Technical test for Data Scientist role at ORO Health

Dataset: 102 image data with three classes, acne (40 images), Herpes Simplex (16 images) and Lichen Planus (46 images)

Task: Multiclass classification

1. Data Preprocessing:

I first unzipped the data and then listed the filenames in three lists to be used for data loading in PyTorch. The following functions have been used for data preprocessing:

Find_image_ratios: This function opens the image files for different classes and get the width, height and height to width ration in different lists. After reading all images, I understood that:

1- The average H/W ration for images is 1.5.
2- There are 31 vertical images among data.

Image_transformation: This function does two things:

1- Rotates the vertical images 90 degrees to become horizontal (since 70% of images are horizontal).
2- Resizes the images to 120×180 tensors.
3- Concatenates all images to one tensor and generates the labels list.

The output is a tensor containing all the images with size $102 \times 3 \times 120 \times 180$ and label tensor with size 102.

Data_augmentation: Since the dataset is very small for training a model and getting the appropriate predictions, I tried to generate augmented images from original ones by using specific transformers from PyTorch. The reason for data augmentation was to first, have a larger train dataset and second, training model for different possible variations in images taken by patients. The list of augmentations is as follow:

Augmentation Type	**Purpose**
Random Rotation	Possible rotations in the pictures taken by patients.
Gaussian Blur	Possible blur or low-resolution images.
Color Jitter	Images with different brightness, contrast, saturation, and hue.
Random Crop	Images of various distances from the skin.
Random Auto Contrast	Images with different contrasts.
Random Equalize	Different equalize images.

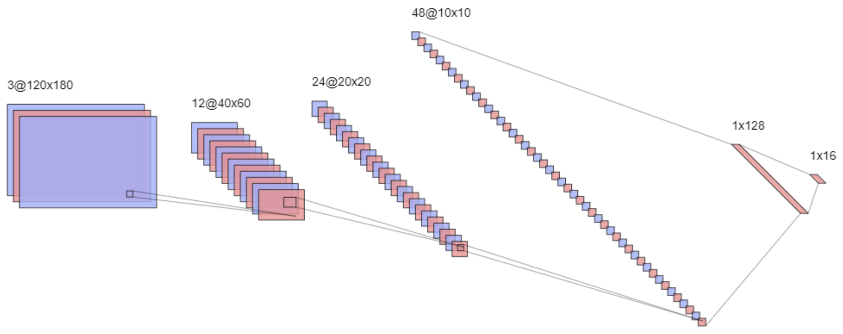
Image_normalization: In this function the values are divided by 255 and normalized to be zero centered, with a mean equal to 0 and standard deviation of 1.

train_test_split: Since the data is imbalanced, in order to have data from all classes in both train and test sets, I extracted the indices of each class from main data, then considered 90% of it as train and 10% of it as test data. At last, I concatenated three train and test splits.

During the data preparation, after transforming images, I first augmented the images then split them to train and test. However, I then realized this is not a correct approach and by doing so, we will see transformed images of test set while training. So, I first split the data and then apply the augmentation and at last normalization.

2. Training:

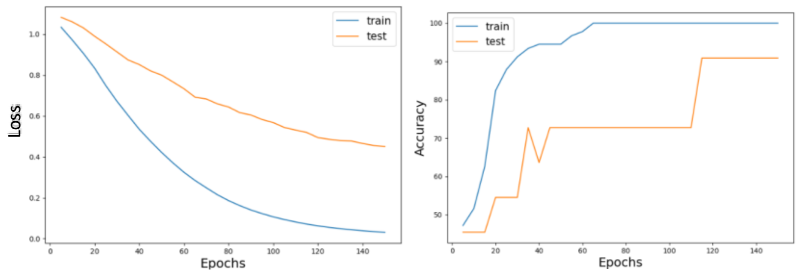
I used a CNN structure to do the classification, as CNNs are very powerful in image classification tasks. There are three 2D convolution layers and two fully connected layer. A softmax layer is applied at the end. I also added batch normalization to the second and third after second and third convolution which improved the performance. At all stages I used ReLu activation function since the convergence of ReLu is faster than tanh/sigmoid functions and it does not have some problems from tanh such as Vanishing Gradient Problem. The kernel size, stride and padding are chosen to gradually decrease the image size. The batch size is set to 6 for all trainings. The used architecture for CNN model is as below:



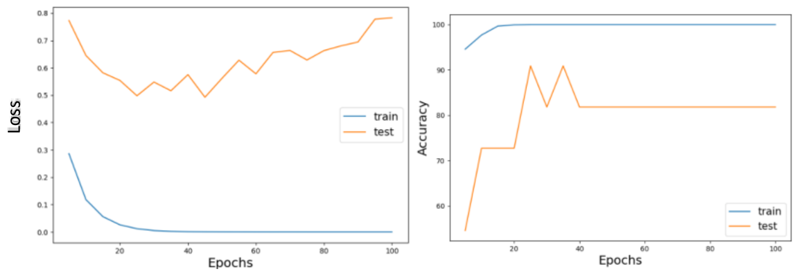
Loss Function: I used Cross Entropy loss function from Pytorch library, since the task was multiclass classification.

Optimizer: Adam optimizer is used. Generally, Adam is faster to converge than SGD; however, SGD generalize better. Also, SGD is shown to be more locally unstable. Hence, I chose Adam with lr=0.00001.

Loss and accuracy for original dataset:



Loss and accuracy for augmented dataset:

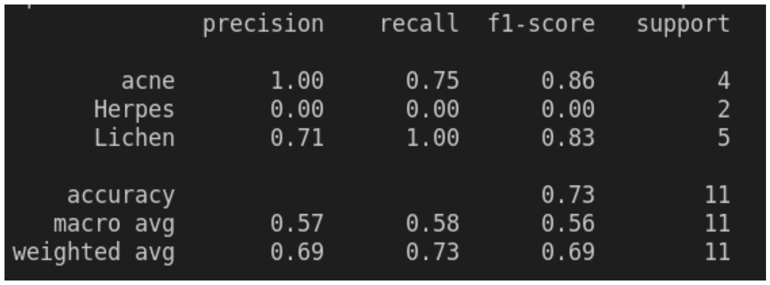


Sufficient number of epochs are chosen based on the above figures as:

Original data: 120 epochs, Augmented data: 35 epochs

3. Classification report

Classification report for original data:

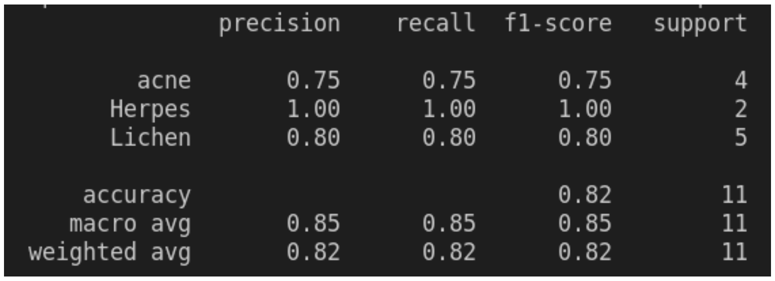


```
              precision    recall  f1-score   support

        acne       1.00      0.75      0.86         4
      Herpes       0.00      0.00      0.00         2
      Lichen       0.71      1.00      0.83         5

    accuracy                           0.73        11
   macro avg       0.57      0.58      0.56        11
weighted avg       0.69      0.73      0.69        11
```

Classification report for augmented data:



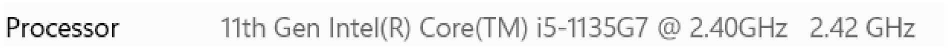
```
              precision    recall  f1-score   support

        acne       0.75      0.75      0.75         4
      Herpes       1.00      1.00      1.00         2
      Lichen       0.80      0.80      0.80         5

    accuracy                           0.82        11
   macro avg       0.85      0.85      0.85        11
weighted avg       0.82      0.82      0.82        11
```

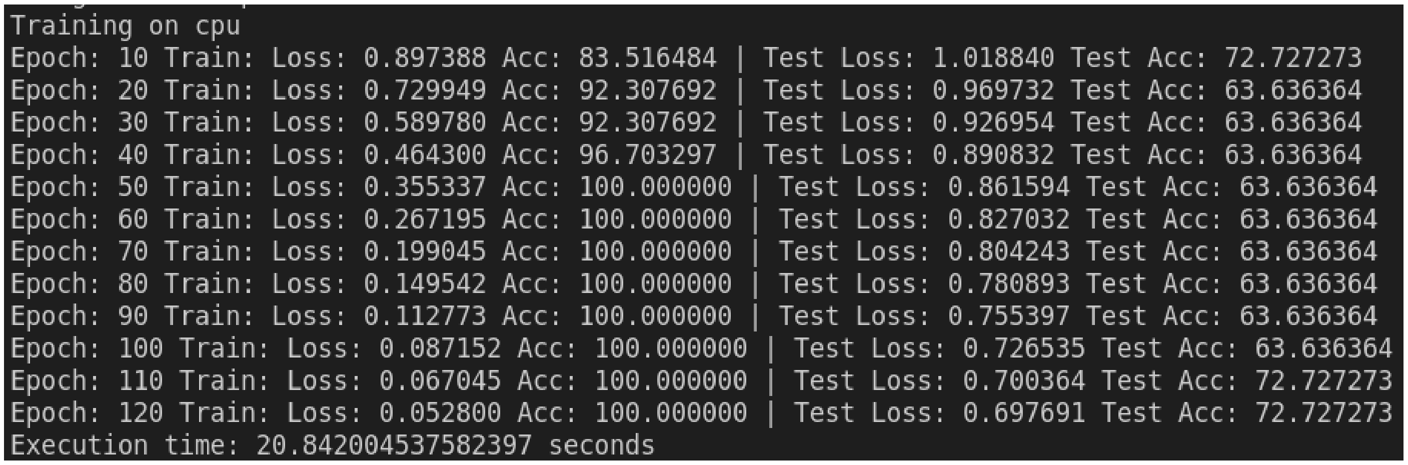
4. General Information

For this test I used VS Code, and all the codes are run on my laptop. I used CPU to run the code with below specification:



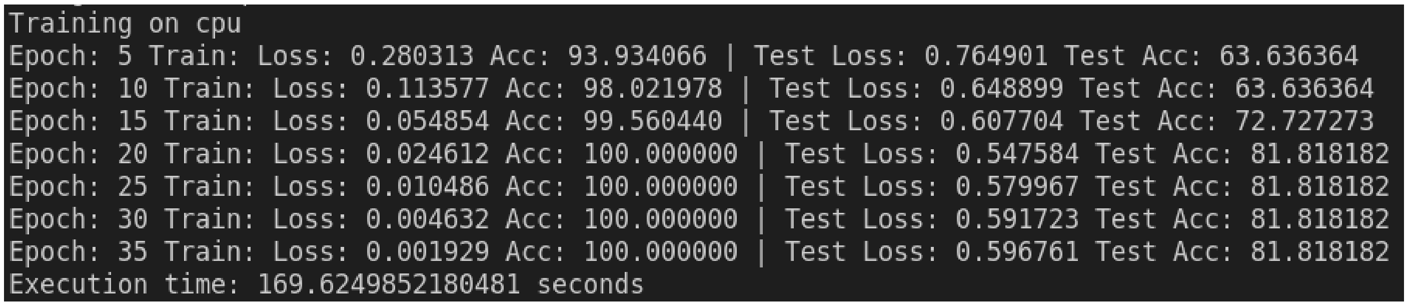
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz

The runtime for the original dataset is as follow:



```
Training on cpu
Epoch: 10 Train: Loss: 0.897388 Acc: 83.516484 | Test Loss: 1.018840 Test Acc: 72.727273
Epoch: 20 Train: Loss: 0.729949 Acc: 92.307692 | Test Loss: 0.969732 Test Acc: 63.636364
Epoch: 30 Train: Loss: 0.589780 Acc: 92.307692 | Test Loss: 0.926954 Test Acc: 63.636364
Epoch: 40 Train: Loss: 0.464300 Acc: 96.703297 | Test Loss: 0.890832 Test Acc: 63.636364
Epoch: 50 Train: Loss: 0.355337 Acc: 100.000000 | Test Loss: 0.861594 Test Acc: 63.636364
Epoch: 60 Train: Loss: 0.267195 Acc: 100.000000 | Test Loss: 0.827032 Test Acc: 63.636364
Epoch: 70 Train: Loss: 0.199045 Acc: 100.000000 | Test Loss: 0.804243 Test Acc: 63.636364
Epoch: 80 Train: Loss: 0.149542 Acc: 100.000000 | Test Loss: 0.780893 Test Acc: 63.636364
Epoch: 90 Train: Loss: 0.112773 Acc: 100.000000 | Test Loss: 0.755397 Test Acc: 63.636364
Epoch: 100 Train: Loss: 0.087152 Acc: 100.000000 | Test Loss: 0.726535 Test Acc: 63.636364
Epoch: 110 Train: Loss: 0.067045 Acc: 100.000000 | Test Loss: 0.700364 Test Acc: 72.727273
Epoch: 120 Train: Loss: 0.052800 Acc: 100.000000 | Test Loss: 0.697691 Test Acc: 72.727273
Execution time: 20.842004537582397 seconds
```

The runtime for augmented data is as follows:



```
Training on cpu
Epoch: 5 Train: Loss: 0.280313 Acc: 93.934066 | Test Loss: 0.764901 Test Acc: 63.636364
Epoch: 10 Train: Loss: 0.113577 Acc: 98.021978 | Test Loss: 0.648899 Test Acc: 63.636364
Epoch: 15 Train: Loss: 0.054854 Acc: 99.560440 | Test Loss: 0.607704 Test Acc: 72.727273
Epoch: 20 Train: Loss: 0.024612 Acc: 100.000000 | Test Loss: 0.547584 Test Acc: 81.818182
Epoch: 25 Train: Loss: 0.010486 Acc: 100.000000 | Test Loss: 0.579967 Test Acc: 81.818182
Epoch: 30 Train: Loss: 0.004632 Acc: 100.000000 | Test Loss: 0.591723 Test Acc: 81.818182
Epoch: 35 Train: Loss: 0.001929 Acc: 100.000000 | Test Loss: 0.596761 Test Acc: 81.818182
Execution time: 169.6249852180481 seconds
```

5. Possible improvements:

First, the dataset is extremely limited which prevents the model to be trained sufficiently and in a short time overfitting occurs.

If I had more time, I would do the followings:

1- Add generalization methods to the model to improve the performance on test data.
2- Use dimensionality reduction methods to reduce the computational cost and complexity.
3- Tried different architectures for my CNN to find the best one.

If I had more computational capacity, I would create multiple datasets with different image sizes, so my model would have been trained better or I would create larger augmented train dataset.