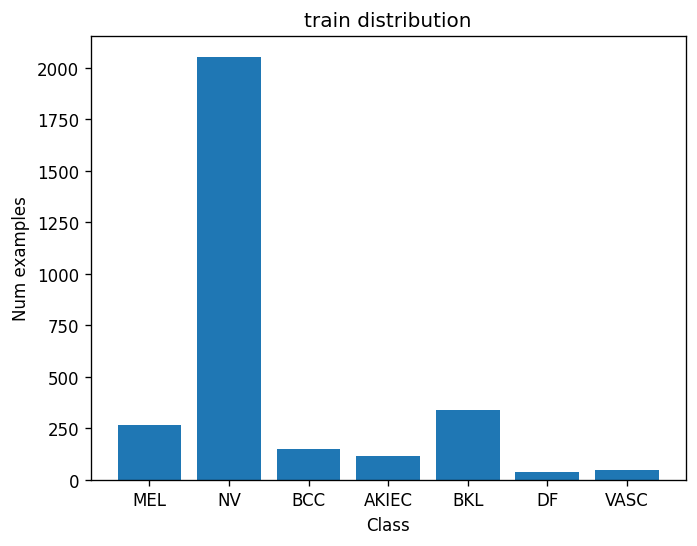
CSE3001 Assignment Report

# Task 1

## Task 1a

### Data issues

In this part, I check for missing values and the distribution of classes in the dataset. There is no missing data in our dataset but the number of samples in the training dataset for each class label is not balanced.

****

*Distribution of Classes in Dataset*

Regarding the distribution graph above, the number of NV samples is much greater than other labels, which acquire 68.3% while the smallest number is 0.17% belonging to the DF and VASC classes.

## Task 1b

### Why not use random\_split?

Using random\_split may lead to a skewed distribution of classes in the training and validation sets, especially in our case, the number of samples in the training dataset for each class label is very imbalanced.

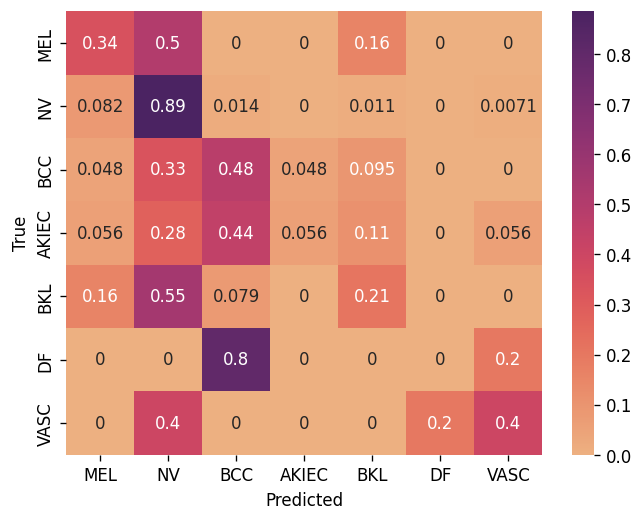
## Task 1c

### Reduce epoch time

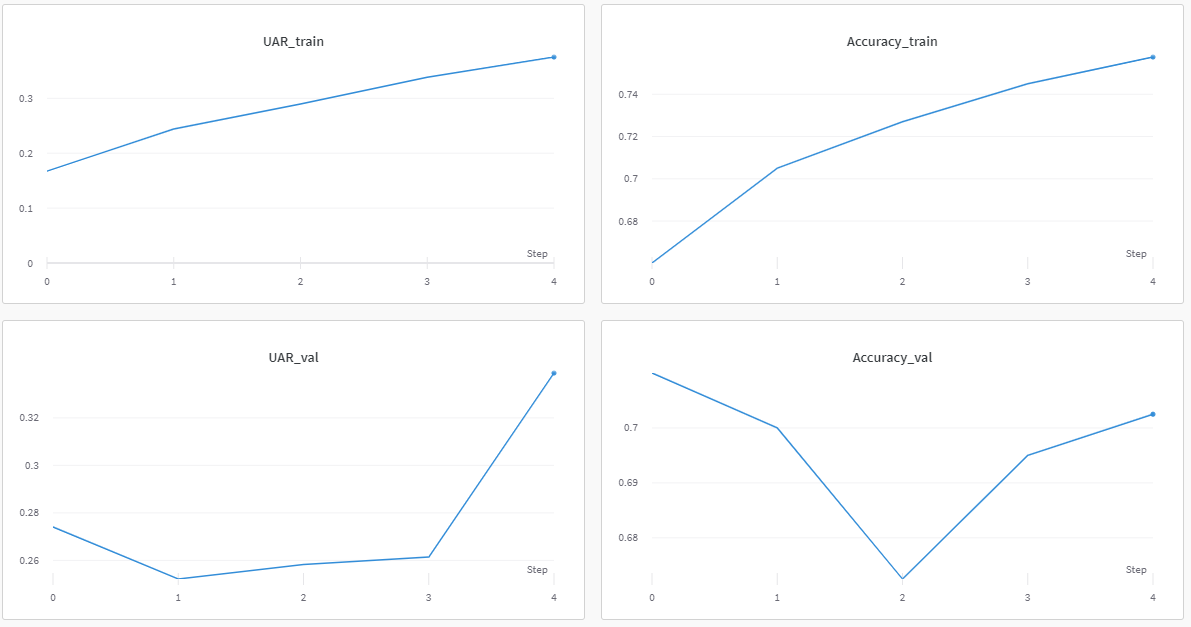
To significantly reduce the time per epoch for debugging purposes while still using real data and real training code, there are 2 considerable approaches:

* Subsampling the Dataset:  
  Instead of using the entire dataset for debugging, create a smaller subset (a random or stratified sample) of data that retains the characteristics and distribution of the original dataset. This allows users to iterate through the code more quickly and catch errors faster.
* Model Simplification:
* Simplify the model architecture during the debugging phase. This can involve reducing the number of layers, nodes, or features within the model. By simplifying the model, users can significantly reduce the computational load, leading to faster training times per epoch.

### Show the confusion matrix and plots of the validation accuracy and UAR in your report, and explain what is going wrong.



*Confusion Matrix*



*plots of the validation accuracy and UAR*

Upon observation of the confusion matrix and the plots of the validation accuracy and Unweighted Average Recall (UAR), I have a few key insights about the performance of our model. The dominance of the NV class within the dataset significantly influences the performance of the model.

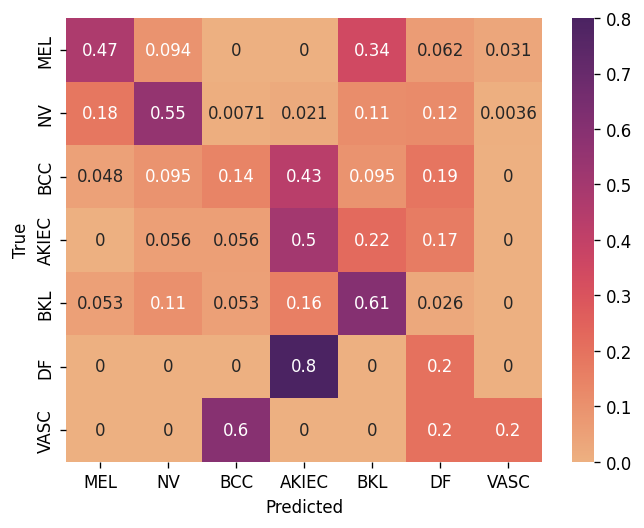
* Confusion Matrix:
  + A considerable number of instances are being misclassified as NV or BCC, indicating a bias towards the majority classes.
  + Limited instances are correctly classified for the minor classes.
* Validation Accuracy:
  + The validation accuracy, at approximately 70%, seems to reflect the large amount of the NV class in the dataset.
* Unweighted Average Recall (UAR):
  + The UAR score of 34% shows that the model is having trouble identifying and categorizing instances from the less common classes.

## Task 1d

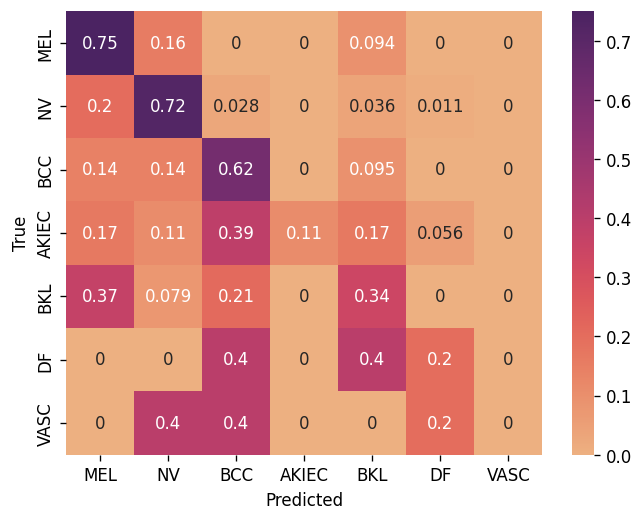
### Account for data issues

To handle the data issues, I used the Weighted loss function, adapting the weight calculation to address the dataset's imbalance.

Initially, using weight = 1 / number of instances of the class didn't return the desired results. Here is the first confusion matrix:

* 

So, I modified the formula to weight = (1/sqrt(num\_of\_sample)), which led to an improved outcome.



The impact of this adjustment on the model's performance was as follows:

* Initially, without any weight adjustments, the model achieved an "accuracy\_val" of 0.70 and a "UAR\_val" of 0.28.
* After applying the weighted loss function with weight (1/num\_of\_samples), the "accuracy\_val" decreased to 0.52, while the "UAR\_val" improved to 0.38.
* Utilizing the normalized weight (1/sqrt(num\_of\_sample)) resulted in an "accuracy\_val" of 0.65 and a "UAR\_val" of 0.36.

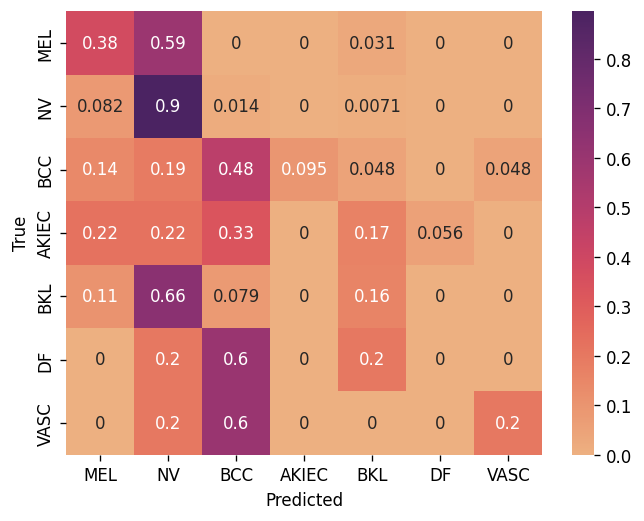
These results demonstrate how the modified weighted loss function helped to mitigate the effects of the imbalanced dataset, leading to an overall improvement in the model's performance.

## Task 1e

### Vertical Flips

Are random vertical flips appropriate for this dataset? Why?

No, random vertical flips may not be suitable for this dataset of skin lesion images. Medical images are usually captured in a specific vertical orientation, and randomly flipping them vertically could spoil the natural appearance of skin lesions, making it difficult for the model to learn common knowledge of the dataset. Moreover, this method is also not helpful in data augmentation. This could potentially lead to a decrease in the model's performance on the dataset.



*Confusion Matrix*

Here is the result while using RandomVerticalFlip:

* accuracy: 70.25%
* UAR: 30.08%

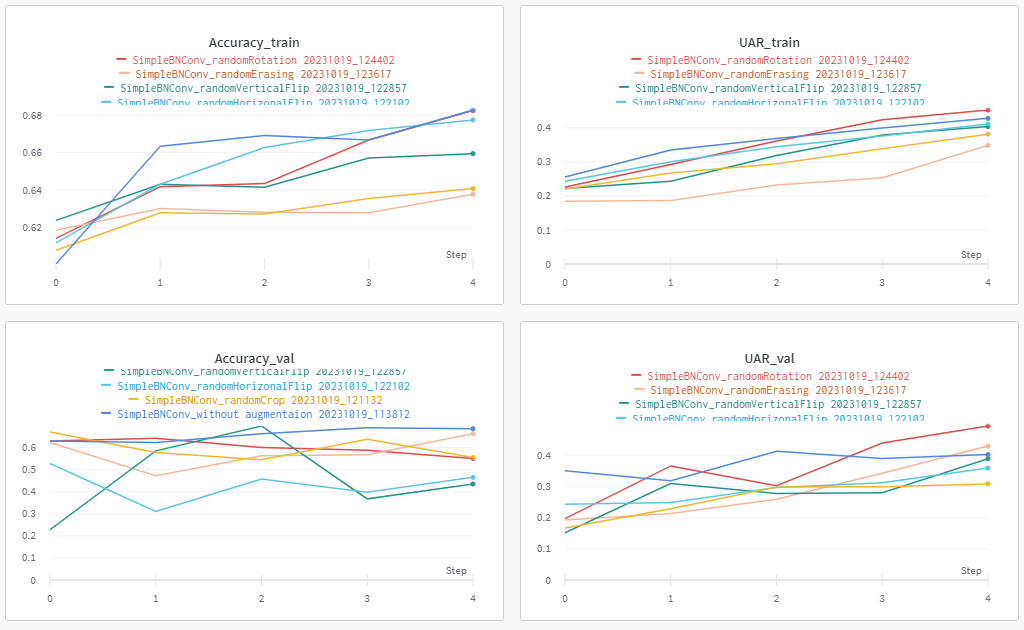
As we see, the result didn’t change much from the experiment without RandomVerticalFlip.

### Effect of Augmentation

What effect did Data Augmentation have on performance? Show a screenshot of the relevant graphs from Weights & Biases for evidence.

I have done some experiments with some data augmentation methods, and here are the methods I used, the colors of the methods below correspond to the color of the their line in the graph from W&B:

* **Without Data Augmentation**
* **transforms.RandomHorizontalFlip(0.5),**
* **transforms.RandomVerticalFlip(0.5),**
* **transforms.RandomErasing(),**
* **transforms.RandomRotation(20),**
* **transforms.RandomCrop((300,400)),**

****

*graphs from Weights & Biases*

Regarding the results and the graphs from Weights & Biases, we can see that RandomRotation and RandomErasing have a positive impact on the performance of the model, while RandomHorizonalFlip, RandomVerticalFlip, and RandomCrop have a negative impact. However, the combination of RandomRotation and RandomErasing methods doesn’t bring a good result.

So in the following experiment, I will use only RandomRotation.

...

### [Challenge] 5 crop augmentation

Apply 5 crop augmentation with crop size 200x300. Make a distinct model which uses 5 crops at once to give a single answer. Include in your report how you did this and report the effect on performance.

Firstly, I create a new transform. Compose in dataset class to perform the FiveCrop method:

self.fivecrop = transforms.Compose(

[

transforms.FiveCrop((200,300)),

transforms.Lambda(lambda crops: torch.cat(crops, dim=0)),

]

After performing FiveCrop, we will get a tuple of 5 images with (200, 300) shapes. So I use transforms.Lambda(lambda crops: torch.cat(crops, dim=0)) to squeeze to a 3D tensor for training.

Secondly, I made a new model based on SimpleBNConv with the first CNN layer is nn.Conv2d(15, 16, kernel\_size=3, padding=1) as the input channels is 3\*5 channel.

## Task 1f

### Experiments

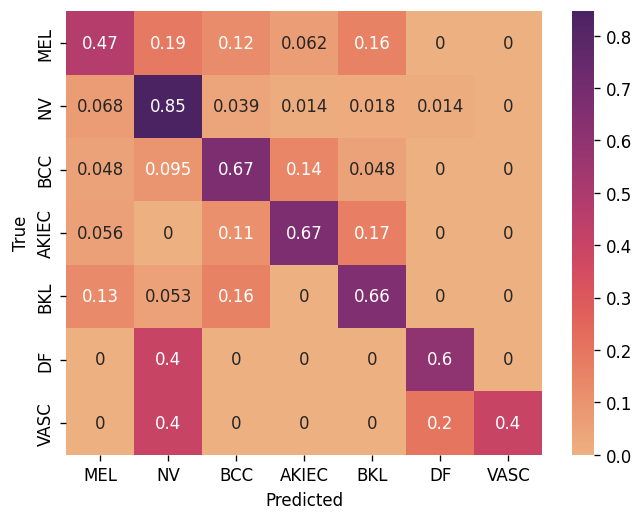
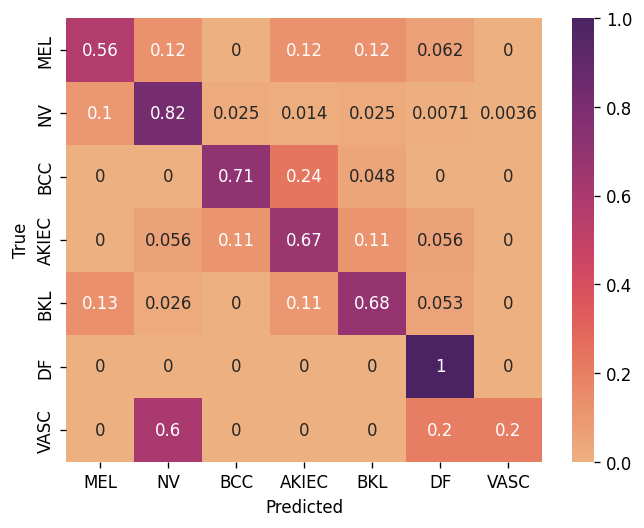
* [Model Training Results in Google Sheet](https://docs.google.com/spreadsheets/d/1k5SZa-CVafDM7zLGQciNjWV2ZRQbRWM7/edit?usp=sharing&ouid=114437346531775137423&rtpof=true&sd=true)
* [Weights and Bias Report](https://api.wandb.ai/links/duc-minh-quang-nguyen/pq3ywedo)
* Write a discussion about the key findings from the experimental results.

After doing some experiments with BATCH\_SIZE and learning rate. I found the best spectrum is BATCH\_SIZE=16 which seems to fasten the training session and also gives a better score with SimpleBNConv and other models. I also examined the optimizer with 2 learning rates 0.001 and 0.003, and with the original learning rate, the optimizer had a better performance. In terms of Optimizer, I did experiments with Stochastic Gradient Descent and Adam, and the results showed that both of them had fairly similar performance. With BATCH\_SIZE = 16, increasing the number of epochs to 10 brings a better result.

BatchNorm and Dropout show amazing results. With the amount of data we have, considering it is not so large if we make our model too complicated, it would be difficult to learn weights. So Dropout did a great job. BatchNorm often plays an important role in the model, it helps the model to normalize before the activation layer, which firmly improves the performance.

Skip Block didn’t perform well in experiments. The reason could be the models we used are simple and the dataset is small. Therefore, Adding skip connections to the models can bring unnecessary complexity, making the model harder to train and optimize, especially in case the dataset is not large enough. Moreover, incorporating skip connections in the model can lead to overfitting, and require more computational resources and longer training times.

In this project, I tried some pre-trained models from Pytorch such as ResNet18, ResNet50\_v2, EfficientNet\_V2\_Small, and DenseNet\_121, and I froze all layers except fully connected layers. The results show that the pre-trained models have better performances in comparison with custom models, although these pre-trained models were trained on different domain datasets. The reason is potentially the architecture of pre-trained models were designed by expert, thus it can capture more knowledge from datasets. The most effective models are Resnet50\_v2 and DenseNet\_121 which scored 62% and 66% UAR respectively.

*Confusion Matrix of Resnet50 Confusion Matrix of DenseNet\_121*

### [Challenge] Batch size

When we halve the batch size, we will double the number of batches required to process the entire dataset.

So, if initially, the number of batches required to process the full dataset in a single epoch was n, then after halving the batch size, it becomes 2\*n. Consequently, the number of times we update the weights per epoch also doubles.

Regarding gradients, reducing the batch size can be beneficial in certain scenarios:

* Noise Reduction: Smaller batch sizes often lead to noisier updates, but this can have a positive effect, helping to prevent overfitting.
* Memory Constraints: Large batch sizes may not fit into the memory of the available hardware, so reducing the batch size can be necessary to allow the model to be trained on the hardware. Fortunately, in our case, BATCH\_SIZE = 64 is still affordable for the hardware of Google Colab.
* Speed of Convergence: Smaller batch sizes can lead to faster convergence in some cases. This is because smaller batches may provide more frequent updates that can help the model escape local minima more efficiently.

However, smaller batch sizes may also slow down the overall training process due to the increased frequency of weight updates, and they can lead to more noisy gradients, which can delay convergence.

Overall, the choice of batch size should be based on practical evaluation and experiments. Different models and datasets could require different batch sizes for optimal performance. In our case, halving BATCH\_SIZE from 64 to 32 improves the results.