Skin Lesion Segmentation and Classification

R. E. Hamadache[a], K. Mouheb[a], S. Lila[a]

[a] *University of Cassino, Cassino , Italy.*

July, 2022

Abstract

Skin cancer counts for one third of all types of cancers and it causes the death of many people every year. Early detection of this disease could increase the chance of survival of the patient but manually detecting cancerous skin lesions is expensive and time-consuming. The aim of this project is to automate the task of skin lesion segmentation and classification based on the ISIC2017 challenge dataset. The project was divided into three main parts. In the first part, advanced image analysis techniques were applied to preprocess and segment skin lesion images. In the second part, machine learning models were trained and compared to find the best set of features and model to perform multi-class skin lesion classification. In the last part, the two tasks of skin lesion segmentation and classification were addressed using deep learning techniques. The results of the project show the potential of automated skin lesion diagnosis systems with deep learning techniques outperforming the traditional image processing and machine learning methods. The results of the former were a jaccard score of 0.7857 for segmentation and a balanced accuracy of 0.7766 for classification compared to 0.6577 jaccard score and 0.6348 balanced accuracy on the test set for the traditional methods.

1. Advanced Image Analysis - Segmentation

Automatic diagnosis of skin lesions requires an accurate segmentation, which is a challenging task due to the different lesion shapes, sizes, colors and borders. In this part of the project, an Otsu's threshold based segmentation approach is used. First, the images are preprocessed to remove some artefacts. Then, the resulting images are segmented through Otsu's thresholding, before being further optimised by post-segmentation steps, consisting of removing other regions that are not the centred lesion. The main steps are shown in [fig.1] and the description of each method is discussed below.

1.1. Pre-segmentation processing

The presence of hairs in skin lesion images, along with other artefacts, makes it hard to segment the lesion. One way to overcome these issues is to first reduce the hairs by inpainting their corresponding pixels with neighboring ones, then mask the artefacts so that they do not interfere in the segmentation.

1.1.1. Hair removal

The following steps aim to detect hairs in the images and reduce them as much as possible. For that, the images are first rescaled such that the longest side of the image is around 500px and the other side is chosen in a way that preserves the aspect ratio. The images are then converted to grayscale. Next, a Blackhat filter with a Cross-shaped Structuring Element (SE) of size 17x17* is applied to them [fig.2-(a)]. Blackhat filters are often used to highlight dark objects over a light background, which is the case here with the hairs over the skin.

Having obtained the hairs' contours with the filter, a binary thresholding is applied to further intensify them [fig.2-(b)], followed by openCV's $'inpaint'$ function to replace hair pixels with pixels similar to the neighboring ones and blend them with the rest of the image [fig.2-(d)]. The resulting images are saved in Drive to be used in the rest of the project.

**Note :* The size of 17*17 was picked after some tests, according to the average hair thickness in the images.

1.1.2. Contrast Enhancement

Improving the images' contrast is essential to obtain better segmentation performances. For that matter, [8]'s texture and color enhancement technique is implemented as follows :

The RGB images [fig.3-(d)] are converted to the CIE-LAB space and the Luminance channel L is extracted and smoothed through a Gaussian filter with the channel's standard deviation [fig.3-(a)]. Then, the Gaussian function [eq.1] [fig.3-(b)] is computed :

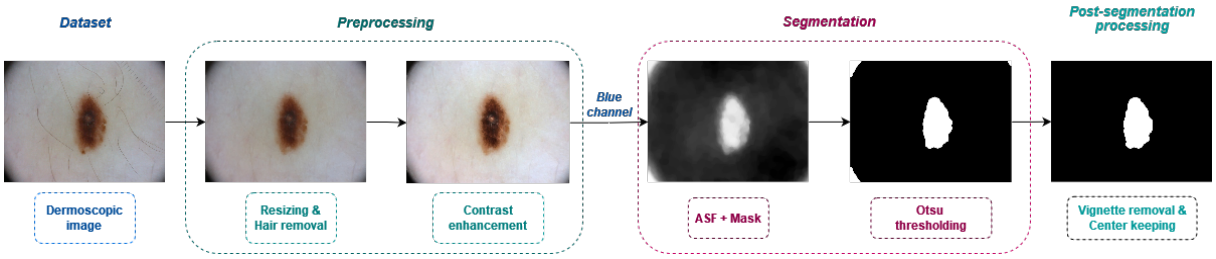


Figure 1: Skin lesion segmentation pipeline

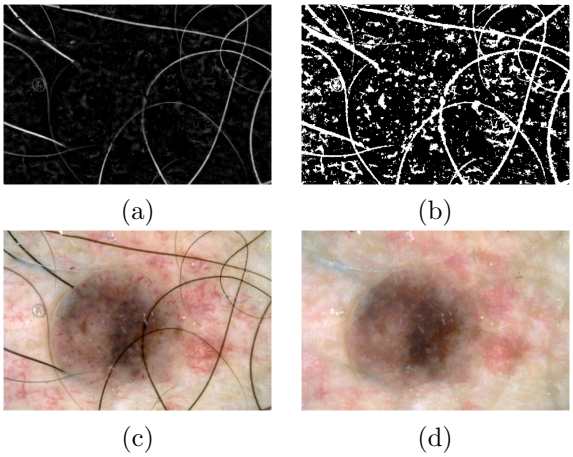


Figure 2: (a) - Blackhat filter image. (b) - Thresholded image. (c) - Original image. (d) - Inpainted image.

$$\rho(u,v,\sigma) = \frac{L(u,v)}{\phi(u,v,\sigma)} - L(u,v) \qquad (1)$$

where σ is the standard deviation of the L channel, $\phi(u,v,\sigma)$ is the Gaussian filtered L channel and L(u,v) is the original L channel.
As the skin lesion pixels have lower intensities, an activation function is used to differentiate the lesion from the healthy skin pixels by looping over all pixels of the image, and whenever a pixel (u,v) gives $\phi(u,v,\sigma) < \rho(u,v,\sigma)$, it will be considered as a lesion pixel $\widehat{L}_{lesion}(u,v)$ [fig.3-(c)].
The color intensities of those lesion pixels are then adjusted by using the 'Multi-Scale Retinex' model [eq.2] :

$$\phi_{Retinex}(u,v) = \frac{\widehat{L}_{lesion}(u,v)}{\widehat{\phi}_L(u,v,\sigma)} \qquad (2)$$

where $\widehat{\phi}_L(u,v,\sigma)$ is the Gaussian filtered L channel in those lesion pixels. At the end, each RGB channel of the original image is multiplied by the $\phi_{Retinex}(u,v)$ in those lesion pixels.
The resulting enhancement can be seen in [fig.3-(e)] :

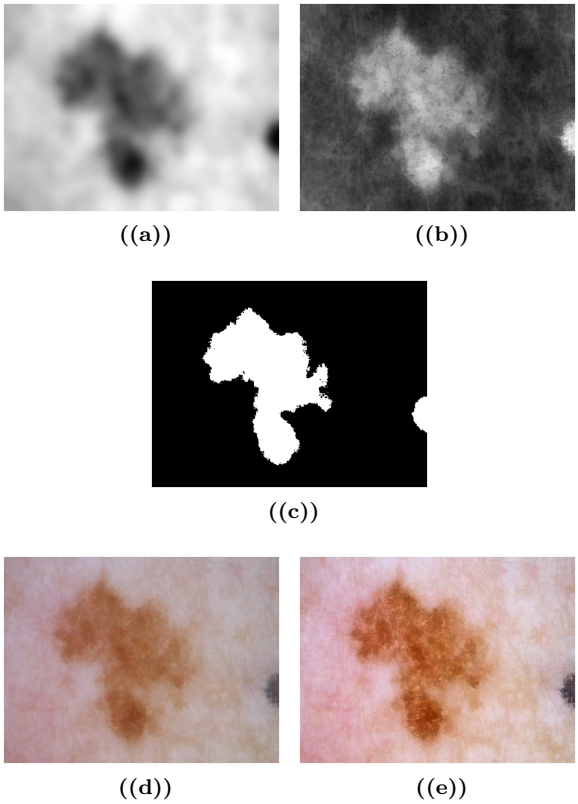
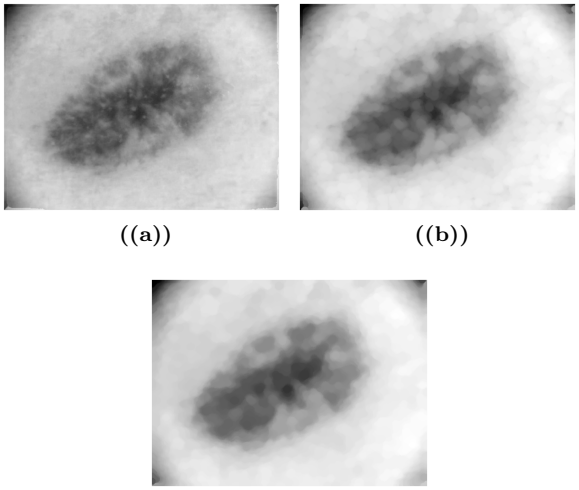


Figure 3: (a) - Gaussian filtered image. (b) - Gaussian function. (c) - Lesion pixels' image. (d) - Original image. (e) - Contrast enhanced image.

1.1.3. Alternating Sequential Filtering

In order to reduce the complexity of the original image and make it more homogeneous, Alternating Sequential Filtering (ASF) is applied on the blue channel* of the enhanced image [fig.4-(a)]. This is done by applying morphological opening [fig.4-(b)], followed by closing [fig.4-(c)], with an Ellipse-shaped SE of size 15x15for both operations.
**Note :* The blue channel of the RGB space was selected because it provides the best information in skin lesion images, as the skin lesion regions usually appear darker and clearer.



((a)) ((b))

((c))

Figure 4: (a) - Blue channel of the original image. (b) - Image after opening. (c) - Resulting ASF image.

1.1.4. Disks Masking

Another artefact present in some dermoscopic images and that can lead to confusion for the segmentation algorithm, is the presence of colorful disks (mainly in the edges of the image) [fig.5-(a)].
Since we are working on the blue channel, the disks that caused the confusion were mainly the blue and the white ones. Therefore, these two sets of disks will be addressed.
To overcome this issue, masking is an effective way for dealing with the disks:
As a first part, the focus will be set on the blue-colored disks. It is noticeable that they appear very dark on the red channel, which is why its inverse (that would be very bright [fig.5-(b)]) will be relied on for the following steps :

- First, a 3-class* multi-Otsu thresholding is applied on the inverse of the red channel. Then, only the 3rd class regions (being the brightest) are retained and followed by a morphological opening to remove any noise (Let $I_{regions}$ be the binary image containing those regions). Each of the aforementioned regions are then labelled by the Connected Components algorithm (Let $Labels_{regions}$ be the matrix having the image's shape and those labels as values [fig.5-(c)]).

- As a second step, an empty rectangle (with thin edges and the same shape as the image) is created and used with a *bitwiseand* operation to get the intersection between this rectangle and $I_{regions}$. This is done to locate all components that lay on the edges of the image so they can be masked afterward (such as vignette, disks).

- Once all components on the edges are found, they get labelled (say $Labels_{inters}$ is the corresponding matrix [fig.5-(d)]) and their coordinates are computed.

- Then, by looping over all the aforementioned coordinates, the value of $Labels^k_{regions}$ is checked in the coordinate k, and whenever it is equal to the intersection's label $Labels^k_{inters}$ (meaning that the corresponding label belongs to a region on the edge), the value of that label will be replaced by 255.

- By keeping only the pixels whose values are 255, a binary mask is obtained [fig.5-(j)] where the components on the edges (coming from the intersection with the rectangle) have high intensities, and all the rest is black. The result of this step is the mask of the blue disks.

For the second part, the focus will be set on the white-colored disks that appear the brightest on the blue channel [fig.5-(f)]. The same previous steps are followed to obtain the corresponding mask for the white disks [fig.5-(k)].
By adding both masks together to combine them and after applying closing to reduce holes, a final binary mask $Mask_{artefacts}$ is obtained [fig.5-(l)], containing all components of the segmented image that lay on the edges.
If the obtained binary mask is empty (in case there is neither a vignette nor objects/disk on the edges), the full ASF image (I_{ASF}) is returned for the segmentation.
**Note :* The choice of 3 classes is due to the possibility of having 2 disks with different colors within the skin lesion image, in which case it will be easier to differentiate between them, or just between the lesion, skin and artefact if present.

1.2. Segmentation

After the disk masking step, the image I_{ASF} is negated to make the lesion appear bright [fig.6-(a)] then it is masked by $Mask_{artefacts}$ from the previous steps. This is achieved using the 'ma.masked_array' function from Numpy, such that when an element of the mask is True (corresponding to the artefacts' pixels), the corresponding element of the image is set as invalid (values replaced by '–') [fig.6-(e)]. As a result, those regions will not be considered during computations, which will help in differentiating only between the intensities of the skin and the lesion without having other objects interfering in the histogram calculation.
With the obtained masked image, a binary Otsu

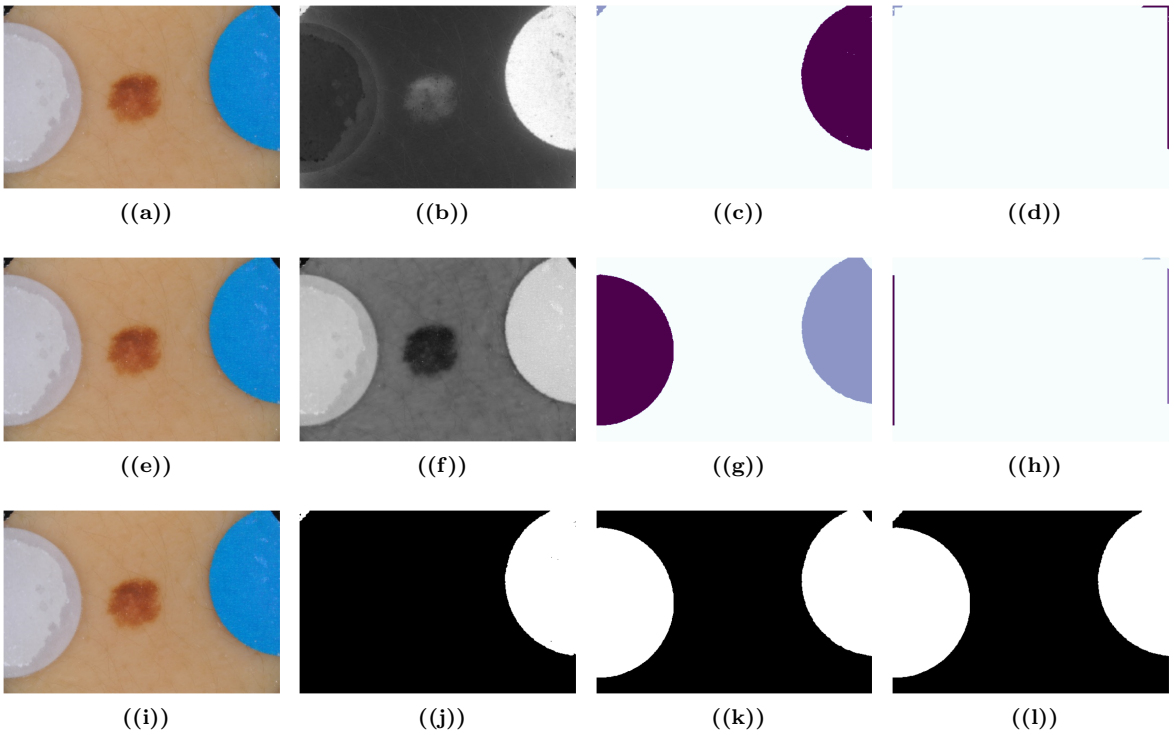


Figure 5: (a),(e),(i) - Original image. (b) - Inverted Red channel. (c) - Labelled regions. (d) - Labelled intersections. (f) - Blue channel. (g) - Labelled regions. (h) - Labelled intersections. (j) - Mask for regions 1. (k) - Mask for regions 2. (l) - Final mask.

is applied, followed by a morphological opening to remove any noise [fig.6-(d)].
In [fig.6-(b)], a segmentation result without masking is shown, which highlights the importance of masking before applying segmentation.

1.3. Post-segmentation processing

The final step for the segmentation part consists of improving the segmented images by further processing them to remove unwanted pixels or regions from the obtained mask. The following subsections describes the procedures used for that matter.

1.3.1. Vignette removal

After segmentation, the vignette and some other parts would still be present in the binary image [fig.7-(a)], as the masking would only ignore them for computing the threshold values and not while computing the regions. For this matter, post segmentation steps are important and this part will focus on finding components on the corners of the image by following the same concept as before and replacing the thin rectangle with a thin corners mask [fig.7-(b)]. I_{seg} is the obtained binary image [fig.7-(c)].
In case the obtained image I_{seg} is empty (when the corners of the image are merged with the lesion and get removed), the first binary image [fig.7-(a)] will be returned. Otherwise, the next step below will be applied to keep only the most centred and largest area component of the lesion.

1.3.2. Central component

As a final step to post-segmentation, the aim of this part is to extract the real lesion from the segmentation obtained in the previous step [fig.7-(c)], by keeping only the largest area in the centre of the image.
To reach that, contours of the segmented image I_{seg} are found, after applying a morphological closing (with an Ellipe-shaped SE of size 5x5). Then for each contour, the moments are computed and the following information is stored:

- The distances between the center of the contour and the image's center.
- The area of the contour.
- The index of the contour.

After that, the contours are sorted in an ascending order according to their distance, then in case there are more than 1 contour in the segmented image, the first 2 contours (with the smallest distances) are compared and the one with the largest area is kept through its index.

Dataset	Number of images	**Jaccard**	Accuracy	Confusion Matrix
Train set	200	**0.812614**	0.935487	$\begin{bmatrix} 1410766 & 20118 \\ 98187 & 443116 \end{bmatrix}$
Train set	2000	**0.704991**	0.930578	$\begin{bmatrix} 5617164 & 140665 \\ 244160 & 946698 \end{bmatrix}$
Validation set	150	**0.650995**	0.91032	$\begin{bmatrix} 8557184 & 250037 \\ 1344247 & 1769885 \end{bmatrix}$
Test set	600	**0.657758**	0.891059	$\begin{bmatrix} 12227630 & 373826 \\ 1642810 & 3600424 \end{bmatrix}$

Table 1: Segmentation scores using Otsu's approach

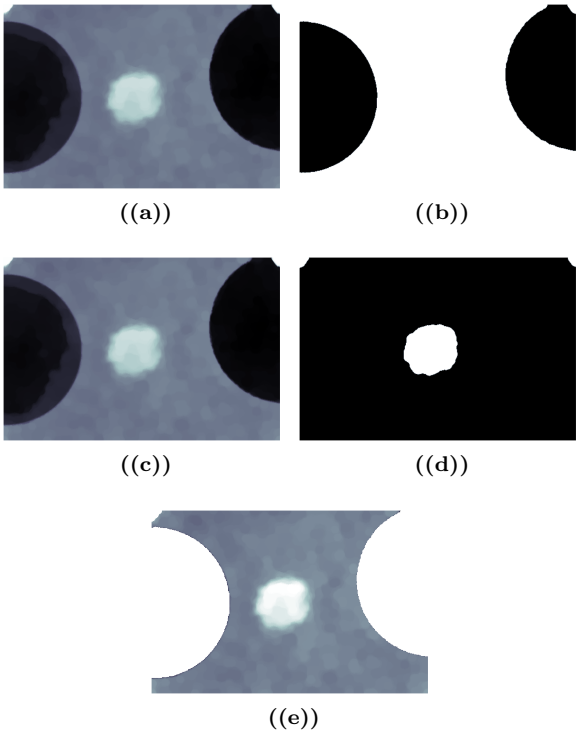


Figure 6: (a)(c) - ASF image (negative). (b) - Segmented image (without mask). (d) - Segmented image (with mask). (e) - Masked negative ASF image.

The final result [fig.7-(d)] is obtained after applying a morphological dilation (Ellipse-shaped SE of size 15x15) to better match the annotated ground truth, which in some cases, are not tight to the lesion but rather include a larger spatial area around it. Some examples are shown in [fig.8].

1.4. Other Segmentation attempts

The following tables [tab.2, tab.3 and tab.4] summarize the obtained segmentation results after trying different segmentation algorithms without masking and while keeping the same post-segmentation approach. The aforementioned algorithms are the

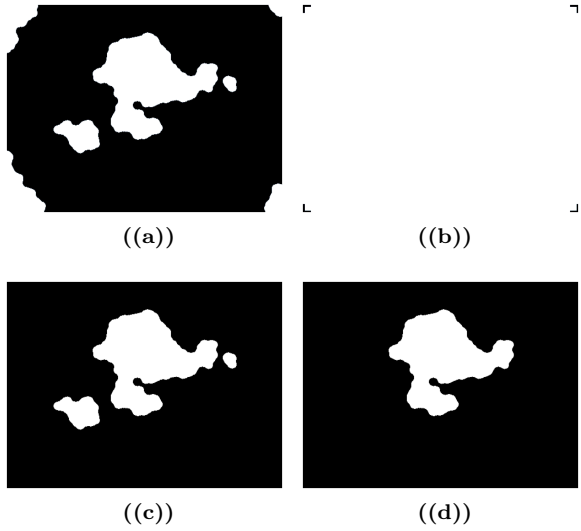


Figure 7: (a) - Segmented image. (b) - Corner intersection with image (inverted for display). (c) - Vignette removed image. (d) - Final segmented image.

following :

- **KMeans Segmentation :** In the segmentation part, instead of using Otsu's thresholding, KMeans was applied on the contrast enhanced images, followed by morphological opening.
- **Watershed Segmentation :** In the same way as for KMeans, the Otsu thresholding was replaced by the Watershed algorithm with distance transform.
- **Segmentation combining Otsu and Retinex :** The Retinex' lesion pixels (supposedly corresponding to the lesion area) were added with Otsu's thresholding segmentation (for a more complete region).

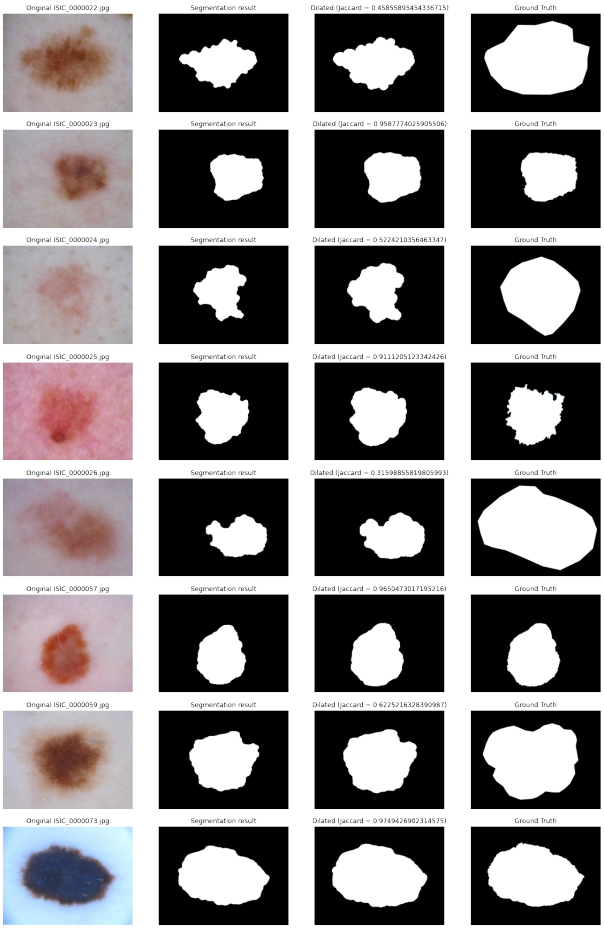


Figure 8: From left to right : Original image - Segmentation result before dilation - Segmentation result after dilation - Ground Truth.

Dataset	Jaccard	Accuracy	Confusion Matrix
Otsu	0.812614	0.935487	1410766 20118 98187 443116
K-Means	0.792088	0.927617	1411787 19097 111106 430198
Watershed	0.77269	0.978839	1412150 18735 127400 413903
Retinex-Otsu	**0.81881**	**0.942582**	1402818 28066 79358 461945

Table 2: Segmentation scores on Train set [200 images]

Dataset	Jaccard	Accuracy	Confusion Matrix
Otsu	**0.657758**	**0.891059**	12227630 373826 1642810 3600424
K-Means	0.604516	0.88042	12288737 312719 1906595 3336639
Watershed	0.654842	0.882602	12076208 525248 1545001 3698234
Retinex-Otsu	0.615912	0.885496	12100245 501211 1564128 3679107

Table 3: Segmentation scores on Test set [600 images]

Dataset	Jaccard	Accuracy	Confusion Matrix
Otsu	**0.650995**	**0.91032**	8557184 250037 1344247 1769885
K-Means	0.512132	0.865307	8313388 493832 1519477 1594655
Watershed	0.585219	0.85568	8035186 772034 1156691 1957441
Retinex-Otsu	0.562184	0.870258	8216103 591117 1518226 1595906

Table 4: Segmentation scores on Validation set [150 images]

1.5. Evaluation

For evaluating the segmentation performance, the resulting binary images are rescaled to their original size before being compared to the Ground Truth images. Then, the Jaccard index is computed, as well as some other metrics. The results for all the methods mentioned are in tables [tab.1], [tab.2], [tab.3] and [tab.4].

1.6. Results

From the previous tables, we can conclude that the best choice is the Otsu approach. Even though, further improvements can be made to include masking for other approaches and further enhance the images for better results.

2. Machine Learning - Classification

This second part of the project is dedicated to the Machine Learning (ML) pipeline for skin lesion multi-class classification and the choice of the most fitted ML model according to the Balanced Accuracy Metric (BMA).
This section will be divided into 3 main parts : Feature extraction and selection, Flat architecture classification and Hierarchical architecture classification.
Feature extraction will be performed on images obtained from the Deep Learning (DL) segmentation process, and for each ML model, the train and validation images are used for training (a total of 2150 images), and the test set (600 images) is used to assess the models performances.

2.1. Feature Extraction and Selection

This section presents the different features extracted from the segmented images and describes the selection procedure to form the feature vector used to fit models for classification.

Feature	Description	n
Shape asymmetry	The ratio between the overlapping area (between the lesion and its horizontal flip) and the lesion area.	1
Compactness	The ratio of the lesion area to the area of a circle having the same perimeter as the lesion : $CI = \frac{4\pi A}{P^2}$	1
Elongation	The ratio of the height and the width of the smallest rectangle containing the lesion.	1
Solidity	The ratio of the lesion area and the area of its convex hull.	1
Color asymmetry H	The difference between the histograms of the top and bottom parts of the lesion after dividing it horizontally.	1
Color asymmetry V	The difference between the histograms of the left and right parts of the lesion after dividing it vertically.	1
Color Variegation	The normalized standard deviation of each of the RGB channels.	3

Table 5: Description of the extracted global features (n : number of features)

2.1.1. Feature extraction

After image segmentation, the feature extraction process is a key part to capture descriptive feature information of each lesion. For this task, 3 types of characteristic features, along with some global ones [3], are focused on : shape, texture and color.

- **Shape features :** Shape features are important for skin lesion classification due to their different shapes in each class. For this reason, Histogram Oriented Gradients (HOG) and Hu moments features are extracted from the blue channel of the lesion images that were previously cropped according to their mask's smallest bounding box.
- **Texture features :** For texture analysis of the images, Local Binary Patterns (LBP), Haralick and Gray Level Co-occurrence Matrix (GLCM) features are extracted (including correlation, homogeneity, contrast, energy and dissimilarity properties for GLCM). The images used correspond to the blue channel of the previously cropped lesion images according to their mask's smallest bounding box.
- **Color features :** Color information is widely used by dermatologists for skin lesion classification. The extraction of color features is very effective, especially that they are invariant to rotation, translation and scaling. For this matter, the cropped lesion images are used to obtain the four central moments from both RGB and HSV spaces : The mean, the standard deviation, the skewness and the variance.
  In addition to that, the color histograms of RGB, HSV and Lab spaces were also extracted.
- **Global features :** The following features will be computed manually, either using the binary segmentation masks, or the latter with the corresponding RGB images. They will include : shape and color asymmetry, compactness, elongation, solidity and color variegation. Their description is shown in [tab.5].

2.1.2. Feature Selection and Concatenation

In this phase, the Principle Component Analysis (PCA) is first performed on the HOG feature vector to reduce its large size (1x3780) to 10% (1x378). Then, different feature combinations are made by vertically concatenating them together to form one feature vector. The different combinations are summarized in [tab.6], and the 7 different feature vectors obtained will be tested with different classifiers to deduce which features are the most relevant for this task.

Due to the class imbalance, we perform data augmentation by using the Random-Over-Sampling method implemented in [4] to obtain a balanced training set. The samples in the minority classes are oversampled with replacement to have the same number of samples as the majority class. The resulting set is then randomly shuffled and a balanced set containing 2150 samples in total is selected to be used for the training procedure. During this procedure the seeds were set to constant values for reproducibility.
The resulting feature vectors are standardized using $StandardScaler()$ function from sci-kit learn which removes the mean and scales to unit variance. Table [tab.7] shows the number of samples in each class in the new training set.

	Features	[Ft. vect.1]	[Ft. vect.2]	[Ft. vect.3]	[Ft. vect.4]	[Ft. vect.5]	[Ft. vect.6]	[Ft. vect.7]
Shape Features	HOG			x	x			
	Hu moments			x	x	x		
Texture Features	LBP	x	x	x		x		
	Haralick	x	x	x		x		
	GLCM		x			x		
Color Features	RGB hist						x	
	HSV hist						x	
	Lab hist						x	
	RGB moments	x	x	x			x	
	HSV moments	x	x	x			x	
Global Features	Shape Asym	x	x	x				x
	Compactness	x	x	x				x
	Elongation	x	x	x				x
	Solidity	x	x	x				x
	Color Asym H	x	x	x				x
	Color Asym V	x	x	x				x
	Color Variegation	x	x	x				x
	Feature vect. size	1x72	1x92	1x457	1x385	1x59	1x2319	1x9

Table 6: Concatenated feature combinations

	Before sampling	After sampling	Final balance
Num. class 0	1449	1449	727
Num. class 1	404	1449	713
Num. class 2	296	1449	710
Total number	2149	4347	2150

Table 7: Number of samples for each class, before and after sampling & shuffling.

2.2. Classification

In this project, two main architectures for multi-class classification are considered : flat architecture and hierarchical architecture.
In the flat model, different classifiers are trained on each of the obtained feature vectors. The models' parameters are tuned using grid search with 5-fold cross validation method to find the best combination of parameters that maximizes the BMA score. After that, the obtained best models were further used to build an ensemble. Two different ensambling methods were tried, first by stacking some classifiers as base models and using a simple one as a meta classifier, and secondly by relying on the majority voting technique.
As for the hierarchical model, the library '*HiClass*' [7] was used to train some classifiers, whose parameters were tuned manually.
The description of each approach will be detailed in the following subsections.

2.2.1. Flat Architecture

The following ML classifiers were trained: Random Forest (RF), Extra Trees (XT), Support Vector Machine (SVM), Gradient Boosting (GB), XGBoost (XGB), K-Nearest Neighbors (KNN) and Logistic Regression (LR). The next subsections describe the parameters and performances of the different algorithms with respect to BMA, Accuracy, Precision and AUC.

2.2.1.1. Single classifiers

As a first part and for each of the obtained feature vectors, the 7 aforementioned classifiers are trained using the scikit-learn *GridSearchCV* function with 5 folds to find the best parameters.

Random Forest : This first classifier is a meta-estimator that fits a number of decision tree classifiers on data subsamples. Its best parameters are found through the grid search method among the different options :

- 'n_estimators' : [100, 500, 1000, 2000] (the number of trees in the forest),
- 'criterion' : ['entropy','gini'] (the function to measure the quality of a split),
- 'max_depth' : [3, 5, 7, 9, 13] (the maximum depth of the tree).

Extra Trees : This other meta estimator, similar to RF, is used with the same previous parameters to be tuned :

- 'n_estimators' : [100, 500, 1000, 2000],
- 'criterion' : ['entropy','gini'],
- 'max_depth' : [3, 5, 7, 9, 13].

Support Vector Machine : The classifier is declared with the following parameters :

- class_weight='balanced', (to automatically adjust weights inversely proportional to class frequencies in the input data),
- decision_function_shape='ovr' (one-vs-rest decision function),

- gamma='auto',
- kernel = 'rbf'.

Only the coefficient C was used in the grid search in a range(1, 11), as some tests showed that it was the most effective range.
Gradient Boosting : The classifier is declared with some fixed parameters :

- 'subsample = 0.7' (the fraction of samples to be used for fitting the individual base learners),
- 'max_features = sqrt' (the amount of features to consider when looking for the best split).

As for the other parameters, the GridSearchCV function is used to choose the best parameters among the different following options :

- 'n_estimators':[100, 500, 1000, 2000] (the number of boosting stages to perform),
- 'max_depth': [3, 5, 7, 9, 13] (the maximum depth to limit the number of nodes in the tree),
- 'learning_rate': [0.01, 0.1] (to shrink the contribution of each tree by that number).

XGBoost : The classifier is declared with the following parameters :

- objective="multi:softmax" (for multi-class classification),
- num_class=3.

Similarly to GB, the other parameters are chosen, through the GridSearchCV function, among the different options as follows :

- 'n_estimators':[100, 500, 1000, 2000] (the number of boosting stages to perform),
- 'max_depth': [3, 5, 7, 9, 13] (the maximum depth to limit the number of nodes in the tree),
- 'learning_rate': [0.01, 0.1] (to shrink the contribution of each tree by that number).

K-Nearest Neighbors : The classifier's best parameters are chosen through the function GridSearchCV from the following options :

- 'n_neighbors':[5, 10, 15, 20, 25, 30],
- 'weights': ['uniform', 'distance'].

Logistic Regression : For this last classifier, only two parameters were used in the grid search :

- 'C': [1.0, 10.0, 100.0, 1000.0],
- 'solver': ['newton-cg', 'lbfgs', 'liblinear'].

The table [tab.8] summarize the scores obtained with the different classifiers for each feature vector.
Note : PCA with 99% was applied on the feature vector 6 to reduce its dimension to (1x237) before being used in classification training.

We notice that the first feature vector, combining texture, global and some color features, gave the best scores in the test set predictions.
The texture features appear to have the best contribution to the classification performances, followed by the global features. As for the low scores for the shape features, this might be due to the smoothness of DL masks, which do not depict the real shape of the lesions that is usually relied on in classifying the skin lesion classes.

The first feature vector will then be used in the remaining subsections. The best classifier parameters obtained by the grid search for this feature vector are :

- Random Forest : best results achieved when using 'gini' criterion, 1000 estimators and max depth of 13.
- Extra Trees : best results achieved when using 'entropy' criterion, 2000 estimators and max_depth of 13.
- SVM : best results achieved when using C=10.
- Gradient Boosting : best results achieved when using 500 estimators, a learning_rate of 0.1 and a max_depth of 13.
- XGBoost : best results achieved when using 100 estimators, a learning_rate of 0.1 and max_depth of 9.
- K-Nearest Neighbors : best results achieved when using 15 neighbors with the 'distance' weight.
- Logistic Regression : best results achieved when using 'newton-cg' solver and C=1000.

According to the previous results, the best models for this dataset and using the first feature vector are RF, followed by SVM. Their scores and ROC curves are shown in [tab.8] and [fig.9].

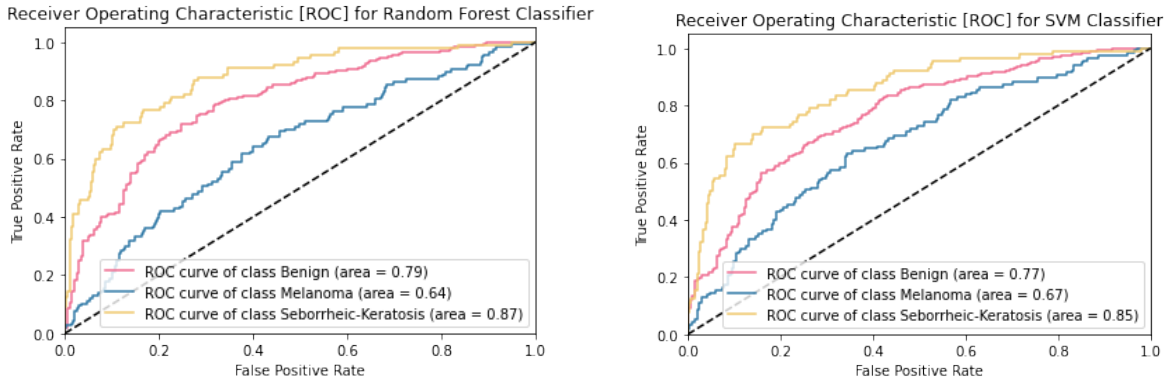


Figure 9: ROC curves of best performing single classifiers (RF and SVM)

2.2.1.2. Stacking classifiers

Stacking aims to combine multiple classification models through a meta-classifier while using cross-validation (5 folds for this case).
In this subsection, the previously obtained classifiers with the best fitted parameters are further manipulated in a stacking strategy. The LR classifier was used as the meta-classifier, and different models were stcaked each time with 'StackingCVClassifier' function.
The best score was given by stacking RF and SVM classifiers, followed by stacking SVM, XT and XGB, which were among the best performing classifiers in the previous subsection. Their scores and ROC curves are shown in [tab.9] and [fig.10].

2.2.1.3. Majority Voting

A last ensemble method was explored, with similar classifier combinations as before, and with the soft voting method, exploiting the probabilities of each class for making the predictions.
The results for each classifier combination is reported in [tab.10].
The best performances were achieved by using [SVM, RF, XT and KNN], followed by using [SVM and RF], and [RF, SVM, XGB, and XT]. Their scores and ROC curves are shown in [tab.10] and [fig.11].

Observations : Amoung the previous flat architecture techniques used to predict skin lesion classes, the majority voting procedure surpassed the other methods, with a BMA score of ***0.630542*** being slightly higher than the stacking one. These scores were achieved based on the single classifiers and might be improved by further tuning their parameters.

2.3. Hierarchical Architecture

In this last section for skin lesion classification, a hierarchical architecture is tested for multi-class classification. It is based on local classifiers that explore the hierarchy of classes in a local perspective [11]. The class predictions in the testing phase are executed in a top-down way (from the highest to the lowest level of the hierarchy), where in each level, the child classes of the previous predicted class are considered.
The used architecture is based on the 'Local Classifier Per Parent Node' approach. It consists of training a traditional flat classifier for each parent class of the class hierarchy to discriminate among their child classes.
For this matter, the first step is to transform the labels into a hierarchical format (as shown in [fig.12]) such that :

- When y_train = 0 → y_train_h receives ['Benign' , '0'],
- When y_train = 1 → y_train_h receives ['Malignant', '1'],
- When y_train = 2 → y_train_h receives ['Malignant', '2'].

Then after preprocessing the train data with the 'standardScaler()' function, the 'LocalClassifierPerParentNode()' function will be used to fit the training data and predict the class labels from the test set.
As the library HiClass [7] does not suggest any grid search option, the parameters for each classifier used will be tuned manually through 'for loops'. The classifiers are RF, XT, SVM, GB, XGB and KNN. The parameters are initially set as in the flat architecture section, and will be modified according to the BMA scores.
Some of the previous feature vectors are used in this part, and the results for each one of them is summarized in [tab.11].
Observation :

- From [tab.11], the best performing classifiers were SVM with feature vector 1, achieving a

	Classifier	Train BMA	Accuracy	Test BMA	Precision
Feature Vector 1 (1x72)	**Random Forest**	**0.850127**	**0.614357**	**0.61404**	**0.57562**
	Extra Trees	0.854661	0.624374	0.589175	0.567975
	SVM	**0.803076**	**0.629382**	**0.607992**	**0.567243**
	GradBoost	0.86692	0.654424	0.585073	0.589156
	XGBoost	0.85696	0.611018	0.591222	0.558948
	KNN	0.780027	0.472454	0.568598	0.530771
	LR	0.688472	0.535893	0.588919	0.521748
Feature Vector 2 (1x92)	**Random Forest**	**0.845047**	**0.594324**	**0.60498**	**0.559723**
	Extra Trees	0.852777	0.602671	0.578686	0.544761
	SVM	0.812812	0.621035	0.58804	0.553049
	GradBoost	0.861458	0.66611	0.602439	0.59754
	XGBoost	0.855953	0.622705	0.60459	0.564845
	KNN	0.78703	0.410684	0.536557	0.500603
	LR	0.691388	0.549249	0.594013	0.522511
Feature Vector 3 (1x457)	**Random Forest**	**0.85085**	**0.644407**	**0.597959**	**0.579612**
	Extra Trees	0.860204	0.672788	0.528208	0.557763
	SVM	0.845647	0.577629	0.481172	0.497581
	GradBoost	0.87364	0.677796	0.539885	0.579504
	XGBoost	0.858667	0.611018	0.529	0.522233
	KNN	0.819189	0.560935	0.469815	0.454352
	LR	0.791235	0.502504	0.478304	0.449785
Feature Vector 4 (1x385)	Random Forest	0.857515	0.649416	0.333636	0.256618
	Extra Trees	0.864214	0.654424	0.333333	0.218141
	SVM	0.835666	0.550918	0.350829	0.361965
	GradBoost	0.862549	0.649416	0.330782	0.218294
	XGBoost	**0.840699**	**0.612688**	**0.393428**	**0.426209**
	KNN	0.75117	0.385643	0.315441	0.326506
	LR	0.762717	0.36394	0.322664	0.329566
Feature Vector 5 (1x59)	**Random Forest**	**0.827854**	**0.609349**	**0.608359**	**0.581194**
	Extra Trees	0.836642	0.607679	0.581816	0.556825
	SVM	0.766321	0.572621	0.564813	0.522904
	GradBoost	0.844249	0.63606	0.551459	0.565557
	XGBoost	0.837973	0.629382	0.58573	0.561254
	KNN	0.775444	0.444073	0.533591	0.49308
	LR	0.647504	0.495826	0.577361	0.511663
Feature Vector 6 (1x2319)	Random Forest	0.839683	0.661102	0.466004	0.474717
	Extra Trees	**0.802996**	**0.664441**	**0.490808**	**0.474151**
	SVM	0.841839	0.657763	0.395533	0.464486
	GradBoost	0.863367	0.651085	0.393841	0.468265
	XGBoost	0.835636	0.594324	0.445994	0.452004
	KNN	0.735554	0.298831	0.410499	0.385295
	LR	0.640839	0.358932	0.461663	0.435702
Feature Vector 7 (1x9)	Random Forest	0.815989	0.567613	0.448102	0.453768
	Extra Trees	**0.788581**	**0.616027**	**0.488738**	**0.489939**
	SVM	0.603995	0.480801	0.439571	0.432474
	GradBoost	0.834477	0.599332	0.412884	0.431212
	XGBoost	0.819014	0.590985	0.459152	0.463856
	KNN	0.750604	0.442404	0.420578	0.417789
	LR	0.52093	0.48414	0.476379	0.458064

Table 8: Single Classifiers - Scores of each classifier for different feature vectors

BMA score of ***0.634842***, followed by XGB (in feature vector 5 and 2).

- The parameters in the training might affect the performances by increasing or decreasing them depending on the classifiers.

Results :
For this classification task, both flat and hierarchical approaches were tested on the ISIC2017 dataset. Different classifiers were used with different combinations of features, and among all the models trained, the hierarchical procedure appears to achieve

SVM	RF	XT	GB	XGB	KNN	Accuracy	Test BMA	Precision
x	x			x		0.63606	0.61426	0.591139
x	x	x				0.649416	0.608505	0.594413
x	x	x		x		0.649416	0.606506	0.592953
x	**x**					**0.657763**	**0.630166**	**0.601439**
x		**x**		**x**		**0.647746**	**0.618213**	**0.600877**
	x	x		x		0.662771	0.618161	0.594625
x		x				0.63773	0.600554	0.579101
x				x		0.647749	0.607944	0.599673
	x	x				0.631052	0.600006	0.581075
	x			x		0.631052	0.603714	0.573536
		x		x		0.642738	0.599108	0.575406
x	x	x			x	0.656093	0.617902	0.595074
x	x	x	x			0.664441	0.594173	0.593534
x			x			0.651085	0.582807	0.581962
	x		x			0.642738	0.581409	0.572552

Table 9: Stacked Classifiers - Scores of each stacking combination

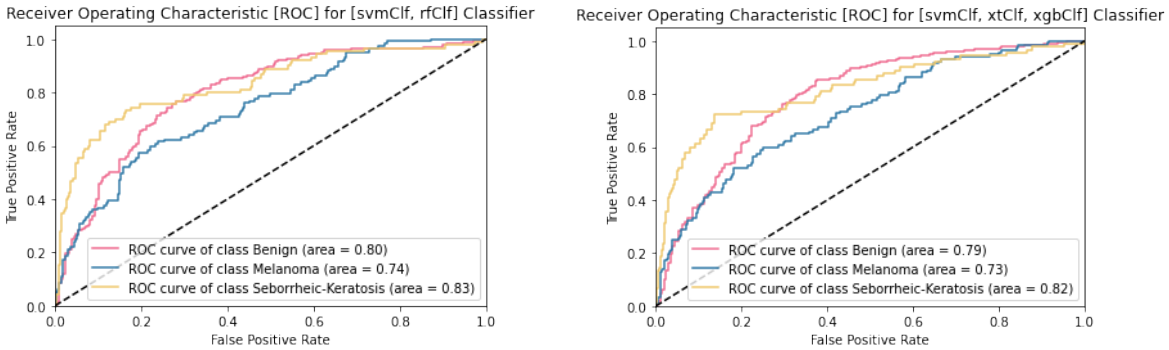


Figure 10: ROC curves of the best performing stacked classifiers

SVM	RF	XT	GB	XGB	KNN	Accuracy	Test BMA	Precision
x	x			x		0.63606	0.595982	0.568874
x	x	x				0.63606	0.596548	0.571297
x	**x**	**x**		**x**		**0.647746**	**0.611639**	**0.588866**
x	**x**					**0.642738**	**0.613651**	**0.579794**
x		x		x		0.642738	0.602237	0.575822
	x	x		x		0.609349	0.591226	0.560185
x		x				0.652755	0.60705	0.58002
x				x		0.634391	0.597131	0.567948
	x	x				0.627713	0.611139	0.582034
	x			x		0.604341	0.587821	0.556495
		x		x		0.612688	0.592072	0.560847
x	**x**	**x**			**x**	**0.629382**	**0.630542**	**0.592956**
x	x	x	x			0.664441	0.599879	0.589975
x			x			0.657763	0.589627	0.581119
	x		x			0.656093	0.590776	0.589055

Table 10: Majority Voting Scores for the different classifier combinations

the most promising results. In fact, the hierarchical method was faster in training and slightly surpassed the other methods in performance, including ensembling ones.

However, the results are still subject to improvement by optimizing the feature vectors and the classifiers' parameters to better fit the data and improve the overall predictions in the test set.

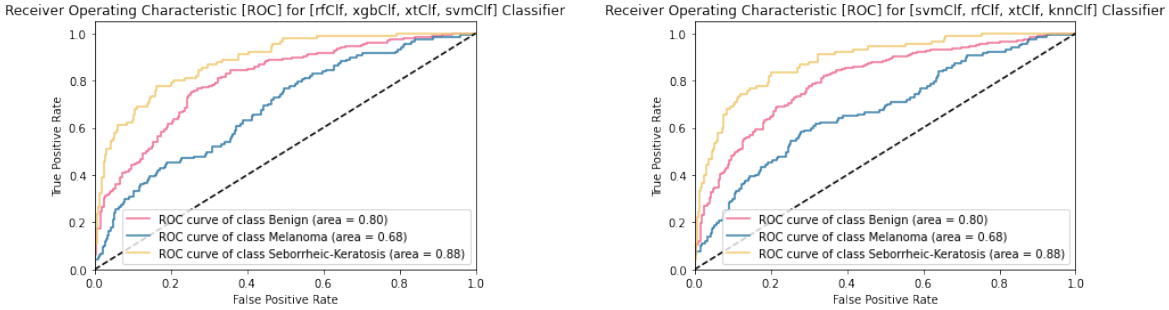


Figure 11: ROC curves of the best performing Majority voting combination

	Classifier	Accuracy	Test BMA	Precision
Feature Vector 1 (1x72)	Random Forest	0.589316	0.627833	0.567467
	Extra Trees	0.602671	0.600961	0.576522
	SVM	**0.624374**	**0.634842**	**0.570158**
	GradBoost	0.629382	0.617985	0.574252
	XGBoost	0.601002	0.620939	0.573582
	KNN	0.416084	0.543132	0.515558
Feature Vector 2 (1x92)	Random Forest	0.594324	0.62924	0.57159
	Extra Trees	0.597663	0.595833	0.558631
	SVM	0.60601	0.624634	0.570029
	GradBoost	0.639399	0.622798	0.585234
	XGBoost	**0.621035**	**0.629999**	**0.569947**
	KNN	0.397329	0.550885	0.517607
Feature Vector 5 (1x59)	Random Forest	0.604341	0.61095	0.588165
	Extra Trees	0.57429	0.578234	0.551007
	SVM	0.612688	0.622907	0.569714
	GradBoost	0.601002	0.619808	0.572181
	XGBoost	**0.624374**	**0.634277**	**0.579313**
	KNN	0.370618	0.524433	0.491056
Feature Vector 7 (1x9)	Random Forest	0.590985	0.477416	0.474131
	Extra Trees	0.524207	0.521035	0.496842
	SVM	0.644407	0.552279	0.52712
	GradBoost	0.542571	0.522696	0.501206
	XGBoost	0.462437	0.491584	0.478883
	KNN	0.445743	0.518754	0.518094

Table 11: Hierarchical architecture - Classification scores

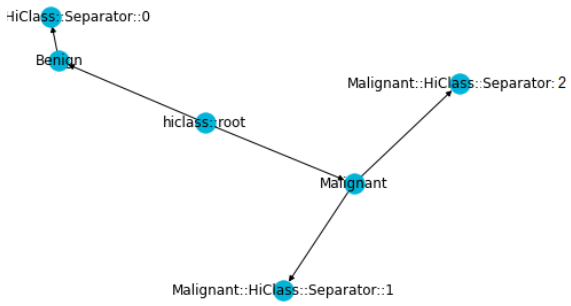


Figure 12: Hierarchical representation of the skin lesion classes

3. Deep Learning

In the deep learning part of the project, PyTorch was used as a framework with Google Colab (16 Gb GPU with High-RAM mode). The deep learning part was divided into two main parts: Segmentation and Classification.

3.1. Segmentation

In this section, the segmentation models, the data pre-processing pipeline, and the training procedures followed in this step will be described. At the end of the section, experimental results will be presented.

3.1.1. Segmentation Models: U-Net

The U-Net model was used for segmentation. To further improve the results, two main mechanisms were tried, Transfer Learning, and Attention. In this section, details are given about the architectures of the models used in this part of the project.

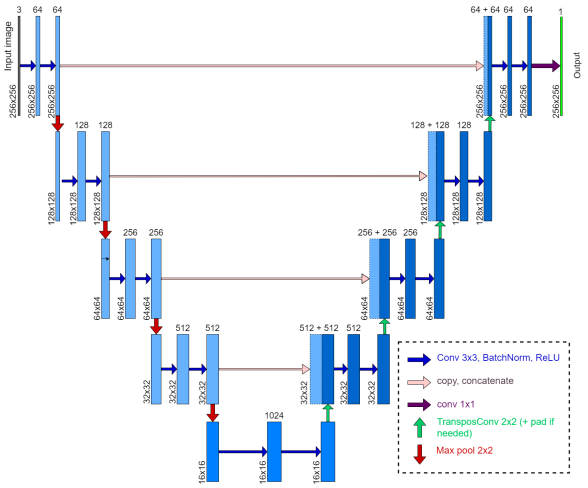


Figure 13: Vanilla U-Net

3.1.1.1. Vanilla U-Net

U-Net [13] is one of the most widely used models for medical image segmentation. The model contains two main parts:

- **Encoder (Contracting) part**: a typical architecture of a convolutional neural network. Thus, known classification CNN architectures such as ResNet and AlexNet can be used as the encoder part of a U-Net. The encoder is used to capture the context of the image. The encoder of the U-Net used in this part consists of a repeated application of a double convolution block followed by a 2x2 max pooling operation with stride 2 to perform down-sampling by a factor of 2. The double convolution block consists of two convolutional layers with kernels of size 3x3, each followed by a batch normalization layer and a rectified linear unit (ReLU). The initial number of channels in the feature maps is set to 64. During each application of the double convolution, the number of feature map channels is doubled to compensate for the loss of spatial resolution due to down sampling. This operation is repeated 5 times. The result of the encoder is a feature map of size 16x16 and 1024 channels (considering an input of size 256x256).
- **Decoder (expansive) part**: a symmetric part to the encoder used for localization. It consists of a repeated application of an up-sampling block. The up-sampling block takes as input the feature maps coming from the preceding decoder level and the feature maps coming via a skip connection from the parallel encoder level. A transposed convolution is applied to up-sample the feature map coming from the preceding decoder layer to double its size while reducing the number of feature maps to half. Different from the original paper, before concatenating, instead of cropping the encoder's feature map, the decoder's feature map is padded with zeros if needed to have the same size as the encoder's feature map. The two feature maps are then concatenated and a double convolution (the same as in the encoder part) is applied to the result of the concatenation.

The output of the decoder level is a feature map of size 256x256 with 64 channels (considering an input of size 256x256). This result passes through a final convolutional layer with a 1x1 filter to map it to a 1-channel result. The latter will later pass through a sigmoid function to map the values of the resulting channel to probabilities in the range [0, 1]. The architecture of this model is visualized in figure 13. As seen from its architecture, U-Net is a *fully convolutional* neural network in the sense that it does not include any fully connected (dense) layers.

3.1.1.2. Attention U-Net: Where to Look for The Skin Lesion

This model has the same architecture as the Vanilla U-net with the following modifications:

- Instead of using transposed convolution, up-sampling with bilinear interpolation followed by a convolutional layer is used at the decoder level to double the size of the feature maps.

- Instead of concatenating the feature maps from the encoder level with the corresponding result at the decoder level, an attention mechanism is used at the level of the skip connection as proposed in [10]. The attention mechanism is summarized below.

Attention Gate: A soft self-attention mechanism is used to reduce false-positive predictions by progressively suppressing feature responses in the background regions without cropping an ROI between network layers. This is achieved by multiplying the input feature map with attention coefficients that give more importance to the activations that are relevant to the task. Additive attention with a sigmoid activation function is used in this architecture. The attention gate takes the coarse feature maps generated at the preceding decoder level as a gating signal **g** and the corresponding feature map coming from the encoder level via the skip connection as an input **x**. Since the feature maps have different numbers of channels, they are linearly mapped to an intermediate space using channel-wise 1x1x1 convolutions followed by a Batch Normalization layer. The results of the linear mapping are then summed and they pass through a ReLU. The attention weights are calculated in the intermediate space using a convolution of kernel size 1 and stride 1 followed by a batch normalization layer and a sigmoid activation function. The output of the attention gate is the input feature map **x** multiplied by the 1 channel attention coefficients. Note that in this case, one attention coefficient is calculated for each *pixel vector*. The architecture of the Attention Gate is shown in figure 14 (a).

The result of the attention gate is concatenated with the feature map coming from the preceding decoder layer. The architecture of Attention U-Net is shown in figure 14 (b).

3.1.1.3. ResNet U-Net

To benefit from Transfer Learning, a U-Net model was created using the pre-trained ResNet18 model (available in *torchvision.models*) as an encoder. The input first passes through the ResNet18 encoder. For each layer of ResNet18, a parallel decoder layer is built. The result of each ResNet18 layer passes through a block of a convolutional layer of kernel size 1x1 and stride 1 followed by a ReLU to add learnable parameters at the level of the skip connection. Each decoder layer takes as input the feature maps coming from the precesing decoder layer and the result of the skip connection coming from the parallel ResNet18 layer. The first input (decoder's feature maps) is up-sampled to double its size while reducing the number of its channels to the half using 2x2 transposed convolution.

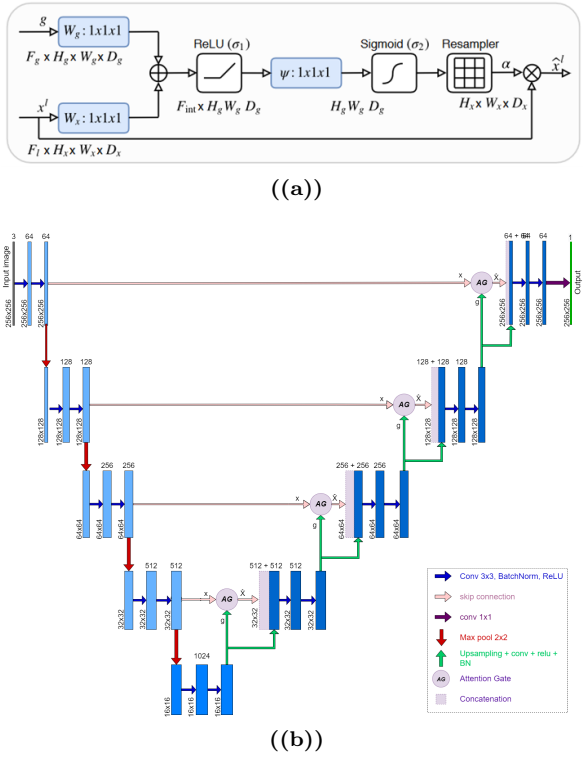


Figure 14: (a) - Attention Gate [10]. (b) - Attention U-Net

The two inputs are then concatenated and the result of the concatenation passes through a double convolution block similarly to the previous models. The encoder was initialized to pretrained ImageNet weights, Kaiming weight initialization was used for the decoder's convolutional layers whereas the batch normalization layers were initialized using constants (1 for the weights and 0 for biases). The decoder's result is then given to a 1x1 convolutional layer to obtain the final 1-channel segmentation output. The architecture of ResNet U-Net is shown in figue 15.

3.1.1.4. Attention ResNet U-Net

Finally, attention gates were added at the level of the skip connections (similarly to Attention U-Net) to the ResNet U-Net model defined in the third architecture. The architecture of Attention ResNet U-Net is described in figure 16.

3.1.2. Data

3.1.2.1. Dataset

The original ISIC2017 dataset was used in this task.

3.1.2.2. Preprocessing

The dataset images have a high resolution and different sizes (from 540x722 to 4000x6000 pixels).

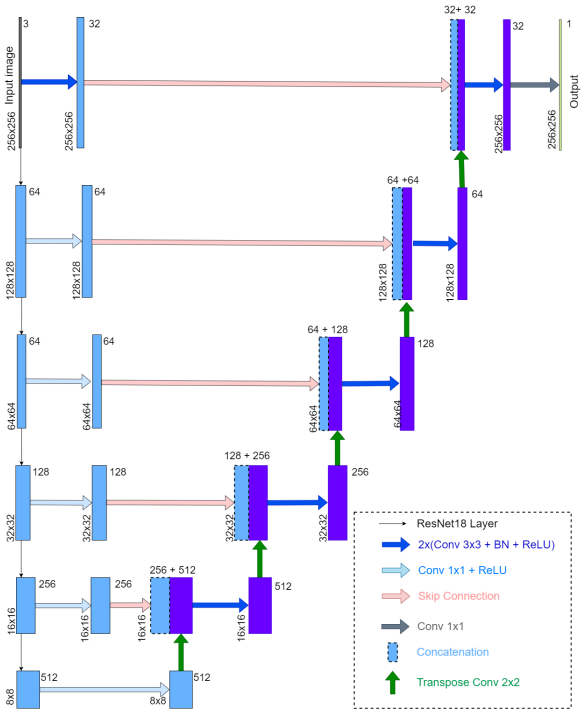


Figure 15: ResNet U-Net

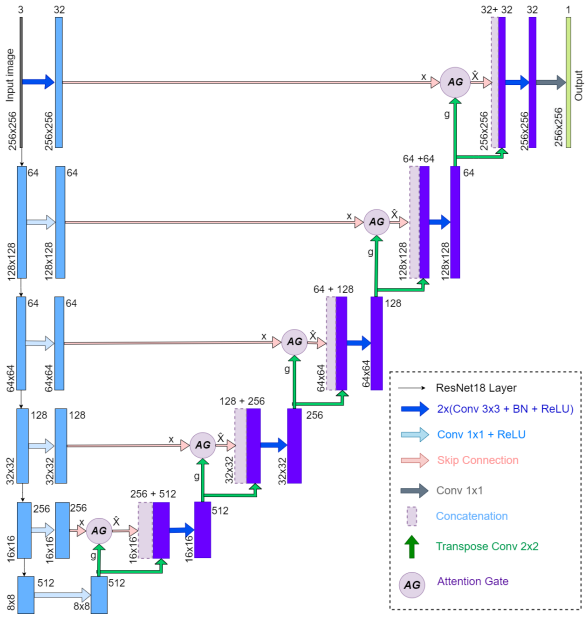


Figure 16: Attention ResNet U-Net

This requires preprocessing the images before feeding them to the network. For each image the following pipeline is applied:

- Images are padded with zeros to make each image have the shape of a square. This square's side length is equal to the longest side of the original image. This step aims to preserve the aspect ratio of the images when resizing.
- The images are then resized to the same size (see the section below).
- The images are converted to a PyTorch tensor.
- The mean and standard deviation of the training set are calculated (across each color channel) and used to normalize the images according to (3):

$$I' = \frac{I - \mu}{\sigma} \tag{3}$$

Where I is the original image, I' is the preprocessed image, $\mu = [0.7079, 0.5916, 0.5469]$ and $\sigma = [0.0937, 0.1113, 0.1257]$.

3.1.2.3. Input Size

An important parameter to be selected is the size of the images to be fed to the models.

For transfer learning-based solutions an image size of 256x256 was used to keep the images (thus the features) close to the original ImageNet dataset that was used to train ResNet18 (224x224).

For U-Net, many of the works in the literature use an image size of 300x400. Two experiments were conducted where the U-Net model was trained with images of size 300x400 (without padding) in the first and with images of size 256x256 in the second (with padding to preserve the aspect ratio).

Similarly, Attention U-Net was trained with images of size 384x384 in the first experiment and 256x256 in the second (with padding in both experiments).

Adam optimization algorithm was used in these experiments with ReduceOnPlateau learning rate scheduler and early stopping based on the validation loss. U-Net was trained using the Jaccard loss and Attention U-Net was trained using BCE loss. The base learning rate was set to 10^{-3} for U-Net and 10^{-2} for Attention U-Net. The Jaccard scores were calculated on the 600 images of the test set after resizing the segmentation results to the original image size. The results of this step are summarized in table 12.

The results of the experiments show that, besides being slower and taking more epochs to converge, using bigger images as input to both models did not improve the results in terms of test Jaccard score.

Considering these results, the image size was set to 256x256 for the rest of the experiments for all models.

Model	Size	#Epochs	Jaccard
U-Net	300x400	61	0.5020
U-Net	256x256	51	0.7280
Att. U-Net	384x484	73	0.7255
Att. U-Net	256x256	50	0.7389

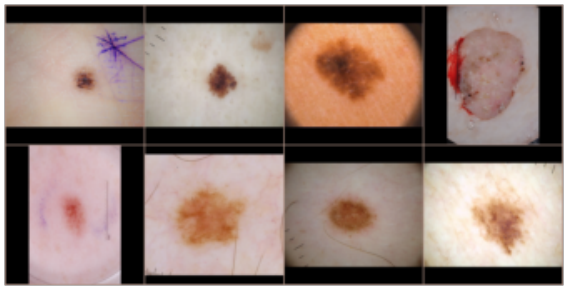
Table 12: Results of experiments on input size

3.1.2.4. Data Augmentation

Since the dataset is relatively small, the Albumentations library was used to perform online data augmentation on the training set. Multiple pipelines were tried, and the one that gave the best results is kept:

- Random 90° rotation with a 0.7 probability.
- Random horizontal flip with a 0.5 probability.
- Random vertical flip with a 0.5 probability.

A batch size of 8 for the training and 16 for the validation and test sets were used. figure 17 shows an example of a training batch.



((a))

((b))

Figure 17: (a) Training batch images. (b) - Training batch masks

3.1.3. Training

Next, the models described in the previous section were trained using different hyperparameters. This section provides more details about the training process.

3.1.3.1. Optimizer

SGD and Adam optimizers were tested on a smaller set using U-Net architecture. Since Adam outperformed SDG by a large margin (for example, a 0.49 test Jaccard score was obtained using SGD against 0.73 using Adam with a U-Net model), Adam optimizer was used for the rest of the experiments. The optimizer's weight decay parameter was set to 10^{-6}.

3.1.3.2. Loss Functions

Since the goal of this task is to maximize the Jaccard score of the test set, each of the models was trained using the Jaccard Loss function.

Jaccard Loss: Also known as IoU loss (intersection over union). This loss function is given by (1-Jaccard score) where the Jaccard score is the ratio between the overlap of the positive instances between two sets (the predicted and the actual skin lesions), and their mutual combined values. Since the results of the network are not binary masks but a map of pixel membership probabilities to the lesion set, a soft version of the Jaccard loss is used. It is given by (4):

$$\mathfrak{L}j(X,Y) = 1 - \frac{I(X,Y)+\epsilon}{U(X,Y)+\epsilon} \tag{4}$$

Where

$$I(X,Y) = \sum_i \sum_j (x_{i,j} * y_{i,j}) \tag{5}$$

$$U(X,Y) = \sum_i \sum_j (x_{i,j} + y_{i,j}) - I(X,Y) \tag{6}$$

Where X is the model's prediction, Y is the ground truth, i is the row index, j is the column index, and ϵ is a smoothing parameter to avoid division by 0.

However, this loss function is not stable and multiple problems were encountered during the training before finding the optimal hyperparameters (such as precision errors resulting in loss values being *nan*). For these reasons, two more loss functions were tried:

Binary Cross Entropy Loss (BCE): Available as BCELossWithLogits() in the nn module of PyTorch, BCE is the standard loss function for segmentation. It is given by (7)

$$\mathfrak{L}(x,y) = -[y.log\sigma(x) + (1-y).log(1-\sigma(x))] \tag{7}$$

where x is the prediction of the model, y is the ground and σ is the sigmoid function.

BCE with Dice Loss: This loss function is given by the sum of the previously described BCE loss

	Model	U-Net	Att. U-Net	Res. U-Net	Att. Res. U-Net
Jaccard Loss	Base LR	10^{-3}	10^{-3}	10^{-4}	10^{-4}
	LR patience	5	7	7	7
	LR decay	0.1	0.5	0.5	0.5
	ES patience	15	15	20	20
BCE Loss	Base LR	10^{-2}	10^{-2}	10^{-4}	10^{-4}
	LR patience	7	5	7	7
	LR decay	0.2	0.2	0.5	0.5
	ES patience	20	20	20	20
BCE + Dice Loss	Base LR	10^{-3}	10^{-3}	10^{-4}	10^{-4}
	LR patience	7	7	7	7
	LR decay	0.5	0.2	0.5	0.5
	ES patience	20	20	20	20

Table 13: Segmentation training recipes

and a soft version of the Dice loss. The Dice loss used is given by (8):

$$\mathfrak{L}d(X,Y) = 1 - \frac{2I(X,Y) + \epsilon}{S(X,Y) + \epsilon} \tag{8}$$

Where

$$I(X,Y) = \sum_i \sum_j (x_{i,j} * y_{i,j}) \tag{9}$$

$$S(X,Y) = \sum_i \sum_j (x_{i,j} + y_{i,j}) \tag{10}$$

Where X is the model's prediction, Y is the ground truth, i is the row index, j is the column index, and ϵ is a smoothing parameter to avoid division by 0.

Combining the two objective functions (Dice and BCE) allows for diversity in the loss while benefitting from the stability of BCE.

In all previously mentioned loss functions, the loss of a batch is given by the mean of the losses of the images in the batch.

3.1.3.3. Other Training Details

ReduceOnPlateau learning rate scheduler was used. It decays the learning rate by a predefined factor (LR decay) if the validation loss does not improve for a predefined number of epochs (LR patience).

An early stopping mechanism was implemented where the training stops if the validation loss does not improve after a chosen number of epochs (ES patience). The maximum number of epochs was set to 100 for all experiments.

When using pre-trained weights, all the weights of the model were fine-tuned.

The base learning rate, learning rate patience, learning rate decay, and early stopping patience were set to different values for each model after trial and error. The training recipes for this stage are summarized in table 13 .

The remaining parameters were set to the default values. The model weights that achieved the highest validation dice score are saved to a checkpoint for model evaluation. The training progress graphs of the loss, dice score and learning rate during the training of each model are available in Appendix I.

3.1.4. Threshold Optimization

The result of a segmentation model is a map of probabilities. Each cell holds a number between 0 and 1 representing the probability of the corresponding pixel belonging to a skin lesion. This result needs to be binarized using a threshold. In most works, this threshold is set to 0.5. It was noticed that this threshold does not always give optimal results. To find an optimal choice of threshold the Bayesian Optimization algorithm implemented in [9] was used.

An intermediate function that takes the threshold as a parameter and returns the Jaccard score calculated over the validation set using that threshold for binarizing the results of a model was created for each model. Bayesian Optimization aims to find the threshold that maximizes this black-box function.

The Bayesian optimization process starts with setting a prior over the Jaccard score as a function of the threshold. After obtaining more data by evaluating the function at new threshold values, the prior is updated to form the posterior distribution of the Jaccard function, in the library used in this step, this posterior is obtained by fitting a Gaussian process to the points that are previously explored. The posterior function is used to construct an acquisition function to determine the next threshold value to be tested. The procedure was set to stop after 10 iterations. The threshold that gave the highest Jaccard score on the valida-

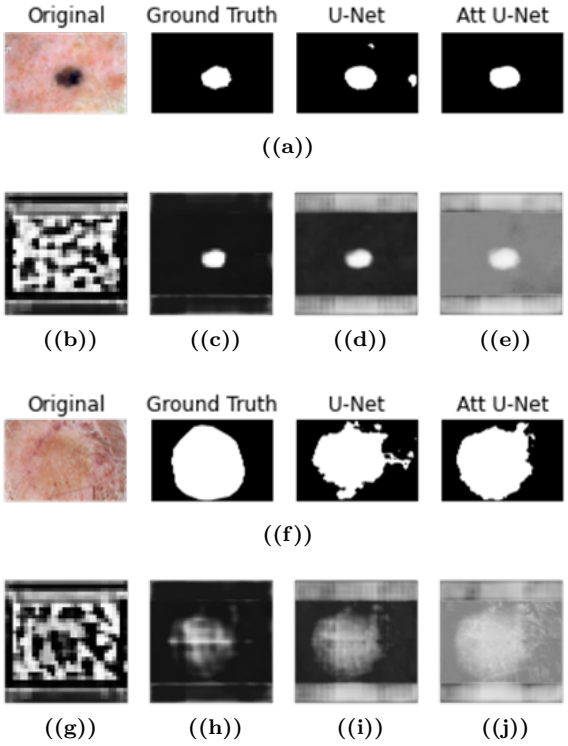


Figure 18: (a) Segmentation results of ISIC0015563. (b-e) Attention coefficients calculated for ISIC0015563. (f) Segmentation results of ISIC0013170. (g-j) Attention coefficients calculated for ISIC0013170.

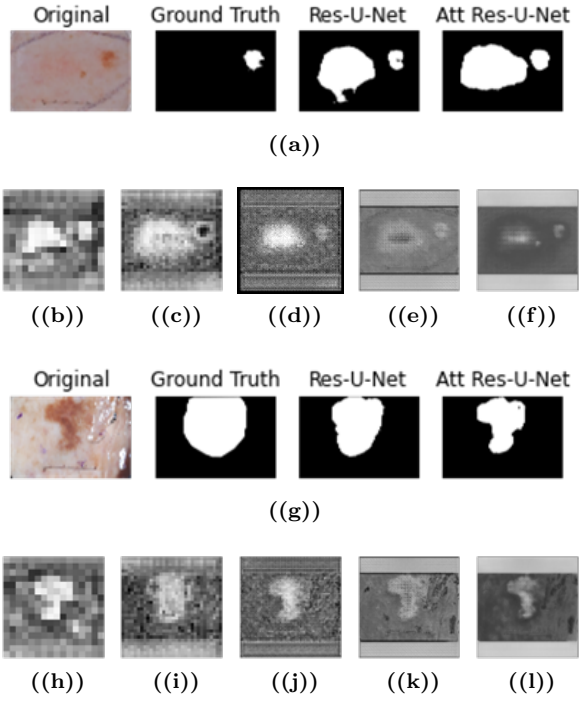


Figure 19: (a) Segmentation results of ISIC0016034. (b-f) Attention coefficients calculated for ISIC0016034. (g) Segmentation results of ISIC0012848. (h-l) Attention coefficients calculated for ISIC0012848

tion set is rounded to 4 decimal places and used to binarize the model's results on the test set during model evaluation.

3.1.5. Experimental results

Each of the models is tested on the 600 images of ISIC2017 test set. The Jaccard score is used to evaluate the performance of the models. It is calculated after binarizing the results using the optimal thresholds and resizing the result back to the original size of the image. The summary of the training, threshold optimization and evaluation of all models is shown in table 14.

The results revealed that when training a model from scratch, combining the BCE loss with the Dice loss achieved the best results whereas when using transfer learning, BCE loss outperformed the other loss functions. It is also seen that ResNet-based U-Net models initialized with pre-trained weights outperform the simple U-Net and Attention U-Net in terms of the Jaccard score and they converge in less epochs.

From the threshold optimization step, it was noticed that models trained with Jaccard loss tend to give very low probabilities (< 0.01) to pixels belonging to the background and very high probabilities (> 0.99) to pixels belonging to the lesion. For example, for Attention U-Net, a threshold of 0.01 results in a validation Jaccard score of 0.6797 whereas a threshold of 0.99 results in a very similar validation Jaccard score of 0.6726.

When the results of U-Net and Attention U-Net are compared, it is seen that Attention U-Net achieves better results because it reduces the number of false-positives as illustrated in figure 18.

From the examples, it is seen that the attention gates seem to "ignore" the regions that were detected as false positives by the U-Net model.

In the case of ResNet-based models, the results show that Attention ResNet U-Net gives a lower Jaccard score compared to ResNet U-Net. A visual inspection of the results showed that, unlike the vanilla U-Net, Resnet U-Net rarely suffers from false positives. In these cases, It was seen that Attention ResNet U-Net also failed in suppressing these regions because of their high similarity to skin lesions as illustrated in figure 19 (a). In some cases where the lesions are not homogeneous, it was noticed that Attention ResNet U-Net seems to suppress regions that were annotated as lesions by the human annotator like the example in figure 19 (g). The results show that the attention mechanism failed in challenging situations such as when there is hair in the image or when the image has a low variance. Training attention gates in this

Model	Loss	Epochs	Threshold	Test Loss	Test Jaccard
Vanilla U-Net	Jaccard	51	0.0100	0.3353	0.7280
	BCE	57	0.3366	0.1570	0.7212
	Dice + BCE	92	0.1728	0.4654	0.7382
Attention U-Net	Jaccard	48	0.0100	0.3399	0.7243
	BCE	50	0.3274	0.1728	0.7389
	Dice + BCE	73	0.2514	0.4118	0.7419
ResNet U-Net	Jaccard	39	0.0100	0.2921	0.7697
	BCE	**27**	**0.2188**	**0.1698**	**0.7857**
	Dice + BCE	38	0.1710	0.3840	0.7785
Attention ResNet U-Net	Jaccard	35	0.0388	0.2825	0.7688
	BCE	36	0.2827	0.1852	0.7785
	Dice + BCE	38	0.1759	0.3730	0.7756

Table 14: Deep learning segmentation results

Method	Jaccard
ASCU-Net [14]	0.7420
Conv-Deconv Nets [16]	0.7650
FAT-Net [15]	0.7653
ResNet U-Net (ours)	0.7857
DL with Auxiliary task [6]	0.7946

Table 15: Comparison with state-of-the-art methods

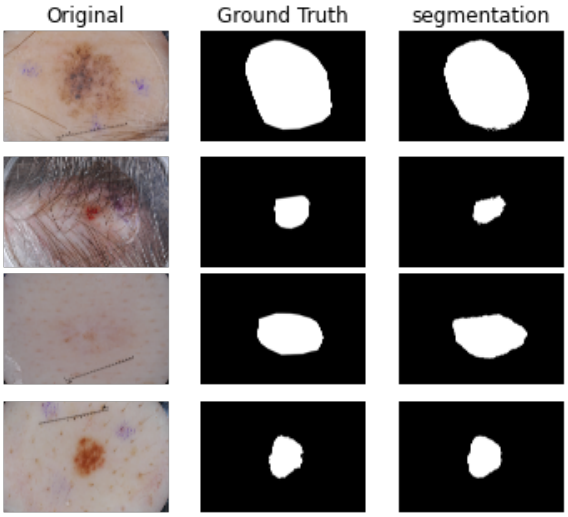


Figure 20: Example segmentation results

case becomes difficult and further finetuning of the training parameters could improve the results.

Among all the models trained, the model that achieved the best result is ResNet U-Net which was trained using BCE loss. The model achieved a loss of 0.1698 and a Jaccard score of 0.7857 on the 600 images of ISIC2017 test set. This result is 0.0207 higher than the best result achieved in the challenge (0.765), and as shown in table 15 it is comparable to the state-of-the-art methods.

This model achieves a Jaccard Score of 0.8451 on the training set and 0.7803 on the validation set. Examples of the segmentation results can be seen in figure 20.

Some failure cases are given in figure 21. It is seen that the model generally fails in challenging cases such as when the skin lesion is heterogeneous or when the lesion's texture and color are highly similar to the healthy skin. It is also worth mentioning that some ground truth masks include a roughly affected area around the lesion that is not detected by the model (e.g. fourth row in figure 21).

3.1.6. Ablation Study

In this section, 3 experiments were conducted to study the contribution of data augmentation, transfer learning, and threshold optimization to the results of the task of skin lesion segmentation. In all experiments, the architecture and the parameters that achieved the best result are used. In the first experiment, the model was trained without data augmentation. The original 2000 images of ISIC2017 training set were used in each epoch. In the second experiment, the model is trained with randomly initialized weights instead of the pre-trained ImageNet weights. In the third experiment, the results of the model are binarized using the standard threshold 0.5. The results are summarized in table 16.

The results show that data augmentation, transfer learning and threshold optimization contributed to the results of the task with an order of 0.02 in the test Jaccard Score.

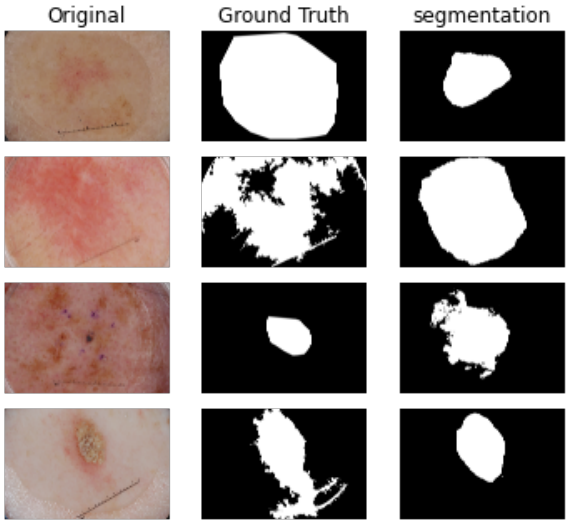


Figure 21: Example segmentation failure cases

Experiment	Jaccard
1	0.7689
2	0.7630
3	0.7560

Table 16: DL segmentation ablation study results

3.2. Classification

The classification part was implemented in two stages. During the first stage, Transfer Learning was used to fine-tune multiple well-known image classification models using different recipes (loss functions, optimizers, hyperparameters). The second stage aims to improve the performance of the classification by exploiting the segmentation masks obtained from part 1 to guide the classification task.

3.2.1. Stage 1

3.2.1.1. Classification Models

The torchvision library provides a large set of pretrained image classification architectures. Among these models 3 were chosen to be fine-tuned to perform multi-class skin lesion classification in this stage of the project.

3.2.1.1.1. DenseNet

Based on the literature review and the results of the challenge, Densenet architectures [2] outperformed the other classification CNNs such as ResNet and Inception in the task of skin lesion segmentation.
DenseNet is a convolutional neural network based on dense blocks. Each layer in the network is fed the feature maps from all preceding layers, a "collective knowledge", as input via skip connections and it passes these inputs on its feature maps by concatenation to all subsequent layers. Densenets are thinner and more compact compared to other architectures.
In this project, a pretrained DenseNet121 network from the torchvision.models module is used. Since the number of classes in our task is 3 (benign, melanoma and seborrheic keratosis) the last linear layer is replaced with a linear layer of 3 output neurones instead of 1000. As the name implies, this network has a total of 120 convolutional layers and a linear layer as a classifier with a total of 8 million parameters.

3.2.1.1.2. Vision Transformer: Is a skin lesion image worth 16x16 words?

The second model tried is Vision Transformer (ViT) [1]. This architecture employs a transformer over patches of images. Input images are split into fixed-sized patches, usually of size 16x16, each patch then passes through a linear layer to obtain the linear embedding. A position embedding is also added by concatenation to the linear embedding. The resulting sequence (embedding for each patch) is then fed to a standard transformer encoder. An extra classification token is added to the sequence to learn the classification labels.

In this project, the model vit_b_16 from the torchvision.models module pre-trained with ImageNet was used. This model has a total of 86.6M parameters.

3.2.1.1.3. ConvNeXt

ConvNexts are the new generation of Convolutional Neural Networks modernized to compete with transformers. ConvNeXts combine micro design techniques that have been separately used in previous research collectively. Thus, the difference between these models and traditional CNNs (ResNet, Inception, VGG...) is mainly in the hyperparameters and training recipes rather than the architecture.
Some of the changes applied in ConvNeXts are adjusting the number of blocks in each stage of the traditional models to make the sliding windows behave more like the patches of vision transformers, using larger kernel sizes to capture global receptive fields, preventing window overlap using the stride, using Gaussian Error Linear Unit (GELU) instead of ReLU and replacing Batch Normalization with Layer Normalization.
In this project, the model convnext_small from the torchvision.models module was used. This network

was pre-trained on the imageNet dataset and has 50M parameters.

3.2.1.2. Data

3.2.1.2.1. Dataset

The original dataset from the ISIC2017 challenge was used for both stages.

3.2.1.2.2. Preprocessing

The same problems found in the segmentation part apply in this task. For classification, the following preprocessing pipeline was used:

- Resizing the images to 260x260.
- Center cropping the images to 256x256 for DenseNet and ConvNeXt and to 224x224 for ViT.
- Transforming the images to a PyTorch Tensor.
- Normalizing with $\mu = [0.7079, 0.5916, 0.5469]$ and $\sigma = [0.0937, 0.1113, 0.1257]$, calculated from the original training set.

3.2.1.2.3. Data Augmentation

Data augmentation is more important in this part of the project since the dataset is imbalanced and there is a limited number of samples in the minority classes (Melanoma: 374 images, Keratosis: 254 images). In this part, the study conducted in [12] is exploited. The pipeline that obtained the best results in this work is used for online data augmentation in this task:

- Random resized cropping to size 256x256 with probability 0.7, scaling between 0.4 and 1, and rotation between 0.75 and 4/3.
- An affine transformation with scaling in [0.8, 1.2], rotation of 0° to 90°, shearing with a degree in [0, 20], with a probability of 0.8.
- Flipping (either horizontally or vertically or both) with a probability of 0.9.
- A color jitter of probability 0.6 with brightness, contrast, and saturation factors in the range [0.7, 1.3] and a hue in [-0.05, 0.05]. The factors are sampled from a uniform distribution by default.

This pipeline was implemented using the Albumentations library. The contribution of this step is later demonstrated in the ablation study.

Data augmentation and shuffling are performed only on the training set. A batch size of 10 for the training and 64 for the validation and test sets are used.

A custom sampler was implemented to create a balanced training loader with an equal number of samples in each class using down-sampling (there will be 254 samples in each class). This is acheived by randomly sampling images from each class without replacement until the dataloader has 254 images from each class. Figure 22 shows an example of a training batch.

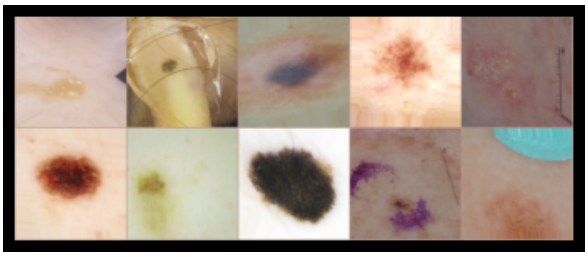


Figure 22: Example training batch

3.2.1.3. Training

The three classification models described previously were trained. This section provides details about the different hyperparameters used in the training procedure.

3.2.1.3.1. Optimizers

Each model was trained using both Adam and SGD optimizers. For SGD, a momentum of 0.9 was used in all experiments.

3.2.1.3.2. Objective functions

To address the problem of class imbalance, each model was trained using two different loss functions that usually work best with imbalanced data:
Multi-Class Focal Loss: As introduced in [5], focal loss generalizes multi-class cross-entropy by introducing a hyper-parameter γ (focusing parameter) that allows focusing on hard examples. It is given by 11.

$$\mathfrak{L}f(p_t) = -\alpha(1-p_t)^{\gamma}\log(p_t) \tag{11}$$

Where α is a weighting factor, p_t is the model's estimated probabilities vector, and γ is the focusing parameter. In this project, γ is set to 2 and α is set to the class weights vector calculated using sci-kit learn's compute_class_weight(). For a given class C, the weight of C is given by 12.

$$W_C = \frac{N}{c * n_C} \tag{12}$$

Where N is the total number of samples in the training set (2000 in this case), c is the number

Loss	Focal Loss		WCE Loss		CE with Sampling	
Optimizer	SGD	Adam	SGD	Adam	SGD	Adam
LR patience	5	7	7	7	7	7
LR decay	0.25	0.5	0.5	0.5	0.5	0.5
Weight decay	10^{-6}	10^{-8}	10^{-8}	10^{-8}	10^{-8}	10^{-8}
ES patience	10	10	15	10	15	10

Table 17: DenseNet training recipes for stage 1

	Model	ViT		ConvNeXt	
	Optimizer	SGD	Adam	SGD	Adam
Focal Loss	LR patience	5	5	5	5
	LR decay	0.1	0.5	0.1	0.1
	Weight decay	10^{-8}	10^{-8}	10^{-6}	0
	ES patience	10	10	10	10
WCE Loss	LR patience	5	10	5	5
	LR decay	0.1	0.5	0.1	0.5
	Weight decay	10^{-8}	0	10^{-8}	10^{-8}
	ES patience	10	15	10	10

Table 18: ViT and ConvNeXt training recipes

of classes (3), and n_C is the number of samples in class C.

Weighted Cross-Entropy: A variant of Cross-Entropy loss given by 13.

$$WCE\left(p_t\right) = -\alpha_t \log\left(p_t\right) \tag{13}$$

Where p_t is the model's estimated probabilities and α is set to the class weights vector given by 12.

The DenseNet model was also trained using the standard multi-class cross-entropy loss (without weights) and the balanced training data loader created using the custom sampler described in the previous section.

3.2.1.3.3. Other Training Details

ReduceOnPlateau learning rate scheduler was used. It decays the learning rate by a predefined factor (LR decay) if the validation loss does not improve for a predefined number of epochs (LR patience). The base learning rate was set to 10^{-4} for all experiments.

An early stopping mechanism was implemented where the training stops if the validation loss does not improve after a chosen number of epochs (ES patience). The maximum number of epochs was set to 100 for all experiments.

In all experiments, all the weights of the models were fine-tuned without freezing any layers.

The learning rate patience, learning rate decay, weight decay and early stopping patience were set to different values for each model after trial and error. The DenseNet training recipes are summarized in table 17 and ViT and ConvNeXt training recipes are summarized in 18.

The remaining parameters were set to the default values. The model weights that achieved the highest validation balanced multi-class accuracy (BMA) score are saved to a checkpoint for model evaluation. The training progress graphs of the loss, BMA and learning rate during the training of each model in this stage are available in Appendix II.

3.2.1.4. Experimental Results

The evaluation metric used in this project is balanced multi-class accuracy. It is given by 14

$$BMA = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{n_c} \tag{14}$$

Where C is the total number of classes (3), TP_c is the number of true positives detected by the model in class c and n_c is the total number of samples in class c.

Each model was tested on the 600 images of the ISIC2017 test set. The test loss, accuracy, and BMA were calculated. The results are summarized in table 19.

The results show that in general, models trained with cross-entropy loss outperformed the models trained with focal loss. It is also seen that training with Adam optimizer provides a faster convergence and higher results in terms of test BMA in most cases. Training Vision Transformers was more challenging than training Convolutional Networks and the best practices used for ConvNets did not always work for ViTs. For example, setting the weight decay of the Adam optimizer to 0 improved the BMA obtained with ViT from 0.5511 to 0.7289. It was noticed that ConvNeXts converge faster than the traditional DenseNet (in as few as 16 epochs) and they provide comparable results. However, DenseNet architectures still acheived better results than ViT and ConvNeXt with the training recipes used in this project. The best test BMA obtained at this stage was 0.7527 with a DenseNet architecture trained using cross-entropy loss and a

Model	Loss function	Optimizer	#Epochs	Test Loss	Test Accuracy	Test BMA
DenseNet	Focal Loss	SGD	67	0.3083	0.6433	0.6839
		Adam	56	0.3792	0.7700	0.7275
	Weighted CE	SGD	66	0.7014	0.7000	0.7319
		Adam	35	0.7558	0.7417	0.7351
	CE + Sampling	SGD	92	0.6613	0.7167	0.7138
		Adam	**58**	**0.6837**	**0.7567**	**0.7527**
ViT	Focal Loss	SGD	53	0.3217	0.6667	0.6729
		Adam	55	0.4889	0.6267	0.6554
	Weighted CE	SGD	37	0.7611	0.6383	0.6802
		Adam	28	0.6922	0.7333	0.7289
ConvNeXt	Focal Loss	SGD	45	0.3341	0.6167	0.6600
		Adam	17	0.3410	0.7350	0.7311
	Weighted CE	SGD	36	0.6937	0.7000	0.7182
		Adam	16	0.7113	0.6367	0.7197

Table 19: Classification results for stage 1

balanced training loader that contains 254 images from each class.

3.2.2. Stage 2

In this stage, the segmentation results obtained from the first part are exploited to improve the classification performance. Since DenseNet architecture achieved higher results in stage 1 it will be used in this stage.

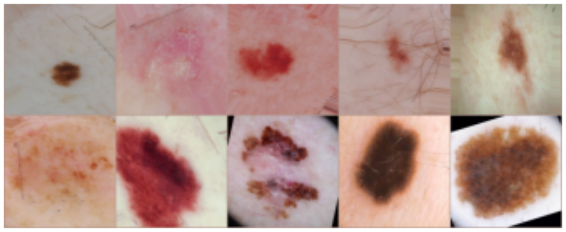
3.2.2.1. Data

The goal of this stage is to use the segmentation masks to inject information to the classification model about the location of the skin lesion to be classified in the image. To achieve this, there are two main options:

Image Masking: Suppressing the background of the image and leaving the original pixel values only at the lesion region then using the original pre-trained model for classification. However, many of the skin lesions are relatively small compared to the image size which will result in the networks inputs containing mostly zeros. Another option is to crop the lesion using a bounding box. In this case the images will have highly diverse sizes and shapes which makes learning the features very challenging. Furthermore, the segmentation results are not accurate enough, thus in many cases parts of the lesion if not the whole lesion will be lost as illustrated in figure 21.

Using the segmentation mask as a 4th channel: this approach requires adjusting the model accordingly by replacing the input convolutional layer to accept inputs with 4 channels instead of 3. In this case, no pre-trained weights are available for the additional channel.

Because the first option is more problematic, the second approach is followed at this stage. Both the image and the segmentation mask pass through the same preprocessing pipeline as in stage 1. The same data augmentation pipeline is used. Before being fed to the networks the segmentation mask is concatenated with the original RGB channels of the image. The resulting 4-channel tensor is then fed to the network. The training batch size was set to 10 while 32 was used as batch size for the validation and test sets. Data augmentation and shuffling were performed only on the training set. Figure 23 shows an example training batch.



((a))

((b))

Figure 23: (a) Training batch images (channels 0 to 2) (b) Training batch masks (channel 3)

3.2.2.2. Classification Model

In this stage, the model DenseNet121 from torchvision.models module is used. The first layer of the model is replaced with a convolutional layer that takes 4-channel inputs instead of 3 by setting the in_features parameter to 4. The remaining parameters are kept the same as in the original architecture. To keep the benefits of transfer learning, the pre-trained weights of the original 3 first channels are loaded to the new first layer while the weights of the 4th channel are initialized randomly. The last linear layer is replaced with a new linear layer with 3 output neurones.

3.2.2.3. Training

The same training procedure including the optimizers, loss functions, learning rate scheduler and early stopping mechanism are used in this stage. The training recipes are given in table 20.

The graphs of metrics progress during the training for this phase are given in Appendix III.

3.2.2.4. Experimental results

Each of the models trained in this stage was tested on the 600 images of the test set and the results are summarized in table 21.

The results show that using the segmentation masks as 4th channel of the model's input improved the performance of the classification task when using all the training samples. However, when using the undersampled training set with cross- entropy loss there is a drop in the test BMA with an order of 0.02. A possible reason for this drop is the lack of pretrained weights for the fourth channel which requires more training samples. This approach is also dependent on the segmentation result and improving the segmentation masks could also improve the classification's perfermance.

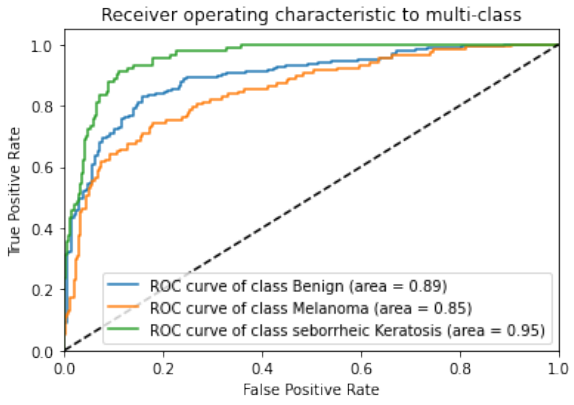
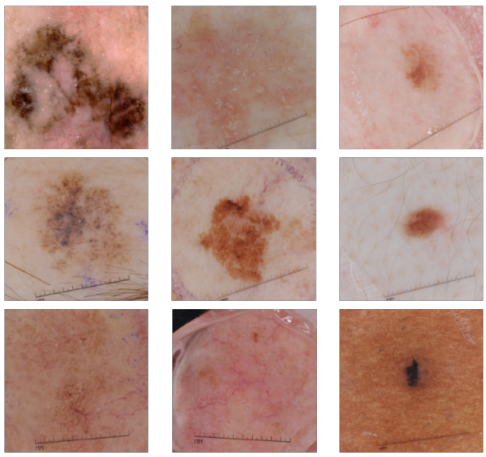


Figure 24: ROC curve of the final classification model

Among all the models trained in this task, the model that performed the best is DenseNet that takes an input that has the RGB image and the segmentation mask as a fourth channel, trained with weighted cross-entropy loss, and Adam as an optimization algorithm. This model achieved a BMA of 0.7766 on the ISIC2017 test set. The accuracies of the benign, melanoma, and seborrheic keratosis classes were 0.7888, 0.6410, and 0.9000 respectively. Table 22 gives more details about the classification performance. Figure 24 shows the ROC curve and the AUC metric for each class. The precision is given by the means of the metric calculated for each label. The ROC curve and the AUC are calculated for each class using the One-Versus-Rest method.

Figure 25 shows examples of correctly classified images while figure 26 shows examples of failure cases.



((a)) ((b)) ((c))

Figure 25: Examples correct classifications (a) Melanoma (b) Keratosis (c) Benign

3.2.3. Ablation Study

At the end of this task, an ablation study was conducted to test the contribution of using the segmentation mask, data augmentation, and transfer learning to the task of skin lesion classification. The results obtained in the final model are also compared with the results obtained using the first approach (masking the image). The parameters that obtained the best results were used in this study.

The following four experiments were performed:

1- The best model's results were compared with the DenseNet model that was trained using the same loss function and optimizer but with RGB images in stage 1 to study the contribution of adding the segmentation mask to the results.

Loss	Focal Loss		WCE Loss		CE with Sampling	
Optimizer	SGD	Adam	SGD	Adam	SGD	Adam
LR patience	5	5	5	7	5	5
LR decay	0.25	0.5	0.5	0.5	0.1	0.1
Weight decay	10^{-8}	10^{-8}	10^{-8}	10^{-8}	10^{-8}	10^{-8}
ES patience	10	10	10	10	10	10

Table 20: DenseNet training recipes for stage 2

Model	Loss function	Optimizer	#Epochs	Test Loss	Test Accuracy	Test BMA
DenseNet	Focal Loss	SGD	52	0.3245	0.6067	0.6690
		Adam	41	0.3029	0.7167	0.7352
	Weighted CE	SGD	34	0.6772	0.7133	0.7358
		Adam	**45**	**0.6644**	**0.7767**	**0.7766**
	CE + Sampling	SGD	48	0.7587	0.6617	0.6992
		Adam	52	0.6546	0.7383	0.7288

Table 21: Classification results for stage 2

Figure 26: Example classification failures: Prediction (Correct)

2- The same model was trained without data augmentation. The original 2000 samples of ISIC2017 training set were used to train the model in each epoch.

3- The model was trained without loading ImageNet weights. All the weights were initialized randomly.

4- Instead of using the mask as a 4th channel, the images were masked where the pixels of the background were replaced with zeros. The images were then used to train the original DenseNet model with transfer learning. Figure 27 shows an example training batch used in this experiment.

The results of the ablation study in terms of BMA calculated on the test set are summarized in table 23. As seen from the results, unlike in the segmentation task, transfer learning and data augmentation have a significant contribution to the task of skin lesion classification where it is seen that data augmentation contributed with an order of 0.2 and transfer learning contributed with an order of 0.4 to the test BMA. On the other hand, it is seen that using the masked images instead of a four-channel input provided a lower BMA on the test set. Finally, when the results of the same model are compared with the results of stage 1, it can be seen that using the segmentation mask in the classification task slightly improved the performance in terms of BMA. This contribution could increase with a more accurate segmentation performance.

BMA	Accuracy	Precision	Confusion Matrix
0.7766	0.7767	0.7033	$\begin{bmatrix} \mathbf{310} & 47 & 36 \\ 23 & \mathbf{75} & 19 \\ 6 & 3 & \mathbf{81} \end{bmatrix}$

Table 22: DL Classification results

Experiment	BMA
1	0.7351
2	0.5970
3	0.3907
4	0.6975

Table 23: DL classification ablation study results

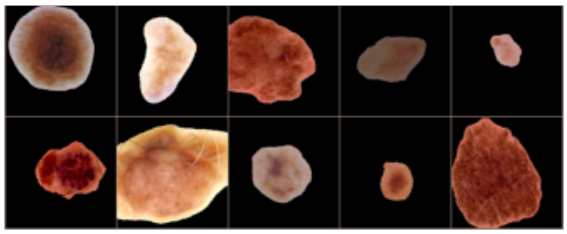


Figure 27: Example training batch of masked images

4. Conclusion

In this project, the tasks of skin lesion segmentation and classification were addressed using two different approaches. Advanced analysis techniques followed by traditional machine learning from one hand and deep learning from another. The automation of these tasks was challenging for different reasons such as the different artifacts present in dermoscopy images, high resolution and the heterogeneity of the lesions. However, the results show a potential of improvement in this task especially using deep learning techniques. Possible future work includes more optimization and preprocessing in the feature engineering step of machine learning, further fine-tuning of the model parameters in deep learning especially in the case of attention-based models as well as using novel attention techniques in the classification task.

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., ≪An image is worth 16x16 words: transformers for image recognition at scale≫, (2020).

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, ≪Densely connected convolutional networks≫, in Proceedings of the ieee conference on computer vision and pattern recognition (2017), pp. 4700–4708.

[3] G. W. Jiji and P. J. DuraiRaj, ≪Content-based image retrieval techniques for the analysis of dermatological lesions using particle swarm optimization technique≫, Applied Soft Computing **30**, 650–662 (2015).

[4] G. Lemaître, F. Nogueira, and C. K. Aridas, ≪Imbalanced learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning≫, Journal of Machine Learning Research **18**, 1–5 (2017).

[5] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, ≪Focal loss for dense object detection≫, in Proceedings of the ieee international conference on computer vision (2017), pp. 2980–2988.

[6] L. Liu, Y. Y. Tsui, and M. Mandal, ≪Skin lesion segmentation using deep learning with auxiliary task≫, Journal of Imaging **7**, 67 (2021).

[7] F. M. Miranda, N. Köehnecke, and B. Y. Renard, ≪Hiclass: a python library for local hierarchical classification compatible with scikit-learn≫, arXiv preprint arXiv:2112.06560 (2021).

[8] M. Nasir, M. Attique Khan, M. Sharif, I. U. Lali, T. Saba, and T. Iqbal, ≪An improved strategy for skin lesion detection and classification using uniform segmentation and feature selection based approach≫, Microscopy research and technique **81**, 528–543 (2018).

[9] F. Nogueira, *Bayesian Optimization: open source constrained global optimization tool for Python*, 2014–.

[10] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, et al., ≪Attention u-net: learning where to look for the pancreas≫, arXiv preprint arXiv:1804.03999 (2018).

[11] B. C. Paes, A. Plastino, and A. A. Freitas, ≪Improving local per level hierarchical classification≫, Journal of Information and Data Management **3**, 394–394 (2012).

[12]F. Perez, C. Vasconcelos, S. Avila, and E. Valle, ≪Data augmentation for skin lesion analysis≫, in *Or 2.0 context-aware operating theaters, computer assisted robotic endoscopy, clinical image-based procedures, and skin image analysis* (Springer, 2018), pp. 303–311.

[13]O. Ronneberger, P. Fischer, and T. Brox, ≪U-net: convolutional networks for biomedical image segmentation≫, in International conference on medical image computing and computer-assisted intervention (Springer, 2015), pp. 234–241.

[14]X. Tong, J. Wei, B. Sun, S. Su, Z. Zuo, and P. Wu, ≪Ascu-net: attention gate, spatial and channel attention u-net for skin lesion segmentation≫, Diagnostics **11**, 501 (2021).

[15]H. Wu, S. Chen, G. Chen, W. Wang, B. Lei, and Z. Wen, ≪Fat-net: feature adaptive transformers for automated skin lesion segmentation≫, Medical Image Analysis **76**, 102327 (2022).

[16]Y. Yuan, ≪Automatic skin lesion segmentation with fully convolutional-deconvolutional networks≫, arXiv preprint arXiv:1703.05165 (2017).

Appendix I: Segmentation Training Progress

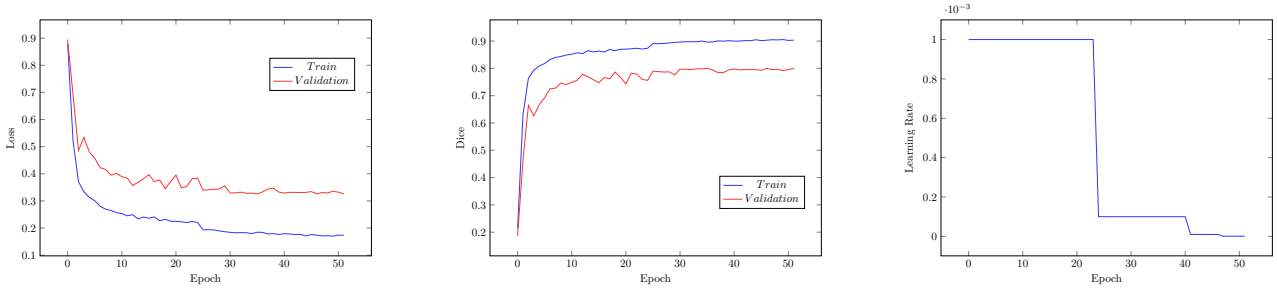


Figure 1: Model: U-Net, Loss: Jaccard

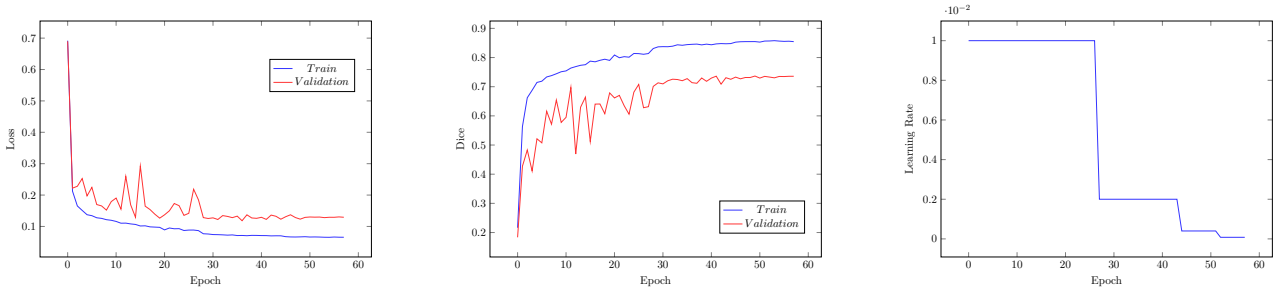


Figure 2: Model: U-Net, Loss: BCE

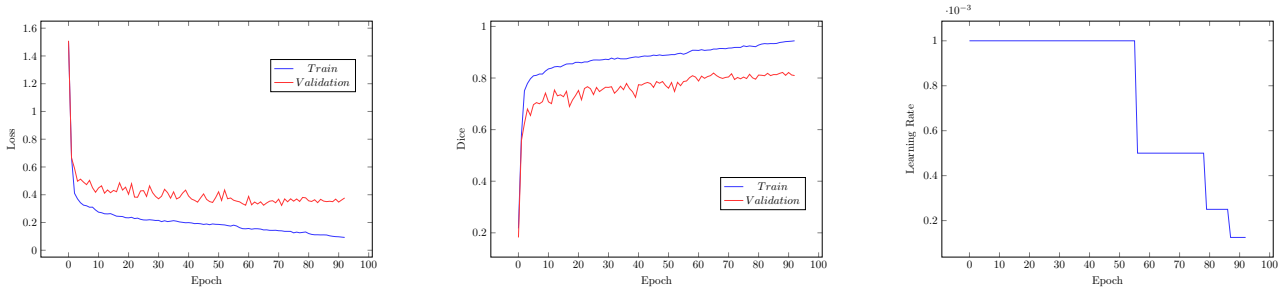


Figure 3: Model: U-Net, Loss: BCE+Dice

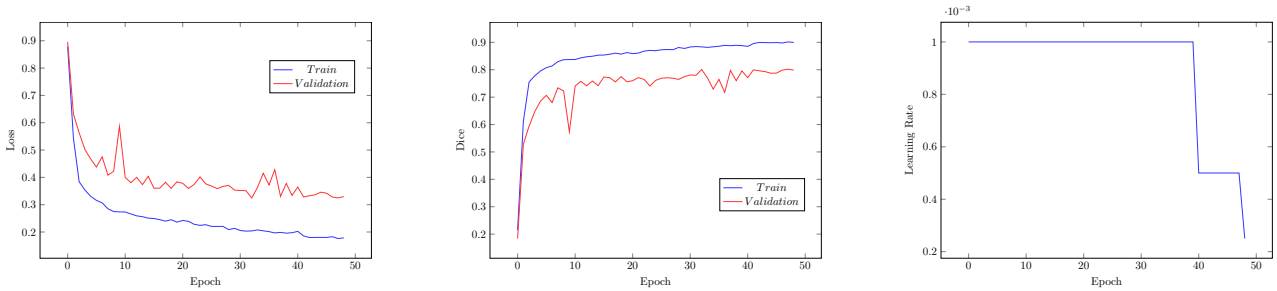


Figure 4: Model: Attention U-Net, Loss: Jaccard

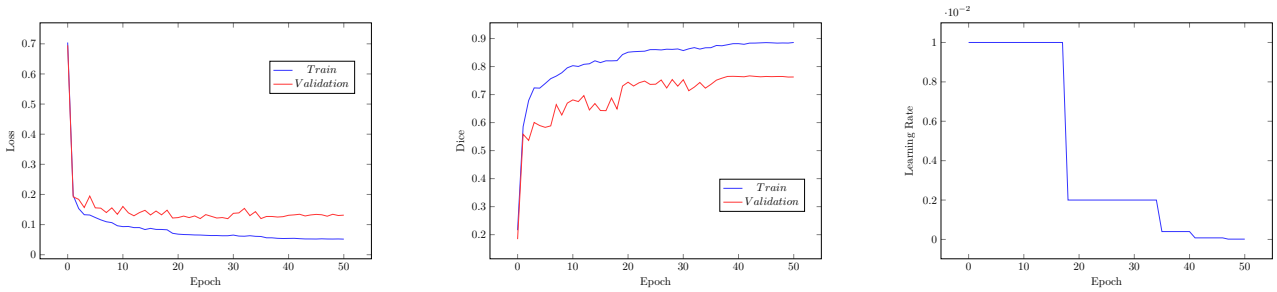


Figure 5: Model: Attention U-Net, Loss: BCE

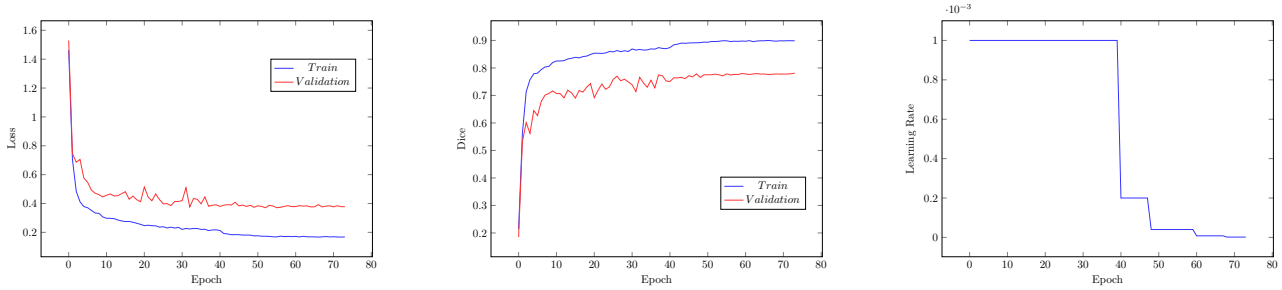


Figure 6: Model: Attention U-Net, Loss: BCE+Dice

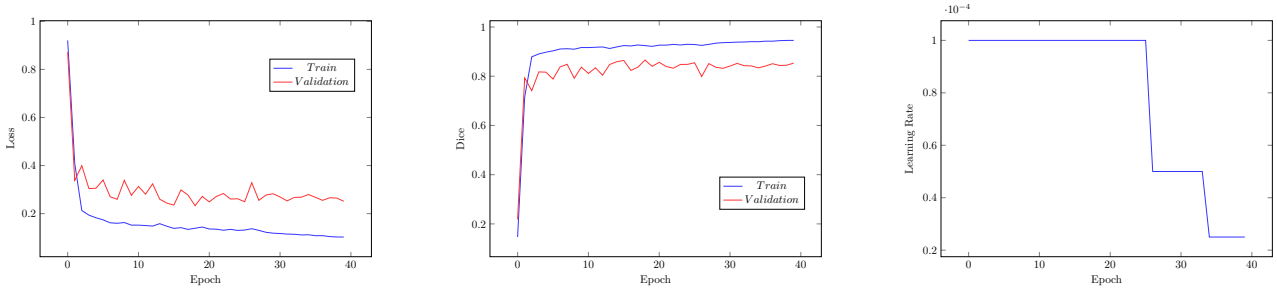


Figure 7: Model: ResNet U-Net, Loss: Jaccard

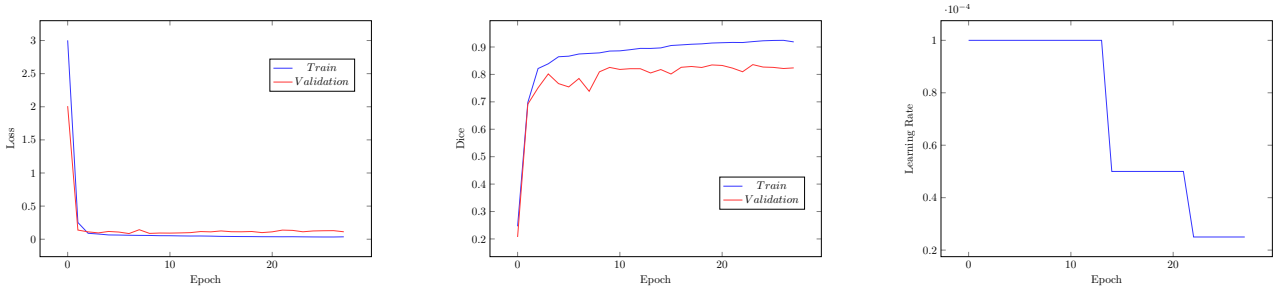


Figure 8: Model: ResNet U-Net, Loss: BCE

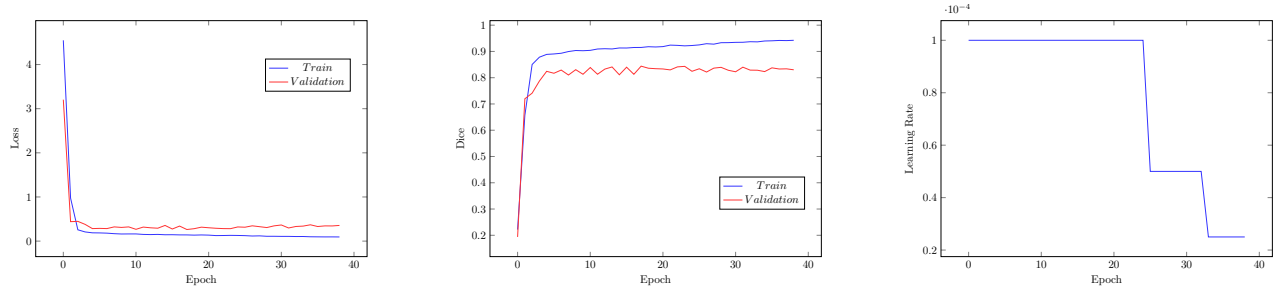


Figure 9: Model: ResNet U-Net, Loss: BCE+Dice

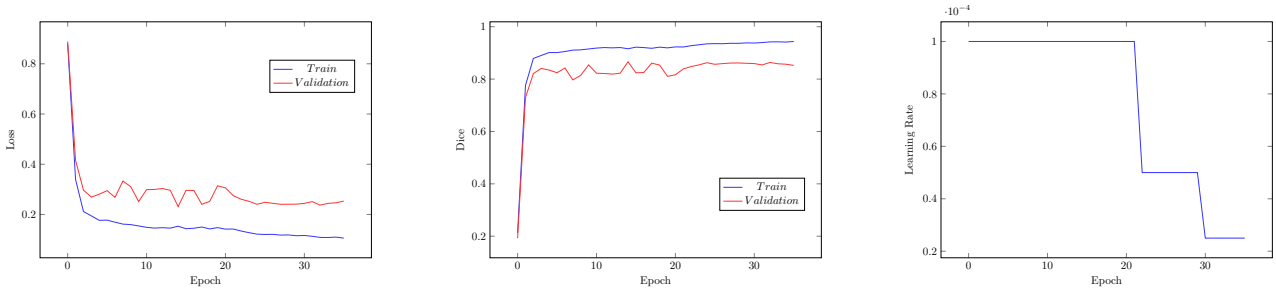


Figure 10: Model: Attention ResNet U-Net, Loss: Jaccard

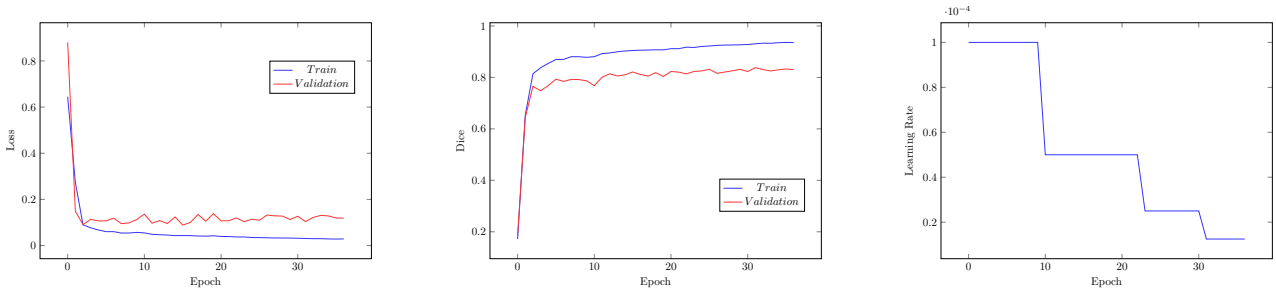


Figure 11: Model: Attention ResNet U-Net, Loss: BCE

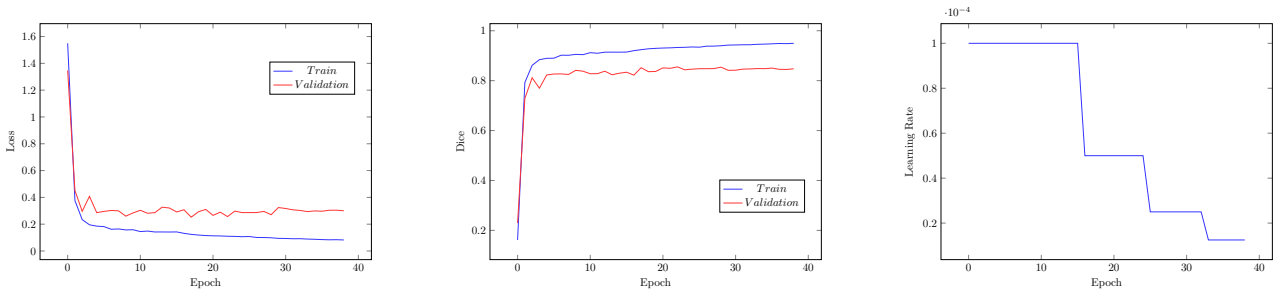


Figure 12: Model: Attention ResNet U-Net, Loss: BCE+Dice

Appendix II: Classification Training Progress - Stage 1

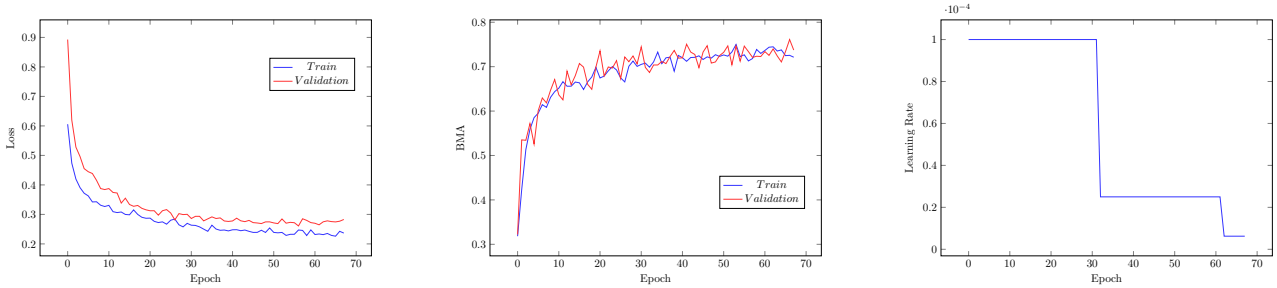


Figure 1: Model: DenseNet, Loss: Focal Loss, Optimizer: SGD

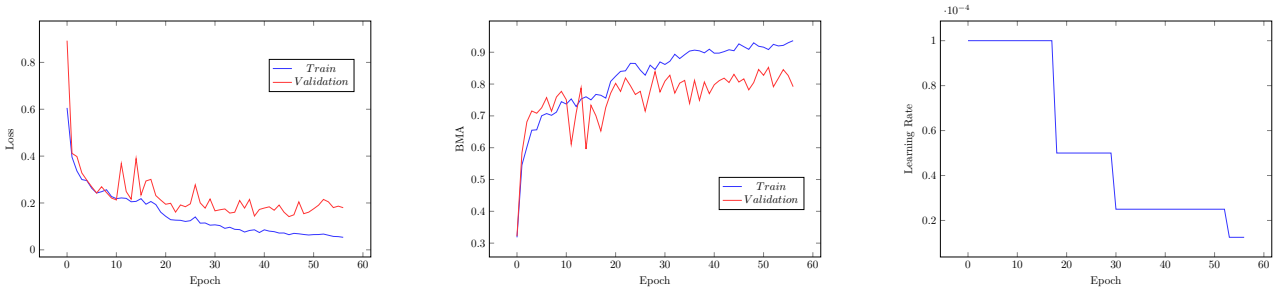


Figure 2: Model: DenseNet, Loss: Focal Loss, Optimizer: Adam

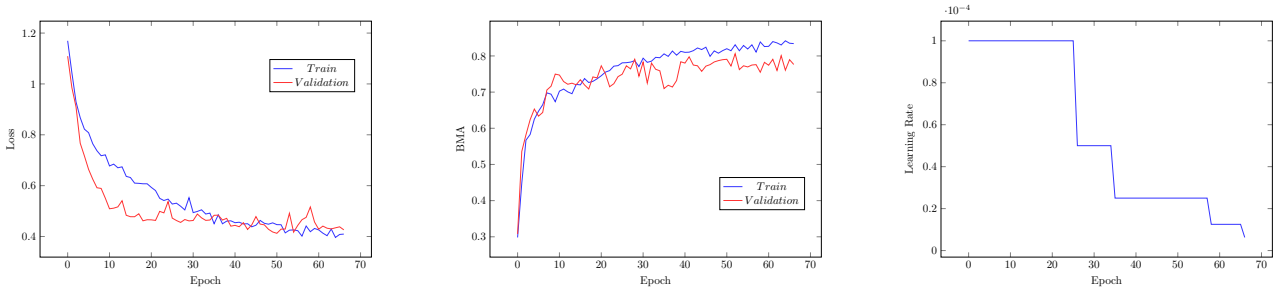


Figure 3: Model: DenseNet, Loss: WCE, Optimizer: SGD

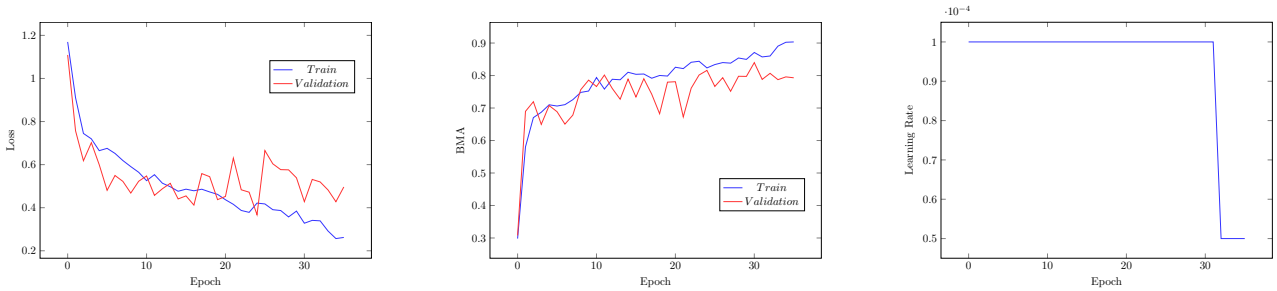


Figure 4: Model: DenseNet, Loss: WCE, Optimizer: Adam

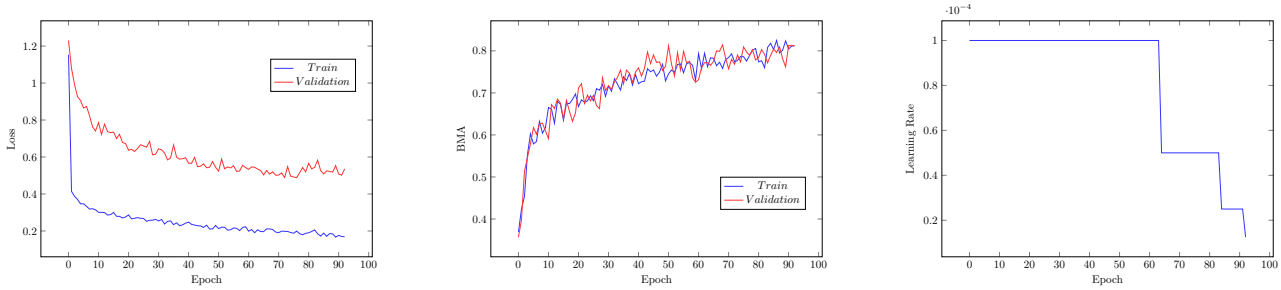


Figure 5: Model: DenseNet, Loss: CE, Optimizer: SGD

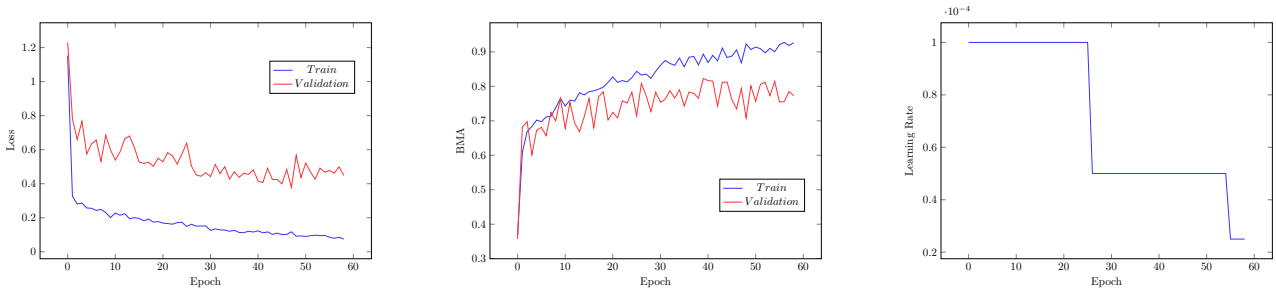


Figure 6: Model: DenseNet, Loss: CE, Optimizer: Adam

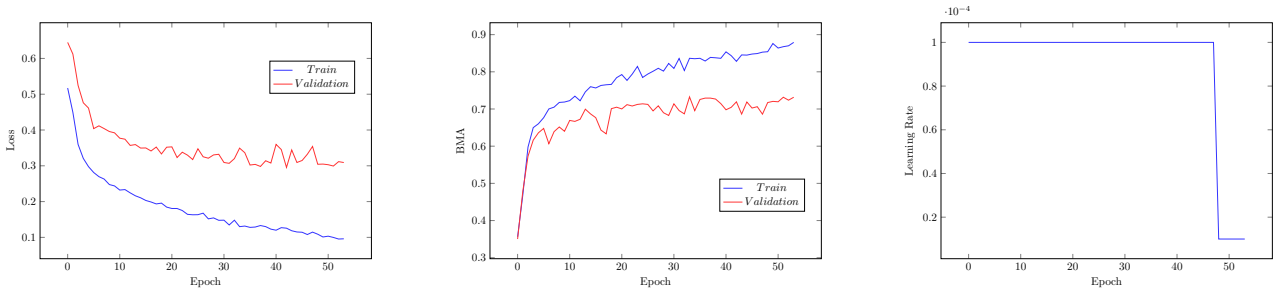


Figure 7: Model: ViT, Loss: Focal Loss, Optimizer: SGD

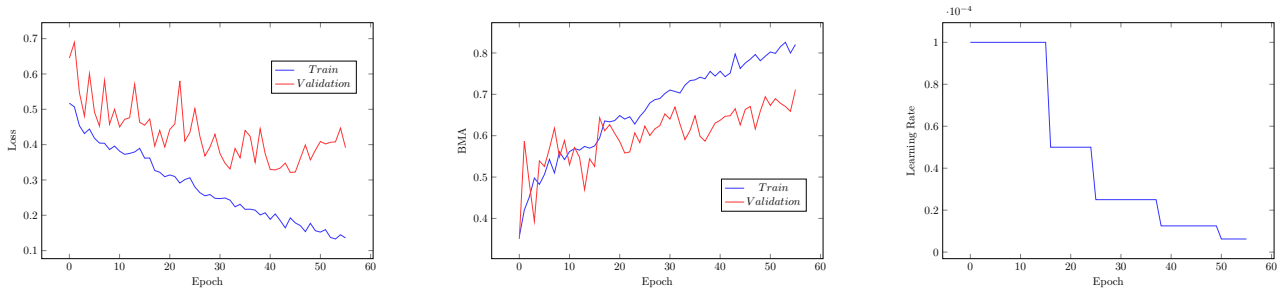


Figure 8: Model: ViT, Loss: Focal Loss, Optimizer: Adam

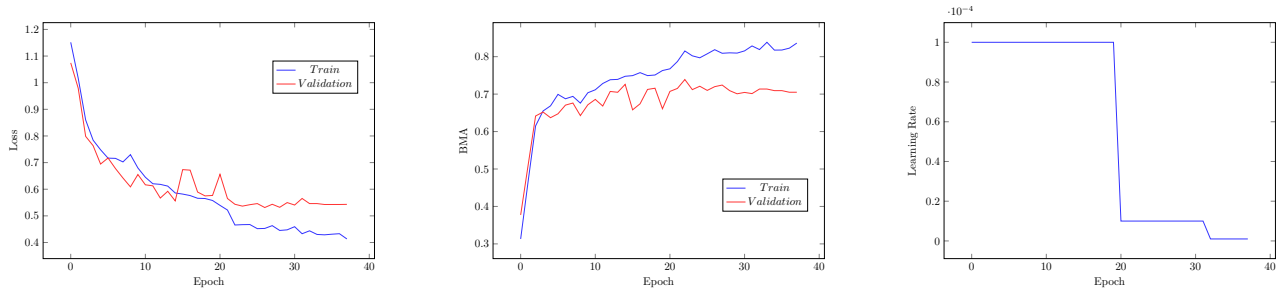


Figure 9: Model: ViT, Loss: WCE, Optimizer: SGD

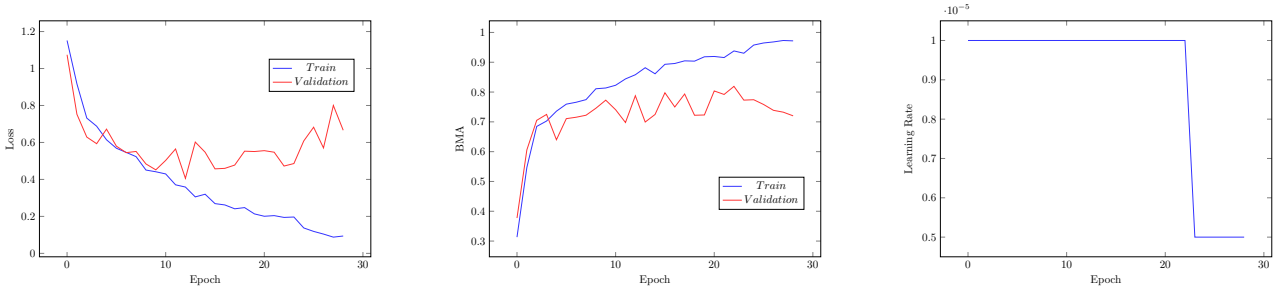


Figure 10: Model: ViT, Loss: WCE, Optimizer: Adam

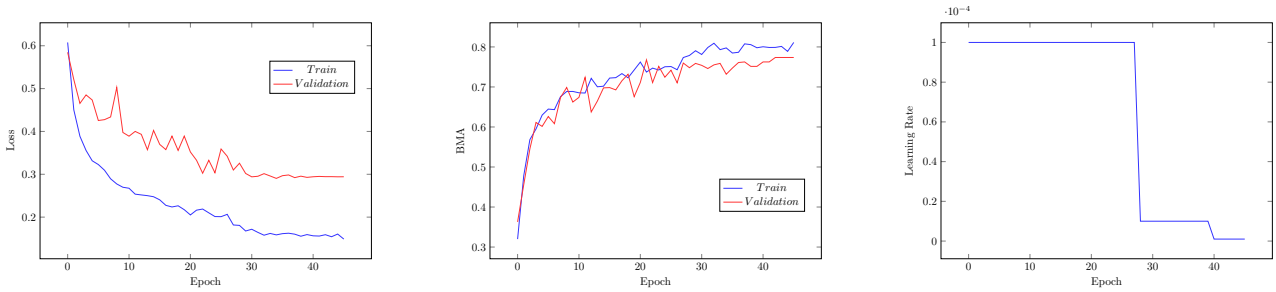


Figure 11: Model: ConvNeXt, Loss: Focal Loss, Optimizer: SGD

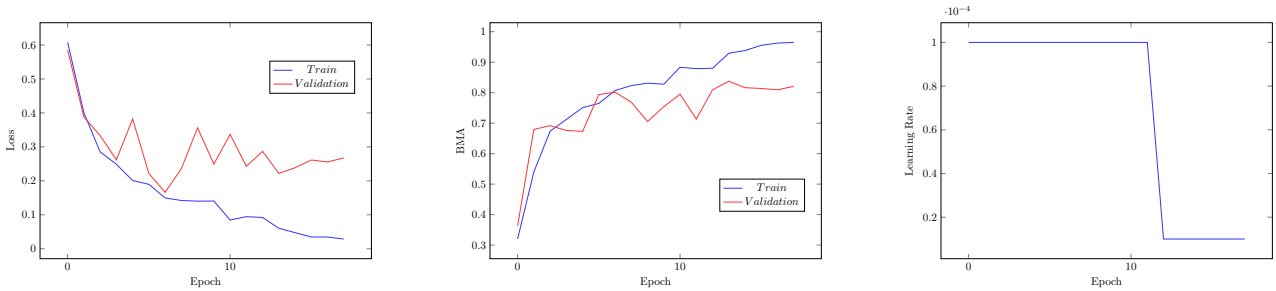


Figure 12: Model: ConvNeXt, Loss: Focal Loss, Optimizer: Adam

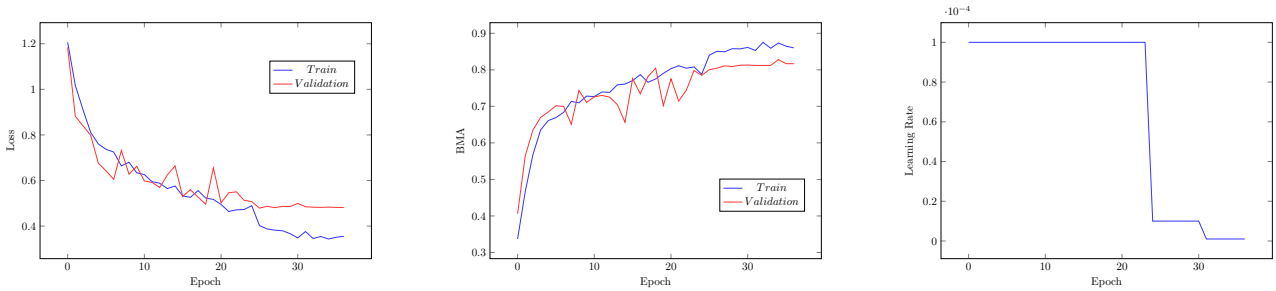


Figure 13: Model: ConvNeXt, Loss: WCE, Optimizer: SGD

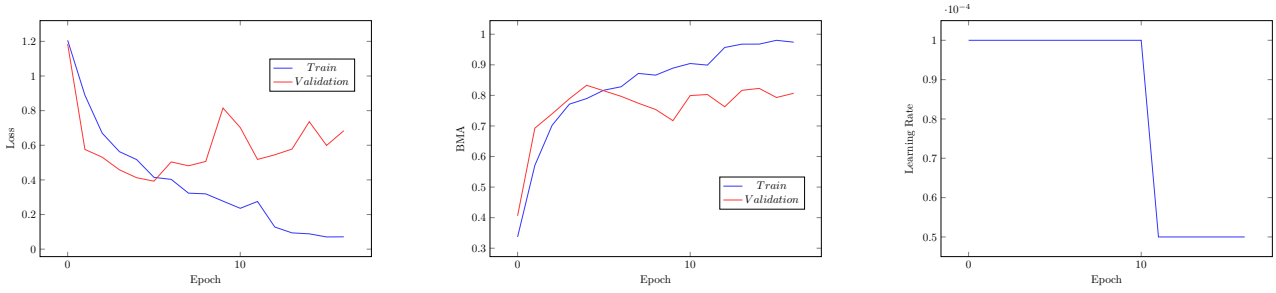


Figure 14: Model: ConvNeXt, Loss: WCE, Optimizer: Adam

Appendix III: Classification Training Progress - Stage 2

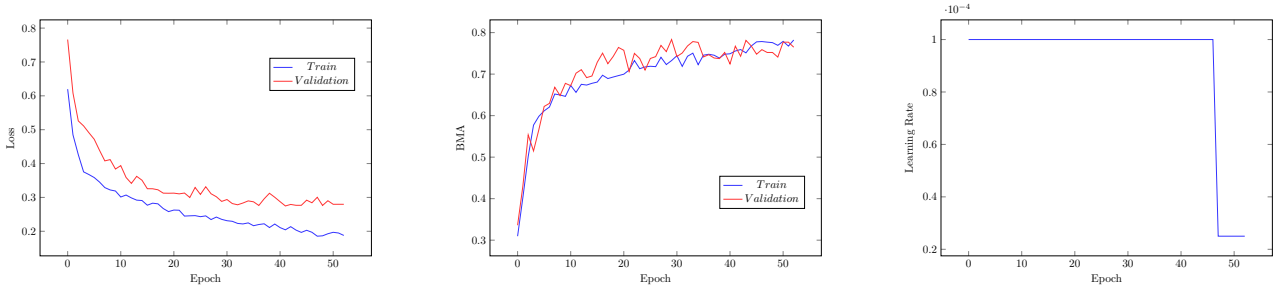


Figure 1: Model: DenseNet, Loss: Focal Loss, Optimizer: SGD

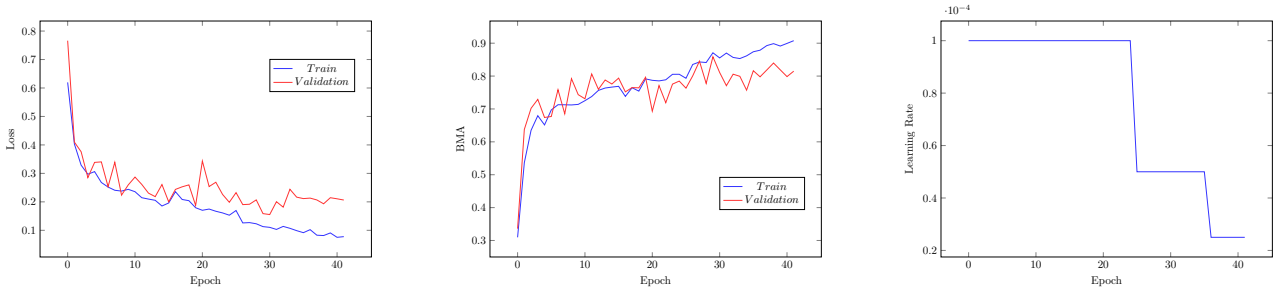


Figure 2: Model: DenseNet, Loss: Focal Loss, Optimizer: Adam

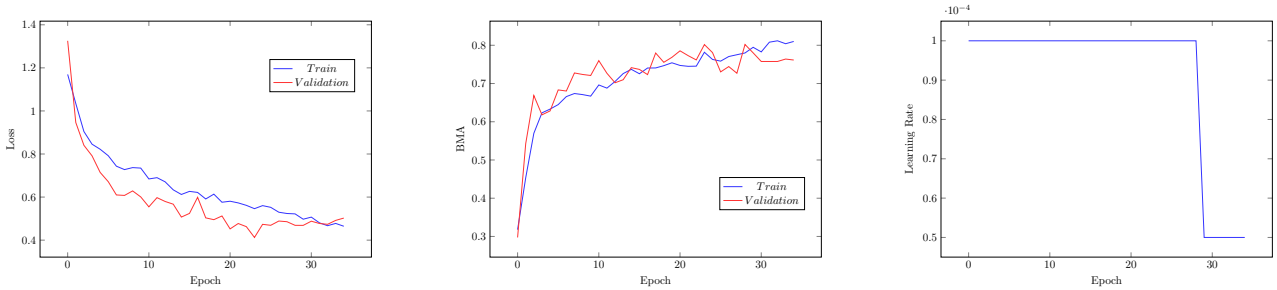


Figure 3: Model: DenseNet, Loss: WCE, Optimizer: SGD

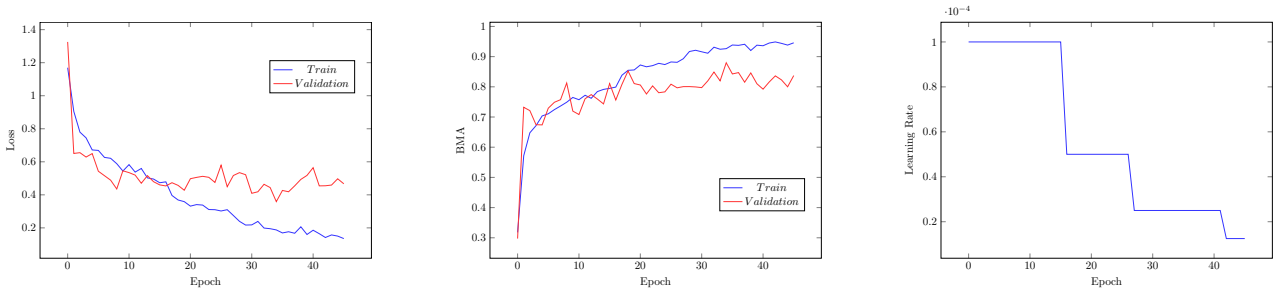


Figure 4: Model: DenseNet, Loss: WCE, Optimizer: Adam

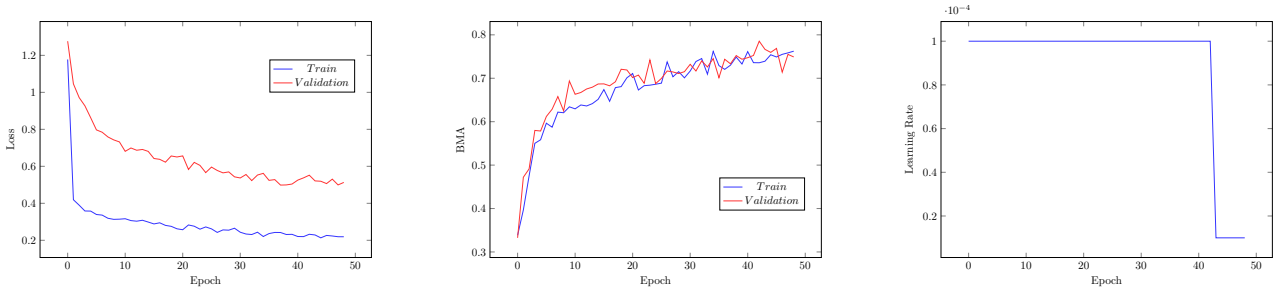


Figure 5: Model: DenseNet, Loss: CE, Optimizer: SGD

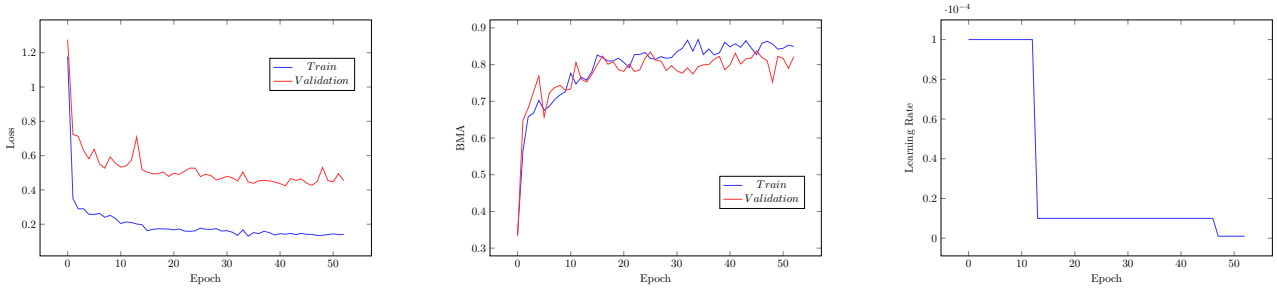


Figure 6: Model: DenseNet, Loss: CE, Optimizer: Adam