Skin Disease Image Classification

Manjiri Bhandarwar MB8070@NYU.EDU
Hannah Phung HMP3554@NYU.EDU
Yen Le YTL2008@NYU.EDU

Milestone 2

1. Methodology

1.1 Data Preprocessing

Our dataset is the HAM1000 which contains 10015 instances of 7 different types of skin diseases. The metadata for this dataset is stored in a CSV file so we loaded it using pandas. Next, to clean the dataset, we had to handle missing values. The age column has 37 null values, which were replaced using mean imputation. Next, we observe that the age value can be 0 and there is an "unknown" sex category. Thus we subsetted the dataset to remove these rows. Although there was a reduction in the number of observations, these reductions were only for three classes "nv", "bkl", and "vasc", 2 of which are majority classes and one is a minority class. Only 4 observations were removed from the minority class. The reason we are dealing with categorical data right now is that for Milestone 3, we may want to explore a way to incorporate it to enhance our model's predictions.

We got the path for the images based on the image_id column, loaded the images, resized them to 125x100 pixels, converted them into NumPy arrays, and stored them in a new column 'image'. Then, we categorically encoded the different lesion types (0-6) and stored them in column 'cell_type_idx'. The "image" column is our input and 'cell_type_idx' is our target.

The dataset is highly imbalanced (see appendix for distribution). A common way to tackle this is by oversampling the minority classes. However, because the class with the smallest samples is 58 times smaller than the class with the largest samples, using random oversampling would cause there to be multiple duplicates of the minority class images, which may cause our model to overfit. To deal with data imbalance, we implement data augmentation using ImageDataGenerator for training data. Before data augmentation, we split our dataset into training, test, and validation sets. First, the dataset was split into train and test using an 80-20 split, then the train set was further split into train and validation using a 90-10 split.

We used ImageDataGenerator to randomly rotate existing images by 0 to 10 degrees, randomly shift images horizontally and vertically by a specified fraction of the total width, randomly zoom in the image by a factor of 0.1, and randomly flip the images.

1.2 Model Building and Exploring Ways to Enhance Model Performance

The process involves using pre-trained (on Imagenet) models in Keras' library, DenseNet201, and ResNet50, freezing 95 layers, and redefining the fully connected (FC) layers to be compatible with our data. In the FC layer, we take the base model's output, apply GlobalAveragePooling2D(), then apply the Dense layer with 'relu' activation and 256 units, and finally a Dense layer with 7 units and softmax, corresponding to our 7 classes. This is consistent with the FC layer of a DenseNet201. We use an exponential decay learning rate scheduler starting with an initial learning rate of 0.01, decay_steps=10000, and decay_rate=0.9. We use an Adam optimizer in which the learning rate scheduler is used. Additionally, the model is compiled using categorical cross-entropy as the loss function and accuracy as the metric for evaluation. Early stopping is set up to monitor validation loss. It stops training if there's no improvement in validation loss for 15 consecutive

epochs and restores the model's weights to the ones that achieved the best validation loss. Each model is trained for 50 epochs and with a batch size of 60. Loss v.s. Epochs, Accuracy v.s. Epochs are plotted for training and validation sets. Then, the model is used to make predictions on the test set and the confusion matrix and classification report are plotted for this.

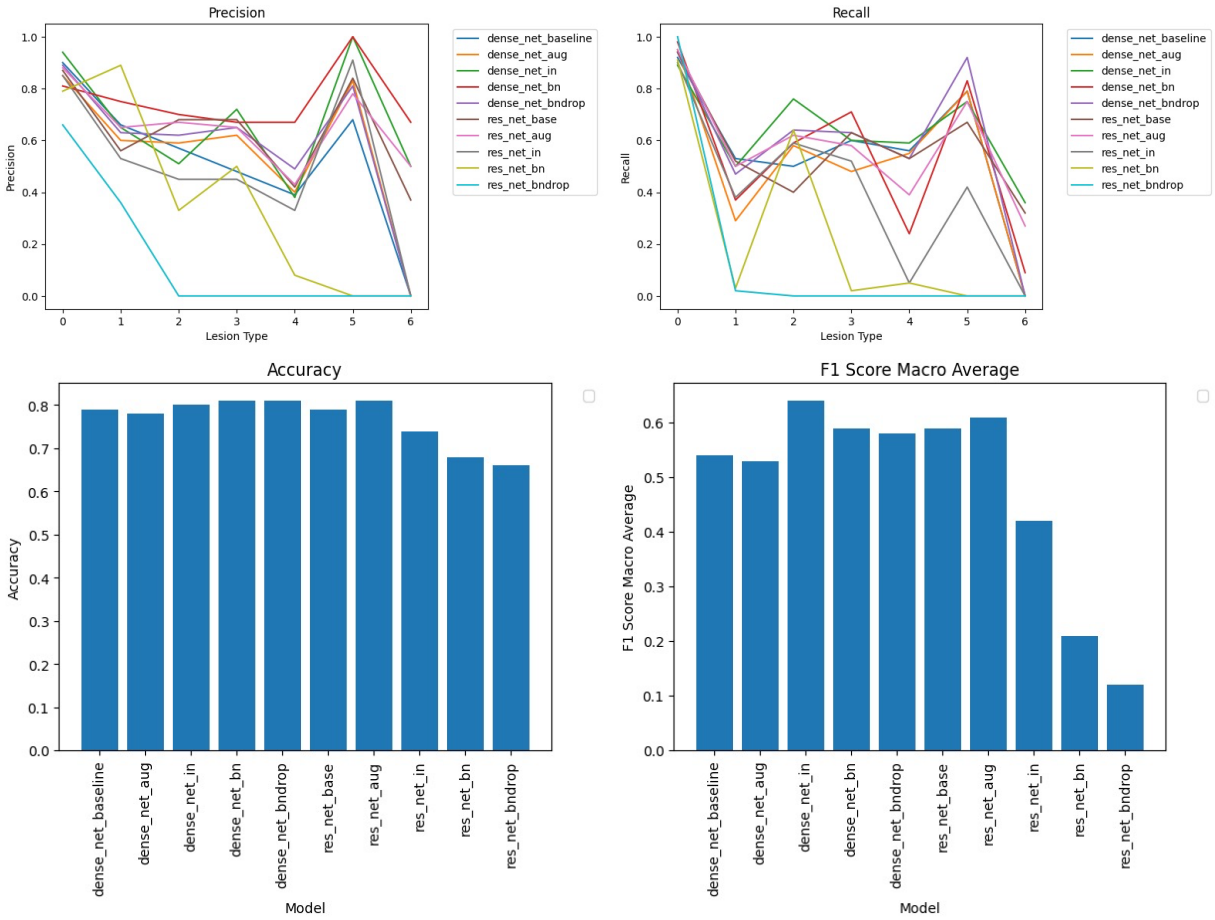
To explore ways of enhancing model performance, we built on our baseline models by including data augmentation, then adding input normalization and standardization, then adding BatchNormalization and Dropout layers with a dropout probability of 0.5. To normalize the data we rescaled the image arrays by dividing it by 255 so that the pixel values are between 0-1. Standardization was only fitted to the training data.

1.3 Hyperparameter Tuning

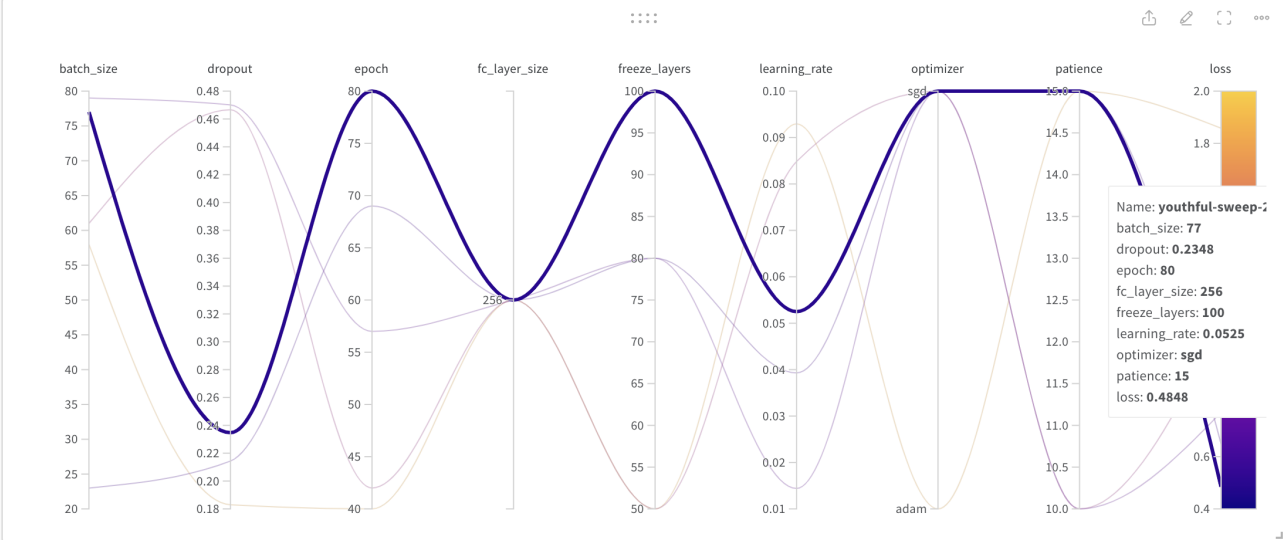
For hyperparameter tuning, we used Weights and Biases (W&B) to run a hyperparameter sweep with random search and select the best combination of hyperparameters by minimizing loss. The hyperparameters include optimizer, fc_layer_size, dropout, learning_rate, freeze_layers, patience, batch_size, epoch (see appendix). Since in our previous steps, the DenseNet201 model consistently outperformed ResNet50, we decided to run the hyperparameter tuning for DenseNet.

2. Results

Due to data imbalance mentioned above, we will focus on the precision-recall and F1 metrics over accuracy since they would better measure the model's performance. Accuracy would be biased towards the majority class.



Best hyperparameter combination



- batch_size: 61
- dropout: 0.4665888846145652
- epoch: 42
- fc_layer_size: 256
- freeze_layers: 50
- learning_rate: 0.08482257818026998
- optimizer: sgd
- patience: 10

The loss for the best hyperparameter combination is 0.4848.

3. Analysis

For DenseNet201, as we progress from the baseline model to the model that uses both batch normalization and dropout, the accuracy stays around the same (0.78-0.80). For ResNet50, accuracy drops after adding input normalization. This is more accurate than a naive classifier that predicts everything as the majority class (approx. 66%).

From the recall and precision graphs, we can see that for the minority classes 3,4,5,6, the highest recall and precision occur for different variations of DenseNet201, indicating that DenseNet201 is generally better than ResNet50. According to Zhang, DenseNet exhibits enhanced capacity through the concatenation of features across multiple layers. Research has demonstrated its better feature utilization efficiency, surpassing the performance of ResNet (Zhang et al.). Choosing the best model depends on one's objectives, but for this paper, we chose the model with the most balanced recall and precision performance, indicated by the macro-average F1 score.

For DenseNet201, the macro-averaged F1 score increased up till and peaked after adding input normalization, and for ResNet50 it increased up till and peaked after adding data augmentation and then started decreasing for both. Data augmentation helps by increasing the volume and diversity of training data which helps prevent overfitting (Mumuni and Mumuni), and

input normalization and standardization help by ensuring algorithmic stability, helping the model learn and converge faster (Bhandari).

Batch normalization and dropout reduced F1 scores for both DenseNet201 and ResNet50, but way more significantly for ResNet50. Because the architecture of these pre-trained models are fixed, we could only add batch normalization and dropout at the end and not after each activation. Thus, batch normalization was not effective in preventing an internal covariate shift. Another problem that could occur is that batch normalization could help training if the batch size is at a decent size but might be effective for inference since the model is predicting a single point of unseen data at a time (Chen et al.).

Overall, we were able to get a model that performs in a more balanced manner across all classes.

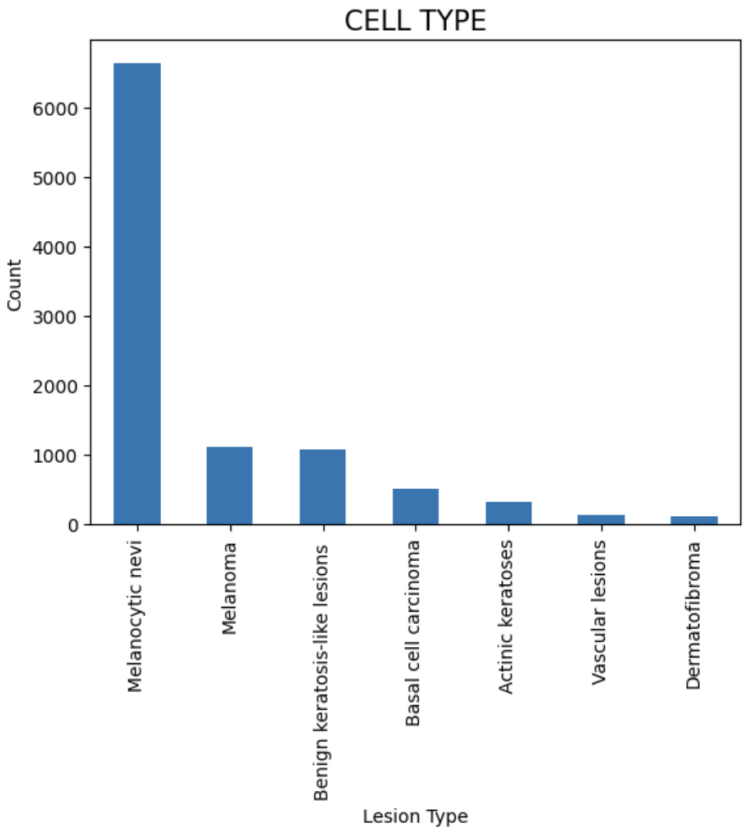
4. Plan for additional analysis

In the next milestone, we plan to get the F1 score for model with tuned hyperparameters. We aim to ensemble different models together to capture a wider range of patterns within our data. This will potentially improve predictive abilities and reduce the likelihood of overfitting (Pramoditha, Rukshan; Shirsat, Mithilesh). We will also try to incorporate categorical data alongside image data to provide a more holistic view of the dataset. This additional input potentially allows the model to learn more context about the patient; thus, helping to improve the classifications of skin disease (Farshadjafari). We will continue to conduct hyperparameter sweeps to find a hyperparameter set that maximizes the models' performance. In hyperparameter tuning, we will implement a Cyclic Learning Rate scheduler to adjust the learning rate between epochs as we train to help our model converge faster. Lastly, to ensure that our model is generalizable to the new dataset, we plan to validate its performance on additional datasets, such as DermNet. In this process, we will only focus on 7 types of skin diseases that is compatible with the primary dataset.

5. Work Plan

Our goal for this Milestone was to replicate different CNN models in our own environment and identify the best models. Manjiri is responsible for data pre-processing and condensing the research paper, running some variations of DenseNet201 and ResNet50, Hannah is responsible for training different variations of DenseNet201 and hyperparmater sweep, Yen is responsible for training different variations of ResNet50. Since we do not have the available code for ConvNeXtL, we decided to train only two types of CNN models and perform hyperparameter tuning on the best model found. We help each other with each step and have multiple weekly check-ins. For Milestone 3, we would like to perform additional analysis by combining models, explore ways to incorporate categorical data as input, and applying models on secondary datasets for audit the generalizations of the model on secondary datasets. Hannah will be responsible for combining models, Manjiri will be responsible for finding ways to incorporate categorical data, and Yen will audit the generalization. If time permits, we will do more iterations for hyperparameter tuning as well and test out different learning rate schedulers.

Appendix A: Distribution of Classes



Appendix B: Hyperparameter Dictionary for Hyperparameter Tuning

```
parameters_dict = {
    'optimizer': {
        'values': ['adam', 'sgd']
        },
    'fc_layer_size': {
        'values': [128, 256, 512]
        },
    'dropout': {
        'distribution': 'uniform',
        'min': 0,
        'max': 0.5
        },
    'learning_rate': {
        # a flat distribution between 0 and 0.1
        'distribution': 'uniform',
        'min': 0.001,
        'max': 0.1

        #'values': [0.1, 0.05, 0.01, 0.005, 0.001]
      },
    'freeze_layers': {
        'values': [50, 80, 100, 150]
        },
    'patience': {
        'values': [5, 10, 15]
        },
    'batch_size': {
        # a flat distribution between 0 and 0.1
        'distribution': 'int_uniform',
        'min': 20,
        'max': 100
     },
    'epoch': {
        # a flat distribution between 0 and 0.1
        'distribution': 'int_uniform',
        'min': 20,
        'max': 100
     }
    }
```

References

Bhandari, A. "Feature engineering: Scaling, normalization, and standardization (Updated 2023)." Analytics Vidhya, 7 July 2023, www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

Chen, Guangyong, et al. "Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks." ArXiv.org E-Print Archive, 2019, arxiv.org/pdf/1905.05928.pdf.

Farshadjafari. "Skincancer Detection-Multiple Models :83%Accuracy." Kaggle, Kaggle, 30 July 2023, www.kaggle.com/code/farshadjafari97/skincancer-detection-multiple-models-83-accuracy.

Mumuni, A., and F. Mumuni. "Data augmentation: A comprehensive survey of modern approaches." Science Direct, Dec. 2022, www.sciencedirect.com/science/article/pii/S2590005622000911.

Pramoditha, Rukshan. "How to Mitigate Overfitting by Creating Ensembles." Medium, Towards Data Science, 5 Oct. 2021, towardsdatascience.com/how-to-mitigate-overfitting-by-creating-ensembles-77e9299b9ad0.

Shirsat, Mithilesh. "Ensemble Learning: Combining Multiple Models for Better Predictions." LinkedIn, 15 Apr. 2023, www.linkedin.com/pulse/ensemble-learning-combining-multiple-models-better-mithilesh-shirsat/.

Wei, Mingjun, et al. A Skin Disease Classification Model Based on DenseNet and ConvNeXt Fusion. MDPI, 14 Jan. 2023. www.mdpi.com/2079-9292/12/2/438.

Zhang, Chaoning et al. ResNet or DenseNet? Introducing Dense Shortcuts to ResNet. 2020, October 23. https://doi.org/10.48550/arXiv.2010.12496