DL-Ops

Lab Assignment 8 - Report

Debonil Ghosh

Roll No: M21AIE225

Executive MTech

Artificial Intelligence

Indian Institute of Technology, Jodhpur

Problem Statement:

# **Question 1 [100 marks]**

Train a ResNet18 model for classification on even classes of CIFAR-10 (for students with even roll no)/ odd classes of FashionMNIST (for students with odd roll no) for 30 epochs. Analyze the training of model using wandb library and “weights & biases” tool covered in the class.

1) Show comparison of the performance of the hyperparameter tuning using plots using the “weights & biases” tool. Hyperparameters: choice of activation function, optimizer

2) Write the observations about GPU (like GPU utilization, temperature etc).

3) Show all the logs about accuracy, loss using plots using “weights & biases” tool

4) Generate the report of the entire experiment using the “weights & biases” tool. Attach this report to your assignment report.

5) Add a visual result analysis on the “weights & biases” tool taught by to Dr Anush, where he shows the prediction for different images. Include this in the assignment video in addition to the other demonstrations.

Note: The comparison should be done at least between 3-4 models (resulting from the choice of hyperparameters)

Reference:

https://wandb.ai/site

## **Solution Google Colab Link:**

# <https://colab.research.google.com/drive/1VLG8ipcuIq4utJUc0sbYIwJByvrE8E14?usp=sharing>

## **Solution Weights & Biases Report Link:**

<https://api.wandb.ai/links/ghosh-11/c7dy4c2n>

# Model Allotted:

ResNet18 on FashionMNIST dataset

[Roll number M21AIE225]

# ResNet18

ResNet-18 is a convolutional neural network architecture developed by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun of Microsoft Research in 2015. It is a variant of the original ResNet architecture, which stands for "Residual Network," and was developed to address the problem of vanishing gradients in very deep neural networks. ResNet-18 is a relatively shallow network, consisting of only 18 layers, but it is still powerful enough to achieve state-of-the-art performance on a variety of computer vision tasks.

The ResNet-18 architecture is based on a series of residual blocks, each of which contains two convolutional layers with batch normalization and a shortcut connection that skips over the second convolutional layer. The shortcut connection allows information to flow directly from the input of the block to the output, bypassing the convolutional layers. This helps to avoid the vanishing gradient problem and allows the network to be trained more effectively.

The first layer of the network is a standard convolutional layer that takes in the input image. This is followed by a max pooling layer that reduces the spatial dimensions of the feature maps. The network then consists of four stages, each of which contains a varying number of residual blocks.

The first stage consists of two residual blocks, each with two convolutional layers with 64 filters. The second stage contains two residual blocks, each with two convolutional layers with 128 filters. The third stage contains two residual blocks, each with two convolutional layers with 256 filters. Finally, the fourth stage contains two residual blocks, each with two convolutional layers with 512 filters.

The output of the final stage is passed through a global average pooling layer, which averages the feature maps over the spatial dimensions. This is followed by a fully connected layer with 1000 units, corresponding to the 1000 classes in the ImageNet dataset. The output of this layer is passed through a softmax activation function to produce the final probability distribution over the classes.

ResNet-18 is a powerful convolutional neural network architecture that addresses the problem of vanishing gradients in deep networks. It achieves state-of-the-art performance on the ImageNet dataset while being relatively small and easy to train. Its success has inspired the development of many other ResNet variants, including ResNet-34, ResNet-50, and ResNet-101, which have achieved even better performance on a variety of computer vision tasks.

## Architecture:

ResNet(

(conv1): Conv2d(1, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)

(layer1): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicBlock(

(conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer2): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer3): Sequential(

(0): BasicBlock(

(conv1): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(128, 256, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer4): Sequential(

(0): BasicBlock(

(conv1): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(256, 512, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(1, 1))

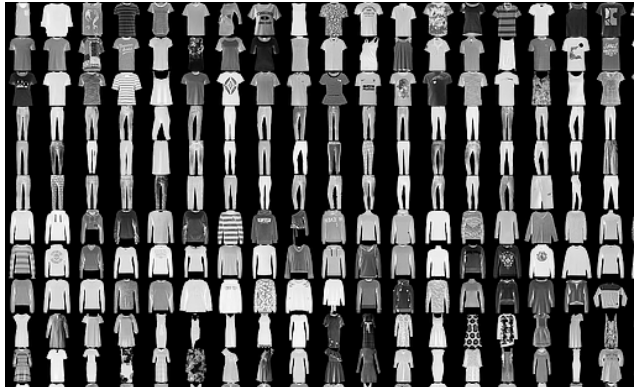
(fc): Linear(in\_features=512, out\_features=5, bias=True)

)

[Above model is modified for 1 channel image input and 5 class output from original ResNet18]

# FashionMNIST Dataset:

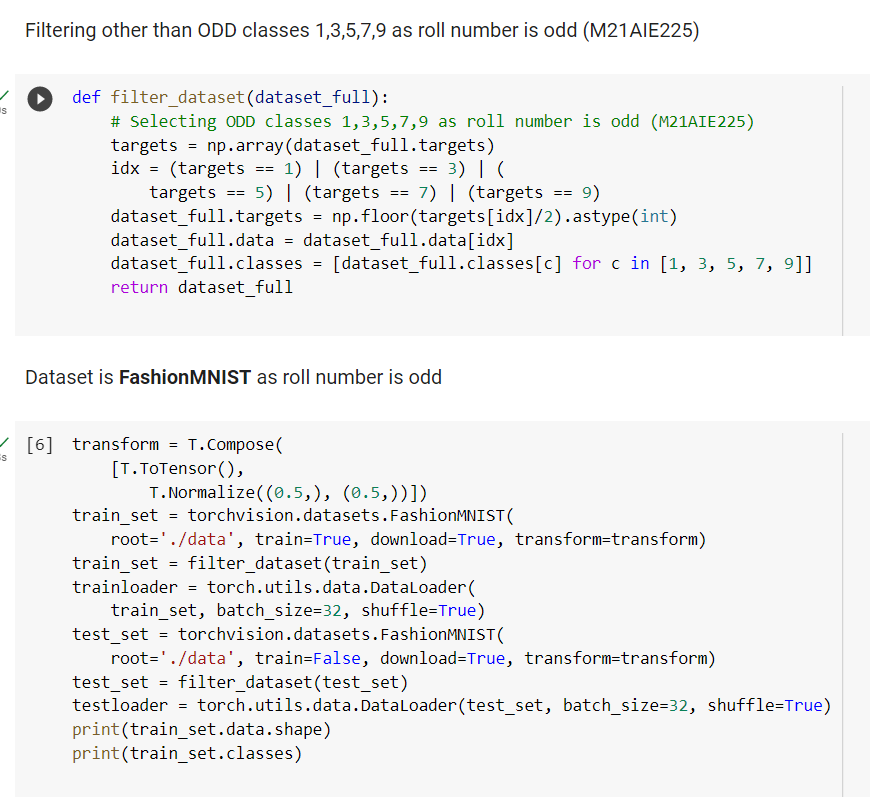
Fashion-MNIST is a dataset of Zalando's article images consisting of **a training set of 60,000** examples and a **test set of 10,000** examples. Each example is a **28x28 grayscale image**, associated with a label from **10 classes**.



# Torch Transformations Applied:

* Normalize to (0.5,), (0.5)
* Transform to tensor

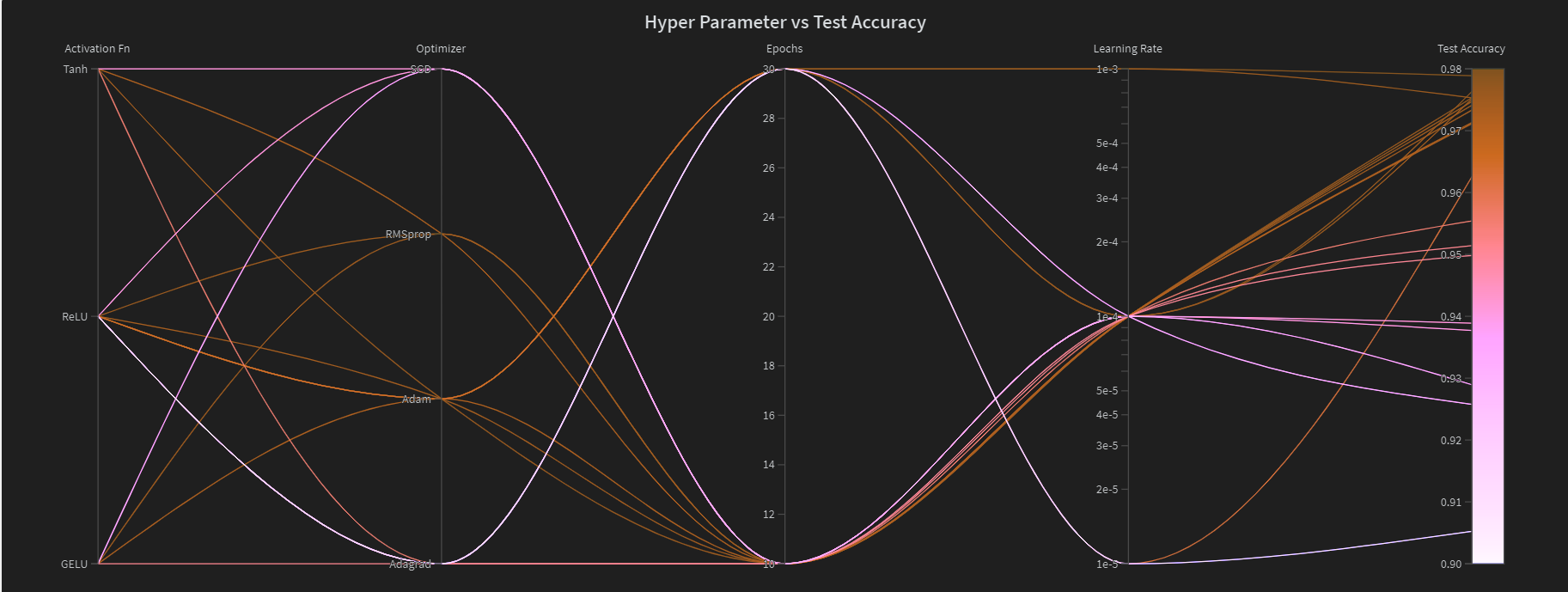
Using default PyTorch data-loader function with the dataset path ‘./data’ and used above mentioned transforms for pre-processing. Selected even classes 0,2,4,6,8 as roll number is odd (M21AIE225).

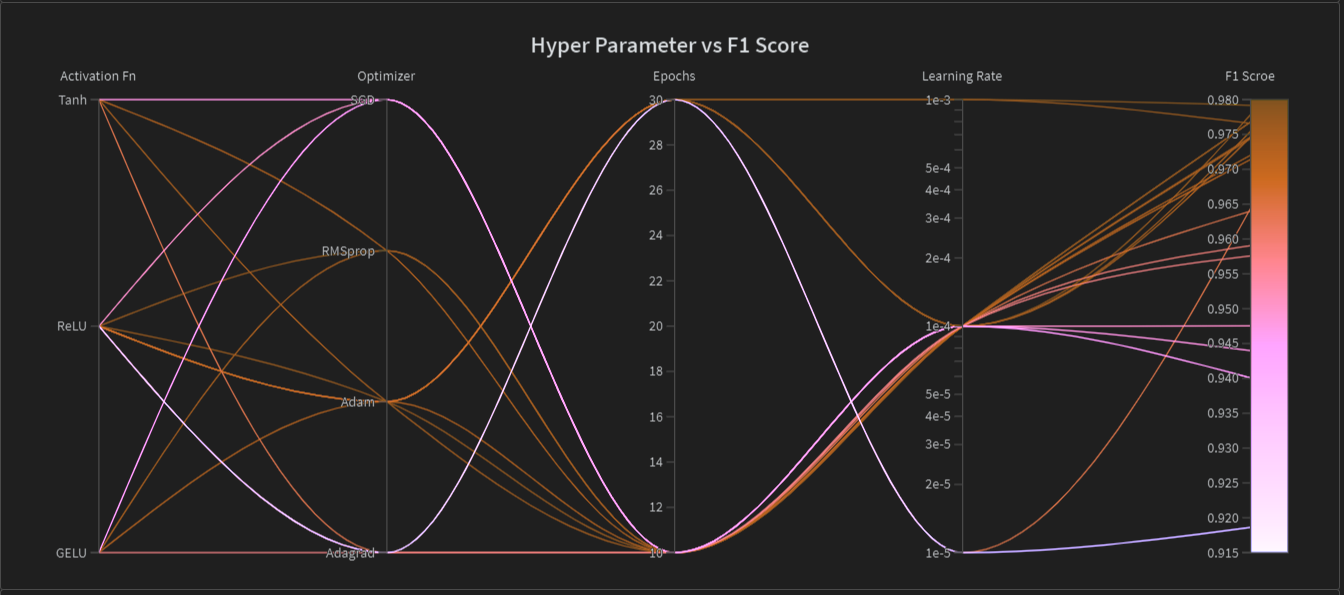


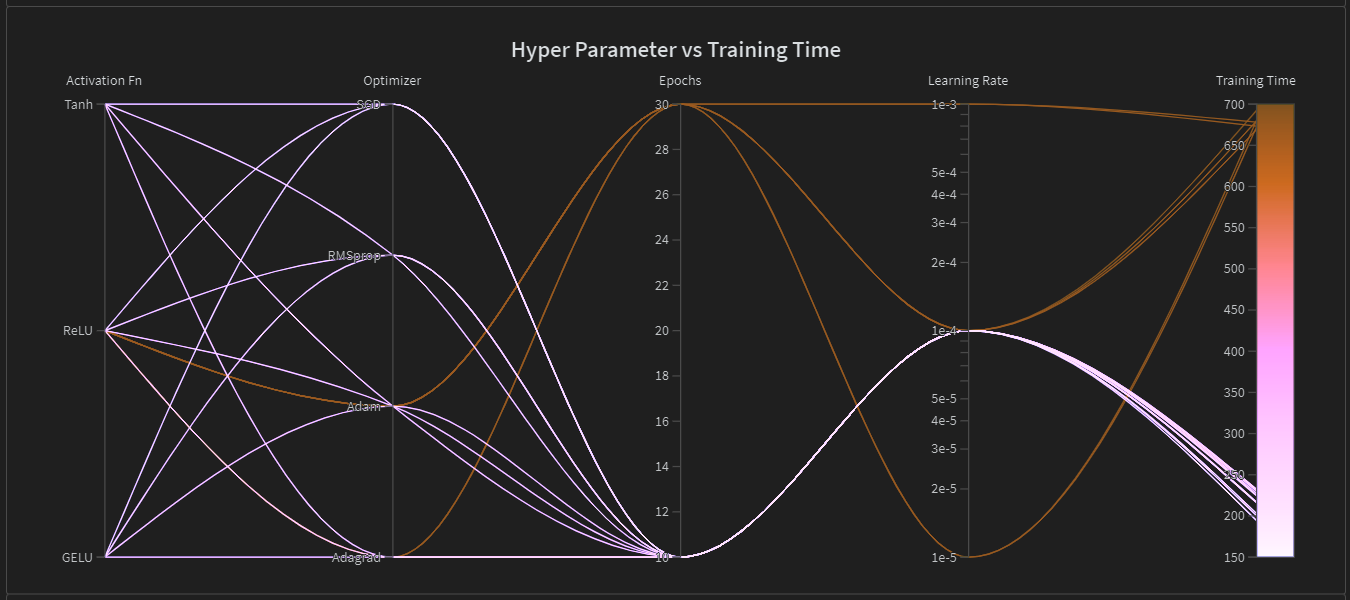
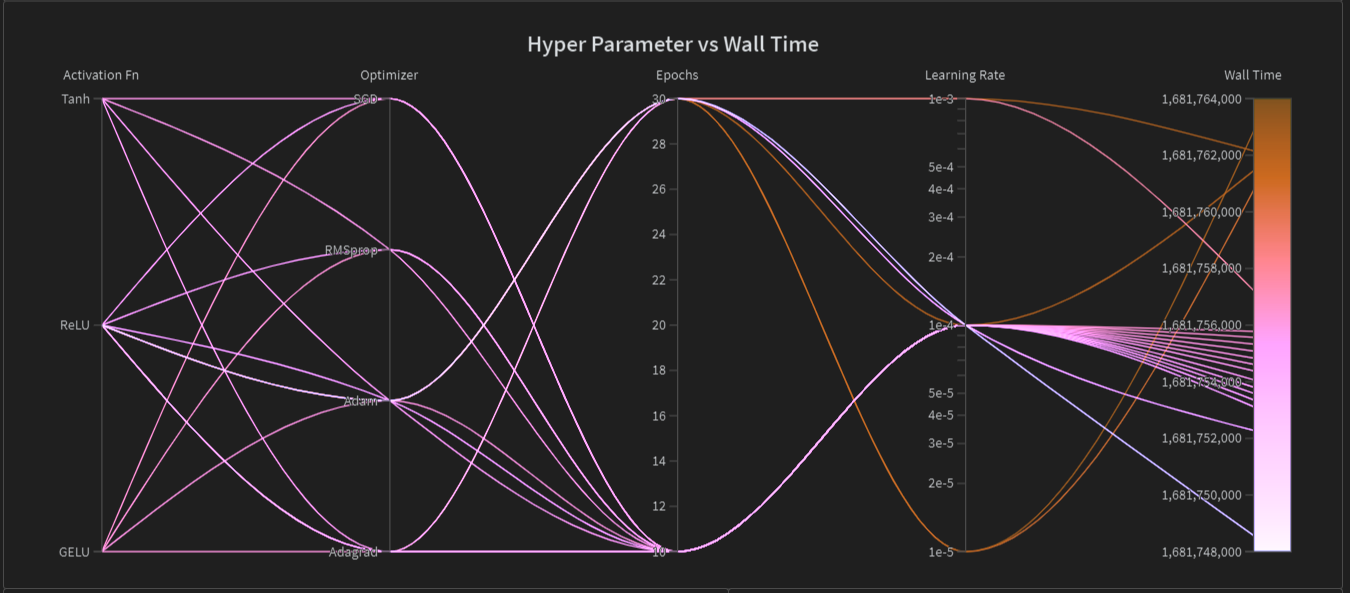
**Hyper Parameters Used:**

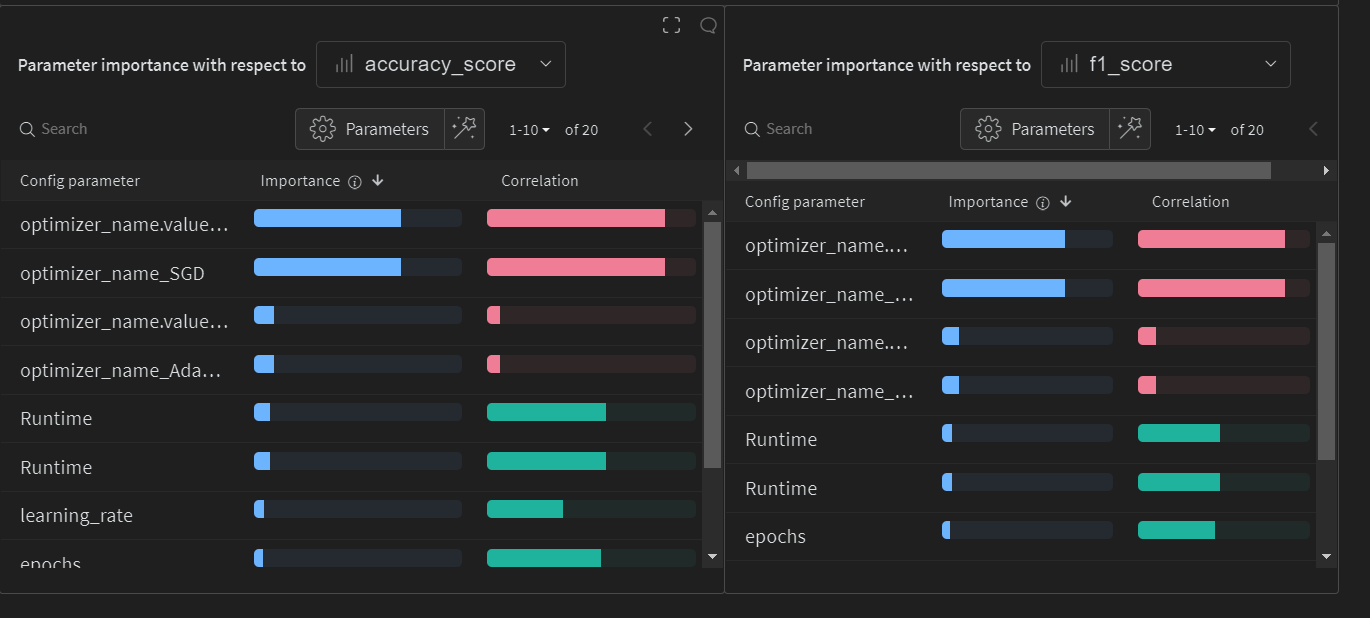
|  |  |
| --- | --- |
| Hyper Parameter | Value |
| Learning Rate | 1e-3, 1e-4, 1e-5 |
| Number of Epochs | 10, 15 |
| Batch Size | 32 |
| Loss Function | CrossEntropyLoss |
| Optimizer | 'Adam','Adagrad','RMSprop','SGD' |
| Activation Functions | 'ReLU','Tanh','GELU' |
| Number of Epochs | 10,30 |

**Comparison** **of the performance of the of different Hyperparameters:**

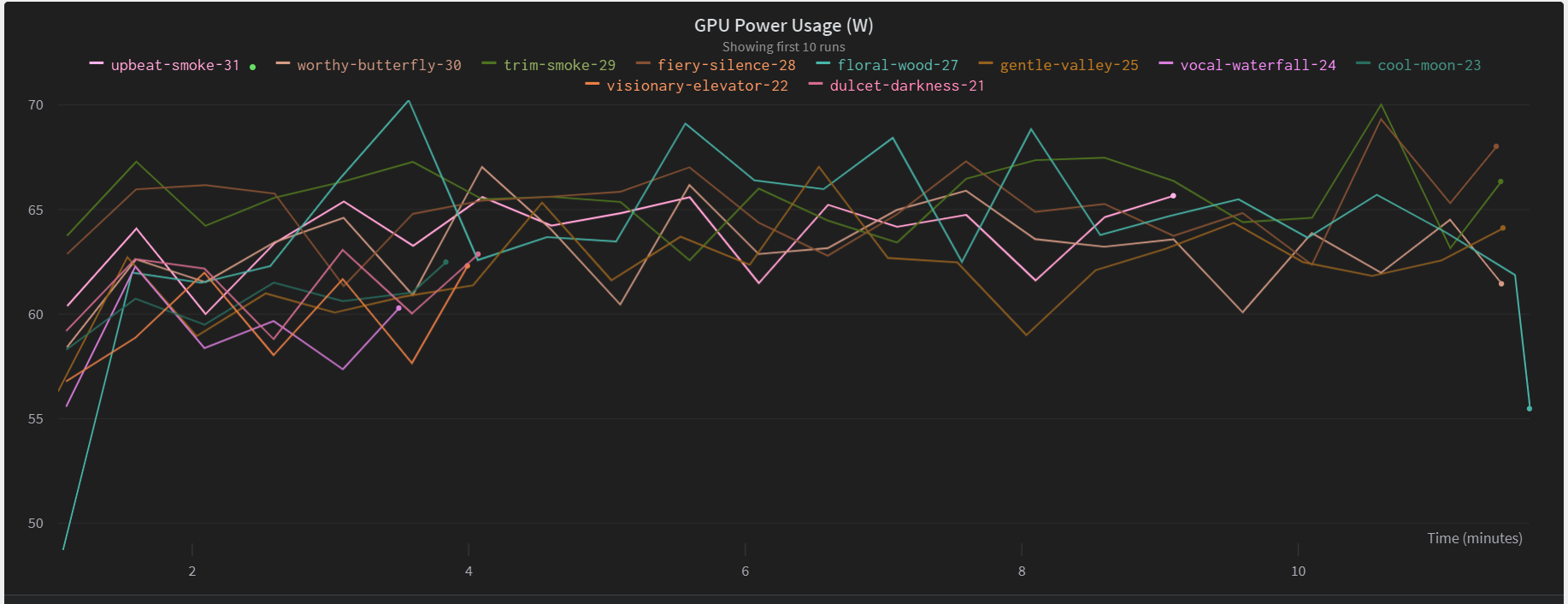


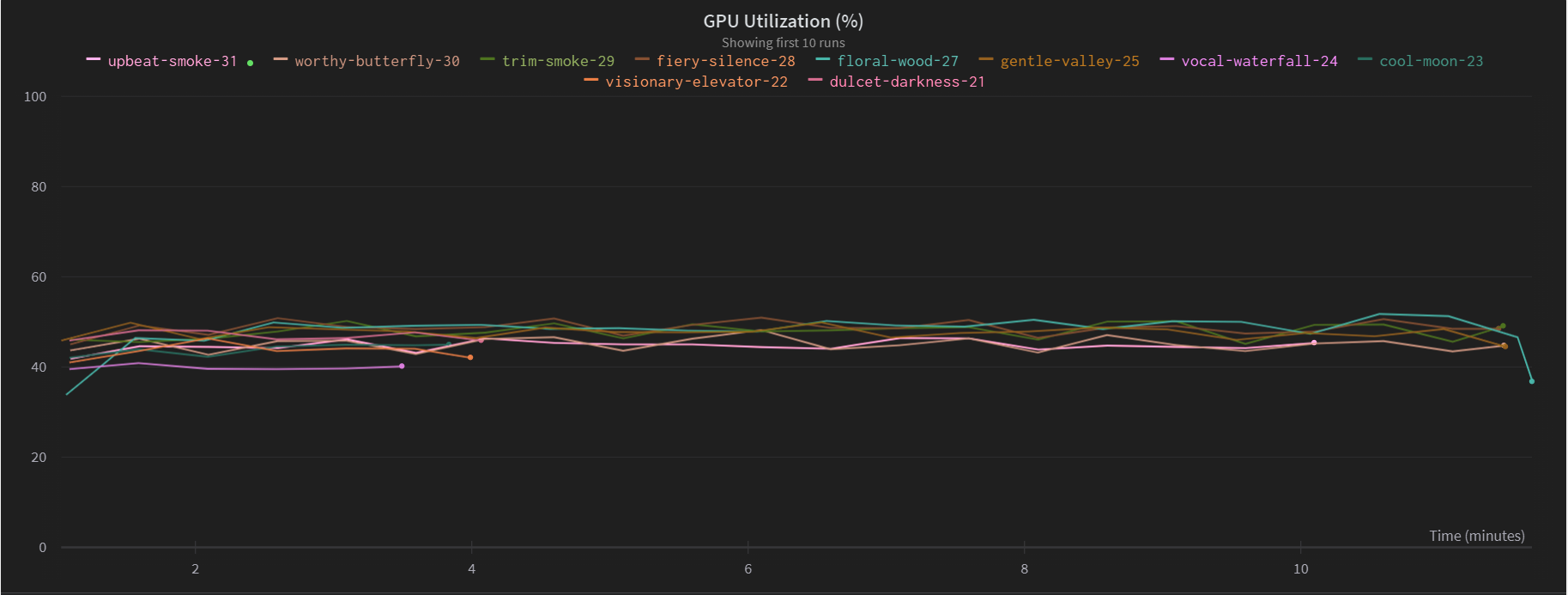


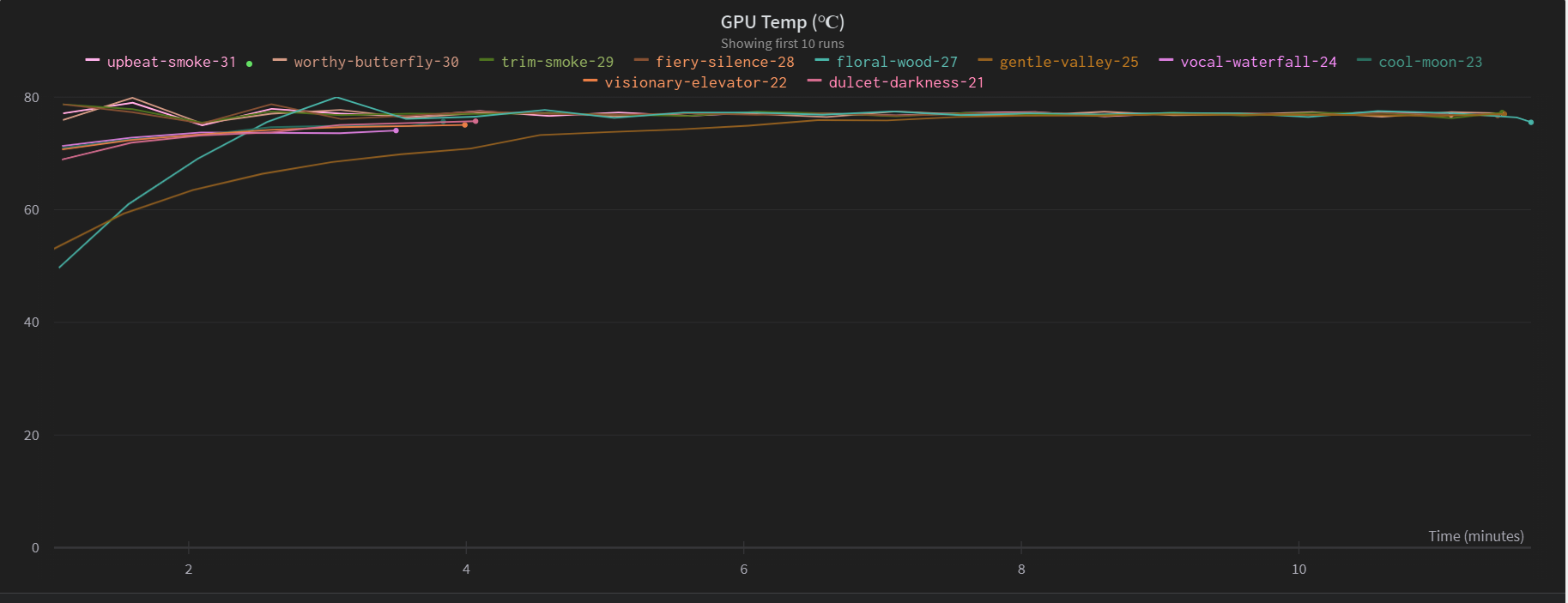
 

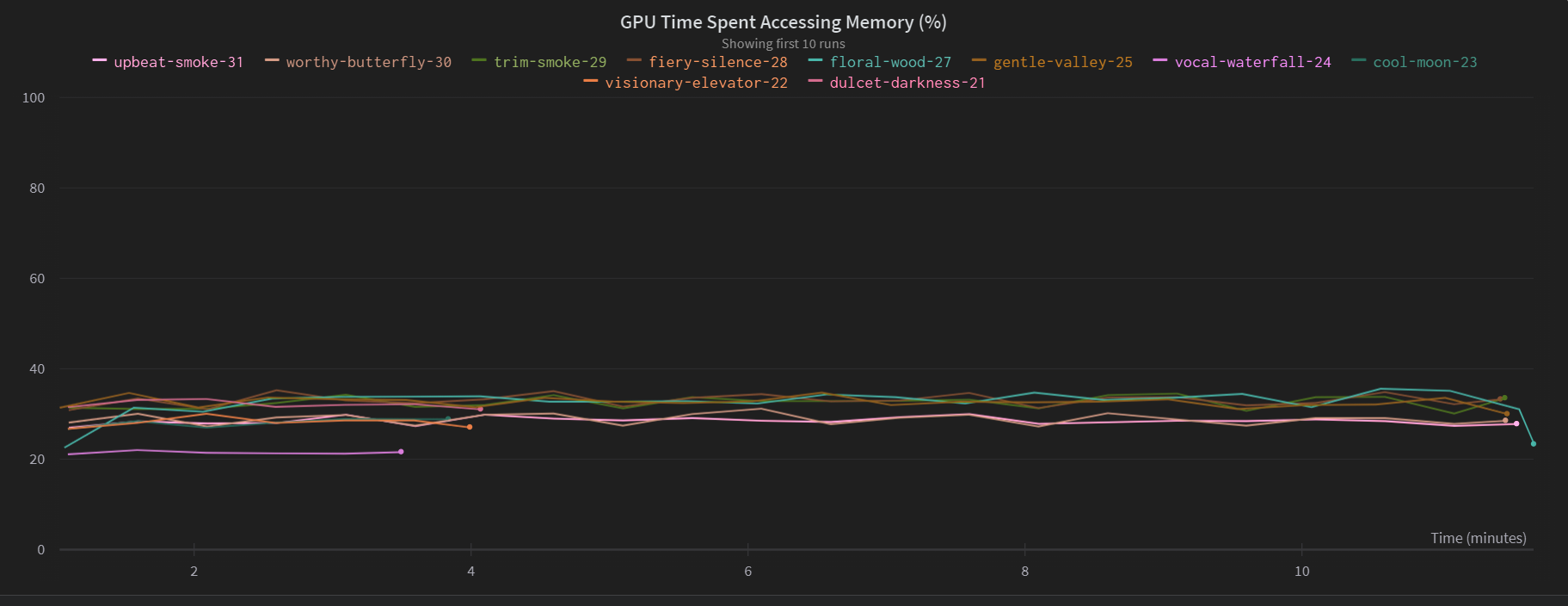


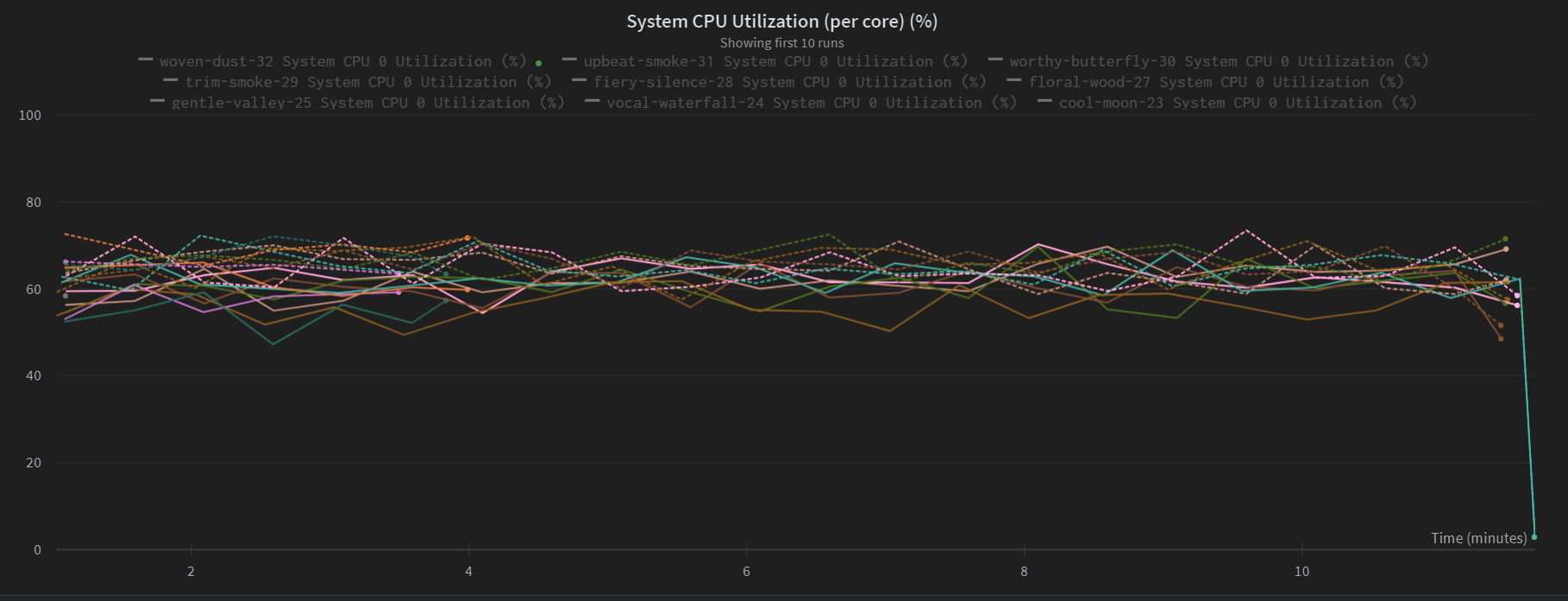
**Observations about GPU and Systems**

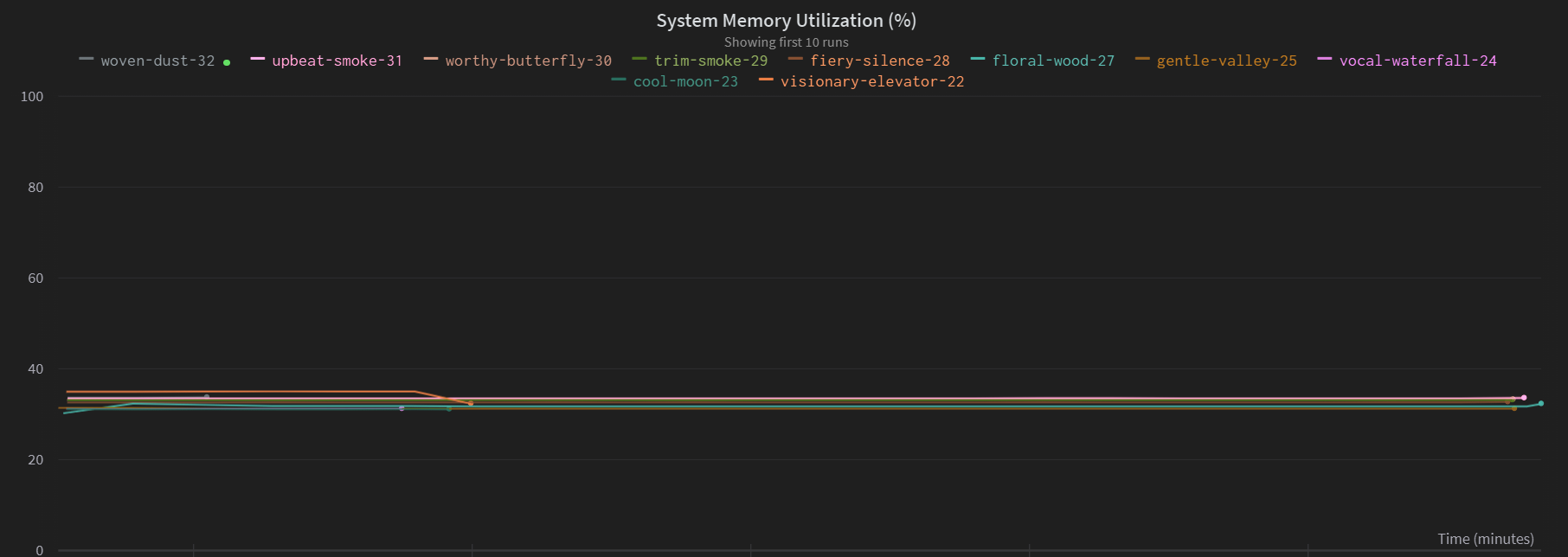


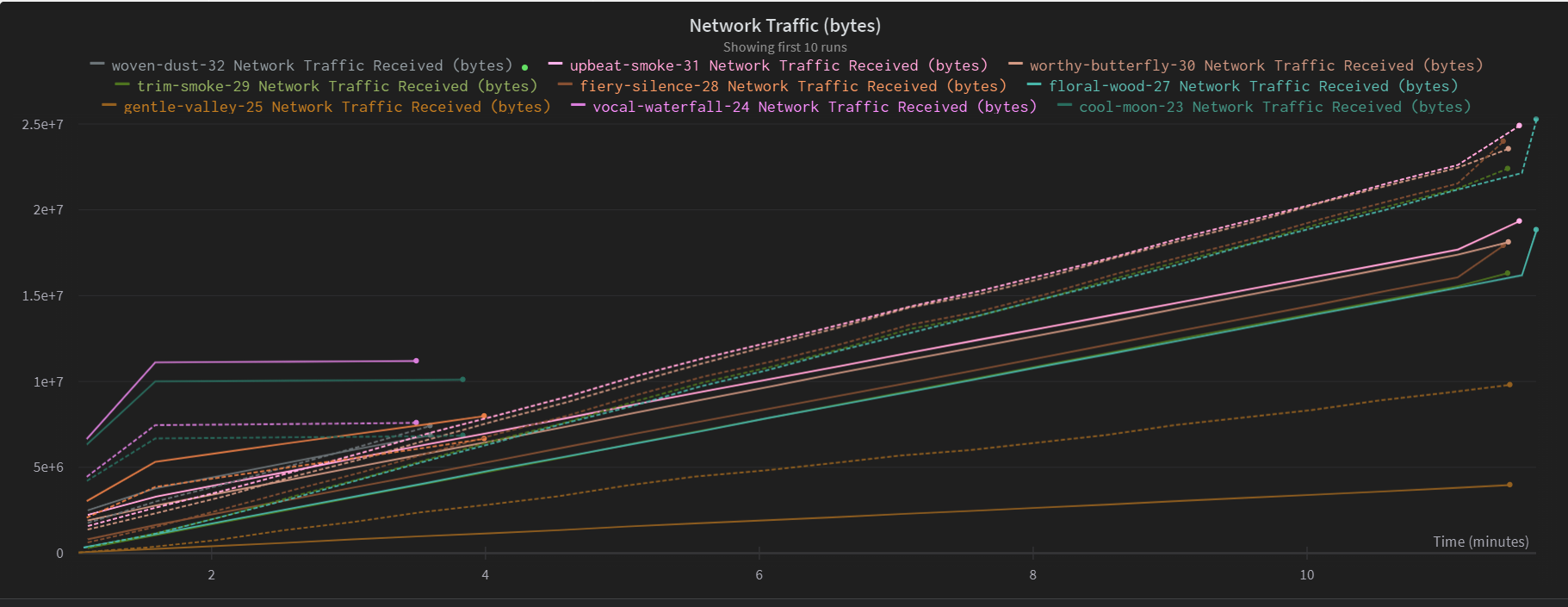




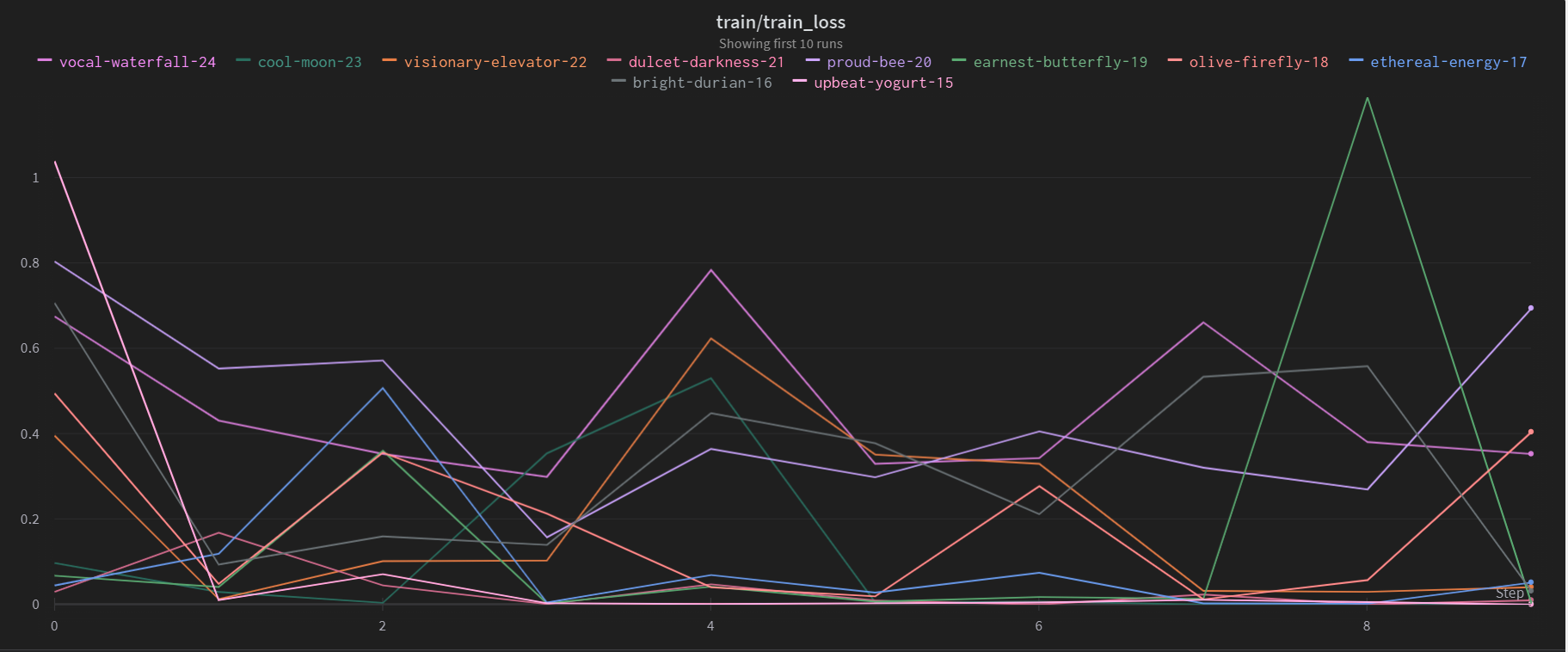


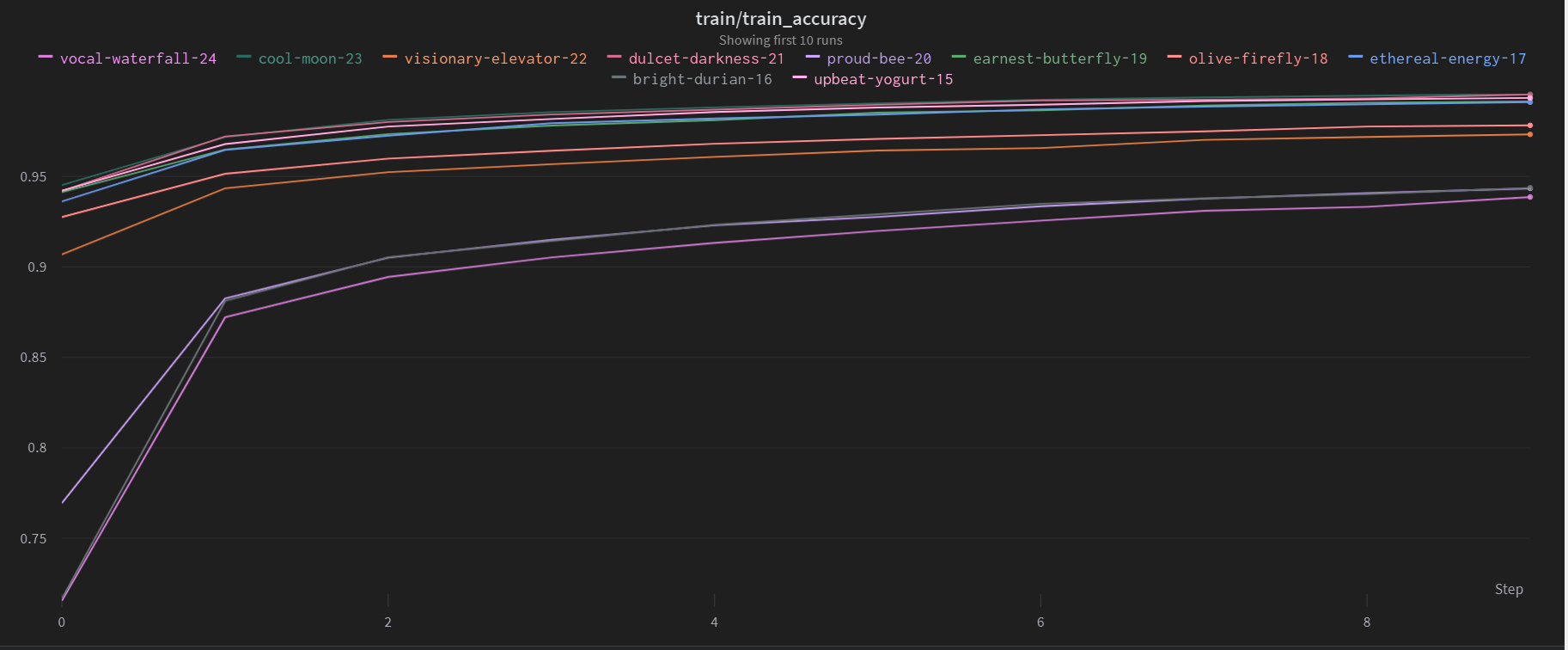


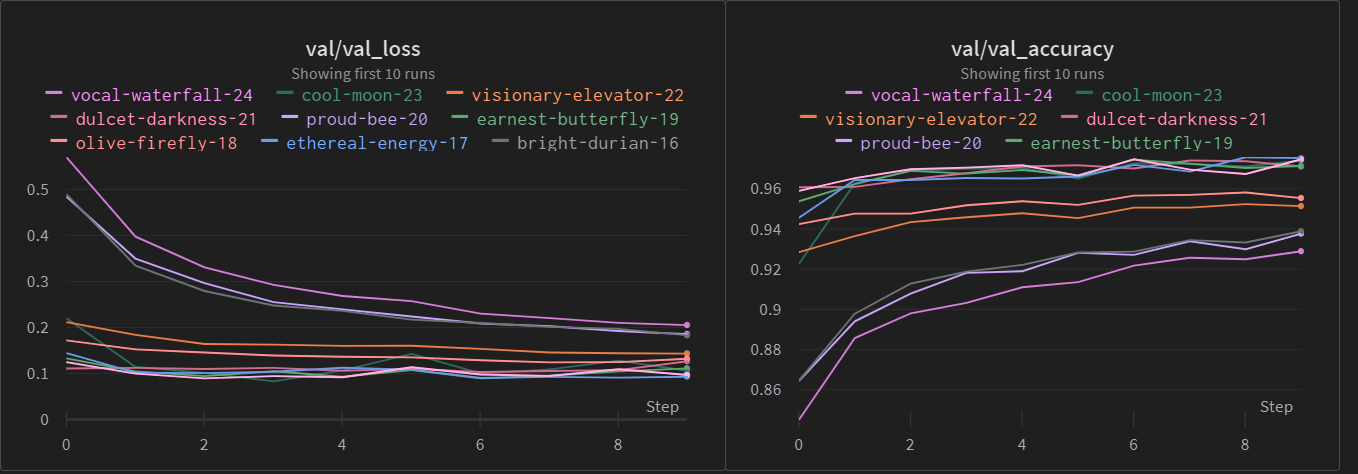




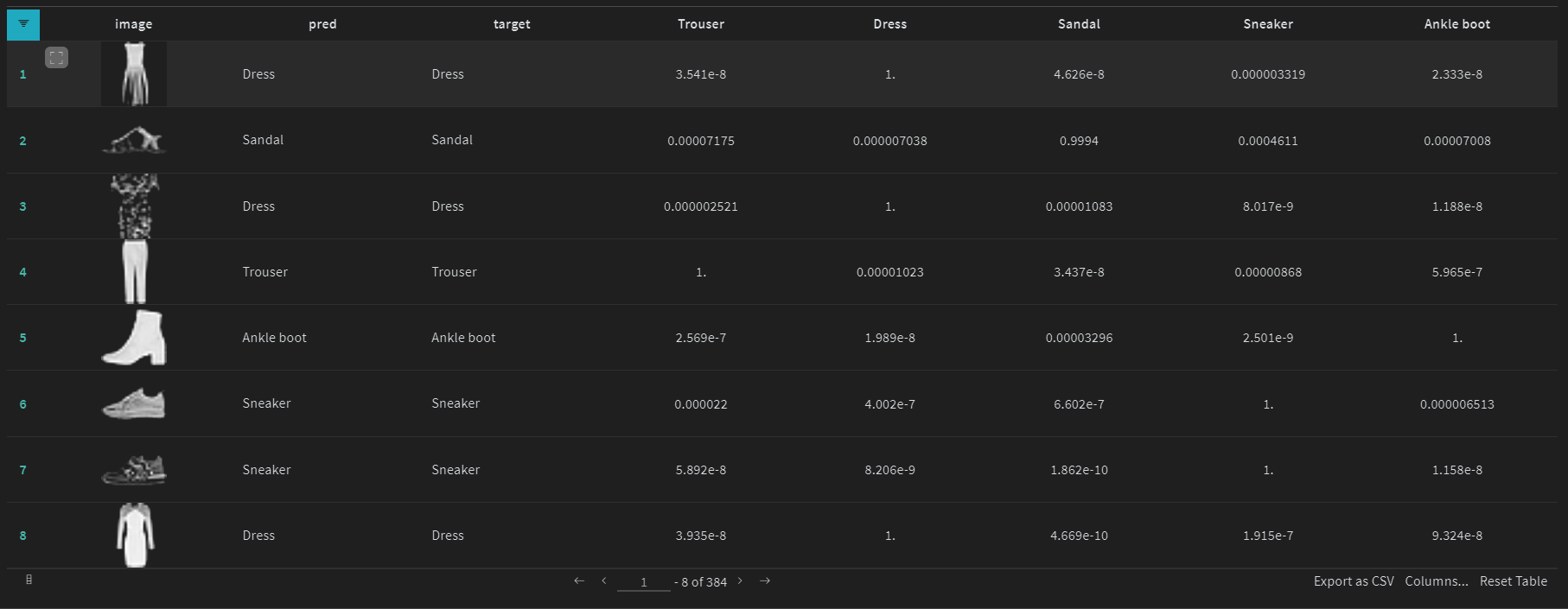
**Logs about training accuracy and loss**

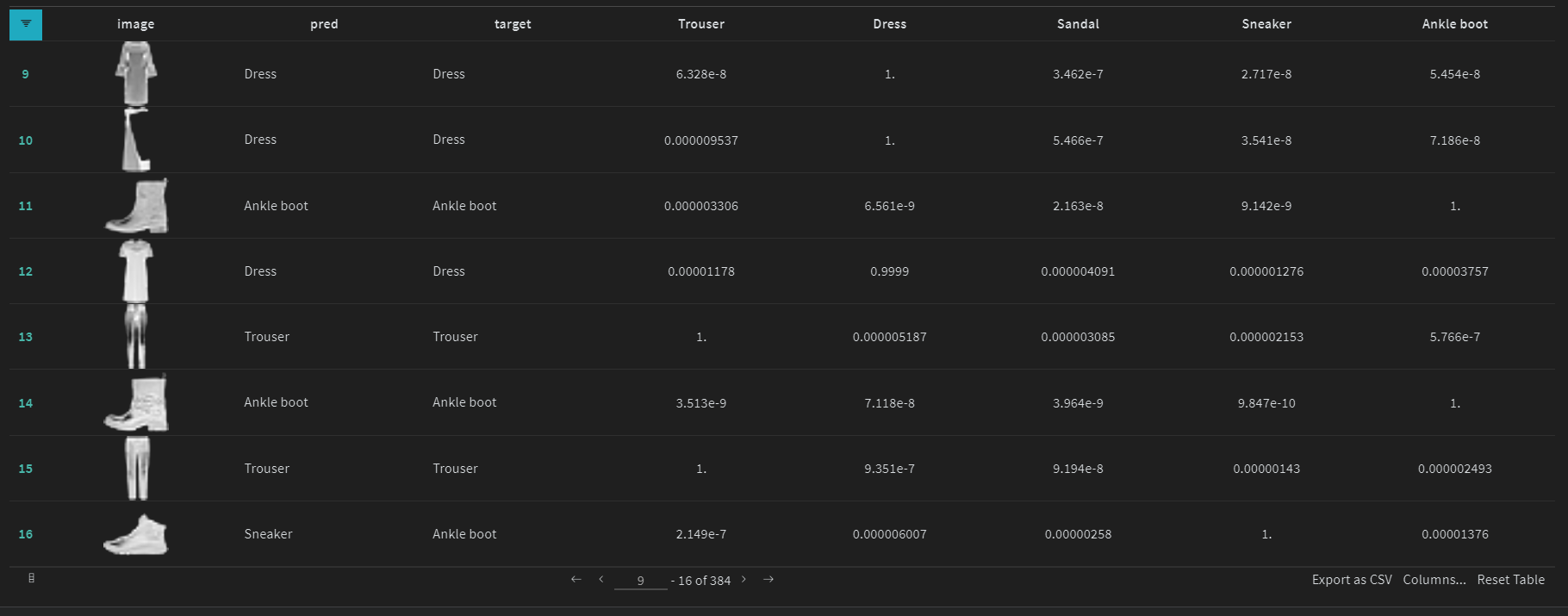


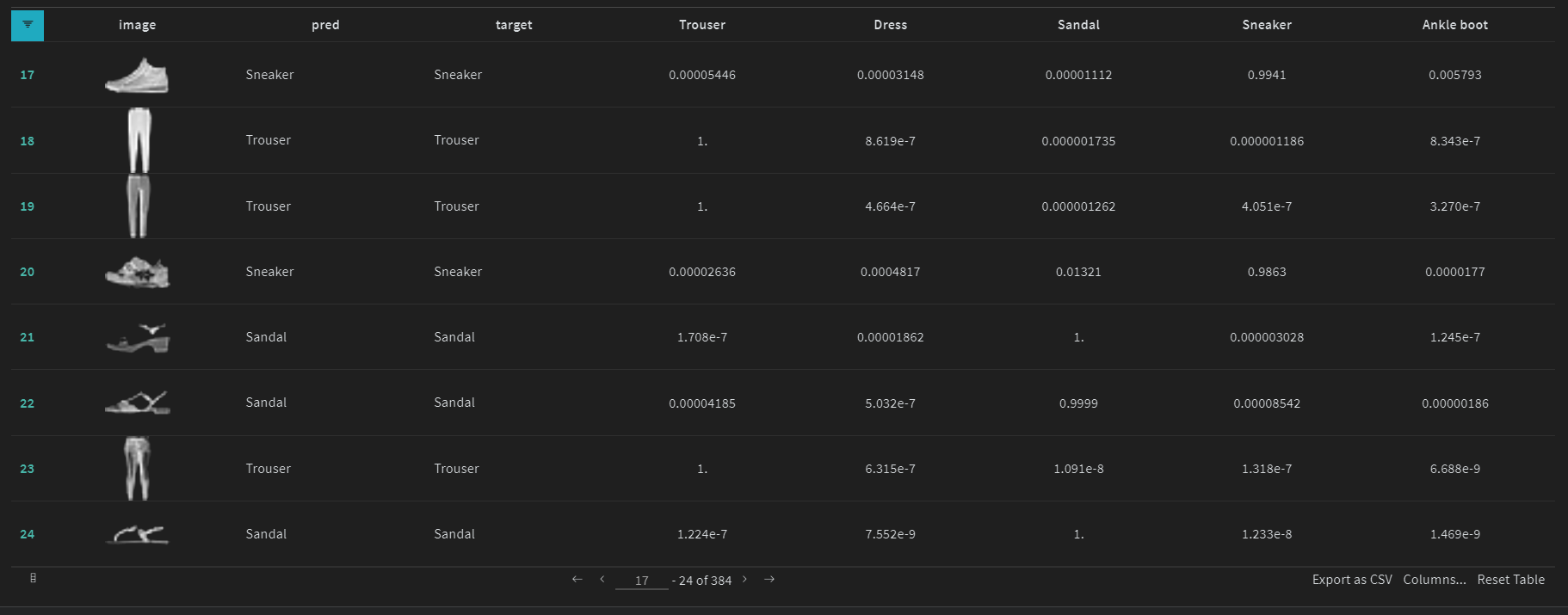




**Visual Result Analysis**







# References:

1. Lectures of DL-Ops class by Professors
2. Implementation Session by Teaching Assistants