUE, RED and GREEN with integer values ranges from 0 to 255. But HSV color space is the better color space for color based image segmentation and object detection. So, firstly, I converted the color space of original image from RGB to HSV image.

HSV color space is consists of 3 matrices, 'Hue', 'Saturation' and 'Value'. 'Hue' represents the color, 'Saturation' represents the amount to which that respective color is mixed with white and 'Value' represents the amount to which that respective color is mixed with black. So, I want to calculate the min, max, average and standard deviation of the normalized 'Hue' and 'Value' matrices of the input images.

- ii. Geometry (5 features: n_contours, contour_area, centroid_x, centroid_y, perimeter)*

Here I study on the contours(border), area, centroid and perimeter features. For the contours, I used the function `findContours` of OpenCV. For my version of OpenCV, there are two arguments in `cv2.findContours()` function, first one is contour retrieval mode, second is contour approximation method. And it outputs the contours and



hierarchy. “contours” is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object. Firstly, I use `cv2.cvtColor` to convert BGR image of `image_mul_mask` (like showed above in the left) to GRAY `gray_image_mask`. Then, I use `cv2.threshold` to define the threshold that I will use in `cv2.findContours`. And with `cv2.drawContours`, we can get the green contour shown in the figure above in the right. Then I use `np.concatenate` to concatenate all the contours in one array called ‘contour’. And `len(contours)` will give the number of contours.

To calculate the area of the lesion location, I used the function `cv2.contourArea(contour)`. To find the centroid of the image, I used moments in OpenCV. The function `cv2.moments()` gives a dictionary of all moment values calculated. From this moments, we can extract useful data like area, centroid etc. Centroid is given by the relations, $C_x = \frac{M_{10}}{M_{00}}$ and $C_y = \frac{M_{01}}{M_{00}}$. The Contour Perimeter is also called arc length. It can be found out using `cv2.arcLength()` function. Second argument specify whether shape is a closed contour (if passed True), or just a curve.

iii. Diameter (2 features: max_diameter,min_diameter)

The first thing to do is to fit a rectangle to the contour by `cv2.minAreaRect(contour)`. The bounding rotated rectangle is drawn with minimum area. This rotated rectangle contains following details: (center (x,y), (width, height), angle of rotation). So, `max_diameter = max(w,h)` and `min_diameter = min(w,h)`.

iv. Asymmetry of shape (2 features: horizontal_asymmetry, vertical_asymmetry)

Firstly, I calculate an affine matrix of 2D rotation to make the rectangle horizontal to the axis of X. I use the function `rot = cv2.getRotationMatrix2D((x, y),`

angle, 1) where (x,y) is the return of the function cv2.minAreaRect. And it returns an affine transformation matrix 'rot' expressed as:

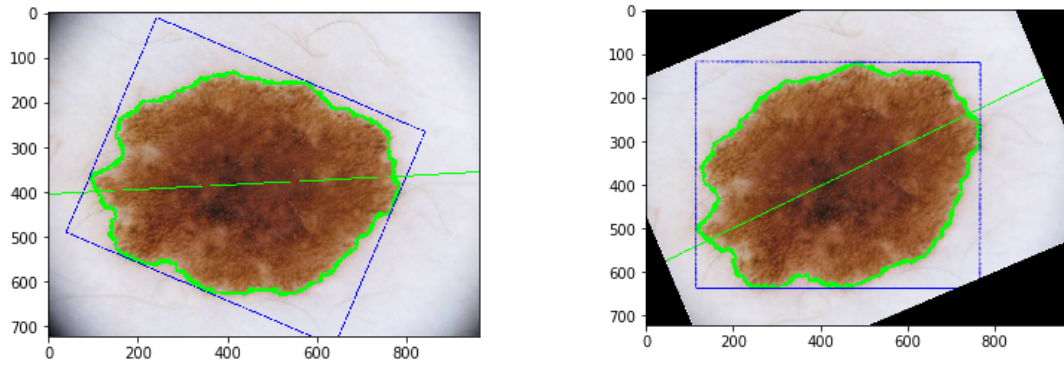
$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot \text{center.x} - \beta \cdot \text{center.y} \\ -\beta & \alpha & \beta \cdot \text{center.x} + (1 - \alpha) \cdot \text{center.y} \end{bmatrix}$$

where

$$\begin{aligned} \alpha &= \text{scale} \cdot \cos \text{angle}, \\ \beta &= \text{scale} \cdot \sin \text{angle} \end{aligned}$$

Then I use the function cv2.warpAffine(image, rot, (cols,rows)) to applies the precedent affine transformation 'rot' to an image.

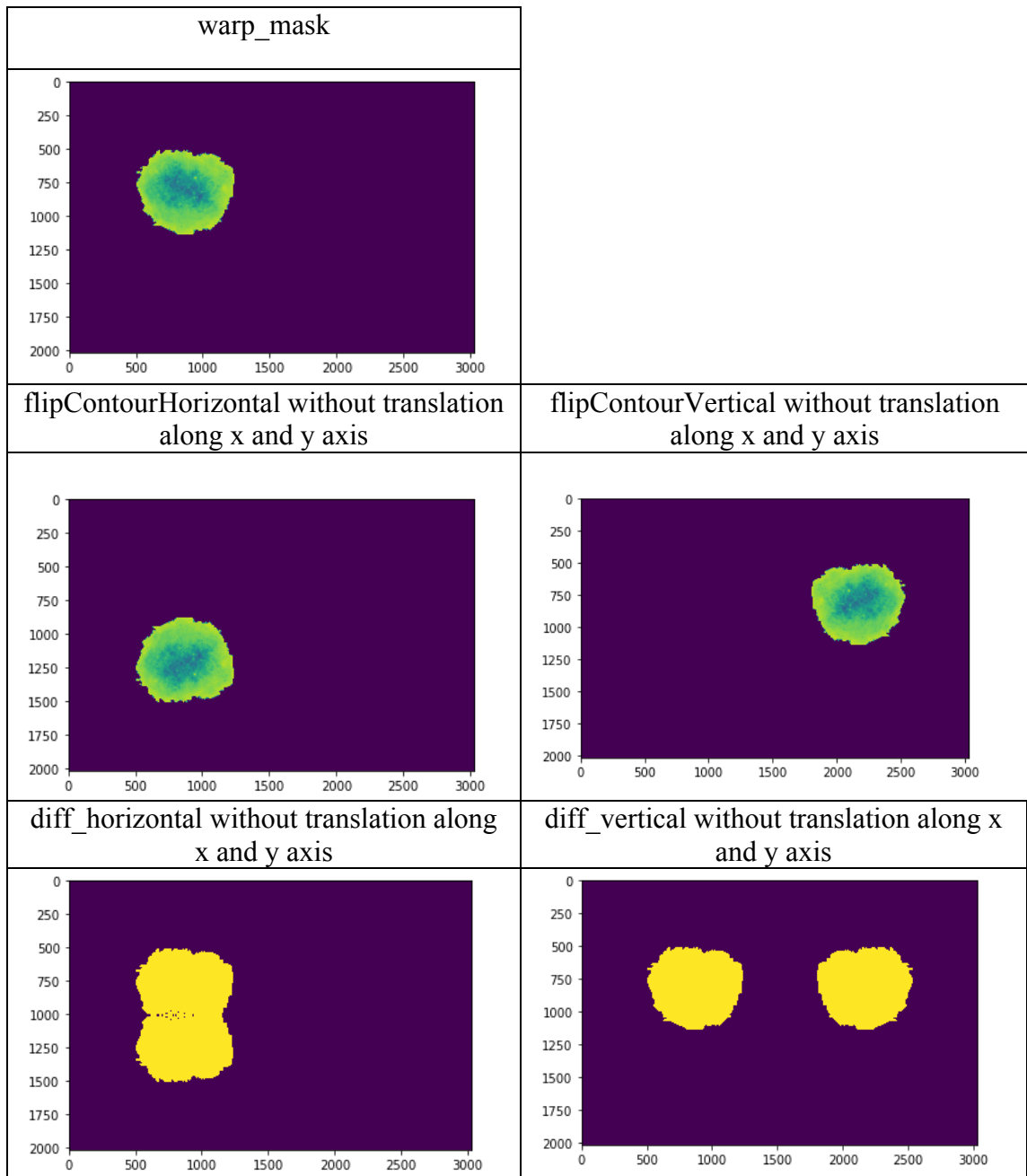
The above two figures show this transformation of rotaion.



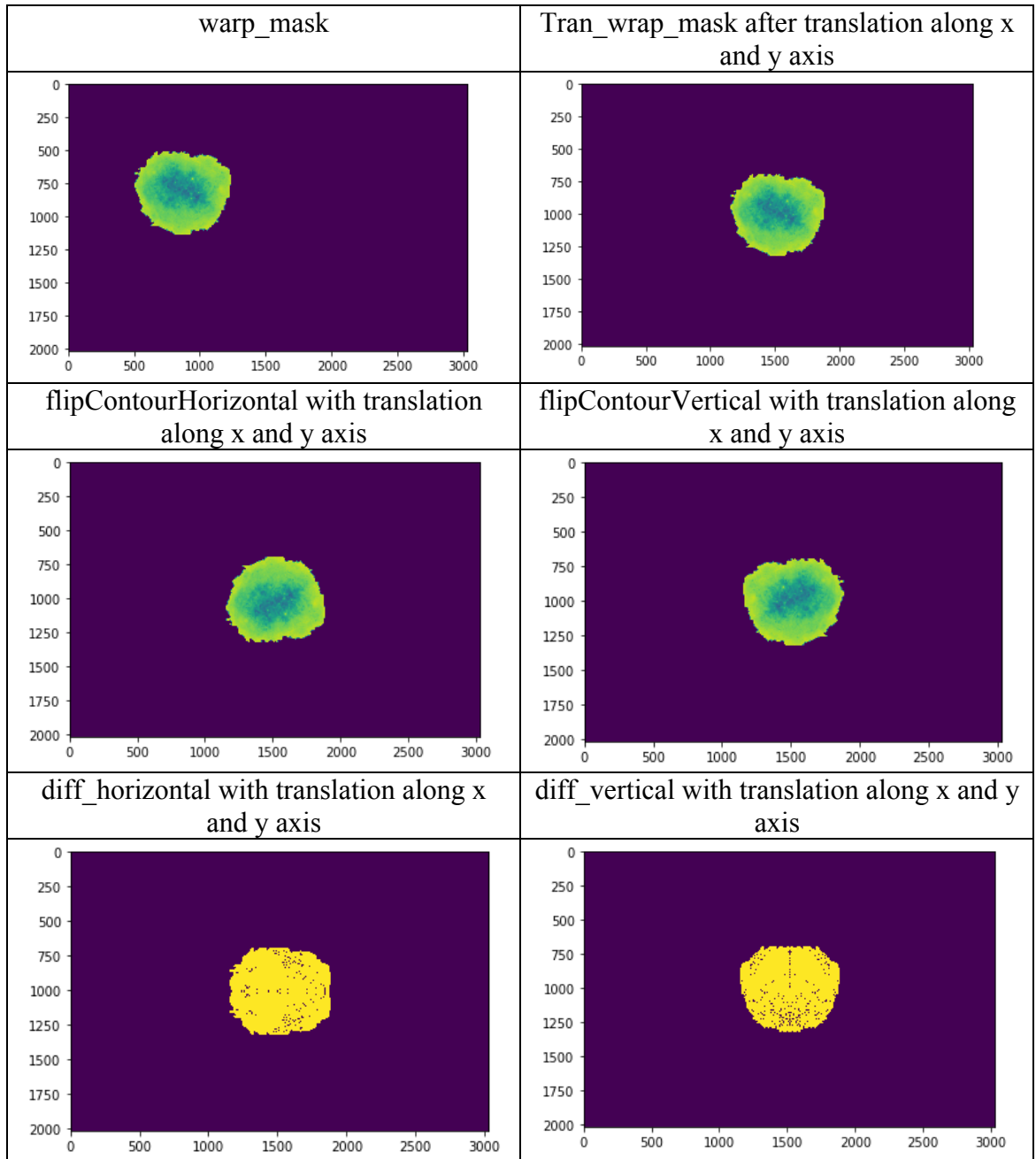
Next, I translate the current warp_mask so that the centroid of lesion mask moves to the center of the image (the half of the image shape). This will assure the estimation of asymmetry by flip (mirroring). To do that, I use the function tran_warp_mask = cv2.warpAffine(warp_mask, H, (cols,rows)) where $H = \text{np.float32}([[1,0,\text{cols}/2-\text{x}], [0,1,\text{rows}/2-\text{y}]])$ and rows, cols = warp_mask.shape.

Then, I use the function cv2.flip to flip the image around its vertical/horizontal axis and the function cv2.compare to compare the difference between the flipped warp mask and the original warp mask. And I use cv2.bitwise_not and cv2.countNonZero to get the number of pixels non-zero which means the pixels where there is a difference. Lastly, I get the parameter of asymmetry by calculating the ratio of the number of different pixels before and after mirroring in the lesion mask to the area of the lesion mask.

To see the reason why we do this translation along x and y axis, there are some tables of illustration:



These differences are not all real geometric asymmetry of the lesion mask; most of them are caused by the fact that the mask is not in the center of flipping (mirroring).



Now, we improve the estimation of asymmetry of the wrap_mask.

So, in total, I extracted 17 features of each training dataset image.

The return of the function feature_extraction2 is a list of all the features along with [min_hue, max_hue, mean_hue, std_hue, min_int, max_int, mean_int, std_int, n_contours, contour_area, centroid_x, centroid_y, perimeter, max_diameter, min_diameter, horizontal_asymmetry, vertical_asymmetry].

III. Classification

i. Pre-processing

Firstly, I shuffle the data randomly. And it is important to scale the data such that each feature has, for instance, average equal to 0 and unit variance. The features have values very far apart from each other. For example, the perimeter can go up to more than 1000000 while asymmetric features of shapes are no more than 10. So, I center and normalize the data using the StandardScaler function. Scaling is important, especially in the case of SVM, to prevent one feature from becoming more important than others.

I tested several classifiers: LDA, QDA, KNN, SVM, Random Forest and Bagging based on Decision Tree. To evaluate their accuracy, I proceeded by cross validation. It was also by cross validation that I chose the hyper-parameters of the classifiers. I used the GridSearchCV function of Scikit-learn to do that.

Since at the beginning, I shuffle the data, the results are different each time we restart the pre-processing of data.

ii. LDA & QDA

Generally, QDA is a better option for large data sets, as it tends to have a lower bias and a higher variance. On the other hand, LDA is more suitable for smaller data sets, and it has a higher bias, and a lower variance. And for the QDA and the LDA, I did not give parameter to settle because they would have an influence too low on the score.

I get the results like:

Fitting LDA done in 0.141s Average and std CV score: 0.5806356245380635 +- 0.021938150752981563
Fitting QDA done in 0.016s Average and std CV score: 0.5479809178257071 +- 0.039009278653188345

I find that it's logic that score of LDA is slightly higher than QDA, but anyway, these simple models are not enough.

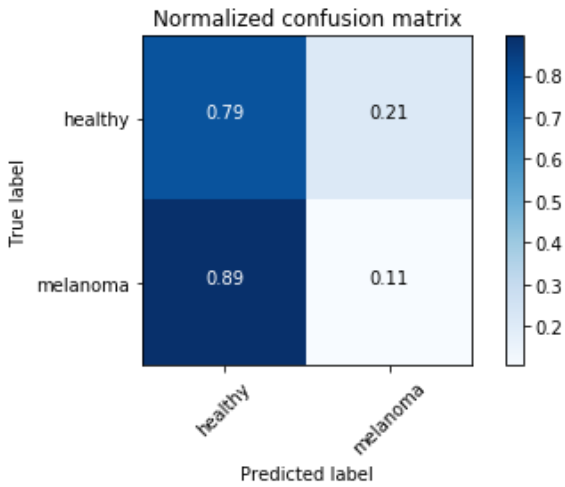
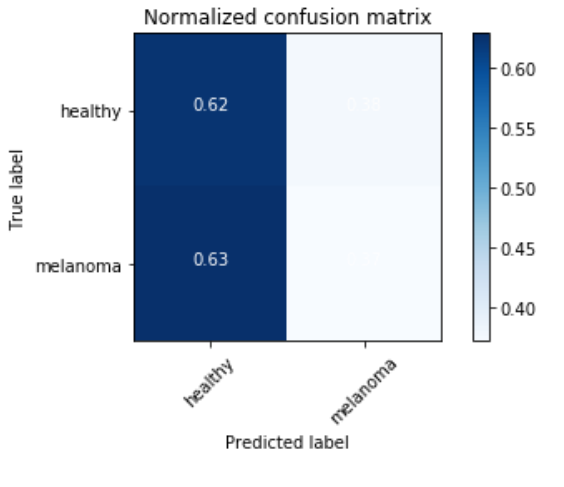
iii. KNN

We know that KNN can have a Bias-variance trade-off to improve the problem that LDA has (higher bias and lower variance). When the number of neighbor k

increases, the variance decreases but the bias is likely to increase. I want now to test the discriminative power of the features. I divide the entire dataset into training and test set and draw the confusion matrices.

I use the GridSearchCV method to find the best hyper-parameter for KNN, which is the number of neighbours. It is estimated by using one cross-validation.

And I also observe that the numbers of healthy and melanoma skin lesion are respectively: 388 and 233. There is a relative unbalancing issue in this dataset. I found that the accuracy is higher in the majority class (healthy); but lower in the minority class (melanoma). So, the core idea of random oversampling is to create a balanced data set by sampling and then train a perfect classifier.

<p>Fitting KNN on original data Best training Score: 0.5634146341463414 Best training params: {'n_neighbors': 6} Normalized confusion matrix [[0.78740157 0.21259843] [0.89473684 0.10526316]]</p>	<p>Fitting KNN on oversampled data Best training Score: 0.6338912133891214 Best training params: {'n_neighbors': 1} Normalized confusion matrix [[0.624 0.376] [0.62820513 0.37179487]]</p>
<p>Normalized confusion matrix</p> 	<p>Normalized confusion matrix</p> 

After oversampling, it seems to be better. Let's use more advanced techniques such as SVM and Random Forest.

iv. SVM

I use the GridSearchCV method to find the best hyper-parameter for SVM. For the SVM, the parameters to set are the kernel, the penalization coefficient, and in the case of the Gaussian kernel, the gamma coefficient.

Fitting Linear SVM on original data	Fitting Linear SVM over-sampled data
Best Score: 0.583 Best params: {'C': 0.001, 'gamma': 0.1, 'kernel': 'rbf'}	Best Score: 0.701 Best params: {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}

By a SVM with a Gaussian kernel, C=1000 and gamma= 0.1, I get the best score so far.

v. *Random Forest*

Now, I move to the Ensemble methods such as Random forests. This classifier is faster than SVM and Bagging. It's parallelizable and appropriate for a large number of features. If the size of the trees is too large, it will cause overfitting. And the importance of feature can be measured.

I use the GridSearchCV method to find the best hyper-parameter for Random Forest, which are the maximum depth of the trees (max_depth) and the number of trees (n_estimators).

Fitting Random Forest on original data	Fitting Random Forest on over-sampled data
Best Score: 0.573170731707317 Best params: {'max_depth': 15, 'n_estimators': 10}	Best Score: 0.6757322175732218 Best params: {'max_depth': 20, 'n_estimators': 30}

Random Forest is less suited to this problem because we don't have a large dataset.

vi. *Bagging based on Decision tree*

Bagging works for unstable predictors like trees. Variance is reduced but the bias can increase a bit.

I use the GridSearchCV method to find the best hyper-parameter for Bagging based on decision tree. For decision trees, the parameters to set that I chose are 'min_samples_leaf' and 'min_samples_split'. And for Bagging, I choose to set n_estimators.

Bagging based on decision tree on original data:	Bagging based on decision tree on oversampled data:
Best Score: 0.5707317073170731 Best params: {'n_estimators': 2000}	Best params: {'min_samples_leaf': 2, 'min_samples_split': 2} Best Score: 0.7380952380952381 Best params: {'n_estimators': 500}

As we can observe from the tables, Bagging based on decision tree works better than Random Forest. But Bagging is much slower than Random Forest.

The final choice of classifiers for these features is therefore SVM with a Gaussian kernel, $C = 1000$ and $\gamma = 0.1$ and Bagging based on trees with 500 decision tree estimators. After the feature extraction of test dataset and the centering and scaling of data, their prediction score calculated by the coefficient of Matthews is around 12, which is not satisfying.

IV. Conclusion

I think the key point and also the main difficulty of this challenge is the feature extraction and the feature selection. The level which Feature Engineering can be achieved depends on the level of understanding of the data and this particular problem domain. I believe that I still have a great work to do in order to improve my extraction of features. In general, we should generate as many features as possible, and believe that model can pick the most useful features. But sometimes doing the Feature Selection first can bring some benefits. For example, the fewer the features, the faster the training. And there may be a linear relationship between some features that affects the performance of the model we select.

When training, we mainly hope to get a model of good performance by adjusting the parameters. For example, for sklearn's RandomForestClassifier, the most important one is the number of trees in the random forest, `n_estimators`, and the maximum number of features `max_features` selected when training each tree. So, we need to have a good understanding of the model we use and how the performance of each parameter affects performance. And in the end, except the models I have fitted, it's also interesting and useful to also try the training with Gradient Boost and CNN.

References:

- [1] Maciel Zortea, Thomas R. Schopf, Kevin Thon, Marc Geilhufe, Kristian Hindberg, Herbert Kirchesch, Kajsa Møllersen, Jörn Schulz, Stein Olav Skrøvseth, Fred Godtliebsen. *Performance of a dermoscopy-based computer vision system for the diagnosis of pigmented skin lesions compared with visual evaluation by experienced dermatologists*. 2014.
- [2] Reinhard Röhner Ernst Wildling, Michael Binder, Harald Ganster, Axel Pinz and Harald Kittler. *Automated melanoma recognition*. 2001.
- [3] https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
- [4] https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html