DL Assignment 3: Ritam Sharma
Multi-Task Learning and DST B20BB030

Multi-task Learning

Question Overview

The main goal is to train the model for the prediction of 8 attributes out of 40 attributes and Perform Attribute Prediction Task on CelebA dataset for the attributes available in the dataset.

Metrics

Accuracy

Accuracy is a commonly used metric in Multi Task Learning (MTL) to evaluate the performance of a model across multiple tasks. In MTL, the goal is to train a single model to perform multiple related tasks simultaneously.

When using accuracy as a metric in MTL, the model's performance is evaluated based on its ability to correctly classify examples across all tasks. For each task, the accuracy is calculated by dividing the number of correctly classified examples by the total number of examples in that task's test set. The overall accuracy of the model is then calculated by averaging the accuracies across all tasks.

One advantage of using accuracy as a metric in MTL is that it is easy to interpret and understand. It provides a clear measure of the model's ability to perform all tasks simultaneously, without the need to evaluate each task separately. Moreover, it is a widely used and well-understood metric, making it easier to compare different models and approaches.

However, accuracy as a metric in MTL also has some limitations. One limitation is that it assumes that all tasks are equally important, which may not always be the case. In addition, accuracy may not be the most appropriate metric for some tasks, especially if the classes are imbalanced or if the cost of misclassification varies across tasks. Therefore, it is important to consider other metrics in addition to accuracy when evaluating the performance of MTL models.

Analysis

Data Exploration

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than **200K** celebrity images, each with **40** attribute annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations, including

Methodology and Reasons for Backbone, Attributes selected

- ❖ Data preparation: To prepare the Celeba dataset to be loaded for PyTorch data loader, a custom PyTorch dataset class needs to be created. The class should inherit from the torch.utils.data.Dataset class and implement the following methods:
    - ➢ __init__: This method initializes the dataset and sets the file paths for the images.
    - ➢ __len__: This method returns the total number of images in the dataset.
    - ➢ __getitem__: This method loads and returns a single image and its corresponding label.
- ❖ Furthermore, since we had to choose only 8 attributes out of 40 attributes I used Mutual Information as a metric. When selecting a subset of attributes for a multi-task learning (MTL) model, it is essential to choose features that are informative for all the tasks involved. Mutual Information (MI) is a statistical measure that can help identify the most relevant features for a set of tasks.
  Mutual Information measures the amount of information shared by two random variables, such as the relationship between an input feature and an output label. In the context of feature selection, we can compute the mutual information between each input feature and all the output labels. This measure captures the degree of dependency between the input feature and each task's output label.

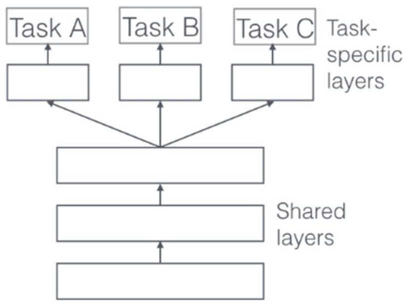
  By computing the mutual information between each input feature and each task, we can rank the features based on their relevance to all tasks. We can then choose the top-k features with the highest mutual information scores as the input features for the multi-task learning model.

  Using mutual information for feature selection in multi-task learning can be helpful because it takes into account the interdependence between features and tasks. It allows us to select features that are informative for multiple tasks simultaneously, which can improve the model's performance on all tasks. The following 8 features selected by this method are :

  1. Attractive
  2. Big_Nose
  3. Double_Chin
  4. Gray_Hair
  5. Heavy_Makeup
  6. Male
  7. No_Beard
  8. Wearing_Lipstick

- ❖ Split the data into training and validation sets : Separated the dataset into a training set for the model's training and a validation.

- ❖ Define the architecture: For the backbone in the model, I used ResNet. This is because of memory constraints. The celebA dataset is quite big. Furthermore, GPU utilization goes down as memory usage goes up, therefore it makes sense to use a smaller-sized model when we are using such a big data and have memory limitation on GoogleCollab, Even though ResNet is much deeper than VGG16 and VGG19, the model size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers



Furthermore, I have used Hard Parameter sharing method as demonstrated above for the Multi Task learning problem. Hard parameter sharing greatly reduces the risk of overfitting. In fact, It is shown that the risk of overfitting the shared parameters is an order N -- where N is the number of tasks -- smaller than overfitting the task-specific parameters, i.e. the output layers. This makes sense intuitively: The more tasks we are learning simultaneously, the more our model has to find a representation that captures all of the tasks and the less is our chance of overfitting on our original task.

- ❖ Train and Evaluate the model: Determine the model's accuracy and performance measures by assessing the model's performance on the test set once it has been trained.

- ❖ Refine the model: To enhance the model's performance, make minor adjustments to the architecture, hyperparameters, or other components. The training and evaluation process should be repeated until the desired performance level is reached.

Loss Function used : Binary Cross Entropy

Binary Cross-Entropy (BCE) loss is a commonly used loss function in machine learning and deep learning algorithms. It is used for binary classification problems where the output of the model is either 0 or 1.

The BCE loss measures the difference between the predicted probability of a sample belonging to class 1 and the actual label. The formula for BCE loss can be expressed as:

$$L(y, \hat{y}) = - [y*\log(\hat{y}) + (1-y)*\log(1-\hat{y})]$$

where y is the actual label (either 0 or 1) and ŷ is the predicted probability of the sample belonging to class 1.

The BCE loss penalizes the model more when it makes a wrong prediction with a high confidence than when it makes a wrong prediction with low confidence. This is because the loss is logarithmic, and the logarithmic function amplifies the difference between high and low probabilities.

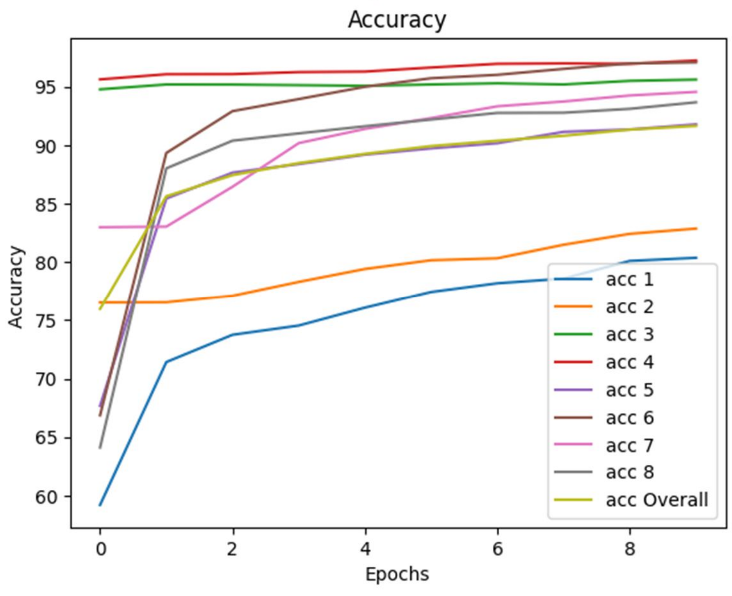
The BCE loss is commonly used in neural networks with a sigmoid activation function in the output layer. The sigmoid function maps the output of the network to a probability value between 0 and 1, making it suitable for binary classification problems.

Overall, the BCE loss is an effective loss function for binary classification problems, and it is widely used in deep learning algorithms.

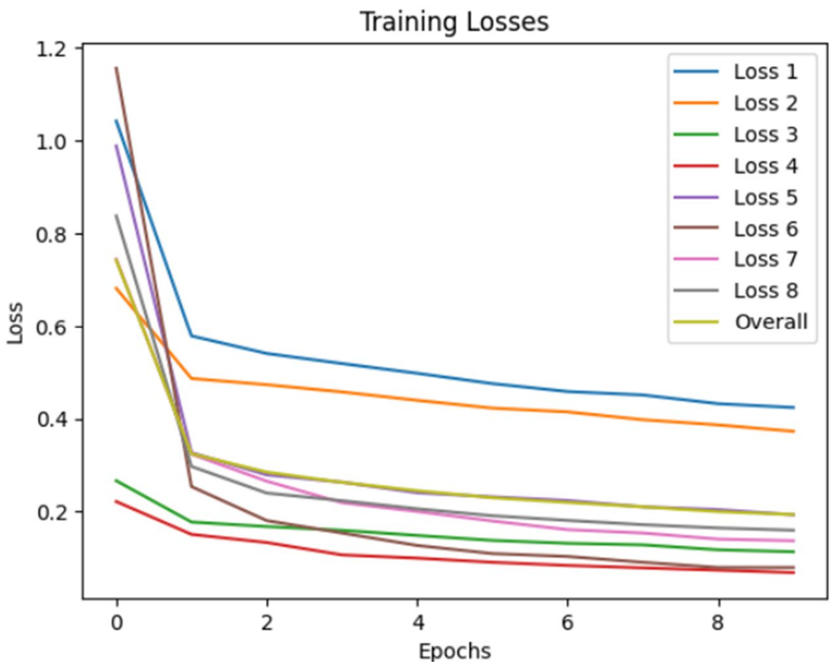
.

Results

Accuracy Curve



Loss Curves

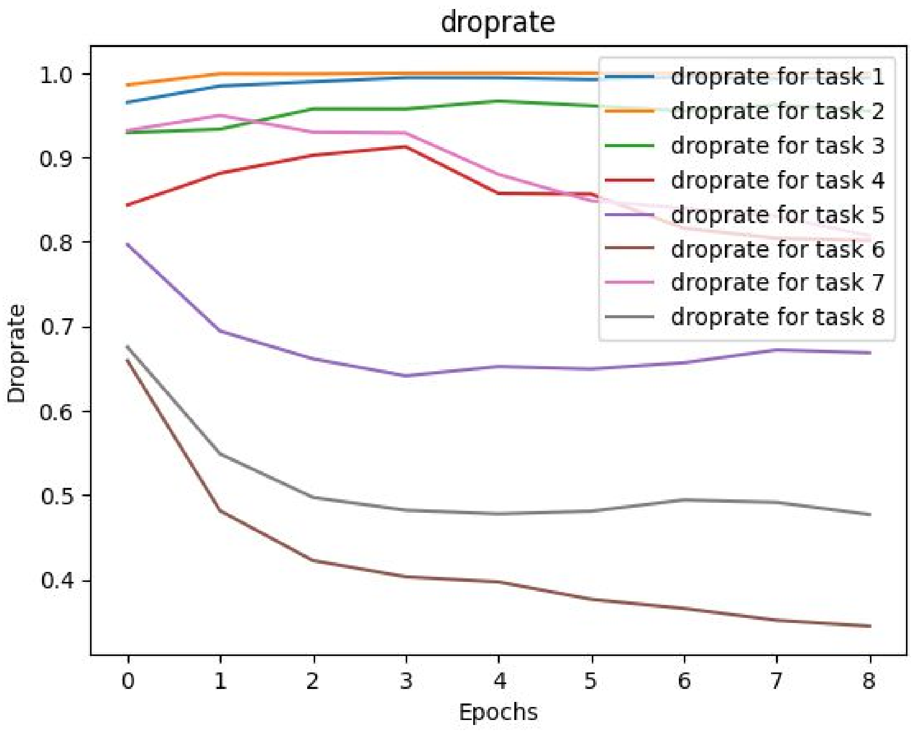


Task-wise accuracy and Overall Accuracy for the Validation dataset

Task	**Overall Loss**	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Task 6**	**Task 7**	**Task 8**
Loss	87.27	70.78	78.55	92.81	96.86	86.16	92.01	90.86	90.16

Bonus Question

Drop rate for each task [Result]



Step by Step explanation for computing Drop rate

Out of the Four metrics, the metric I chose for the calculation of the drop rate was *Metric based Task Incompleteness.* The proposed algorithm assumes that the losses (or errors) of different tasks can tell us how well each task is progressing. By comparing the current loss of a task to its initial loss, we can calculate how much progress the task has made so far.

Then, we calculate the expected value (or average) of this progress ratio across all tasks. This tells us how much progress we can expect all tasks to have made, on average.

Using this expected progress ratio, we can figure out which tasks are ahead or behind the average. If a task has a higher ratio, it means that it has made more progress than expected, and is ahead of schedule. If a task has a lower ratio, it means that it has made less progress than expected, and is behind schedule.

This helps us to balance the workload between tasks by identifying which ones need more attention and resources to catch up to the expected progress rate.

The pseudocode for calculating the drop rate is given below, I shall explain it in detail in the coming steps :

Pseudocode for finding drop rate

- Pb_t is the regularization metric
- V1_t is the loss of task t at epoch 1
- Vn_t is the loss of task t at epoch n
- In_t is the incompleteness of task t at epoch
- E(I(n)) is the expected value across all tasks at epoch n
- Gn_t is the independent Bernoulli random variables for activating/dropping for task t at epoch n

```
initialize a regularization metric to avoid catastrophic
forgetting
Pb_t=1
FOR n IN number_of_epochs:
IF (epoch==0):

   Calculate V1 for every task
   V1_t =loss at epoch 0 for a task t(Do this for all the tasks)

ELSE:

  Calculate Vn for every task

  Vn_t= loss at epoch n for a task t(Do this for all the tasks)

  Calculate, the amount of "incompleteness"
  In_t = Vn_t/In_t(Do this for all the tasks)

  Calculate expected value across all tasks
  E(I(n)) = mean([In_t for t in range(num_tasks)])

  Calculate relative incompleteness for every task t at epoch n
  Pn_t = MIN(1,In_t/E(I(n)))(Do this for all the tasks)

  Find the final probability
  Pn_t = lambdap*Pn_t + (1-lambdap)*Pb_t

  Gn_t=Bernoulli(Pn_t )
```

STEP 1:

The first step is to initialize the regularization metric to 1, this is done to prevent a task to completely remaining OFF, resulting in catastrophic forgetting, each task needs to get some chance to be active. To do so, a regularization metric

is given as Pb_t = 1.

STEP 2 :

The next step is to calculate V1_t, which is the loss of a task t at the first epoch. This is done to get the incompleteness ratio.

STEP 3:

Now, that we have V1_t, we will calculate Vn_t(loss of task t at epoch n) , by getting the ratio of these two, we will get out incompleteness ratio given by:

$$I_{(k,t)} = \frac{V_{(k,t)}}{V_{(1,t)}}.$$

(Note: here epoch 'n' is denoted by 'k')

For a task whose learning is incomplete with a negligible decrease in loss, the value for In_t would be close to 1. On the contrary, a task that has been completely learned would have a small value of 0.

Further, it could be a case where a task has digressed and has its Vn_t ≥ V1_t.This would result in In_t ≥ 1. Using In_t, the relative incompleteness of each task t can be found as:

$$\mathcal{P}_{(u,k,t)} = \min\left(1, \frac{I_{(k,t)}}{E(I_{(k)})}\right).$$

(Note: here epoch 'n' is denoted by 'k' and the drop rate by P(u,k,t))

E(I(n)) is the expected value of In_t across all tasks at the nth epoch. penalizes faster learning tasks but does not reward a slow task as the value is upper bound to 1. It makes all tasks that are "uncompleted" and slower than estimated E(I(n)) to obtain the highest activation chance while reducing the compute cycles of faster tasks.

STEP 4 (To implement DST from drop rate)

Now, from the drop rate, we will calculate task-wise activation probability since we are using a single metric to calculate drop rate, task-wise activation probability is modified accordingly. We will add regularization like it was done in the reference paper . Therefore, the final probability to schedule a particular task t at kth epoch is

$$P_{(k,t)} = \lambda_u P_{(u,k,t)} + \lambda_b P_{(b,t)}$$

Here, λ are non-negative weights given to individual metrics such that sum(λi) = 1. These λi can be altered to address specific variations of data, network, and learning in MTL setup, therefore we, have

$$1 = \lambda_u + \lambda_b$$

On combining the above two parameters, we get the following equation which is implemented in the code

$$P_{(k,t)} = \lambda_u P_{(u,k,t)} + (1 - \lambda_u) P_{(b,t)}$$

Now, Gn_t is sampled form Pn_t as:

$$G_{(k,t)} \sim \text{Bernoulli}(P_{(k,t)})$$

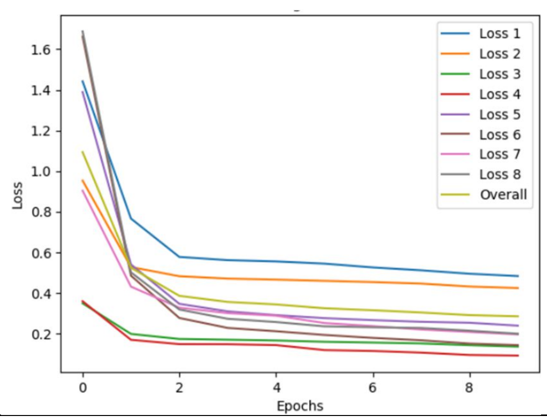
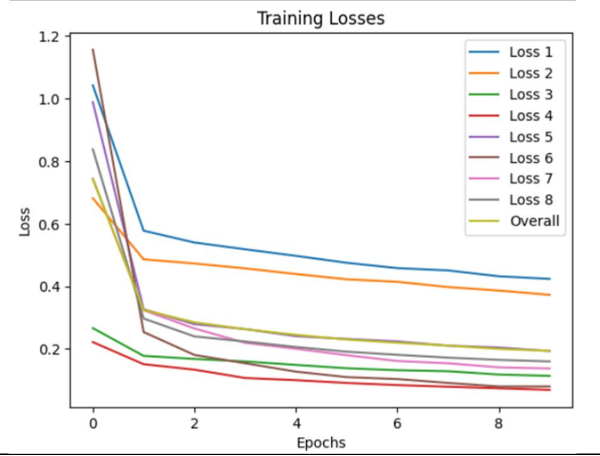
In this context, the Bernoulli distribution models a binary random variable G(k,t) that takes on two possible values: 1 (success) with probability P(k,t), and 0 (failure) with probability 1 - P(k,t).

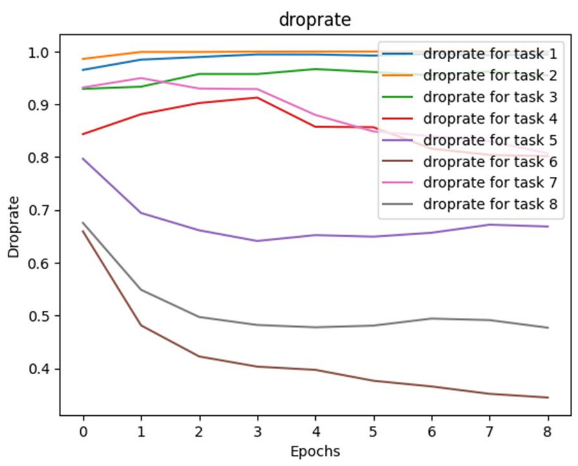
The notation G(k,t) ~ Bernoulli(P(k,t)) means that the random variable G(k,t) follows a Bernoulli distribution with success probability P(k,t). The notation "~" can be read as "is distributed as".

For example, if P(k,t) = 0.8, then G(k,t) ~ Bernoulli(0.8) means that the random variable G(k,t) takes on the value 1 with probability 0.8 and the value 0 with probability 0.2.

Gn_T denotes an ON/OFF bit for the t th task at n th epoch, then this is multiplied with its specific task loss before all the task losses are added to get the overall loss

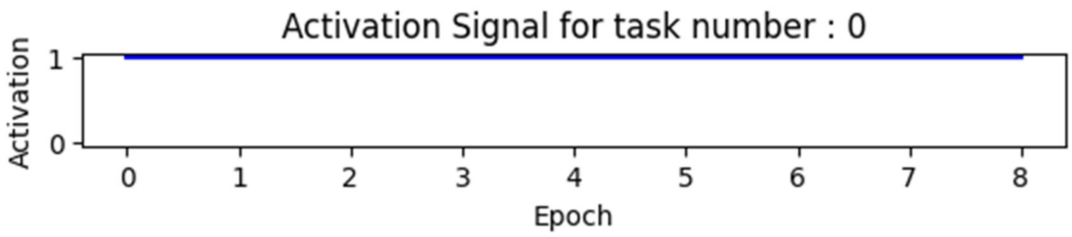
Result and analysis of Drop rates, Accuracies, Loss curve and activation

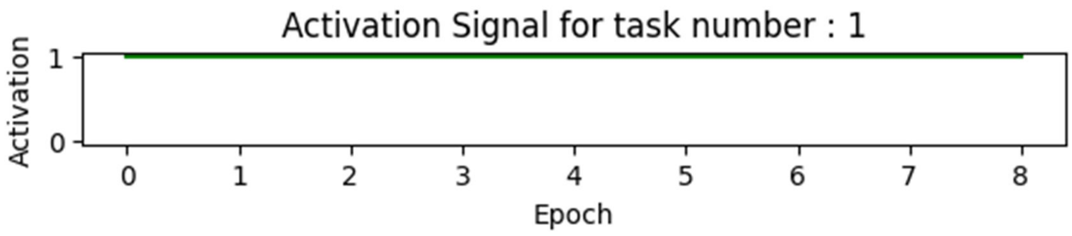
Loss with DST	Loss without DST
	

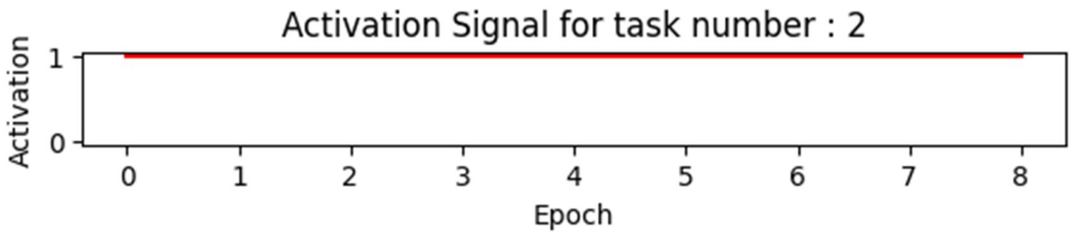


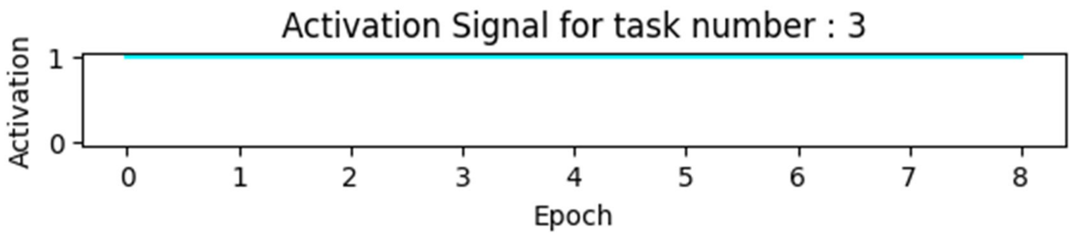
Analysis:

❖ For Task 1 and Task 2 we can notice in the plot for the loss curve without DST that these tasks have the least decrease in loss values as compared to other Tasks, therefore it is beneficial for their activation to be 1 so that their losses are accounted for . And in the plot for loss curves with DST implementation, we can see that these tasks have a much better loss curve, therefore, their activation is mostly ON

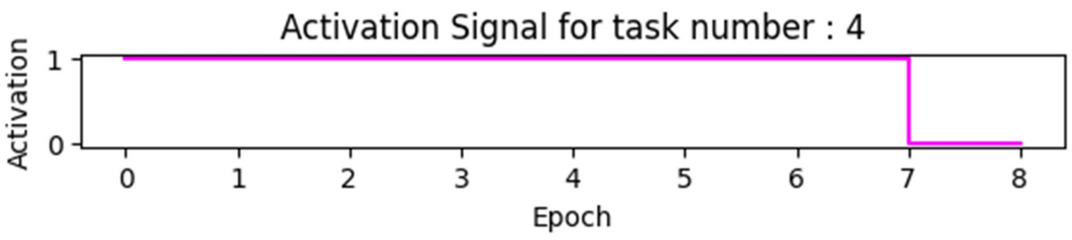


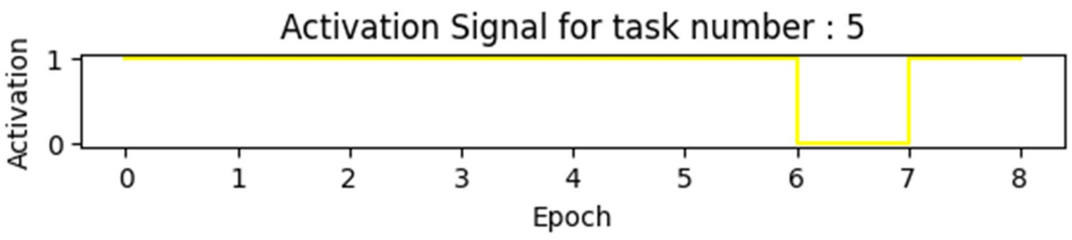


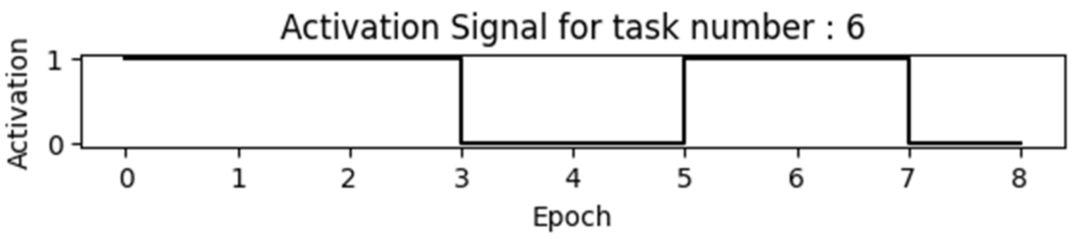


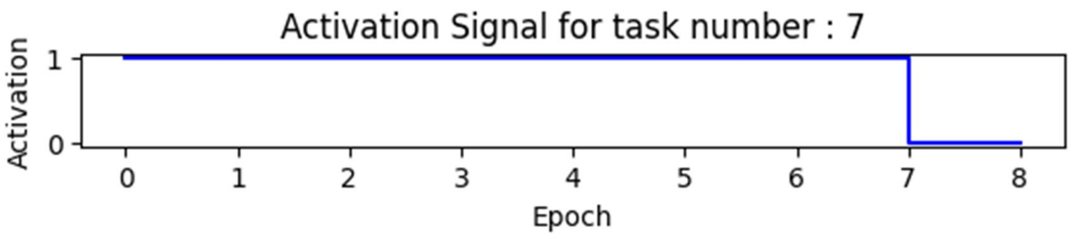


❖ Now unlike the above tasks we can observe that for task 5,6,7,8 , the loss curve relatively falls down faster and in a more steeper fashion. Therefore they have dropouts closer to 0 as compared to task 1,2,3,4.Therefore these tasks have more OFF values as we can see from the steps plots below



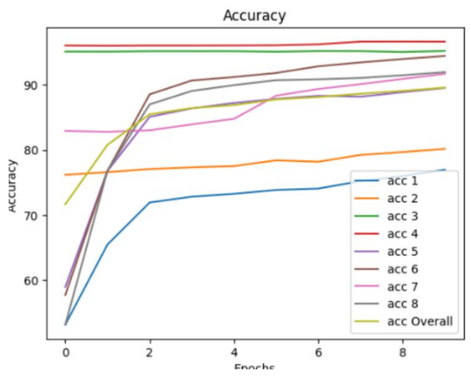






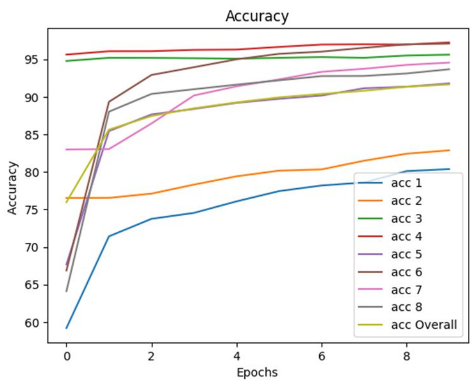
Accuracy Gain

With DST



Task	Overall Loss	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Value :	88.79	75.36	80.41	95.83	96.70	87.48	93.26	91.30	90.0

Without DST



Task	Overall Loss	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Loss	87.27	70.78	78.55	92.81	96.86	86.16	92.01	90.86	90.16

Besides Task 4 and Task 8, All the other Tasks had an increase in accuracy. The increase in accuracy was most for Task 1 which had the lowest Accuracy. Therefore Implementing DST with Task Incompleteness Metric Benefits the Learning process in MTL and reduced negative transfer

REFERENCES:

[1] LIU, ZIWEI, PING LUO, XIAOGANG WANG, AND XIAOOU TANG. "LARGE-SCALE CELEBFACES ATTRIBUTES

(CELEBA) DATASET." RETRIEVED AUGUST 15, NO. 2018 (2018): 11.

[2] MALHOTRA, AAKARSH, MAYANK VATSA, AND RICHA SINGH. "DROPPED SCHEDULED TASK: MITIGATING

NEGATIVE TRANSFER IN MULTI-TASK LEARNING USING DYNAMIC TASK DROPPING." TRANSACTIONS ON

MACHINE LEARNING RESEARCH.