Deep Learning

CSL7590

PROGRAMMING ASSIGNMENT - 3

REPORT

Submitted By:
Saurav Kumar
B20EE081
(Department of Electrical Engineering)

Submitted To:
Prof. Mayank Vatsa
(Department of Computer Science & Technology)

Question No 1:(CeleBa Dataset)

About dataset:

In this question, we imported all the important libraries and imported all the data of the celeba dataset, which is in a folder consisting of the celeba images and each image attributes like

```
Index(['5_o_Clock_Shadow', 'Arched_Eyebrows', 'Attractive', 'Bags_Under_Eyes',
       'Bald', 'Bangs', 'Big_Lips', 'Big_Nose', 'Black_Hair', 'Blond_Hair',
       'Blurry', 'Brown_Hair', 'Bushy_Eyebrows', 'Chubby', 'Double_Chin',
       'Eyeglasses', 'Goatee', 'Gray_Hair', 'Heavy_Makeup', 'High_Cheekbones',
       'Male', 'Mouth_Slightly_Open', 'Mustache', 'Narrow_Eyes', 'No_Beard',
       'Oval_Face', 'Pale_Skin', 'Pointy_Nose', 'Receding_Hairline',
       'Rosy_Cheeks', 'Sideburns', 'Smiling', 'Straight_Hair', 'Wavy_Hair',
       'Wearing_Earrings', 'Wearing_Hat', 'Wearing_Lipstick',
       'Wearing_Necklace', 'Wearing_Necktie', 'Young'],
      dtype='object')
```

and the length of the given is 202599 images, etc from these images, we have to classify the multitask classifications, in this, it is mentioned that we have to use the 8 features for the multitask classification using the vgg16 model. And we choose the attributes arched_eyebrows, big_lips, big_nose,black_hair,wearing_earings, and the wearing_hat, wearing_nelklace and the young attributes, because by plotting the heatmap which shows the relationship between the two attributes, and help to correlate the attribute which is more important attributes in from given data and we see that the attributes like:

`'Arched_Eyebrows','Big_Lips','Big_Nose','Black_Hair','Wearing_Earrings','Wearing_Hat','Wearing_Necklace','Young'` gives the best attributes.

dataset visualization:

For data visualisation, we first find the path of the images using the os.listdir inbuilt functions. Then we plot the images of the dataset using the PIL images.open(‘giving the location’) as seen below:

Also we subplot the random images from the given dataset to visualize the datasets clearly what type of different images is present based on their attributes mentioned using the matplotlib.pyplot libraries:

dataset Preprocessing:

After that, we created the dataset and implemented the class for creating the dataset and this class we passed the folder directory of images and the attributes data frame and transform which helps to normalize the images to improve the performance and training stability of the model, inside the class there are the two functions one which returns the length of the data and to other is which helps to create the dataset so that we can pass into the data loader and ready for the data to fit in the model, after the creating the

datasets, we split the dataset using the torch.utils.data.random_split (which helps to divide the dataset into the training and testing the data) in the ratio of 0.3 of the total data will be for testing the model and 0.7 of the total data for training the model.
After that we use the torch.data.utils.dataloader to sample mini-batches from a dataset, giving us the flexibility to choose from different sampling strategies for training data and testing data and taking the batch_size = 128 and shuffle = True, and for testing the whether the dataloader is formed correct or not we iterate it and over the trainloader, then we found the length of the trainloader and testloader.

```
length of the trainloader: 188
length of the testloader: 47
```

Model:

Here we created the multitask model, we first create the base model using torchvision.models.vgg16(), then after that the base model is passed into the class of the model which named as multiTaskModel along with number of classes we have to classify, then after the base_model, whatever the output came we pass into first hidden layer of neural network then we apply the relu activation function to bring the non-linearities in model then again we passed this output to second layer of neural network and then we return the output.
Inslide the vgg16() model there is sequential layer of convolution2D and relu, maxpool2d is made up of, we can visualize our model using summary(model).

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
MultiTaskModel                                     --
├─VGG: 1-1                                         --
│    └─Sequential: 2-1                             --
│    │    └─Conv2d: 3-1                            (1,792)
│    │    └─ReLU: 3-2                              --
│    │    └─Conv2d: 3-3                            (36,928)
│    │    └─ReLU: 3-4                              --
│    │    └─MaxPool2d: 3-5                         --
│    │    └─Conv2d: 3-6                            (73,856)
│    │    └─ReLU: 3-7                              --
│    │    └─Conv2d: 3-8                            (147,584)
│    │    └─ReLU: 3-9                              --
│    │    └─MaxPool2d: 3-10                        --
│    │    └─Conv2d: 3-11                           (295,168)
│    │    └─ReLU: 3-12                             --
│    │    └─Conv2d: 3-13                           (590,080)
│    │    └─ReLU: 3-14                             --
```

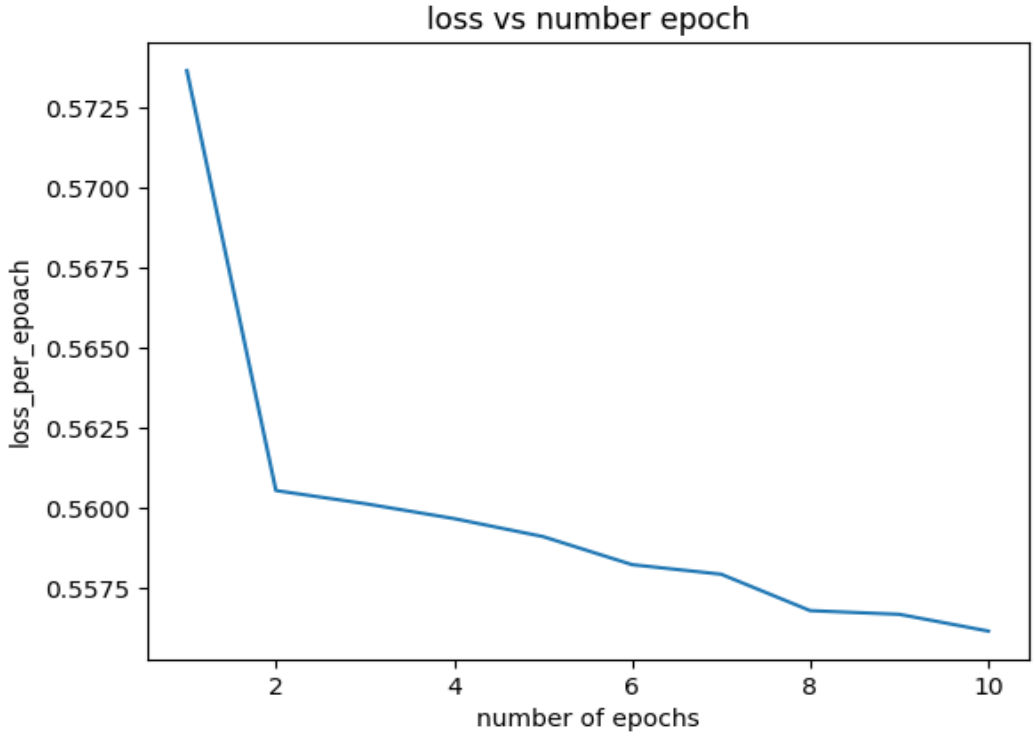
```
|    |    └─Conv2d: 3-15                  (590,080)
|    |    └─ReLU: 3-16                    --
|    |    └─MaxPool2d: 3-17               --
|    |    └─Conv2d: 3-18                  (1,180,160)
|    |    └─ReLU: 3-19                    --
|    |    └─Conv2d: 3-20                  (2,359,808)
|    |    └─ReLU: 3-21                    --
|    |    └─Conv2d: 3-22                  (2,359,808)
|    |    └─ReLU: 3-23                    --
|    |    └─MaxPool2d: 3-24               --
|    |    └─Conv2d: 3-25                  (2,359,808)
|    |    └─ReLU: 3-26                    --
|    |    └─Conv2d: 3-27                  (2,359,808)
|    |    └─ReLU: 3-28                    --
|    |    └─Conv2d: 3-29                  (2,359,808)
|    |    └─ReLU: 3-30                    --
|    |    └─MaxPool2d: 3-31               --
|    └─AdaptiveAvgPool2d: 2-2             --
|    └─Sequential: 2-3                    --
|    |    └─Linear: 3-32                  (102,764,544)
|    |    └─ReLU: 3-33                    --
|    |    └─Dropout: 3-34                 --
|    |    └─Linear: 3-35                  (16,781,312)
|    |    └─ReLU: 3-36                    --
|    |    └─Dropout: 3-37                 --
|    |    └─Linear: 3-38                  (4,097,000)
├─Linear: 1-2                            64,064
├─Linear: 1-3                            520
==========================================================================
Total params: 138,422,128
Trainable params: 64,584
Non-trainable params: 138,357,544
==========================================================================
```

Training and Testing :

Then we use the adam as optimization function and we use the crossentropyloss as loss function and then train the model, taking the number of epoch of 10, and we compute the loss and accuracy in each epochs.

```
Epoch: 1, Training Loss: 0.574, Training Accuracy: 52.884
Epoch: 2, Training Loss: 0.561, Training Accuracy: 87.177
Epoch: 3, Training Loss: 0.560, Training Accuracy: 92.296
Epoch: 4, Training Loss: 0.560, Training Accuracy: 78.003
Epoch: 5, Training Loss: 0.559, Training Accuracy: 84.781
Epoch: 6, Training Loss: 0.558, Training Accuracy: 84.931
Epoch: 7, Training Loss: 0.558, Training Accuracy: 93.281
```

```
Epoch: 8, Training Loss: 0.557, Training Accuracy: 79.908
Epoch: 9, Training Loss: 0.557, Training Accuracy: 94.158
Epoch: 10, Training Loss: 0.556, Training Accuracy: 90.986
```



After training the data on the trained model for multitask classification we get below:

```
Testing data: Average loss: 0.0044, testing Accuracy: 5361/6000 (0.8935)
```

After testing the data on the trained model for multitask classification we computed the individual attributes accuracies of the testing data, so to check which attributes plays the major roles in the celeba dataset, ie. what kind of images is majority present in celeba dataset, using the multitask classification, and we obtained the informationon the selected all eights attributes are given below:

```
Indiviual accuracy of the attributes are the given below on test_images:
([('Wearing_Necklace', 0.7535286357010456),
  ('image_id', 0.8767597160418144),
  ('Big_Lips', 0.8905061729067875),
  ('Arched_Eyebrows', 0.7019060036346726),
  ('Black_Hair', 0.72719582610218),
  ('Big_Nose', 0.9242120366354806),
  ('Wearing_Hat', 0.7345722414593183),
  ('Wearing_Earrings', 0.8265499450119461)]
```

Bonus Question:

Question 1

In dropped Scheduled task, A task t can continue to be active at epoch k in the proposed DST method with a probability P(k,t). If not, there is a likelihood that the jobs won't be completed (1 P(k,t)). A weighted mixture of five separate metrics, one of which is a regularizer, makes up P(k,t).

Metric based on Network Depth

The issue of vanishing gradients in deep neural networks during backpropagation, which might have an impact on multi-task learning (MTL) setups with tasks of various depths, is discussed in the paragraph. This issue has been addressed using methods like deeper networks, residual connections, or more loss terms, but these solutions only function well when tasks are at the same depth. The GradNorm mechanism has been developed to change gradients for quicker training while still providing tasks of various depths with an equal chance. The paper proposes utilising the measure P(d,t), which indicates the proportion of computing cycles allotted to task t depending on its network depth d, to allocate more computation cycles to deeper tasks based on their network depth. P(d,t) values that are higher mean that the task, here P(d,t) is:

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

Metric based on Training Sample Count

In this metrics discussed about the issues of overfitting and underfitting in deep neural networks during training are discussed, along with how they affect multi-task learning (MTL) configurations with tasks that require different quantities of labelled training data. Positive or negative transfer effects may result from the transfer of knowledge from tasks requiring more labelled samples to tasks requiring fewer samples. The suggested method involves calculating the number of calculation cycles for each task proportional to its ground-truth count in order to remedy this. This ensures that computation cycles are increased for tasks with more annotated training samples and decreased for jobs with fewer annotated training samples. The ratio of the considered task's ground-truth count to its maximum ground-truth availability is known as the measure P(c,t). A larger value indicates that the task has more labelled instances and thus more computing cycles. The

metric P(c,t) is defined as the ratio of the ground-truth count of the considered task t to the maximum ground-truth availability count of all tasks.

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

Metric based on Task being Stagnant

Some activities could stagnate in an MTL environment during training, at which point the rate of training loss declines to almost nothing. An easy, quick learning assignment that has undergone training may experience this. Further training on similar activities must to be avoided to prevent overfitting. On the other hand, some jobs might be overlooked because they are not given priority in MTL, and these tasks call for greater explicit emphasis and computational resources. A metric that measures both incompleteness and stagnation simultaneously is suggested as a way to empower incomplete stagnant tasks. By observing the drop in loss values over a few epochs, stagnancy can be measured for each task, and the local rate of change of losses may be derived using the loss values. the local rate of change of losses can be calculated using the loss value in the current and previous epoch. The local rate of change of loss R(k,t) for t th task at k th epoch can be expressed as

$$R_{(k,t)} = \frac{1}{\mathcal{P}_{(u,k,t)}} \times \frac{V_{(k,t)} - V_{(k-1,t)}}{V_{(k-1,t)}} \qquad \forall k \quad k \ge 2.$$

Metric based on Regularization

In multiclass classification, regularization methods are used to control the complexity of the model and avoid the model from fitting to noise in the training data. To prevent a task from 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 P(b,t) = 1.

Question 2:

For every t th task These four parameters are each quantified by a corresponding metric as part of the MTL task to determine the drop rate.

1. P(d,t) and P(c,t) calculate drop rates based on the number of training samples and the depth of the task, respectively.

2. P(u,k,t) and P(r,k,t) quantify the drop rate based on task stagnancy and incompleteness, respectively, at the kth epoch. Where P(u,k,t) is

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

Where,I(k,t) is the amount of “incompleteness” during training for task t at the k th epoch and E(I(k)) is the expected value of I(k,t),
And P(r,k,t) is

$$\mathcal{P}_{(r,k,t)} = \begin{cases} 1 & \text{if } k = 1; \\ \min\left(1, \frac{E(R'_{(k)})}{R'_{(k,t)}}\right) & \text{if } k \geq 2. \end{cases}$$

Where R(k,t) is the local change of loss.
These distinct measurements show us how powerful or underpowered a task is. Together, these metrics form the task-wise activation probability P(k,t), which has a range of [0, 1].

Question 3:

This paper majorly focused on the proposed DST algorithm and its performance in mitigating negative transfer in multi-task learning. compared the results against STL, MTL and random task drop, The experiments conducted in the paper demonstrate the effectiveness of the DST algorithm in mitigating negative transfer and improving overall multi-task learning performance. The results show that DST outperforms existing multi-task learning methods in terms of accuracy and stability, especially when dealing with imbalanced and diverse tasks. the results are also compared against other task prioritization or scheduling algorithms, we observe that negative transfer is a significant challenge in multi-task learning, and the proposed DST algorithm addresses this issue by dropping stagnant and poorly performing tasks dynamically. The task-dropping probability is determined by a metric that considers the task's depth, ground-truth count, and stagnancy. The experiments conducted in the paper demonstrate the effectiveness of the DST algorithm in mitigating negative transfer and improving overall multi-task learning performance. The results show that DST outperforms existing multi-task learning methods in terms of accuracy and stability, especially when dealing with imbalanced and diverse tasks. Also mentioned about the unilateral MTL faces, there were tasks including gender classification and wearing spectacles.

Question 4:

The Dropped Scheduled Task (DST) technique dynamically drops some tasks during training to reduce negative transfer in multi-task learning. To regulate the frequency of task drops, the algorithm uses a drop rate, which is the likelihood of dropping a task at each iteration.

To implement the DST algorithms, The task-wise activation probability can be changed based on the drop rate to perform the DST algorithm. Assume we have K tasks and p dropouts per task. The task-wise activation probability can therefore be determined as follows for each task i:

$$P(i) = (1 - p) * (n(i) / N)$$

where n(i) is the number of times task i has been dropped during training, and N is the total number of drops across all tasks. This formula ensures that tasks that have been dropped more frequently in the past will have a higher activation probability.

The tasks to include in each training iteration can be chosen using the task-wise activation probabilities that we have obtained. To be more precise, we can choose replacement tasks from the list of tasks based on their probability of activation and train the network for one iteration using those tasks. We carry out this process again, deleting tasks as necessary in accordance with the drop-off rate.
We can compare DST to a baseline strategy without task dropping to assess the performance increase. Both techniques can be used to train the same multi-task network, and we can compare the results' accuracy on a validation set. The impact of various drop rates on the effectiveness of the DST algorithm can also be examined. By lowering negative transfer and allowing each task to receive the right amount of training, the DST algorithm has the potential to enhance the performance of multi-task learning.