Intro: skin condition analysis and diagnosis falls under the study of Dioscopy. This currently is manualy done by human experts. I wanted to see with this project if I could do extraction upon images of skin diseases and determine if the image extraction features melanoma.

For this objected I referenced and took some operations from three different studies. The first study was one done by Zafar K. etal. This study features a way to perform some preprocessing upon images of skin diseases to effectively remove hair strands as well as a small amount of noise removal. I used this main method for the preprocessing of the images that I will mention in the methods section.[1] For the actual extraction I wanted to see if there was a paper that features a simple yet effective method of skin lesion extraction. For this I referenced a study done by Garnavi R et.al. This study focused on how accurate different color channels could be for performing feature extraction and edge detection upon different skin lesions and diseases. I followed this study's method of extraction when performing the main method of feature extraction. I was unable to follow every possible step they did but will once again go more in depth during the methods section.[2] Lastly I needed a study that had referenceable features and traits of different skin lesions I could reference. For this I referenced a journal paper about Dermoscopy done by Soyer et.al. This study categorized different features and patterns that take place in different skin diseases and lesions. With these tables I had to limit down the actual symptoms and features chosen to be those specifically for melanoma.[3]

For the actual implementation of the process it was broken up into three main stages, preprocessing, lesion extraction, and identification and diagnosis. During the preprocessing stage I used Blackhatting to be able to extract the hairs present within the image. This extraction would allow me to form a binary mask which I could then use the perform inpainting.[1] Most of this process was done with opencv library functions as since they were already implemented but I found a kernel size of 14 worked best for the blackhatting operation. The inpainting was also done with the base inpainting functionality in opencv with an area of 40. This was the range it would look to find the colors to fill in the space with. A problem with this method though was the time, as well as the slight noise and artifacts that would appear after performing the inpainting. To fix this I also applied a low pass gaussian blur filter to the inpainted image. This would soften the image removing the artifact and noise. For the feature extraction I followed the same procedure done by the Garnavi study which was am implementation of Otsu thresholding.[2] A problem I ran into though was that the otsu thresholding function present in opencv only worked on grayscale images. This made me realize that the paper must have used a custom implementation to allow for the XYR color spectrum, which is the XY colors plus Red from the RGB color space. Looking back at the study I noticed that there was only a 2% difference between the XYR color space and a once channel color space. This 2% change didn't seem monumental enough to put in the same full custom implementation so I just stuck with the base extraction while finding a combination of otsu and truncated thresholding, finished off with binary thresholding to form a binary mask.[2] This final mask had a problem of having a large amount of space within it so to be able to fill in this space I used the floodfill function to attempt

to do a full fill of the final mask. This flood fill worked rather well but there were still some artifacts within the mask. Originally I tried to use the same edge/corner finding that I did with assignment 2 but that led to a large amount of problems with stray pixels within the images with a width greater than 1000. To find a new way to solve this I found that I could instead remove all connected pixels from the edges of the mask, and then do a distance removal from those edges as the actual main extraction was in the middle of the image. Although this was a very quick and dirty method it worked shockingly well while having very little impact on the final measurements. I would then take this mask and apply it to the inpainted image to get the final extraction which I then compared to a manually made ground truth to get the accuracy measurements provided.[2] For the final diagnosis I went through the table for the melanoma features within the paper done by Soyer et.al. A problem I instantly came across was that I would not be able to detect a chunk of the symptoms and features using a solely image processing base method which is what I was planning on doing. Some of these features I was unable to achieve simply because they were subjective measurements that would be identifiable by humans but not a machine. An example of this could be the Common pattern of unspecific. This described the main shape of a melanoma lesion or tumor being of an anomalous shape or form. This could be achieved with an identification model along with machine learning but that would still be fairly difficult as how do you teach the search for nothing? A different type of pattern that I was unable to use or search for was that of specific shape patterns. These were not used because they were impossible but just because of my sole image processing method without use of any machine learning additions. An example of this would be the Globular, or cobblestone patterns. These describe the detection or identification of round glob-like patterns within the lesion/tumor, or the cobblestone pattern. Although there is a circle identification method they generally measure items that actually resemble an actual circle shape. Seeing as how when taken down to the pixel and measuring what to us may look like a circle would instead have more ease measuring out as a rectangle makes it difficult to search for some of these patterns. Patterns I was able to search for were that of patterns specified around difference or “Nonsubjective patterns”. An example I could say this as is what was searched for in the parallel-ridge pattern and reticular pattern. Parallel-ridges are the ridges between different segments of the legion. These ridges are mainly identified by the slight difference in pigmentation in them. This is similar to how you can tell the difference between a canyon's top and bottom by a color change. This identification of parallel ridges I feel also coincides with reticular patterns which are defined by a fine network like or netlike surface. As net patterns are normally seen by their distinction between a background this is a difference that I am capable of measuring. Using these I was able to measure for these styles of distinct measurements. I took out the measures that overlapped, such as the reticular overlapping in the same detection format at parallel ridges, and the large amount of specified black and brown spots that also falls under the general pigmentation of either black or brown. After all of this I ended with the main measurements of checking for the amount of pixels within the legion that were of the color black or brown, the amount of clusters or segments of the extraction, and finally the amount of ridges that were presents within the segmentation. To

measure the colors was easy as I just had to get the distance of each pixel from a black and dark brown, and see the percentage of them that were of this color. I determined that a good measurement would be that in these extraction at least 40 percent of the pixels should be either black or brown. I chose this to account for the amount of pixels that were taken up by camera flash along with blue pigments also taking up a majority of the other part of the segmentation, but the black and brown should be the dominant colors present within the segmentations. The second of ridges is measured by taking the difference between the colors of pixels two pixels away from each other within the legion. I decided on this as since the images input as data were fairly large I was not getting distinction between pixels of vastly different colors because of the blurring done in preprocessing. As this also cut down my search space by half I found even if this smaller search space took even a slightly large segment of the legion it should be able to show that parallel ridges were present within the legion. For the comparison value I found that 15% of the legion space was good as it allowed for a comparison against the entire feature space that showed the appearance of legions while accounting for the smaller feature space, requiring a very large amount of differences to be present to qualify that parallel ridges were present in the legion. Lastly I looked at the clustering, as melanoma lesions are generally made up of large components with very few if any small satellite like components. To measure this while taking into account even possible post mask artifacts I set the cluster limit to a hard 4. As this would account for any bizarre fills while making it a definite cutoff for any other item that may have multi component appearances such as acne.[3] Of these patterns that are detected the program only gives a confident Melanoma detection on three out of three of these patterns being detected if not it will give only a Most likely signal if it is two out of three, and if only one pattern is found it is output as confidently not Melanoma.

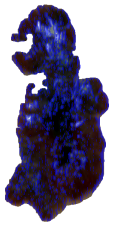
Results:

For the measurements I had two comparisons per image. I compared first against the mask that did the extraction and the actual extraction. I just wanted to make sure that there will not be any weird results or variations when doing the results.

Photo order: original Image, Final Inpainted image, Converted image, equalized image, mask, Flood, Ground truth, Extracted image, returned results including diagnosis



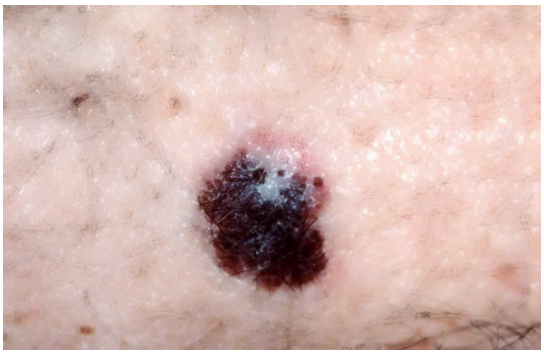


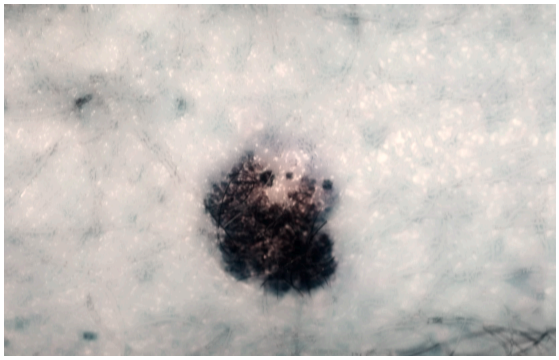



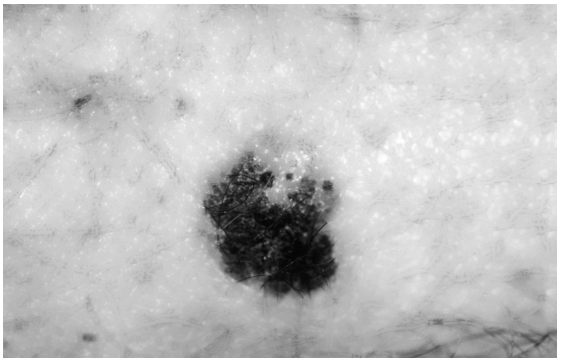
```
---------Comparison against extract----------
TN: 1.66729e+06 TP: 99501 FP: 3398 FN: 7567
Sensitivity: 92.9325
Specificity: 99.5482
Accuracy: 99.3832
Similarity: 94.7777
---------Comparison against MASK2----------
TN: 1.66729e+06 TP: 99501 FP: 3398 FN: 7567Sensitivity: 92.9325
Specificity: 99.5482
Accuracy: 99.3832
Similarity: 94.7777
```

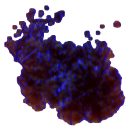
```
Area of Legion:51445
clusters: 1
18729
precentage that are ridges: 36.4059
There is parallel-ridges pressent
more than 40% of legions are blown or black
made of few clusters
 The extracted leagion fulfills 3/3 of the detectable melanoma characteristics
The extracted legion is CONFIDENTLY melanoma
```

Image 2





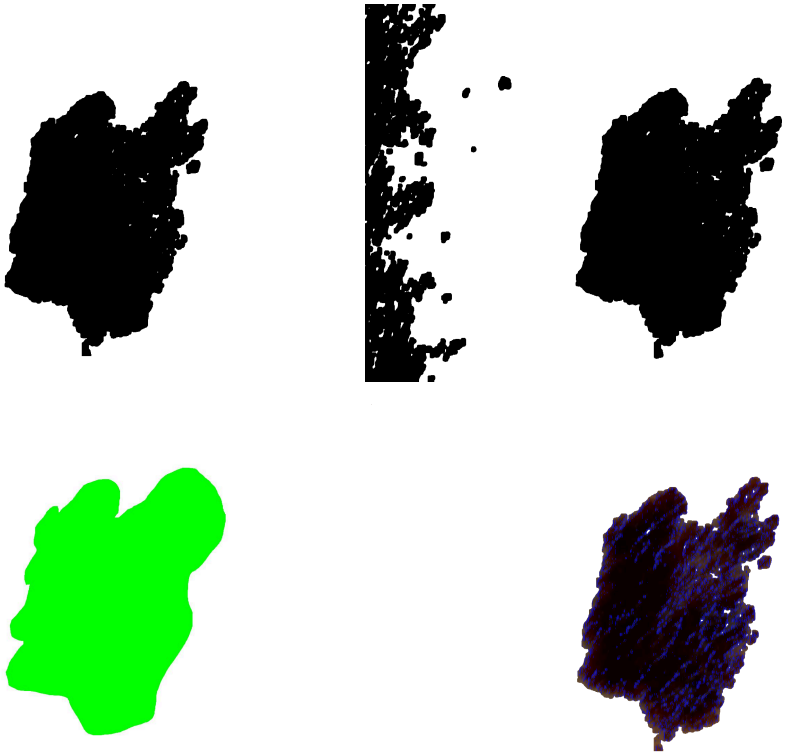




```
---------Comparison against extract----------
TN: 1.58314e+06 TP: 85780 FP: 1564 FN: 48241
Sensitivity: 64.0049
Specificity: 97.0429
Accuracy: 97.1022
Similarity: 77.501
---------Comparison against MASK2----------
TN: 1.58314e+06 TP: 85780 FP: 1564 FN: 48241Sensitivity: 64.0049
Specificity: 97.0429
Accuracy: 97.1022
Similarity: 77.501
```

```
Area of Legion:43700
clusters: 8
10787
precentage that are ridges: 24.6842
There is parallel-ridges pressent
more than 40% of legions are blown or black
 The extracted leagion fulfills 2/3 of the detectable melanoma characteristics
The extracted legion is MOST LIKELY melanoma
```

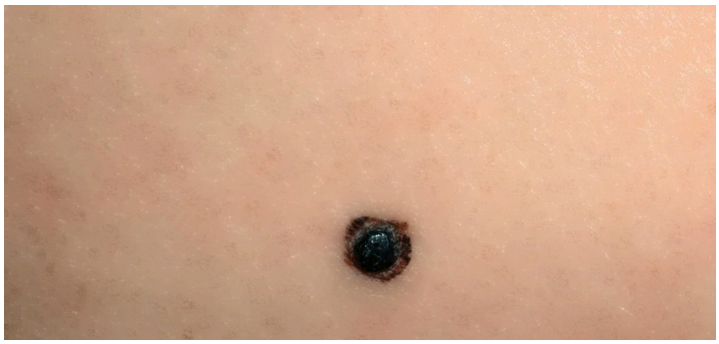
Image 3

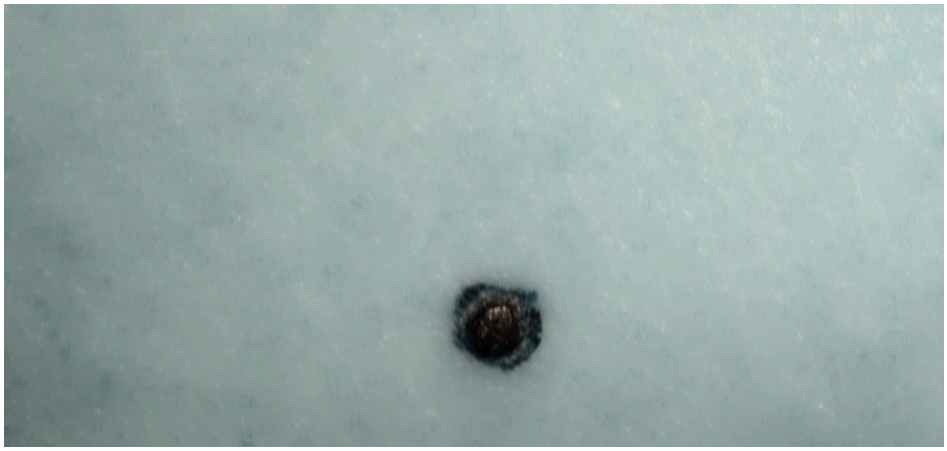


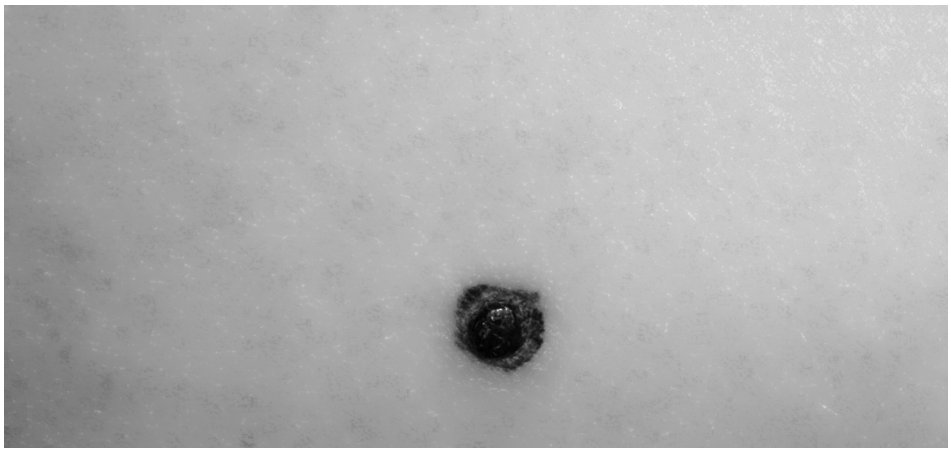
```
---------Comparison against extract----------
TN: 1.47481e+06 TP: 285953 FP: 6318 FN: 31997
Sensitivity: 89.9365
Specificity: 97.8765
Accuracy: 97.8703
Similarity: 93.7211
---------Comparison against MASK2----------
TN: 1.47481e+06 TP: 285953 FP: 6318 FN: 31997Sensitivity: 89.9365
Specificity: 97.8765
Accuracy: 97.8703
Similarity: 93.7211
```

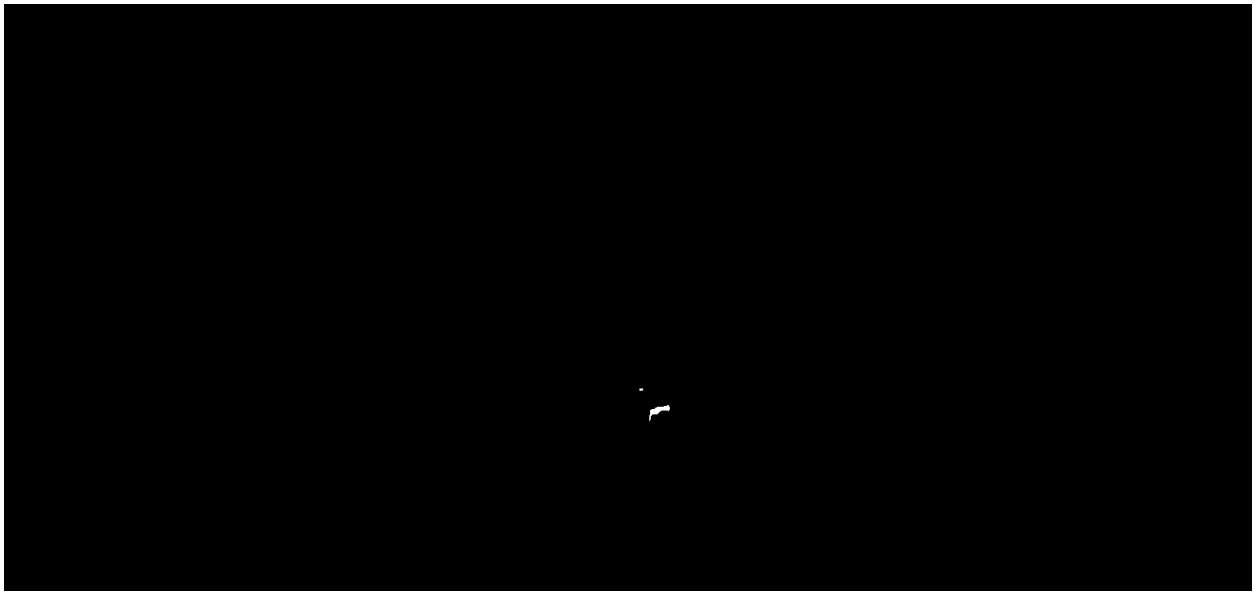
```
Area of Legion:146094
clusters: 3
29915
precentage that are ridges: 20.4765
There is parallel-ridges pressent
more than 40% of legions are blown or black
made of few clusters
 The extracted leagion fulfills 3/3 of the detectable melanoma characteristics
The extracted legion is CONFIDENTLY melanoma
```

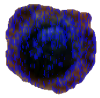
Image 4







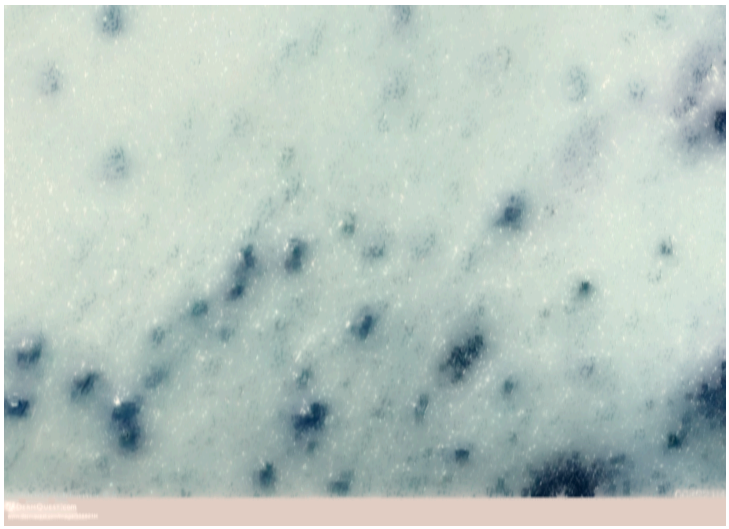


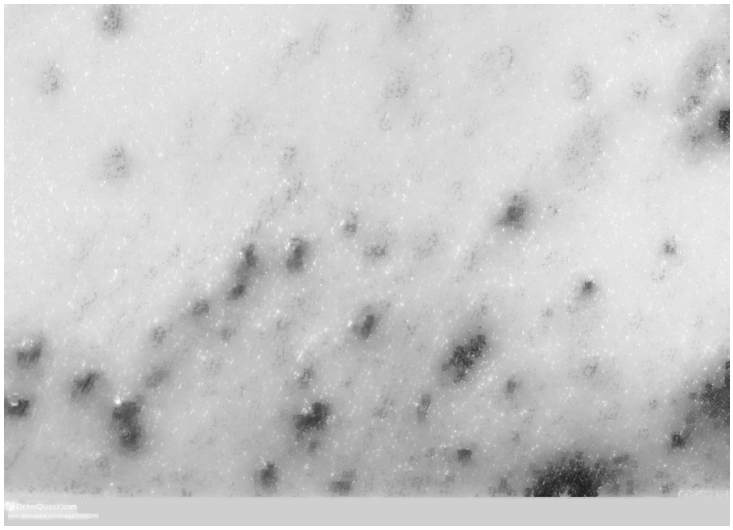


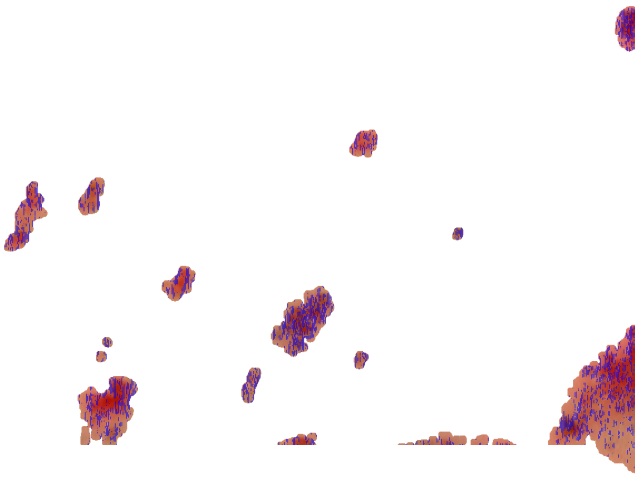
```
---------Comparison against extract----------
TN: 1.24422e+06 TP: 18699 FP: 725 FN: 792
Sensitivity: 95.9366
Specificity: 99.9364
Accuracy: 99.88
Similarity: 96.1018
---------Comparison against MASK2----------
TN: 1.24422e+06 TP: 18699 FP: 725 FN: 792Sensitivity: 95.9366
Specificity: 99.9364
Accuracy: 99.88
Similarity: 96.1018
771 1640 771 1640
```

```
Area of Legion:9702
clusters: 1
4162
precentage that are ridges: 42.8984
There is parallel-ridges pressent
more than 40% of legions are blown or black
made of few clusters
 The extracted leagion fulfills 3/3 of the detectable melanoma characteristics
The extracted legion is CONFIDENTLY melanoma
```

Image 5 Acne:

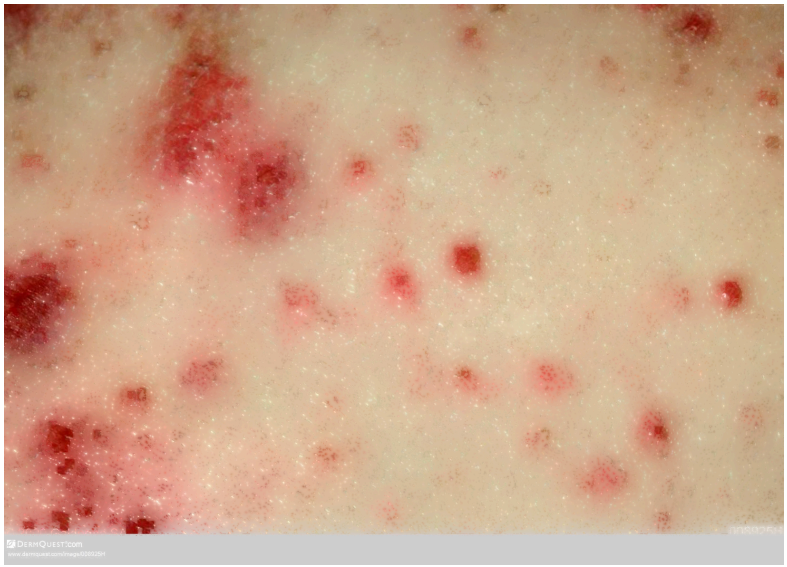


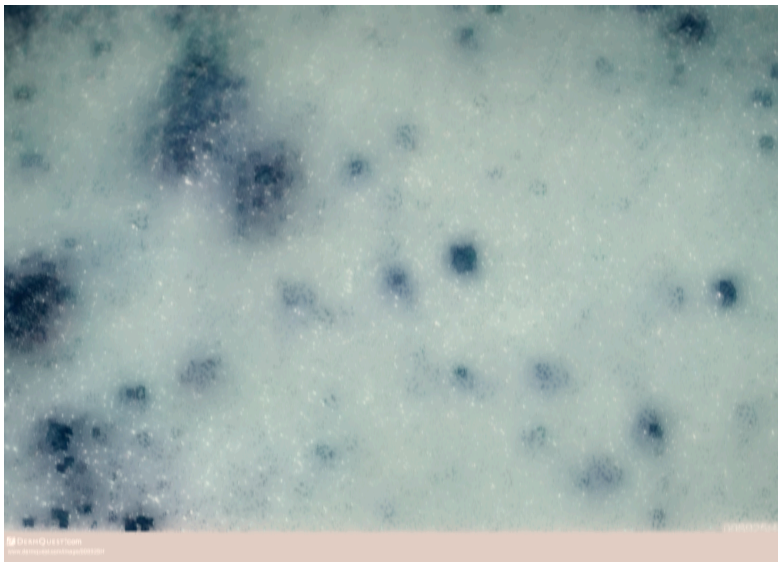


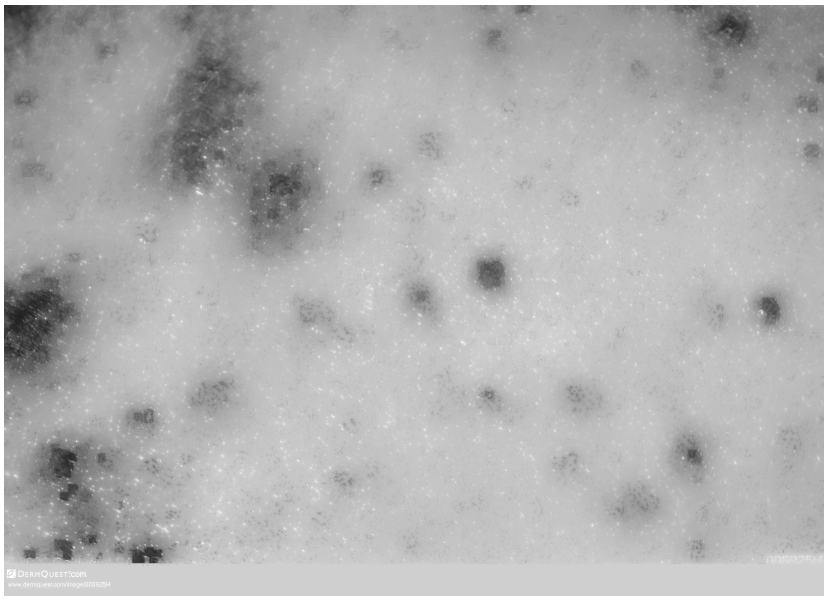


```
Area of Legion:25093
clusters: 17
5668
precentage that are ridges: 22.588
There is parallel-ridges pressent
 The extracted leagion fulfills 1/3 of the detectable melanoma characteristics
The extracted legion is CONFIDENTLY not melanoma
```

Image 6 Acne 2






DERMQUEST.com

```
Area of Legion:30369
clusters: 17
9821
precentage that are ridges: 32.3389
There is parallel-ridges pressent
 The extracted leagion fulfills 1/3 of the detectable melanoma characteristics
The extracted legion is CONFIDENTLY not melanoma
```

Discussion:

For how simple of a method my methods of extraction and diagnosis were I was actually surprised with how accurate the results were and how effective the extraction process was. I first noted with the system that there was a main distinction between what images worked well with the system. Images that worked best were images that featured a close up image of the legion with as much of nothing else besides the skin. The other is just the time it takes to operate the inpainting and noise removal that is very strenuous and time consuming. For the actual extraction through the system worked extremely well and fast. With a majority of the extractions getting scores in accuracy of the scores sitting mainly in the 96 to 99 range this falls right in line with the range that the Garnavi paper had their single channel color space extraction accuracy fall in.[2] The diagnosis works also surprisingly well as I wanted to put in specifically acne because I feel that is has the change to muddy the works with the clusters, and possibility of being preprocessed poorly in a way to actually make the system think that it was a singular blob, but luckily my system made sure to avoid this and was able to diagnose accurately. If given more time just like with the diagnosis features I would like to include many more skin diseases so that instead of deciding if it is just melanoma I can give a percentage readout of what disease a legion most likely falls under.

Conclusion:

With my system showing good results that are in line with the referenced studies, as well as having potential room for improvements I feel that i have shown the idea that image processing techniques could be used to allow for not human dermoscopy, and allow for a lot more potential when it comes to identifying and diagnosing skin diseases.

References:

[1] Zafar, K., Gilani, S. O., Waris, A., Ahmed, A., Jamil, M., Khan, M. N., & Sohail Kashif, A. (2020). Skin lesion segmentation from dermoscopic images using convolutional neural network. Sensors, 20(6), 1601. https://doi.org/10.3390/s20061601

[2] Garnavi, R., Aldeen, M., Celebi, M. E., Varigos, G., & Finch, S. (2011). Border detection in dermoscopy images using hybrid thresholding on optimized color channels. Computerized Medical Imaging and Graphics, 35(2), 105–115. https://doi.org/10.1016/j.compmedimag.2010.08.001

[3] Soyer, Peter & Argenziano, Giuseppe & Ruocco, V & Chimenti, Sergio. (2001). Dermoscopy of Pigmented Skin Lesions. European journal of dermatology : EJD. 11. 483-98.

Source Code:

```
#include <opencv2/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/video.hpp>
#include <opencv2/highgui/highgui_c.h>
//#include <opencv2/xphoto/inpainting.hpp>
#include <iostream>
#include <deque>

using namespace cv;
using namespace std;

int main() {

	//Preprecessing
	cv::Mat img = cv::imread("../OpenCVImages/Inpaints/InpaintAC0.png",
IMREAD_GRAYSCALE);
	cv::Mat origimg = cv::imread("../OpenCVImages/Inpaints/InpaintAC0.png");
	cv::Mat GT = cv::imread("../OpenCVImages/Inpaints/GGT6.jpg",
IMREAD_GRAYSCALE);; //Ground truth

	//Method for reading a bunch of images; Doing just a large chunck of inpainting and
black hatting so I can just run it on all the images
		//Creating BlackHatFilter
		cv::cvtColor(img, img, cv::COLOR_RGB2GRAY);
		Point filtersize = Point(14, 14);
		Mat kernal = getStructuringElement(MORPH_RECT, filtersize);
		vector<cv::String> fn;
		glob("../OpenCVImages/Upload_20230214-235509~/Acne/*.jpg", fn, false);

		vector<Mat> images;
		size_t count = fn.size(); //number of png files in images folder
		for (size_t i = 0; i < count; i++)
			images.push_back(imread(fn[i]));
```

```
		int i = 0;
		for (Mat img : images) { //227
			//Mat eq;
			Mat origimg = img;
			cvtColor(img, img, COLOR_BGR2GRAY);
			equalizeHist(img, img);
			Mat blackhatimg, mask;
			//Run blackhat filter
			cv::morphologyEx(img, blackhatimg, MORPH_BLACKHAT, kernal);
			//create mask by binary threshholding using the blackhat
			cv::threshold(blackhatimg, mask, 12, 255, THRESH_BINARY);
			std::cout << "Black Hatting done" << endl;
			//namedWindow("Blackhat", WINDOW_AUTOSIZE);
			/*cv::imshow("Blackhat", mask);
			cv::moveWindow("Blackhat", 0, 90);
			cv::waitKey(0);*/
			/*for (int x = 0; x < img.cols; x++)
			{
				for (int y = 0; y < img.rows; y++)
				{
					if (img.at<Vec3f>(y, x) > 180)
					{
						img.at<Vec3f>(y, x);
					}
				}
			}*/
			waitKey(0);
			std::cout << "Doing inpainting" << endl;
			Mat inpaint;
			cv::inpaint(origimg, mask, inpaint, 40, INPAINT_TELEA);
			//cv::Mat origimg = cv::imread("../OpenCVImages/Inpaint.png");

			cout << "Writing" << endl;
			imwrite("../OpenCVImages/Inpaints/InpaintAC" + to_string(i) + ".png",
inpaint);
			//Mat compare;
			//imwrite("../OpenCVImages/Inpaints/Compares/Compare" + to_string(i)
+ ".png", compare);
			++i;
		}
```

```
		vector<cv::String> fn;
		glob("../OpenCVImages/Inpaints/*.png", fn, false);
		vector<Mat> images;
		size_t count = fn.size(); //number of png files in images folder
		for (size_t i = 0; i < count; i++)
			images.push_back(imread(fn[i]));
		vector<cv::String> fn2;
		glob("../OpenCVImages/Inpaints/*.jpg", fn2, false);

		vector<Mat> GTS;
		count = fn2.size(); //number of png files in images folder
		for (size_t i = 0; i < count; i++)
			GTS.push_back(imread(fn2[i]));
		int CTR = 0;
		for (Mat img : images) { //227
			Mat origimg = img.clone();
			cvtColor(img, img, cv::COLOR_BGR2GRAY);
			Mat k = getGaussianKernel(8, 8);
			filter2D(origimg, origimg, 0, k, Point(-1, -1), 0.0,
BORDER_REPLICATE);

			Mat Converted;
			//
			cvtColor(origimg, Converted, cv::COLOR_RGB2GRAY);


			//Threshold done in paper " Otsu method [21] as the
		//following:
		//	2
		//		b(t) = 2 − 2
		//		ω(t) = ω1(t)ω2(t)[1(t)− 2(t)]2 (2)
		//		where i are the mean values of the two clusters.Starting from an
		//	initial threshold value of t = 1, ωi and 2
		//		i are updated iteratively and
		//		in each iteration 2
		//		b(t) is calculated.The optimal threshold corresponds to the
maximum value of 2
		//		b(t).The output binary image has
		//		values of 1 for all pixels in the input image with luminance greater
		//		than the threshold level and 0 for the remaining pixels."
```

```
		//They used a modified otsu method. I will see if I can try and do a manual version of their method BUT
		//For now I will do the base otsu thresholding method
		Mat mask2;
		long double thres = threshold(Converted, mask2, 0, 255, THRESH_OTSU + THRESH_TRUNC);
		cout << "Otsu Threshold : " << thres << endl;
		thres = threshold(mask2, mask2, 0, 255, THRESH_BINARY + THRESH_OTSU);

		cout << "Otsu Threshold : " << thres << endl;
		//Fill hols in mask
		Point ksize = Point(14, 14);
		Mat Kern = getStructuringElement(MORPH_ELLIPSE, ksize);
		morphologyEx(mask2, mask2, MORPH_CLOSE, Kern);
		ksize = Point(4, 4);
		Mat flood = mask2.clone();
		floodFill(flood, Point(0, 0), Scalar(0));

		Mat tmp = mask2.clone();
		/*namedWindow("Mask2 preeflood", WINDOW_AUTOSIZE);
		cv::imshow("Mask2 preeflood", tmp);
		cv::moveWindow("Mask2 preeflood", 0, 90);*/

		bool equal = true;
		for (int x = 0; x < flood.cols; x++)
		{
			for (int y = 0; y < flood.rows; y++)
			{
				if (flood.at<uchar>(Point(x, y)) != mask2.at<uchar>(Point(x, y)))
				{
					cout << "NOT EQUAL AT " << Point(x, y) << endl;
					equal = false;
					break;
				}
			}
			if (equal == false) {
```

```
                        break;
                }
        }
        if (equal == false) {
                mask2 = (mask2 | flood);
        }

        if (equal == false) {
                for (int x = 0; x < flood.cols; x++)
                {
                        for (int y = 0; y < Converted.rows; y++)
                        {
                                //AT WORKED WHEN PUTTING IT IN A
EMPTY MAT UNLIKE DATA! This will confuse me till the end of time as I have no clue what i
sleepily did on tuesday
                                if (flood.at<uchar>(Point(x, y)) == 255)
                                {
                                        mask2.at<uchar>(Point(x, y)) = 0;
                                }
                        }
                }
        }
        /*namedWindow("flood", WINDOW_AUTOSIZE);
        cv::imshow("flood", flood);
        cv::moveWindow("flood", 0, 90);*/
        vector<tuple<Point, Point>> edges;
        //Fill in mask 2
        cout << "grabbing edges" << endl;
        for (int x = 0; x < mask2.rows; x++)
        {
                Point left, right = Point(-1, -1);
                //cout << "left = " << left << endl;
                for (int y = 0; y < mask2.cols; y++)
                {
                        //AT WORKED WHEN PUTTING IT IN A EMPTY MAT
UNLIKE DATA! This will confuse me till the end of time as I have no clue what i sleepily did
on tuesday
                        if (mask2.at<uchar>(Point(y, x)) == 0 && left == Point(0,
0))
                        {
```

```
                                        left = Point(y, x);
                                    }
                                    if (mask2.at<uchar>(Point(y, x)) == 255 && y - 1 >= 0 &&
mask2.at<uchar>(Point(y - 1, x)) == 0 && left != Point(0, 0)) {
                                        right = Point(y - 1, x);
                                        edges.push_back(make_tuple(left, right));
                                        //cout << "left = " << left << " Right = " << right <<
endl;
                                        left = right = Point(0, 0);
                                    }
                                }
                            }

                            int height = 0;
                            int width = 0;

                            for (auto tmp : edges)
                            {
                                width += (get<0>(tmp).y + get<1>(tmp).y) / 2;
                                height += (get<0>(tmp).x + get<1>(tmp).x) / 2;
                            }

                            height = height / edges.size();
                            width = width / edges.size();
                            cout << "h = " << height << " w= " << width << endl;
                            int x = width;
                            int y = height;
                            int hthresh, wthresh;
                            Point tmph, tmpw;
                            hthresh = wthresh = -1;
                            //while(hthresh == -1) {
                            //      //cout << "checking " << Point(mask2.cols - width, y) << endl;
                            //      if (mask2.at<uchar>(Point(mask2.cols - width, y)) == 255 &&
mask2.at<uchar>(Point(mask2.cols - width, y - 1)) == 0) {
                            //          hthresh =  y - 1;
                            //      };
                            //      ++y;
                            //}

                            cout << "edge count" << edges.size() << endl;
```

```
                        for (auto tmp : edges)
                        {

                                if (get<0>(tmp).y > hthresh) {
                                        //cout << get<0>(tmp) << " " << get<1>(tmp) << " " <<
(get<1>(tmp).x < 500) << endl;
                                }
                                if (get<1>(tmp).x >= mask2.cols - 10) {
                                        //cout << "width catch" << get<0>(tmp) << " " <<
get<1>(tmp) << " " << (get<1>(tmp).x < 500) << endl;
                                }
                                if (get<0>(tmp).x == 0 || get<1>(tmp).x < 500 || get<1>(tmp).y >
1020) {
                                        if (get<0>(tmp).y > hthresh) {
                                                //cout << "filling " << get<0>(tmp) << "-" <<
get<1>(tmp) << endl;
                                        }
                                        for (int x = get<0>(tmp).x; x <= get<1>(tmp).x; ++x) {
                                                //cout << "filled at " << Point(x, get<1>(tmp).y) <<
endl;
                                                mask2.at<uchar>(Point(x, get<1>(tmp).y)) = 255;
                                        }
                                }

                        }

                        /*for (int x = 0; x < mask2.cols; ++x) {

                                if (mask2.at<uchar>(Point(x,mask2.rows - 1)) == 0)
                                {
                                        cout << Point(x, mask2.rows - 1) << endl;
                                }
                        }*/



                        //      if (left.x == 100000000 || right.x == -100000000) {
                        //              continue;
                        //      }
                        //      else {
```

```
//                  edges.push_back(make_tuple(left, right));
//          }
//}
//for (auto t : edges) {
//          cout << get<0>(t) << " " << get<1>(t) << endl;
//}
//
////cout << "Doing edge fill" << endl;
//for (auto t : edges) {
//          Point left = get<0>(t);
//          Point right = get<1>(t);
//          int x = left.x;
//          //cout << "filling " << left << "-" << right << endl;
//          for (int y = left.y; y <= right.y; ++y) {
//                    //cout << "filling " << left << "-" << right << endl;
//                    mask2.at<uchar>(Point(y, x)) = 0;
//          }

//}


                    //So the XoYoR color space stands for " XoYoR, where “o” stands for
                    /*logical OR, which combines the X and Y color channels from the XYZ
                              color space with the R color channel from the RGB color space
                              So I need to have two different items to compare*/
          cvtColor(origimg, Converted, COLOR_BGR2XYZ);
          Mat extract = Converted.clone();
          cout << mask2.rows << " " << mask2.cols << " " << extract.rows << " " << extract.cols << endl;
          for (int x = 0; x < Converted.cols; x++)
          {
                    for (int y = 0; y < Converted.rows; y++)
                    {
                              //AT WORKED WHEN PUTTING IT IN A EMPTY MAT UNLIKE DATA! This will confuse me till the end of time as I have no clue what i sleepily did on tuesday
                              if (mask2.at<uchar>(Point(x, y)))
```

```
                        {
                                extract.at<Vec3b>(Point(x, y)) = Vec3f(255, 255,
255);
                        }
                }
        }
        //if (CTR <= 3) {
                //Calculate measurements

                float TP = 0; // TRUE Positives
                float TN = 0;
                float FP = 0; // FALSE POSITIVES
                float FN = 0;
                auto black = Vec3b(255, 255, 255);
                for (int x = 0; x < GT.cols; x++)
                {
                        for (int y = 0; y < GT.rows; y++)
                        {
                                //correct Background
                                if (GT.at<uchar>(Point(x, y)) == uchar(255))
                                {
                                        //Backgrounds line up
                                        if (extract.at<Vec3b>(Point(x, y)) == black)
                                        {
                                                ++TN;
                                        }
                                        else
                                        {
                                                //false positive
                                                ++FP;
                                        }

                                }
                                //Correct Extract
                                if (GT.at<uchar>(Point(x, y)) != uchar(255)) {
                                        if (extract.at<Vec3b>(Point(x, y)) != black)
                                        {
                                                //Correct extract
                                                ++TP;
                                        }
```

```
                                        else {
                                                //FALSE Background
                                                ++FN;
                                        }
                                }
                        }
                }

                float Sensitivity = (TP / (TP + FN)) * 100.0;
                float Specificity = (TN / (TN + FN)) * 100.0;
                float Accuracy = ((TP + TN) / (TP + FP + FN + TN)) * 100.0;
                float Similarity = ((2 * TP) / (2 * TP + FN + FP)) * 100.0;
                std::cout << int(GT.at<uchar>(Point(0, 0))) <<
extract.at<Vec3b>(Point(0, 0)) << (GT.at<uchar>(Point(0, 0)) == uchar(225)) <<
(extract.at<Vec3b>(Point(0, 0))) << endl;
                std::cout << "---------Comparison against extract----------" << endl;
                std::cout << "TN: " << TN << " TP: " << TP << " FP: " << FP <<
" FN: " << FN << endl;
                std::cout << "Sensitivity: " << Sensitivity << endl;
                std::cout << "Specificity: " << Specificity << endl;
                std::cout << "Accuracy: " << Accuracy << endl;
                std::cout << "Similarity: " << Similarity << endl;
                TP = TN = FP = FN = 0;

                for (int x = 0; x < GT.cols; x++)
                {
                        for (int y = 0; y < GT.rows; y++)
                        {
                                //correct Background
                                if (GT.at<uchar>(Point(x, y)) == uchar(255))
                                {
                                        if (mask2.at<uchar>(Point(x, y)) == 255)
                                        {
                                                ++TN;
                                        }
                                        else
                                        {
                                                //false positive
                                                ++FP;
                                        }
```

```
                                        }
                                        //Correct Extract
                                        if (GT.at<uchar>(Point(x, y)) != uchar(255)) {
                                                if (mask2.at<uchar>(Point(x, y)) != 255)
                                                {
                                                        //Correct extract
                                                        ++TP;
                                                }
                                                else {
                                                        //FALSE Background
                                                        ++FN;
                                                }
                                        }

                                }
                        }

                        Sensitivity = (TP / (TP + FN)) * 100.0;
                        Specificity = (TN / (TN + FN)) * 100.0;
                        Accuracy = ((TP + TN) / (TP + FP + FN + TN)) * 100.0;
                        Similarity = ((2 * TP) / (2 * TP + FN + FP)) * 100.0;
                        std::cout << "---------Comparison against MASK2----------" <<
endl;
                        std::cout << "TN: " << TN << " TP: " << TP << " FP: " << FP <<
" FN: " << FN;
                        std::cout << "Sensitivity: " << Sensitivity << endl;
                        std::cout << "Specificity: " << Specificity << endl;
                        std::cout << "Accuracy: " << Accuracy << endl;
                        std::cout << "Similarity: " << Similarity << endl;
                //}
                Mat greyex;
                cv::cvtColor(extract, greyex, COLOR_XYZ2BGR);
                cv::cvtColor(greyex, greyex, COLOR_BGR2GRAY);


                //Attempt to diagnose
                ///Cant really do circles becuase circles are far from perferct and very far
from imperfect
                //CAN do parallel ridge pattern detection.
```

```
cv::cvtColor(extract, extract, COLOR_XYZ2BGR);
extract = origimg.clone();
std::cout << mask2.rows << " " << mask2.cols << " " << extract.rows << "
" << extract.cols << endl;
for (int x = 0; x < Converted.cols; x++)
{
	for (int y = 0; y < Converted.rows; y++)
	{
		//AT WORKED WHEN PUTTING IT IN A EMPTY MAT
UNLIKE DATA! This will confuse me till the end of time as I have no clue what i sleepily did
on tuesday
		if (mask2.at<uchar>(Point(x, y)))
		{
			extract.at<Vec3b>(Point(x, y)) = Vec3f(255, 255,
255);
		}
	}
}


//For a rough throw together NOT BAD

float ridges = 0;
float lesionarea = 0;
float brown = 0;
float black_ = 0;
Vec3b brwn = (92, 64, 51);
for (int x = 0; x < extract.cols; x += 2)
{
	for (int y = 0; y < extract.rows; y++)
	{
		if (extract.at<Vec3b>(Point(x, y)) != Vec3b(255, 255, 255))
{
			lesionarea++;
		}
		if (x - 1 >= 0 && extract.at<Vec3b>(Point(x, y)) !=
Vec3b(255, 255, 255) && extract.at<Vec3b>(Point(x - 1, y)) != Vec3b(255, 255, 255)) {
			float dist = pow(extract.at<Vec3b>(Point(x, y))[0] -
extract.at<Vec3b>(Point(x - 1, y))[0], 2) + pow(extract.at<Vec3b>(Point(x, y))[1] -
```

```
extract.at<Vec3b>(Point(x - 1, y))[1], 2) + pow(extract.at<Vec3b>(Point(x, y))[2] -
extract.at<Vec3b>(Point(x - 1, y))[2], 2);
                                                dist = sqrt(dist);
                                                //cout << dist << " " << extract.at<Vec3b>(Point(x,
y)) << " " << extract.at<Vec3b>(Point(x - 1, y)) << endl;
                                                //Distance needs to be VERY LOW because of
bluring
                                                if (dist >= 6) {
                                                        extract.at<Vec3b>(Point(x, y)) = Vec3b(255,
0, 0);
                                                        ++ridges;
                                                }
                                                //Brown Color is called "wood bark"
BGR<6,10,35>
                                                dist = pow(extract.at<Vec3b>(Point(x, y))[0] - 6, 2)
+ pow(extract.at<Vec3b>(Point(x, y))[1] - 10, 2) + pow(extract.at<Vec3b>(Point(x, y))[2] - 35,
2);
                                                dist = sqrt(dist);
                                                if (dist <= 30) {
                                                        ++brown;
                                                }
                                                dist = pow(extract.at<Vec3b>(Point(x, y))[0] - 0, 2)
+ pow(extract.at<Vec3b>(Point(x, y))[1] - 0, 2) + pow(extract.at<Vec3b>(Point(x, y))[2] - 0, 2);
                                                dist = sqrt(dist);
                                                if (dist <= 35) {
                                                        ++black_;
                                                }
                                        }
                                }
                        }

                        //Auto Clustering
                        int i = 0;
                        Mat clustSearch = mask2.clone();
                        for (int x = 0; x < clustSearch.cols; x++)
                        {
                                for (int y = 0; y < clustSearch.rows; y++)
                                {
                                        if (clustSearch.at<uchar>(Point(x, y)) == 0) {
                                                floodFill(clustSearch, Point(x, y), Scalar(i + 50));
```

```
                                        ++i;
                                }
                        }
                }



                namedWindow("clustSearch", WINDOW_AUTOSIZE);
                cv::imshow("clustSearch", clustSearch);
                cv::moveWindow("clustSearch", 0, 90);
                int count = 0;

                std::cout << "Area of Legion:" << lesionarea << endl;
                std::cout << "clusters: " << i << endl;
                std::cout << ridges << endl;
                std::cout << "precentage that are ridges: " << (ridges / lesionarea) * 100.0
<< endl;
                if (((ridges / lesionarea) * 100.0) > 15) {
                        std::cout << "There is parallel-ridges pressent" << endl;
                        ++count;
                }
                if ((brown + black_ / lesionarea) * 100.0 >= 40) {
                        cout << "more than 40% of legions are blown or black" << endl;
                        ++count;
                }

                if (i < 4) {
                        cout << "made of few clusters" << endl;
                        ++count;
                }

                cout << " The extracted leagion fulfills " << count << "/3 of the detectable
melanoma characteristics" << endl;
                if (count <= 1) {
                        cout << "The extracted legion is CONFIDENTLY not melanoma"
<< endl;
                }
                if (count == 2) {
                        cout << "The extracted legion is MOST LIKELY melanoma" <<
endl;
```

```
                        }
                        if (count == 3) {
                                cout << "The extracted legion is CONFIDENTLY melanoma" <<
endl;
                        }

                        //cout << "residents of b town: " << (brown + black_) << " " << ((brown +
black_) / lesionarea) * 100.0 << endl;
                        //namedWindow("Orginial", WINDOW_AUTOSIZE);
                        //cv::imshow("Original", origimg);
                        //cv::moveWindow("Original", 0, 90);



                        ///*namedWindow("Mask", WINDOW_AUTOSIZE);
                        //cv::imshow("Mask", mask);
                        //cv::moveWindow("Mask", 0, 90);*/

                        ///*namedWindow("inpaint", WINDOW_AUTOSIZE);
                        //cv::imshow("inpaint", inpaint);
                        //cv::moveWindow("inpaint", 0, 90);*/

                        //namedWindow("Converted", WINDOW_AUTOSIZE);
                        //cv::imshow("Converted", Converted);
                        //cv::moveWindow("Converted", 0, 90);

                        //namedWindow("Mask2", WINDOW_AUTOSIZE);
                        //cv::imshow("Mask2", mask2);
                        //cv::moveWindow("Mask2", 0, 90);

                        //namedWindow("equalize", WINDOW_AUTOSIZE);
                        //cv::imshow("equalize", img);
                        //cv::moveWindow("equalize", 0, 90);

                        /*namedWindow("GT", WINDOW_AUTOSIZE);
                        cv::imshow("GT", GT);
                        cv::moveWindow("GT", 0, 90);*/
                        cvtColor(mask2, mask2, COLOR_GRAY2BGR);
                        /*namedWindow("extract", WINDOW_AUTOSIZE);
                        cv::imshow("extract", extract);
```

```
                        cv::moveWindow("extract", 0, 90);*/
                        /*imwrite("../OpenCVImages/Inpaints/Converted" + to_string(CTR) +
".png", Converted);
                        imwrite("../OpenCVImages/Inpaints/mask2_" + to_string(CTR) + ".png",
mask2);
                        imwrite("../OpenCVImages/Inpaints/equalized" + to_string(CTR) +
".png", img);
                        imwrite("../OpenCVImages/Inpaints/Extract" + to_string(CTR) + ".png",
extract);
                        imwrite("../OpenCVImages/Inpaints/flood" + to_string(CTR) + ".png",
flood);*/
                        //Melanoma Diagnosis
                        //++CTR;
                //}
        //Go in and see if I can diagnose if something is melanoma or not based off of visual
symtoms recognized


        cv::waitKey(0);
        cv::destroyAllWindows();
        return 0;
}
```