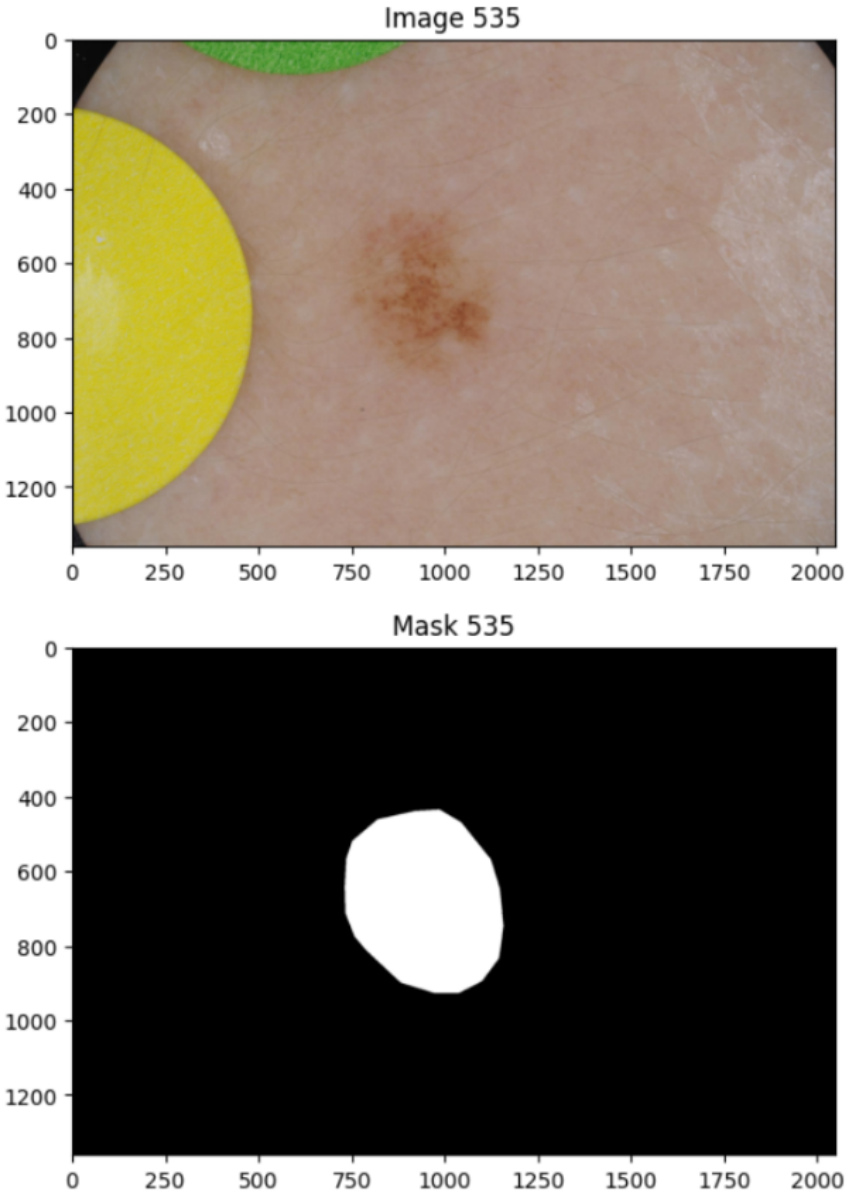
AUTOMATIC DIAGNOSTIC SYSTEM OF SKIN LESIONS FROM DERMOSCOPIC IMAGES



BY:
ALBERTO CASTELLANO MACIAS (100414242)
MOHAMED AFIF CHIFAOUI (100452024)

1. INTRODUCTION

For this assignment, we had two objectives. First of all, we needed to create a custom network from scratch without the use of external packages, in order to obtain the best possible model that adapts to our problem, which is automatically diagnosing skin lesions from dermoscopic images. Secondly, we needed to use an existing pre-trained model and fine-tune it for this same particular problem. The main goal was to obtain the best possible AUC score for both models.

2. DESCRIPTION OF CUSTOM NETWORK

Before starting to build our **custom network** (CNN), we had a look at some of the images in the dataset to try to better understand the problem.

Then, we moved on to the possible **transformations** we could implement on the images to improve their quality. For this, we actually tried many different combinations and even implemented additional ones such as ColorJitter, to adapt the brightness of the images. However, we realized that some transformations actually did not help improving the images and we ended up removing CenterCrop and implementing **RandomCrop** with a size of 256, as it worked better than a size of 224. Also, we lowered the **batch size** of all data loaders to 8 for training and 64 for validation and test, as well as setting the maxSize to 2000 for training our optimal model.

Furthermore, we went on to create our custom network. We will explain every step of our design that led us to an improvement in the average AUC score of our model. The first thing we did was obviously change the **number of classes** in the output to 3, as required.

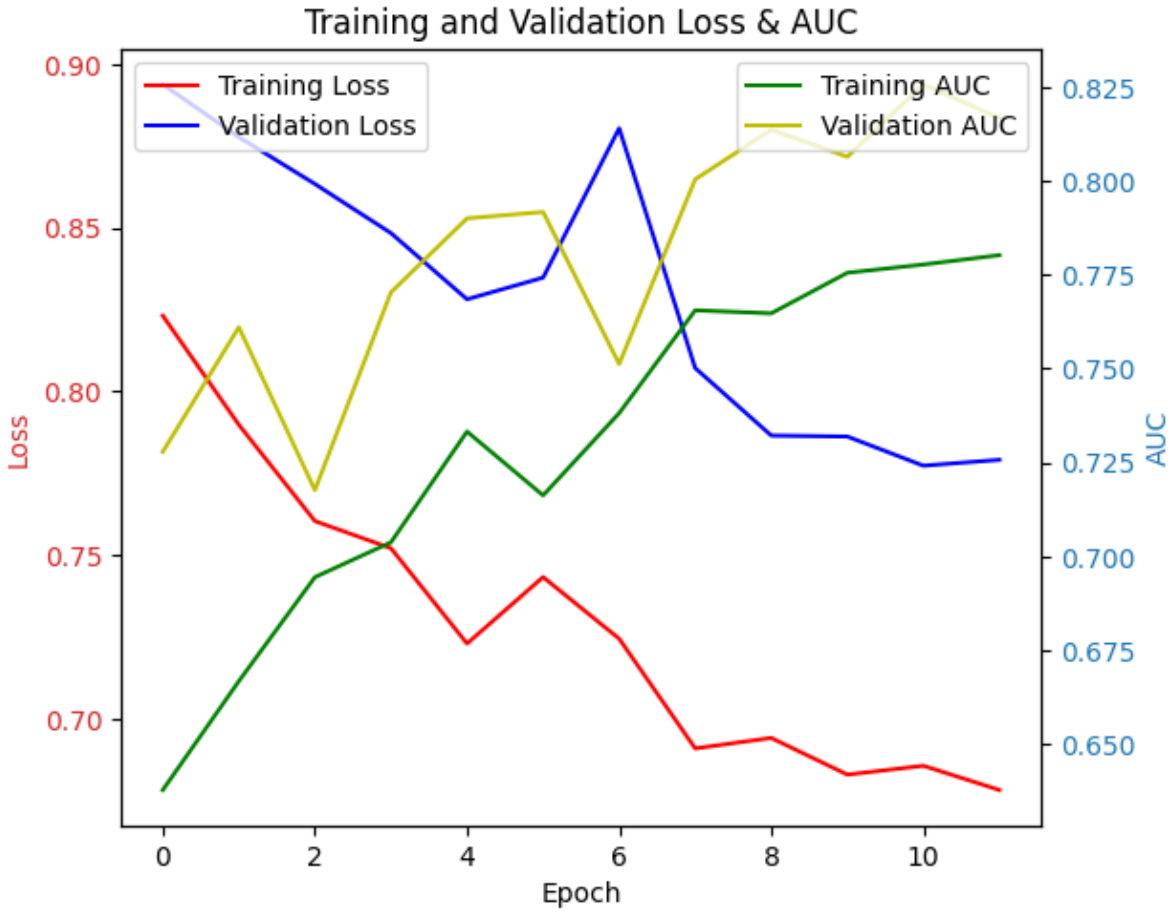
Following that, we realized that we had too many linear or **fully connected layers** as the model was probably overfitting and failing to generalize well, so we reduced them to only 2. We then tried increasing the number of **convolutional layers** to 3 and 4, and realized that 3 layers were ideal for optimal feature extraction.

We also added **batch normalization** to all layers, which significantly improved our score, and then added a **dropout of 0.5** to the linear layers to prevent overfitting. We additionally played with the **maxpooling** and decided it was best to keep it in all convolutional layers as it was.

Moreover, we also played a lot with the number of input and output **channels** for every layer until we found a good combination that optimally fits our problem.

By implementing all of the previously mentioned steps and decisions, we were able to obtain an average **AUC score of 0.8257** during the training of our model for 12 epochs.

As we can see from the plot below, the validation loss is above the training loss with a 0.1 difference, so there might be a bit of overfitting but not much, as also validation loss is usually decreasing (except for epoch 6). Also, the best average AUC score in validation was obtained in epoch 11.



3. DESCRIPTION OF PRE-TRAINED MODEL

For the second part of the assignment, we had the task of fine-tuning an existing pre-trained model, which in this case was changing the last layer of the network to produce an output of 3 classes. Therefore, we had to research for different models that could improve the performance for our problem and we ended up trying the following models:

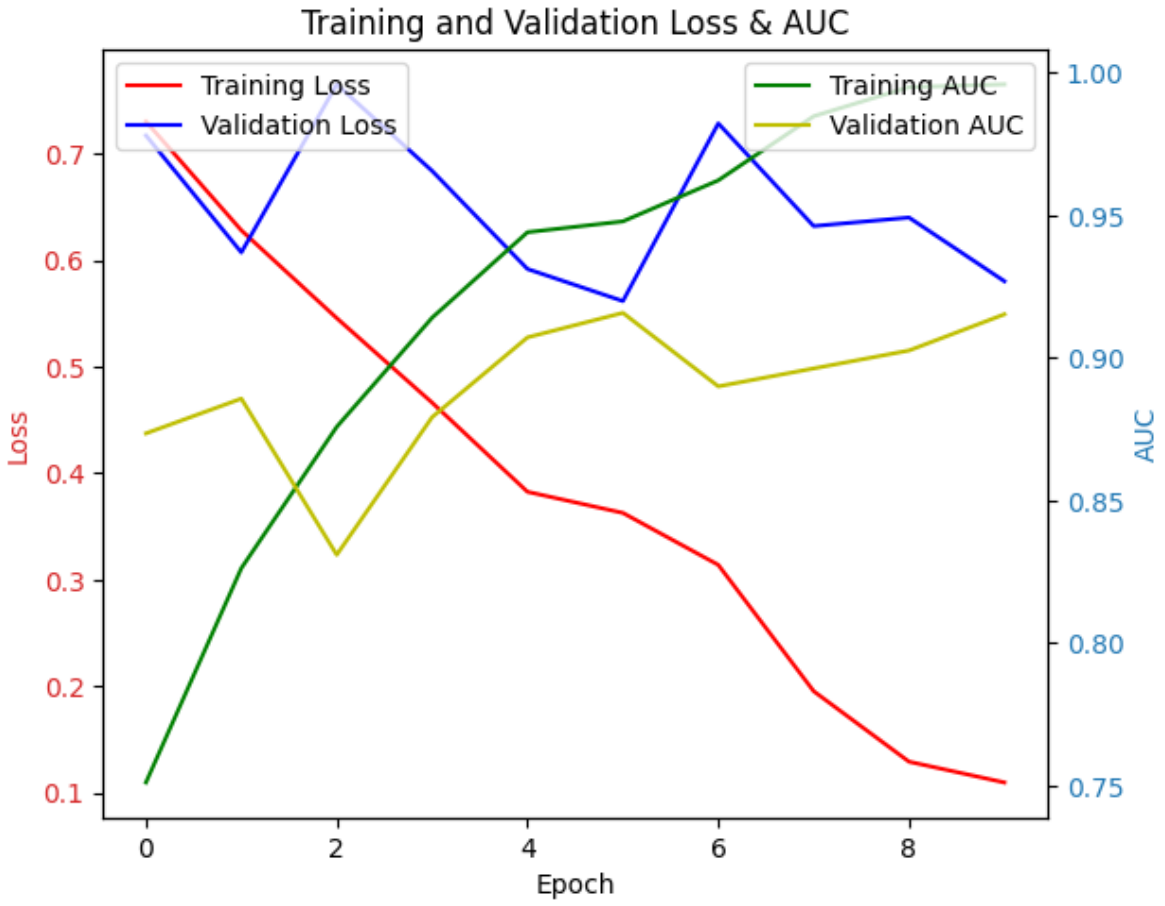
- AlexNet: this is the base model given by the professor
- MobileNet: has a good trade-off between model size and accuracy
- GoogleNet: has the ability to capture multi-scale features efficiently
- ResNet: very deep network, known for image classification purposes
- ResNext: an extension of ResNet
- EfficientNet: maintains a high accuracy and balance

These are the scores we obtained by using the aforementioned models:

Pre-trained Model	AUC score
AlexNet (given)	0.7344
MobileNet	0.8769
GoogleNet	0.8906
ResNext	0.8914
ResNet	0.9053
EfficientNet	0.9158

As we can see from the table above, the best performing pre-trained model was EfficientNet, with an average AUC score of 0.9158 obtained in epoch 5. We also want to note that after some trials, we decided to just train the model for 10 epochs, even though there might have been some overfitting as seen in the plot below. However, that did not happen for other models such as ResNet, where we had to train the model for just 2 epochs to obtain the highest AUC score and avoid overfitting.

This is the loss and AUC plot for our best performing fine-tuned pre-trained model:



4. CONCLUSIONS

During the course of this assignment, we believe that we improved our knowledge of CNNs as we were trying out different things in the implementation of our custom network. We had to go through a trial and error process to see which parameters and architecture best suited our specific problem. This made us understand in much more detail how to design a CNN when you have a specific dataset and characteristics.

Furthermore, we believe that we ended up obtaining fairly good results in terms of AUC scores and we are satisfied with our work. However, there is always room for improvement and maybe there could have been a better combination of parameters that we missed out on or we could have reduced overfitting even more, but we are happy with our results.

5. REFERENCES

- Torchvision models: https://pytorch.org/vision/0.9/models.html
- EfficientNet: https://pytorch.org/hub/nvidia_deeplearningexamples_efficientnet/
- Batch Normalization: BatchNorm2d — PyTorch 2.1 documentation
- Dropout: Dropout — PyTorch 2.1 documentation
- Maxpool: MaxPool2d — PyTorch 2.1 documentation
- Batch Size: Determining the Right Batch Size for a Neural Network to Get Better and Faster Results | by Rukshan Pramoditha | Data Science 365 | Medium