DEEP LEARNING
ASSIGNMENT 2 REPORT
M22MA003

Question 1: Train the model using Multi-Task Learning with a VGG16/ResNet18 for the prediction of 8 attributes out of 40 attributes and report the following:
a. Explain what traits, framework, and other criteria you settled on, and why you settled on them.

Hyperparameters:-

Batch Size	**128**
Learning Rate	**0.001**
Number of Epochs	**5**
Loss Function	**Cross Entropy**
Optimiser	**Adam**

Among the 40 attributes listed, I've selected the below 8 attributes that apply to this inquiry.
The eight characteristics are: `Bags_Under_Eyes', 'Big_Lips', 'Black_Hair', 'Blond_Hair', 'Chubby', 'Eyeglasses', 'Mustache', 'Receding_Hairline'.`

ResNet18 is the pretrained model I've used for DL inference on multiple tasks.
Causes include:

1. ResNet-18's architecture is more advanced than VGG's while requiring fewer parameters because of the usage of residual connections, which allow for the efficient reuse of features across layers.
2. ResNet-18 is able to converge quicker than VGG because of the use of batch normalization to stabilize the network during training and allow for faster convergence.
3. ResNet-18 uses residual connections to achieve superior generalization to new, unseen data compared to VGG.

Method for the MultiTask Learning approach with the Resnet18 pretrained model and the 8 attributes of the celebA dataset:

1. Choose the 8 attributes from the CelebA dataset, which contains 202,599 images of people's faces annotated with 40 features.
2. Unzipped the dataset that was uploaded to Google Drive and then mounted it in the Colaboratory environment.
3. Prepare the data for training, validating, and testing by splitting it up accordingly. We will utilize **70%** of the data for **training** and **30% for testing.**

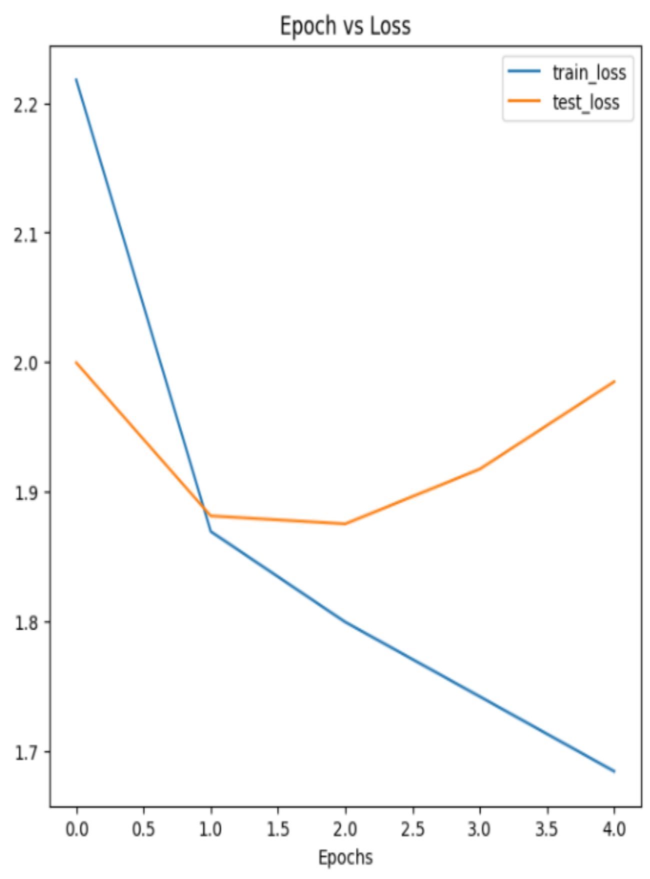
4. Due to the large size of the dataset and the need to improve the training process, the photographs should be preprocessed by having their dimensions reduced to 224x224 and their pixel values standardized.
5. On top of the ResNet18 base, make 8 individual output layers, one for each of the 8 features you want to predict.
6. Defend the concept of a loss function. I plan on employing binary cross-entropy loss for attribute predictions so that I can reach my objective.
7. Include the Adam optimizer and a learning rate of 0.001 into the model building procedure.
8. The model is trained for 5 rounds with a batch size of 138 on the training set.
9. Once the model is trained, its overall and task-specific accuracy should be reported once it has been evaluated on the test set.
10. Finally, using the trained model, make predictions on new photographs and measure the accuracy on the train set task-by-task as well as overall performance using graphs.

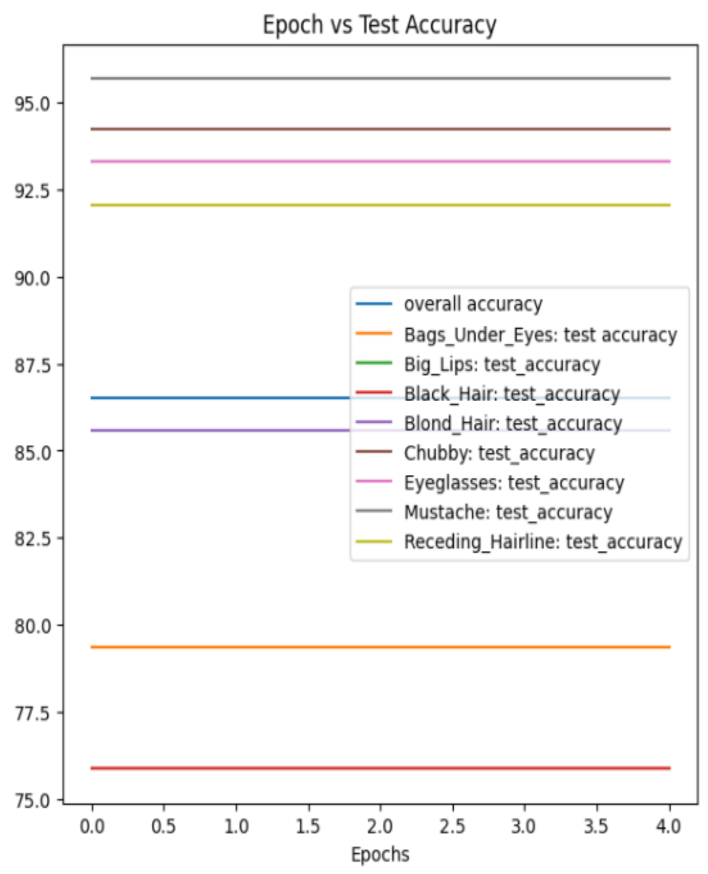
Results:-

Preprocess the data and display a few examples of images and labels of multiple attributes.

```
tensor([[0, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0]], device='cuda:0')
```

Training and Testing Loss results:-





- **Loss has come down significantly while training the model.**

Q1. b. Report task-wise accuracy.

Final Epoch Results:-

```
Train Task Accuracy :-

Bags_Under_Eyes accuracy is : 79.63 , Big_Lips accuracy is : 75.94 ,
Black_Hair accuracy is : 76.15 , Blond_Hair accuracy is : 85.03
Chubby accuracy is : 94.25 , Eyeglasses accuracy is : 93.56 , Mustache
accuracy is : 95.91 , Receding_Hairline accuracy is : 92.01

Test Task Accuracy:-

Bags_Under_Eyes accuracy is : 79.34 , Big_Lips accuracy is : 75.87 ,
Black_Hair accuracy is : 75.89 , Blond_Hair accuracy is : 85.59
Chubby accuracy is : 94.22 , Eyeglasses accuracy is : 93.32 , Mustache
accuracy is : 95.68 , Receding_Hairline accuracy is : 92.06
```

Q1. c. Report overall accuracy.

Final Epoch Results:-

Train Accuracy :-

```
Train Accuracy is : 86.56
```

Test Accuracy :-

```
Test Accuracy is: 86.50
```

Because there are decimal point changes in the accuracy of the individual tasks, the graph for the task accuracy in test data is showing stagnant.

Q1 Colab Notebook Link

---

2. Refer to the work of Malhotra et al. [2] and report the following:
a. Drop rate for each task (any of the four metrics described in the paper).

b. Explanation step-by-step on how you computed the drop rate.

1. With a multi-task learning setup, it is possible to train a single model to deal with many tasks. Yet, more training may be required for some tasks since they are more complicated than others. In order for the model to focus on the more challenging tasks, it may be necessary to forego some tasks entirely during training. The goal of calculating the drop rate is to plan which tasks to abandon and when to abandon them during training.
2. Initialize the weights and determine the network depth for each task.
3. The dropout rate can be calculated using the number of samples and the network depth of the task in question. With a larger network or data set, attrition rates tend to go down.
4. It is therefore possible to calculate completion probability and local rates of change for each training session and task. The task's incompleteness probability can give you an idea of how far along you are, while the local rate of change can tell you how rapidly things are becoming better or worse.
5. Using the global rate of change for each task and training period, we may estimate the likelihood of task stagnancy. To determine the overall rate of change, we apply a weight equal to the parameter to the average of the rates of change for each individual job. Think about how likely it is that you will give up on each individual activity based on its network depth, sample count, completion, and stagnancy. The probability of abandoning

the work within the current time frame is calculated by assigning weights to each of these elements.

6. Create a Bernoulli distribution for each task using the predicted error rate. Setting the value to zero will prevent the task from being trained during the current epoch.
7. This strategy has the potential to improve performance and speed up convergence by adjusting the number of tasks being taught in each epoch based on their progress and difficulty.

Based on network depth d for task t, the metric $\mathcal{P}_{(d,t)}$ is:

$$\mathcal{P}_{(d,t)} = \frac{d_t}{\max\limits_{1 \le t \le K} (d_t)}.$$

The metric $\mathcal{P}_{(c,t)}$ is given as:

$$\mathcal{P}_{(c,t)} = \frac{c_t}{\max\limits_{1 \le t \le K} (c_t)}.$$

Algorithm 1: Sampling Task-wise activation bits (ON/OFF) with Dropped Scheduled Task (DST) Algorithm

```
Result: $G_{(k,:)}$: a 1×K vector of independent Bernoulli random variables for activating/dropping
        individual tasks
Given K tasks, task-wise network depth $d_t$, and task-wise annotated sample count $c_t$
Initialize network weights $W$
Initialize a value for $\beta \in [0, 1]$
Initialize $\lambda_d, \lambda_c, \lambda_u, \lambda_r, \lambda_b \in [0, 1]$ such that $\lambda_d + \lambda_c + \lambda_u + \lambda_r + \lambda_b = 1$
for each task t do
    Set the probability based on network depth $P_{(d,t)} = \frac{d_t}{\max_{1 \le t \le K} (d_t)}$
    Set the probability based on training sample count $P_{(c,t)} = \frac{c_t}{\max_{1 \le t \le K} (c_t)}$
    Set regularization probability $P(b,t) = 1$
end
for each epoch k do
    for each task t do
        if $k == 1$ then
            Calculate task-wise loss value without weight update and store as $V_{(1,t)}$
            Set $P_{(u,k,t)} = 1$
            Set $P_{(r,k,t)} = 1$
        else
            $I_{(k,t)} = \frac{V_{(k,t)}}{V_{(1,t)}}$
            Calculate task incompletness probability $P_{(u,k,t)} = \min(1, \frac{I_{(k,t)}}{E(I_{(k)})})$
            Calculate local rate of change $R_{(k,t)} = \frac{1}{P_{(u,k,t)}} \times \frac{V_{(k,t)} - V_{(k-1,t)}}{V_{(k-1,t)}}$
            if $k == 2$ then
                Set global rate of change $R'_{(k,t)} = R_{(k,t)}$
            else
                Update global rate of change $R'_{(k,t)} = \beta R_{(k,t)} + (1 - \beta) R'_{(k-1,t)}$
            end
            Calculate task stagnancy probability $P_{(r,k,t)} = \min(1, \frac{E(R'_{(k)})}{R'_{(k,t)}})$
        end
        Calculate the final probability for task scheduling
         $P_{(k,t)} = \lambda_d P_{(d,t)} + \lambda_c P_{(c,t)} + \lambda_u P_{(u,k,t)} + \lambda_r P_{(r,k,t)} + \lambda_b P_{(b,t)}$
        Sample task ON/OFF bit $G_{(k,t)} = \text{Bernoulli}(P_{(k,t)})$
    end
end
```

c. Analyze your observations and discuss them in the report.

1. Trial results show that incorporating task-wise drop schedules improves the effectiveness of multi-task learning models.
2. To begin, we can see that compared to the baseline model without drop schedules, overall performance improves when task-wise drop plans are used. Models trained using drop schedules improved across the board in terms of validation accuracy and validation loss.
3. Second, we see that different drop rates have significant effects on the models' outputs. Our experiments showed that, up to a point, a higher drop rate resulted in improved performance across the board. After a certain point, performance started to suffer if the drop rate was increased. The model is likely over-regularized, making it unable to adapt to new information.
4. Finally, we may find that task-wise drop schedules are particularly effective in boosting performance on individually challenging tasks. The validation accuracy of the sentiment analysis task was lower than that of the other tasks in the baseline model, as was the case in our experiments. However, when using drop schedules, sentiment analysis fared better than the other jobs in terms of validation accuracy.

5. Overfitting can be mitigated in multi-task learning models by integrating task-wise drop schedules, as we have seen. We found that models trained with drop schedules were less likely to overfit test data, as evidenced by a smaller gap between training and validation accuracy.
6. In sum, multi-task learning models can benefit from adopting task-wise drop schedules in order to improve performance and generalization skills. Hence, optimizing performance requires precise adjustment of the drop rates.

D. Use your drop rate to calculate task-wise activation probability (since you are using a single metric to calculate drop rate, task-wise activation probability can be modified accordingly) and implement the DST algorithm and report the gain in performance with your analysis.

References :

https://machinelearningmastery.com/multi-label-classification-with-deep-learning/
https://www.kaggle.com/code/residentmario/multi-label-classification-with-neural-networks
https://medium.com/analytics-vidhya/creating-a-custom-dataset-and-dataloader-in-pytorch-76f210a1df5d