DL-Ops Lab Assignment-8 Report

Task:
Note: All the files are uploaded in the zipped folder, so there is no need to rerun the task.

Objectives:

Question 1.

●

Procedure:

- **Main.py file**
- Imports the necessary libraries for data loading, transforms, model training, and evaluation.
- Defines the device to be used for training, whether GPU (cuda) or CPU.
- Defines the data transforms for the train and test datasets. It applies trivial data augmentation to the train dataset, such as random horizontal flipping, rotation, and color jittering, followed by converting the data to tensor.
- Creates the train and test datasets using the FashonMNIST dataset from the torchvision library, with the specified transforms.
- Creates a subset of the train and test datasets, containing only images that belong to the odd classes (1, 3, 5, 7, 9), by applying masks on the original datasets.
- Converts the train and test subsets to PyTorch data loaders with the specified batch size, number of workers for loading data, and shuffling (train only).
- Defines the Resnet-18 model architecture, early stopping criterion, loss function, accuracy function, and optimizer.
- Sets the hyperparameters (learning rate, betas, epsilon, weight decay, and the number of epochs) to try using a grid search approach.
- Iterates over all the hyperparameter combinations, initializing the model and optimizer with the current hyperparameters.
- Starts a new Weights and Biases run to track the current hyperparameters and model performance.
- Trains the model on the training subset using the current hyperparameters, using the train() function from the engine module. This function iterates over the batches of data, computes the forward and backward passes, updates the model parameters, and computes the loss and accuracy metrics.
- Evaluates the model performance on the test subset after each epoch using the evaluate() function from the engine module. This function iterates over the batches of data, computes the forward pass, and aggregates the predictions and ground truth labels to compute the accuracy metric.
- Logs the current hyperparameters, loss, and accuracy values to the Weights and Biases dashboard.
- Stops the training process if the validation loss does not improve after a certain number of epochs (early stopping).
- Saves the best model checkpoint based on the validation accuracy.

- **Engine.py file**
- log_image_table: This function takes images, predicted, labels, and probs as input. It creates a wandb table to log the images, labels, and predictions to. The function logs the table to the dashboard.
- validate_model: This function takes the model, valid_dl, loss_func, device, log_images, and batch_idx as input. It computes the performance of the model on the validation dataset and logs a wandb table. It returns the validation loss and accuracy.
- training_step: This function takes the model, dataloader, loss_fn, acc_fn, optimizer, device, and profiler as input. It performs the training step for one epoch. It returns the training loss and training accuracy.
- testing_step: This function takes the model, dataloader, loss_fn, acc_fn, and device as input. It performs the testing step for one epoch. It returns the testing loss and testing accuracy.
- The code imports the required libraries such as torch, torchmetrics, wandb, and nn (from torch). The wandb library is used for logging the metrics to the dashboard. The torchmetrics library is used to compute the accuracy of the model. The nn library is used to define the neural network model.

- **Models.py file**
- import torch: imports the PyTorch library, used for building and training deep learning models.
- from torch import nn: imports the neural network module nn from PyTorch, used for defining neural network layers and models.
- import numpy as np: imports the NumPy library, used for working with arrays and matrices.
- from torchvision.models import resnet18: imports the ResNet-18 architecture from the torchvision.models module, which provides pre-trained models for computer vision tasks.
- def get_resnet_18():: defines a function named get_resnet_18() that returns a ResNet-18 model with a linear classifier output layer with 10 classes.
- model = resnet18(): creates a new instance of the ResNet-18 architecture.
- model.fc = nn.Linear(in_features=512,out_features=10,bias=True): replaces the final fully connected layer of the ResNet-18 model (model.fc) with a new linear layer (nn.Linear) that maps from the ResNet-18 output features (512) to the number of classes (10).
- return model: returns the modified ResNet-18 model.
- class EarlyStopping:: defines a class named EarlyStopping that implements an early stopping criterion for training a neural network.
- def __init__(self, tolerance=5, min_delta=0):: defines an initializer method for the EarlyStopping class that sets the tolerance and minimum delta values for early stopping.
- self.tolerance = tolerance: sets the tolerance for the difference between validation and training loss.
- self.min_delta = min_delta: sets the minimum improvement in validation loss required to continue training.

- self.counter = 0: initializes a counter for the number of epochs with no improvement in validation loss.
- self.early_stop = False: initializes a boolean flag for whether to stop training early.
- def __call__(self, train_loss, validation_loss):: defines a callable method for the EarlyStopping class that compares the training and validation loss and updates the early stopping flag if necessary.
- if (validation_loss - train_loss) > self.min_delta:: checks if the difference between validation and training loss is greater than the minimum improvement required.
- self.counter +=1: increments the counter for epochs with no improvement in validation loss.
- if self.counter >= self.tolerance:: checks if the counter exceeds the tolerance for epochs with no improvement.
- self.early_stop = True: sets the early stopping flag to True if the tolerance is exceeded.

Results:

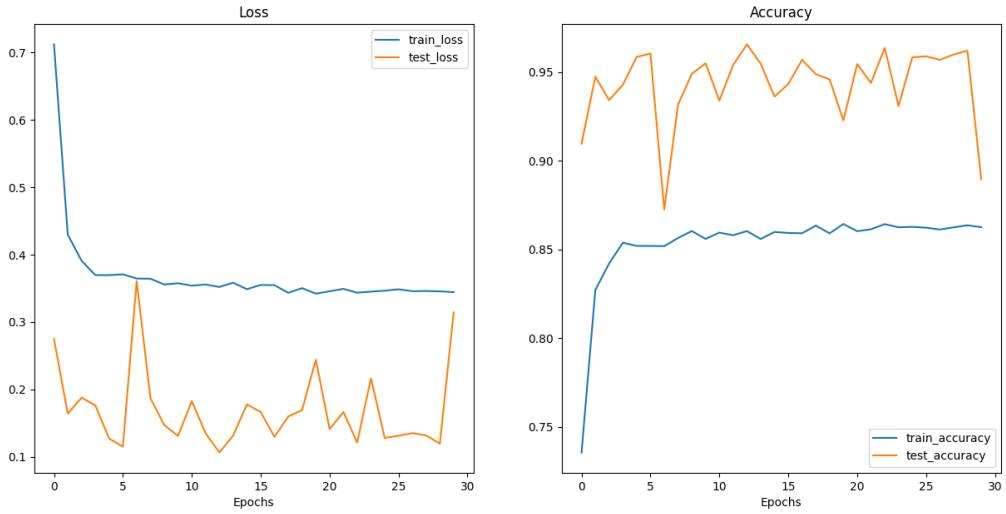

Fig: ResNet_epoch_30_optim_adam_lr_0.01_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.001

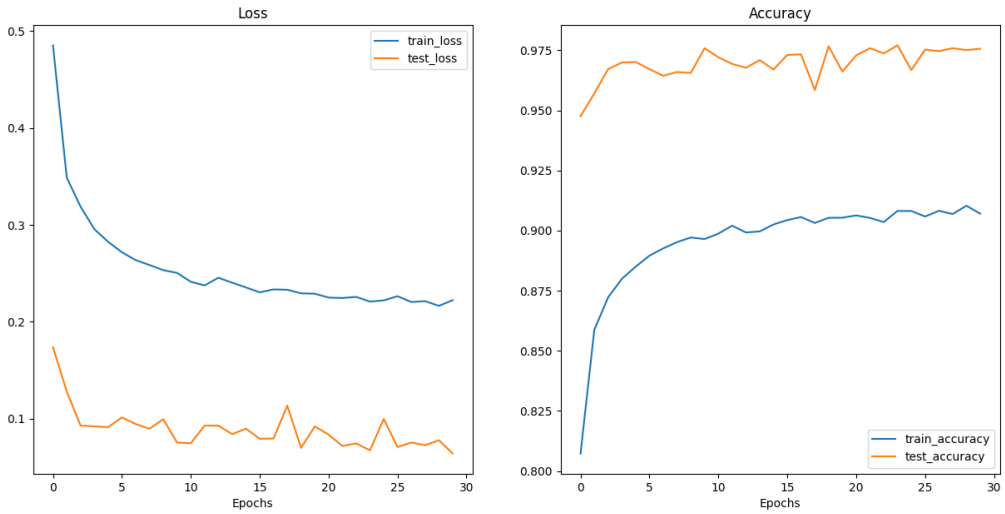

Fig: ResNet_epoch_30_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.001

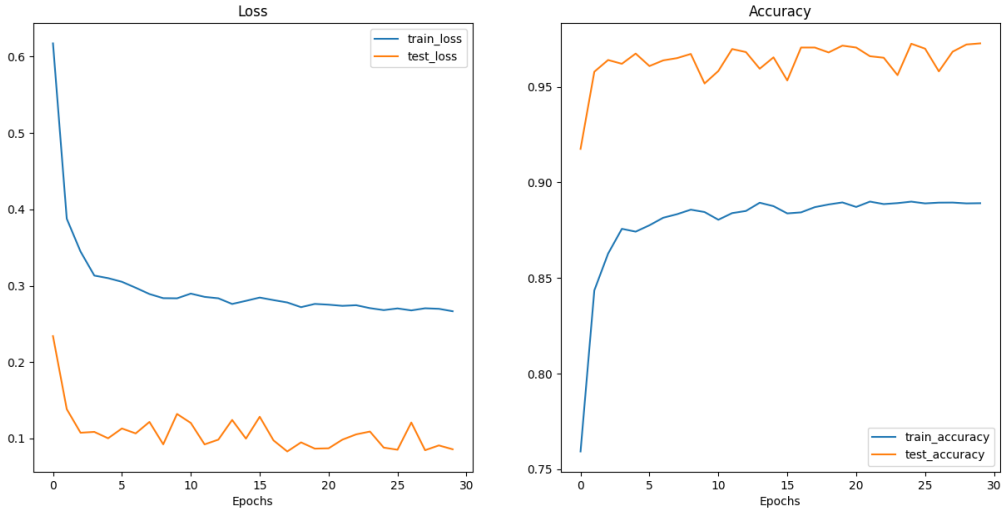

Fig: ResNet_epoch_30_optim_adam_lr_0.01_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

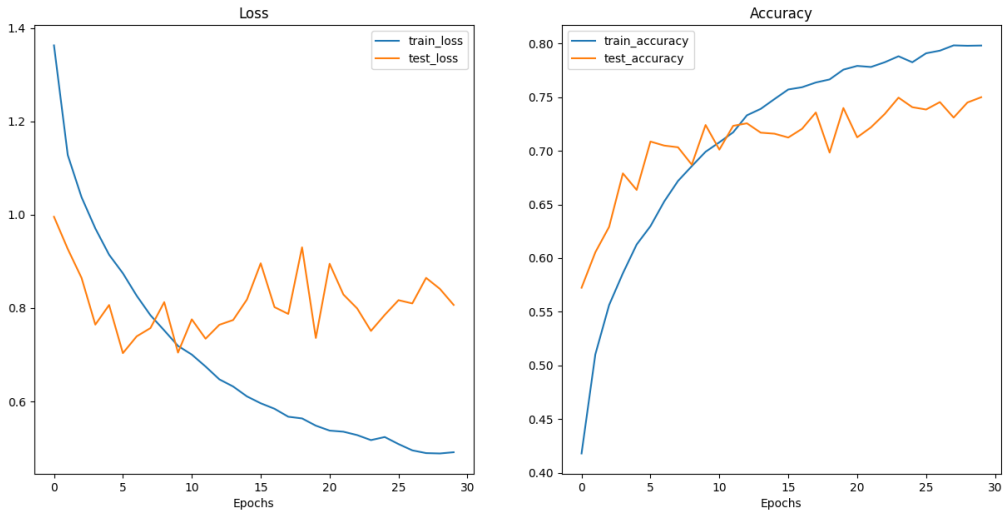

Fig: ResNet_epoch_30_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.001

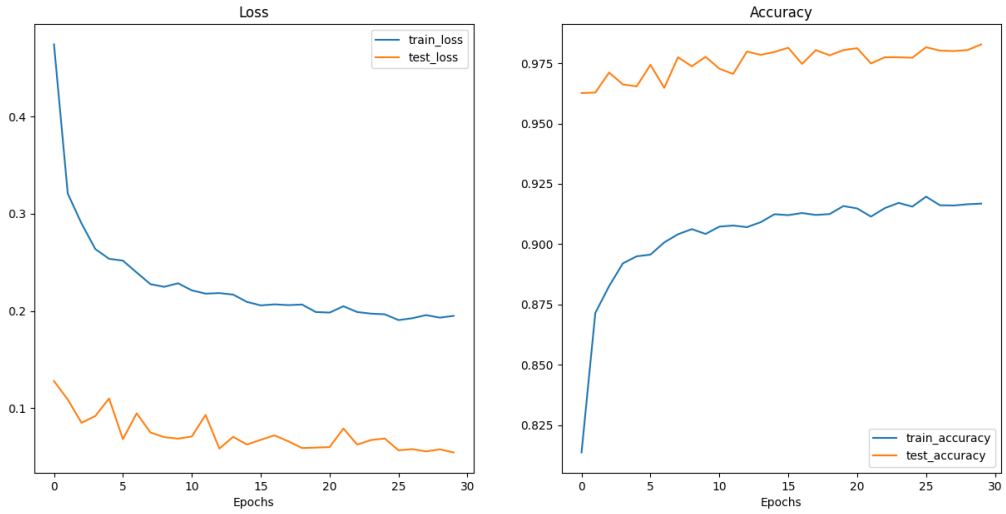

Fig: ResNet_epoch_30_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

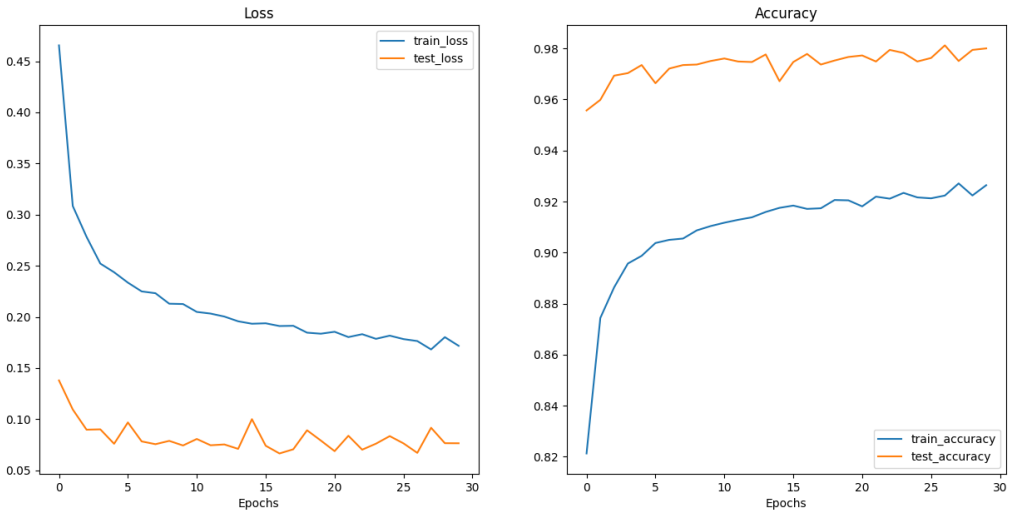

Fig: ResNet_epoch_30_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

WandB Results:

- Wandb Table
- Test losses for all runs
- Test Accuracy for all runs
- Train losses for all runs
- Train Accuracy for all runs
- Process GPU Power Usage (W)
- Process GPU Temp (℃)
- Process GPU Utilization (%)

Explanation of the Results:
The reason behind the above performance:-

- Random Initialization: When the ResNet-18 model is randomly initialized, the weights of the model are set randomly. As a result, the model is not optimized to perform well on the given tasks, and the accuracy is lower than what can be achieved with a pre-trained model.

- Limited Data: The model was trained on a limited dataset, which can cause overfitting or underfitting on the given tasks. Some tasks have high accuracy, indicating that the model is able to learn the relevant features from the limited data. In contrast, others have lower accuracy, indicating that the model was not able to learn enough information to perform well on those tasks.

- The complexity of the Tasks: The tasks with higher accuracies may be more straightforward in nature, and the ResNet-18 model with random initialization may have been able to learn the relevant features from the limited data. More complex tasks may require more training data or a pre-trained model for better accuracy.

- Optimization: The ResNet-18 model with random initialization may require more optimization to perform well on the given tasks. Fine-tuning or adjusting the learning rate can potentially improve the model's accuracy.

- Hyperparameters: The performance of the ResNet-18 model with random initialization is highly dependent on the choice of hyperparameters, such as the learning rate, batch size, and number of training epochs. Optimal hyperparameters need to be chosen to achieve better accuracy on the given tasks.

References:

- https://pytorch.org/docs/stable/index.html
- https://numpy.org/doc/stable/reference/index.html
- https://torchmetrics.readthedocs.io/en/stable/
- https://github.com/elena-ecn/optuna-optimization-for-PyTorch-CNN/blob/main/optuna_optimization.py
- https://colab.research.google.com/drive/1sVKpN4-yZviCwvBYdjIlu-o8Z4TFL68O?usp=sharing
- https://horovod.ai/#:~:text=Horovod%20is%20a%20distributed%20deep,weeks%20to%20hours%20and%20minutes
- https://colab.research.google.com/drive/1BRr-nL5pevvrjY1SXAV0XfZ9OHKq1DGW?usp=sharing
- https://colab.research.google.com/drive/1mVQXbROal23jz6tTM70RlUxn8FHgI_xk?usp=sharing
- https://colab.research.google.com/github/wandb/examples/blob/master/colabs/intro/Intro_to_Weights_%26_Biases.ipynb
- https://stackoverflow.com/
- https://www.youtube.com/watch?v=Z_ikDlimN6A&t=10644s
- Class ppts.