DL-Ops Lab Assignment-7 Report

Task: to train a Convolutional Neural Network on two datasets using slurm and submit the jobs
Note: All the files are uploaded in the zipped folder, so there is no need to rerun the task.

Objectives:

Question 1.

- Train MobileNet_V2 on CIFAR-10 dataset
- datasets and extract them to folders on the GPU server and preprocess the data.
- Set the loss function, optimiser, and metrics and compile the model
- Also measure the training time (use timeit module for instance) for 10 and 15 epochs
- identify the set of hyperparameters that results in similar performance as compared to the best performance in the previous step but with lower training time

Procedure:

- Imports necessary libraries and modules, such as PyTorch, torchvision, numpy, torchmetrics, and others.
- Defines a transform to be used on the CIFAR10 dataset, which includes an image augmentation technique and converting the images to tensors.
- Loads the CIFAR10 dataset and converts it to torch DataLoader objects after applying different types of transformation.
- Defines a model using MobileNetV2 architecture, adds a classifier layer with dropout and a linear layer for the output, and sets the model to the available device (either "cuda" or "cpu").
- Initializes hyperparameters to be used in training the model, including learning rate, betas, eps, weight decay, and number of epochs.
- Trains the model using a grid search approach, where each combination of hyperparameters is trained and evaluated separately. For each combination, the model is initialized, an Adam optimizer is created using the specified hyperparameters, and the train function from the engine module is called to train the model.
- After training, the results are plotted using the plot_curves function from the plotting module, which displays loss and accuracy curves for the train and test sets. The results are saved with a filename that includes the hyperparameters used for that particular training.
- The total time taken for training the model for each combination of hyperparameters is printed, and the current status of the training (i.e., current experiment number out of the total number of experiments) is displayed as well.

Results:

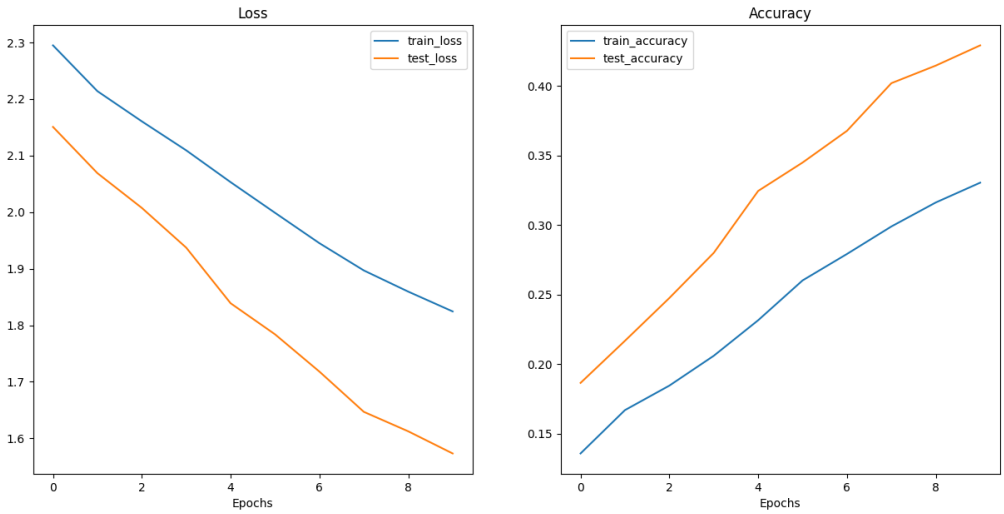


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0.0001

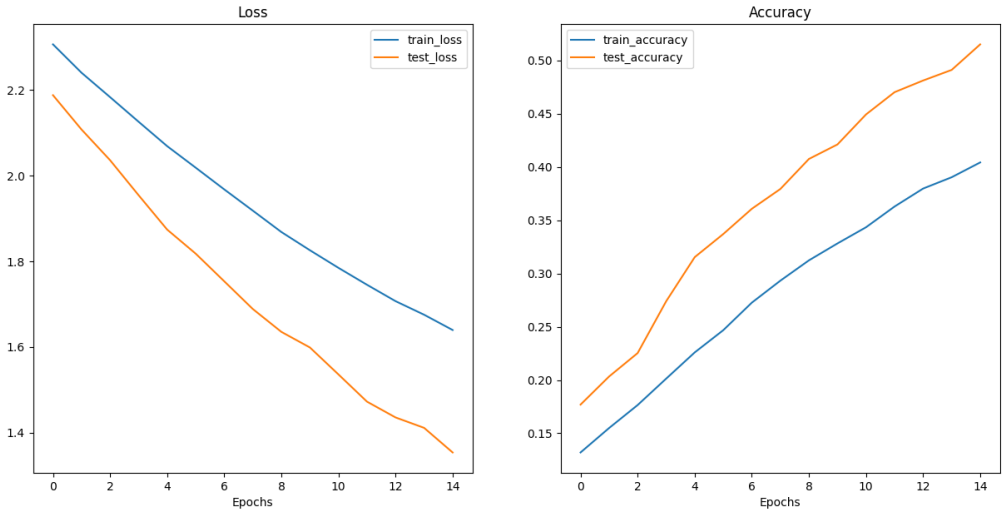


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0

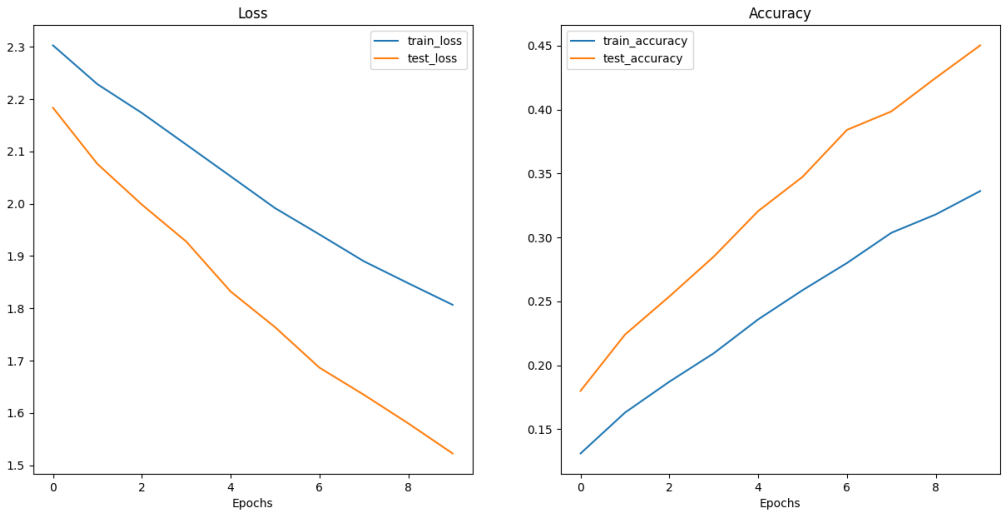


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0.0001

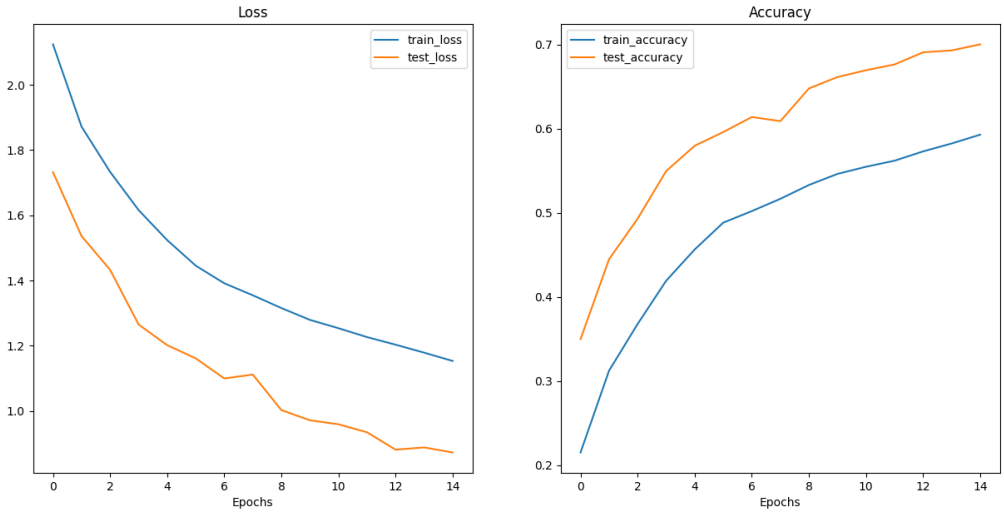


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0.0001

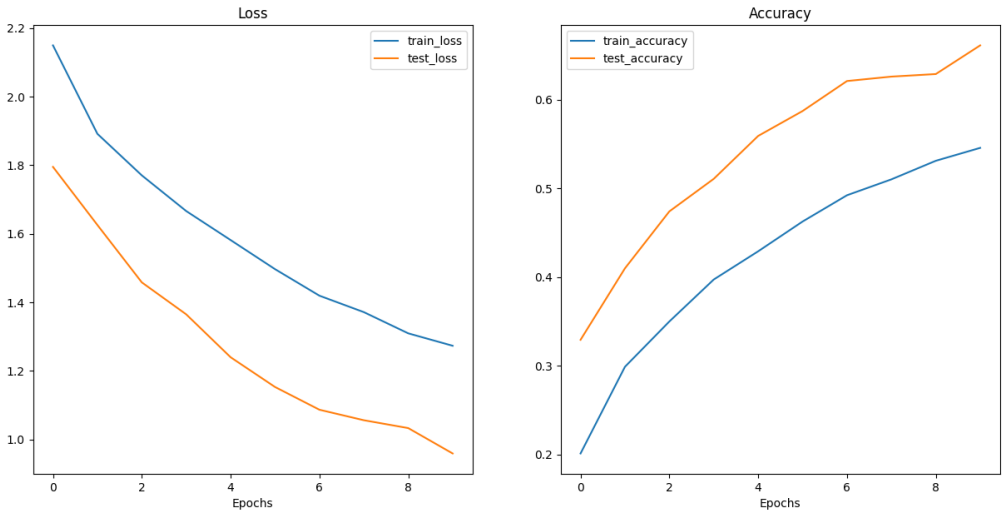


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0

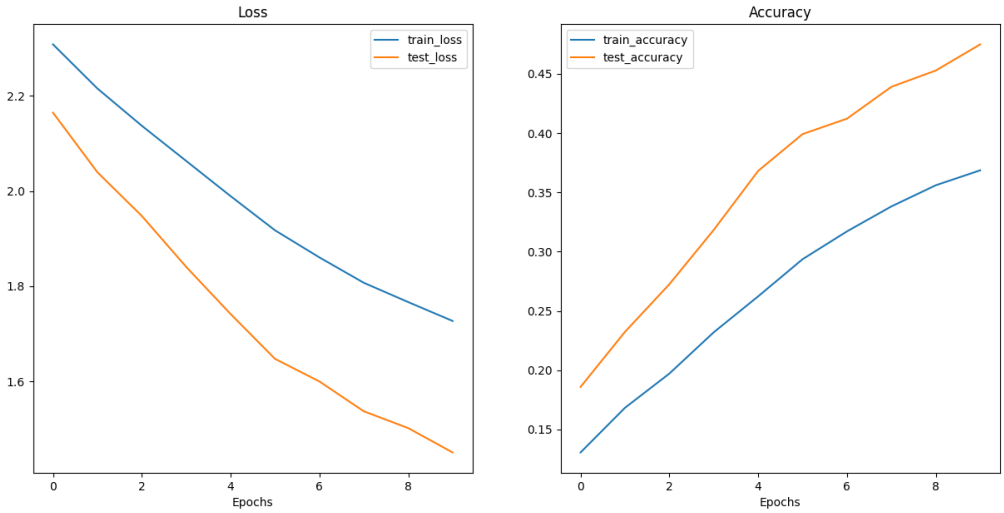


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0.0001

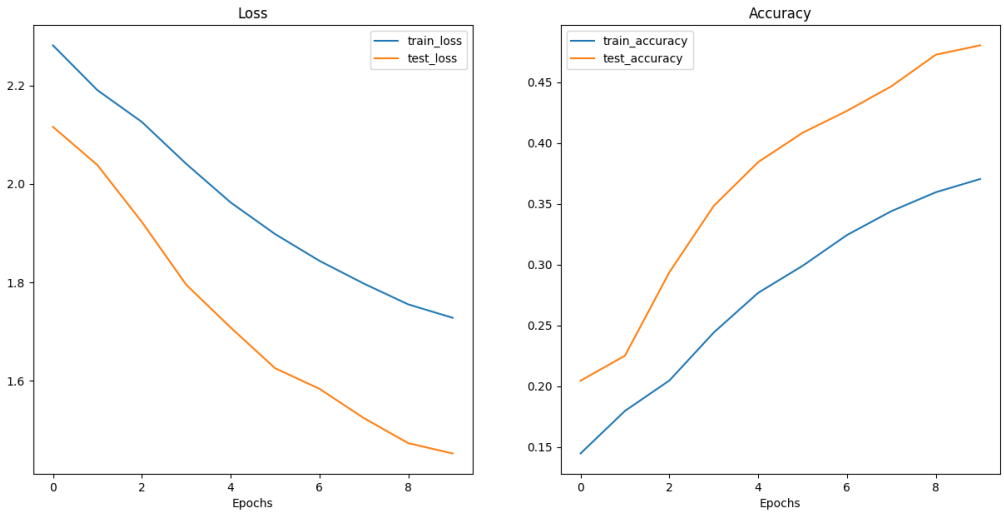


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0

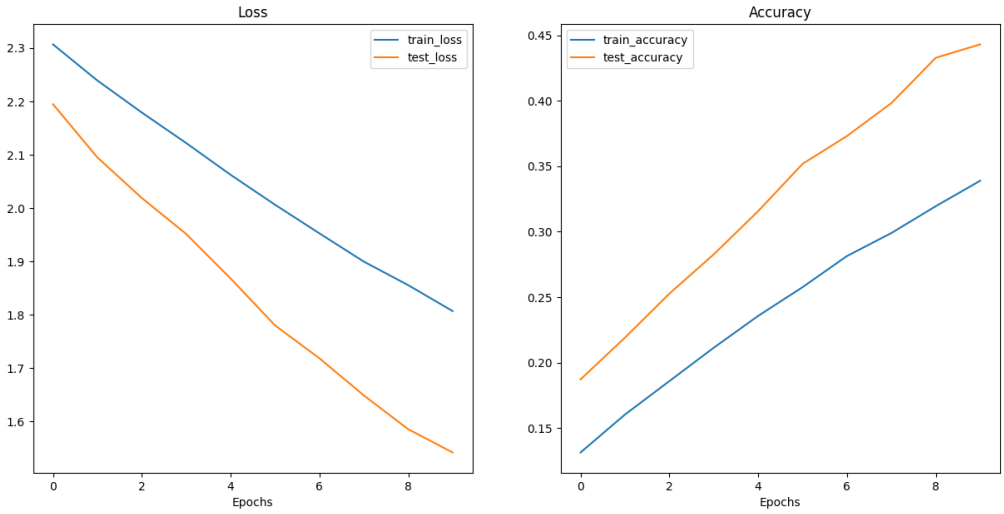


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0

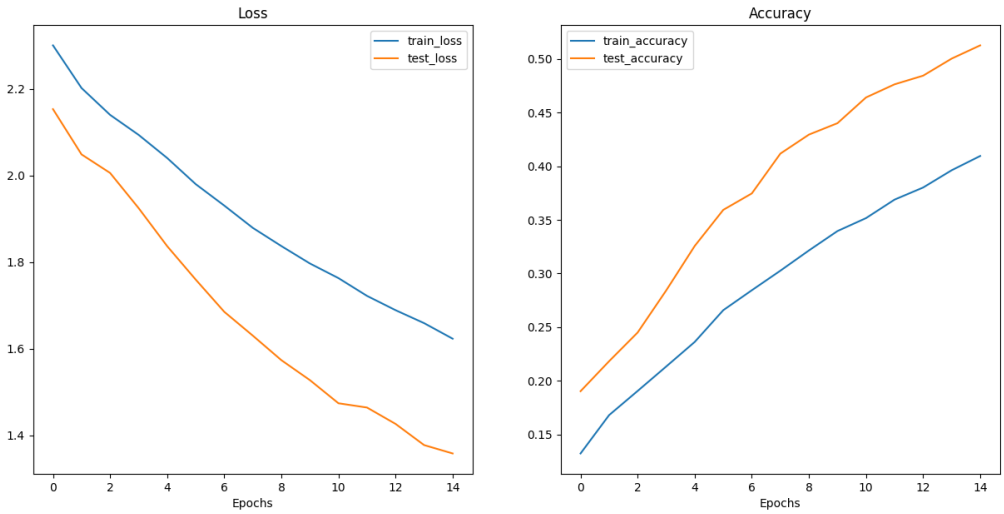


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0

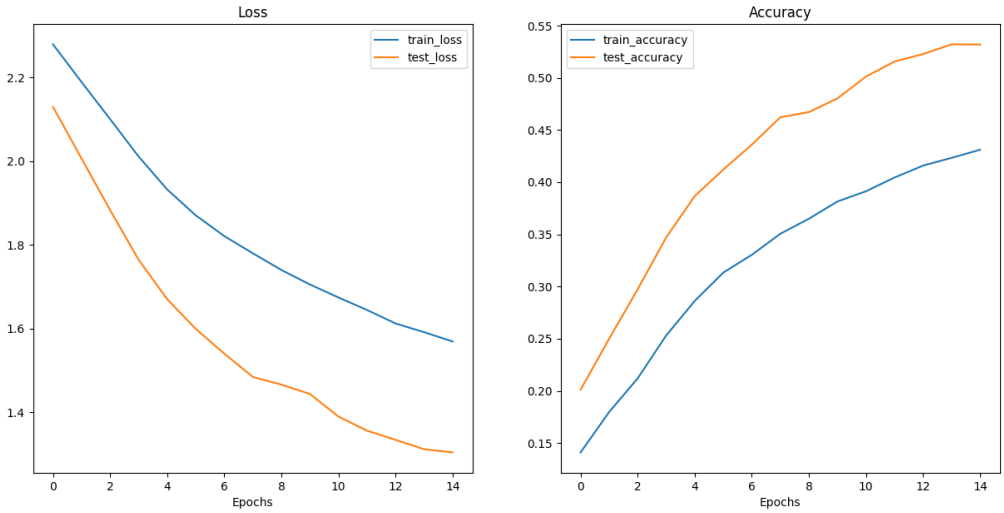


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0

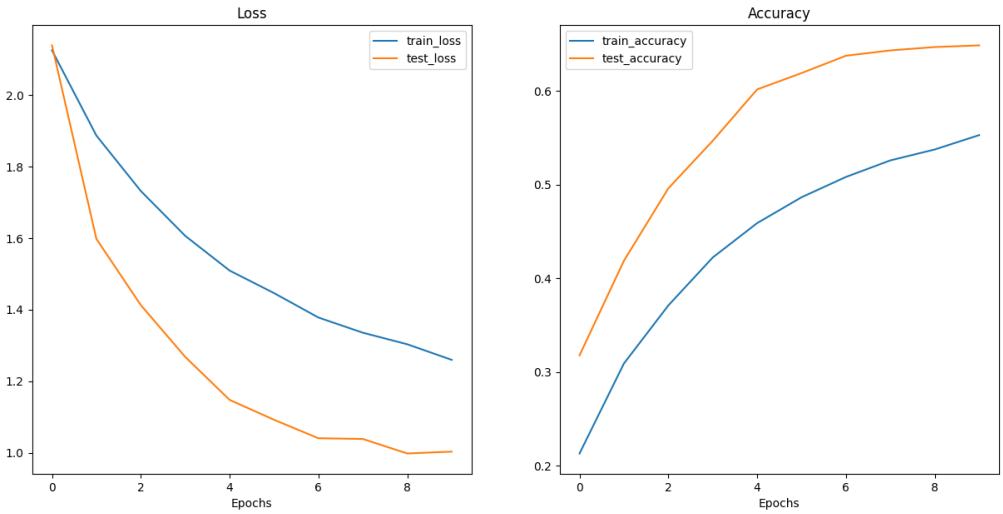


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0.0001

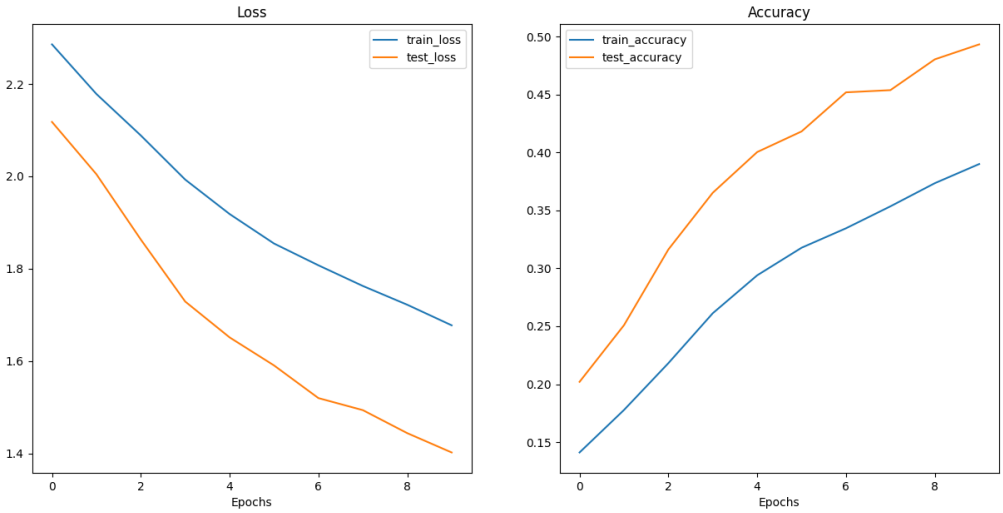


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0

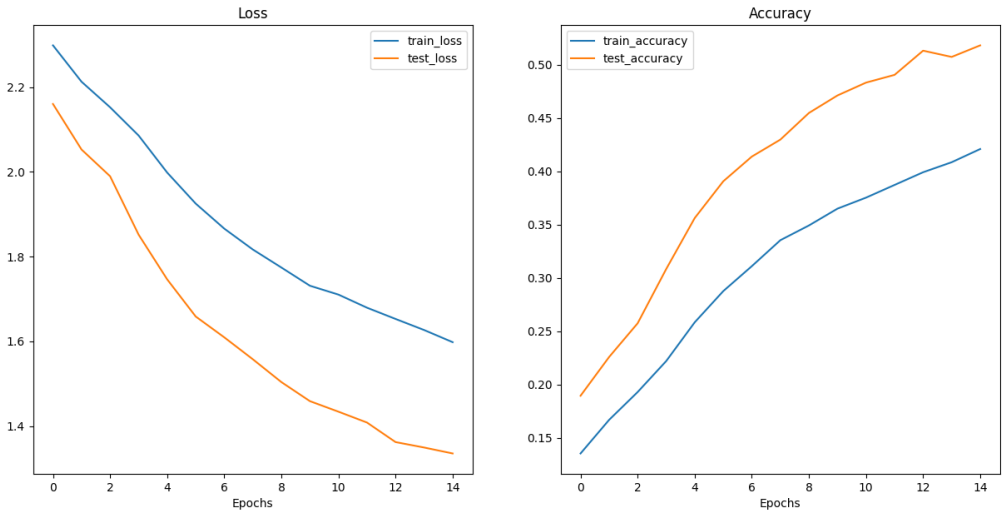


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0.0001

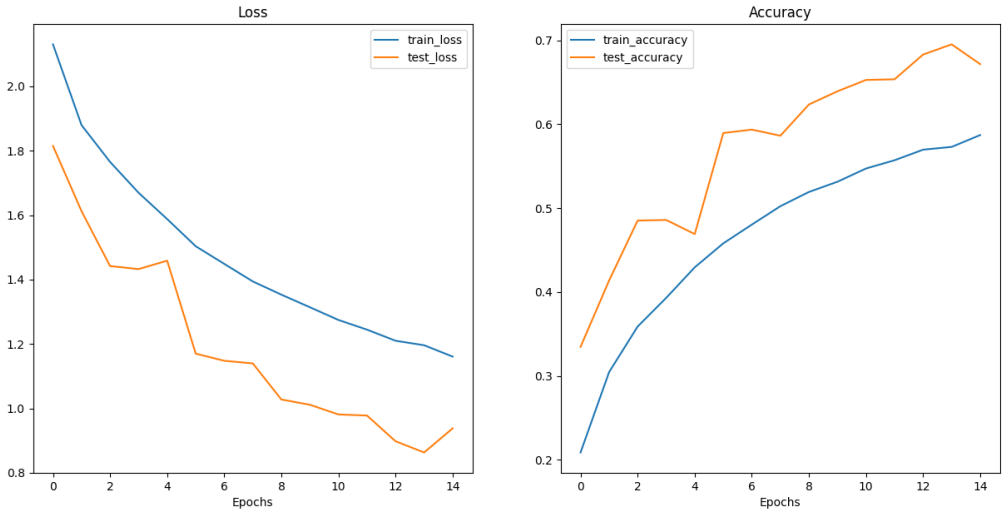


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0

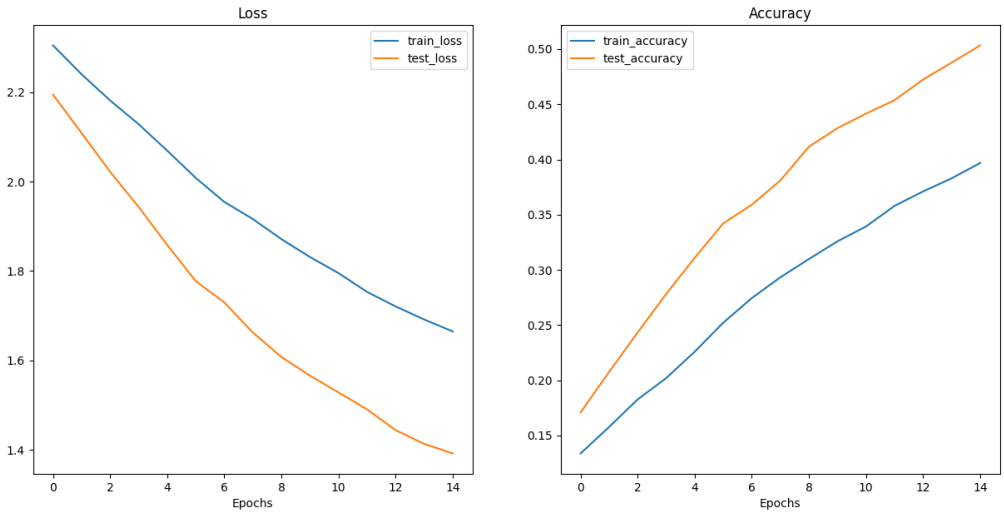


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0.0001

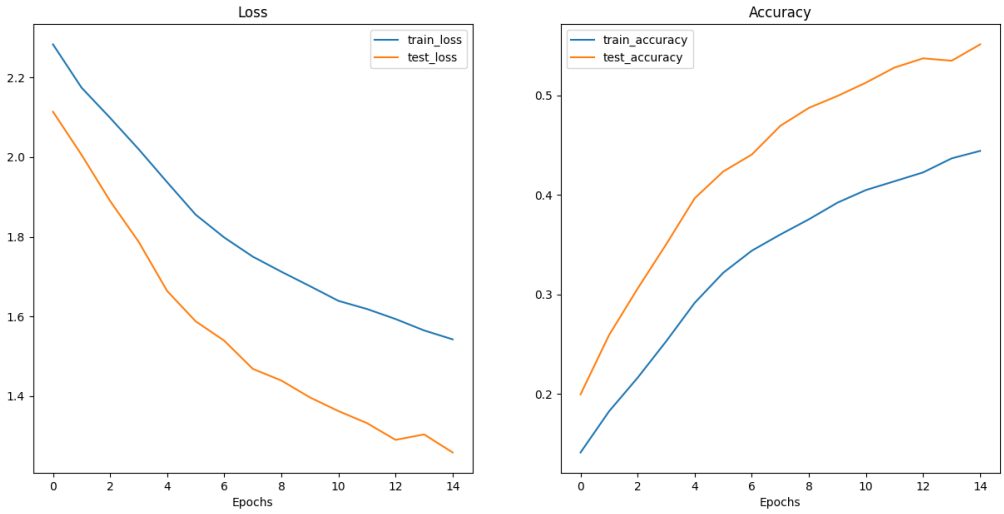


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0

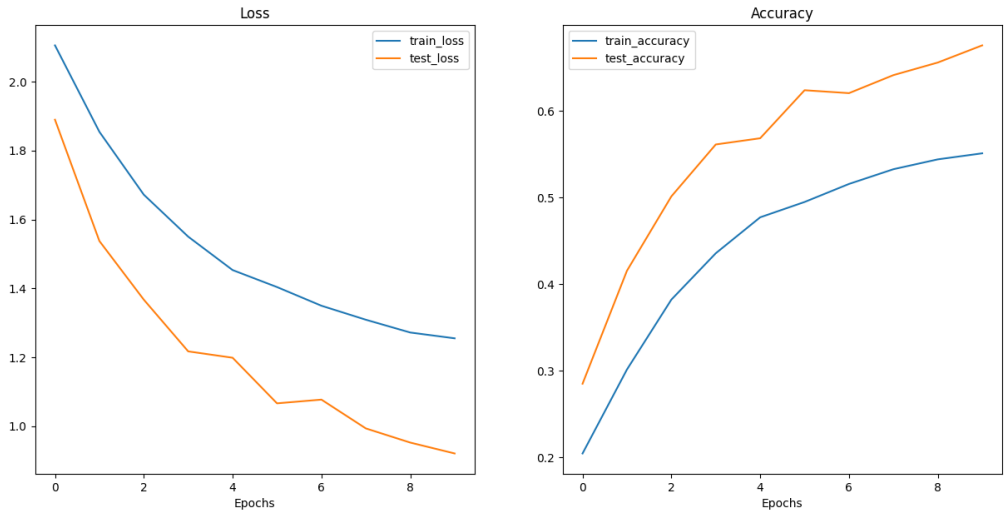


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0

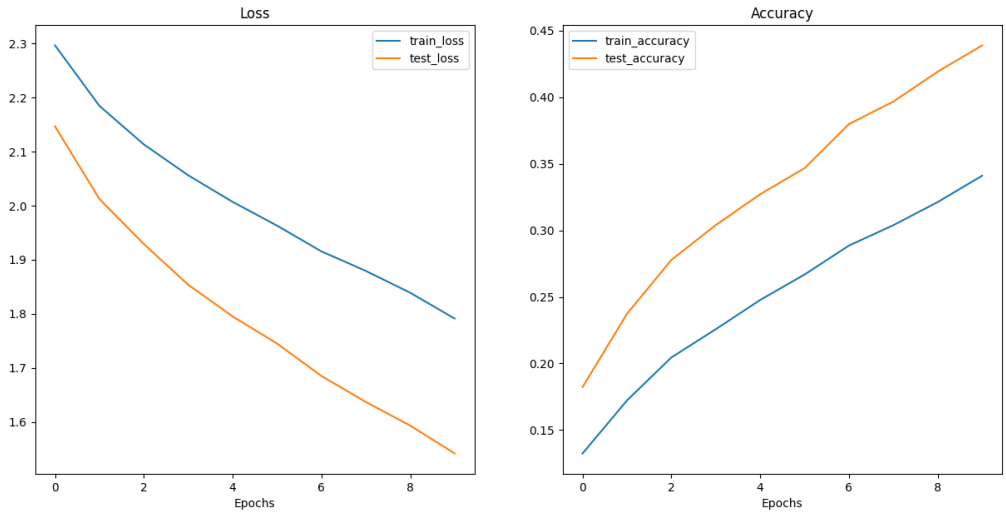


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0

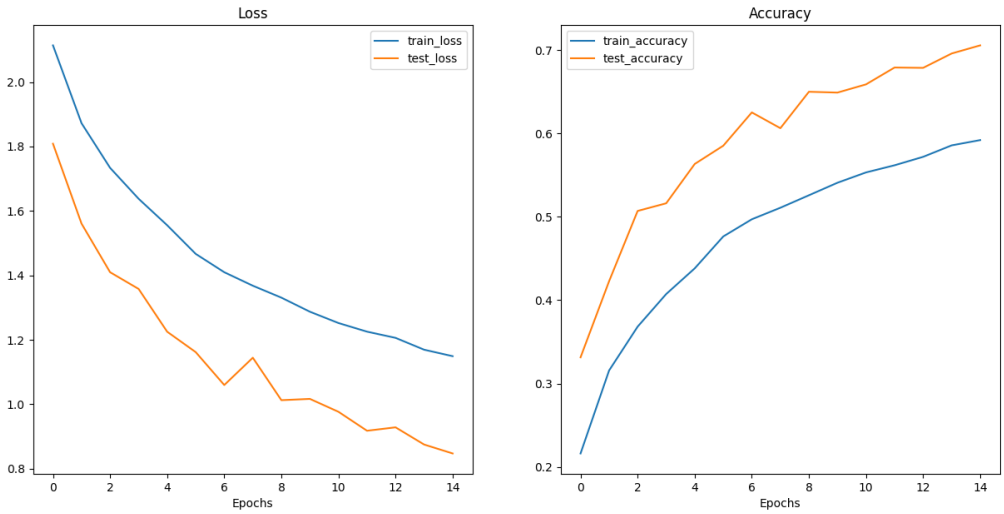


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

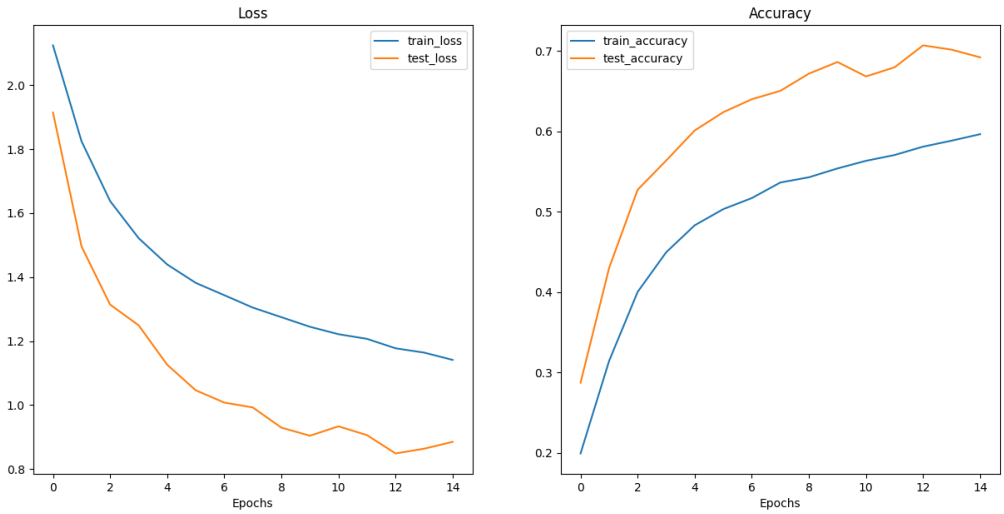


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0

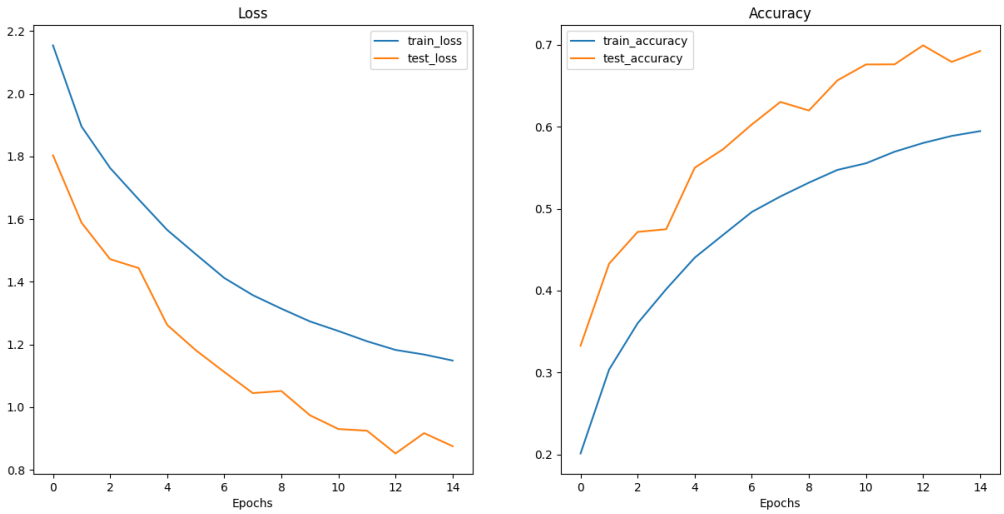


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0

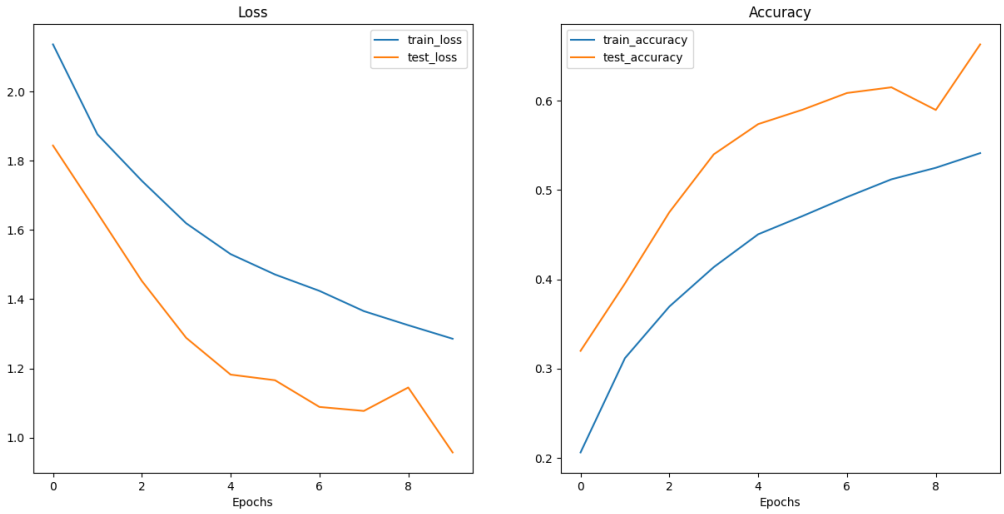


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

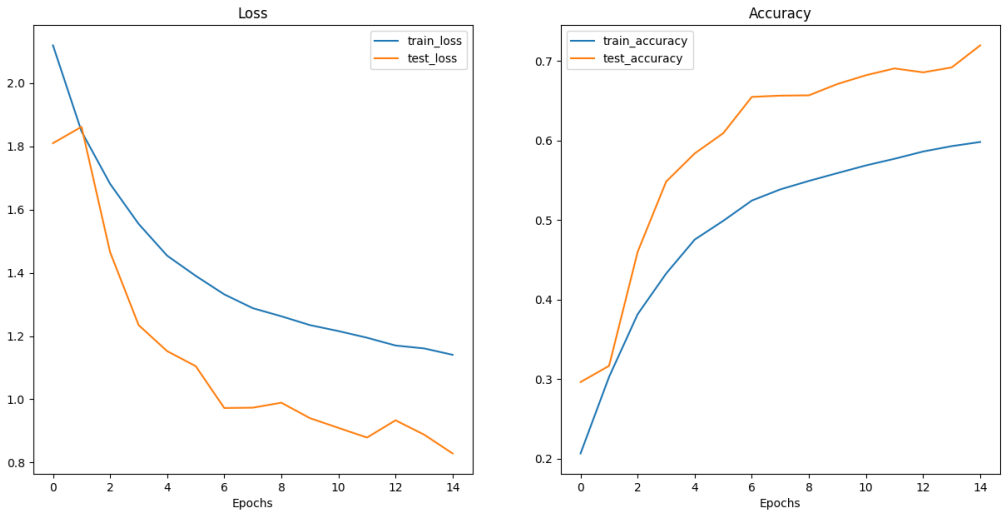


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0

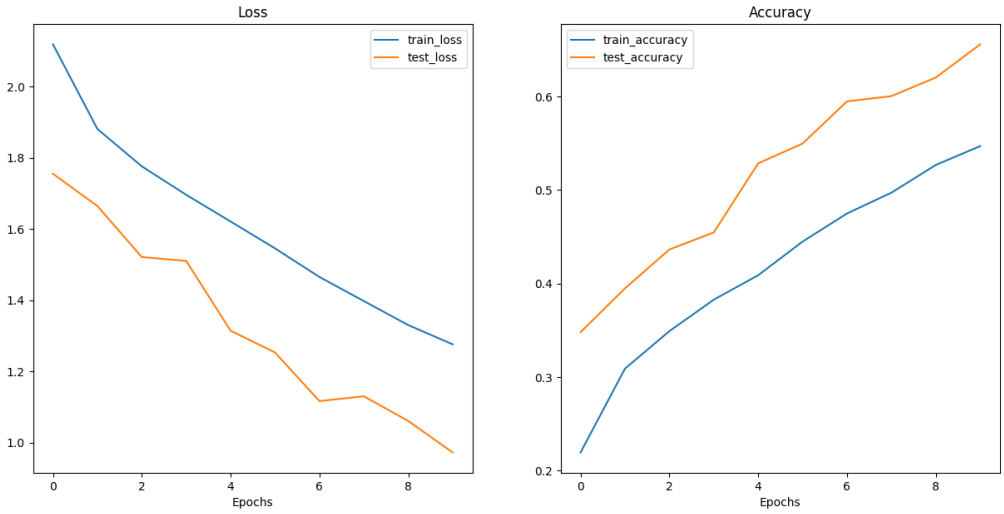


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0.0001

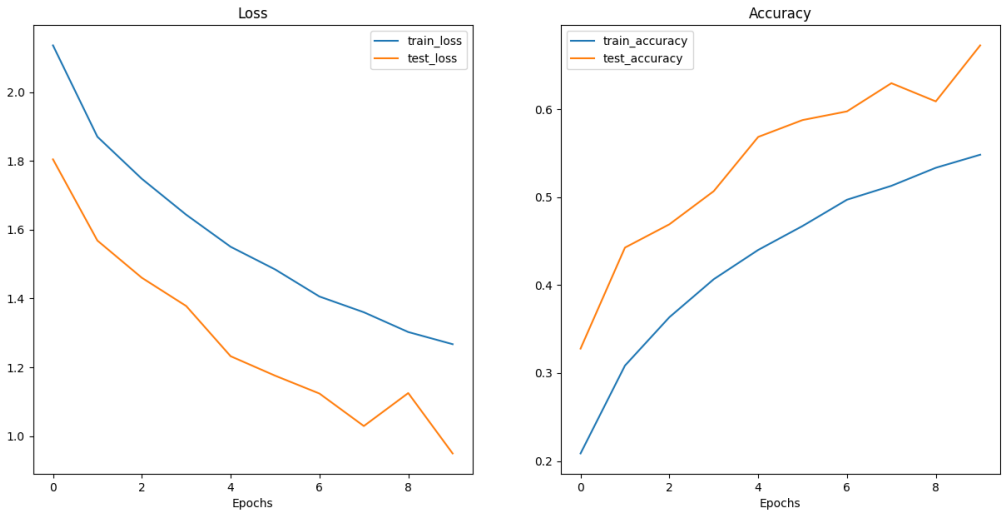


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0.0001

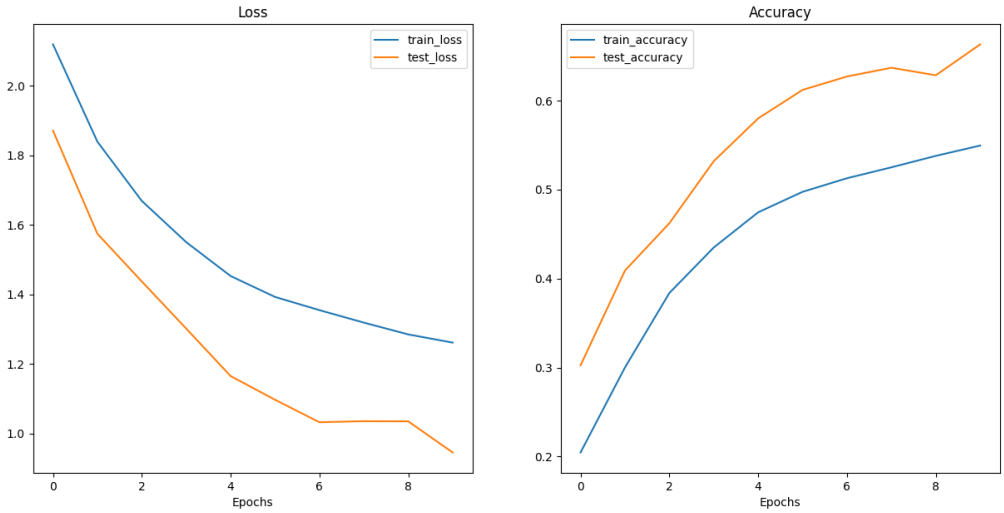


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.8, 0.888)_eps_1e-09_weight_decay_0

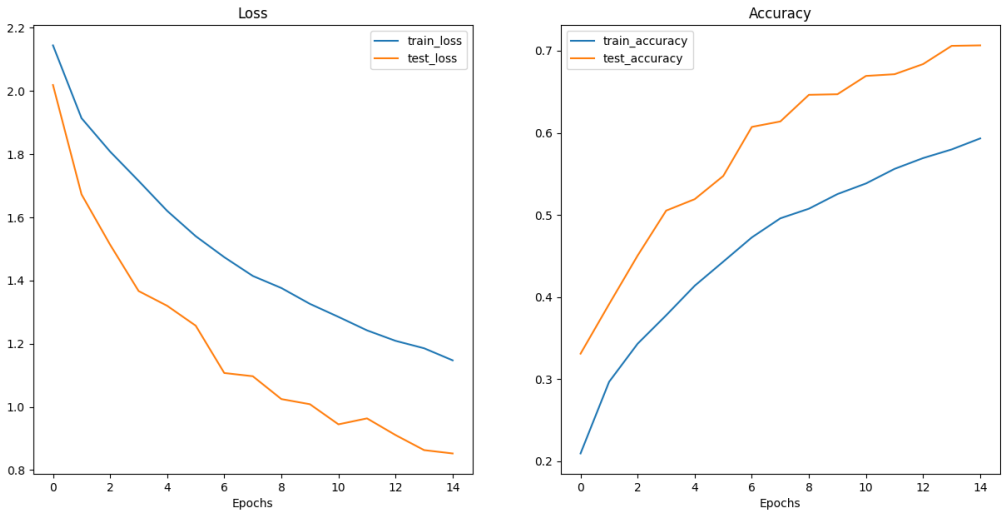


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0.0001

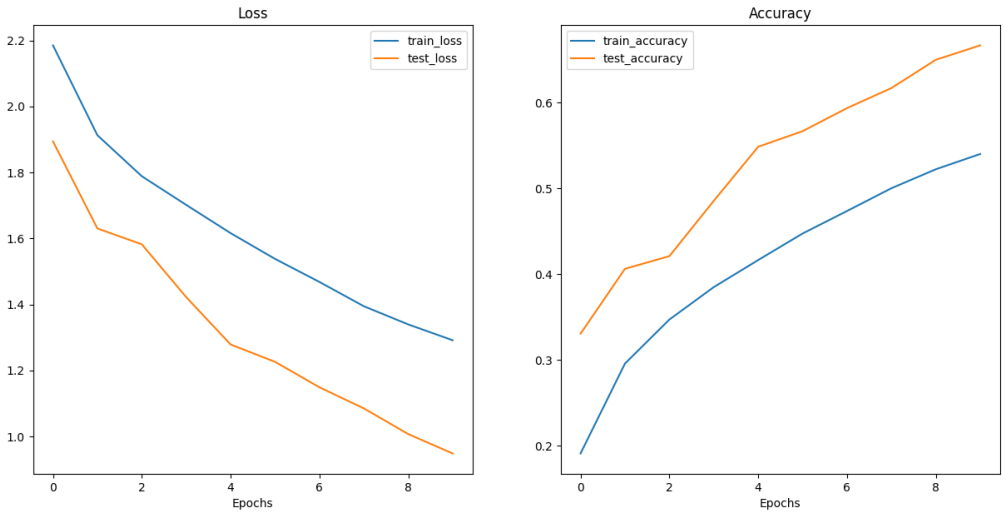


Fig: MobileNetV2_epoch_10_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0.0001

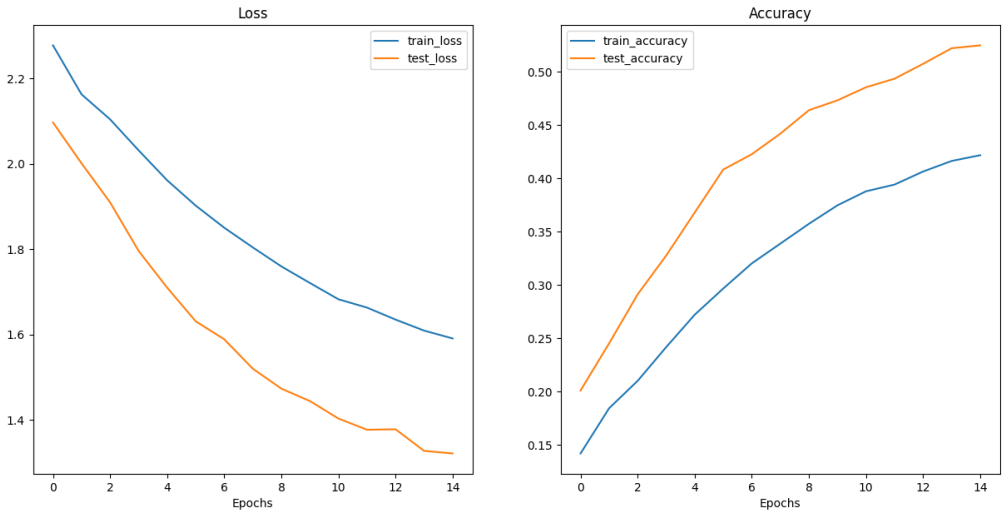


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.8, 0.888)_eps_1e-08_weight_decay_0.0001

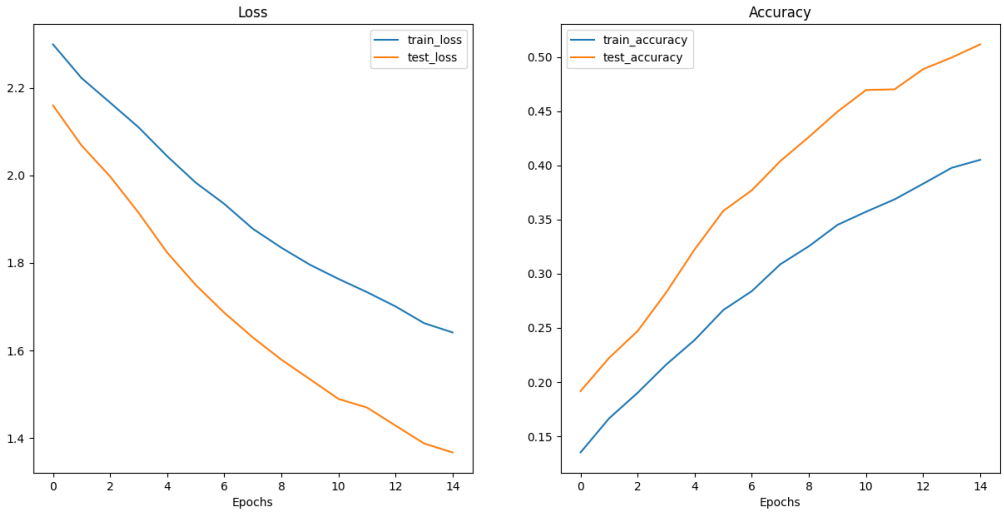


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.0001_betas_(0.9, 0.999)_eps_1e-08_weight_decay_0.0001

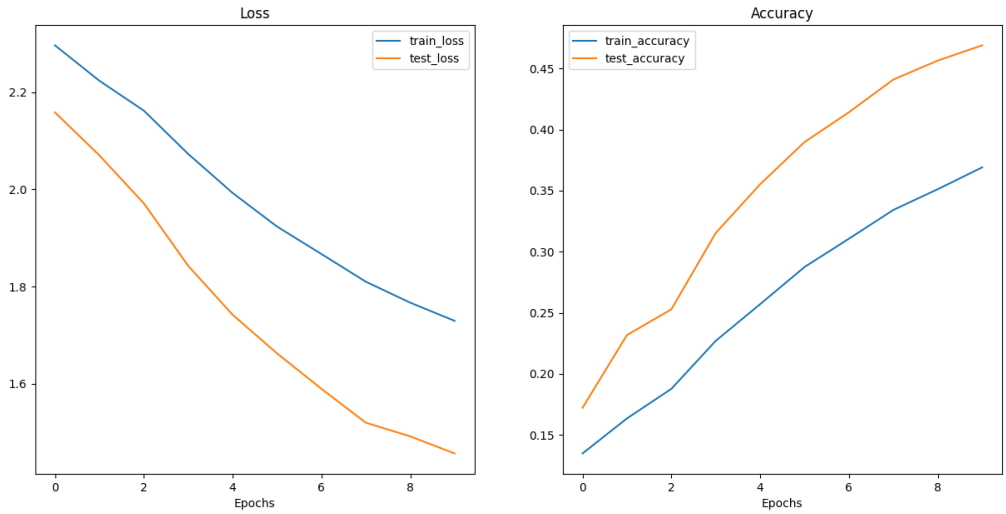


Fig: MobileNetV2_epoch_15_optim_adam_lr_0.001_betas_(0.9, 0.999)_eps_1e-09_weight_decay_0.0001

- Conclusions from the above Cuves:
  - The best test accuracy is 71.95 using “lr: 0.001, betas: (0.8, 0.888), eps: 1e-08, weight_decay: 0.001, epochs: 15”
  - total training time: 565.837 sec. I.e. ~9.12 mins
  - Second best accuracy is 70.65 using “parms: lr: 0.001, betas: (0.9, 0.999), eps: 1e-08, weight_decay: 0.0001, epochs: 15”
  - total training time: 527.065 sec. I.e. ~8.40 mins
  - And Also, 66.33 accuracy can be achieved using: “lr: 0.001, betas: (0.8, 0.888), eps: 1e-09, weight_decay: 0.001, epochs: 10”
  - total training time: 368.568 sec. I.e. ~6.5 mins
- Please note that all the models have been trained from scratch and not fine-tuned, resulting in the above performance.
- Also Note: the run log file is also attached with the code for further running logs.
- Bash file to schedule the job.

run_job.sh U

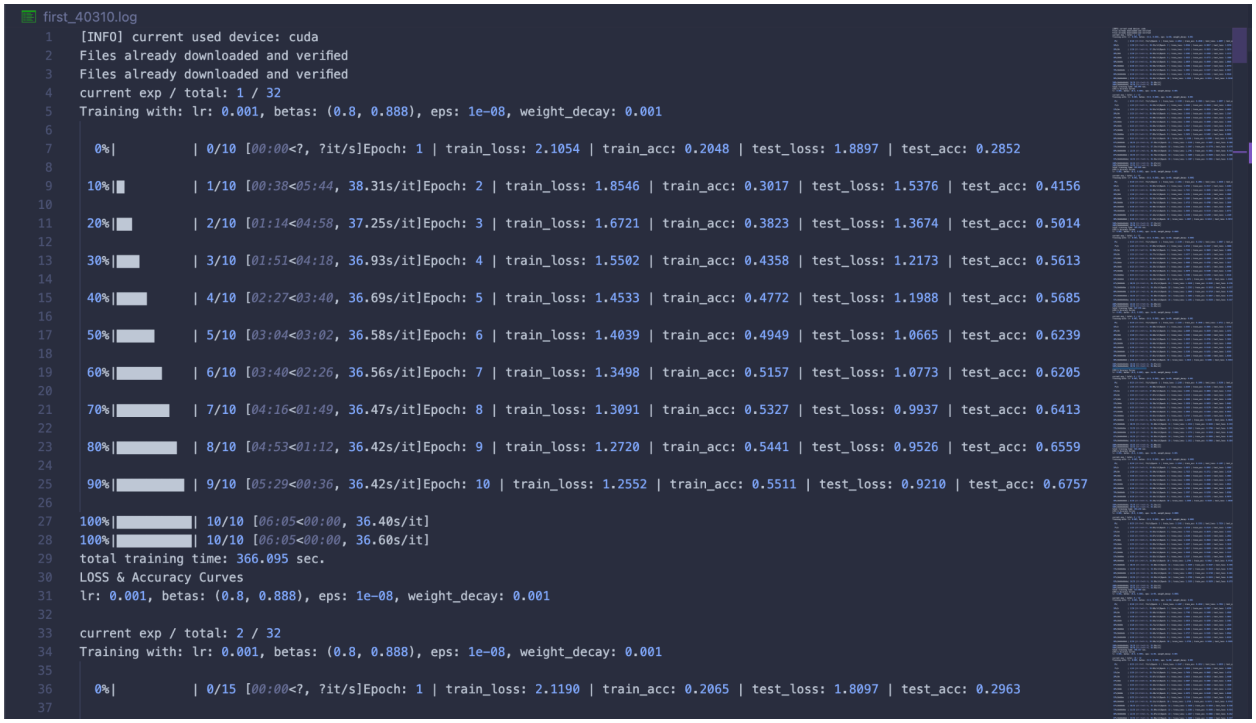
run_job.sh

```
#!/bin/bash
#SBATCH --job-name=pytorch_dlops_lab_ass_7  # Job name
#SBATCH --partition=gpu2    #Partition name can be test/small/medium/large/gpu #Partition “gpu” should be used only for gpu jobs
#SBATCH --nodes=1           # Run all processes on a single node
#SBATCH --ntasks=1          # Run a single task
#SBATCH --cpus-per-task=4   # Number of CPU cores per task
#SBATCH --gres=gpu      # Include gpu for the task (only for GPU jobs)
#SBATCH --mem=8gb           # Total memory limit
#SBATCH --time=10:00:00     # Time limit hrs:min:sec
#SBATCH --output=first_%j.log # Standard output and error log

module load python/3.8

python3 -u main.py
```

- Log File snapshot:



References:

https://pytorch.org/docs/stable/index.html
https://slurm.schedmd.com/man_index.html
https://cc.iitj.ac.in/hpc/
https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html
https://pytorch.org/docs/master/profiler.html
https://numpy.org/doc/stable/reference/index.html
https://torchmetrics.readthedocs.io/en/stable/
https://stackoverflow.com/
https://www.youtube.com/playlist?list=PL5l6Qz3Xhfi9Jn9-iMKJisYsSW5tRzPSd
DL-OPS Class videos.