- model is trained on Google Colab that has a session time limit of 12 hours and a GPU capacity of 12 gb. This makes training a model with a more complex architecture or a greater batch size very challenging.

- Dataset was obtained from the repository created by the original paper publishing team, <https://github.com/udacity/dermatologist-ai>

[add dataset info]. Given the limited storage capacity on Google Drive, the dataset was downloaded every time the Colab environment was setup for training.

- The framework used is Pytorch

- Batch size = 64, greater batch couldn’t be loaded due to limited GPU memory and and a smaller one wasn’t enough for regularisation.

- We are using transforms to augment the images. The transform include random rotations (with a maximum limit of 5 degrees with respect to the centre of the image) , random horizontal flip with a probability of 10% and the same for a vertical flip.

- The images are then converted to tensors and right before training are scaled to the range [-1, 1].

- We tried creating a model from scratch, combining dense layers and convolution layers. But it we soon ran out of GPU space and even our best models couldn’t come close to the accuracy we were getting using pre-trained models, so we decided to go with transfer learning

- We are using a pre-trained resnet152 model and removing its last layer, the classifying layer and replacing it with our own sequence of layers, viz.

1. Leaky Relu : with a negative slope of 0.06. The value was fine tuned after testing multiple times. This activation function was chosen to make sure gradients flow back to change the pre-trained weights t conform it to the new dataset.
2. Dropout : this is the first dropout layer in the sequence with a probability of 20%. This value too was obtained after fine tuning.
3. Dense Layer : this is the first fully connected layer which takes the input from the pre-trained layers of the resnet152 model and outputs a layer of 512 nodes.
4. Batch Norm : since the resnet152 model utilizes batch normalization multiple times we added one after the first dense layer and as expected it helped in decreasing the validation loss.
5. Leaky Relu : once again we used leaky relu to make sure the previous layer get gradients to update their weights, although the negative slope value we use here is lower given the lesser number of layers it’s supposed to update.
6. Dropout : another dropout layer added to avoid overfitting with a 50% probability of dropping a node.
7. Dense Layer : this final fully connected layer is used for the final classification. It classifies the 512 output nodes from the previous dense layer into 3 categories, i.e., melanoma, nevus and seborrheic\_keratosis.

- We initialize the weights of the two dense layers added separately using normal xavier distribution

- We also froze some initial layers of the resnet152 model to prevent it’s weight from changing because the high level features are almost similar for both datasets, the one we’re using and the imagenet dataset. Plus our dataset is not as large as the imagenet.

- the model is trained for 25 epochs with the initial learning rate of 0.05.

- the criterion we’re using is the CrossEntropyLoss as it’s the most popular and of the most efficient loss function for classification problems.

- we’re using the Stochastic Gradient Descent optimizer to update weights of the model. We’ve included a momentum with value 0.85 to prevent the loss function from getting stuck in a local minima.

- we’ve used a learning rate scheduler to prevent oscillating around a minima. The scheduler reduces learning rate by a factor of 5 if the validation loss doesn’t go below the lowest validation loss for three epochs consecutively.

- during the training we save the model with the minimum validation loss as a checkpoint, i.e., we save its weights in a file.

- after the training is complete we load the checkpoint with the minimum loss into the model