Measuring fairness in automatic skin disease diagnosis with a convolutional neural network

The aim of this project is to measure fairness in automatic skin disease diagnosis using a convolutional neural network. The dataset used for this work is a compilation of various images of skin diseases taken by the medical personnel from the pedriatic ward at Sant'Orsola hospital in Bologna. These images were collected using consumer-grade cameras, resulting in considerable variance in illumination, angle, and quality. The first part of this work involes processing the images to create a dataset with uniform image sizes. The second part focuses on determining the skin tone of each sample using the Individual Typology Angle. The third part involves training a simple convolutional neural network and evaluating its performance across different skin tones by computing traditional metrics for each skin tone and different fairness metrics.

First step: preprocessing the dataset with a cropping algorithm

1. Initial setup: importing (and installing) the necessary libraries

2. Generating crops with the cropping algorithm

The dataset is composed of about 8000 images taken with different consumer cameras. For this reason, the images are affected by high variability in illumination, angle, quality, size, blurriness, etc. Furthermore, the regions which contain the skin disease are not isolated.

The algorithm used to process such images is adapted by from the one developed by Alessandro D'Amico, Riccardo Murgia and Mazeyar Moeini Feizabadi [https://github.com/eskinderit/experiments-synthetic-generation-clinical-skin-images/blob/main/generate_images.ipynb.]. The functions used in the next cells are imported from the 'generate_images.py' file.

The idea is to detect the regions of the image in which the skin disease is present by

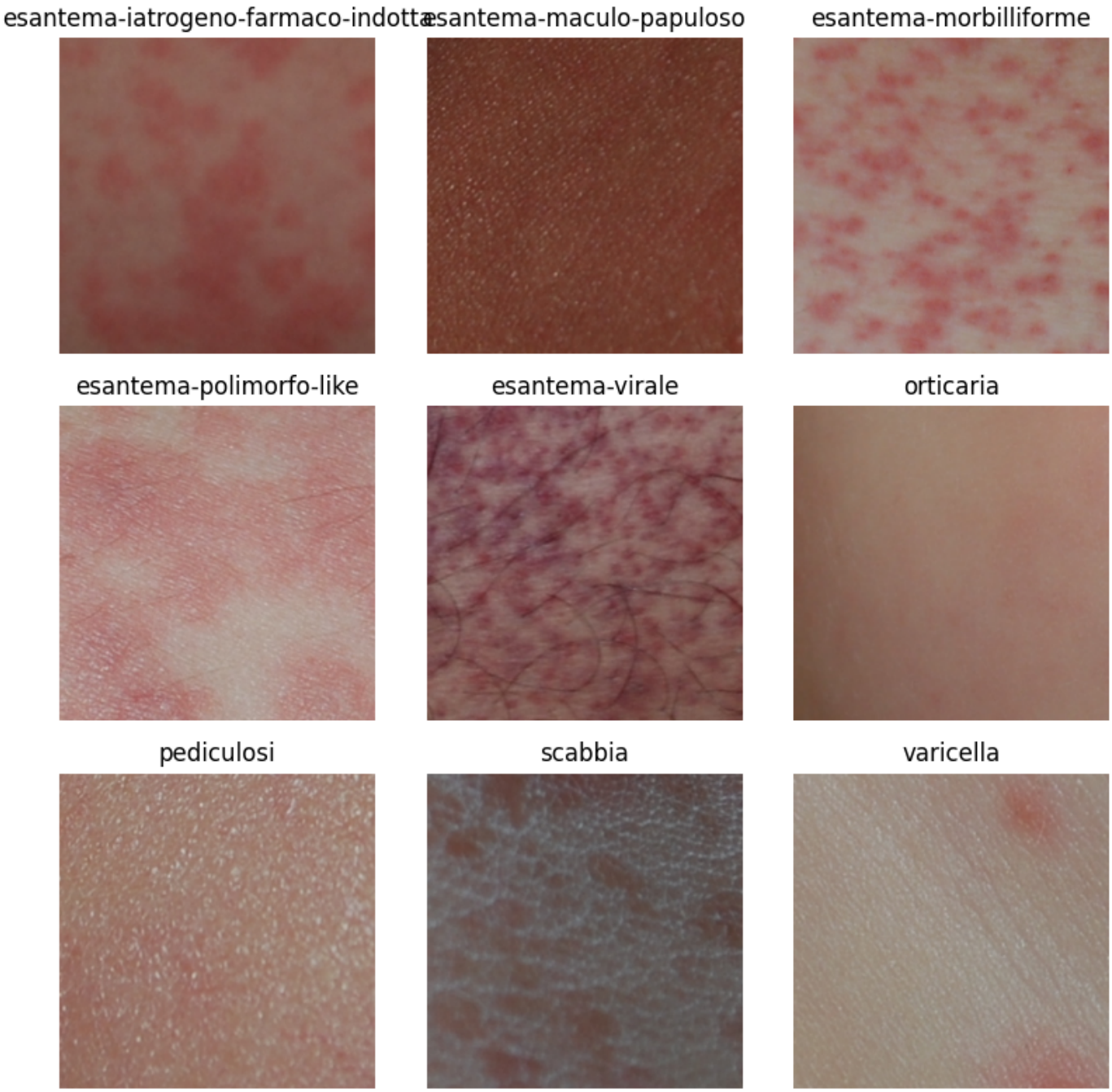
considering the binary mask of the image and employing a sliding-window approach. In particular:

1. The algorithm starts from the top-left corner of the image and takes a fixed-size patch (in our case the size is 256x256).
2. The algorithm extracts the binary mask relative to that patch and computes the disease coverage in that patch, i.e. the ratio of the positive label (1) in the mask over the negative label (0). This is done to assess the presence of contrast, which could indicate that the skin disease is present. If the disease coverage is higher than a threshold (high contrast), the patch is taken. If it is lower, the patch is discarded.
3. Repeat the process by sliding the patch extractor by a certain amount. This process results in a lot of patches, the majority of which overlap and, thus, are redundant. To overcome this issue, a non-maxima suppression procedure is run over the obtained patches: if two patches overlap over a certain limit, the patch with lower disease coverage is discarded. The last step involves discarding those crops which exhibit low contrast, in hope of cleaning the dataset from poorly illuminated or blurry patches.

Important: the folder indicated by the 'main_path' variable should contain the nine disease folders of the dataset for this code to work.

2.1 Examples of generated crops

Important: after the previous steps, a 'colab/' folder with the nine diseases subfolders was created manually. Each disease subfolder contains **just the cropped images** (no masks). From now on, the code will only work if the 'colab/' folder is present. Alternatively, the following code will create the aforementioned folder with the subfolders (it will take some time).



We can clearly see that, despite our efforts, not all crops are of good quality: the generated crops contain several artifacts, such as blurrying, poor illumination, regions in which the disease is not present, unnecessary body parts and edges.

Second step: computation of the skin color using the Individual Typology Angle

1. Initial setup: importing (and installing) the necessary libraries

2. Creating new masks

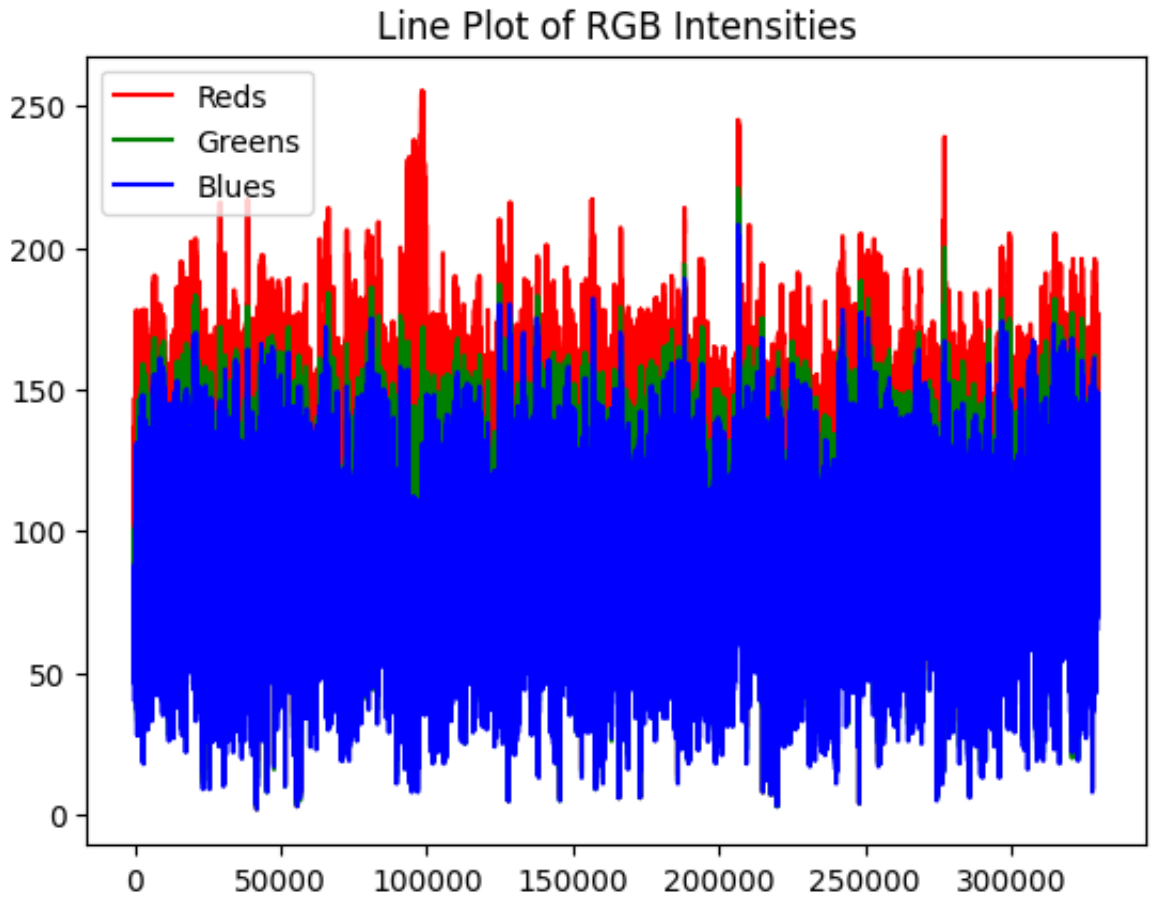
The following code will:

- create new binary masks and store them in the folder named 'crops_masks' in the directory relative to each disease; this is necessary to perform the 'bitwise and' operation between the original image and the binary mask, to assure that the ITA computation involves only the regions of skin in which the disease is **not** present.
- copy the cropped images from the 'crops' directory to the 'crops_masks' directory.

3. Plotting RGB distributions

```
  0%|          | 0/27683 [00:00<?, ?it/s]
  0%|          | 0/11994 [00:00<?, ?it/s]
  0%|          | 0/5919 [00:00<?, ?it/s]
  0%|          | 0/12398 [00:00<?, ?it/s]
  0%|          | 0/85080 [00:00<?, ?it/s]
  0%|          | 0/134422 [00:00<?, ?it/s]
  0%|          | 0/10369 [00:00<?, ?it/s]
  0%|          | 0/32390 [00:00<?, ?it/s]
  0%|          | 0/9405 [00:00<?, ?it/s]

<matplotlib.legend.Legend at 0x29becf719a0>
```



4. Extracting skin tones with Individual Typology Angle

The Individual Typology Angle (ITA) is used to determine the skin tone of an image. Its formula is defined as:

$$ITA = arctan(\frac{L^* - 50}{b^*}) \cdot \frac{180^\circ}{\pi}$$

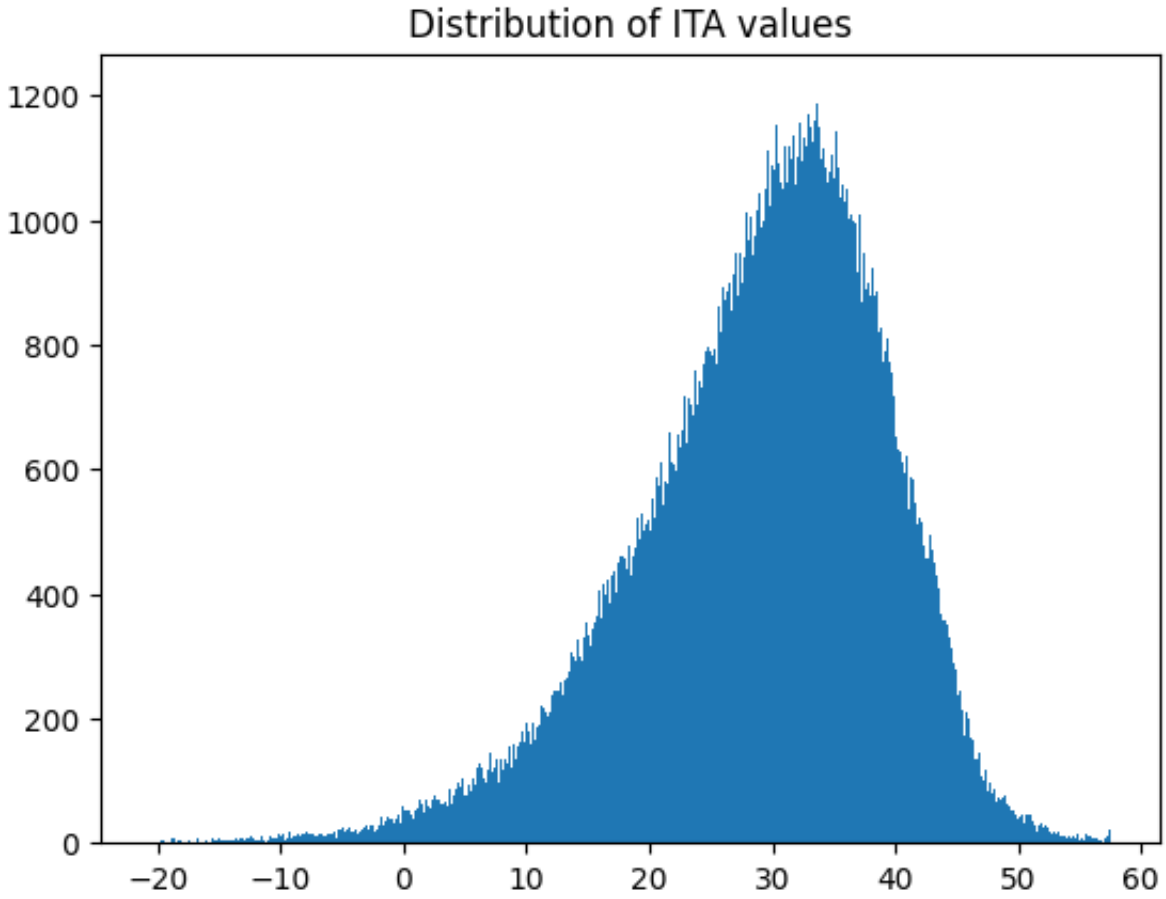
in the CIELAB color space, where L^* represents the perceptual lightness and b^* represents the blue-yellow opponents. To compute the ITA value, the *get_ita* function from the derm-ita library was used.

```
  0%|          | 0/27683 [00:00<?, ?it/s]
esantema-iatrogeno-farmaco-indotta processed!
  0%|          | 0/11994 [00:00<?, ?it/s]
esantema-maculo-papuloso processed!
  0%|          | 0/5919 [00:00<?, ?it/s]
esantema-morbilliforme processed!
  0%|          | 0/12398 [00:00<?, ?it/s]
esantema-polimorfo-like processed!
  0%|          | 0/85080 [00:00<?, ?it/s]
esantema-virale processed!
  0%|          | 0/134422 [00:00<?, ?it/s]
orticaria processed!
```

```
  0%|          | 0/10369 [00:00<?, ?it/s]
pediculosi processed!
  0%|          | 0/32390 [00:00<?, ?it/s]
scabbia processed!
  0%|          | 0/9405 [00:00<?, ?it/s]
varicella processed!
```

4.1 Checking the distribution of the ITA values and finding the optimal ranges with Gaussian Mixture

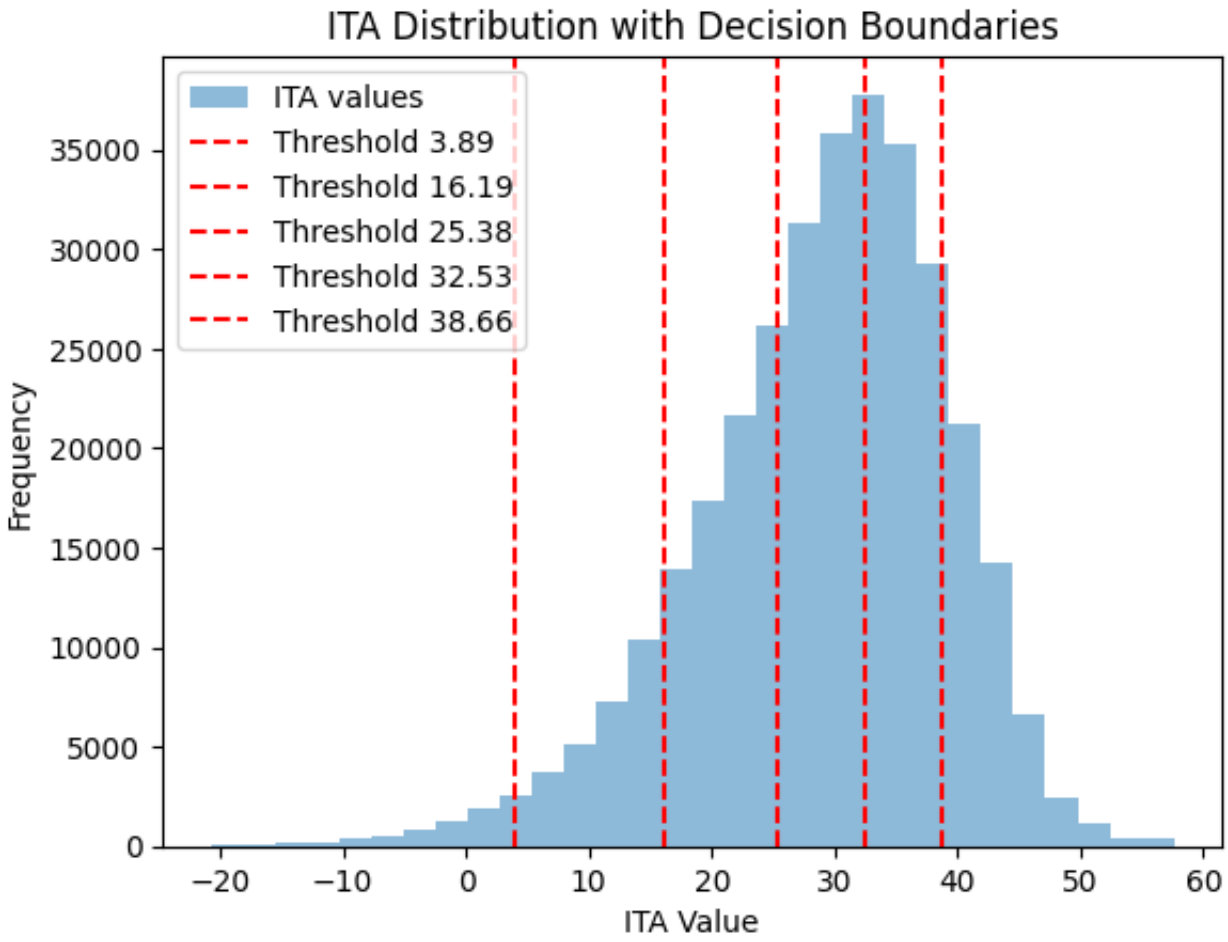
We start by plotting the distribution of the ITA values across the dataset. We can see that the distribution resembles a Gaussian.



We know retrieve the decision boundaries to associate each ITA value to its corresponding skin tone:

- we fit the ITA values distribution with a Gaussian Mixture with six components
- we retrieve the decision boundaries (thresholds) relative to each Gaussian analytically

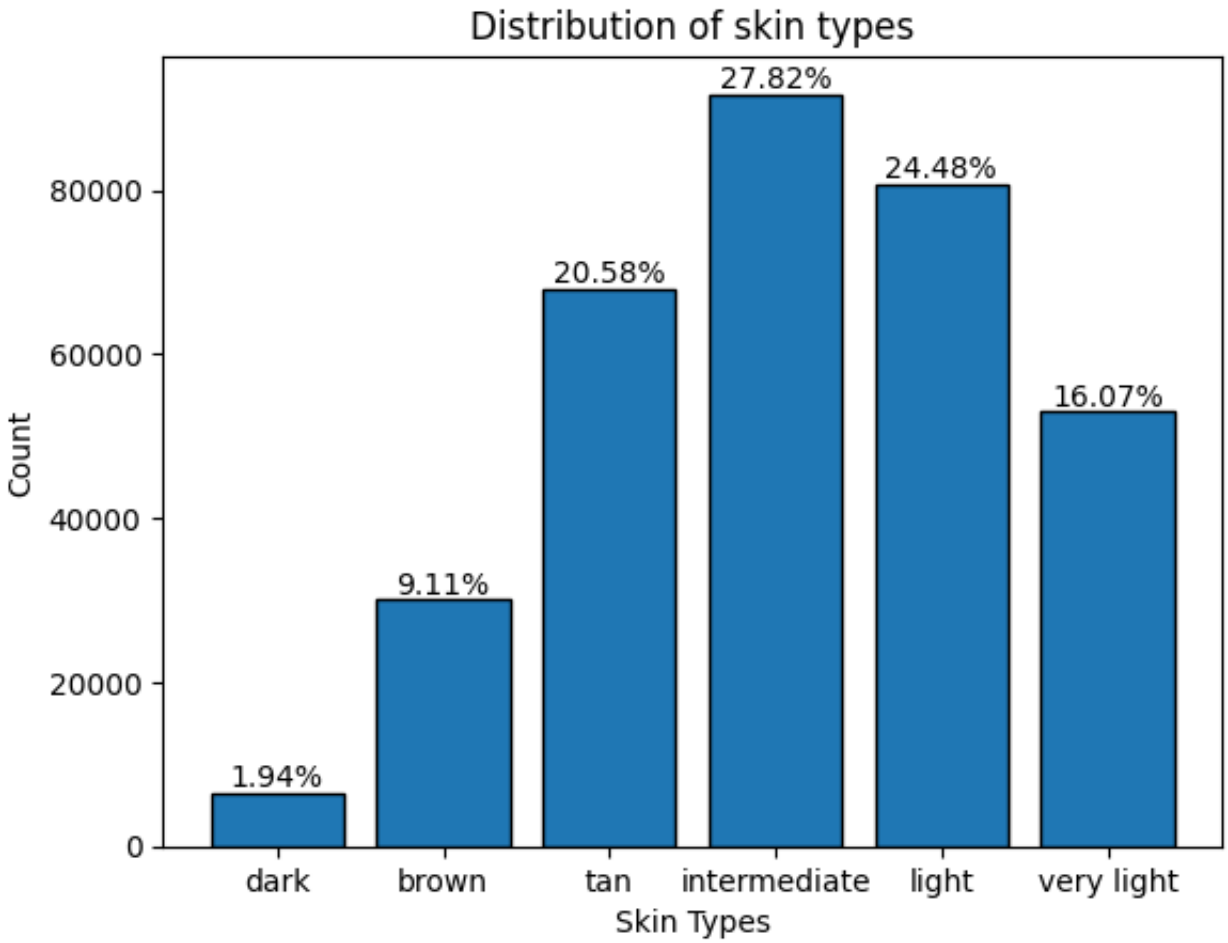
```
[25.02878746]
GMM-based cluster centers (means): [-3.31561216 11.09737726 21.29170701 29.4587466
3 35.60450325 41.72048681]
Calculated thresholds for skin tones: [3.8908825485804783, 16.194542136009495, 25.
375226818525324, 32.53162493682256, 38.66249502605597]
```



We apply the found thresholds to our ITA values to label each image with its corresponding skin tone. In particular, the six possible values for the skin tone are:

1. **very light** (type 1 on the Fitzpatrick scale)
2. **light** (type 2)
3. **intermediate** (type 3)
4. **tan** (type 4)
5. **brown** (type 5)
6. **dark** (type 6)

Finally, we visualize the distribution of the six different skin tones across the dataset.



We find the classification results into csv files to retrieve the labels later during step 3.

5. Show some results of the classification

As an example, we visualize the results of the skin tone classification for the **esantema morbilliforme**. We can clearly observe that the skin tone is misclassified when the image is subject to poor illumination, i.e. less illuminated images are wrongly classified as darker skin tones: the ITA computation is not so robust when it comes to poor illumination.

Third step: training a CNN and assessing its performance across the different skin

tone groups

1. Initial setup: importing (and installing) the necessary libraries

```
cpu
```

2. Building the dataset class

```
329660
```

3. Building the network

The chosen model is a Convolutional Neural Network (CNN) with **five convolutional layers**, where each layer is followed by a MaxPooling layer. Each convolutional block doubles the channels and decreases the spatial dimension to capture high level features.

4. Training the network

4.1 Training function

The training function uses an early stopper to prevent the waste of computational resources. At each epoch, the model is saved only if the mean F1 score of that epoch is higher than the previous one.

4.2 Hyperparameter tuning

The chosen optimizer and loss are SGD with momentum (μ = 0.9) and Binary Cross Entropy. The two hyperparameters which need to be optimized are the **learning rate** and the **batch size**. We try the following combinations of both:

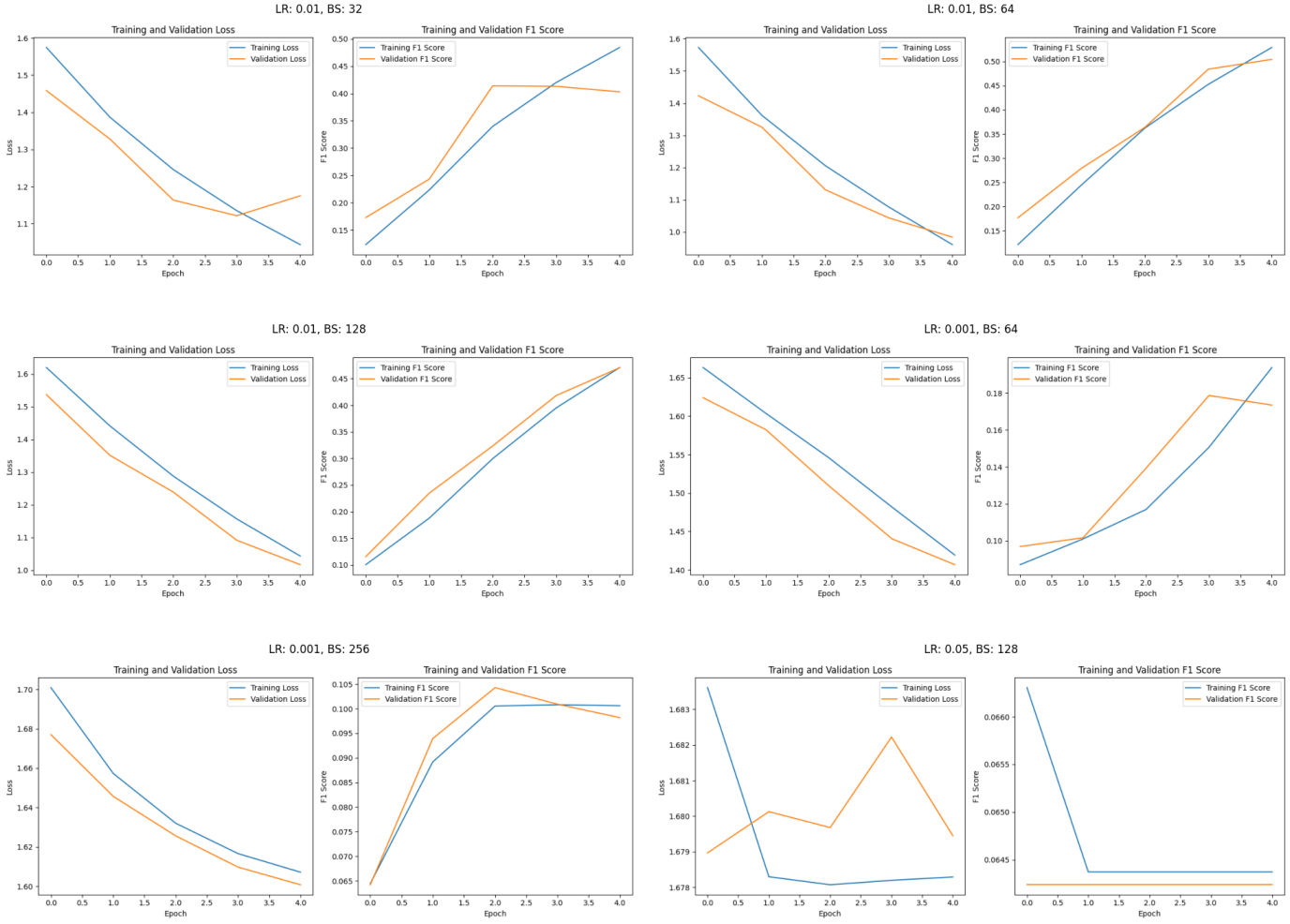
1. learning rate = 0.01, batch size = 32
2. learning rate = 0.01, batch size = 64
3. learning rate = 0.01, batch size = 128
4. learning rate = 0.001, batch size = 64

5. learning rate = 0.001, batch size = 256
6. learning rate = 0.05, batch size = 128

We train the model for **five** epochs for each of these combinations, and choose the most promising combination to train the final model.

```
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 32
AND LEARNING RATE = 0.01**********************************
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 64
AND LEARNING RATE = 0.01**********************************
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 12
8 AND LEARNING RATE = 0.01**********************************
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 64
AND LEARNING RATE = 0.001**********************************
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 25
6 AND LEARNING RATE = 0.001**********************************
**********************************TRAINING THE MODEL FOR 5 EPOCHS, BATCH SIZE = 12
8 AND LEARNING RATE = 0.005**********************************
Net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  (fc): Linear(in_features=32768, out_features=9, bias=True)
)
```

4.2.1 Results



From the plots, we can certainly rule out the batch size value of 0.05 (last plot), because it leads to divergence of the loss. Furthermore, an initial learning rate of 0.01 assures faster convergence with respect to 0.001. Finally, a batch size equal to 128 seems to assure the best F1 and loss trend (the smaller batch sizes, i.e. 32 and 64, seem to be prone to overfitting). Thus, the optimal combination results to be **batch size = 128** and **learning rate = 0.01**.

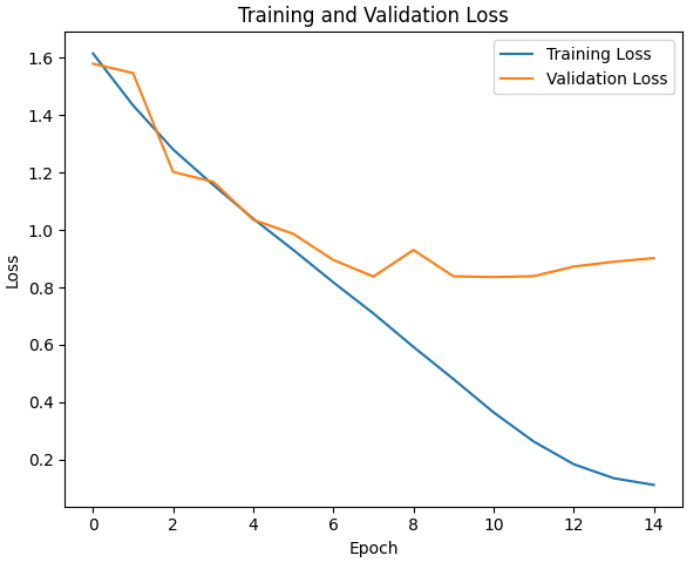
4.3 Final training

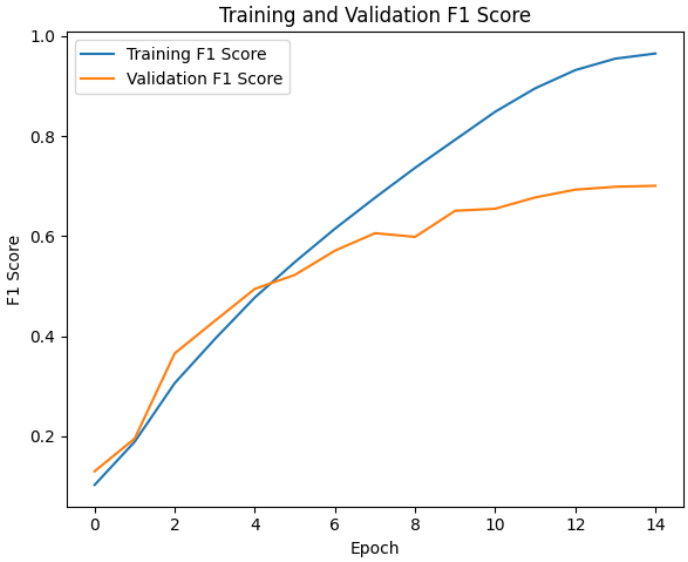
For the final training, the network is trained for 15 epochs using a **Cosine Annealing Learning Rate Schedule**.

```
**********************************TRAINING THE MODEL FOR 15 EPOCHS, BATCH SIZE = 1
28 AND INITIAL LEARNING RATE = 0.01**********************************
Net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  (fc): Linear(in_features=32768, out_features=9, bias=True)
)
```

```
0it [00:00, ?it/s]
[Epoch 1, Batch     1 / 1546] Training Loss: 2.1962897777557373
[Epoch 1, Batch  1001 / 1546] Training Loss: 1.648
0it [00:00, ?it/s]
[Epoch 1, Validation Loss: 1.579]
UPDATING LEARNING RATE TO:  0.009890738003669028
0it [00:00, ?it/s]
[Epoch 2, Batch     1 / 1546] Training Loss: 1.5619908571243286
[Epoch 2, Batch  1001 / 1546] Training Loss: 1.470
0it [00:00, ?it/s]
[Epoch 2, Validation Loss: 1.547]
UPDATING LEARNING RATE TO:  0.009567727288213004
0it [00:00, ?it/s]
[Epoch 3, Batch     1 / 1546] Training Loss: 1.5135180950164795
[Epoch 3, Batch  1001 / 1546] Training Loss: 1.301
0it [00:00, ?it/s]
[Epoch 3, Validation Loss: 1.202]
UPDATING LEARNING RATE TO:  0.009045084971874737
0it [00:00, ?it/s]
[Epoch 4, Batch     1 / 1546] Training Loss: 1.1489756107330322
[Epoch 4, Batch  1001 / 1546] Training Loss: 1.170
0it [00:00, ?it/s]
[Epoch 4, Validation Loss: 1.168]
UPDATING LEARNING RATE TO:  0.008345653031794291
0it [00:00, ?it/s]
[Epoch 5, Batch     1 / 1546] Training Loss: 1.0804492235183716
[Epoch 5, Batch  1001 / 1546] Training Loss: 1.051
0it [00:00, ?it/s]
[Epoch 5, Validation Loss: 1.036]
UPDATING LEARNING RATE TO:  0.0075
0it [00:00, ?it/s]
[Epoch 6, Batch     1 / 1546] Training Loss: 1.0715714693069458
[Epoch 6, Batch  1001 / 1546] Training Loss: 0.942
0it [00:00, ?it/s]
[Epoch 6, Validation Loss: 0.987]
UPDATING LEARNING RATE TO:  0.006545084971874737
0it [00:00, ?it/s]
[Epoch 7, Batch     1 / 1546] Training Loss: 0.838587760925293
[Epoch 7, Batch  1001 / 1546] Training Loss: 0.821
0it [00:00, ?it/s]
[Epoch 7, Validation Loss: 0.896]
UPDATING LEARNING RATE TO:  0.005522642316338268
0it [00:00, ?it/s]
[Epoch 8, Batch     1 / 1546] Training Loss: 0.9043705463409424
[Epoch 8, Batch  1001 / 1546] Training Loss: 0.711
0it [00:00, ?it/s]
[Epoch 8, Validation Loss: 0.838]
UPDATING LEARNING RATE TO:  0.004477357683661734
0it [00:00, ?it/s]
[Epoch 9, Batch     1 / 1546] Training Loss: 0.6415271759033203
[Epoch 9, Batch  1001 / 1546] Training Loss: 0.591
0it [00:00, ?it/s]
[Epoch 9, Validation Loss: 0.931]
UPDATING LEARNING RATE TO:  0.003454915028125264
0it [00:00, ?it/s]
[Epoch 10, Batch     1 / 1546] Training Loss: 0.5375970005989075
[Epoch 10, Batch  1001 / 1546] Training Loss: 0.478
```

```
0it [00:00, ?it/s]
[Epoch 10, Validation Loss: 0.839]
UPDATING LEARNING RATE TO:  0.0025000000000000014
0it [00:00, ?it/s]
[Epoch 11, Batch     1 / 1546] Training Loss: 0.32278046011924744
[Epoch 11, Batch  1001 / 1546] Training Loss: 0.360
0it [00:00, ?it/s]
[Epoch 11, Validation Loss: 0.836]
UPDATING LEARNING RATE TO:  0.0016543469682057106
0it [00:00, ?it/s]
[Epoch 12, Batch     1 / 1546] Training Loss: 0.2870432436466217
[Epoch 12, Batch  1001 / 1546] Training Loss: 0.261
0it [00:00, ?it/s]
[Epoch 12, Validation Loss: 0.839]
UPDATING LEARNING RATE TO:  0.0009549150281252633
0it [00:00, ?it/s]
[Epoch 13, Batch     1 / 1546] Training Loss: 0.171899676322937
[Epoch 13, Batch  1001 / 1546] Training Loss: 0.184
0it [00:00, ?it/s]
[Epoch 13, Validation Loss: 0.873]
UPDATING LEARNING RATE TO:  0.0004322727117869951
0it [00:00, ?it/s]
[Epoch 14, Batch     1 / 1546] Training Loss: 0.08497598022222519
[Epoch 14, Batch  1001 / 1546] Training Loss: 0.136
0it [00:00, ?it/s]
[Epoch 14, Validation Loss: 0.890]
UPDATING LEARNING RATE TO:  0.00010926199633097157
0it [00:00, ?it/s]
[Epoch 15, Batch     1 / 1546] Training Loss: 0.10084985941648483
[Epoch 15, Batch  1001 / 1546] Training Loss: 0.113
0it [00:00, ?it/s]
[Epoch 15, Validation Loss: 0.902]
train_losses: [1.6149594355061323, 1.4341852250876432, 1.2806274321841082, 1.15719
56246050332, 1.0395397815328058, 0.9307365683889327, 0.8176825840368308, 0.7094625
290507333, 0.5927626850495518, 0.48051011426444196, 0.3647436335644315, 0.26324533
78150179, 0.1841648974796856, 0.13533238456000177, 0.11205320994949201]
validation_losses: [1.579335401917613, 1.546839066485102, 1.2024625953315764, 1.16
81634103373963, 1.0358979415523915, 0.986555875964867, 0.8956890071547309, 0.83786
48950841076, 0.9305247420719428, 0.838774155623229, 0.8362785159617432, 0.83914370
30346819, 0.8728210104181785, 0.889505781926388, 0.9021401644446129]
train f1 scores:  [0.10242586146798777, 0.18872084069467504, 0.30594954052694245,
0.39391703480879475, 0.4772504083576396, 0.5482022321366358, 0.6147623741948082,
0.6767996841143518, 0.736452826894486, 0.7925962189484639, 0.8486725459327662, 0.8
956222336148725, 0.9319485810083754, 0.9549156591442548, 0.9651504500188643]
validation f1 scores:  [0.12960860423334403, 0.19507021764122612, 0.36553236599257
916, 0.43039080444086825, 0.49449405099271726, 0.5224100421246277, 0.5708940824440
97, 0.6058216313811499, 0.5984198533005355, 0.6505916701155022, 0.654715119909390
1, 0.6773188911597855, 0.6928899917073346, 0.6986738301994938, 0.7006033855755979]
Finished Training
```





5. Evaluating the network

This step involves the evaluation of the network's performance according to 'traditional' metrics (accuracy, F1 scores) and fairness metrics, which are specifically designed to assess the presence of bias.

First, we retrieve the predictions of the network on the test set:

```
  0%|          | 0/516 [00:00<?, ?it/s]
```

5.1 Accuracy and F1 score

We compute the overall accuracy and F1 score:

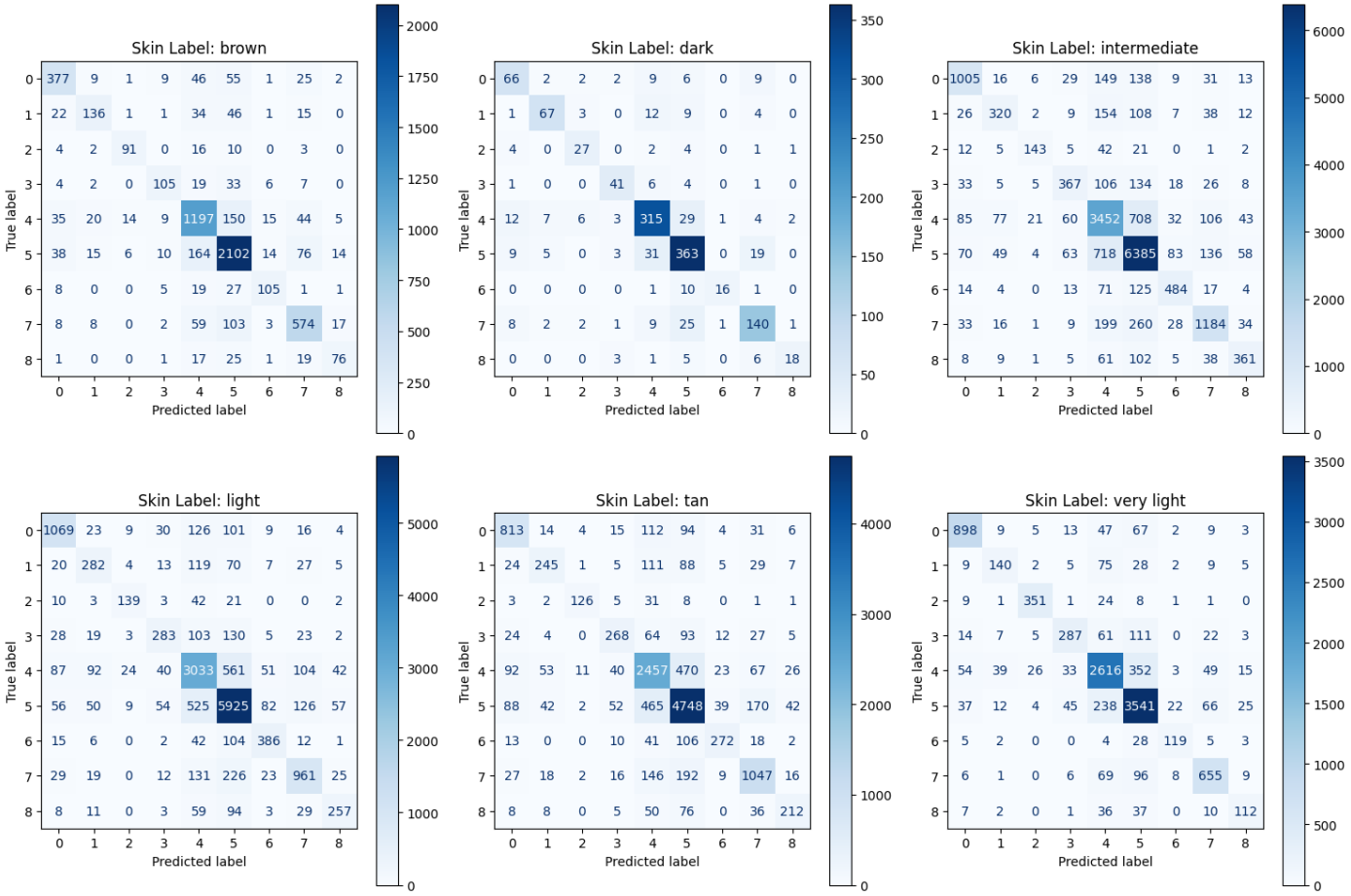
```
Overall accuracy: 77.0%
Overall F1 score: 0.77
```

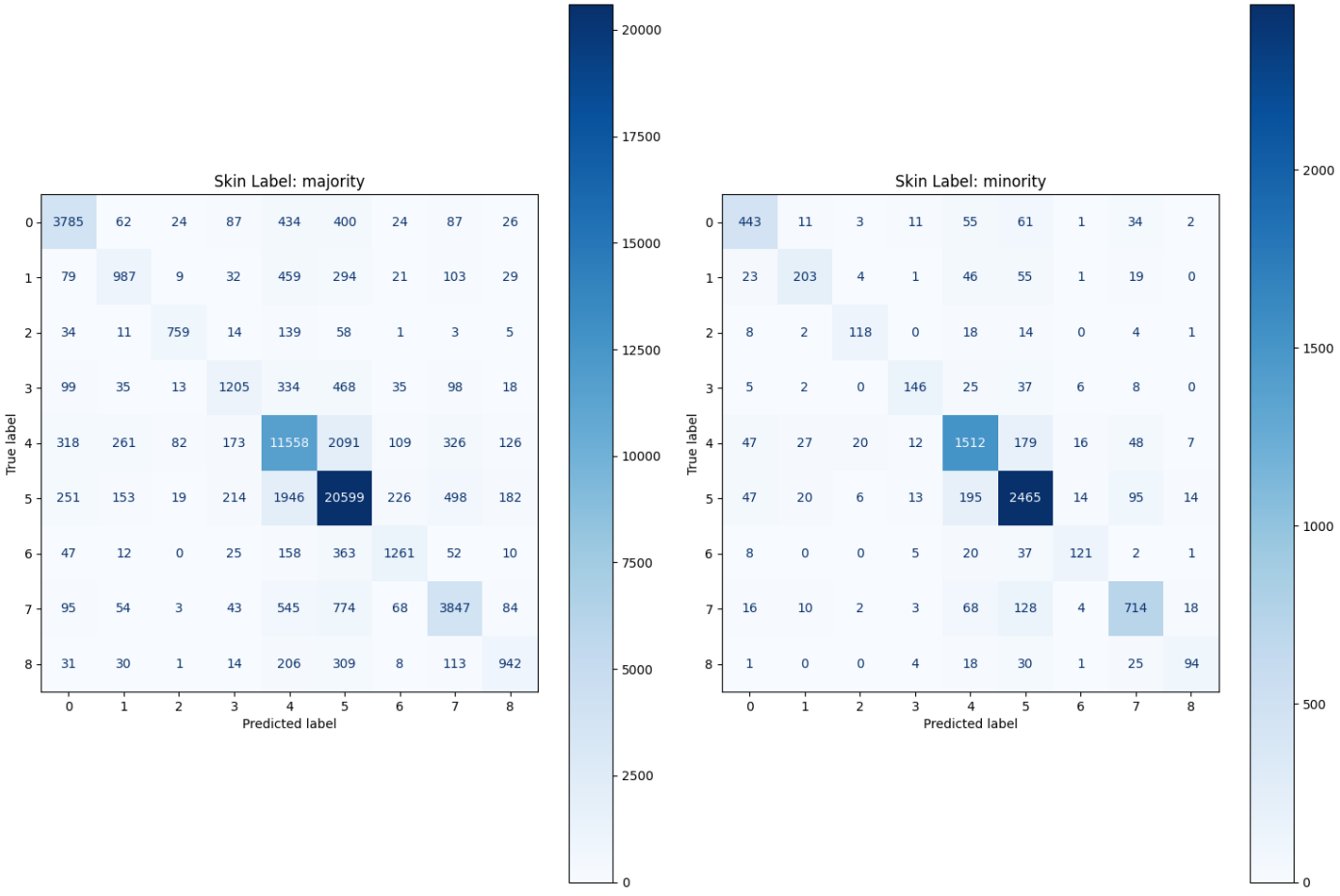
We evaluate the accuracy of the network for every skin tone.

```
Accuracy for skin tone light: 76.5%
Accuracy for skin tone very light: 82.0%
Accuracy for skin tone tan: 75.5%
Accuracy for skin tone intermediate: 75.1%
Accuracy for skin tone brown: 78.2%
Accuracy for skin tone dark: 78.4%
```

We also evaluate the F1 score of the network for every skin tone.

```
F1 score for skin tone very light: 0.82
F1 score for skin tone brown: 0.78
F1 score for skin tone light: 0.76
F1 score for skin tone intermediate: 0.75
F1 score for skin tone dark: 0.78
F1 score for skin tone tan: 0.75
```





5.2 Fairness metrics

5.2.1 Disparate Impact

In the binary case, **disparate impact** is defined as

$$\frac{Pr(\hat{Y} = 1|X \in minority\_group)}{Pr(\hat{Y} = 1|X \in majority\_group)}$$

i.e. the ratio between the selection rate for the minority and majority group. In our case, we have a multiclass task at hand. We handle it by computing the Disparate Impact for each disease, considering $\hat{Y} = 1$ if the single disease is detected, $\hat{Y} = 0$ otherwise. Plus, we aggregate the skin tones considering the *brown* and *dark* skin tones as the **minority group** and the remaining four skin tones as the **majority group**. The disparate impact is computed using the implementation provided by the library *holisticai*.

```
Disparate impact for disease esantema-iatrogeno-farmaco-indotta: 0.99

Disparate impact for disease esantema-maculo-papuloso: 1.35

Disparate impact for disease esantema-morbilliforme: 1.32

Disparate impact for disease esantema-polimorfo-like: 0.85

Disparate impact for disease esantema-virale: 0.98

Disparate impact for disease orticaria: 0.93

Disparate impact for disease pediculosi: 0.74

Disparate impact for disease scabbia: 1.46

Disparate impact for disease varicella: 0.76
```

5.2.2 Equalized Odds Ratio

A classifier satisfies Equalized Odds under a distribution over (X, A, Y) (where A indicates the sensitive feature) if its prediction $\hat{Y}$ is conditionally independent of the sensitive feature A given the label Y . This is equivalent to $E(\hat{Y}|A = a, Y = y) = E(\hat{Y}|Y = y)\ \forall\ a, y$. Equalized odds requires that the true positive rate, $Pr(\hat{Y} = 1|Y = 1)$, and the false positive rate, $Pr(\hat{Y} = 1|Y = 0)$, are equal across groups. In our case, Equalized Odds Ratio (EOR) was computed for every disease using the implementation of the library **fairlearn**, in which EOR is defined as 'the smaller of two metrics: *true positive rate ratio* and *false positive rate ratio*. The former is the ratio between the smallest and largest of $Pr(\hat{Y} = 1|A = a, Y = 1)$, across all values of the sensitive feature(s). The latter is defined similarly, but for $Pr(\hat{Y} = 1|A = a, Y = 0)$. The equalized odds ratio of 1 means that all groups have the same true positive, true negative, false positive, and false negative rates'

[https://fairlearn.org/main/api_reference/generated/fairlearn.metrics.equalized_odds_rati
Even in this case, the skin tones were aggregated into the same two groups used to compute the Disparate Impact Ratio.

```
Equalized Odds Ratio for esantema-iatrogeno-farmaco-indotta: 0.78

Equalized Odds Ratio for esantema-maculo-papuloso: 0.85

Equalized Odds Ratio for esantema-morbilliforme: 0.55

Equalized Odds Ratio for esantema-polimorfo-like: 0.63

Equalized Odds Ratio for esantema-virale: 0.82

Equalized Odds Ratio for orticaria: 0.86

Equalized Odds Ratio for pediculosi: 0.68

Equalized Odds Ratio for scabbia: 0.67

Equalized Odds Ratio for varicella: 0.70
```

5.2.3 Predictive Rate Ratio

The **Predictive Rate Parity** is achieved if the **Positive Predictive Value (PPV)**, also named as **precision**, is the same across all groups. The formula for the precision is

$$PPV = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

The **Predictive Rate Ratio** compares the precision between the two groups. It is calculated by taking the ratio of the PPV of one group to the PPV of another group:

$$Predictive\ Rate\ Ratio = \frac{PPV_{minority\_group}}{PPV_{majority\_group}}$$

In our implementation, we compute this value for each disease, with the usual division of the skin tones into the minority group and majority group.

We can retrieve the Predictive Rate Ratio value by considering the precision matrix relative to each disease for the minority and majority group. The function for the precision matrix is taken from the library **holisticai**. Observing the matrix, we simply compute the ratio between the precision of the minority group and of the majority group.

	0	1	2	3	4	5	6	7	
majority	0.767904	0.490313	0.741211	0.522777	0.768280	0.855156	0.654046	0.697805	0.
minority	0.713366	0.576705	0.715152	0.637555	0.809422	0.859184	0.623711	0.741433	0.

```
Predictive Rate Ratio for disease esantema-iatrogeno-farmaco-indotta: 0.93

Predictive Rate Ratio for disease esantema-maculo-papuloso: 1.18

Predictive Rate Ratio for disease esantema-morbilliforme: 0.96

Predictive Rate Ratio for disease esantema-polimorfo-like: 1.22

Predictive Rate Ratio for disease esantema-virale: 1.05

Predictive Rate Ratio for disease orticaria: 1.00

Predictive Rate Ratio for disease pediculosi: 0.95

Predictive Rate Ratio for disease scabbia: 1.06

Predictive Rate Ratio for disease varicella: 0.95
```