PROJECT Nº 2 - Skin Lesion Analysis (based on the ISIC Challenges)

Carolina Pinto
up201506006@fe.up.pt

Helena Montenegro
up201604184@fe.up.pt

Juliana Marques
up201605568@fe.up.pt

Computer Vision - 2019/2020

Faculdade de Engenharia da Universidade do Porto (FEUP)

Abstract

The goal of this project is to implement and analyse different approaches to perform automatic classification and semantic segmentation on skin lesions from dermoscopic images.

1 Task 1 – Binary image classification

The goal of the first task is to implement two or more different approaches to identify skin lesions as benign or malignant. The training data set is composed of 900 images. The testing data set consists of 379 images, where 304 are classified as benign and 75 are classified as malignant. We will compare the results obtained in each approach with a baseline and with each other.

1.1 Metrics used to evaluate the classification models

- **Accuracy:** Percentage of lesions correctly classified.
- **Precision:** Percentage of lesions predicted as malignant correctly classified.
- **Recall:** Percentage of malignant lesions correctly identified.
- **F1 score:** Considers both precision and recall.

1.2 Baseline

The baseline is obtained by identifying every single image as benign since there are more benign lesions than malignant ones. The following confusion matrix represents the baseline:

Prediction	Reality	
	Malignant	Benign
Malignant	0	0
Benign	75	304

Since the test dataset is unbalanced, the baseline has a very high **accuracy** of 80,2%. However, the **recall** is 0% since none of the malignant lesions are correctly identified. No lesions were identified as malignant, therefore there is no **precision** or **F1 score**.

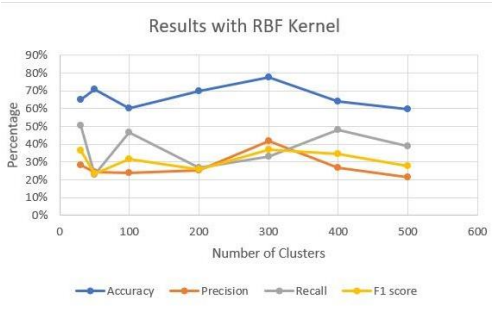
1.3 Approach 1: Bag of Visual Words and SVM for classification

This approach has the following steps:

1. SIFT to detect features in the training set.
2. Bag of Words with K-means for clustering, created using the descriptors obtained in step 1.
3. SVM for classification.
   a. Train SVM with training set using bag of words descriptor extractor.
   b. Use trained SVM to classify the images in the testing set.

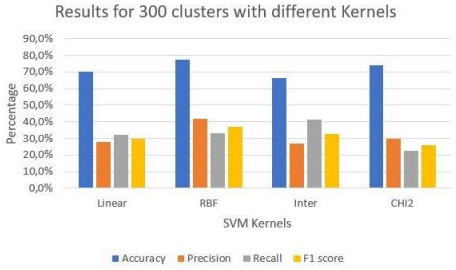
We tested this approach with different numbers of clusters and with different kernels in the SVM to improve the results as much as possible.

The results, in terms of accuracy, precision, recall and f1 score obtained using the RBF Kernel, for different numbers of clusters, can be seen in the following graph:



For a number of K = 300 clusters, the **accuracy**, **precision** and **F1 score** present the best results. When it comes to **recall**, there are other values for K which have better results, such as K = 30, K = 100, and K = 400. Overall, we consider that K = 300 presents better results than the other numbers of clusters analysed.

After discovering the best number of clusters to use for the bag of words, we analysed different kernels for the SVM, as can be seen in this picture:



We obtained the best results using **radial basis function (RBF)** as kernel, for all metrics, except for the **recall**, where histogram intersection kernel gave better results. Overall, we considered that **RBF** is the most appropriate kernel to use.

1.3.1 Results

The best results were obtained with 300 clusters and with the **RBF Kernel**. Here is the corresponding confusion matrix:

Prediction	Reality	
	Malignant	Benign
Malignant	25	35
Benign	50	269

With this model we obtained the following values for each metric:

Accuracy: 77,6% **Recall:** 33,3%

Precision: 41,7% **F1 score:** 37,0%

By examining these metrics, we can see that this approach surpasses the baseline when it comes to precision, recall, and f1 score. The only metric where this approach does not surpass the baseline is the accuracy, which is very high in the baseline since the dataset is so unbalanced. However, the difference between the accuracy obtained in this approach and in the baseline is only 2.6%. Taking this into consideration and taking into consideration that all other metrics are better, we can conclude that this approach that uses SIFT for feature detection, bag of words with K-means, and SVM for classification is more useful than the baseline.

1.4 Approach 2: Binary classification with VGG-16

1.4.1 Dataset

The training set of 900 images was divided into a train set and validation set. The validation set is used to provide an unbiased evaluation of the model fit on the training set, while tuning hyperparameters. The testing set will provide an unbiased evaluation of a final model.

The validation set consists of 15% of the training dataset and was randomly selected from the 900 images. It is important to note that both the validation and training set maintain the same class distribution as the complete set of 900 images.

1.4.2 Architecture

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” and achieves 92.7% top-5 test accuracy in ImageNet (a dataset of over 14 million images belonging to 1000 classes). The original architecture has 2 fully

connected layers, each with 4096 nodes followed by a 'softmax' classifier for output. In our implementation, we removed the fully connected layers at the end of the network and replaced them with one fully connected layer with 512 nodes followed by a 'sigmoid' classifier.

Initially, the network was trained from scratch but, as we have a very small dataset to train a full-scale model, we decided to use transfer learning. The fact that the network is very large and takes a very long time to train was also a reason for using transfer learning. Transfer learning allows us to reduce training time by re-using the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the ImageNet image recognition tasks. We, therefore, take advantage of useful learned features using the "ImageNet" pre-trained weights.

The summary of the model can be seen in Annex 1.1.

1.4.3 Class imbalance

The number of examples belonging to the negative class ('Benign') makes up 80.78% of the dataset while the positive class ('Malignant') makes up the remaining 19,22%. To handle this class imbalance, we used class weights in 'model.fit()' that reflect our class distribution. By setting the 'class_weights' parameter, misclassification errors regarding the less frequent class are upweighted in the loss function. We also used careful bias initialization (Annex 1.2.) by setting the output layer's bias to reflect the imbalance present in the dataset. This initialization helps with initial convergence and reduces training time.

1.4.4 Overfitting

As the dataset is very small and our network is very large, we observed some overfitting while training the network. To avoid the network overfitting, we performed data augmentation while training, such as horizontal flips, rotations and small translations with the "ImageDataGenerator" class. We used a regularization technique called Dropout on the fully connected layer and experimented with different values (0.2, 0.5, 0.7). This technique consists of randomly selected neurons being ignored ("dropped") during training, which, in turn, results in a network with better capacity for generalization that is less likely to overfit on the training data. Lastly, we also tried using L2 regularization of 5e-4.

1.4.5 Training

The new fully connected layer of 512 nodes that we included was randomly initialized, however, the convolutional layers have already learned rich, discriminative features. If we allow the gradient to backpropagate from these random values all the way through the network, these powerful features could be destroyed. To avoid this, we freeze all layers from our base model and only train the new fully connected "head".

We tried different optimizers such as RMsprop, SGD and Adam with different learning rates varying from 1e-6 to 1e-2 and with different batch sizes (32, 64, 128). We started with a batch size of 32 to ensure that each batch had a decent chance of containing a few positive examples. We also tried adding a different number of fully connected layers (1 or 2) with a different number of neurons each. Our best results were obtained with one fully connected layer of 512 nodes with the Adam optimizer with a learning rate of 1e-5, a batch size of 32 and a dropout value of 0.5.

In Annex 1.3, there are graphs that show how metrics like accuracy, loss, precision and recall changed throughout the training of the model.

1.4.6 Results

We obtained the following results on the test set composed of 379 images:

Accuracy: 82,3% **F1-score:** 54%
Precision: 55,7% **AUC:** 76,2%
Recall: 52%

Prediction	Reality	
	Benign	Malignant
Benign	273	31
Malignant	36	39

The results obtained surpass the baseline in all metrics, which means that this model is more useful than a trivial model where all the images are classified as benign. In Annex 1.4. we can see the results obtained for the test set in more detail.

1.5 Comparison between Approach 1 and Approach 2

After comparing the results of both approaches, we can conclude that the Approach 2, where CNN was used, resulted in a higher accuracy, which was even higher than the baseline, and better values for both recall and precision. As such, we can conclude that Approach 2 is better than Approach 1 when applied to this problem. This conclusion was to be expected as CNNs are currently the state of the art in image classification problems.

2 Task 2 - Multi-class image classification

This task focuses on multi-class image classification and we adapted the most successful method applied in task 1 for this task. This problem consists of predicting the type of disease present in dermoscopy images, considering a total of 7 classes: Melanoma, Melanocytic nevus, Basal cell carcinoma, Actinic keratosis, Benign keratosis, Dermatofibroma, and Vascular lesion.

The summary of the model can be seen in Annex 2.1.

2.1 Dataset

The available training dataset contains 10015 images which will be used for both training and testing, as no ground-truth labels are provided for validation/test sets. The dataset was split into train, test and validation sets the following way: 20% for test set, 15% for validation set and 65% corresponds to the train set. The class distribution of the complete set is maintained in test, train, and validation sets.

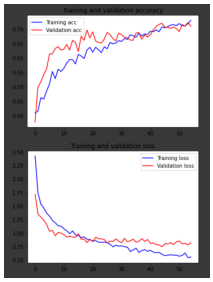
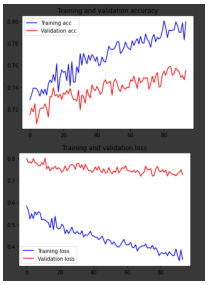
2.2 Training

In this task, there is also a big class imbalance present in the training dataset and so we also used class weights (that reflect the class distribution) for weighting the loss function during training.

As this problem involves multi-class classification, the 'sigmoid' classifier was replaced with 'softmax'. Data augmentation is also performed using the 'ImageDataGenerator' class to prevent overfitting and to increase the model's ability to generalize. A dropout of 0.5 is also used as a regularization technique to prevent overfitting.

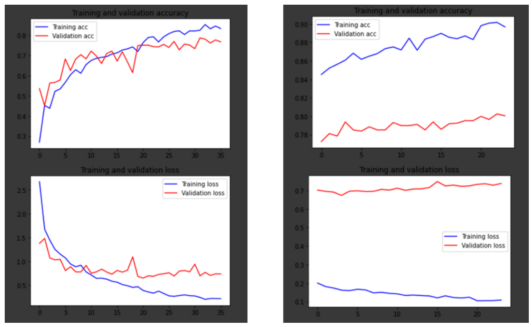
We started by training the head (the last fully connected layers) for 50 epochs with the Adam optimizer (learning rate of 1e-4) and a batch size of 128. Afterwards, we performed fine-tuning on the whole network by unfreezing all convolutional blocks and resumed training with a very small learning rate (1e-6). We used a very small learning rate so that the convolutional filters were not dramatically changed. We obtained an accuracy of 71% on the test set after training the head, and 75% after fine-tuning the whole network for 100 epochs. After examining the loss and accuracy graphs, we can conclude that, with more training, a higher accuracy can be achieved as the training and validation loss are still decreasing (albeit very slowly) while fine-tuning the whole network.

The graphs below present the results of the accuracy and loss during training when only the head is trained (on the left) and after unfreezing the convolutional blocks (on the right). (Annex 2.3)

The initial layers of convolutional networks learn simple and general features such as edges, while the deeper convolutional layers learn more complex and specific features. As our dataset is substantially different from ImageNet and larger from the one available in Task1, we decided to train the last convolutional block along with the head of the model (while the other convolutional blocks remained frozen). This meant a higher risk of the network overfitting, but it could also result in a higher validation accuracy in less training time.

The graphs below present the results of the accuracy and loss during training when the head + last convolutional block is trained (on the left) and after unfreezing the rest of the convolutional blocks (on the right). (Annex 2.4)



2.3 Results

The following results are relative to the test set composed of 2005 images. We obtained an accuracy of 78% after training the head along with the last convolutional block for only 35 epochs. This was using the Adam optimizer (learning rate of 1e-4). Finally, we obtained 81% accuracy after fine-tuning the whole network for 20 epochs with a learning rate of 1e-6. Annex 2.2 shows the results in more detail.

The resulting confusion matrix is presented below.

Prediction	Reality						
	AKIEC	BCC	BKL	DF	MEL	NV	VASC
AKIEC	36	4	15	2	7	2	0
BCC	3	84	6	1	2	7	0
BKL	6	2	165	2	24	21	0
DF	1	1	2	14	1	4	0
MEL	2	2	20	0	159	38	2
NV	1	13	49	2	141	1131	4
VASC	0	0	1	0	0	1	27

We can see that the model is particularly good at classifying 'Melanocytic nevus'. This was to be expected since most images belong to this class (over 4000 images). It is interesting to note how the model is also very good at identifying 'Vascular lesion', despite there being only 90 images in the train set for that class. After examining the examples belonging to this class, we think this happens because of the more distinctive features present in these images that make it easier for the model to tell them apart. These 2 classes present the best results in terms of precision and recall.

Overall, the results seem satisfactory for multiclass classification considering there are 7 different classes and a limited dataset.

3 Task 3 - Image semantic segmentation

The goal of the third task is to implement a CNN-based approach to perform semantic segmentation on lesions.

3.1 Dataset

There are two types of features: globules and streaks. The training data set is composed by 807 images, with 1614 binary masks, one for each feature. The testing data set consists of 334 images. We divided the training dataset in training and validation datasets, with 640 and 167 images, respectively. We performed the segmentation individually first on globules and then we used the model that gave the best results for identifying globules on streaks.

3.2 Architecture

We implemented a UNET model, depicted in Annex 3.5. UNET is a model developed by Olaf Ronneberger for biomedical image segmentation. It is composed by a contraction path with convolutional layers and max pooling layers. During the contraction path, after each convolutional layer, we used batch normalization layers to increase stability in the network. The contraction path is followed by an expanding path composed by regular convolutional layers and transposed convolutional layers.

We compiled the model with the Adam optimizer, with a learning rate of 3e-5. As loss functions, we tried various functions, such as: dice loss, jaccard loss, binary cross entropy, and weighted loss function. In the end, we decided to use the dice loss function, which will be explained in the results section. The summary of the model is specified in Annex 3.4.

3.3 Training

Since the training set was quite small, we used the "ImageDataGenerator" class on Keras to perform data augmentation on the training dataset. We also used sample weights to battle imbalance in the dataset.

In Annex 3.1 we can see two graphs, related to the loss and accuracy in the training and validation datasets during training. In the loss function, the loss of the training set tends to zero, while the loss in the validation set has some fluctuations and is decreasing slower than in the training set. In terms of accuracy, both the validation and training sets have similar curves, with accuracy being slightly worse in the validation dataset.

In Annex 3.3 we can see how the loss and accuracy of the model improved during the training, for streaks. These graphs are similar to the ones obtained for globules, with loss function decreasing slower on the validation dataset than on the training dataset and accuracy increasing with similar curves for training and validation datasets.

Training the network with more epochs results in the accuracy graph tending to 98% which is the baseline where every pixel is classified as absence of features, as can be seen in Annex 3.2.

3.4 Results

The metrics used to evaluate the model were: accuracy, precision, recall, intersection over union and dice coefficient.

After training the model with 20 epochs, for globules, we obtained the following results, for each loss function:

	Dice Loss	**Jaccard Loss**	**Weighted Loss**	**Binary Cross Entropy**
Accuracy	89,82%	88,12%	89,39%	56,20%
Precision	1,52%	1,56%	1,53%	1,66%
Recall	8,19%	10,09%	8,68%	44,34%
IoU	49,56%	47,56%	49,32%	49,40%
Dice coefficient	86,21%	84,21%	85,57%	60,00%

Looking at this table, we can conclude that the Binary Cross Entropy loss function is not appropriate for this task, since it has such small values for accuracy when compared to the other functions. Looking at the remaining loss functions, we have achieved similar results. We decided to use the Dice loss function, since it presents better accuracy, intersection over union value and dice coefficient.

For streaks, using the Dice loss function, we have obtained:

Accuracy: 88,59% **Recall:** 11,27%
Precision: 0,18% **IoU:** 49,96%
Dice Coefficient: 83,61%

We could not obtain satisfying results despite everything we tried, as can be seen by the low values for Intersection over Union. The data is very unbalanced and, as such, we could not obtain better results than the baseline. The maximum accuracy we have ever obtained was accompanied by zero in both precision and recall, where the UNET model was detecting solely the absence of features in all images.

Conclusion

CNNs produce better results in image classification tasks. With CNNs, we have obtained satisfactory results when it comes to both binary classification and multi-class classification. For future improvements, the accuracy obtained in classification tasks could be greatly improved by combining the results of different CNN models (an ensemble of CNNs). When it comes to segmentation, while UNET is recognized by being appropriate for biomedical images, we were not able to obtain very good results.

Annexes

1. Task 1

Annex 1.1 – Summary of the model

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 27,560,769
Trainable params: 12,846,081
Non-trainable params: 14,714,688
```

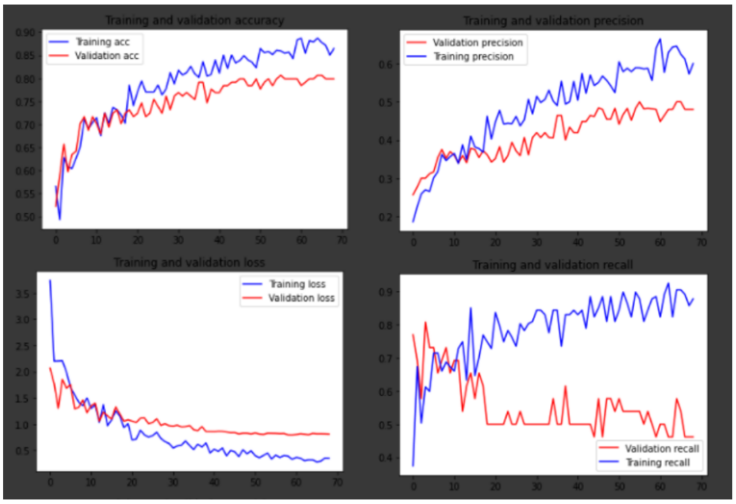
Annex 1.2 – Equation used for bias initialization

$$p_0 = pos/(pos + neg) = 1/(1 + e^{-b_0})$$

$$b_0 = -log_e(1/p_0 - 1)$$

$$b_0 = log_e(pos/neg)$$

Annex 1.3 – Results of training the VGG model



Annex 1.4 – Results of applying the VGG model

	precision	recall	f1-score	support
Benign	0.88	0.90	0.89	304
Malignant	0.56	0.52	0.54	75
accuracy			0.82	379
macro avg	0.72	0.71	0.71	379
weighted avg	0.82	0.82	0.82	379

2. Task 2

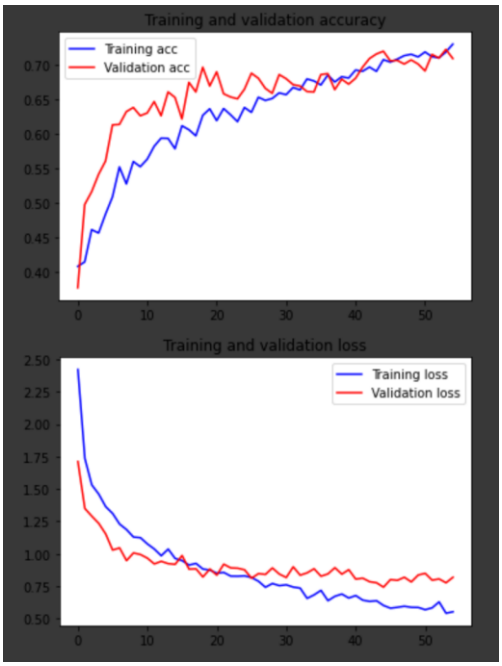
Annex 2.1 - Summary of the model

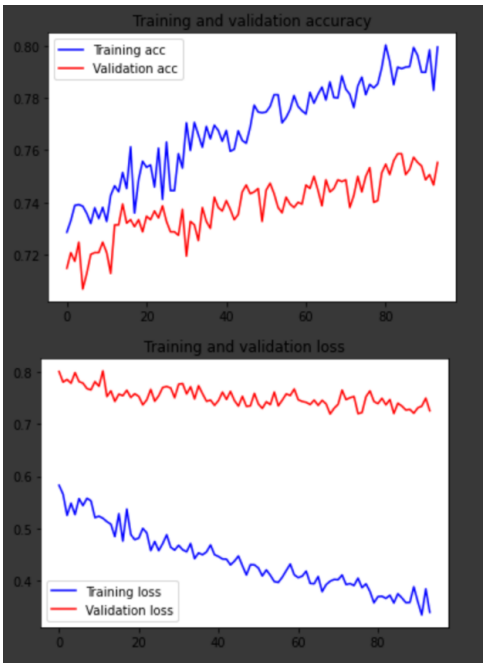
```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense_26 (Dense)             (None, 512)               12845568
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_27 (Dense)             (None, 7)                 3591
=================================================================
Total params: 27,563,847
Trainable params: 12,849,159
Non-trainable params: 14,714,688
_________________________________________________________________
```

Annex 2.2 – Results of applying the model

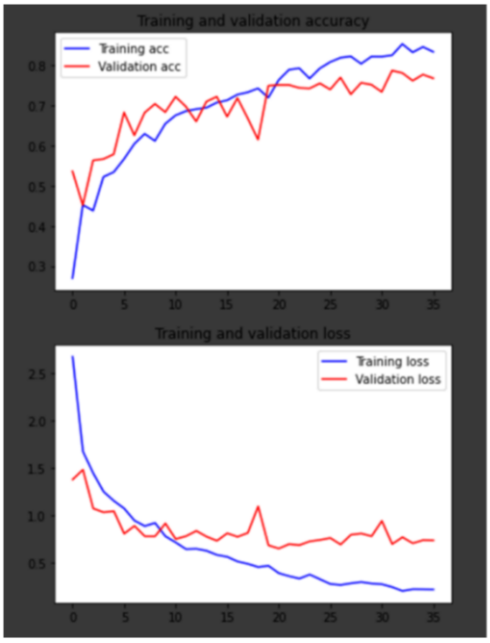
	precision	recall	f1-score	support
AKIEC	0.73	0.55	0.63	66
BCC	0.79	0.82	0.80	103
BKL	0.64	0.75	0.69	220
DF	0.67	0.61	0.64	23
MEL	0.48	0.71	0.57	223
NV	0.94	0.84	0.89	1341
VASC	0.82	0.93	0.87	29
accuracy			0.81	2005
macro avg	0.72	0.74	0.73	2005
weighted avg	0.84	0.81	0.82	2005

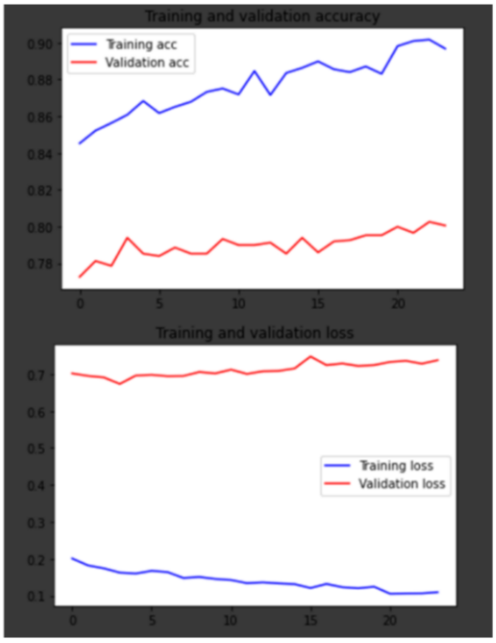
Annex 2.3 – Results





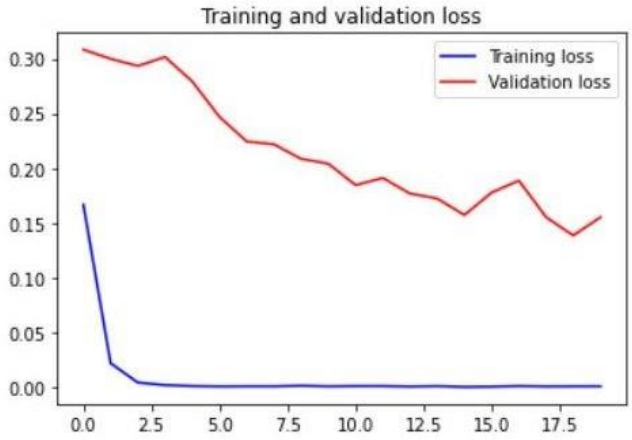
Annex 2.4 – Results

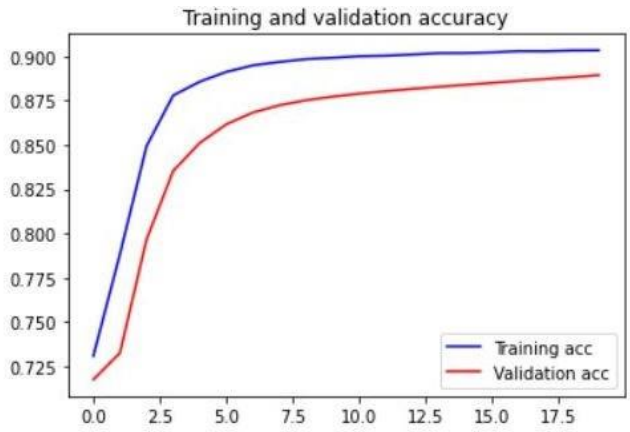




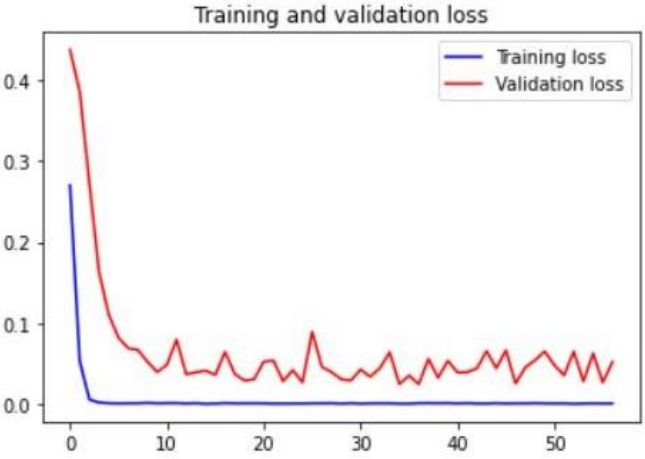
Task 3

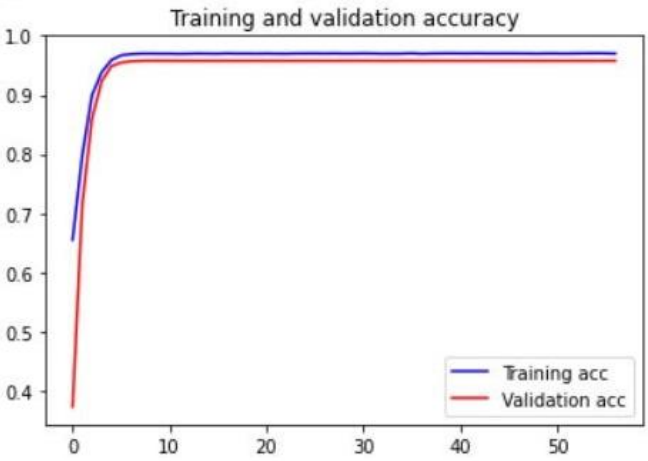
Annex 3.1 - Dice loss function accuracy and loss graphs for globules.



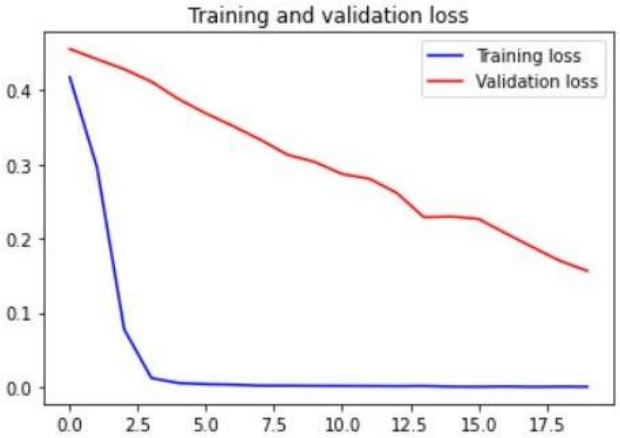


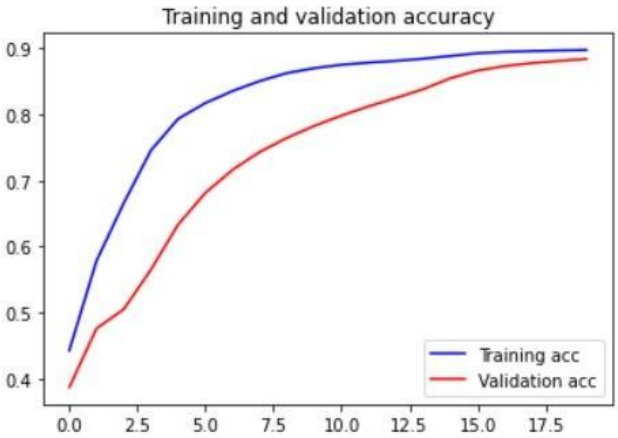
Annex 3.2 - Dice loss function accuracy and loss graphs for globules with 60 epochs.





Annex 3.3 - Dice loss function accuracy and loss graphs for streaks.



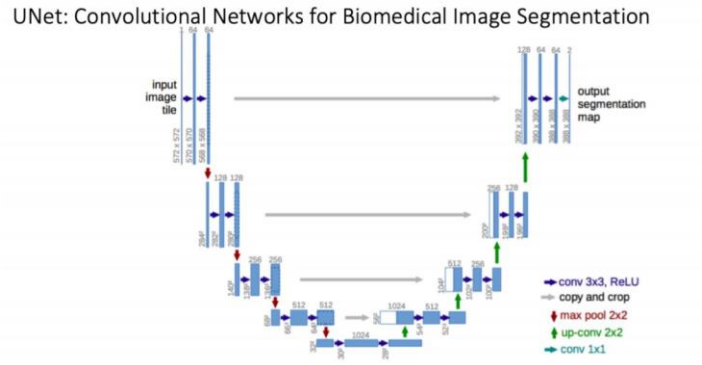


Annex 3.4 - Summary of UNET Model

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 128, 128, 3)	0	
conv2d_1 (Conv2D)	(None, 128, 128, 64)	1792	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 64)	256	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 64)	36928	batch_normalization_1[0][0]
batch_normalization_2 (BatchNor	(None, 128, 128, 64)	256	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 64, 64, 128)	512	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 128)	147584	batch_normalization_3[0][0]
batch_normalization_4 (BatchNor	(None, 64, 64, 128)	512	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_2[0][0]
batch_normalization_5 (BatchNor	(None, 32, 32, 256)	1024	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 256)	590080	batch_normalization_5[0][0]
batch_normalization_6 (BatchNor	(None, 32, 32, 256)	1024	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 256)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_3[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 512)	2048	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 512)	2359808	batch_normalization_7[0][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 512)	2048	conv2d_8[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 512)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 8, 8, 1024)	4719616	max_pooling2d_4[0][0]
conv2d_10 (Conv2D)	(None, 8, 8, 1024)	9438208	conv2d_9[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 16, 16, 512)	4719104	conv2d_10[0][0]
concatenate_1 (Concatenate)	(None, 16, 16, 1024)	0	batch_normalization_8[0][0] conv2d_transpose_1[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 512)	4719104	concatenate_1[0][0]
conv2d_12 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_11[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 32, 32, 256)	1179904	conv2d_12[0][0]
concatenate_2 (Concatenate)	(None, 32, 32, 512)	0	batch_normalization_6[0][0] conv2d_transpose_2[0][0]
conv2d_13 (Conv2D)	(None, 32, 32, 256)	1179904	concatenate_2[0][0]
conv2d_14 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_13[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 64, 64, 128)	295040	conv2d_14[0][0]
concatenate_3 (Concatenate)	(None, 64, 64, 256)	0	batch_normalization_4[0][0] conv2d_transpose_3[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 128)	295040	concatenate_3[0][0]
conv2d_16 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_15[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 128, 128, 64)	73792	conv2d_16[0][0]
concatenate_4 (Concatenate)	(None, 128, 128, 128	0	batch_normalization_2[0][0] conv2d_transpose_4[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 64)	73792	concatenate_4[0][0]
conv2d_18 (Conv2D)	(None, 128, 128, 64)	36928	conv2d_17[0][0]
conv2d_19 (Conv2D)	(None, 128, 128, 3)	195	conv2d_18[0][0]

Total params: 34,521,155
Trainable params: 34,517,315
Non-trainable params: 3,840

Annex 3.5 - UNET Model



4. Task 1 - Code

Annex 4.1 – Bag of Words with SVM classifier code (Approach 1)

```
import cv2
import csv
import numpy as np
import pickle
import argparse

parser = argparse.ArgumentParser(description="Feature detection")
parser.add_argument('-d', '--descriptors', default=False, dest='read_desc', action='store_true')
parser.add_argument('-b', '--bow', default=False, dest='read_bow', action='store_true')
parser.add_argument('-t', '--train', default=False, dest='read_svm', action='store_true')
args = parser.parse_args()

keys = {
    'benign': 0,
    'malignant': 1,
}

train_img_path = '../data/task1/training/ISBI2016_ISIC_Part3_Training_Data'
train_data_path = '../data/task1/training/ISBI2016_ISIC_Part3_Training_GroundTruth.csv'
test_img_path = '../data/task1/test/ISBI2016_ISIC_Part3_Test_Data'
test_data_path = '../data/task1/test/ISBI2016_ISIC_Part3_Test_GroundTruth.csv'



# Get descriptors using sift
def get_descriptors(detector):
    if args.read_desc:
        print('Loading descriptors...')
        with open('../data/feature_detection/all_descriptors.pkl', 'rb') as inputfile:
            return pickle.load(inputfile)

    print('Calculating descriptors...')
    labels = []
    all_descriptors = []
    with open(train_data_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            img = cv2.imread(train_img_path + '/' + row[0] + '.jpg', 0) # read image as grayscale
            keypoints, descriptors = detector.detectAndCompute(img, None) # compute descriptors and keypoints
            all_descriptors.extend(descriptors)
    with open('../data/feature_detection/all_descriptors.pkl', 'wb') as outputfile:
        pickle.dump(all_descriptors, outputfile)
    return all_descriptors


# Train SVM classifier
def train(bow_extractor, detector):
    if args.read_svm:
        print('Loading trained svm...')
        return cv2.ml.SVM_load('../data/feature_detection/svm.pkl')
    print('Training...')
    svm = cv2.ml.SVM_create()
    svm.setType(cv2.ml.SVM_C_SVC)
    svm.setKernel(cv2.ml.SVM_RBF)
    svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
    img_labels = []
    img_descriptors = []
    with open(train_data_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            img = cv2.imread(train_img_path + '/' + row[0] + '.jpg', 0) # read image as grayscale
            keypoints, descriptors = detector.detectAndCompute(img, None) # compute descriptors and keypoints
            descriptor = bow_extractor.compute(img, keypoints)
            img_descriptors.append(descriptor[0])
            img_labels.append(keys[row[1]])
    svm.train(np.array(img_descriptors), cv2.ml.ROW_SAMPLE, np.array(img_labels))
    svm.save('../data/feature_detection/svm.pkl')
    return svm
```

```python
# Use SVM classifier on test dataset
def test(bow_extractor, svm, detector):
    print('Testing...')
    total = 0
    right = 0
    benign_right = 0
    benign_wrong = 0 # malignant picture classified as benign
    malignant_right = 0
    malignant_wrong = 0 # benign picture classified as malignant
    with open(test_data_path) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            img = cv2.imread(test_img_path + '/' + row[0] + '.jpg', 0) # read image as grayscale
            keypoints, descriptors = detector.detectAndCompute(img, None) # compute descriptors and keypoints
            b = []
            b.append(bow_extractor.compute(img, keypoints)[0])
            prediction = svm.predict(np.array(b))
            pred = np.squeeze(prediction[1].astype(int))
            if pred == int(float(row[1])):
                right += 1
                if pred == 0:
                    benign_right += 1
                else:
                    malignant_right += 1
            else:
                if pred == 0:
                    benign_wrong += 1
                else:
                    malignant_wrong += 1
            total += 1
    print('Total: ' + str(total))
    print('Malignant correctly identified: ' + str(malignant_right))
    print('Benign correctly identified: ' + str(benign_right))
    print('Malignant identified as benign: ' + str(benign_wrong))
    print('Benign identified as malignant: ' + str(malignant_wrong))
    print('Accuracy: ' + str(float(right) / float(total)))


def main():
    # create feature detector with SIFT
    detector = cv2.xfeatures2d.SIFT_create()
    # obtain descriptors
    all_descriptors = np.array(get_descriptors(detector))
    # create or load bow
    if args.read_bow:
        print('Loading bag of words...')
        with open('../data/feature_detection/bow.pkl', 'rb') as inputfile:
            bow_cluster = pickle.load(inputfile)
    else:
        print('Training Bag of Words...')
        bow_trainer = cv2.BOWKMeansTrainer(300)
        bow_cluster = bow_trainer.cluster(all_descriptors)
        with open('../data/feature_detection/bow.pkl', 'wb') as outputfile:
            pickle.dump(bow_cluster, outputfile)
    matcher = cv2.FlannBasedMatcher()
    bow_extractor = cv2.BOWImgDescriptorExtractor(detector, matcher)
    bow_extractor.setVocabulary(bow_cluster)
    # obtain svm classifier
    svm = train(bow_extractor, detector)
    # use classifier on test dataset
    test(bow_extractor, svm, detector)

main()
```

Annex 4.2 – Binary classification with VGG-16 (Approach 2)

```
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import load_model, Model
```

```
import random as rn
import numpy as np

np.random.seed(42)
rn.seed(12345)
session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1,
                                        inter_op_parallelism_threads=1)
tf.random.set_seed(1234)
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
#K.set_session(sess)
tf.compat.v1.keras.backend.set_session(sess)
```

```
vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in vgg_conv.layers:
    layer.trainable = False
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
METRICS = [
      keras.metrics.BinaryAccuracy(name='accuracy'),
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.AUC(name='auc'),
      keras.metrics.TruePositives(name='tp'),
      keras.metrics.FalsePositives(name='fp'),
]

def make_model(metrics = METRICS, output_bias=None):

  if output_bias is not None:
    output_bias = tf.keras.initializers.Constant(output_bias)

  headModel = vgg_conv.output
  headModel = keras.layers.Flatten(name="flatten")(headModel)
  # headModel = keras.layers.Dense(512, activation="relu")(headModel)
  # headModel = keras.layers.Dropout(0.5)(headModel)
  headModel = keras.layers.Dense(512, activation="relu")(headModel)
  headModel = keras.layers.Dropout(0.5)(headModel)
  headModel = keras.layers.Dense(1, activation="sigmoid")(headModel)

  model = Model(inputs=vgg_conv.input, outputs=headModel)

  model.compile(
      optimizer=keras.optimizers.Adam(lr=1e-5),
      loss=keras.losses.BinaryCrossentropy(),
      metrics=metrics)

  return model
```

```
pos = 173
neg = 727
initial_bias = np.log([pos/neg])
print(initial_bias)
```

```
model = make_model(output_bias=initial_bias)
#model.save_weights('initial_weights_1fc_256.h5')
model.summary()
```

```
train_datagen = ImageDataGenerator(
      preprocessing_function=preprocess_input,
      rotation_range=10,
      width_shift_range=0.1,
      height_shift_range=0.1,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_batchsize = 64
val_batchsize = 16
train_generator = train_datagen.flow_from_directory(
        directory="/content/Dataset2/train2",
        target_size=(224, 224),
        batch_size=train_batchsize,
        class_mode='binary',
        shuffle=True,
        seed=42)

validation_generator = validation_datagen.flow_from_directory(
        directory="/content/Dataset2/valid2",
        target_size=(224, 224),
        batch_size=val_batchsize,
        class_mode='binary',
        shuffle=False,
        seed=42)
```

```
pos = 173
neg = 727
total = 900
weight_for_0 = (1 / neg)*(total)/2.0
weight_for_1 = (1 / pos)*(total)/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print(weight_for_0)
print(weight_for_1)
```

```
# Optional Finetuning

#model.load_weights('/content/vgg16_1fc512_head_038.h5')
for layer in vgg_conv.layers:
    layer.trainable = True
for layer2 in model.layers:
    print(layer2, layer2.trainable)

opt = keras.optimizers.Adam(lr=1e-6)
model.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

```
# Train the model

#model = make_model()
model.load_weights('/content/vgg16_1fc256_head_003.h5')
#model = load_model("/content/vgg16_1fc256_head_003.h5")

checkpoint = tf.keras.callbacks.ModelCheckpoint('vgg16_1fc256_head_{epoch:03d}.h5', monitor='val_precision', verbose=1, save_best_only=True,  mode='max', save_freq='epoch')
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='min')
history = model.fit(
     train_generator,
     epochs=100,
     class_weight=class_weight,
     validation_data=validation_generator,
     callbacks=[checkpoint,early],
     verbose=1,
     use_multiprocessing=False,
     max_queue_size=10,               # maximum size for the generator queue
     workers=1,
     )
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
val_precision = history.history['val_precision']
precision = history.history['precision']
val_recall = history.history['val_recall']
recall = history.history['recall']

epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')

plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()
plt.plot(epochs, val_precision, 'r', label='Validation precision')
plt.plot(epochs, precision, 'b', label='Training precision')
plt.title('Training and validation precision')
plt.legend()
plt.figure()
plt.plot(epochs, val_recall, 'r', label='Validation recall')
plt.plot(epochs, recall, 'b', label='Training recall')
plt.title('Training and validation recall')
plt.legend()

plt.show()
```

```
tsdata = ImageDataGenerator(preprocessing_function=preprocess_input)
from sklearn.metrics import classification_report, confusion_matrix

test_generator = tsdata.flow_from_directory(directory="/content/Dataset/test", target_size=(224,224), class_mode='binary',shuffle=False, batch_size=1, seed=42)

model.load_weights('/content/vgg16_1fc256_head_068.h5')
results = model.evaluate(test_generator)
print(results)

test_generator.reset()
predIdxs = model.predict(test_generator)
predictions = predIdxs >=0.5

print(classification_report(test_generator.classes, predictions, target_names=test_generator.class_indices.keys()))
print("Confusion Matrix:")
print(confusion_matrix(test_generator.classes, predictions))
```

5. Task 2 - Code

```
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import load_model, Model
import random as rn
import numpy as np

np.random.seed(42)
rn.seed(12345)
session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1,
                                        inter_op_parallelism_threads=1)
tf.random.set_seed(1234)
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
#K.set_session(sess)
tf.compat.v1.keras.backend.set session(sess)
 vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

 for layer in vgg_conv.layers:
     layer.trainable = False
 for layer in vgg_conv.layers:
     print(layer, layer.trainable)
```

```
def make_model():

  headModel = vgg_conv.output
  headModel = keras.layers.Flatten(name="flatten")(headModel)
  headModel = keras.layers.Dense(512, activation="relu")(headModel)
  headModel = keras.layers.Dropout(0.5)(headModel)
  headModel = keras.layers.Dense(7, activation="softmax")(headModel)

  model = Model(inputs=vgg_conv.input, outputs=headModel)

  model.compile(
      optimizer=keras.optimizers.Adam(lr=1e-4),
      loss=keras.losses.categorical_crossentropy,
      metrics=['accuracy'])

  return model

model = make_model()
#model.save_weights('initial_vgg16_1fc_512_head.h5')
model.summary()
```

```
# adding regularization
regularizer = tf.keras.regularizers.l2(5e-4)

for layer in model.layers:
    for attr in ['kernel_regularizer']:
        if hasattr(layer, attr):
          print("I have")
          setattr(layer, attr, regularizer)

model_json = model.to_json()
model = tf.keras.models.model_from_json(model_json)

model.compile(
    optimizer=keras.optimizers.Adam(lr=1e-4),
    loss=keras.losses.categorical_crossentropy,
    metrics=['accuracy'])

# Reload the model weights
model.load_weights('/content/drive/My Drive/VCOM/Task2/tuning2_1fc128_head_021.h5')
```

```
train_datagen = ImageDataGenerator(
      preprocessing_function=preprocess_input,
      rotation_range=10,
      width_shift_range=0.1,
      height_shift_range=0.1,
      #shear_range=0.1,
      zoom_range=0.1,
      horizontal_flip=True,
      vertical_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_batchsize = 128
val_batchsize = 32
train_generator = train_datagen.flow_from_directory(
        directory="/content/Dataset/train",
        target_size=(224, 224),
        batch_size=train_batchsize,
        shuffle=True,
        seed=42)

validation_generator = validation_datagen.flow_from_directory(
        directory="/content/Dataset/valid",
        target_size=(224, 224),
        batch_size=val_batchsize,
        shuffle=False,
        seed=42)
```

```
akiec = 212
bcc = 334
bkl = 714
df = 74
mel = 723
nv = 4358
vasc = 91

total = 6506
# Scaling by total/7 helps keep the loss to a similar magnitude.
weight_for_0 = (1 / akiec) * (total)/7.0
weight_for_1 = (1 / bcc) * (total)/7.0
weight_for_2 = (1 / bkl) * (total)/7.0
weight_for_3 = (1 / df) * (total)/7.0
weight_for_4 = (1 / mel) * (total)/7.0
weight_for_5 = (1 / nv) * (total)/7.0
weight_for_6 = (1 / vasc) * (total)/7.0

class_weight = {0: weight_for_0, 1: weight_for_1, 2: weight_for_2, 3: weight_for_3, 4: weight_for_4,
                5: weight_for_5, 6: weight_for_6}
```

```
# Optional Finetuning

#model.load_weights('/content/tuning2_1fc512_head_038.h5')
for layer in vgg_conv.layers:
    layer.trainable = True
for layer2 in model.layers:
    print(layer2, layer2.trainable)

opt = keras.optimizers.Adam(lr=1e-6)
model.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

```
# Train the model
model = load_model('/content/grafico_1fc512_001.h5')
#model.load_weights()

checkpoint = tf.keras.callbacks.ModelCheckpoint('vgg16_1fc512_{epoch:03d}.h5', monitor='val_accuracy', verbose=1, save_best_only=True,  mode='max', save_freq='epoch')
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=15, verbose=1, mode='min')
history = model.fit(
      train_generator,
      epochs=100,
      class_weight=class_weight,
      validation_data=validation_generator,
      callbacks=[checkpoint,early],
      verbose=1,
      use_multiprocessing=False,
      max_queue_size=10,              # maximum size for the generator queue
      workers=1,
      )
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')

plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
from sklearn.metrics import classification_report, confusion_matrix
tsdata = ImageDataGenerator(preprocessing_function=preprocess_input)
test_generator = tsdata.flow_from_directory(directory="/content/Dataset/test", target_size=(224,224), shuffle=False, batch_size=1, seed=42)

#model = load_model('test0002.h5')
model.load_weights('/content/finetuning3_1fc512_all_027.h5')

results = model.evaluate(test_generator)
print(results)
test_generator.reset()
predIdxs = model.predict(test_generator)
predIdxs = np.argmax(predIdxs, axis=1)
print(classification_report(test_generator.classes, predIdxs,
  target_names=test_generator.class_indices.keys()))
print(confusion_matrix(test_generator.classes, predIdxs))
```

6. Task 3 - Code

```
import random as rn
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.layers.merge import add, concatenate
from keras.layers import Conv2D, Conv2DTranspose, Dropout, Input, UpSampling2D, MaxPooling2D, Concatenate, BatchNormalization
from keras.models import Model
from keras.metrics import Accuracy, BinaryAccuracy, Precision, Recall, AUC, TruePositives, FalsePositives, MeanIoU, TrueNegatives, FalseNegatives
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.losses import BinaryCrossentropy
from keras.optimizers import Adam
from keras import backend as K
import matplotlib.pyplot as plt

# Lesion being analysed: 'globules' or 'streaks'
LESION = 'globules'

def obtain_train_generator():
  SEED = 42
  # ImageDataGenerator used to obtain the training set with data augmentation
  train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,          # data augmentation - rotation
    width_shift_range=0.2,      # data augmentation - horizontal shift
    height_shift_range=0.2,     # data augmentation - vertical shift
    horizontal_flip=True,       # data augmentation - flip image
    zoom_range=0.2,             # data augmentation - zoom
    brightness_range=[0.8,1.2]  # data augmentation - brightness
  )
  batchsize = 80
```

```
  train_generator = train_datagen.flow_from_directory(
    directory="/content/Dataset/training",
    classes = ['ISBI2016_ISIC_Part2B_Training_Data'],
    target_size=(128, 128),
    batch_size=batchsize,
    class_mode = None,
    seed=SEED
  )

  # obtain masks according to the lesion specified in LESION
  masks_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2,
    brightness_range=[0.8,1.2]
  )
  masks_generator = masks_datagen.flow_from_directory(
    directory="/content/Dataset/training/ISBI2016_ISIC_Part2B_Training_GroundTruth",
    classes = [LESION],
    target_size=(128, 128),
    batch_size=batchsize,
    class_mode = None,
    seed=SEED
  )
  train_generator = zip(train_generator, masks_generator)
```

```
  # addition of class weights, since the data set is so unbalanced
  sample_weight = np.zeros((batchsize, 128))
  sample_weight[:, 0] = 1
  sample_weight[:, 1] = 90
  for (img, mask) in train_generator:
    yield (img, mask, sample_weight)

def obtain_validation_generator():
  # obtain validation images
  valid_datagen = ImageDataGenerator(
    rescale=1./255
  )
  batchsize = 80
  valid_generator = valid_datagen.flow_from_directory(
    directory="/content/Dataset/validation",
    classes = ['data'],
    target_size=(128, 128),
    batch_size=batchsize,
    class_mode = None
  )

  # obtain validation masks
  masks_datagen = ImageDataGenerator(
    rescale=1./255
  )
  masks_generator = masks_datagen.flow_from_directory(
    directory="/content/Dataset/validation/groundtruth",
    classes = [LESION],
    target_size=(128, 128),
    batch_size=batchsize,
    class_mode = None
  )
  valid_generator = zip(valid_generator, masks_generator)
  for (img, mask) in valid_generator:
    yield (img, mask)

def obtain_test_generator():
  # obtain test images
  test_datagen = ImageDataGenerator(
    rescale=1./255
  )
  batchsize = 1
  test_generator = test_datagen.flow_from_directory(
    directory="/content/Dataset/test",
    classes = ['ISBI2016_ISIC_Part2B_Test_Data'],
    target_size=(128, 128),
    batch_size=batchsize,
    class_mode = None
  )

  # obtain test masks
  masks_datagen = ImageDataGenerator(
    rescale=1./255
  )
```

```
masks_generator = masks_datagen.flow_from_directory(
  directory="/content/Dataset/test/ISBI2016_ISIC_Part2B_Test_GroundTruth",
  classes = [LESION],
  target_size=(128, 128),
  batch_size=batchsize,
  class_mode = None,
  shuffle=False
)
test_generator = zip(test_generator, masks_generator)
for (img, mask) in test_generator:
  yield (img, mask)
```

```
# Dice loss function
def dice_loss(y_true, y_pred):
  return 1-dice_coef(y_true, y_pred)

# Dice coefficients function
def dice_coef(y_true, y_pred, smooth=1):
  intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
  return (2. * intersection + smooth) / (K.sum(K.square(y_true),-1) + K.sum(K.square(y_pred),-1) + smooth)

# Jaccard loss function
def jaccard_loss(y_true, y_pred, smooth=1):
  intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
  sum_ = K.sum(K.abs(y_true) + K.abs(y_pred), axis=-1)
  jac = (intersection + smooth) / (sum_ - intersection + smooth)
  return (1 - jac) * smooth

# Weighted loss function
def weighted_loss(y_true, y_pred):
  pos_weight = 0.01
  loss = tf.nn.weighted_cross_entropy_with_logits(
    y_true, y_pred, pos_weight, name=None
  )
  return loss
```

```
# Creation of UNET model
def create_unet_model():
  inputs = Input(shape=(128, 128, 3))
  x1 = Conv2D(64, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
  x1 = BatchNormalization()(x1)
  x1 = Conv2D(64, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x1)
  x1 = BatchNormalization()(x1)
  x2 = MaxPooling2D(pool_size=(2, 2))(x1)
  x2 = Conv2D(128, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x2)
  x2 = BatchNormalization()(x2)
  x2 = Conv2D(128, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x2)
  x2 = BatchNormalization()(x2)
  x3 = MaxPooling2D(pool_size=(2, 2))(x2)
  x3 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x3)
  x3 = BatchNormalization()(x3)
  x3 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x3)
  x3 = BatchNormalization()(x3)
  x4 = MaxPooling2D(pool_size=(2, 2))(x3)
  x4 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x4)
  x4 = BatchNormalization()(x4)
  x4 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x4)
  x4 = BatchNormalization()(x4)
  x5 = MaxPooling2D(pool_size=(2, 2))(x4)
  x5 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x5)
  x5 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x5)
```

```
x6 = Conv2DTranspose(512, (3, 3), strides = (2, 2), padding = 'same')(x5)
x6 = Concatenate()([x4, x6])
x6 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2DTranspose(256, (3, 3), strides = (2, 2), padding = 'same')(x6)
x6 = Concatenate()([x3, x6])
x6 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2DTranspose(128, (3, 3), strides = (2, 2), padding = 'same')(x6)
x6 = Concatenate()([x2, x6])
x6 = Conv2D(128, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2D(128, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2DTranspose(64, (3, 3), strides = (2, 2), padding = 'same')(x6)
x6 = Concatenate()([x1, x6])
x6 = Conv2D(64, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2D(64, (3, 3), activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(x6)
x6 = Conv2D(3, (1, 1), activation = 'sigmoid')(x6)

model = Model(inputs, x6)

metrics = [
  BinaryAccuracy(name='accuracy'),
  Precision(name='precision'),
  Recall(name='recall'),
  TruePositives(name='tp'),
  FalsePositives(name='fp'),
  MeanIoU(name='iou', num_classes=2),
  dice_coef
]
```

```
  # Model compilation with Dice loss function
  model.compile(
      optimizer=Adam(lr=3e-5),
      loss=dice_loss,
      metrics=metrics,
      sample_weight_mode='temporal'
  )
  return model

model = create_unet_model()

EPOCHS = 20
checkpoint = ModelCheckpoint('unet.hdf5', monitor='val_loss', verbose=1, save_best_only=True)
train_generator = obtain_train_generator()
validation_generator = obtain_validation_generator()

# Train model
model_history = model.fit_generator(
    train_generator, epochs=EPOCHS,
    steps_per_epoch=8,
    validation_steps=3,
    validation_data=validation_generator,
    callbacks=[checkpoint],
    verbose=1,
    use_multiprocessing=False,
    max_queue_size=10,
    workers=1
)
```

```
# Show train results
acc = model_history.history['accuracy']
val_acc = model_history.history['val_accuracy']
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
val_precision = model_history.history['val_precision']
precision = model_history.history['precision']
recall = model_history.history['recall']
val_recall = model_history.history['val_recall']
dice_coef = model_history.history['dice_coef']
val_dice_coef = model_history.history['val_dice_coef']

epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')

plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()
plt.plot(epochs, val_precision, 'r', label='Validation precision')
plt.plot(epochs, precision, 'b', label='Training precision')
plt.title('Training and validation precision')
plt.legend()
plt.figure()
plt.plot(epochs, val_recall, 'r', label='Validation recall')
plt.plot(epochs, recall, 'b', label='Training recall')
plt.title('Training and validation recall')
plt.legend()
plt.figure()
plt.plot(epochs, val_dice_coef, 'r', label='Validation dice coef')
plt.plot(epochs, dice_coef, 'b', label='Training dice coef')
plt.title('Training and validation dice coef')
plt.legend()

plt.show()

# Evaluate trained model on test dataset
test_generator = obtain_test_generator()
results = model.evaluate(test_generator, steps=335)
print("Loss: " + str(results[0]))
print("Accuracy: " + str(results[1]))
print("Precision: " + str(results[2]))
print("Recall: " + str(results[3]))
print("TP: " + str(results[4]))
print("FP: " + str(results[5]))
print(results)
```