University of Groningen

bachelor thesis

Automatic separation of skin lesions from healthy human skin

Author:
Sander Feringa
Sietse Ludger Geertsema

Supervisor:
Dr. M. Biehl
Dr. M.H.F. Wilkinson

July 12, 2012

Abstract

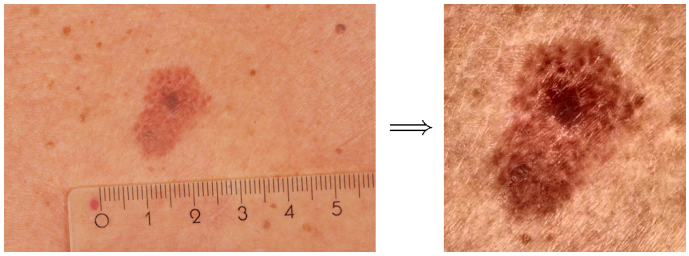
This bachelor thesis presents an implementation of a skin lesion segmentation system in digital images using Learning Vector Quantization (LVQ) that potentially could be used by physicians for skin lesion classification systems. A feature vector is extracted with different feature types including mean colour value and its standard deviation value, histograms and edge detection using two different colour models. For all images in the training dataset for use by LVQ, two subimages of an image with a skin lesion are taken, one subimage shows sick skin, one shows healthy skin. New images are divided up into clusters and each cluster is being compared in the classification phase of LVQ. This comparison is done between each clusters' feature vector and the feature vectors of the prototypes in the training set that have been made with the prototype generation phase of LVQ. A neighbourhood filter is also tested as a post-processing tool. F-scores are used as a result measurement and the maximum mean F-score produced by this software is for a dataset of 40 Melanoma images - with the neighbourhood filter not applied - a score of 0,79 for $K = 5$.

Contents

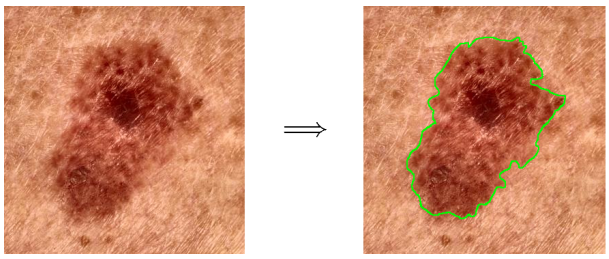
1 Introduction

Skin lesions are a common disease problem and physicians depend on their own abilities to distinguish different kinds of lesions. Since physicians are still human, mistakes can be made and different medical doctors can give (hopefully only slightly) different diagnoses. Determining the type of disease can be seen as a classification problem. To help physicians with this classification - or even to give patients a second opinion - an automatic classification system would be of great use. Such a skin lesion detection system can be described with the following steps:

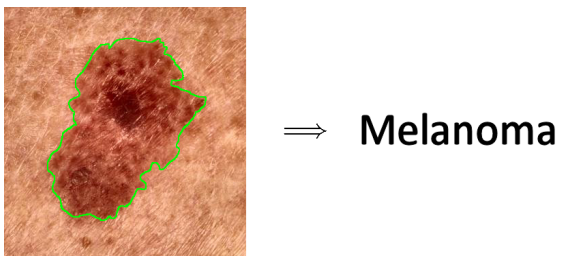
1. Separation of skin from non-skin.



2. Separation of unhealthy skin from healthy skin.



3. Classification of unhealthy skin.



At the University of Groningen research has been done within the Intelligent Systems group on skin lesion detection and classification systems [1] [2] [3]. We have looked into a specific part of that research in which we focus on the separation of non-healthy skin from healthy skin in digital pictures (step 2 of the list).

Classification can be accomplished with comparisons between "objects" that need to be classified and examples of those same "objects" that already have been classified. Since a standard bit-wise comparison of two digital images only gives a yes or no answer - and only a yes when they are exactly the same - some other form of information needs to be extracted from the images. These so called features can be extracted from images with the use of different techniques. Classification algorithms (or classifiers) like Learning Vector Quantization (LVQ) can be used on the extracted features to divide the original image into the healthy skin and non-healthy skin sub-groups. Using this selection, a classification system can be constructed that can identify skin lesions.

2 Background Information

There are different types of techniques relevant to Intelligent Systems that can be used to build a segmentation system. Feature vectors are used as a medium to describe features of an image. Different colour models can be used to encode the image and for the actual "smart" classification system learning algorithms are suitable.

2.1 Feature Vectors

A feature vector is an n-dimensional vector of numerical features that represents an object. The feature vectors are input values for classification algorithms that can be used for the actual skin lesion detection system. Each step mentioned in the introduction requires feature vectors.

Features can be extracted from digital images. Raw pixel data might be used for very specific tasks, but an abstraction of the data such as a histogram per colour of the picture is generally a better solution. An important advantage arises from the use of a feature set that is independent of the size of the picture (in pixels). Getting the features from an image is seen as a form of dimension reduction and is often called feature extraction.

Pre-processing can be used in combination with - or is an integral part of - feature extraction. Existing input data can be transformed into data that can give better results when used in classifiers later on. The authors of [4] give a good overview of steps that exist combined with some advice about pre-processing:

"In particular, one should beware of not losing information at the feature construction stage. It may be a good idea to add the raw features to the preprocessed data or at least to compare the performances obtained with either representation. We argue that it is always better to err on the side of being too inclusive rather than to risk discarding useful information."

However, including most or even all the information is not always the best solution. As more features are added, the dimensionality of the feature vector increases. And when the feature vector gets large the so called "Curse of Dimensionality" [5] [6] can start to present itself. It consists of multiple phenomena where the volume of the space of all data increases so fast that the available data becomes sparse. Irrelevant ("noise") dimensions can reduce the quality of the final result and in the end the inclusion of more dimensions will result in a negative effect on the overall performance of the system.

From these two sides it can be concluded that the selection of features for a feature vector and the correct use of pre-processing are very important for the whole skin lesion detection system. Adding more dimensions can have a positive or negative overall effect depending on a lot of variables.

2.2 Colour Spaces

Images can be represented using different colour spaces. Each space has its own advantages and disadvantages. All the following spaces have been developed by the CIE (Commision Internationale de l'Eclairage - the International Commission on Illumination). In our case all pictures are delivered to us in RGB format. But it could be the case that changing the colour space in the pre-processing phase results in information that could otherwise not have been obtained from RGB images.

2.2.1 RGB

The RGB (abbreviation of red, green and blue, the three primary colours of light) space is used in many digital cameras and computer displays. Colours are produced by adding red, green and blue values in different proportions. For instance yellow is made by adding the same value of red and green light, with no blue. When adding blue, the result will become whiter. Using 8-bit values for each primary colour $(0 - 255)$ there are a total of $256^3 = 16.7 \cdot 10^6$ possibilities. These are almost all the colours human eyes can see. Almost, because a few colours would need negative values to be represented by the 8 bit per channel RGB system. RGB is a very logical system to measure colour, since human eyes work with cones that are sensitive for red, green and blue light, although humans do not necessarily perceive colours that way.

2.2.2 CIELAB

The Lab colour space is defined by a dimension L for lightness and a and b for the colour-opponent dimensions. It is based on nonlinearly compressed CIE XYZ colour space coordinates. We use the CIE 1976 (L* a* b*) colour space (or CIELAB) version. Because of the separate lightness channel and the different way of storing colour in the two colour channels, there is a possibility that extra information from the images can be extracted compared to images in RGB colour space.

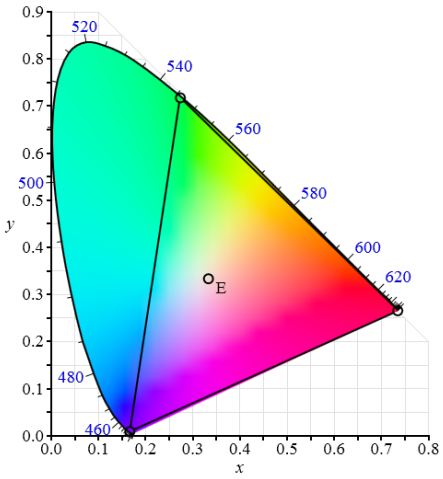


Figure 1: The CIE XYZ horseshoe, including the RGB triangle

CIELAB - or L*a*b* - is derived from the main colour space CIE 1931 XYZ, which defines all the colours within reach of human perception. Therefor RGB can be described a subset of L*a*b*. The intention of L*a*b* colour space is to generate a colour space which can be computed simply from the XYZ space, but is more perceptually uniform than XYZ. Perceptual uniformity means that a change of the same amount in a colour value should produce a change of about the same visual importance.

2.3 Classifiers

Since LVQ is in some ways an evolution of k Nearest Neighbours (or k-NN) a short summary is given about the history and function of k-NN, followed by information about LVQ itself.

2.3.1 k-Nearest Neighbour

k-Nearest Neighbour (k-NN) is a very simple and popular object classification algorithm and its history goes back as far as 1951 [7, 8, 9]. Given datapoints of two or more classes a new datapoint - or object - needs to be classified. Distances are calculated between the object and all the other datapoints in the dataset which classes are known. The object is assigned to the class which won the majority vote from a total of number k of nearest neighbours.

k-NN is a very good classifier because it uses all the data and it looks at multiple connections

between the unclassified sample and other samples. There is a major drawback though: a large amount of data is required and this makes k-NN slow. Often there is not enough data or using all of the data is too inefficient. This results in a need for more efficient algorithms. Different measures can be used for determining distances. Euclidean distance is frequently used without further justification [10]:

$$L_{a,b} = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{1}$$

with feature vectors a and b, n being the number of dimensions and L the distance between all values.

2.3.2 Learning Vector Quantization

One way to get better efficiency than with k-NN is to reduce the amount of distance measures that need to be calculated. Learning Vector Quantization (LVQ) is a classification method based on prototypes of the data and was introduced by Teuvo Kohonen [11]. The goal is to choose prototypes in a way that each prototype is a good representation for one of the classes. Using prototypes, only the distances between the object and the prototypes have to be calculated. In the simplest version of LVQ, distances are calculated using the squared Euclidean distance [12]:

$$d[w, \xi] = \sum_{j} (w_j - \xi_j)^2 \tag{2}$$

For a set of prototypes w, a given feature vector ξ, a number of classes C and class labels S:

$$\begin{aligned} &w^1, w^2, ..., w^K \qquad w^K \in \mathbb{R}^N \\ &S^1, S^2, ..., S^K \qquad S^K \in \{1, 2, ..., C\} \end{aligned}$$

The original formulation of LVQ, termed LVQ1, employs a winner takes all principle. If $d[w, \xi] \geq 0$ we can define a winning prototype w^* using:

$$w^{i*} = argmin_j \{d[w^j, \xi^t]\} \tag{3}$$

with ξ being assigned to class S^{i*}. For the training of the LVQ an initial randomized w^k is chosen together with a sequential presentation of labelled examples:

$$\{\xi^t, \sigma^t\} \;\; t = 1, 2, ..., P, 1, 2, ... \tag{4}$$

Combining this we come to a final learning algorithm:

$$w^{i*} \rightarrow w^{i*} + \eta_w \Psi(S^{i*}, \sigma^t)(\xi^t - w^{i*}) \tag{5}$$

Where η_w is the learning rate, and for $\Psi(S^{i*}, \sigma^t)$:

$$\Psi(S^{i*}, \sigma^t) = \begin{cases} +1 & \text{if } S = \sigma \\ -1 & \text{else.} \end{cases} \tag{6}$$

In the training phase the prototypes move closer to the datapoint belonging to the same class. When it belongs to a different class it is moved further away. After this procedure, the prototypes should be at optimum positions to represent their class. After a certain amount of epochs the situation is stable - the prototypes (almost) do not move anymore - and the training phase is complete. LVQ can be used on its own as a classifier by doing an k-NN like classification for which only the distances between the new object and the finalized prototypes made with LVQ have to be calculated.

2.3.3 Relevance Learning Vector Quantization

Lots of different versions of LVQ exist, but one that in particular is interesting: Relevance Learning Vector Quantization (RLVQ) [13]. RLVQ uses an extra vector of the same size as a feature vector that includes values between 0 and 1 and for which the sum of all values together is 1. With this extra relevance vector λ different weight (also known as relevance) can be given to certain feature(values) in the feature vector:

$$d_\lambda = \sum_j \lambda_j (x_j - w_j)^2. \tag{7}$$

Hebbian Learning can be used as an extra step to let the system learn which features are important and which are not. This way pruning can be applied on the features for overall speed gain without the system classification performance being to much affected. Sometimes the classification performance can even be improved with RLVQ.

3 Implementation

We can combine the techniques mentioned in the previous chapter in the following steps:

1. Input: digital image
2. Pre-processing (optional)
3. Feature extraction
4. Learning algorithms
5. Post-processing (optional)
6. Output: mask showing position of non-healthy skin in original image

The idea of this project is to test the feasibility of skin-lesion segmentation, not to produce a final production system for doctors to use in a real world situation. Matlab was chosen as the development platform because it already includes a large set of image manipulation tools and algorithms and because the authors of this thesis already had experience with implementing LVQ in Matlab. Special care was taken to build the Matlab code in such a way that it could be easily extended with new features if necessary.

A set of good quality pictures for training and testing is needed. Our supervisors had a dataset of images available that had already been used before in other research in skin lesions detection systems. The dataset consists of 11 disease types (see Table 1), with 40 images each. Of each image we have the original high resolution images, a smaller resolution excision of one skin lesion with surrounding healthy skin and 2 small subimages: one showing a piece of healthy skin and one showing sick skin in the image (see Figure 2). These two smaller sub images can be used to train LVQ and to test as whole images later.

Table 1: Skin lesion types

Actinic keratosis	Basal cell carcinoma
Eczema	Lentigo
Melanoma	Mycosis
Nevocellular naevus	Squamous cell carcinoma
Psoriasis vulgaris	Pustular psoriasis
Seborrheic verruca	

When an image is imported into Matlab, it is converted into a 3D matrix filled with integers representing pixels. For each colour or luminance channel there is a matrix the size of the image containing the grey value image of that channel. To automate the process of doing LVQ on a set of pictures of the same disease extra Matlab functions where created.

3.1 Feature Vector

The first step is to determine a good set of features for the feature vector. The features are implemented as a sequential list values that have been normalized in the $[0, 1]$ range, stored in one vector called the feature vector. After a process of testing the following features are used:

- **Mean and standard deviation:** The mean is an easily extracted feature and is implemented by representing each image with a mean for each colour or luminance channel. For every mean the standard deviation is also calculated. All values are normalized.
- **Histogram:** For each colour and luminance channel a histogram is calculated. Because both colour spaces are 8 bit, 256 discrete intervals - or so called "bins" - are used. The histograms are normalized, otherwise a histogram of a big image will have larger numbers per value than a histogram of a small image. To do this the values of the histogram are divided by the size of the image.
- **Edge:** All of the above features are based on colour or luminance information from the pictures. With the use of edge detection comes the possibility to also retrieve data from a different type of information: edges telling something about the structure in images. If we look at images of skin lesions we can already see a difference in structures and textures between sick skin and healthy skin. Matlab's own edge detection functions were deemed "not good enough" for this project because they all generate a 1 bit image per colour channel which results in too much information loss. A custom implementation of "Sobel" edge detec-

(a) Original image

(b) Skin lesion subsection

(c) Healthy skin subsection

(d) Sick skin subsection

Figure 2: Image types in the dataset from 1 image of a patient with the disease Naevocellularis

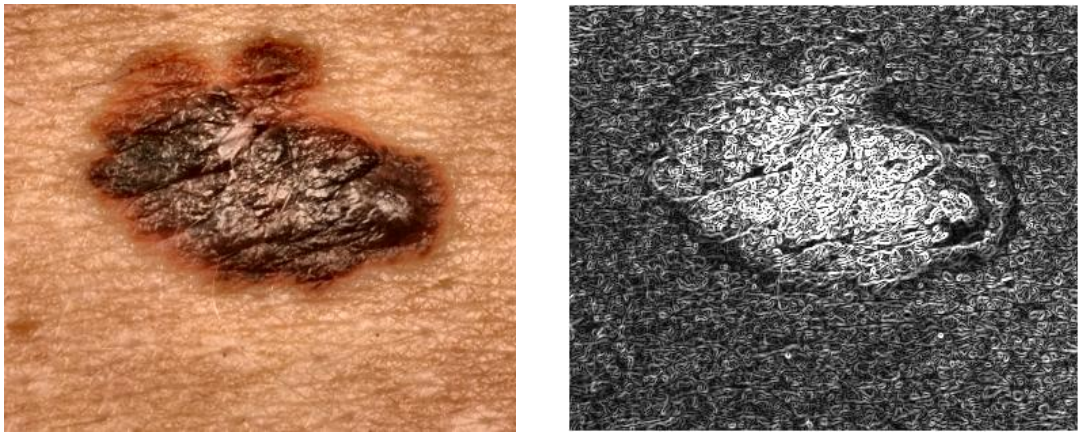
tion [14] was employed that results in three grey value image (one for each colour channel). From each of these channels a histogram is obtained.

Sobel Operator for an image A, with a 2D convolution $\star$ and where G_x and G_y are two images which at each point contain the horizontal and vertical derivative approximations:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \star A \qquad \text{and} \qquad G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \star A$$

At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{G_x^2 + G_y^2} \tag{8}$$



(a) Original image with a Melanoma

(b) Sobel filter output image

Figure 3: Sobel edge detection filter

The distance between two feature vectors need to be determined and Euclidean distance is used for this. Table 2 shows the sizes of all the features that make up the final feature vector:

Table 2: Feature Vector

Type	Size
RGB mean	3
Lab mean	3
RGB std	3
Lab std	3
RGB histogram	3*256
Lab histogram	3*256
RGB sobel histogram	3*256
Total	2316

3.2 Generating prototypes and the classification process

LVQ [1] delivers a few feature vectors - called prototypes - from the whole dataset. There are two classes: sick and healthy. K is a variable given to the algorithm to define the number of prototypes the system needs per class.

1. Select randomly K feature vectors from both the healthy and sick ones. Combine both to one list named *prototypes*.
2. Combine all feature vectors, both of the sick and healthy images, in one list, called *data*.
3. For multiple *epochs* - or until the prototypes are not moving anymore - repeat a loop:
   (a) Randomize the dataset and enter another loop:
      i. Pick a fv of the dataset.
      ii. Find the closest prototype using Euclidean distance.
   (b) If the closest prototype is of the same class, move the prototype a little in the direction of the current feature vector. If not, move it a little in the opposite direction.

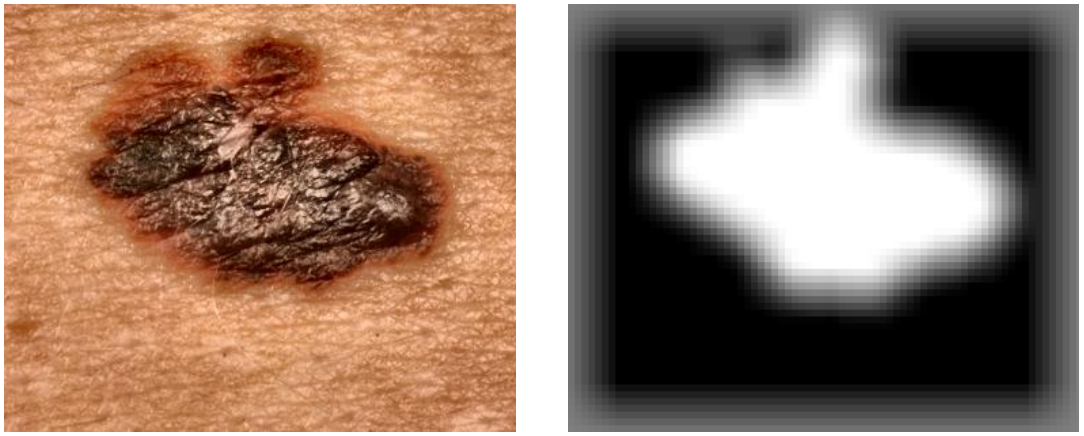
The classification of an object is done by a majority vote of its neighbours (which are in this case the prototypes generated in the step before). The object is assigned to the class with the highest number of K neighbours of the same class.

1. For a given disease: start with making a vector of feature vectors for each (sub)image (healthy or not healthy) in the dataset. As told before each main image has two subimages, one showing only healthy skin and one showing only sick skin.
2. Loop through the whole image in clusters of adjustable size with an adjustable step size. Now the real classification starts:
   (a) There are two lists with feature vectors, one healthy (fv_h) and one sick (fv_s) for all the training images and there is a feature vector for each of the test clusters (fv_t).
   (b) For all the feature vectors in both lists, the distance is calculated between fv_h and fv_t, and fv_s and fv_t using Euclidean distance.
   (c) This results in two lists with distances: one for healthy and one for sick. Now both lists are combined in one table, with a value to indicate whether it is sick or healthy.
   (d) This table is sorted on distance.
   (e) The K smallest distances are taken.
   (f) Count the amount of sick and healthy values in those k results. If there are more sick than healthy, it is classified as sick, or if more healthy than sick, it is classified as healthy.
3. This will result in a number, 1 if sick and -1 if healthy.
4. Each pixel in the test-image will now have a positive or negative number. Each pixel that has a positive number is multiple times classified as sick, so it can be concluded that it probably is classifiable as sick skin.
5. This information is used to create a mask of the test-image. While looping trough all pixels in the test-image, check for every pixel if it has a positive or negative number. If it has the former it is saved as sick skin, if it is the latter it is omitted. All the omitted pixels combined are seen as healthy skin.
6. This results in an image where - if everything went right - just spots of sick skin remain.

[1] The relevance vector of RLVQ has been implemented but is not used in this project. The λ vector is a predivined vector which consists of 2316 times $1/2316$ to have a total λ value of 1. This way it has no influence on the end result. More information about RLVQ is located in the Future work chapter.

3.3 Post-processing with neighbourhood averaging

Post-processing techniques can be used to try to improve results. Figure 4 show the resulting "score value" image (this is the output of the LVQ process) as a grey value image. Since the output image is non-binary, the exact border between the sick and healthy skin can be influenced by thresholding after the learning algorithms have done their work.



(a) Original image with a Melanoma

(b) Score value image

Figure 4: Score value greyscale image

The score value images can be used to do a form of neighbourhood averaging. Score value images can be filled with false positive outliers in the healthy skin or false negative "holes" in the sick skin that need to be removed as good as possible. If the neighbourhood filter is not applied a standard binary value image is created from the score value image using the mean colour value of the whole image as a standard threshold.

The neighbourhood filter has been implemented in the following way:

1. Generate a new image (im_b), the same size as the current one (im_a), filled with only zeros.
2. Compute the mean ($mean_a$) of im_a and take all pixels with a value higher than this mean.
3. Loop over the image with the same clusters as in LVQ:
   (a) Generate a new image, containing the current cluster, and its 8 neighbours
   (b) Take the mean ($mean_b$) of this cluster
   (c) If $mean_b$ is higher than $mean_a$ of the whole image, replace the zeros in im_b with ones at the place of the current cluster
4. Return im_b.

4 Results

Ground truth images have been made to compare with the results of the program. The results themselves have been made quantifiable using the F-score.

4.1 Ground truth

Far accurate scientific results ground truth images are needed. This way, comparisons can be made with what the "optimal" version of the program should accomplish. These images are made up of the original images with a manually added clearly visible line that shows where the best segmentation-edge is. It would be best if multiple dermatologists (even dermatologists can disagree on where the line should be) would draw these edges and that the average of those edges is used. Having multiple dermatologists at work making these ground truth images is too big a task for a bachelor project, so the following option was that the authors of this report would themselves generate these ground truth images and let them be checked by a dermatologist. But in the end it was concluded by the authors and their supervisors that this was too complicated to organize on short notice. Since making these images is labour intensive they have only been made for the 40 melanoma pictures in the dataset. Therefor only results have been calculated for the disease Melanoma.

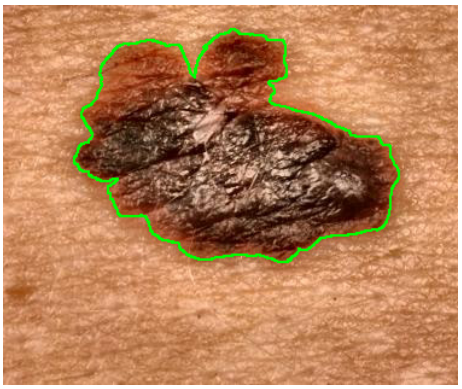


Figure 5: Ground truth image with a green line showing the segmentation-edge

4.2 F-score

F-scores are a way of testing the accuracy of a test. It takes both precision (the number of correct results divided by the number of all returned results) and recall (the number of correct results divided by the number of results that should have been returned) in account to compute a score. From the ground truth image a binary image is created that shows sick skin in white and healthy skin in black. The binary output of the program (which is in the same colour pattern) is then compared with the binary ground truth image. The best score of an F-score is 1, worst is 0. White on white and black on black is correctly classified, black on white or white on black is wrongly classified. From this data the number of true positives, false positives and false negatives can be calculated and used for the calculation of the precision, recall and the F-score:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{9}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{10}$$

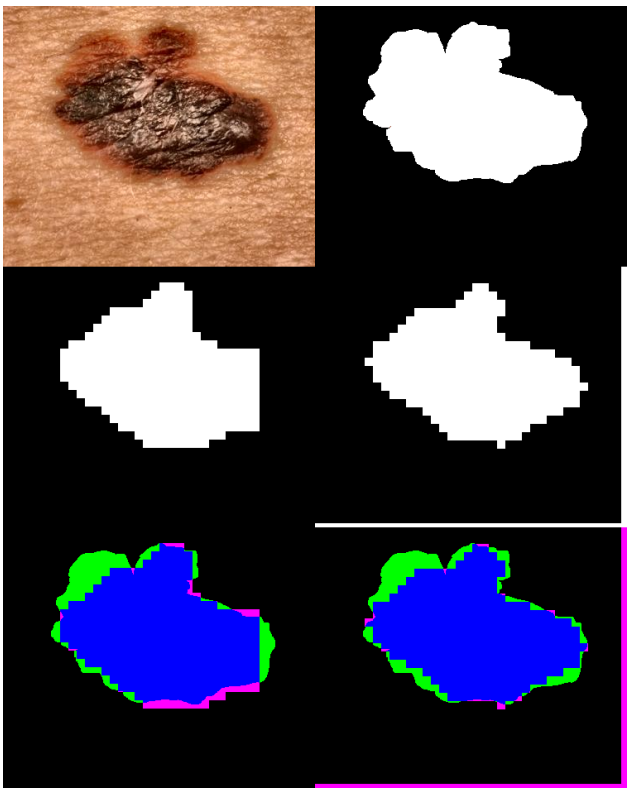
$$F\text{–}score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{11}$$

4.3 Results

In this project LVQ is applied first on the 40 sick and 40 healthy subimages to generate K number of prototypes and using a learning rate value η. The learning rate is in all cases 0.01 and stays constant in time. Cluster size value p and stepsize value s are defined[2].

Glossary:

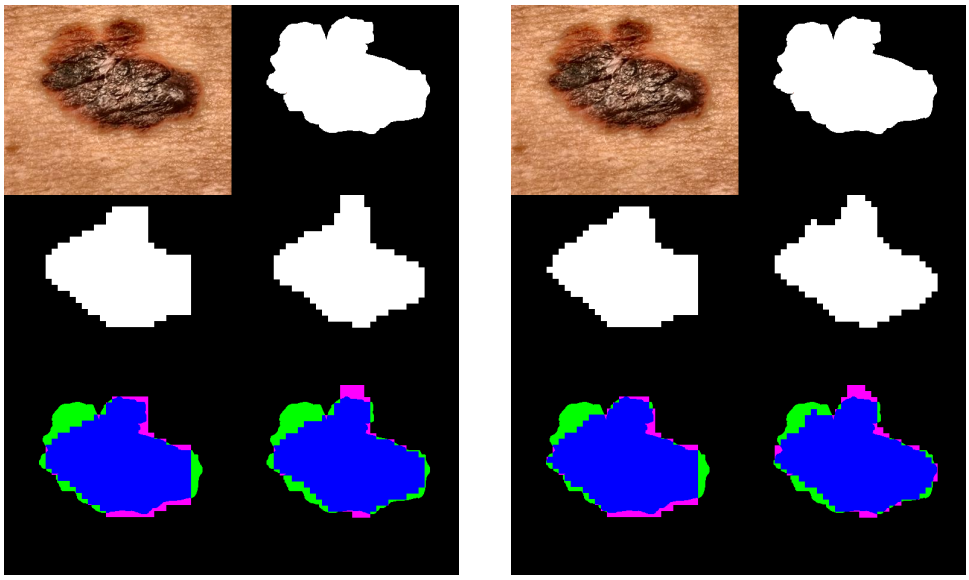
- The original image is in the top left corner, the segmentation-mask made from the ground truth image (sick skin is white, healthy skin is black) is in the upper right corner, the middle row shows the segmentation made by the program and the lower row shows the program's result overlaid with the ground truth image.
- For the lower two rows: the left column of images is post-processed with the neighbour algorithm and the right column is the standard implementation without post-processing.
- The blue colour represents pixels that are correctly classified as sick, the green colour represents pixels that should have been classified as sick (false negative) and purple represents pixels that are classified as sick while they are healthy (false positive).



(a) K = 1, F = 0.899152, F = 0.870299

Figure 6: (glossary on page 15) p = 20, s = 10, η = 0.01

[2]See the Future work chapter for more variables options that can be explored

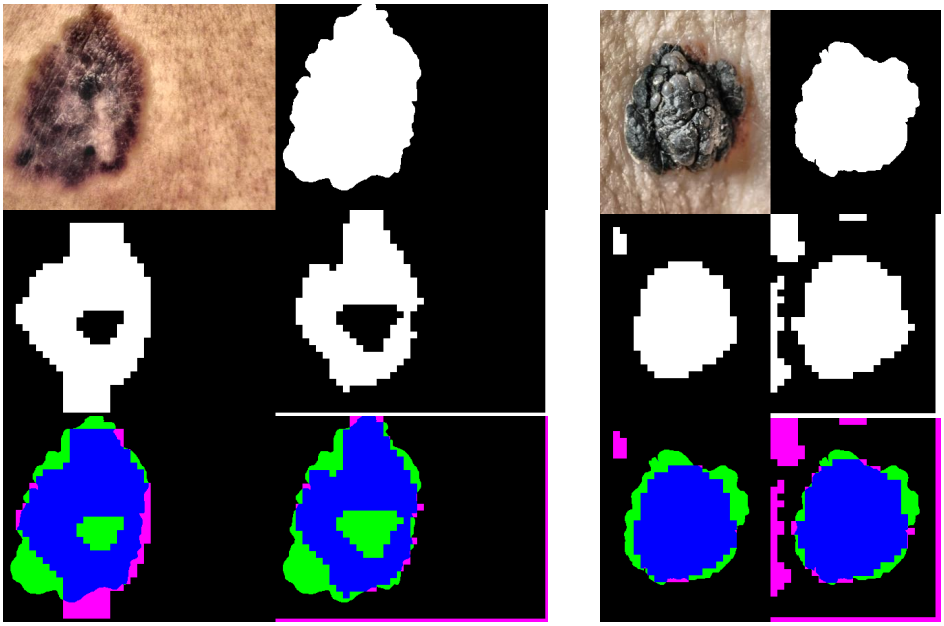


(a) K = 3, F = 0.895903, F = 0.892779

(b) K = 5, F = 0.905712, F = 0.914051

Figure 7: (glossary on page 15) p = 20, s = 10, $\eta = 0.01$

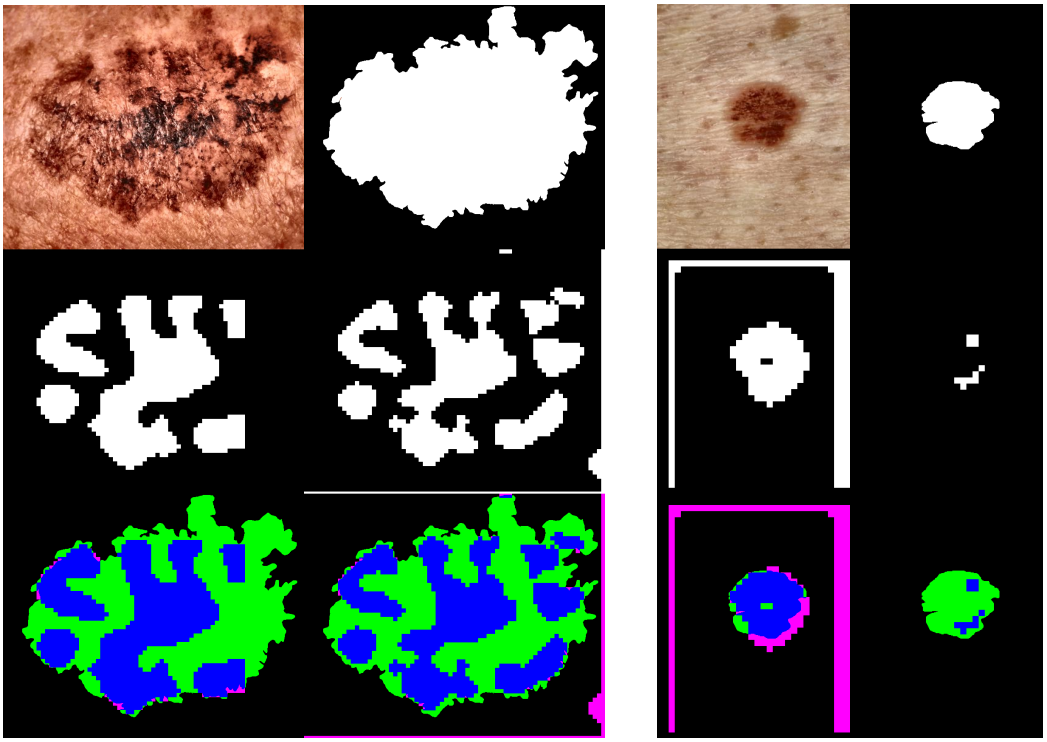
The two images in Figure 8 show some positive looking results. The one on the left shows clearly that the very dark spots on the skin lesion are easily detected by the program, but when the colour gets lighter the program starts to have trouble detecting the lesion. Especially lighter spots in the middle of the lesion are hard to discover. The neighbourhood algorithm often does its job in making the hole smaller or even filter it out completely. The image on the right shows the problems the program can have with a lot of false positive clusters outside the skin lesion area. The standard version sees two whole edges of clusters on the borders of the image as being sick skin. The neighbourhood filter does its job in removing these artefacts and increasing the F-score.



(a) Another example 1, K = 1, F = 0.832924, F = 0.783183

(b) Another example 2, K = 1, F = 0.849181, F = 0.817740

Figure 8: (glossary on page 15) p = 20, s = 10, $\eta = 0.01$



(a) Another example 3, K = 1, F = 0.639154, F = 0.644788

(b) Another example 4, K = 3, F = 0.736536, F =0.194432

Figure 9: (glossary on page 15) p = 20, s = 10, $\eta = 0.01$

The two images in Figure 9 give more output examples. The image in Figure 9a is not easy to segment because it does not have really sharp edges. The edges between the lighter and darker spots in the skin lesion are very blurry and even for a human it is difficult to distinguish between sick and healthy skin. The result is that the program creates "islands" of sick skin patches. The neighbourhood algorithm isn't strong enough to fix this problem: it produces the "islands" slightly larger, but not large enough to connect them all together. Only the darker parts that are clearly sick skin are classified as such.

The image in Figure 9b shows that the standard algorithm has a hard time classifying clusters as sick skin. The neighbourhood filter works well in this case to increase the number of sick skin clusters. However, it also introduces large clusters of skin being classified as sick at three of the edges of the image, resulting in false positives. Somehow the filter is overdoing the filtering it was set out to do.

Comparrisons have been made for three different K values: K = 1, K = 3 and K = 5. To generate the final results of the Melanoma dataset one iteration has been done for K = 1 [3]. For K = 3 and K = 5 five iterations are done and all calculated F-scores are averaged to result in mean and standard deviation values as showed in Table 3.

[3]Doing more iterations is useless since the prototypes are always the same. This comes from using the average value of all dimensions of all datapoints for generating the prototypes for K = 1.

Table 3: Results

		Neighbourhood filter	Standard
K = 1	Mean	0.7788	0.7437
	Std	0.1707	0.1050
K = 3	Mean	0.7680	0.7234
	Std	0.1802	0.1257
K = 5	Mean	0.7892	0.7907
	Std	0.1728	0.0780

The final F-scores show that for the standard situation there is no explainable relation between the K values: K = 3 has the lowest mean but also the highest standard deviation of the three. There is a 6.7% difference in F-scores between the lowest and the highest score for the mean and K = 5 has the lowest standard deviation and the highest mean score. For the neighbourhood algorithm version we can only see a difference of 2.1% between the lowest and the highest mean of the 3 runs. The standard deviation is here as good as the same for all 3 runs. For both K = 1 and K = 3 the application of the neighbourhood algorithm seems to be useful to increase the F-scores. However, for K = 5 there is even a slight decrease in his case, but its not a significantly significant decrease.

Looking at Figure 10 something else has to be mentioned: one of the pictures in the K = 3 run using the neighbourhood algorithm resulted in an F-sore of NaN (not a number warning by Matlab, caused by a divide by 0, we changed it to 0.0 for our F-score calculations) that has a negative effect of the mean F-score of the test run with those parameters. It can be concluded that this is actually the result of the image not being suitable enough for this program because it was taken under quite a low angle from the skin surface as opposed to all the other pictures that have been taken from above. Parts of the image are even out of focus so detail is lost.[4]

(a) Another example 5, K = 3, F = 0.0, F = 0.871678

Figure 10: (glossary on page 15) p = 20, s = 10, $\eta = 0.01$

[4]Actually, it could also be summarized that the program is not suitable enough for these images. But the authors of this thesis do not want to publicize this openly since it can be bad for their morality.

5 Conclusion

The goal of this bachelor thesis is to research the possibility of using an automated software system to segment images with skin lesions into sick and healthy skin. The automation of this process can help physicians identify skin lesions.

The example pictures have shown that the program certainly has potential for being utilized for this task, but it also shows that it is not yet ready. The F-scores are all closely packed together. In some pictures the neighbourhood filter does a marvellous job at removing clusters that shouldn't be included. In other pictures however it has positive and negative effects or even only negative effects. The maximum mean F-score for the dataset of 40 Melanoma images at this point is 79.0% for K = 5 with the neighbourhood filter not applied.

However, the scores can be quite different even with some minor changes in the process. For instance by using a better selection of images in the dataset, with professionally checked ground truth images, with the adaption of a better neighbourhood filter or by using the relevance learning of the RLVQ algorithm instead of the standard vector that is used now.

Getting to the final version of the program took several iterations and a lot can be done to let it perform better. The speed of the program increased with the iterations as well. For instance: first converting the whole image to L*a*b* for the feature extraction process before starting the clustering process instead of first clustering and then converting each cluster to L*a*b* made the whole program three times faster. Not all of Matlabs built-in features for vector operations have been exploited though.

The presence of a considerable amount of for loops in the code means that Matlab is still not used in its optimal form using vector operations.

Final conclusion: it should be possible to produce an automatic sick skin segmentation program for practical use since the software of this thesis shows great potential. However, a lot more research and testing needs to be done before that goal has been reached.

6 Future work

There are numerous ways to improve the performance of the program:

- **Better ground truth images**: The results of this project were made using ground truth images that have not been checked by dermatologist. It would be best if future study would include ground truth images that are annotated by professionals.
- **Predefined rules for images**: Some of the pictures in the dataset used in this project are not good enough. Especially pictures with bad lighting, low resolution pictures or pictures taken under an angle are almost useless. If a version of the program is used in a real-life situation it would be useful to define a set of rules for input images. These rules should include the minimum and maximum size of the skin lesion on the image compared to healthy skin, the pictures should always be taken from above, taken with diffuse light and no specular highlight and the minimum resolution.

  Extra care must be taken in choosing the two small subimages used to represent sick and healthy skin. The subimages must be made in such a way that all relevant types of sick or healthy skin are present. This is important for both the images in the training set as well as any image that needs to be segmented. Through this procedure there is a greater chance that all the types of sick or healthy skin are correctly classified in an image and fewer false negatives and false positives are present in the final output.
- **Variables**: In this project most of the variables - like clustersize p, stepsize s and learning rate η - have been chosen in such a way that they delivered moderate speed and quality performance for the software. More experiments with different variable values need to be done to discover this higher quality performance.
- **Pre-processing**: There is a change that the results of LVQ can be enhanced when certain pre-processing filters are applied on the image. One example is contrast enhancement by stretching. Images taken by digital cameras often have a contrast range that is less than the full 256 values per colour that are available. Stretching the contrast of the image to the maximum value range can enhance the final results of LVQ.
- **Max Tree**: There are other feature types possible besides the ones used now. One of them is the Max Tree. Adèle Caillot writes in [3] that a Max Tree is made by segmenting the image on multiple levels of granularity. For a grey scale image pixels can be grouped together that are local to each other and have almost the same intensity value. A node of the tree is defined as a local group of pixels. The leaves of the tree are defined as the maximal intensity areas. Each of the children of a node represents a subset of the node. A Max Tree is made for images with bright spots on a dark background. It is also possible to search for dark spots on a light background. This can be done in two ways: by using the inverted image and a Max Tree, or by inverting the order structure in the Max Tree so it becomes a Min Tree. Skin lesions are often darker than the healthy skin surrounding them so the converted Max Tree variant is a possible feature type.
- **Connected filters**: Another feature type that can be used are connected filters, as for instance presented in [15]. Connected filters are a form of morphological filters that preserve connectivity. They work with connected components in an image instead of just pixels. This way they work at a higher level of connections than just simple pixels can give. Since the program needs to show clear differences between patterns and textures in sick skin compared to healthy skin, these connected filters have great potential.
- **Hole fill algorithm**: As seen in some of the output images, sometimes one or more holes are left in the middle of a skin lesion. Or there are false positive outliers present in the healthy skin segment. The use of our neighbour filter is a partial solution in solving this problem. However, there are special hole fill algorithms available that - as a post-processing

filter - could search for holes in the image and classify the relevant pixels correctly. There is a disadvantage with the use of these algorithms: they are slow.

- **Relevance learning**: RLVQ has been implemented, but is not used. It can be useful to test with different relevance vectors and see what there influences are. Seen the size of the set of features that are included in the program, a substantial number of combinations of relevance values have to be tested. An even better opportunity would be a system in which the relevance values are learned to find the optimum relevance vector. Versions with matrices also exist and could be explored [16, 17].
- **Divergences instead of standard Euclidean distance**: Like most previous research done by others, Euclidean distance was used to calculate the distances between feature vectors. However, there could be other distance measures that give better results. Divergences have already been researched by others [10, 18] and they could be implemented in this program.

7 References

[1] B. van de Wal, Automatic Classification of Hand Dermatitis, Rijksuniversiteit Groningen, 2007.

[2] T. Havinga, Automatic Severity Assessment of Hand Eczema, Rijksuniversiteit Groningen, 2010.

[3] A. Caillot, Salience-based pseudo pattern spectra for automatic analysis of skin lesions, 2010.

[4] I. Guyon and A. Elisseeff, An Introduction to Feature Extraction, Feature Extraction: Foundations & Applications, 1-25, Springer, 2006.

[5] R.E. Bellman and Rand Corporation, Dynamic Programming, Rand Corporation research study, Princeton University Press, 1957

[6] R.E. Bellman, Adaptive control processes: a guided tour, Rand Corporation Research studies, Princeton University Press, 1961.

[7] E. Fix and J.L.Hodges, Jr., Discriminatory analysis, nonparametric discrimination: Consistency properties, Report Number 4, Project Number 21-49-004, USAF School of Aviation Medicine, 1951.

[8] T.M. Cover and P.E. Hart, Nearest neighbor pattern classification. IEEE Trans. Inform. Theory, IT-13(1), 21-27, 1967.

[9] R.O. Duda, P.E. Hart and D.G. Stork, Pattern Classification, Wiley-Interscience, second edition, 2000.

[10] E. Mwebaze, P. Schneider, F.M. Schleif, J.R. Aduwo, J.A. Quinn, S. Haase, T. Villmann and M. Biehl, Divergence-based classification in learning vector quantization, 2011.

[11] T. Kohonen, Self-Organizing Maps, Berlin, 1995.

[12] M. Biehl, Supervised Learning: Learning Vector Quantization, Relevance Learning, course Introduction Intelligent Systems, Rijksuniversiteit Groningen, 2011.

[13] T. Bojer, B. Hammer, D. Schunk, K. Tluk von Toschanowitz, Relevance determination in learning vector quantization, in: M.Verleysen (ed.), European Symposium on Artificial Neural Networks'2001, D-facto publications, 271-276, 2001

[14] E. Trucco and A. Verri, Introductory Techniques for 3-D Computer Vision, Prentice Hall, Chapter 4.2, 1998.

[15] P. Salembier, M.H.F. Wilkinson, Connected Operators A review of region-based morphological image processing techniques, IEEE Signal Processing Magazine, Volume 26, Issue 6, pp. 136-157, Nov. 2009.

[16] P. Schneider, M. Biehl, B. Hammer, Relevance Matrices in LVQ, in: M. Verleysen (ed.), European Symposium on Artificial Neural Networks (ESANN 2007), d-side publishing, pp. 37-42, 2007.

[17] P. Schneider, M. Biehl, B. Hammer, Adaptive Relevance Matrices in Learning Vector Quantization, Neural Computation, 2009.

[18] K. Bunte, Adaptive Dissimilarity Measures Dimension Reduction and Visualization, PhD Thesis, Chapter 9, Rijksuniversiteit Groningen, 2011.

Appendix

On the following 5 pages are the ground truth versions of all 40 Melanoma images in the dataset.

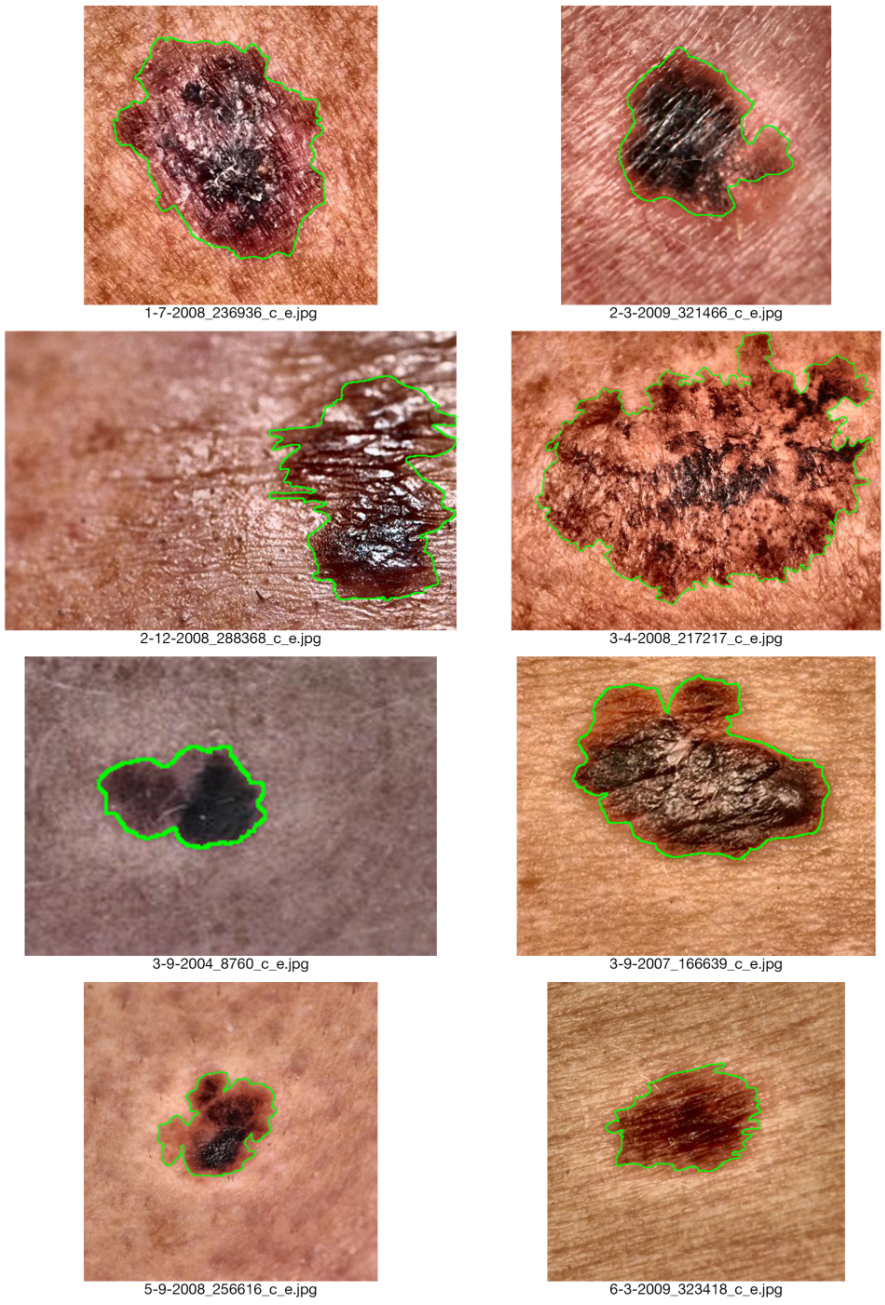


Figure 11

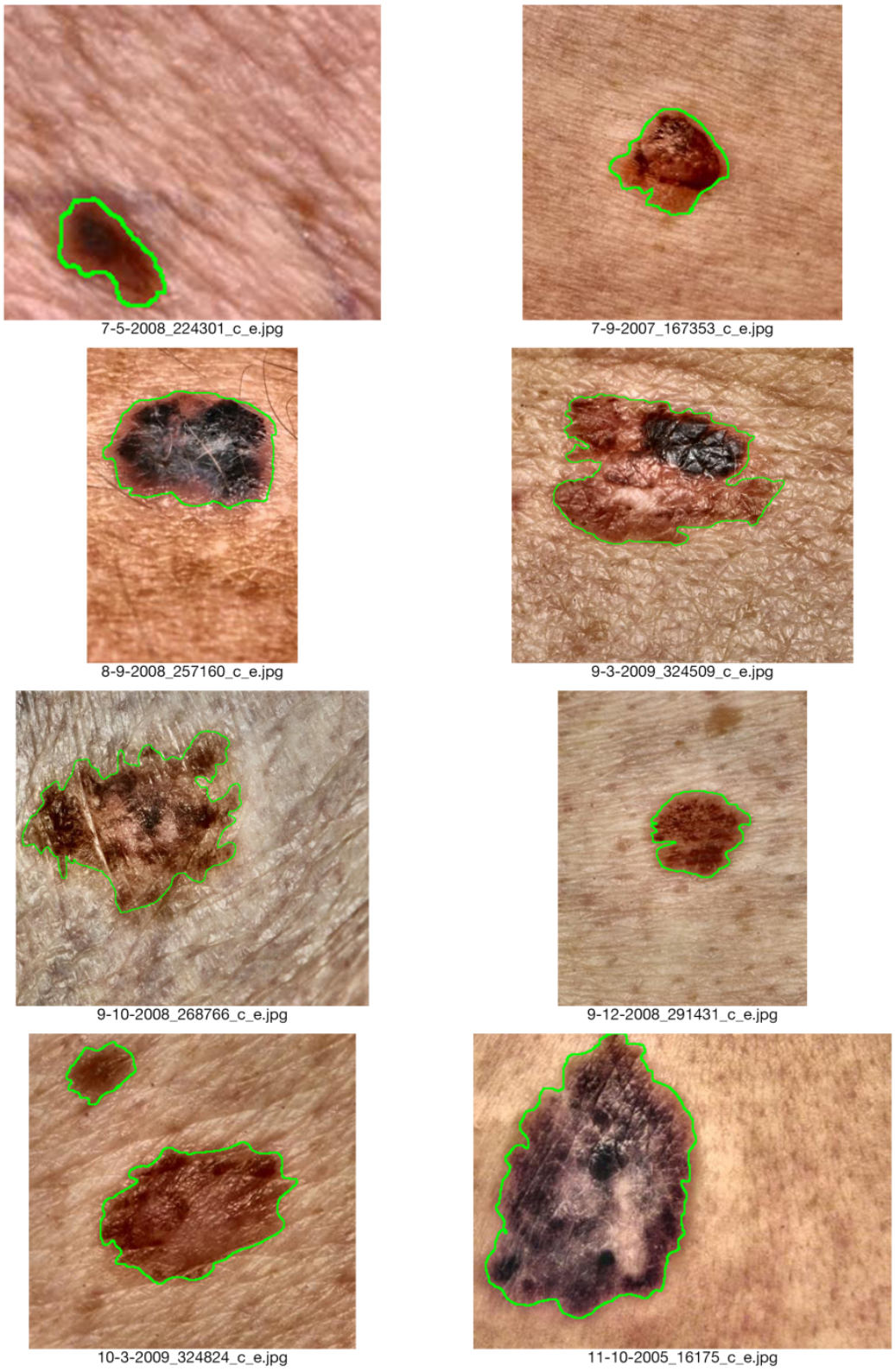
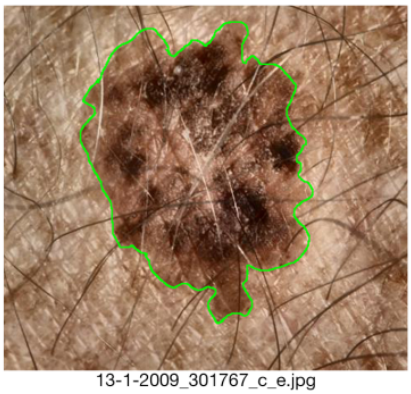
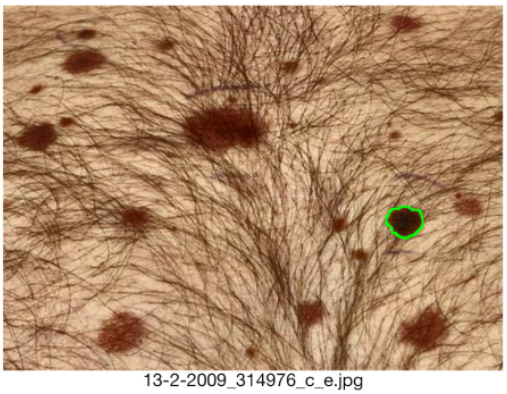

7-5-2008_224301_c_e.jpg
7-9-2007_167353_c_e.jpg
8-9-2008_257160_c_e.jpg
9-3-2009_324509_c_e.jpg
9-10-2008_268766_c_e.jpg
9-12-2008_291431_c_e.jpg
10-3-2009_324824_c_e.jpg
11-10-2005_16175_c_e.jpg

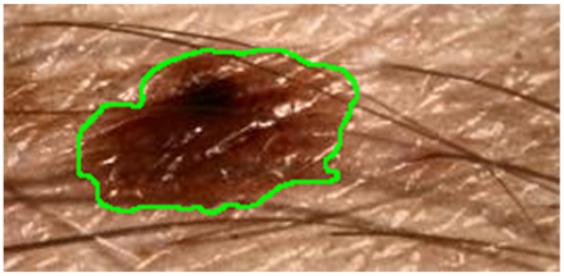
Figure 12

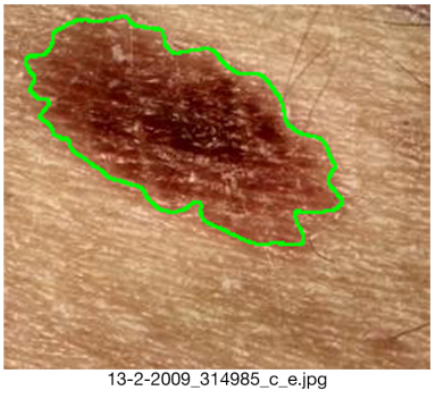

11-12-2006_136733_c_e.jpg


12-9-2008_258765_c_e.jpg


13-1-2009_301767_c_e.jpg


13-2-2009_314976_c_e.jpg


13-2-2009_314981_c_e.jpg


13-2-2009_314985_c_e.jpg


13-6-2002_896_c_e.jpg

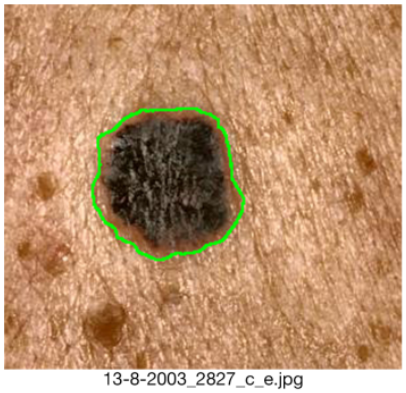

13-8-2003_2827_c_e.jpg

Figure 13

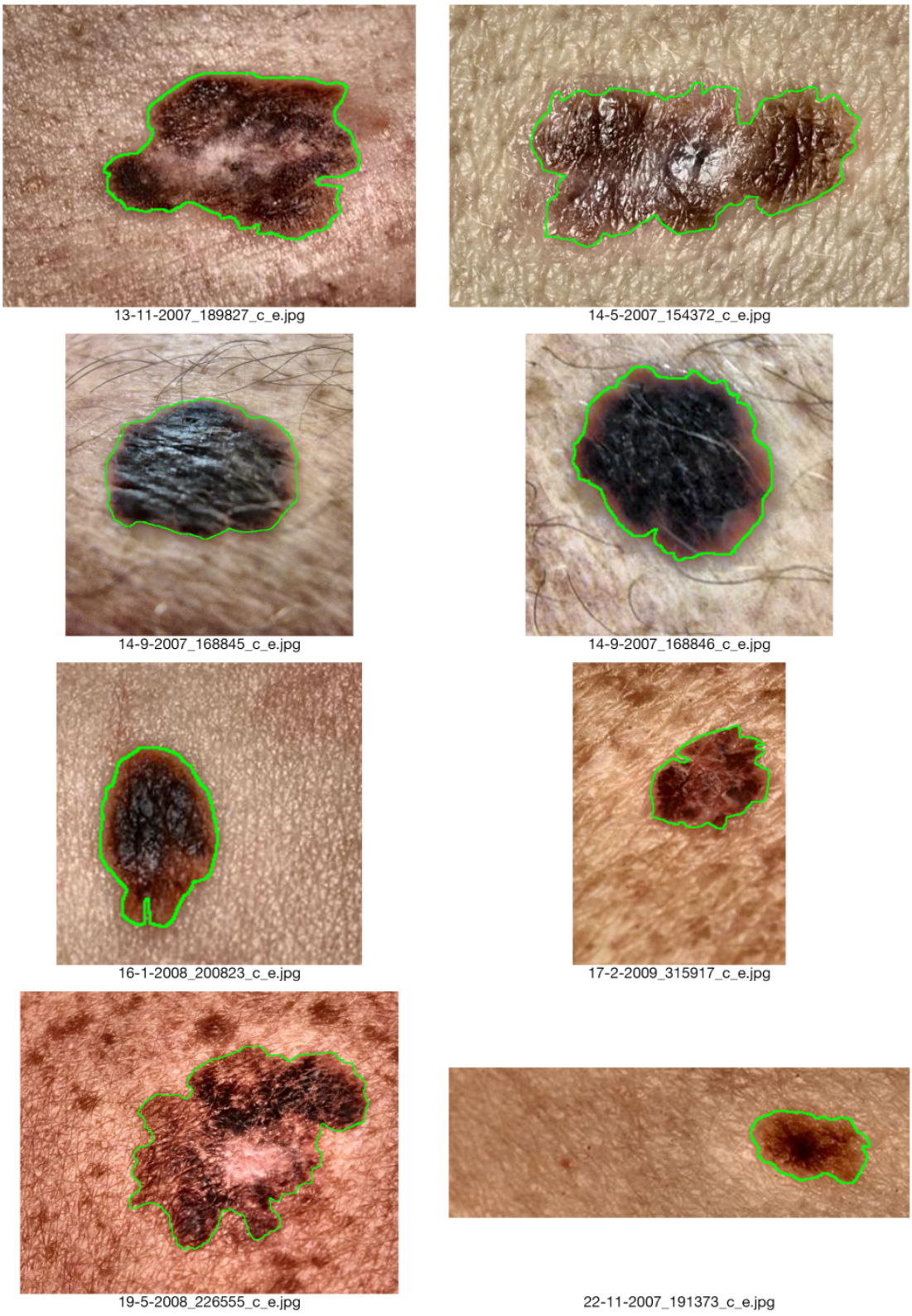

13-11-2007_189827_c_e.jpg
14-5-2007_154372_c_e.jpg
14-9-2007_168845_c_e.jpg
14-9-2007_168846_c_e.jpg
16-1-2008_200823_c_e.jpg
17-2-2009_315917_c_e.jpg
19-5-2008_226555_c_e.jpg
22-11-2007_191373_c_e.jpg

Figure 14

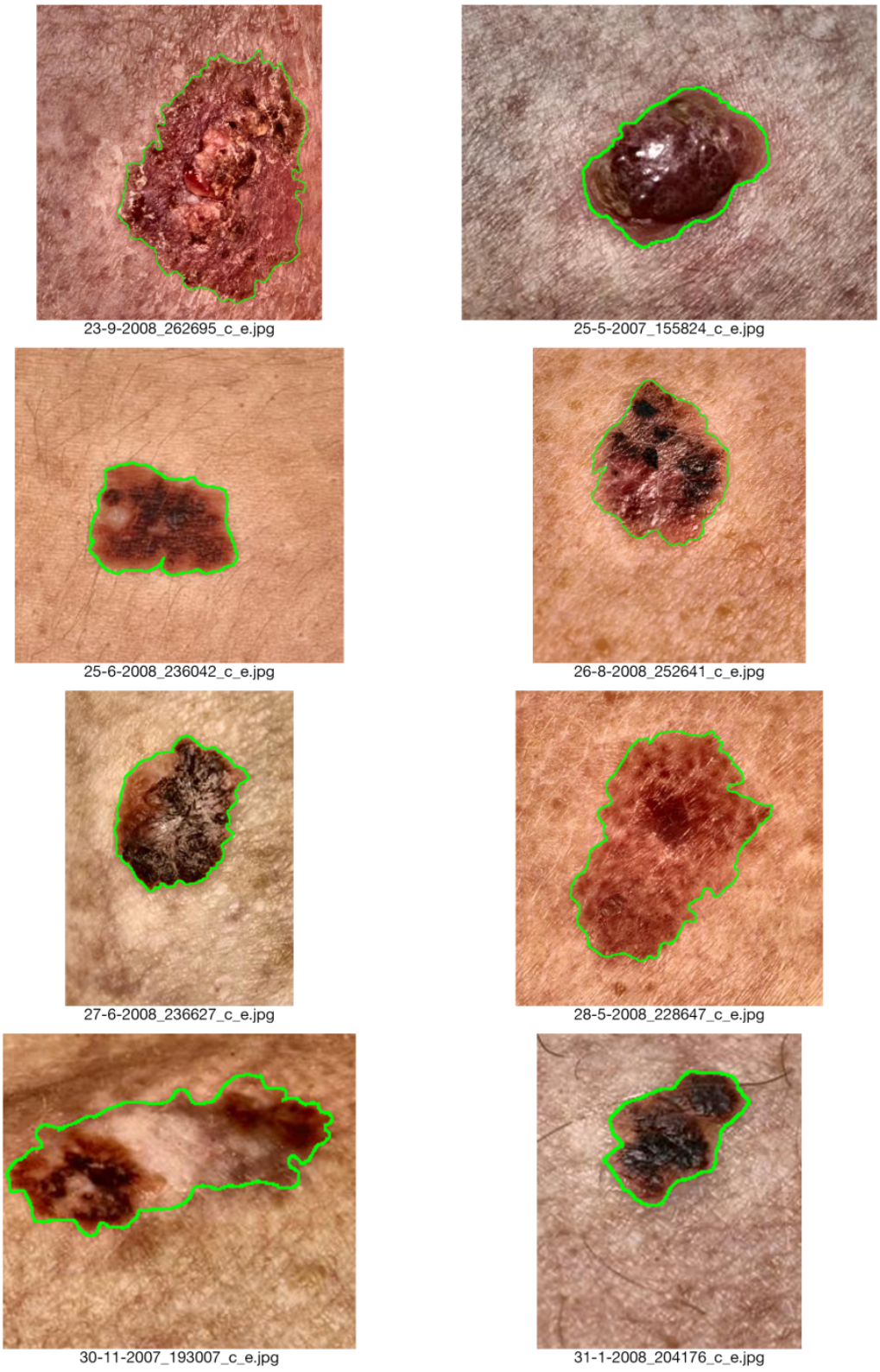

23-9-2008_262695_c_e.jpg
25-5-2007_155824_c_e.jpg
25-6-2008_236042_c_e.jpg
26-8-2008_252641_c_e.jpg
27-6-2008_236627_c_e.jpg
28-5-2008_228647_c_e.jpg
30-11-2007_193007_c_e.jpg
31-1-2008_204176_c_e.jpg

Figure 15