Skin Lesions Classification using Computer Vision and Convolutional Neural Networks

Image Processing and Artificial Vision
Master Degree in Computer Science Engineering
Polytechnic University of Bari

Biagio Montaruli - b.montaruli@studenti.poliba.it

9 September 2019

1 Introduction

Today, skin cancer is a public health and economic issue, in fact several researches state that skin diseases are one of the most common human illnesses, affecting every age, gender and pervading many cultures, summing up to between 30% and 70% of people in the United States [1]. The recommended way to detect early skin diseases is to be aware of new or changing skin growths. One of the most adopted techniques for skin lesion analysis is the so-called ABCD analysis, that consists of scanning the skin area of interest for asymmetry (A), border irregularity (B), colors variegation (C) and diameter (D) [2, 3].

Due to the fact that with early detection, the 5 year survival rate of the most deadly form of skin lesion, melanoma, can be up to 99% [4], the possibility to detect skin cancer in early stages and do it through automatic skin lesions classification systems, having performance at least or greater than traditional detection methods, is a rather promising challenge.

The emergence of powerful machine learning techniques such as deep learning has enabled the development of intelligent medical image analysis systems that, through the use of Artificial and Convolutional Neural Networks, have been proved to have remarkable performance in comparison to standard methods. As discussed in [1, 5], several researches have been carried out in order to apply deep learning for automatic skin lesions analysis, however a limiting factor in training of neural networks is given by the small size and lack of diversity of available datasets of skin lesion images. To overcome these issues, Tschandl et al. (2018) released the HAM10000 ("Human

Against Machine with 10000 training images") dataset [5] that was used for the ISIC 2018 classification challenge and also in the on-going ISIC 2019 challenge.

The HAM10000 dataset has also been used in this work because it provides a very well-organized and comprehensive source of dermatoscopic images. In particular, it consists of 10015 dermatoscopic RGB images (with shape 600 × 450 pixels) in JPEG format that have been collected over a period of 20 years from two different sites, the Department of Dermatology at the Medical University of Vienna (Austria) and the skin cancer practice of Cliff Rosendahl in Queensland, Australia. In addition, it includes a representative collection of all important diagnostic categories in the realm of pigmented lesions: actinic keratoses and intraepithelial carcinoma/Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis-like lesions such as solar lentigines and seborrheic keratoses (bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv) and vascular lesions like angiomas and angiokeratomas (vasc). Furthermore, the 7 categories/classes in which images have been subdivided, can be further grouped in order to have only two classes (this is particularly useful for binary classification) as suggested in [1]: one containing malignant lesions (mel, bcc, akiec) and the other containing benign lesions (bkl, df, nv, vasc). It is important to note that the authors of [1], decided to put the actinic keratosis (akiec) in the malignant class since it is considered a precancerous lesion.

Given that background, this work will guide its efforts in applying deep learning and some image processing techniques to images of the HAM10000 dataset in order to develop an advanced skin lesion classification system: after applying some pre-processing techniques to enhance contrast and remove hairs, images are processed using two distinct pipelines. The first one uses the AlexNet convolutional neural network (CNN) for image classification [6], while the other adopts the Support Vector Machine classifier that is trained using features extracted from the lesion regions. In particular, the second pipeline first applies a region-based segmentation technique using the marker-based watershed algorithm in order to subdivide the image in regions of interest (ROI), then, among all the regions, we find the one representing the lesion and finally we perform feature extraction based on (an extension of) ABCD analysis, so that the extracted features can be used for the training of the SVM machine learning model. Since the pipelines provide two different methods for classifying dermatoscopic images, another important goal of this project is to compare their results and performance in order to analyze in details the difference between shallow and deep learning methods.

The remainder of the report is organized as follows. Section 2 provides a detailed description of the project implementation by explaining in details the two pipelines and the involved image processing techniques. Section 3 describes both textually and graphically the obtained results. Finally, Section 4 draws the conclusions by making a comparison between the two pipelines and proposes some future work possibilities.

2 Project implementation

This section describes in details the project implementation and the adopted image processing techniques. The project has been written in Python and consists in a Jupyter notebook containing both the source code and the project documentation. It has been developed in the Google Colaboratory (Colab) environment because Colab has been designed to develop machine learning projects and indeed it consists is a free Jupyter notebook environment that runs entirely in the cloud. Two of the most useful features offered by Colab (that are also used in this project) are the availability of a free-to-use Nvidia K80 GPU for up to 12 hours and the integration with Google Drive. The use of a GPU can greatly speed up the execution speed especially when training a neural network, while regarding Google Drive, it can also be mounted in the runtime's virtual machine in order to import files saved in the user's Drive. In particular, in this project the HAM10000 dataset is downloaded and saved into Google Drive so that it can be easily imported each time the project is run (it is important to note that if the notebook is restarted, all local files in the virtual machine are deleted).
The main Python libraries used to design the source code are scikit-learn [7, 8], scikit-image [9] and Keras [10]. The former is an open source machine learning framework, built on top of NumPy, SciPy and matplotlib, that has been designed to be efficient, easy-to-use and includes a wide range of supervised and unsupervised algorithms. On the other hand, scikit-image is an open source image processing library based on NumPy and SciPy, which includes several modules that implement all the main image processing algorithms for segmentation, geometric transformations, color space manipulation, filtering, morphology operations, feature extraction and so on. The third one, Keras, is one of the most popular Python libraries for deep learning, designed to be interoperable with the most well-known frameworks for machine learning and neural network applications like Tensorflow, Theano and PyTorch, through the use of high-level and user-friendly APIs.

As described in the previous section, the implementation is made of two distinct pipelines and, even though both aim to classify the images of the HAM10000 dataset, they use different methods for images classification. In particular, they both share some initialization steps that consist in the workspace preparation, which splits the HAM10000 dataset into the training and test sets, and the following images preprocessing. Then, the program runs the first pipeline which trains the AlexNet CNN using the training images and evaluates performance using the test set. Later, we run the second pipeline that applies region-based segmentation in order to find, in each

image, the region containing the lesion, from which the main significant features are extracted using a method based on ABCD analysis. The extracted features belonging to training images are used to train the Support Vector Machine (SVM) classifier, while features related to test images are used for performance evaluation.
In summary, the main steps of this work are:

1. Workspace preparation:
    - Import all the necessary libraries.
    - Create the `dataset`, `train` and `test` directories.
    - Mount the Google Drive at `/content/drive`.
    - If not in the Drive, download the HAM10000 dataset from Harvard Dataverse and then extract it into the `dataset` folder.
    - Declare all the necessary functions that will be used in the next steps.
    - Split the HAM10000 dataset into the training and test sets.
2. Preprocessing:
    - Enhance contrast using the Contrast Limited Adaptive Histogram Equalization (CLAHE) technique.
    - Remove hairs using morphological closing combined with the median filter in order to simulate the DullRazor filter [11].
3. Pipeline #1: Classification using the AlexNet CNN
    - Build the AlexNet model.
    - Train the AlexNet model using the training images.
    - Evaluate performance over the test set and plot the obtained results.
4. Pipeline #2: Classification using the SVM model based on ABCDT features
    - Apply image segmentation in order to subdivide each image into regions of interest (ROI).
    - Find the region containing the lesion through the lesion identification algorithm.
    - Perform feature extraction using the ABCDT method based on the ABCD analysis, which extracts features related to asymmetry (A), border irregularity (B), color variegation (C), diameter of the lesion (D) and texture (T).

- Train the SVM classifier and find the best combination of its hyperparameters.
- Evaluate performance over the test set and plot the obtained results.

Although above we have provided an overview of the project structure, a detailed description of its implementation is provided in the follow.

2.1 Workspace initialization

After importing all the necessary libraries such as Keras (`keras`), scikit-learn (`sklearn`), scikit-image (`skimage`), Matplotlib (`matplotlib`) [12] and Seaborn (`seaborn`), the program creates the following three directories:

- *dataset*: it will contain all the images of the HAM10000 datset.
- *train*: it will contain the preprocessed images of the HAM10000 datset to be used for training the classifiers.
- *test*: it will contain the preprocessed images of the HAM10000 datset to be used for doing performance evaluation.

Then, we mount an instance of Google Drive at `/content/drive` so that we can import and extract the HAM10000 dataset tarballs into the `dataset` folder. If the tarballs are not found, they are downloaded from HAM10000 dataset from Harvard Dataverse.

As the last thing to complete the initialization, the program declares all the necessary functions and classes, which are:

- `image_show()`: it allows to plot a single image and set some useful parameters like `size` and `cmap`, using `matplotlib`.
- `imshow_all()`: it allows to plot a series of images side-by-side using `matplotlib`.
- `imshow_with_histogram()`: it allows to plot an image along with its histogram using `matplotlib`.
- `morph_closing_each()`: this function is "decorated" with the `adapt_rgb()` decorator that is used in conjunction with the `each_channel` handler provided by the *scikit-image* library.
  In this way, when `morph_closing_each()` receives in input an RGB image, it passes each of the RGB channels to the morphological closing operator (implemented in *scikit-image* through the `morphology.closing()`) one-by-one, and stitches the results back into a new RGB image.

- `median_filter_each()`: this user-defined function is "decorated" with the `adapt_rgb` decorator that is used in conjunction with the `each_channel` handler provided by the *scikit-image* library.
  In this way, when `median_filter_each()` is called passing an RGB image, it passes each of the RGB channels to the median filter (implemented in *scikit-image* through the `filters.median()`) one-by-one, and stitches the results back into a new RGB image.

- `build_train_test()`: it chooses (randomly) `num_samples` images from the HAM10000 dataset (if `num_samples` is zero (by default) or less than zero, all the samples in the dataset will be chosen) and splits them into the training and test sets in a stratified fashion. Through the `train_set_frac` parameter, the user can choose the proportion of the dataset to include in the train split.

- `preprocessing()`: this function is used to apply some image processing techniques to the original images contained in the training and test sets. The images are firstly processed with the CLAHE technique to enhance contrast and then hairs are removed from images using the morphological closing followed by the median filter.

- `images_segmentation()`: this function, as its name suggests, is used to perform region-based segmentation of the images contained both in the training and test sets. It basically consists in finding the main regions of interest (ROI) by using the watershed transform and then selects among them, the one that more probably represents the skin lesion.

- `features_extraction()`: it is used to extract some features using ABCDT analysis in order to allow images classification.

- `AlexNet()`: this function defines the AlexNet model.

- `Metrics`: this class will be used to compute the performance metrics (accuracy, precision, recall and F1-score) at the end of each epoch during the training of the AlexNet CNN.

- `plot_history()`: this function is used at the end of the training process to plot the trend of performance metrics (accuracy, precision, recall and F1-score) over all the epochs.

- `plot_confusion_matrix()`: this function receives in input the list of ground truth (correct) target values (`true_labels`), the estimated targets as returned by a classifier (`predicted_labels`) and a list of class labels in order to, first build the confusion matrix using the `confusion_matrix()` method of `sklearn.metrics` module, and then plot it using the `heatmap()` function of Seaborn Python library.

- `run_alexnet()`: it is in charge of running the AlexNet model in order to train it over the training set and evaluating its performance over the test set.

2.2 Preprocessing

After the initialization phase, images contained in the training and test sets are preprocessed using the `preprocessing()` function. The preprocessing consists in the following 3 steps:

1. **Contrast-Limited Adaptive Histogram Equalization (CLAHE)**: It is an algorithm for local contrast enhancement, that uses histograms computed over different tile regions of the image [13]. The main idea behind CLAHE (but also the ordinary histogram equalization) is that each pixel is transformed based on the histogram of a square surrounding the pixel by using a transformation function. The transformation function is proportional to the cumulative distribution function (CDF) of pixel values in the neighbourhood. However, unlike the ordinary histogram equalization that tends to overamplify the contrast (and thus causing noise to be amplified) in near-constant regions of the image [14], CLAHE limits the contrast amplification by clipping the histogram at a predefined value before computing the cumulative distribution function (CDF). This limits the slope of the CDF and therefore the noise amplification.
The main advantage of applying CLAHE to medical images like the ones of HAM10000 dataset, is the possibility to improve contrast and highlight important details since medical images can be affected by poor illumination and low contrast.

2. **Morphological Closing**: Morphological closing is a fundamental operation that belongs to morphological image processing. Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an binary or grayscale image. Morphological techniques probe an image with a structuring element having a simple shape like a circle or a square. The structuring element is positioned at all possible locations in the image and it is compared with the corresponding neighbourhood of pixels. Some operations test whether the element "fits" within the neighbourhood, while others test whether it "hits" or intersects the neighbourhood.
In particular, considering a binary image, when a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighbourhood under the structuring element. The structuring element is said to fit the image if, for each of its pixels set to 1, the corresponding image pixel is also 1. Similarly, a structuring element is said to hit, or intersect, an image if, at least for

one of its pixels set to 1, the corresponding image pixel is also 1 (see Figure 1).

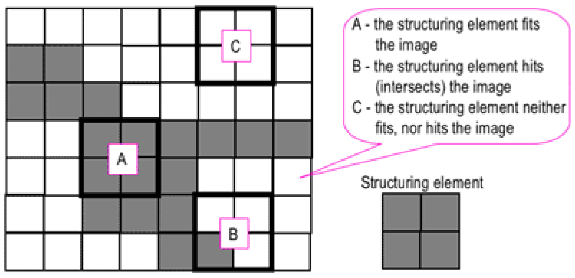
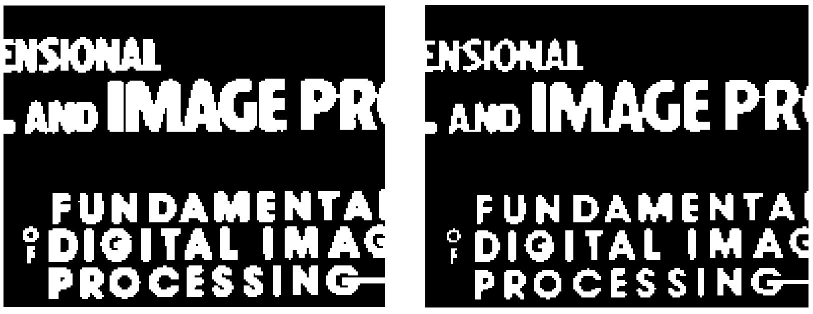

C
A
B
A - the structuring element fits the image
B - the structuring element hits (intersects) the image
C - the structuring element neither fits, nor hits the image
Structuring element

Figure 1: Probing of an image with a structuring element (white and grey pixels have zero and non-zero values, respectively)

The main morphological operators are:

- *Erosion*: The erosion of a binary image *f* by a structuring element *s* produces a new binary image *g* with ones (1s) in all locations (x, y) of a structuring element's origin at which that structuring element *s* fits the input image *f*, that is *g*(x, y) = 1 is *s* fits *f* and 0 otherwise, for all pixels (x, y). In practice, morphological erosion sets a pixel at place (x, y) to the minimum over all pixels in the neighborhood centered at (x, y).
  As shown in Figure 2, erosion is used to shrink bright regions and enlarge dark regions, so it can be used to remove small-scale details but at the same time it can reduce the size of regions of interest. Another useful application is the possibility of highlighting boundaries of a region of interest by subtracting the eroded image from the original image.

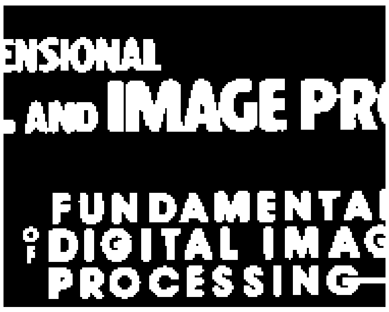


(a) Original binary image. (b) Erosion with a 2×2 square.

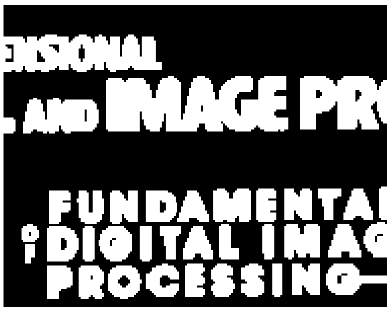
Figure 2: Morphological erosion.

- *Dilation:* The dilation of an image *f* by a structuring element *s* produces a new binary image *g* with ones (1s) in all locations

(x, y) of a structuring element's origin at which that structuring element s hits the the input image f, that is $g(\mathrm{x}, \mathrm{y}) = 1$ if s hits f and 0 otherwise, for all pixels (x, y). In practice, morphological dilation sets a pixel at place (x, y) to the maximum over all pixels in the neighborhood centered at (x, y).
As opposed to erosion, dilation enlarges bright regions and shrinks dark regions, so it can be used to fill/reduce holes enclosed by a single region and gaps between different regions (see Figure 3).



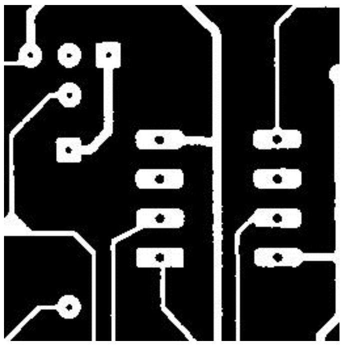
(a) Original binary image.



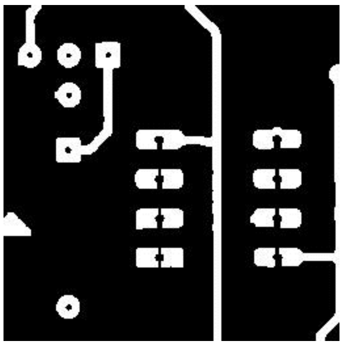
(b) Dilation with a 2×2 square.

Figure 3: Morphological dilation.

- *Opening:* The opening is a compound morphological operation that consists of an erosion followed by a dilation.
  Opening can remove small bright spots and connect small dark regions. In fact, as shown in Figure 4, this tends to "open" up (dark) gaps between (bright) regions.



(a) Original binary image.



(b) Opening with a 5×5 square.

Figure 4: Morphological opening.

- *Closing:* The closing is a compound morphological operation that consists of an dilation followed by an erosion.

As opposed to opening, closing can remove small dark spots and connect small bright cracks (see Figure 5). This tends to "close" up (dark) gaps between (bright) regions.

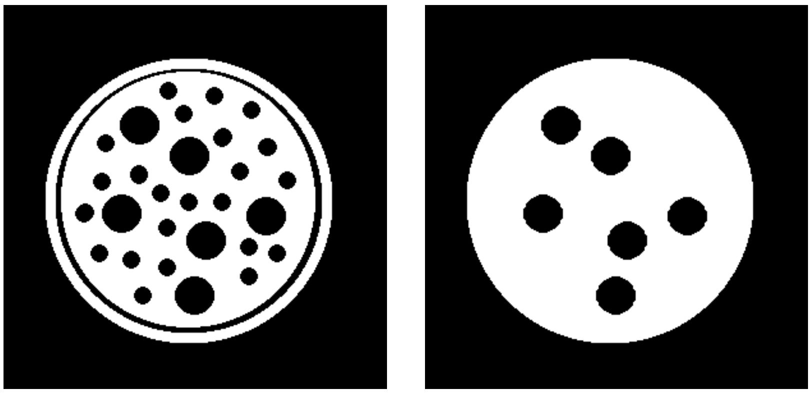

(a) Original binary image. (b) Closing with a 22 pixel diameter disk.

Figure 5: Morphological closing.

3. **Median Filter**: The median filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. It is also widely used in pre-processing steps to improve the results of later processing such as segmentation and feature extraction, in fact one of the main advantage is that it preserves edges while removing noise. It is applied to each pixel of the image by using a window or structuring element having a simple shape as a disk or a square. The pixel under the origin of the structuring element is replaced with the median of the neighbouring pixels. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into ascending order and then replacing the pixel being considered with the middle pixel value (if the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used).

In this work, we apply CLAHE through the `equalize_adapthist()` method of `skimage.exposure` in order to enhance the contrast and brightness of the images, while `morph_closing_each()` is used to apply morphological closing to equalized images using the a disk of radius 7 in order to remove hairs and stains. Morphological closing is followed by the image filtering using the median filter (`median_filter_each()`) in order to perform a fast and effective linear interpolation. This is the basic approach used in the DullRazor filter in order to effectively remove hairs from images representing skin lesions.

2.3 Pipeline #1: Classification using the AlexNet CNN

Convolutional neural networks are a milestone of computer vision since they allow to achieve reasonable performance on hard visual recognition tasks that, in some domains, match or exceed human performance. For this reason this subsection provides a gentle introduction to neural networks and then explains how they are used in this project for image classification.

An *artificial neural network (ANN)*, is a parametric machine learning model whose main goal is to approximate a function $f : X \rightarrow Y$, where $Y = \{C_1, \ldots, C_k\}$ or $Y = \Re$. So, an ANN defines a mapping $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation. The structure of an ANN consists in a collection of connected units, also called nodes or artificial neurons, that are organized in layers so that each neuron in a layer is connected to some other neurons in the next layer. Moreover, the final layer of an ANN is called the output layer, while the intermediate layers are called hidden layers because the training data do not show the desired output for each of these layers. The number of layers in an ANN defines the depth of the network, instead the dimensionality (i.e., the number of neurons) of the hidden layers determines the width of the model. Designing and training a neural network usually requires to define:

- the depth and width of the network
- the activation function that will be used to compute the values of hidden layers.
- the loss, or cost, function.

Regarding the depth and width of a neural network, they usually depend on the problem we need to solve and in practice a deep and narrow network is usually easier to train and provides better generalization results [15].
As for the loss and activation functions, their choice usually depends on the type of output units. In most cases, since an ANN defines a distribution $P(y|\mathbf{x}; \boldsymbol{\theta})$, we can simply use the principle of maximum likelihood and so this means that we can use the cross-entropy between the training data and the model's predictions as the loss function [15].
In particular, for regression problems, we have output linear units and so we can adopt a linear activation function $\mathbf{y} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$ assuming a Gaussian noise distribution model $P(t|\mathbf{x}) = \mathcal{N}(t|y, \beta^{-1})$. Moreover, in this case the maximum likelihood corresponds to minimizing the mean squared error.
In case of binary classification problems, a good choice for the activation function is the sigmoid function $y = \sigma(\mathbf{w}^T\mathbf{x} + b)$, instead the maximum likelihood (and so the loss function) corresponds to a Bernoulli distribution $J(\boldsymbol{\theta}) = -\ln(\sigma((2t - 1)\alpha))$, with $\alpha = \mathbf{w}^T\mathbf{x} + b$ (binary cross-entropy).
Finally, for multi-class classification problems, we can choose the softmax activation function $y_i = softmax(\alpha)_i = \frac{e^{\alpha_i}}{\sum_j e^{\alpha_j}}$ and the maximum likelihood

corresponds to a Multinomial distribution $J(\boldsymbol{\theta})_i = -\ln(softmax(\alpha)_i)$.
For the units in the hidden layer, there are several options in choosing the activation function, however some of the most common choices that are used in practical situations are the rectified linear unit (ReLu) function $g(\alpha) = max(0, \alpha)$, the sigmoid $g(\alpha) = \sigma(\alpha)$ or the hyperbolic tangent function $g(\alpha) = \tanh(\alpha)$.

In order to train the network, we need to compute the gradients of loss functions with respect to the network parameters $\boldsymbol{\theta}$. In ANNs, this can be done using the *backpropagation* algorithm, which is characterized by two phases:

- in the first phase, called forward propagation, the input $\mathbf{x}$ provides the initial information that then propagates through the hidden units at each layer and finally produces $\mathbf{y}$ from the output layer, which is used to compute a scalar error through the loss function $J(\boldsymbol{\theta})$.
- the second phase called back propagation distributes the error term back up through the layers (it is used to propagate gradient computation backward through the whole network).

Convolutional neural networks (CNNs) are a specialized kind of artificial neural networks that use convolution in place of general matrix multiplication in at least one of their layers and they are especially designed for processing images and multidimensional signals. The general structure of an CNN for image classification is shown in Figure 6.
A convolutional layer of a CNN typically consists of three stages [15]:

1. *Convolutional stage*: it performs several convolutions in parallel between the input and the kernel in order to produce a set of linear activations which are also referred to as the feature maps. Convolution is based on two main ideas:
   - sparse interactions or sparse connectivity: it is accomplished by making the kernel smaller than the input. This implies that we need to store fewer parameters and so we reduce the memory requirements of the model and improve its statistical efficiency. In fact, if there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters; however in a CNN, since we limit the number of connections (outputs) to k (where $k < n$), the convolution operation requires only $k \times n$ parameters.
   - Parameter sharing: it is a regularization method that consists in using the same parameter for more than one function in a model. This means that, rather than learning a separate set of parameters for every location, we learn only one set of them.
2. *Detector stage*: each feature map obtained in the previous stage is run through a non-linear activation function (ReLu, tanh, ...).

3. *Pooling stage*: we use a pooling function to modify the output of the layer further in order to make the representation approximately invariant to small translations of the input. In particular, a pooling function replaces the output of the network at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood, instead the average pooling function includes the average of a rectangular neighborhood.

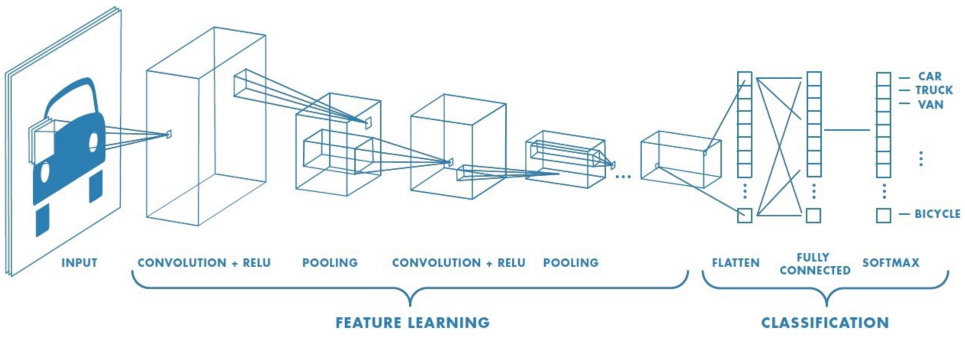


Figure 6: Example of a Convolutional Neural Network.

AlexNet is a well-known CNN designed by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, in fact it was the winning entry in ILSVRC 2012 competition [6]. The AlexNet structure is shown in Figure 7. It has 60 million parameters, 650,000 neurons and its structure consists of 5 convolutional layers and 3 fully-connected layers:

- The first convolutional layer contains 96 kernels (filters) of size 11 $\times$ 11 $\times$ 3 and it is followed by an Overlapping Max Pooling layer of size 3 $\times$ 3 with 2 strides.
- The second convolutional layer contains 256 kernels (filters) of size 5 $\times$ 5 $\times$ 3 and, as the previous layer, it is followed by an Overlapping Max Pooling layer of size 3 $\times$ 3 with 2 strides.
- The third and fourth convolutional layers contain 384 kernels (filters) of size 3 $\times$ 3 $\times$ 3.
- The fifth convolutional layer contains 256 kernels (filters) of size 3 $\times$ 3 $\times$ 3 and it is followed by an Overlapping Max Pooling layer of size 3 $\times$ 3 with 2 strides.
- The last three layers are fully-connected layers and, in particular, the last one (the output layer) contains a number of nodes equals to the number of classes.

In addition, after all the convolutional and fully connected layers, the ReLU is applied as activation function. In order to reduce overfitting, the authors of AlexNet proposed two techniques:

- *data augmentation*: it consists in making minor alterations to the existing dataset so that they are label-preserving transformation such as flips, translations and rotations in order to increase the dataset size.
- *dropout*: it consists of setting to zero the output of each hidden neuron with probability 0.5. In this way, every time an input is presented, the neural network samples a different architecture and the neurons that are "dropped out" in this way do not contribute neither to the forward phase nor to backpropagation.

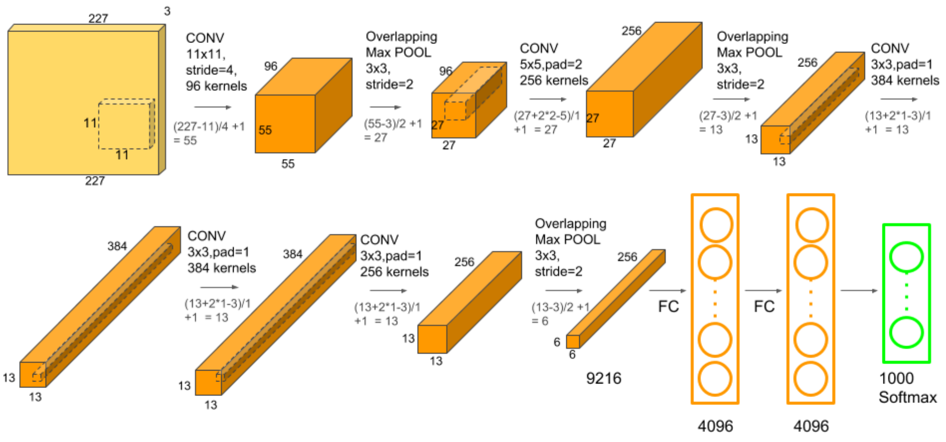


Figure 7: AlexNet structure.

The AlexNet CNN has been implemented through the `AlexNet()` user-defined function using the Keras Sequential model APIs, which allow to build a CNN model as a linear stack of layers. Moreover, the image classification for the first pipeline is done using the `run_alexnet()` function. In details, this function receives in input the training and test sets (represented as two DataFrames), the list of class labels and performs the following steps:

- Creates three instances of `ImageDataGenerator` that represent the training, validation and test sets. The aim of `ImageDataGenerator` class is to generate batches of tensor image data with the possibility of having real-time data augmentation (including random rotations, resizing, shearing, etc.). Considering the validation and test generators, they just take a batch of images and rescale each pixel value from [0, 255] to [0.0, 1.0] without applying any transformations. Instead, regarding the training ImageDataGenerator, if the `adv_preproc` input parameter is set to `True`, some random transformations are applied (horizontal and vertical shift, zooming, rotation, horizontal flipping and brightness shifting), otherwise pixels of training images are only rescaled as for the validation and test generators.

- For each generator, uses the `flow_from_dataframe()` method that takes one of the dataframes received in input and the path of a directory containing the images, and returns an iterator to generate batches of augmented/normalized data. The `flow_from_dataframe()` method has other several useful parameters like `x_col`, which represents the column in the dataframe that contains the image filenames, `y_col`, which indicates the column in the dataframe that has the target data, `batch_size`, representing the number of images in each batch, and `target_size`, a tuple of integers in the form (height, width) that represents the dimensions to which all images found will be resized.

- Builds the AlexNet model and trains it using the `fit_generator()` method which receives in input the training generator along with other useful parameters such as `batch_size`, which determines the number of samples in each batch, and `steps_per_epoch`, representing the number of batch iterations before a training epoch is considered finished. It is important to recall that the training of the network is based on the mini-batch approach: instead of training the network over all the training samples, we divide the training data in small sets called batches and we train the neural network over all those batches, one batch at time. The process of training the network (that actually consists in the backpropagation algorithm) over all the training samples defines an epoch and it is repeated `num_epoch` time. Each epoch consists in `steps_per_epoch` iterations (backpropagations) and each iteration uses `batch_size` images. In addition, at the end of each epoch, we use the `Metrics` class in order to evaluate the performance of the model on-the-fly (i.e., during the training) by using the validation generator.

- Gets the predicted labels using `predict_generator()` method of the AlexNet model and evaluates generalization performance over the test set by using `accuracy_score()`, `f1_score()`, `precision_score()`, `recall_score()` and `classification_report()` methods of *scikit-learn* library.

2.4 Pipeline #2: Classification using the SVM model based on ABCDT features

2.4.1 Segmentation

At the beginning of the second pipeline, preprocessed images contained both in the training and test sets are segmented using the `images_segmentation()` function. In particular, the segmentation process consists in the following steps:

1. *Find an elevation map using the Sobel gradient of the image:*
   The Sobel filter is used in image processing and computer vision to detect edges and uses two 3×3 kernels which are convolved with the original image to calculate an approximations of the gradient of the image intensity function (that is two derivatives - one for horizontal changes, and one for vertical changes).

2. *Find markers of the background and the lesion based on the extreme parts of the histogram of gray values:*
   Markers are found using the threshold obtained through the IsoData filter and they are used to separate the background from the lesion region. IsoData is an histogram-based threshold, also known as Ridler-Calvard method or inter-means, based on an iterative algorithm that divides the image into two regions, the target object (foreground) and background [16]. At the beginning, the algorithm computes the histogram of the grayscale image and then set an initial threshold. At this point the algorithm iterates over the threshold value until it does not change by performing the following two steps:

   - compute the mean of all the pixels whose gray level is equal or less than the threshold and the mean of all the pixels whose gray level is greater than the threshold.
   - update the threshold to the average of the two means, that is:

   $$threshold = \frac{mean(image <= threshold) + mean(image > threshold)}{2}$$

   The Ridler-Calvard method can be regarded as the k-means algorithm applied to a gray-level histogram. Another important point is that, although iterative algorithms (as k-means) require a good starting point, the Ridler-Calvard is powerful because any initial value (also inadequate) will converge to the same solution, because the computation is one-dimensional [16].

3. *Use the watershed transform to fill regions of the elevation map starting from the markers determined in the previous step:*
   Watershed is one of the main algorithms used for segmentation, that

is, for separating different objects/regions in an image [17, 18]. The main idea behind watershed is that any grayscale image can be viewed as a topographic surface/map where high intensity denotes peaks and hills while low intensity denotes valleys. The algorithm starts by filling every isolated valleys (local minima) with different colored water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colors, will start to merge. To avoid that, the algorithm builds barriers in the locations where water merges. It continues the work of filling water and building barriers until all the peaks are under water. Then, the created barriers represent the segmentation result, that is they are the contours of segmented regions. However this approach can led to oversegmentation (especially for medical images) due to noise or any other irregularities in the image. For these reasons, many computer vision libraries like *OpenCV* and *scikit-image* implement a *marker-based watershed* algorithm where the user can specify which are all valley points that have to be merged and which are not. So, in practice we give different labels to the regions we know, for example the regions that represent the background, the ones that represent objects with one color, while the regions which we are not sure of anything are labeled with 0. This is our marker. Then, we apply the watershed algorithm that will update the marker image with the given labels, and the boundaries of detected objects will have a value of -1. Another possibility to find markers is to use the local minima of the gradient of the image.

4. *Improve segmentation:*
   This is achieved through the following 3 steps:

   (a) Fill small holes in the found regions using the `binary_fill_holes()` function of `scipy.ndimage` module.

   (b) Remove small objects that have an area less than 800 pixels using the `remove_small_objects()` function of `skimage.morphology` module. This is useful to exclude some regions that does not represent a lesion.

   (c) Clear regions connected to the image borders using `clear_border()` function of `skimage.segmentation` module. This operation is very useful when there are contour regions that have a big area and so they can be exchanged with the lesion. However, this can also create some issues when the lesion region is connected to the image borders. In order to (try to) overcome this issue, we use a (simple) lesion identification algorithm.

5. *Apply the connected components labeling algorithm:*
   It uses the `label()` method of `skimage.morphology` module in order

to assign the same label to all the pixels that are in the same region. In this way different regions obtained after segmentation are labeled using distinct labels.

6. *Apply the lesion identification algorithm:*
   The basic idea behind the lesion identification algorithm is to find the region representing the target lesion among the ones obtained from the previous steps, and this is achieved by considering the area (the number of pixels of the region) and extent (the ratio of pixels in the region to pixels in the total bounding box) features of each region.
   In fact, the basic assumption is that the region representing the lesion is the one having the greatest area (if the image has been properly filtered and segmented). Since there could be regions on the boundaries with a big area, they could be potentially chosen instead of the lesion region. For this reason, all the contour regions are excluded at step 4. This solution works in most cases, however if the lesion is on the boundaries it is excluded and so either we haven't any label (if the lesion is the only region/object detected after the segmentation) or we have (usually) small regions that can be confused with the lesion region. In order to (try to) solve both the above issues, the lesion identification algorithm works as follow: at the beginning it checks whether there is at least one region, it finds the largest region and then it checks if that region has an area of at least 1200 pixels. If so, the algorithm selects it as the target region, otherwise the lesion region may have been excluded and, in this case, the algorithm backtracks to the original segmented image, applies again the connected components labeling algorithm and extracts the new region properties. Finally, in order to choose the target region, we use area and extent features by checking, among the three largest regions (sorted in descending order, i.e. from the one having the largest area), the first one that has an extent greater than 0.5.
   Another possible approach could be to select as the target region the one having the largest extent among the three largest regions found (this solution is commented in the source code).

Finally, the function plots the original image in which the contours of all the segmented regions are highlighted, and the original image in which only the lesion region is highlighted. In addition, this function returns the detected regions of training and test images as two distinct Python dictionaries which will be used in the feature extraction phase.

2.4.2 Feature extraction

In order to classify images, we need to extract from them some of the most representative features and this is done through the `features_extraction()`

function that is applied both to training and test images.
Conventional diagnostic features are related to asymmetry, border irregularity, color variegation and diameter of skin lesions which is called ABCD analysis [2, 3]. This work presents an extension of the ABCD analysis called ABCDT, where the last T stands for texture. The ABCDT analysis has been performed by using the following parameters:

- **Asymmetry:** Asymmetry of the lesion is characterized through the asymmetric index (AI) and eccentricity. In order to obtain the former parameter, AI, we first find the difference (the non-overlapping region) between the lesion image and its horizontal flip, and the one between the lesion image and its vertical flip. Then, we compute the ratio between these differences and the lesion area, and finally we take their average. AI can be expressed with the following formula:

  $$AI = \frac{\frac{(image\_area - image\_hflip\_area)}{image\_area} + \frac{(image\_area - image\_vflip\_area)}{image\_area}}{2}$$

  The latter represents the eccentricity of the ellipse that has the same second-moments as the region. By definition, eccentricity is the ratio of the focal distance (distance between focal points) over the major axis length.

- **Border irregularity:** Border irregularity is represented through the so-called compact index (CI), that is computed using the following formula [3]:

  $$CI = \frac{P^2}{4\pi A}$$

  where P is the image perimeter and A is the image area.

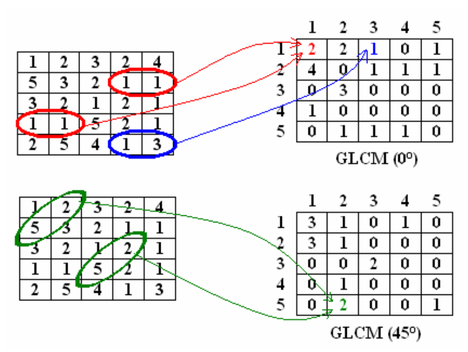
- **Color variegation:** Color variegation is quantified by the normalized standard deviation of red, green and blue components of the lesion [3]. They are expressed by the following formulas:

  $$C_r = \sigma_r/M_r, \quad C_g = \sigma_g/M_g, \quad C_b = \sigma_b/M_b.$$

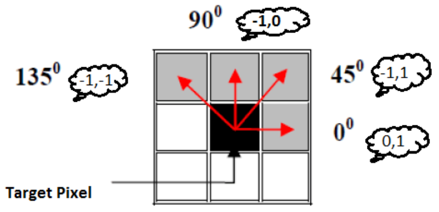
  where $\sigma_r, \sigma_g, \sigma_b$ are respectively the standard deviations of red, green and blue components of lesion area, while M_r, M_g, M_b are the maximum values of red, green and blue components in lesion region. The normalization process is important because objects with higher normal skin pigmentation would also tend to show higher pigmentation in the lesion itself and this needs to be accounted for in order to make any comparison between cases.

- **Diameter:** The diameter of the lesion is computed as the diameter of a circle having the same area as the lesion region.

- **Texture:** Texture defines the consistency of patterns and colors in an image. To classify objects in an image based on texture, we have to look for the consistent spread of patterns and colors in the object's surface.
  One of the main methods to characterize an image based on texture is the so-called *Haralick Texture* and it is performed through the computation of the *Gray Level Co-occurrence Matrix (GLCM)* [19]. The main concept behind GLCM is adjacency of pixels in the image: it looks for pairs of adjacent pixel values that occur in an image and keeps recording it over the entire image. There are four types of adjacency and hence four GLCM matrices can be constructed from a single image: Left-to-Right, Top-to-Bottom, Top Left-to-Bottom Right, Top Right-to-Bottom Left.



(a) Example of GLCM.



(b) Types of adjacency.

Figure 8: Gray Level Co-occurrence Matrix (GLCM).

  In this work, the GLCM matrix is calculated through the `greycomatrix()` method of the `skimage.feature` module. In particular, among its parameters, we set `distances` to `[1]`, which means that the adjacency distance for each pair of pixels is set to 1, and `angles` is set to `[0, 40°, 90°, 135°]` so that we consider all the possible types of adjacency.
  From the GCLM we extract some of the main texture properties that will be used by our classifier:

  - *Correlation:* It measures the joint probability occurrence of the specified pixel pairs and it is mainly used to describe the details of relevant elements in each row and column in the process of segmentation. Correlation is computed using the following formula:

$$\sum_{x,y=0}^{levels-1} GLCM(x,y)\left[\frac{(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right]$$

– *Homogeneity:* It measures the closeness of the distribution of elements in the GLCM with respect to the GLCM diagonal. It is computed as:

$$\sum_{x,y=0}^{levels-1} \frac{GLCM(x,y)}{1+(x-y)^2}$$

– *Energy:* It is computed as the square root of the sum of squared elements in the GLCM. It is also known as uniformity and is expressed through the following formula:

$$\sqrt{\sum_{x,y=0}^{levels-1} GLCM(i,j)^2}$$

– *Contrast:* It measures the local variations in the gray-level co-occurrence matrix and is expressed as:

$$\sum_{x,y=0}^{levels-1} GLCM(i,j)(i-j)^2$$

All the above statistical measures have been computed over the GLCM using the `greycoprops()` method of the `skimage.feature` module.
Finally, the `features_extraction()` function returns a Python dictionary containing, for each segmented lesion, the list of all extracted features if `out_df` is set (by default) to False, otherwise the function constructs and returns a Pandas `DataFrame`, a two-dimensional tabular data structure where each row contains all the features related to a specific lesion. The main advantage of using a `DataFrame` is that it provides multiple methods for convenient data filtering, statistical analysis and fast data access.

2.4.3 Classification

The last step of the second pipeline consists in the image classification using the SVM classifier. It is implemented through the `svm.SVC` class of *scikit-learn* library. Support vector machines are a particularly powerful and flexible class of supervised algorithms for both classification and regression [20]. They are an example of such a *maximum margin estimator* in fact their main goal is to construct a hyperplane (a line in a 2D space or a plane in a 3D space) in a N-dimensional space that has the largest distance to the nearest training data points of any class (also called functional margin), because in general the larger the margin, the higher the generalization power of the classifier. The samples/points closest to the separating hyperplane are called the support vectors. In summary, SVM solves an optimization problem such that:

1. Support vectors have the greatest possible distance from the functional margin (i.e., the separating hyperplane).

2. The classes lie on different sides of the maximum-margin hyperplane.

In case of a binary linear classifier, the optimization problem is equivalent to maximizing the Geometric Margin (γ) shown in the following equation:

$$\gamma = \min_{i...N} \frac{\left(\boldsymbol{w^T x_i} + \boldsymbol{b}\right) y_i}{||\boldsymbol{x}||}$$

where $\mathbf{x_i}$ is a training sample, y_i is its target value, which can assume two values (1 and -1) for a binary classifier, and the separating hyperplane is parameterized by $\mathbf{w}$ and $\mathbf{b}$.

One of the most interesting features of SVMs is that if the training data are not linearly separable, instead of fitting a nonlinear function, data are firstly mapped to a new space by using a non-linear basis function, also known as *kernel.* Through the use of a kernel function (gaussian, sigmoid, radial basis function, polynomial and so on), we can convert the optimization problem to a form whose complexity depends on N, the number of training instances, and not on the number of dimensions, because in general the new space has many more dimensions than the original space.

Moreover, in order to tune the hyperparameters of the SVC model (or put in other words, in order to find the best combination/values of the hyperparameters), we use the `GridSearchCV` class of *scikit-learn* library. The grid search method provided by `GridSearchCV` exhaustively generates candidates from a grid of hyperparameter values specified through `param_grid` in order to optimize them through cross-validation. After finding the best values for the hyperparameters, they are used to re-fit the model so that it can provide the best generalization over the test data.

3 Obtained results

This section described the results obtained from the two pipelines and the performance evaluation adopted for image classification. Evaluating performance is a core part of building an effective machine learning model since it allows to analyze the generalization power of the model itself. In order to evaluate performance of the classifiers used in the two pipelines, this project adopts the following performance metrics:

- *Accuracy:* it is defined as $\frac{TP+TN}{TP+TN+FP+FN}$ and in our context represents the number of images that are correctly classified among all the test images.

- *Precision:* it is defined as $\frac{TP}{TP+FP}$ and represents the ability of the classifier to avoid false positives. Since we have a multi-class classification problem, the `average` parameter of the `precision_score()` method is set to `weighted` in order to compute a weighted precision (the `precision_score()` method calculates the precision for each label, and find their average weighted by support, that is the number of true instances for each label).
- *Recall:* it is defined as $\frac{TP}{TP+FN}$ and represents the ability of the classifier to avoid false negatives. Since we have a multi-class classification problem, the `average` parameter of the `recall_score()` method is set to `weighted` in order to compute a weighted recall (the `recall_score()` method computes the recall for each label, and find their average weighted by support).
- *F1-score:* it is defined as $2 \cdot \frac{precision \cdot recall}{precision + recall}$ and can be interpreted as a weighted average of precision and recall. As for precision and recall, the `average` parameter of the `f1_score()` method is set to `weighted` in order to compute a weighted f1-score (the `f1_score()` method computes the f1-scores for each label, and find their average weighted by support).

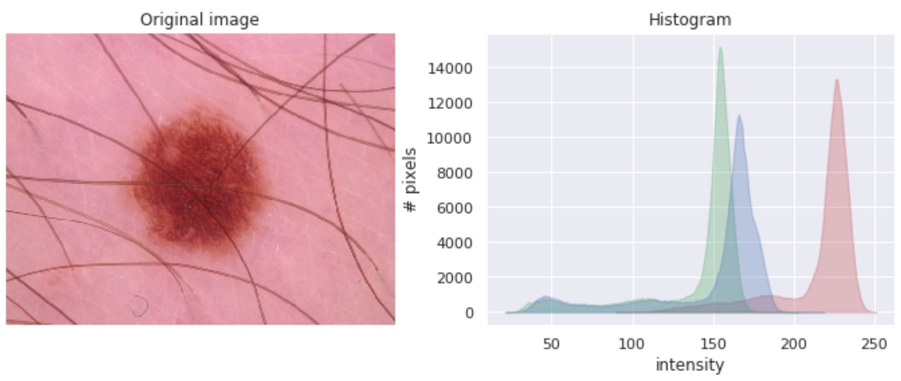
It is important to note that in the above definitions of the metrics, TP indicates the number of true positives, TN indicates the number of true negatives, FP indicates the number of false positives and FN indicates the number of false negatives.

3.1 Initialization

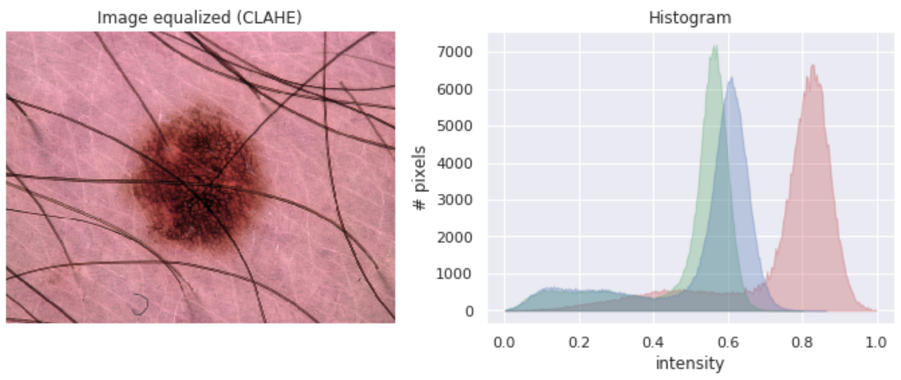
For the initial set-up, we randomly choose 2000 images from the HAM10000 dataset and then the program runs the `build_train_test()` function which creates the `train` and `test` directories containing 1500 images (that is 75% of the chosen images) and 500 images (25% of the chosen images), respectively. As the next step, the program runs the two pipelines, whose results are described in separate subsections in the follow.

3.2 Preprocessing

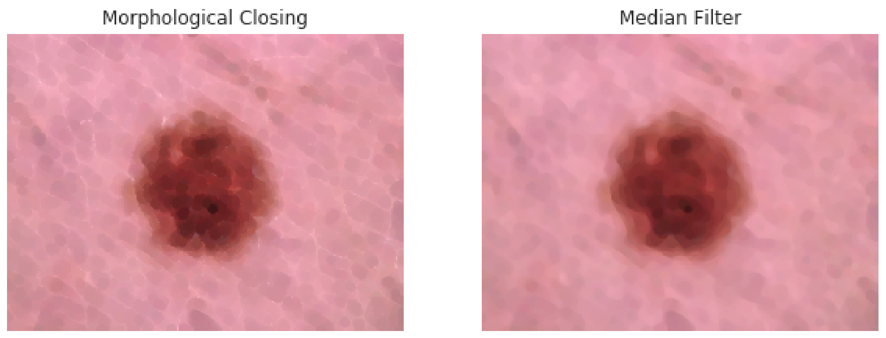
Results related to the preprocessing phase are shown in Figures 9 and 10. In both the figures, by comparing the histogram of the original image (a) with the one of the image preprocessed (b) with the CLAHE technique, we can state that the contrast has been quite improved. In addition, the experimental DullRazor filter (c) obtained through the combination between the morphological closing and the median filter effectively removes hairs in the images. It is important to note that, although details get worse, the median filter is particularly effective at closing gaps and preserving edges (which are are of critical importance for segmentation) on the image obtained after applying the morphological closing.



(a) Skin lesion "ISIC_0030947": original image with its histogram.

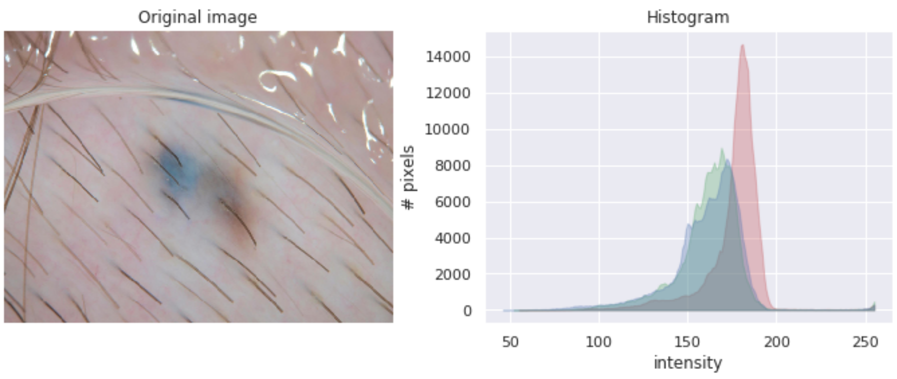


(b) Skin lesion "ISIC_0030947": histogram equalization using CLAHE.

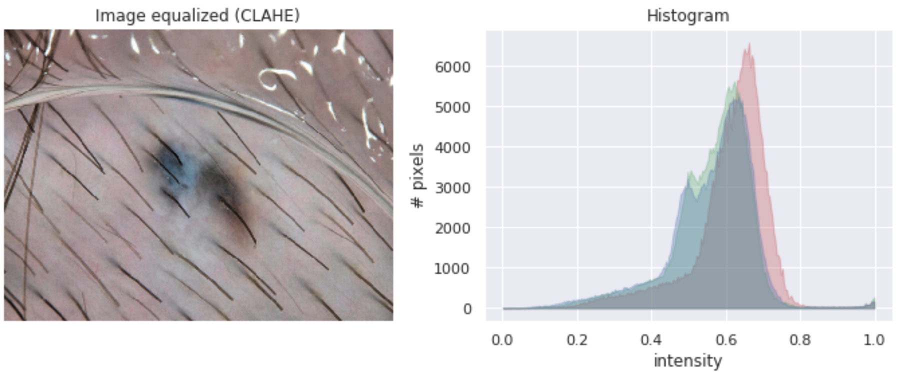


(c) Skin lesion "ISIC_0030947": hair removal simulating DullRazor filter.

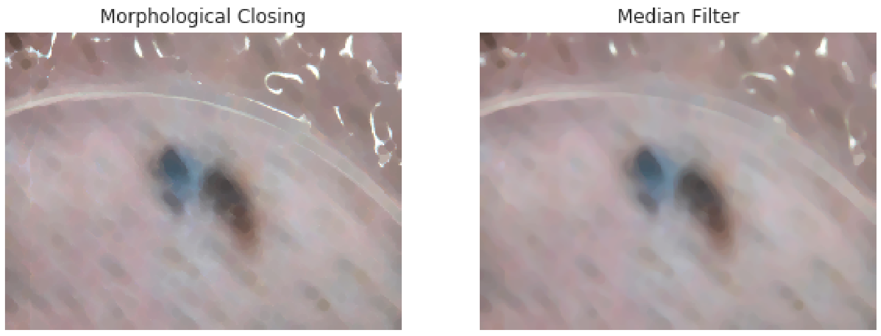
Figure 9: Preprocessing of skin lesion "ISIC_0030947".



(a) Skin lesion "ISIC_0025542": original image with its histogram.



(b) Skin lesion "ISIC_0025542": histogram equalization using CLAHE.

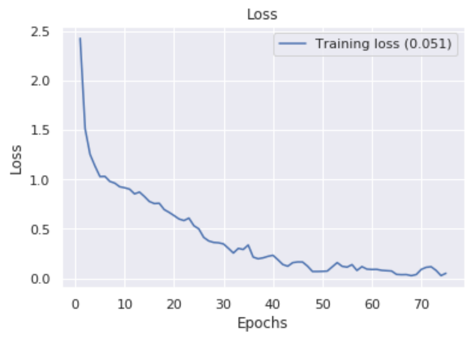


(c) Skin lesion "ISIC_0025542": hair removal simulating DullRazor filter.

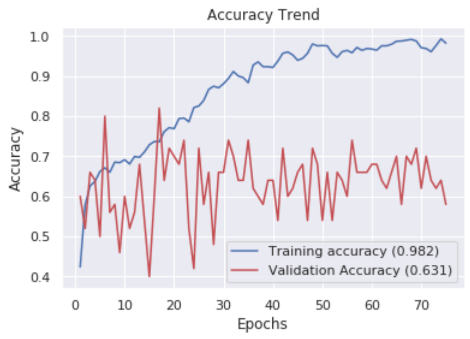
Figure 10: Preprocessing of skin lesion "ISIC_0025542".

3.3 Pipeline #1: Classification using the AlexNet CNN

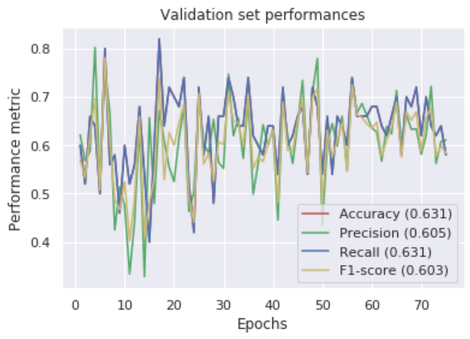
The training images are subdivided into batches of 50 images and the AlexNet CNN is trained (without applying image augmentation) for 75 epochs, thus the number of batch iterations is $1500/50 = 30$. As explained in Section 2, at the end of each epoch we evaluate the performance of the model using the validation generator, whose results are shown in Figure 11. In particular, Figures 11a and 11b show the trend of the loss function and the accuracy (of both the training and validation sets) for each epoch, while Figure 11c represents the trend of the performance metrics (accuracy, precision, recall and f1-score) with respect to each epoch.



(a)



(b)



(c)

Figure 11: Pipeline #1: Performance evaluation over the validation set.

Moreover, the obtained results related to performance evaluation over the test set are shown in the follow:

```
Accuracy: 0.658
F1-score: 0.635
Recall: 0.658
Precision: 0.618
```

To get a more comprehensive view on the obtained results, we also plot the confusion matrix (see Figure 12), where each row represents the instances in a predicted class while each column represents the instances in an actual class.

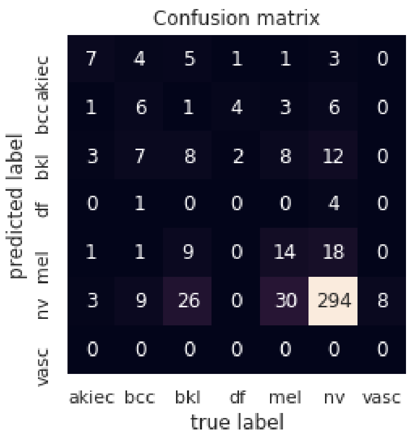
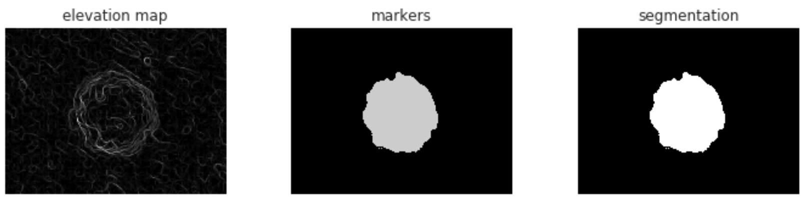


Figure 12: Confusion matrix obtained from Pipeline #1

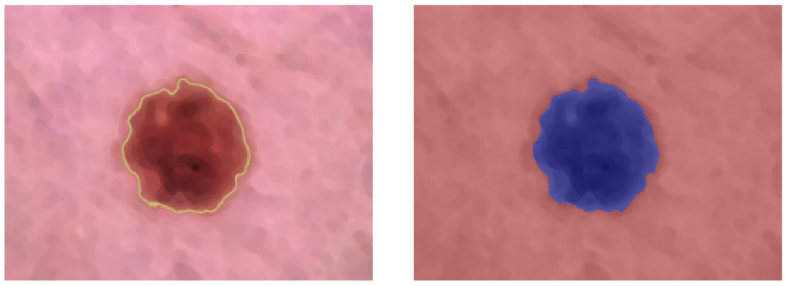
3.4 Pipeline #2: Classification using the SVM model based on ABCDT features

3.4.1 Segmentation

Results related to the segmentation phase of the second pipeline are shown in Figures 13 and 14. As described in Section 2, images of the training and test sets are segmented using the marker-based watershed algorithm (see Figures 13a and 14a), which makes use of the markers and the elevation map computed using the IsoData and Sobel filters, respectively. In addition, as clearly shown in Figure 14a, the segmentation is improved by further steps that consist in filling small holes, removing small objects that have an area less than 800 pixels and removing regions connected to the image contours, if any. The last step of segmentation is identify the region representing the lesion and this is done by the lesion identification algorithm described in Section 2. The results of the lesion identification are shown in Figures 13b and 14b. According to the obtained results, we can state that the segmentation method adopted in this work is quite effective in identifying the regions of interest and finding the one that represents the lesion.

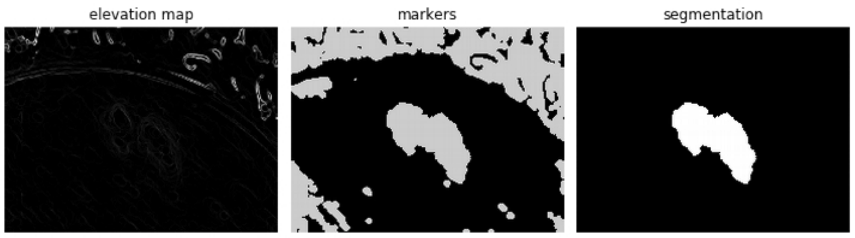


(a) Skin lesion "ISIC_0030947": region-based segmentation.

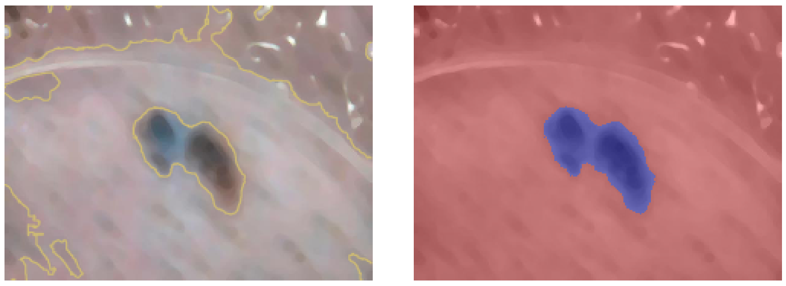


(b) Skin lesion "ISIC_0030947": lesion identification.

Figure 13: Segmentation of skin lesion "ISIC_0030947".



(a) Skin lesion "ISIC_0025542": region-based segmentation.



(b) Skin lesion "ISIC_0025542": lesion identification.

Figure 14: Segmentation of skin lesion "ISIC_0025542".

3.4.2 Feature extraction

Features extracted from the sample images used in the previous subsections are shown in Figures 15 and 17. They are subdivided into 5 categories by following the ABCDT analysis described in Section 2 and are used to train the SVM classifier.

```
-- ASYMMETRY --
Diff area horizontal: 1442.000
Diff area vertical: 2521.000
Asymmetric Index: 0.061
Eccentricity: 0.340

-- BORDER IRREGULARITY --
Compact Index: 1.476

-- COLOR VARIEGATION --
Red Std Deviation: 0.157
Green Std Deviation: 0.226
Blue Std Deviation: 0.229

-- DIAMETER --
Equivalent diameter: 203.212
Diameter (mm): 53.851

-- TEXTURE --
Correlation: 0.999
Homogeneity: 0.735
Energy: 0.108
Contrast: 1.521
```

Figure 15: Feature extraction of skin lesion "ISIC_0030947".

3.4.3 Classification

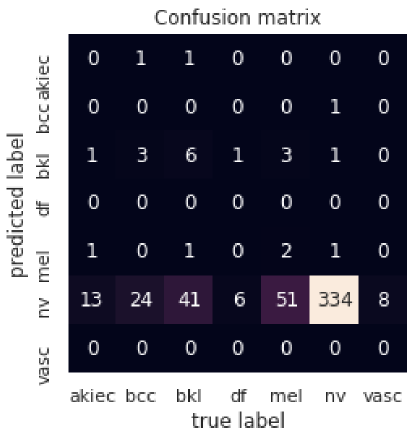
As for images classification using the SVM classifier, the best model (i.e., the best values of the hyperparameters) found using `GridSearchCV` is:

```
SVC(C=10000.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Moreover, the obtained results related to performance evaluation over the test set are:

```
Accuracy: 0.684
F1-score: 0.579
Recall: 0.684
Precision: 0.556
```

As for the first pipeline, at the end of the classification process, the program also plots the confusion matrix (see Figure 16) in order to have a more depth insight over the obtained results and analyze the values of performance metrics for each target class.



predicted \ true	akiec	bcc	bkl	df	mel	nv	vasc
akiec	0	1	1	0	0	0	0
bcc	0	0	0	0	0	1	0
bkl	1	3	6	1	3	1	0
df	0	0	0	0	0	0	0
mel	1	0	1	0	2	1	0
nv	13	24	41	6	51	334	8
vasc	0	0	0	0	0	0	0

Figure 16: Confusion matrix obtained from Pipeline #2

```
-- ASYMMETRY --
Diff area horizontal: 6899.000
Diff area vertical: 8633.000
Asymmetric Index: 0.477
Eccentricity: 0.891

-- BORDER IRREGULARITY --
Compact Index: 2.173

-- COLOR VARIEGATION --
Red Std Deviation: 0.167
Green Std Deviation: 0.159
Blue Std Deviation: 0.156

-- DIAMETER --
Equivalent diameter: 143.942
Diameter (mm): 38.145

-- TEXTURE --
Correlation: 0.996
Homogeneity: 0.737
Energy: 0.084
Contrast: 3.041
```

Figure 17: Feature extraction of skin lesion "ISIC_0025542".

4 Conclusions

In this work we presented an automatic classification system of dermatoscopic images (belonging to the HAM10000 dataset) that is able to run in the Cloud, in fact it has been implemented as a Jupyter notebook that is executed in the Google Colab environment. The project consists of two different pipelines: the first one uses the AlexNet CNN for image classification while in the second one, after segmenting the images and extracting some features from the lesion region using the ABCDT analysis, images are classified using the SVM machine learning model. According to the results described in Section 3, the second pipeline outperforms the first one in terms of accuracy and recall (although with a difference of only 2.6%), while the first pipeline offers better results in terms of f1-score and precision (with a difference of 6.2% as for the precision and 5,6% considering the f1-score). From these results, it can be deduced that through the combination of an optimized classifier, a state-of-the-art segmentation method and a well-tested feature extraction approach it is possible to achieve performance levels that tend to the ones obtained through deep learning techniques. However, it is worth noting that better results could be achieved by adopting some more advanced classification models and by training it over the whole dataset. In fact, the obtained results could be biased by the dataset imbalance and the use of "only" 2000 samples when running the program.

For these reason, as for future developments, we plan to improve this work by using other cutting-edge CNN models such as ResNet-152, DeepLab-v3 or Inception-v4 which have been proved to achieve excellent results in classification and segmentation of medical images [1, 21]. Moreover, other ways to increase the generalization performance are transfer learning and ensemble learning (or ensembling). The former, i.e. transfer learning, is about using the knowledge learned from dealing with one problem to address another problem. In the domain of CNNs, transfer learning often consists in using a CNN as a fixed feature extractor, or by fine-tuning powerful pre-trained deep learning models for the desired task [1, 21]. The main advantages of transfer learning is that, not only it usually allows to achieve better results, but also to overcome some significant issues related to CNNs such as the need of a powerful hardware to support their computational complexity, an appropriate weights initialization and a careful design of the network architecture. The latter, ensemble learning, consists in using multiple CNN models to obtain better predictive performance than could be obtained from any of the constituent models alone [22]. The two main methods to combine multiple models are *multiexpert combination*, in which the base-learners are combined in parallel like in Voting and Stacking, and *multistage combination* that uses a serial approach where the next base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough, as happens in Cascading.

In conclusion, this work aims to be a good starting point in developing advanced skin lesion classification systems capable of democratize and distribute access to health care, in order to save many lives through early diagnosis and significantly reduce health costs.

References

[1] Danilo Barros Mendes and Nilton Correia da Silva. "Skin Lesions Classification Using Convolutional Neural Networks in Clinical Images". In: *CoRR* abs/1812.02316 (2018). URL: http://arxiv.org/abs/1812.02316.

[2] M. Messadi, A. Bessaid, and A. Taleb-Ahmed. "Extraction of specific parameters for skin tumour classification". In: *Journal of Medical Engineering & Technology* 33.4 (2009), pp. 288–295.

[3] Zhishun She, Y. Liu, and A. Damatoa. "Combination of features from skin pattern and ABCD analysis for lesion classification". In: *Skin Research and Technology* 13.1 (2007), pp. 25–33.

[4] Noel C. F. Codella et al. "Skin Lesion Analysis Toward Melanoma Detection 2018: A Challenge Hosted by the International Skin Imaging Collaboration (ISIC)". In: *CoRR* abs/1902.03368 (2019). URL: http://arxiv.org/abs/1902.03368.

[5] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions". In: *Scientific data* 5 (2018), p. 180161.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: 2012.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[9] Stéfan van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[10] François Chollet et al. *Keras*. https://keras.io. 2015.

[11] Tim Lee et al. "Dullrazor®: A software approach to hair removal from images". In: *Computers in Biology and Medicine* 27.6 (1997), pp. 533 –543.

[12] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[13] *Contrast-Limited Adaptive Histogram Equalization (CLAHE). Wikipedia.* Available online: https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Contrast_Limited_AHE.

[14] *Histogram Equalization. Wikipedia.* Available online: https://en.wikipedia.org/wiki/Histogram_equalization.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[16] S. Birchfield. *Image Processing and Analysis.* Cengage Learning, Incorporated, 2017. ISBN: 9781305635739. URL: https://books.google.it/books?id=sybmnQAACAAJ.

[17] Fernand Meyer. "The watershed concept and its use in segmentation: a brief history". In: *arXiv preprint arXiv:1202.0216* (2012).

[18] *Image Segmentation with Watershed Algorithm. OpenCV.* Available online: https://https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html.

[19] Mryka Hall-Beyer. *GLCM Texture: A Tutorial v. 3.0 March 2017.* Tech. rep. 2017.

[20] Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006. URL: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

[21] Navid Alemi Koohbanani et al. "Leveraging Transfer Learning for Segmenting Lesions and their Attributes in Dermoscopy Images". In: *ArXiv* abs/1809.10243 (2018).

[22] Ethem Alpaydin. *Introduction to Machine Learning.* 2nd. The MIT Press, 2010.