Deep Learning Report

DLOps Assignment - 4

Ayush Abrol

B20AI052

Question 01

Aim:

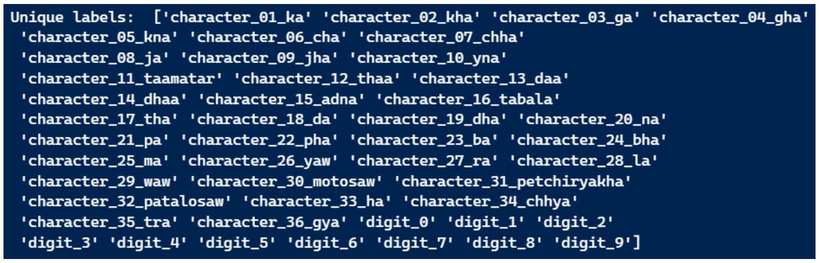
Train a Conditional Deep Convolutional Generative Adversarial Network (cDCGAN) on Dataset.

- Generate 50 Samples of each class from your trained generator.
- Train a ResNet18 classifier on the given dataset and treat the generated samples as test dataset and report following.
  - F1 Score for each class
  - Confusion matrix

Reference: https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/

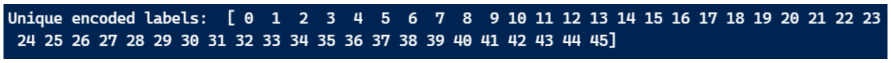
Procedure:

- First of all, I started the assignment by creating a new .ipynb file and doing all the necessary imports, such as:
  - numpy, pandas, matplotlib
  - torch (nn, nn.functional, optim, utils.data, autograd.Variable)
  - torchvision (transforms, utils, datasets, models)
  - tqdm for progress bar
  - PIL, copy, timeit
  - warnings
- Then, I read the GAN_data.csv file which contained the pixel-wise information for 92000 Hindi Language characters along with their labels.
- Total number of pixels = 1024 = 32x32
- Normalized and pre-processed the data and encoded the labels.



```
Unique labels:  ['character_01_ka' 'character_02_kha' 'character_03_ga' 'character_04_gha'
 'character_05_kna' 'character_06_cha' 'character_07_chha'
 'character_08_ja' 'character_09_jha' 'character_10_yna'
 'character_11_taamatar' 'character_12_thaa' 'character_13_daa'
 'character_14_dhaa' 'character_15_adna' 'character_16_tabala'
 'character_17_tha' 'character_18_da' 'character_19_dha' 'character_20_na'
 'character_21_pa' 'character_22_pha' 'character_23_ba' 'character_24_bha'
 'character_25_ma' 'character_26_yaw' 'character_27_ra' 'character_28_la'
 'character_29_waw' 'character_30_motosaw' 'character_31_petchiryakha'
 'character_32_patalosaw' 'character_33_ha' 'character_34_chhya'
 'character_35_tra' 'character_36_gya' 'digit_0' 'digit_1' 'digit_2'
 'digit_3' 'digit_4' 'digit_5' 'digit_6' 'digit_7' 'digit_8' 'digit_9']
```

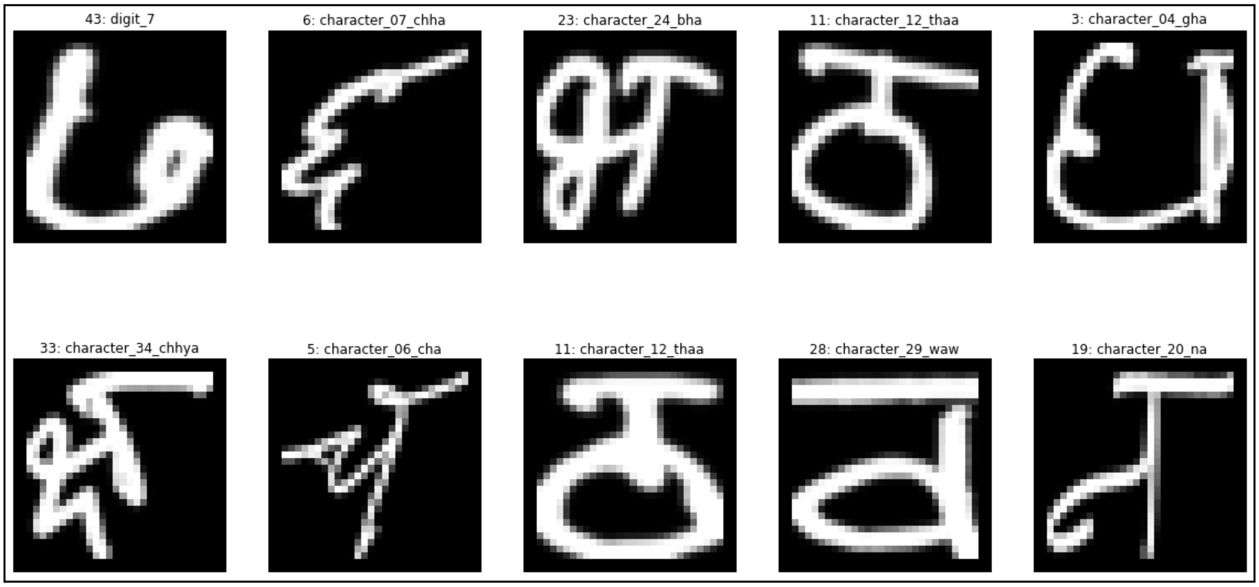
Before encoding the labels



```
Unique encoded labels:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45]
```

After encoding the labels

- Then, I shuffled the dataset.
- Visualization of some random samples from the dataset.



- Then, I created an ImageDataset class that takes in pixel data and labels. It has methods to get the length of the dataset and retrieve an item at a given index. The method to retrieve an item returns a tuple with two tensors: one tensor is the pixel data for an image and the other tensor is the corresponding label for that image.
- Then, created a PyTorch DataLoader of the dataset obtained with batch_size = 128 on a set of 92000 images.
- Then, I set my device to a GPU environment.

GANs

- In the context of generative adversarial networks (GANs), the generator and discriminator are two components that are trained together to generate realistic data.

- The generator is a neural network model that takes as input a random noise vector and outputs a sample of synthetic data. The goal of the generator is to generate synthetic data that is realistic enough to fool the discriminator.
- The discriminator, on the other hand, is a neural network model that takes as input a sample of data, either real or synthetic, and outputs a probability that the sample is real. The goal of the discriminator is to distinguish between real and synthetic data.
- During training, the generator generates synthetic data and passes it to the discriminator along with real data. The discriminator then evaluates the probability that the data is real. The generator uses the feedback from the discriminator to update its weights in order to generate synthetic data that is more likely to be classified as real. Meanwhile, the discriminator is also updated to become better at distinguishing between real and synthetic data.
- Through this process of iterative training, the generator and discriminator are trained to work together, with the generator generating increasingly realistic synthetic data and the discriminator becoming increasingly better at distinguishing between real and synthetic data. Ultimately, the goal is to achieve a balance where the generator is generating data that is realistic enough to fool the discriminator, and the discriminator is unable to distinguish between real and synthetic data.

The Generator Class

- It takes in a latent noise vector, a class label, and outputs an image. It has the following attributes:
    - **n_classes**: the number of classes in the dataset.
    - **embedding_dim**: the dimension of the embedding for each class.
    - **latent_dim**: the dimension of the noise vector.

  The module has three main components:

    1. **label_conditioned_generator**: a sequential model that takes the class label as input, passes it through an embedding layer, two linear layers with leaky ReLU activation functions, and outputs a 8x8 feature map.

2. **latent**: a sequential model that takes the noise vector as input and passes it through a linear layer with a leaky ReLU activation function, and outputs a 8x8 feature map with 256 channels.
3. **model**: a sequential model that takes the concatenated output of label_conditioned_generator and latent as input, and passes it through two transposed convolutional layers with batch normalization and ReLU activation functions, and outputs an image tensor with pixel values in the range [-1, 1] using a tanh activation function.

- The forward() method takes inputs of the noise vector and class label, passes them through the latent and label_conditioned_generator models, concatenates their outputs, and passes the concatenated tensor through the final model to generate an image tensor.

The Discriminator Class

- The discriminator takes as input an image and its corresponding label, and outputs a scalar value between 0 and 1, indicating the probability that the image is real (i.e., taken from the training data) or fake (i.e., generated by the generator).
- The discriminator has two main parts: a label conditioning component and a convolutional neural network component.
- The label conditioning component is a neural network that takes as input the label and generates an embedding that is the same size as the input image. This embedding is then concatenated with the input image to produce a tensor that is passed to the convolutional neural network component.
- The convolutional neural network component consists of several layers of convolution, batch normalization, leaky ReLU activation, and dropout, followed by a fully connected layer with a sigmoid activation function. This component is designed to process the concatenated image and label tensor and produce a scalar value that indicates the probability that the image is real.
- During training, the discriminator is trained to maximize the probability of correctly classifying real and fake images, while the generator is trained to minimize the probability of the discriminator correctly classifying its generated images as fake.

The ultimate goal is to achieve a balance where the generator produces images that are indistinguishable from real images, as determined by the discriminator.

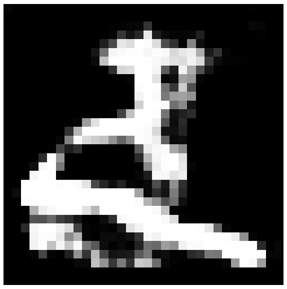
- Then, I instantiated the generator and discriminator objects with:
  - Latent_dim = 100
  - N_classes = 46
  - embedding _dim = 16
- Defined Adam optimizer for both of them with:
  - Learning_rates = 0.0002
  - Betas = (0.5, 0.999)
- Defined generator loss function which uses binaryCrossEntropy loss between the fake_outputs and the labels.
- Defined the discriminator loss function which uses binaryCrossEntropy loss between the outputs and the labels.
- Set num_epochs = 268 because of computation issues.

Training for the GAN

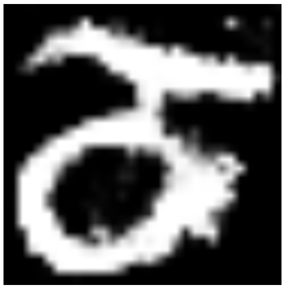
- It begins by looping through each epoch and then looping through each batch in the training data.
- For each batch, it first sets the gradients to zero for the Discriminator model and then moves the real images and labels to the device. It then creates a real target variable of ones and a fake target variable of zeros for use in calculating the Discriminator loss.
- The Discriminator is then passed the real images and labels to calculate the loss using the discriminator_loss function. Next, noise is generated from a normal distribution and used to create fake images with the generator model, which are passed to the Discriminator. The output is then used to calculate the Discriminator loss.
- The total Discriminator loss is calculated as the average of the real and fake losses, and the gradients are back propagated through the Discriminator model.

- The gradients are then set to zero for the Generator model, and the Discriminator loss is calculated again using the generated images as input to the Discriminator. The Generator loss is then calculated using the discriminator_loss function with the output from the Discriminator as input and the real target variable.
- The gradients are then back propagated through the Generator model, and the Discriminator and Generator losses are appended to their respective lists.
- If the index is divisible by 100, the losses are printed. After each epoch, the total time elapsed is printed, and if the epoch number is divisible by 5, the Generator and Discriminator models are saved, and a fake image is generated and saved.
- The loop ends when all epochs have been completed.
- Saved the trained model.

- I continuously saved the images generated along the training process after 2 epochs.



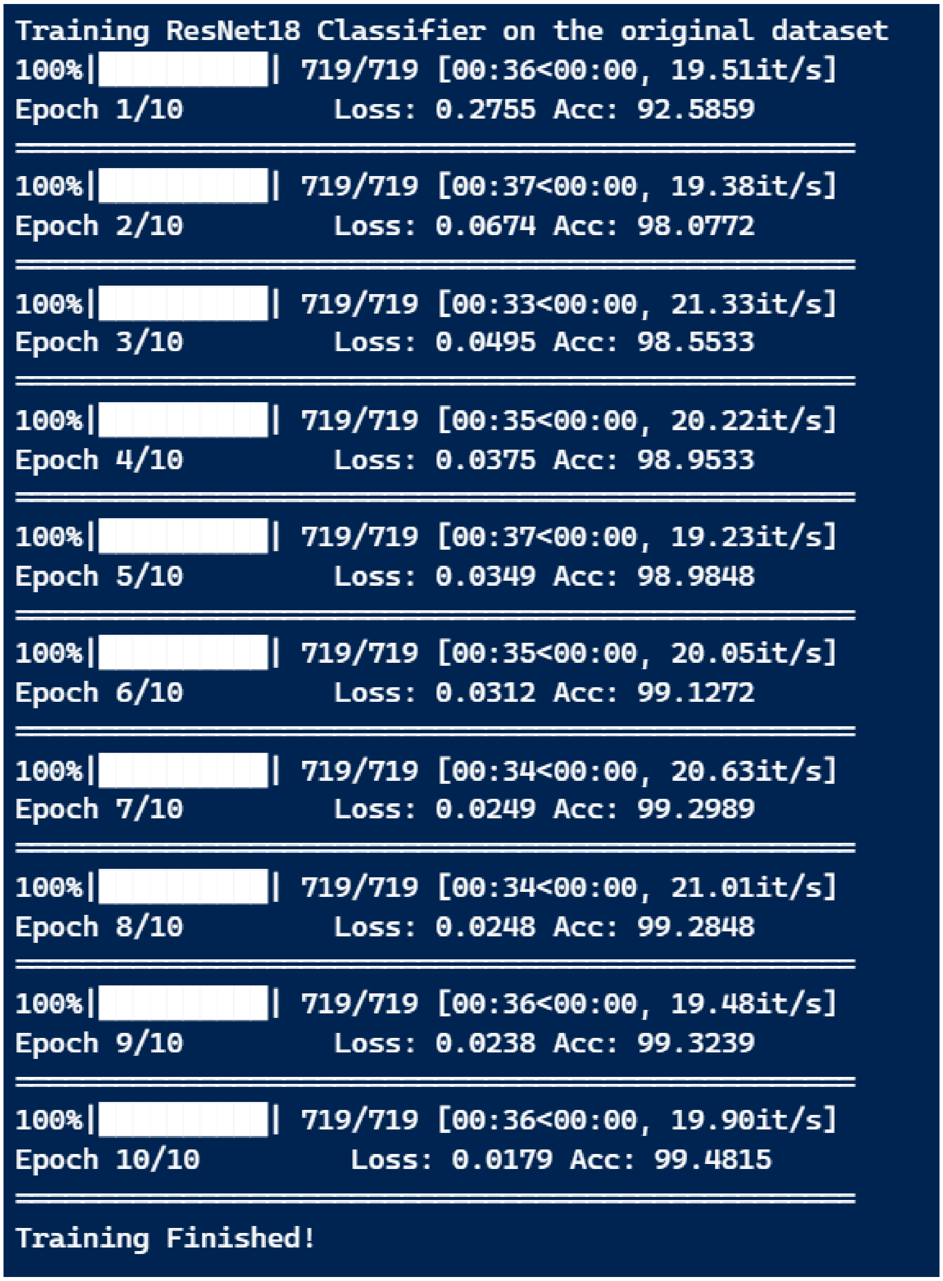
After 2 epochs



After 268 epochs

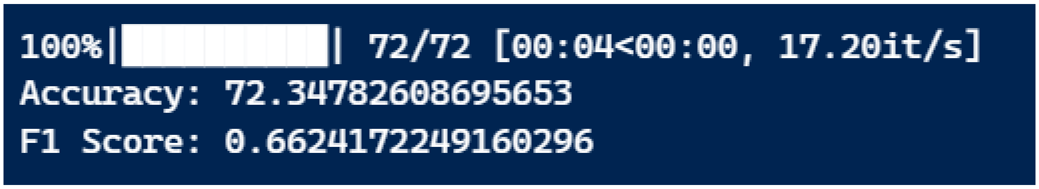
- Then, I generated 50 samples for each class and saved it in a directory which contained subdirectories named as their class label. (Images submitted in the zip file).

- Then, I imported a pretrained ResNet-18 model from torchvision.modes and fine-tuned it for 46 classes.
- Defined the optimizer as Adam with lr=3e-4 and loss function as CrossEntropyLoss for the model.
- Trained it for 10 epochs.



```
Training ResNet18 Classifier on the original dataset
100%|██████████| 719/719 [00:36<00:00, 19.51it/s]
Epoch 1/10         Loss: 0.2755 Acc: 92.5859
==================================================
100%|██████████| 719/719 [00:37<00:00, 19.38it/s]
Epoch 2/10         Loss: 0.0674 Acc: 98.0772
==================================================
100%|██████████| 719/719 [00:33<00:00, 21.33it/s]
Epoch 3/10         Loss: 0.0495 Acc: 98.5533
==================================================
100%|██████████| 719/719 [00:35<00:00, 20.22it/s]
Epoch 4/10         Loss: 0.0375 Acc: 98.9533
==================================================
100%|██████████| 719/719 [00:37<00:00, 19.23it/s]
Epoch 5/10         Loss: 0.0349 Acc: 98.9848
==================================================
100%|██████████| 719/719 [00:35<00:00, 20.05it/s]
Epoch 6/10         Loss: 0.0312 Acc: 99.1272
==================================================
100%|██████████| 719/719 [00:34<00:00, 20.63it/s]
Epoch 7/10         Loss: 0.0249 Acc: 99.2989
==================================================
100%|██████████| 719/719 [00:34<00:00, 21.01it/s]
Epoch 8/10         Loss: 0.0248 Acc: 99.2848
==================================================
100%|██████████| 719/719 [00:36<00:00, 19.48it/s]
Epoch 9/10         Loss: 0.0238 Acc: 99.3239
==================================================
100%|██████████| 719/719 [00:36<00:00, 19.90it/s]
Epoch 10/10         Loss: 0.0179 Acc: 99.4815
==================================================
Training Finished!
```

- Created a test dataloader from the generated images. Set batch_size=32.
- Tested the resnet18 model on this testloader and obtained the following results:



```
100%|██████████| 72/72 [00:04<00:00, 17.20it/s]
Accuracy: 72.34782608695653
F1 Score: 0.6624172249160296
```

Question 02

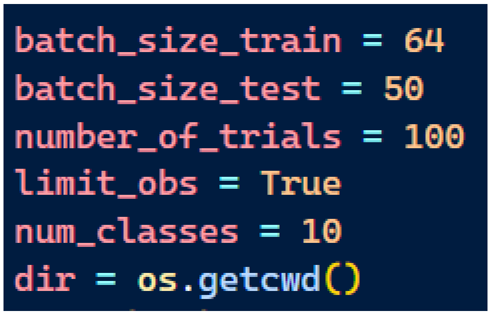
Aim:

- Train a CNN based classification model and perform Optimized Hyperparameter Tuning using Optuna Library on the below-mentioned dataset. Perform 100 trials.
- Hyperparameters should be
  - No of Convolution Layers 3 to 6
  - Number of Epochs 10 to 50
  - Learning rate 0.0001 to 0.1
  - Report the observations and the best trial. Report how many trials were pruned

For Even Roll Number B20AI052 - Dataset: MNIST

Procedure:

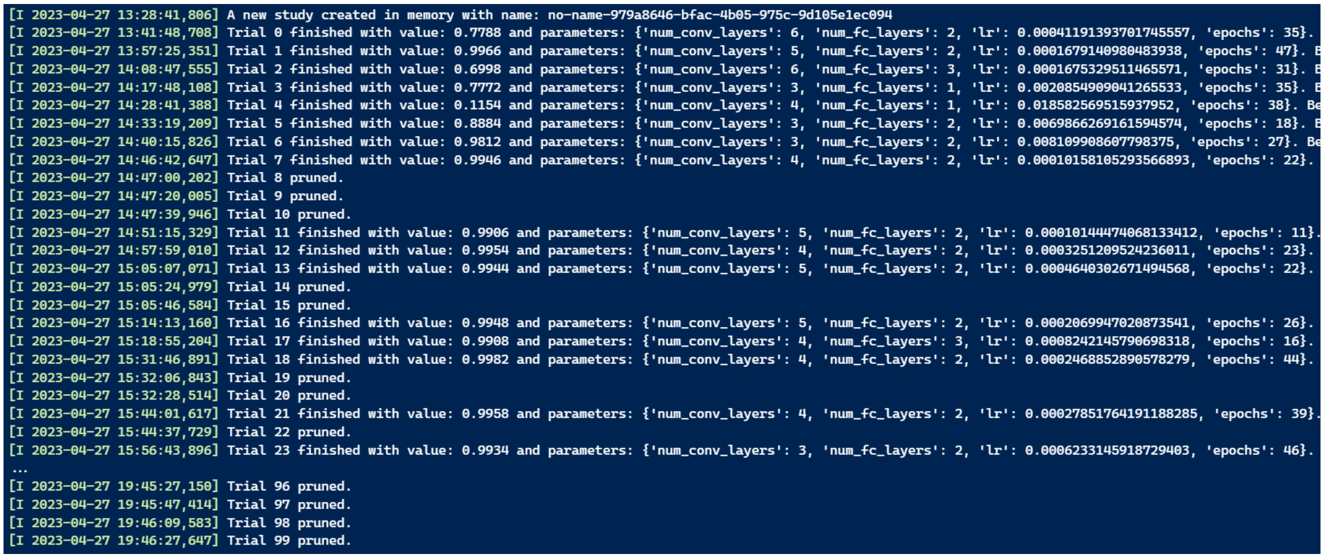
- Imported necessary libraries other than the libraries used in Question-01 such as:
  - os
  - optuna
  - TrialState from optuna.trial
- Set up the configuration for training:



```
batch_size_train = 64
batch_size_test = 50
number_of_trials = 100
limit_obs = True
num_classes = 10
dir = os.getcwd()
```

- Then, created a function get_data_mnist which returns the train_loader and test_loader with the given batch_size.
- Then, defined a class CNN such that:
  - The constructor __init__ defines the architecture of the network. It takes in three arguments: trial, num_conv_layers, and num_fc_layers.

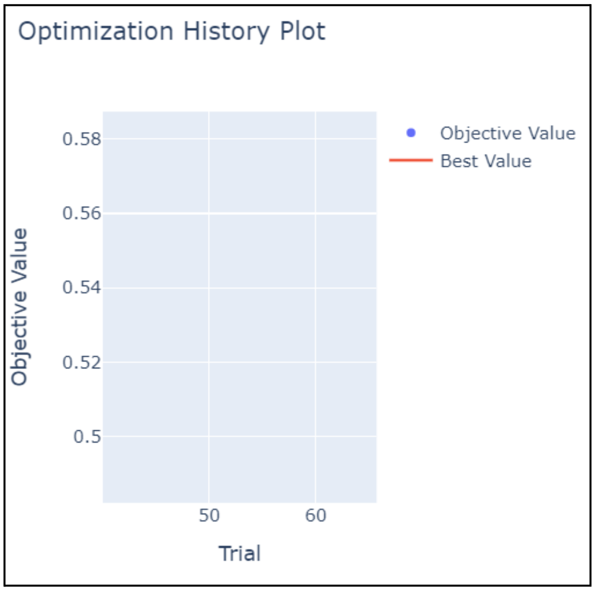
  - The num_conv_layers argument specifies the number of convolutional layers in the network, while the num_fc_layers argument specifies the number of fully connected layers in the network.
  - The conv_layers attribute is a ModuleList that contains the convolutional layers of the network. Each convolutional layer is defined by a nn.Conv2d module with the specified number of input and output channels, kernel size, stride, and padding.
  - The fc_layers attribute is another ModuleList that contains the fully connected layers of the network. Each fully connected layer is defined by a nn.Linear module with the specified number of input and output features.
  - The forward method defines the forward pass of the network. It takes in an input tensor x and passes it through each convolutional layer followed by a ReLU activation function. The resulting tensor is then flattened and passed through each fully connected layer followed by a ReLU activation function. The final output is returned.
- Defined a get_cnn function where:
  - num_conv_layers is set as trial.suggest() from 3 to 6 as given in the question.
  - num_fc_layers is set as trial.suggest() from 1 to 2.
- Then, I defined an objective function such that this function will be optimized by Optuna. The function takes a trial object as input, which is used to generate different sets of hyperparameters for the CNN model.
  - Inside the function, a CNN model is initialized using the hyperparameters generated by trial. Then, the function trains the model on the MNIST dataset and evaluates its performance on a separate test set. During training, the model is optimized using the Adam optimizer with a learning rate sampled from a log-uniform distribution.
  - After each epoch, the accuracy of the model on the test set is computed and reported back to Optuna using trial.report(). If the trial is found to be unpromising based on intermediate results, trial.should_prune() returns True, and the trial is pruned. Otherwise, the function returns the final accuracy of the model on the test set.
- Then, created an optuna study with direction set as “maximise”.
- Optimized the objective function.



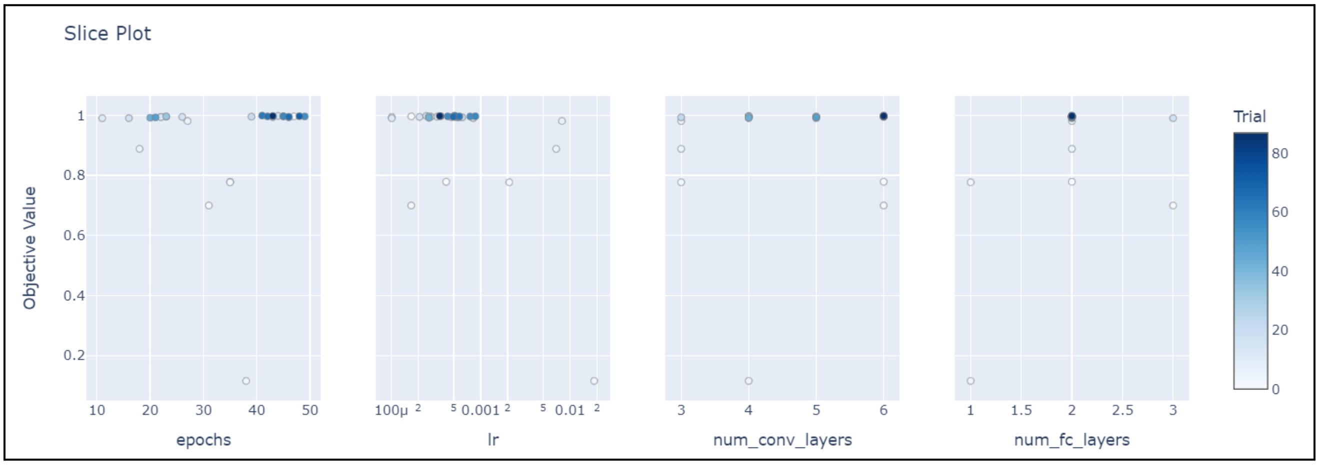
```
[I 2023-04-27 13:28:41,806] A new study created in memory with name: no-name-979a8646-bfac-4b05-975c-9d105e1ec094
[I 2023-04-27 13:41:48,708] Trial 0 finished with value: 0.7788 and parameters: {'num_conv_layers': 6, 'num_fc_layers': 2, 'lr': 0.00041191393701745557, 'epochs': 35}.
[I 2023-04-27 13:57:25,351] Trial 1 finished with value: 0.9966 and parameters: {'num_conv_layers': 5, 'num_fc_layers': 2, 'lr': 0.0001679140980483938, 'epochs': 47}. B
[I 2023-04-27 14:08:47,555] Trial 2 finished with value: 0.6998 and parameters: {'num_conv_layers': 6, 'num_fc_layers': 3, 'lr': 0.0001675329511465571, 'epochs': 31}. B
[I 2023-04-27 14:17:48,108] Trial 3 finished with value: 0.7772 and parameters: {'num_conv_layers': 3, 'num_fc_layers': 1, 'lr': 0.0020854909041265533, 'epochs': 35}. B
[I 2023-04-27 14:28:41,388] Trial 4 finished with value: 0.1154 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 1, 'lr': 0.018582569515937952, 'epochs': 38}. Be
[I 2023-04-27 14:33:19,209] Trial 5 finished with value: 0.8884 and parameters: {'num_conv_layers': 3, 'num_fc_layers': 2, 'lr': 0.0069866269161594574, 'epochs': 18}. B
[I 2023-04-27 14:40:15,826] Trial 6 finished with value: 0.9812 and parameters: {'num_conv_layers': 3, 'num_fc_layers': 2, 'lr': 0.008109908607798375, 'epochs': 27}. Be
[I 2023-04-27 14:46:42,647] Trial 7 finished with value: 0.9946 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 2, 'lr': 0.00010158105293566893, 'epochs': 22}.
[I 2023-04-27 14:47:00,202] Trial 8 pruned.
[I 2023-04-27 14:47:20,005] Trial 9 pruned.
[I 2023-04-27 14:47:39,946] Trial 10 pruned.
[I 2023-04-27 14:51:15,329] Trial 11 finished with value: 0.9906 and parameters: {'num_conv_layers': 5, 'num_fc_layers': 2, 'lr': 0.00010144474068133412, 'epochs': 11}.
[I 2023-04-27 14:57:59,010] Trial 12 finished with value: 0.9954 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 2, 'lr': 0.0003251209524236011, 'epochs': 23}.
[I 2023-04-27 15:05:07,071] Trial 13 finished with value: 0.9944 and parameters: {'num_conv_layers': 5, 'num_fc_layers': 2, 'lr': 0.0004640302671494568, 'epochs': 22}.
[I 2023-04-27 15:05:24,979] Trial 14 pruned.
[I 2023-04-27 15:05:46,584] Trial 15 pruned.
[I 2023-04-27 15:14:13,160] Trial 16 finished with value: 0.9948 and parameters: {'num_conv_layers': 5, 'num_fc_layers': 2, 'lr': 0.0002069947020873541, 'epochs': 26}.
[I 2023-04-27 15:18:55,204] Trial 17 finished with value: 0.9908 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 3, 'lr': 0.0008242145790698318, 'epochs': 16}.
[I 2023-04-27 15:31:46,891] Trial 18 finished with value: 0.9982 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 2, 'lr': 0.0002468852890578279, 'epochs': 44}.
[I 2023-04-27 15:32:06,843] Trial 19 pruned.
[I 2023-04-27 15:32:28,514] Trial 20 pruned.
[I 2023-04-27 15:44:01,617] Trial 21 finished with value: 0.9958 and parameters: {'num_conv_layers': 4, 'num_fc_layers': 2, 'lr': 0.00027851764191188285, 'epochs': 39}.
[I 2023-04-27 15:44:37,729] Trial 22 pruned.
[I 2023-04-27 15:56:43,896] Trial 23 finished with value: 0.9934 and parameters: {'num_conv_layers': 3, 'num_fc_layers': 2, 'lr': 0.0006233145918729403, 'epochs': 46}.
 ...
[I 2023-04-27 19:45:27,150] Trial 96 pruned.
[I 2023-04-27 19:45:47,414] Trial 97 pruned.
[I 2023-04-27 19:46:09,583] Trial 98 pruned.
[I 2023-04-27 19:46:27,647] Trial 99 pruned.
```

- It took 376 minutes to do this hyperparameter tuning.

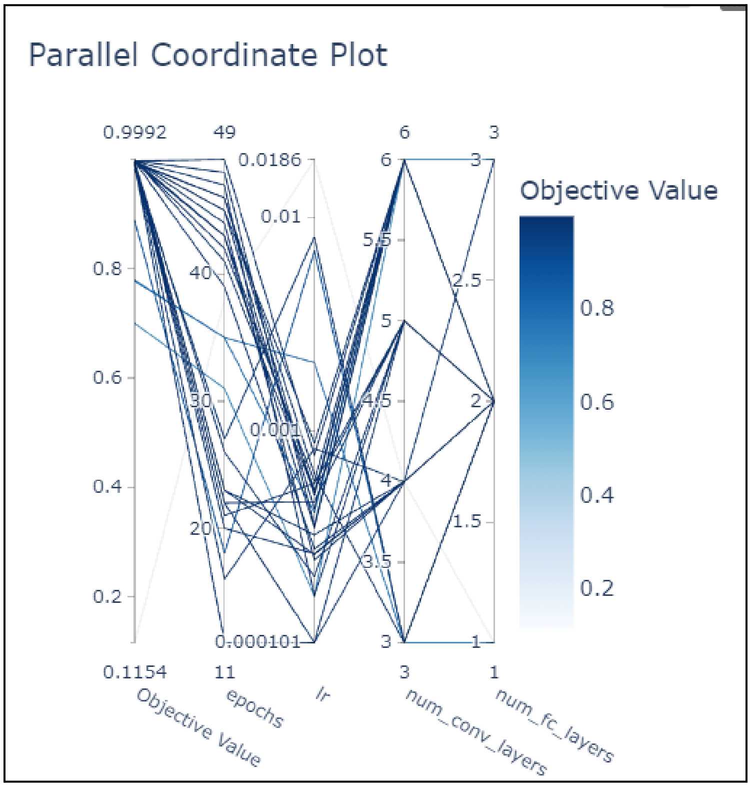
- Plot for the optimization history of all the trials in the study



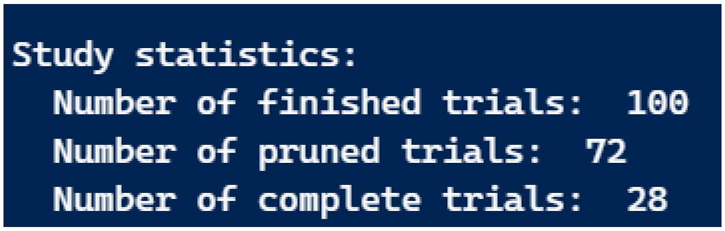
- Slice Plot



- Parallel Coordinate Plot

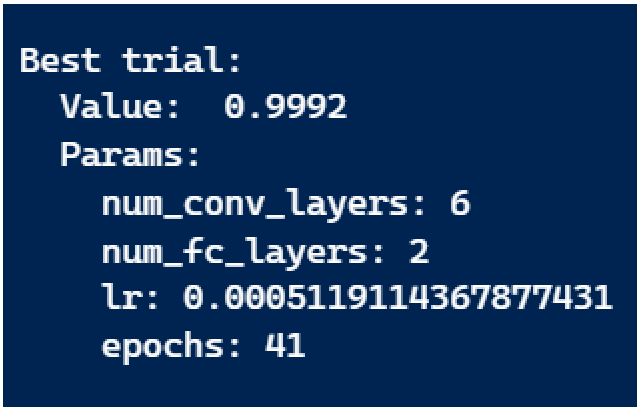


- Study Statistics Obtained were:



```
Study statistics:
  Number of finished trials:  100
  Number of pruned trials:  72
  Number of complete trials:  28
```

- Best Trial was:



```
Best trial:
  Value:  0.9992
  Params:
    num_conv_layers: 6
    num_fc_layers: 2
    lr: 0.0005119114367877431
    epochs: 41
```

- At last, I saved the best trial.

END OF ASSIGNMENT