Deep Learning Project

Multilabel Classification on Celebrity Face attributes

Group 19

Bharath Velamala, Pranshu Singh Rawat

5th May, 2024

Problem Statement:

The project's problem statement encompasses multi-label classification on an image dataset using CNN architectures. The objective is to predict labeled features within images and compare the results of two CNN architectures. Additionally, the project aims to facilitate content-based retrieval, enabling image retrieval based on specific attributes such as big eyes or big nose, which can be utilized to filter images from a group. Moreover, it aims to contribute to decision-making processes in the entertainment industry, such as casting decisions, by determining attributes like attractiveness based on facial features.

Data:

The dataset selected for this project is sourced from Kaggle and comprises celebrity face attributes. It includes approximately 202,599 face images annotated with around 40 attributes. These attributes, such as Male, Attractiveness, and Young, are represented in binary form, indicating whether a particular feature is present in each image. The dataset, derived from a research paper, also contains a CSV file that serves as an evaluation set, partitioning the data into test, train, and validation sets based on image file names. Other files within the dataset, although present, are not utilized for this project. The images encompass a wide range of pose variations, background complexities, and a diverse set of individuals, making it an ideal resource for robust model training. All data resides within a folder, where images are categorized into separate folders for training, testing, and validation. The csv files, however, are located outside the main source folder.

In summary, this dataset provides a comprehensive platform for exploring and advancing the capabilities of deep learning models in face detection and attribute recognition tasks. Its abundance of images, detailed annotations, and diverse characteristics make it a valuable asset for research and development in the field of computer vision.

Exploratory Data Analysis: Prior to model development, an exploratory data analysis was conducted to understand the distribution of attributes, detect class imbalances, and gain insights into the dataset's characteristics. Upon exploring the dataset, we discovered that the distribution of labels across the 40 features is not balanced. In other words, the dataset exhibits class imbalance issues, as certain classes may be overrepresented while others are underrepresented. Subsequently, we addressed this imbalance by employing class weights during the model training process, a strategy which will be elaborated upon later. Furthermore, we conducted visualization of the distribution of notable classes such as Male and Young. Additionally, we examined the correlation between each class to identify any high correlations between them.

Project Goals:

The project's primary objective is to create a dependable multi-label classification model for accurately identifying attributes linked to celebrity faces in images. Furthermore, the project aims to assess the efficacy of deep learning methodologies, particularly ResNet50 and DenseNet121 convolutional neural networks, in meeting these goals. Lastly, the project seeks to compare the performance of both models to determine which yields superior results.

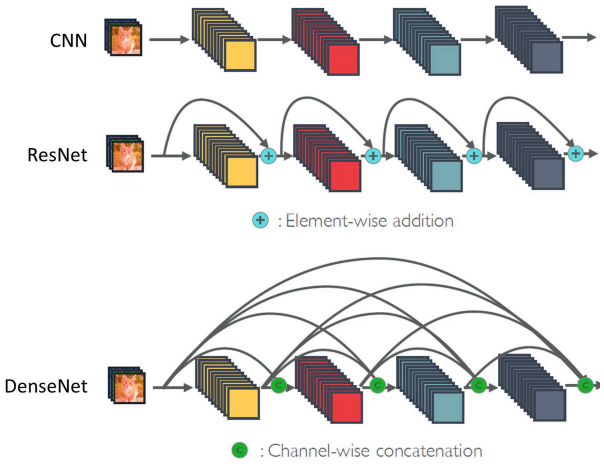
Deep Learning Methods:

CNN architecture is chosen for this problem due to the nature of the dataset, which comprises images. Additionally, CNNs excel in capturing spatial dependencies within images, which is crucial for tasks like facial attribute recognition. Their ability to detect features at different scales, from fine details to global context, allows them to effectively capture the diverse characteristics present in celebrity face images. We have chosen two of the well-known CNN architectures which are ResNet50 and DenseNet121. ResNet50 is known for its unique residual connection design, ResNet-50 consists of 50 layers, facilitating the training of deeper networks. DenseNet121 is renowned for densely connected[1] layers, DenseNet121 fosters feature reuse and enhances gradient flow with 121 layers.

ResNet50: ResNet architecture adheres to two fundamental design principles. Initially, the number of filters in each layer remains consistent, corresponding to the size of the output feature map. Subsequently, if the feature map undergoes size reduction by half, the number of filters doubles to uphold the time complexity of each layer. ResNet architecture incorporates residual connections, enabling the preservation of information across layers and effectively addressing the vanishing gradient problem.

DenseNet121: DenseNet architecture simplifies the connectivity pattern between layers introduced in other architectures. Each layer receives input from all preceding layers, promoting feature propagation throughout the network. Interestingly, DenseNets require fewer parameters compared to traditional CNNs due to their unique connectivity pattern, which eliminates the need to learn redundant feature maps. DenseNets address the challenges of training very deep networks by ensuring that each layer has direct access to gradients from the loss function and the original input image, mitigating issues with information and gradient flow.

The main difference between ResNet and DenseNet is DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them. In ResNet features are passed down in an element wise addition process. We employ pre-trained PyTorch models to train our image data using both CNN architectures. Leveraging pre-trained models allows us to capitalize on existing knowledge and features learned from large-scale datasets, enhancing the efficiency and effectiveness of our training process.



[1] Image source: https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

Modeling and Evaluation:

Due to limitations in computational resources, we opted to train our model using a subset of 40,000 images, validate it on 5,000 images, and reserve another 5,000 images for testing purposes. To commence the process, we identified several preprocessing steps that are outlined in detail below.

Dataset Preparation: To get started with modeling we need to implement the data preparation where we need to transform the images. Image transformations are essential preprocessing steps applied to the training, validation, and testing data to ensure consistency and improve model performance. In this implementation, transformations include resizing images to a fixed size of 64x64 pixels, horizontal flipping for data augmentation, converting images to PyTorch tensors, and normalization using mean and standard deviation values derived from the ImageNet dataset. These transformations help in standardizing the input data and reducing computational complexity by decreasing image size.

We then have created the *GetCelebDataset* class which is responsible for preparing the dataset by loading images and their corresponding labels. Images are loaded using PIL (Python Imaging Library) and converted to RGB format. Labels are converted to PyTorch tensors. Additionally, the specified transformations are applied to the images. This class ensures that the data is properly formatted and ready for training, validation, and testing.

We then have used PyTorch's DataLoader to efficiently load batches of data during training, validation, and testing phases. It enables parallel data loading using multiple workers, which is particularly beneficial for GPU utilization. The *dataloader* function sets up data loaders for the training, validation, and test datasets, with specified batch sizes and device configurations.

Class Weight Generation: To counter the issue of class imbalance within the dataset, class weights are generated to ensure that the model is trained effectively across all classes. This process involves calculating the frequency of each class label in the dataset and computing corresponding weights. These weights are inversely proportional to the class frequencies, ensuring that less frequent classes receive higher weights during training. By incorporating class weights into the loss function, the model learns to give more importance to minority classes, thus improving overall performance and reducing bias towards dominant classes.

As mentioned earlier we are using a pre-trained ResNet50 and DenseNet121 models as the backbone architecture, the model benefits from the feature representations learned from large-scale datasets like ImageNet. This transfer learning approach allows the model to adapt to the nuances of the celebrity face attribute dataset more efficiently, even with limited labeled data.

Training Procedure: The training procedure is a critical phase where the model learns to understand the complex relationships between input images and their associated attribute labels. The iterative nature of training, spanning multiple epochs, enables the model to gradually refine its parameters and improve its predictive capabilities. During each epoch, mini-batches of images are sampled from the training loader, ensuring that the model learns from a diverse set of examples in each iteration. Within each epoch, the model undergoes forward propagation, where input images are passed through the network to generate predictions. These predictions are then compared against the ground truth labels using a binary cross-entropy loss function as all of our classes are having binary values. The gradients of the loss with respect to the model parameters are computed during backpropagation, and optimization algorithms like Adam adjust the parameters to minimize the loss.

To address the challenge of class imbalance within the dataset, a weighted binary cross-entropy loss function is employed. This ensures that the model assigns appropriate importance to each attribute class during training, effectively learning from both prevalent and rare attributes. Throughout the training process, both the training and validation losses are monitored. Training loss reflects the model's performance on the training data, while validation loss provides insights into its generalization ability on unseen validation data. By comparing these loss values across epochs, it becomes possible to assess whether the model is converging towards an optimal solution.

Evaluation: We have created an *accuracy_score* function that calculates the overall accuracy of the model's predictions on the test dataset. It compares the predicted labels with the true labels and calculates the proportion of correctly predicted labels out of all labels in the dataset. This metric provides a comprehensive measure of the model's overall performance in correctly classifying images across all attribute classes. We have created a *get_multilabel_evaluation* function that computes precision, recall, and F1-score metrics specifically for the test dataset. It evaluates the model's performance in correctly identifying positive and negative instances for each attribute class. By comparing the model's predictions with the true labels, these metrics offer insights into the model's ability to generalize to unseen data and accurately predict attribute labels in real-world scenarios.

Fine-tuning and Hyperparameter Optimization: Fine-tuning involves adjusting the hyperparameters of the model, such as learning rate and dropout probability, to optimize its performance. Hyperparameters are typically tuned based on the validation performance of the model. Techniques like learning rate schedules and early stopping are employed to prevent overfitting and ensure the model generalizes well to unseen data.

Hyperparameters which we have chosen are batch size which determines the number of samples processed in each iteration during training. A larger batch size may lead to faster convergence and smoother optimization but requires more memory. Conversely, a smaller batch size may provide more stochasticity and generalization but can result in slower training. Learning Rate, which controls the step size of parameter updates during

optimization. A higher learning rate may lead to faster convergence but risks overshooting the optimal solution. Conversely, a lower learning rate may ensure more stable training but may take longer to converge. Number of Features, which determines the dimensionality of the feature space in the final fully connected layer of the model. This parameter directly influences the model's capacity to capture and represent complex patterns within the data. Dropout, which is a regularization technique used to prevent overfitting by randomly dropping out units (along with their connections) during training. The dropout probability determines the probability of dropping out a unit during each training iteration. A higher dropout probability increases regularization and may prevent overfitting but can also hinder learning.

Hyperparameter	Values
Number of Epochs	8
Batch Size	64
Learning Rate	0.01, 0.001
Number of Features	512
Drop Out	0.3, 0.5

Execution: A parameter grid is defined to explore different combinations of hyperparameters. The grid consists of various values for batch size, learning rate, number of features, and dropout probability. The parameter grid is iterated over using nested loops, allowing for exhaustive exploration of all possible combinations of hyperparameters. For each combination, a new model is trained using the specified hyperparameters. The training process involves setting up data loaders with the specified batch size, defining the model architecture, and training the model on the training data.

After training each model, its performance is evaluated on the validation set to assess its generalization ability. The model's validation performance is crucial in determining the optimal set of hyperparameters. Metrics such as validation loss and accuracy scores are computed for each model, providing insights into its performance across different hyperparameter configurations. Once all models have been trained and evaluated, the model with the best validation performance is selected as the final model. This decision is based on the metrics obtained during validation, such as the lowest validation loss or highest validation accuracy. The selected model represents the optimal combination of hyperparameters that yields the best performance on the validation set.

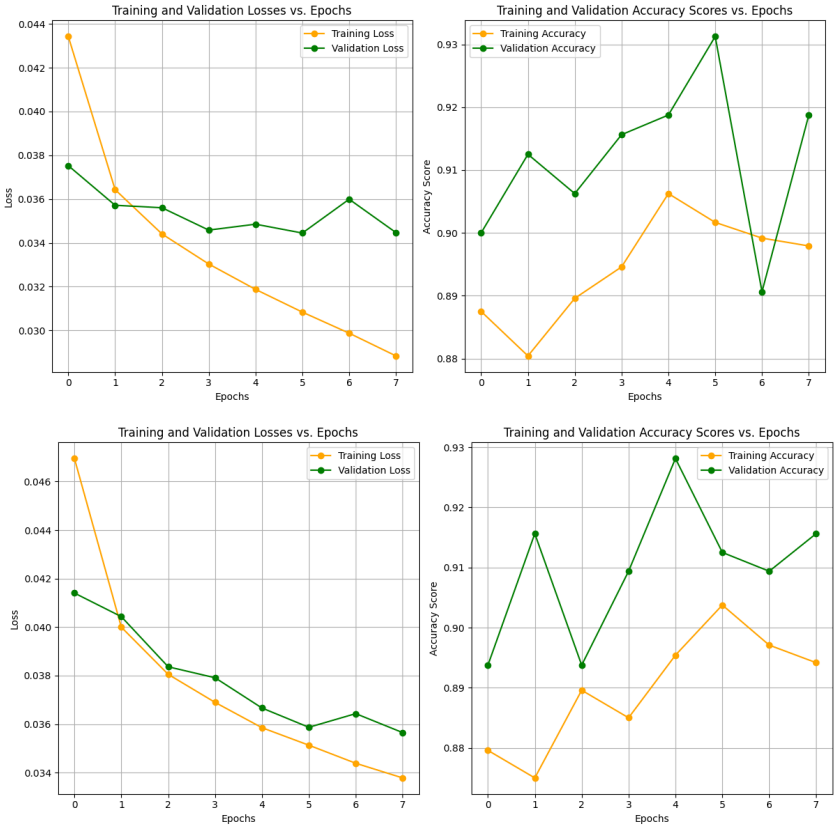
Finally, the selected model's state dictionary is saved to our system for future use. This allows for easy retrieval and deployment of the trained model for making predictions on new, unseen data. Additionally, the model's hyperparameters are stored in the model name, providing a reference for the configuration used during training.

Results:

We've computed the accuracy for both training and validation sets across various hyperparameter configurations. This analysis guides our selection of the model to be

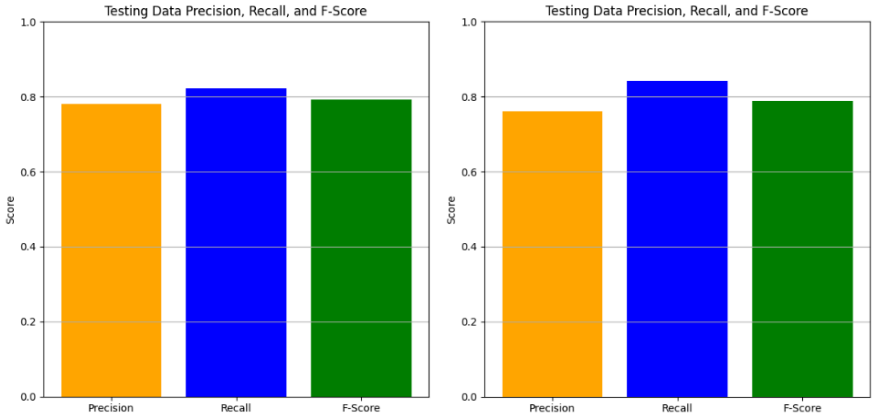
tested on the test dataset. The model we selected was for batch_size = 64, learning_rate = 0.001, num_features = 512, drop_out = 0.3 for both ResNet50 and DenseNet121. The training and validation results for that are as follows:

This visualization depicts the training and validation losses and accuracies across epochs for the ResNet50 (top) implementation. Notably, both training and validation losses exhibit a decreasing trend with each epoch. The next visualization is for DenseNet121 (bottom) but it is also following the same trend as ResNEt50. However, an unintended observation for both the models is that the accuracy of the validation set surpasses that of the testing set. The accuracy for validation is averaging above 90 but training is averaging below 90 for both of the trained models.

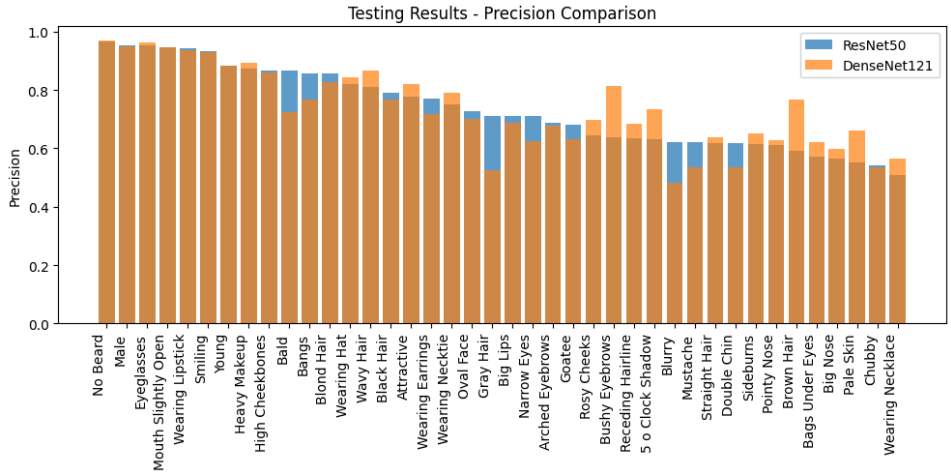


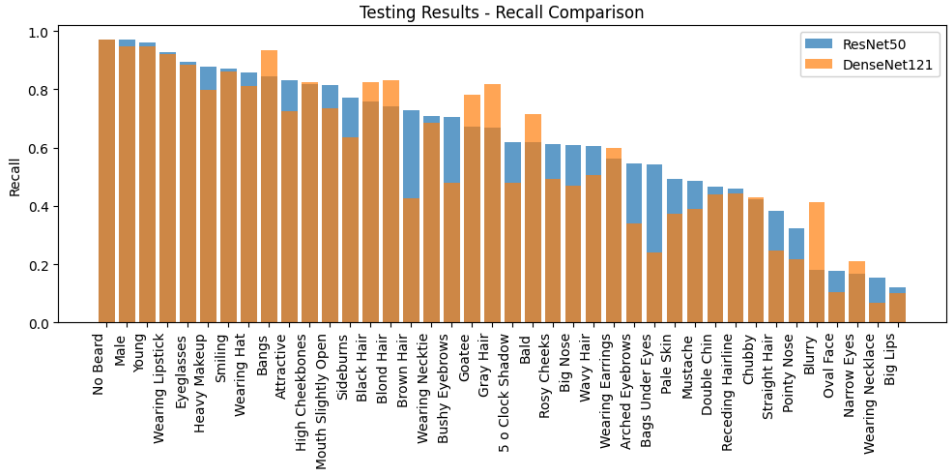
The precision, recall, and F1-score for the test data have been computed and are presented below: ResNet50(left) and DenseNet121(right)

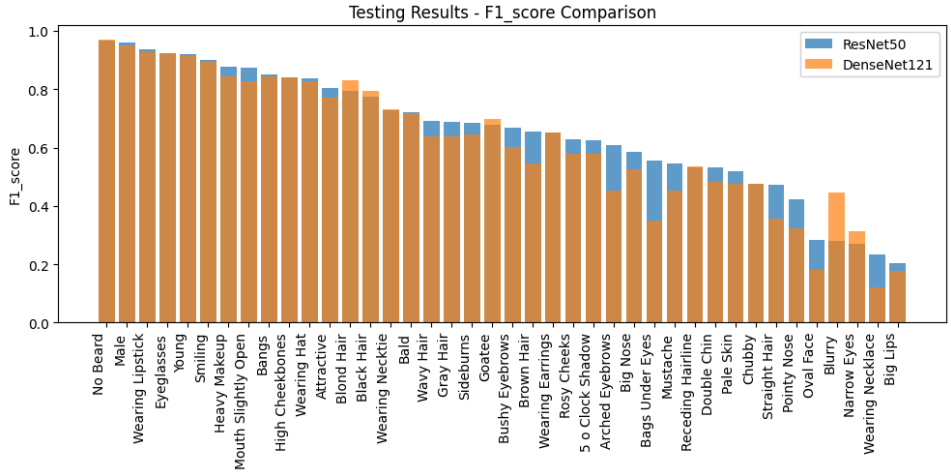
The Overall Recall score for Denset121 is greater than ResNet50 but Precision and F1-Score are greater for ResNet50. The calculated scores are weighted averages across all the 40 classes. For a better understanding on how it performed on each we have also computed that information and presented below.



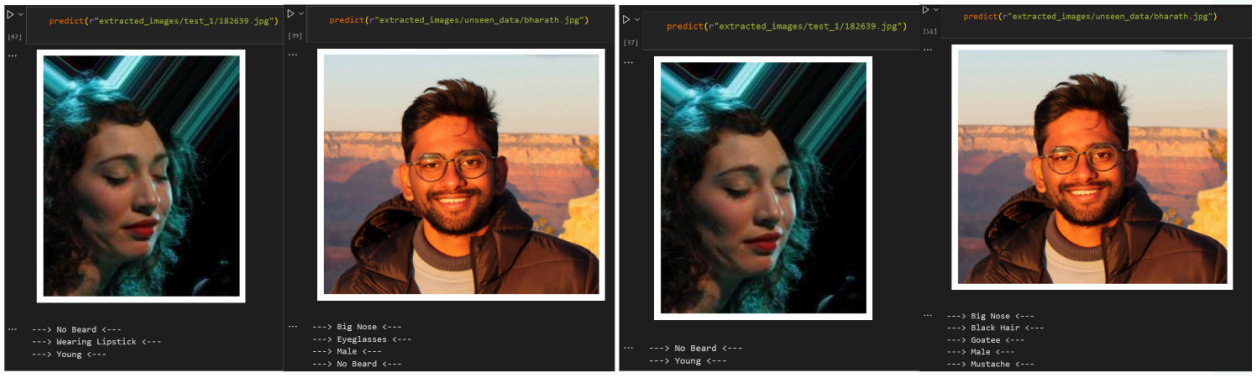
We used *sklearn* to get the classification report on all the classes and then saved that to a csv file to later visualize and compare. After visualizing the below comparison we can identify that the model's prediction is not equivalent to all the classes involved. Both models seem to perform exceptionally well in the *No Beard*, *Male*, *Eyeglasses*, *Young* class. And there are classes in which ResNet50 seems to outperform DenseNet121 and vice versa. From each metric's lowest values we can say that the model is not able to perform well for those classes as it was unable to learn those classes from the images.







Afterward, we load the best-performing model trained on the training data and evaluate its accuracy on the test dataset and one unseen image to understand its performance. We have taken the leftmost image from our test dataset and the right image is of our own as we wanted to see how it would perform on unseen image.



ResNet50 DenseNet121

It's evident that both models can accurately classify most attributes or classes for each image, with the exception of the "No Beard" classification for ResNet50, despite high metrics for that class. Interestingly, each model seems to have learned different attributes, as the classifications differ between models for the same image.

Conclusion:

Our analysis revealed several key findings. Firstly, both ResNet50 and DenseNet121 demonstrated promising performance in accurately predicting multiple attributes associated with celebrity faces. However, ResNet50 exhibited slightly better overall performance metrics, including accuracy, precision, recall, and F1-score, compared to DenseNet121. However, training on more images we would hope that DenseNet121 would outperform ResNEt50. Despite the promising results, our project encountered several challenges and limitations. Class imbalance within the dataset posed a significant challenge, which we addressed by applying class weights during model training.

In addition to our primary experimentation, we expanded our training dataset to include 40,000 images and extended the training duration to 12 epochs. Unexpectedly, this adjustment resulted in misclassification of certain labels or attributes, deviating from our intended outcomes. Furthermore, we attempted to streamline the classification task by reducing the number of labels from 40 to 25, yet this modification also led to inaccuracies in label prediction. In conclusion, our project highlights the potential of deep learning techniques in multi-label classification tasks, particularly in the domain of celebrity face attribute prediction. While there are challenges to overcome and areas for further exploration, our findings contribute to the growing body of knowledge in computer vision and image analysis.

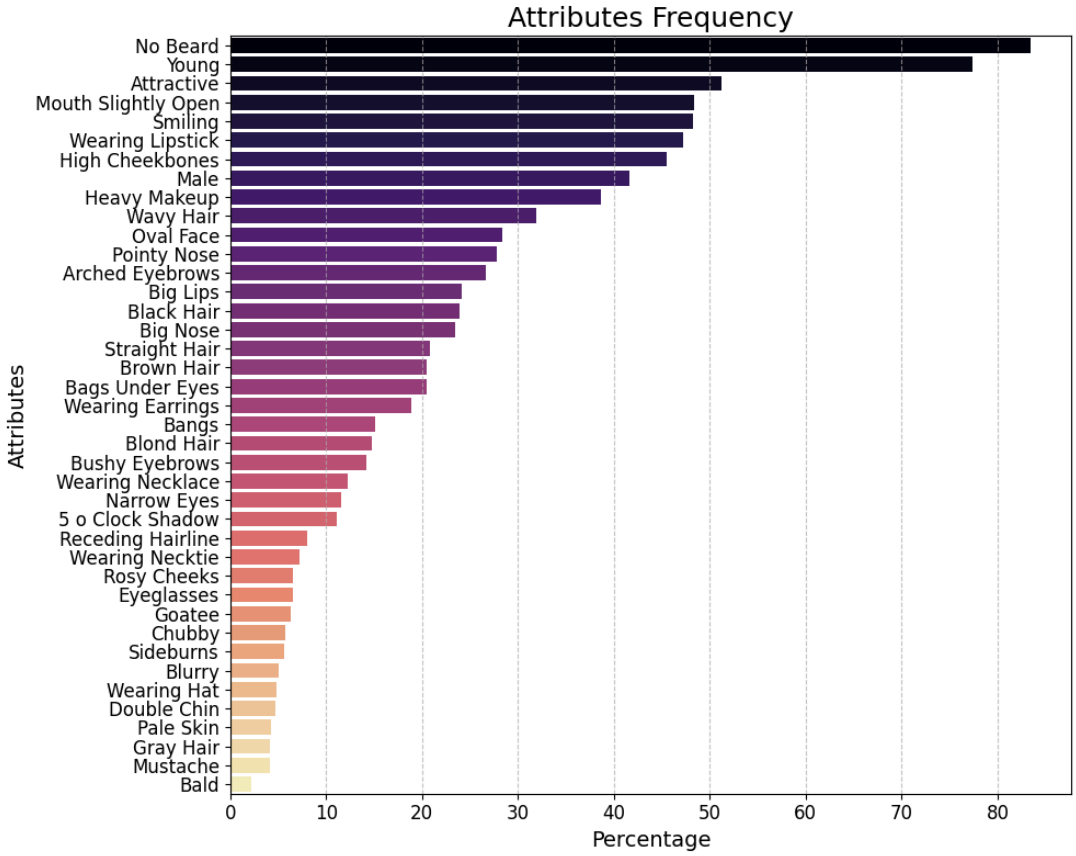
Other Information:
Code Link: https://github.com/bharath03-a/multilabel_image_classification
Code File Names: MIS_548_DL_PROJECT_ResNet50.ipynb, MIS_548_Metrics.ipynb, MIS_548_DL_PROJECT_DenseNet121.ipynb.

References:

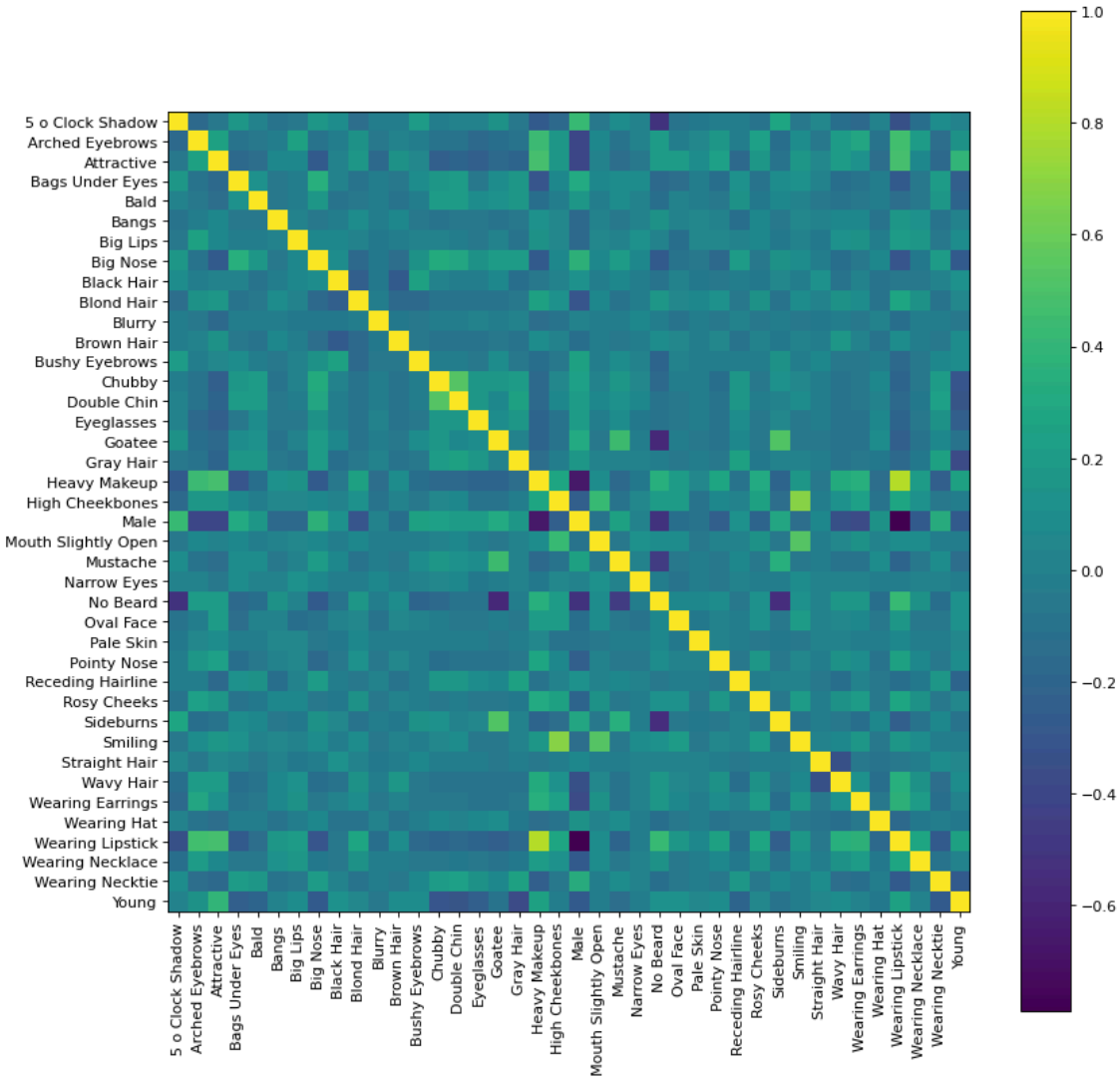
1) https://www.researchgate.net/figure/The-architecture-of-ResNet-50-model_fig4_349717475
2) https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a
3) https://learnopencv.com/multi-label-image-classification-with-pytorch-image-tagging/

Relevant Images:

1. Class or attribute frequencies showing the class imbalance issue



2. Correlation matrix of all the classes or attributes involved



3. ResNet50 model training - batch_size = 64, learning_rate = 0.001, num_features = 512, drop_out = 0.3

```
Model: | batch_size = 64 learning_rate = 0.001, num_features = 512, drop_out = 0.3 |
================================================================================
Epoch 1/8: 100%|██████████| 625/625 [01:08<00:00,  9.17batch/s, accuracy_score=0.887, loss=0.0383]
Epoch [1/8], Train Loss: 0.0434, Train accuracy Score: 0.8875, Val Loss: 0.0375, Val accuracy Score: 0.9000
Epoch 2/8: 100%|██████████| 625/625 [01:06<00:00,  9.35batch/s, accuracy_score=0.88, loss=0.0375]
Epoch [2/8], Train Loss: 0.0364, Train accuracy Score: 0.8804, Val Loss: 0.0357, Val accuracy Score: 0.9125
Epoch 3/8: 100%|██████████| 625/625 [01:04<00:00,  9.73batch/s, accuracy_score=0.89, loss=0.0358]
Epoch [3/8], Train Loss: 0.0344, Train accuracy Score: 0.8896, Val Loss: 0.0356, Val accuracy Score: 0.9062
Epoch 4/8: 100%|██████████| 625/625 [01:04<00:00,  9.71batch/s, accuracy_score=0.895, loss=0.0323]
Epoch [4/8], Train Loss: 0.0330, Train accuracy Score: 0.8946, Val Loss: 0.0346, Val accuracy Score: 0.9156
Epoch 5/8: 100%|██████████| 625/625 [01:04<00:00,  9.69batch/s, accuracy_score=0.906, loss=0.0344]
Epoch [5/8], Train Loss: 0.0319, Train accuracy Score: 0.9062, Val Loss: 0.0349, Val accuracy Score: 0.9187
Epoch 6/8: 100%|██████████| 625/625 [01:04<00:00,  9.67batch/s, accuracy_score=0.902, loss=0.0271]
Epoch [6/8], Train Loss: 0.0308, Train accuracy Score: 0.9017, Val Loss: 0.0344, Val accuracy Score: 0.9313
Epoch 7/8: 100%|██████████| 625/625 [01:05<00:00,  9.51batch/s, accuracy_score=0.899, loss=0.0306]
Epoch [7/8], Train Loss: 0.0299, Train accuracy Score: 0.8992, Val Loss: 0.0360, Val accuracy Score: 0.8906
Epoch 8/8: 100%|██████████| 625/625 [01:05<00:00,  9.58batch/s, accuracy_score=0.898, loss=0.0324]
Epoch [8/8], Train Loss: 0.0288, Train accuracy Score: 0.8979, Val Loss: 0.0345, Val accuracy Score: 0.9187
```

4. DenseNet121 model training - batch_size = 64, learning_rate = 0.001, num_features = 512, drop_out = 0.3

```
Model: | batch_size = 64 learning_rate = 0.001, num_features = 512, drop_out = 0.3 |
================================================================================
Epoch 1/8: 100%|██████████| 625/625 [01:11<00:00,  8.68batch/s, accuracy_score=0.88, loss=0.0389]
Epoch [1/8], Train Loss: 0.0470, Train accuracy Score: 0.8796, Val Loss: 0.0414, Val accuracy Score: 0.8938
Epoch 2/8: 100%|██████████| 625/625 [01:07<00:00,  9.30batch/s, accuracy_score=0.875, loss=0.0398]
Epoch [2/8], Train Loss: 0.0400, Train accuracy Score: 0.8750, Val Loss: 0.0404, Val accuracy Score: 0.9156
Epoch 3/8: 100%|██████████| 625/625 [01:09<00:00,  8.97batch/s, accuracy_score=0.89, loss=0.0351]
Epoch [3/8], Train Loss: 0.0381, Train accuracy Score: 0.8896, Val Loss: 0.0384, Val accuracy Score: 0.8938
Epoch 4/8: 100%|██████████| 625/625 [01:25<00:00,  7.31batch/s, accuracy_score=0.885, loss=0.0334]
Epoch [4/8], Train Loss: 0.0369, Train accuracy Score: 0.8850, Val Loss: 0.0379, Val accuracy Score: 0.9094
Epoch 5/8: 100%|██████████| 625/625 [01:50<00:00,  5.67batch/s, accuracy_score=0.895, loss=0.034]
Epoch [5/8], Train Loss: 0.0359, Train accuracy Score: 0.8954, Val Loss: 0.0367, Val accuracy Score: 0.9281
Epoch 6/8: 100%|██████████| 625/625 [01:29<00:00,  6.96batch/s, accuracy_score=0.904, loss=0.0357]
Epoch [6/8], Train Loss: 0.0351, Train accuracy Score: 0.9038, Val Loss: 0.0359, Val accuracy Score: 0.9125
Epoch 7/8: 100%|██████████| 625/625 [01:05<00:00,  9.49batch/s, accuracy_score=0.897, loss=0.0343]
Epoch [7/8], Train Loss: 0.0344, Train accuracy Score: 0.8971, Val Loss: 0.0364, Val accuracy Score: 0.9094
Epoch 8/8: 100%|██████████| 625/625 [01:07<00:00,  9.32batch/s, accuracy_score=0.894, loss=0.0317]
Epoch [8/8], Train Loss: 0.0338, Train accuracy Score: 0.8942, Val Loss: 0.0357, Val accuracy Score: 0.9156
```