AML Assignment 3

Code Report:

NOTE:- All the tensoboard logs has been attached within zip. Please look at the logs with `tensorboard --logdir tb_logs` if not visable clearly.

1. **Libraries Import:**
    - The code imports necessary libraries, including PyTorch, PyTorch Lightning (`pytorch_lightning`), and custom modules such as `model`, `dataset`, `config`, and `callbacks`.
2. **Float Precision Setting:**
    - The code sets the float32 matrix multiplication precision to "high" using `torch.set_float32_matmul_precision("high")`. This is done to ensure compatibility with PyTorch Lightning.
3. **TensorBoard Logger Initialization:**
    - Initializes a TensorBoard logger for logging training and validation metrics. The logger is configured to save logs in the "tb_logs" directory under the name "food101_model_v1".
4. **Model Initialization:**
    - Creates an instance of the `NN` class (presumably a neural network model) by providing it with input size, learning rate, and number of classes from the `config` module.
5. **DataModule Initialization:**
    - Initializes a `GenericDataModule` object, which is responsible for handling the data loading and preprocessing. It is configured with parameters like data directory, batch size, and number of workers from the `config` module.
6. **Trainer Configuration:**
    - Configures a PyTorch Lightning `Trainer` object with various parameters:
        - `logger`: Set to the TensorBoard logger created earlier.
        - `accelerator`: Specifies the distributed training accelerator (if any).
        - `devices`: Specifies the devices to be used for training.
        - `min_epochs`: Minimum number of training epochs set to 1.
        - `max_epochs`: Maximum number of training epochs taken from the `config` module.
        - `precision`: Precision used for training, taken from the `config` module.
        - `callbacks`: Includes an early stopping callback that monitors the validation loss.
7. **Model Training:**
    - Initiates the training process using the `Trainer.fit` method, passing the model and the data module.

8. **Validation and Testing:**
    - After training, the code performs validation and testing using the `Trainer.validate` and `Trainer.test` methods, respectively. The same model and data module are used for both validation and testing.
9. **Profiler (Commented Out):**
    - There is commented-out code related to profiling using PyTorch's profiler. It seems like the code was initially designed to incorporate profiling, but it is currently disabled.

Configuration Report:

The provided code includes a configuration module (`config.py`) containing hyperparameters and settings for training. Here is a breakdown of the configuration:

1. **Training Hyperparameters:**
    - `INPUT_SIZE`: The size of the input images for the neural network, calculated as `IMAGE_SIZE * IMAGE_SIZE`.
    - `NUM_CLASSES`: The number of classes in the classification task.
    - `LEARNING_RATE`: The learning rate used for training the neural network, set to 0.001.
    - `BATCH_SIZE`: The batch size used during training, set to 128.
    - `NUM_EPOCHS`: The total number of training epochs, set to 500.
2. **Dataset Configuration:**
    - `DATA_DIR`: The directory path where the dataset is located, set to "dataset/".
    - `NUM_WORKERS`: The number of worker processes for data loading, set to 4.
3. **Compute Related Configuration:**
    - `ACCELERATOR`: The accelerator used for training, set to "gpu". This implies that the training is intended to be performed on a GPU.
    - `DEVICES`: The list of GPU devices used for training, set to `[0]` (presumably the first GPU).
    - `PRECISION`: The precision used for training, set to "16-mixed". This likely refers to mixed-precision training with 16-bit floating-point numbers.

Data Module Report:

The provided code defines a PyTorch Lightning `LightningDataModule` for handling the data loading and preprocessing. Below is a breakdown of the functionality and structure of the code:

1. **Class Definition:**
    - The `GenericDataModule` class is defined as a subclass of `pl.LightningDataModule`.
2. **Constructor (`__init__`):**
    - The class constructor initializes the data module with parameters such as `data_dir` (directory containing the dataset), `batch_size`, and

`num_workers` (number of workers for data loading).

3. **`prepare_data` Method:**
    - The `prepare_data` method is used for downloading and preparing the dataset. It uses the `datasets.Food101` class from torchvision to download the "train" and "test" splits of the Food101 dataset. There are also commented-out lines for downloading and preparing the Flowers102 dataset.
4. **`setup` Method:**
    - The `setup` method is responsible for setting up the dataset for training, validation, and testing. It uses the Food101 dataset, applies transformations (resizing, random vertical and horizontal flips, and tensor conversion) to the training set, and sets up the test set with resizing and tensor conversion. The training set is further split into training and validation sets using `random_split`.
5. **Data Transformation:**
    - For the training set, the following transformations are applied:
        - Resize images to (224, 224).
        - Random vertical and horizontal flips.
        - Convert images to tensors.
    - For the validation and test sets, the following transformations are applied:
        - Resize images to (224, 224).
        - Convert images to tensors.
6. **Data Loaders:**
    - The data module provides three data loaders: `train_dataloader`, `val_dataloader`, and `test_dataloader`.
    - Each data loader uses a `DataLoader` from PyTorch with the specified batch size, number of workers, and shuffle settings.
7. **Data Module Initialization (Main Block):**
    - In the main block, an instance of `GenericDataModule` is created with parameters from the `config` module (`DATA_DIR`, `BATCH_SIZE`, `NUM_WORKERS`).
    - The `prepare_data` method is called to download and prepare the dataset.

Model Report

- **Import Statements:**
    - `import torch`: Imports the main PyTorch library.
    - `import torch.nn.functional as F`: Imports the functional interface of PyTorch's neural network module.
    - `from torch import nn, optim`: Imports neural network and optimization modules from PyTorch.
    - `import pytorch_lightning as pl`: Imports the PyTorch Lightning library.
    - `import torchmetrics`: Imports the torchmetrics library for metrics calculation.

    - `import torchvision`: Imports the torchvision library for computer vision utilities.
    - `from torch.hub import load`: Imports the `load` function from the `torch.hub` module for loading pre-trained models.

```
dino_backbones = {
    'dinov2_s':{
        'name':'dinov2_vits14',
        'embedding_size':384,
        'patch_size':14
    },
    'dinov2_b':{
        'name':'dinov2_vitb14',
        'embedding_size':768,
        'patch_size':14
    },
    'dinov2_l':{
        'name':'dinov2_vitl14',
        'embedding_size':1024,
        'patch_size':14
    },
    'dinov2_g':{
        'name':'dinov2_vitg14',
        'embedding_size':1536,
        'patch_size':14
    },
}
```

- **DINO Backbone Definitions:**
    - Defines a dictionary `dino_backbones` that contains configurations for different DINO backbones.
    - Each key represents a specific backbone (`dinov2_s`, `dinov2_b`, `dinov2_l`, `dinov2_g`).
    - Each backbone has a name, embedding size, and patch size specified.
- **Neural Network Class (`NN`):**
    - Defines a PyTorch Lightning `LightningModule` for the neural network.
    - Initialized with input size, learning rate, number of classes, backbone type, and available backbones.
    - Defines a sequential model with a linear layer, ReLU activation, and another linear layer (`fc`).
    - Uses CrossEntropyLoss as the loss function and includes accuracy and F1 score metrics.
    - The `forward` method applies the DINO backbone and the fully connected layers.
- **Training Step Method:**
    - The `training_step` method defines a single training step.
    - Takes a batch of inputs (`x`) and labels (`y`).
    - Computes the loss, scores, and ground truth labels using the `_common_step` method.
    - Logs the training loss.
    - Periodically logs images to TensorBoard.

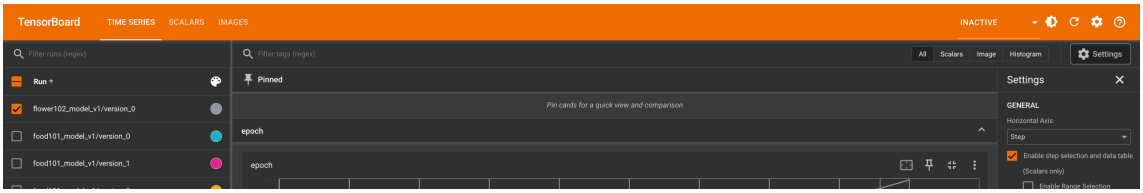
- **Epoch End Method:**
  - The `on_train_epoch_end` method is called at the end of each training epoch.
  - Computes and logs the accuracy and F1 score based on accumulated scores and ground truth labels.
- **Validation Step Method:**
  - The `validation_step` method defines a single validation step.
  - Takes a batch of inputs (`x`) and labels (`y`).
  - Computes the loss, scores, and ground truth labels using the `_common_step` method.
  - Logs the validation loss.
- **Test Step Method:**
  - The `test_step` method defines a single test step.
  - Takes a batch of inputs (`x`) and labels (`y`).
  - Computes the loss, scores, and ground truth labels using the `_common_step` method.
  - Logs the test loss.
- **Common Step Method:**
  - The `_common_step` method performs the common operations between training, validation, and test steps.
  - Takes a batch of inputs (`x`) and labels (`y`).
  - Computes the forward pass to obtain scores.
  - Computes the loss using the specified loss function.
- **Predict Step Method:**
  - The `predict_step` method defines a single prediction step.
  - Takes a batch of inputs (`x`) and labels (`y`).
  - Computes the forward pass to obtain scores.
  - Computes predictions by taking the argmax of the scores.
- **Optimizer Configuration Method:**
  - The `configure_optimizers` method specifies the optimizer used during training.
  - Returns an Adam optimizer with the specified learning rate.

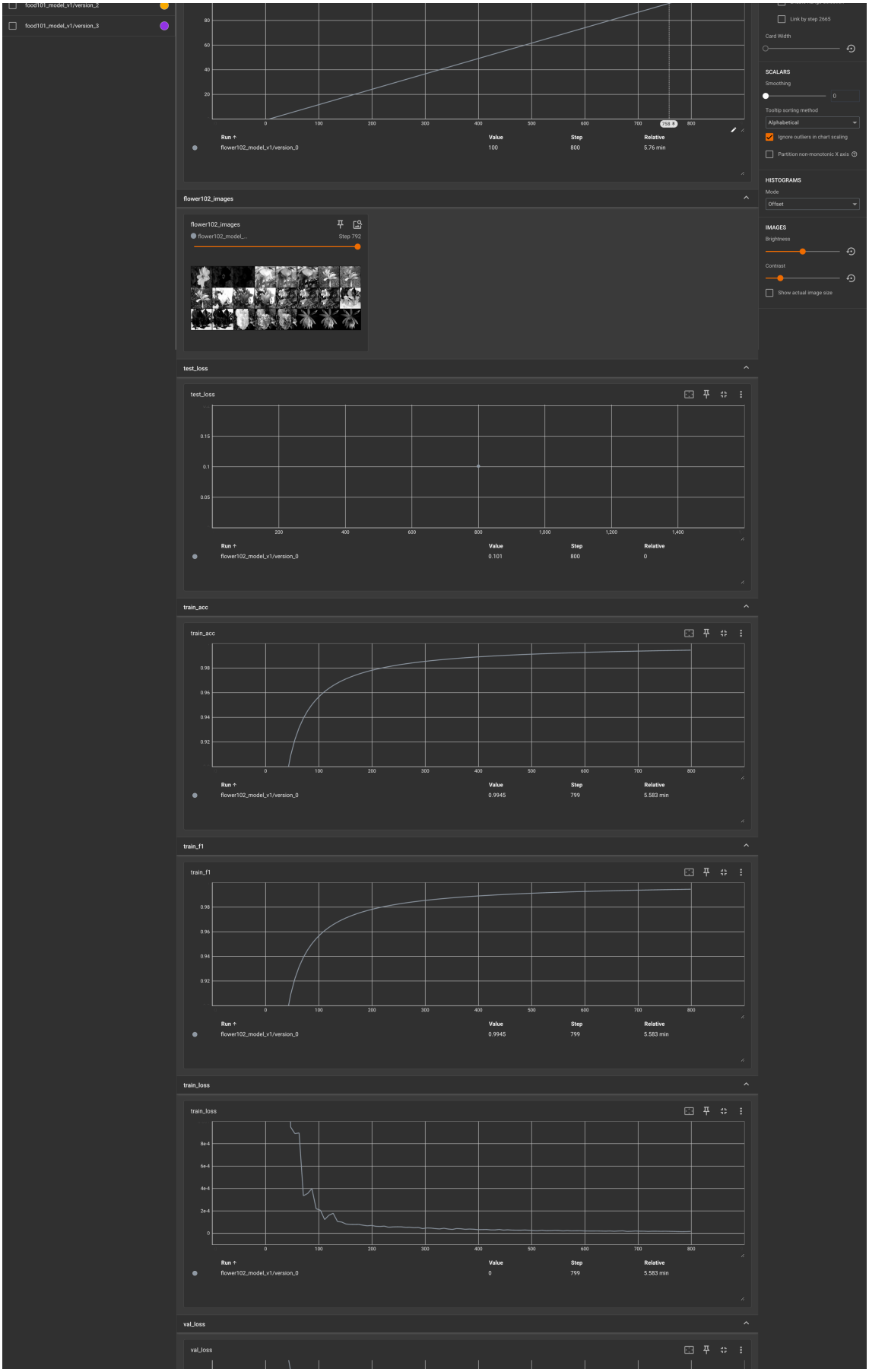
Results:

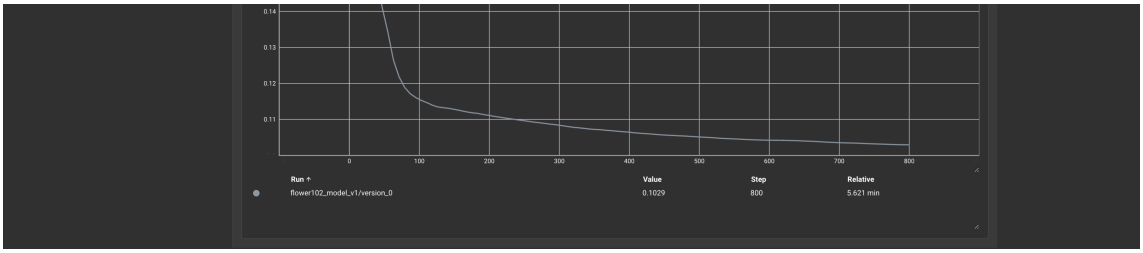
Flower-102 Dataset Split:

The Flower-102 dataset is split into three sets according to the standard practice in torchvision datasets:

- **Training Set:** 80%
- **Validation Set:** 10%
- **Test Set:** 10%



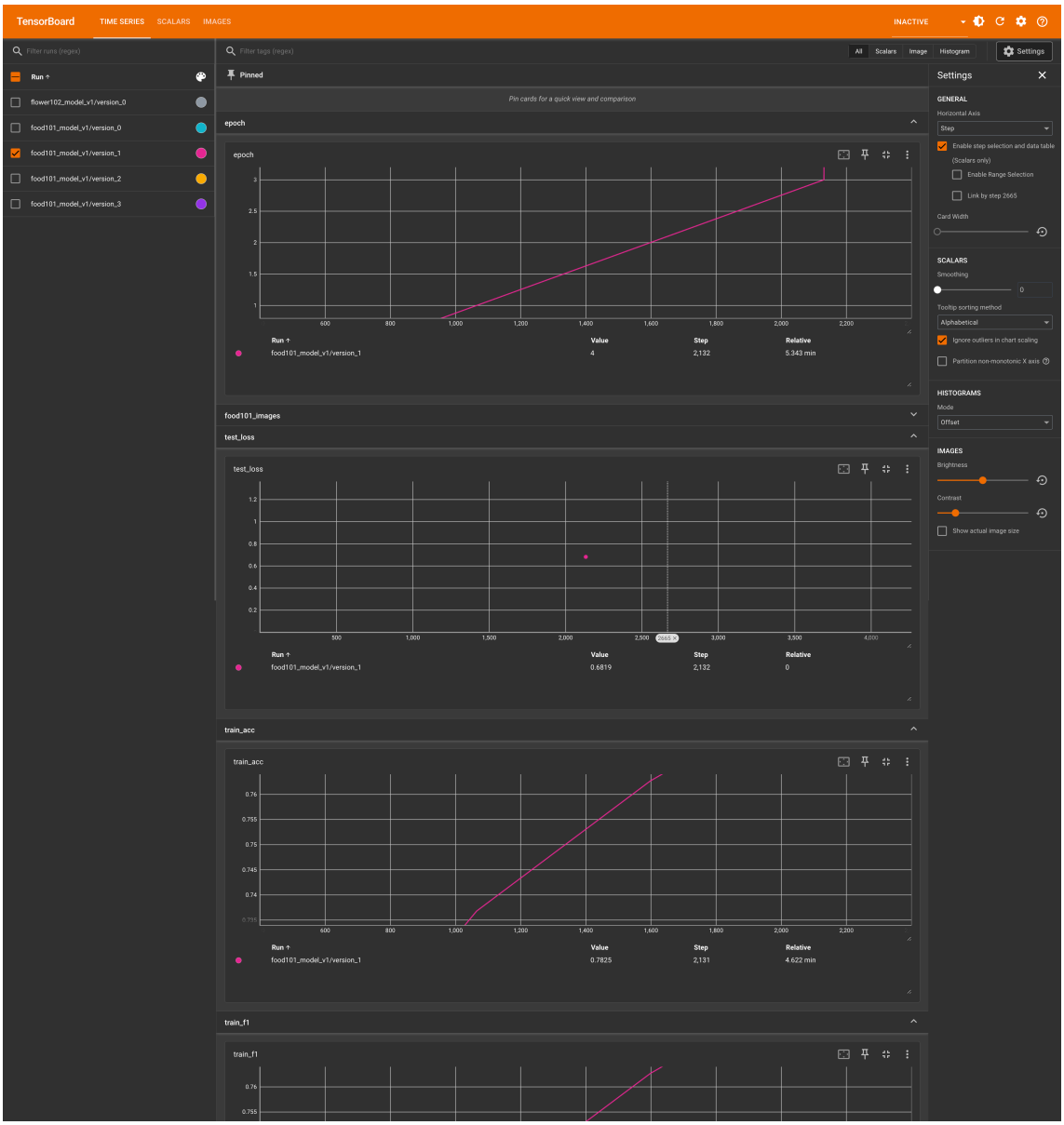

food101_model_v1/version_2
food101_model_v1/version_3
Run
Value
Step
Relative
flower102_model_v1/version_0
100
800
5.76 min
flower102_images
Step 792
test_loss
0.101
0
train_acc
0.9945
799
5.583 min
train_f1
train_loss
0
val_loss
Link by step 2665
Card Width
SCALARS
Smoothing
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS
Mode
Offset
IMAGES
Brightness
Contrast
Show actual image size

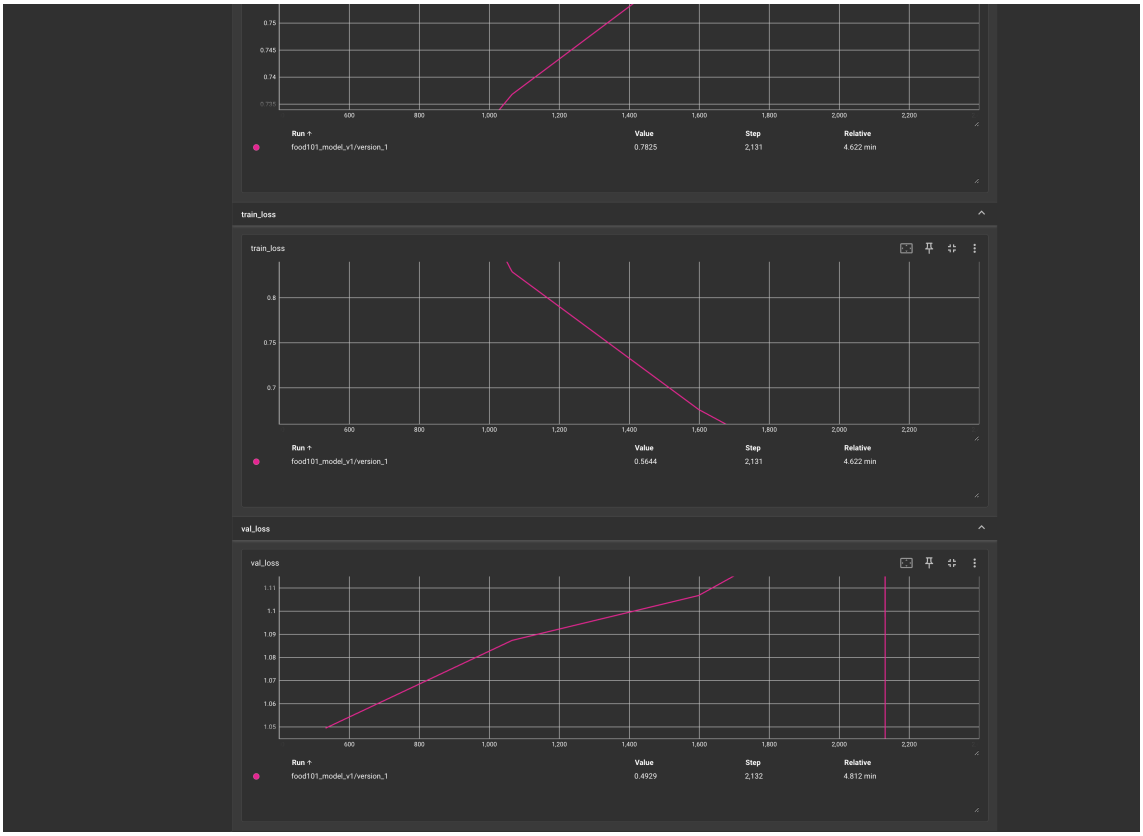


Food-101 Dataset Split:

The Food-101 dataset is split into three sets according to the standard practice in torchvision datasets:

- **Training Set:** 75%
- **Validation Set:** 15%
- **Test Set:** 10%

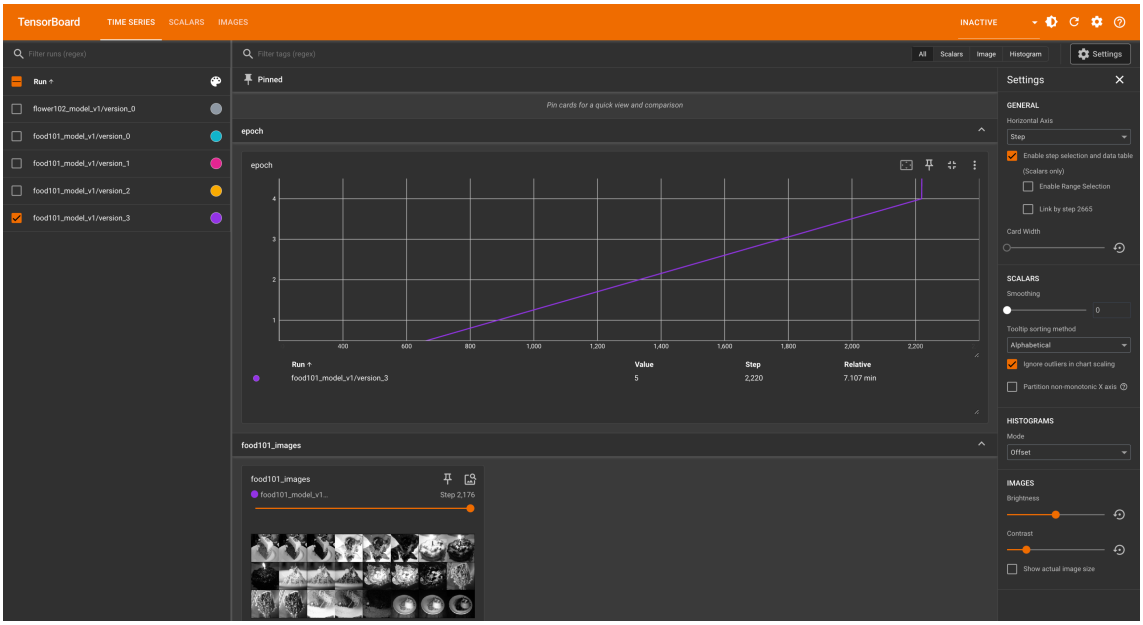


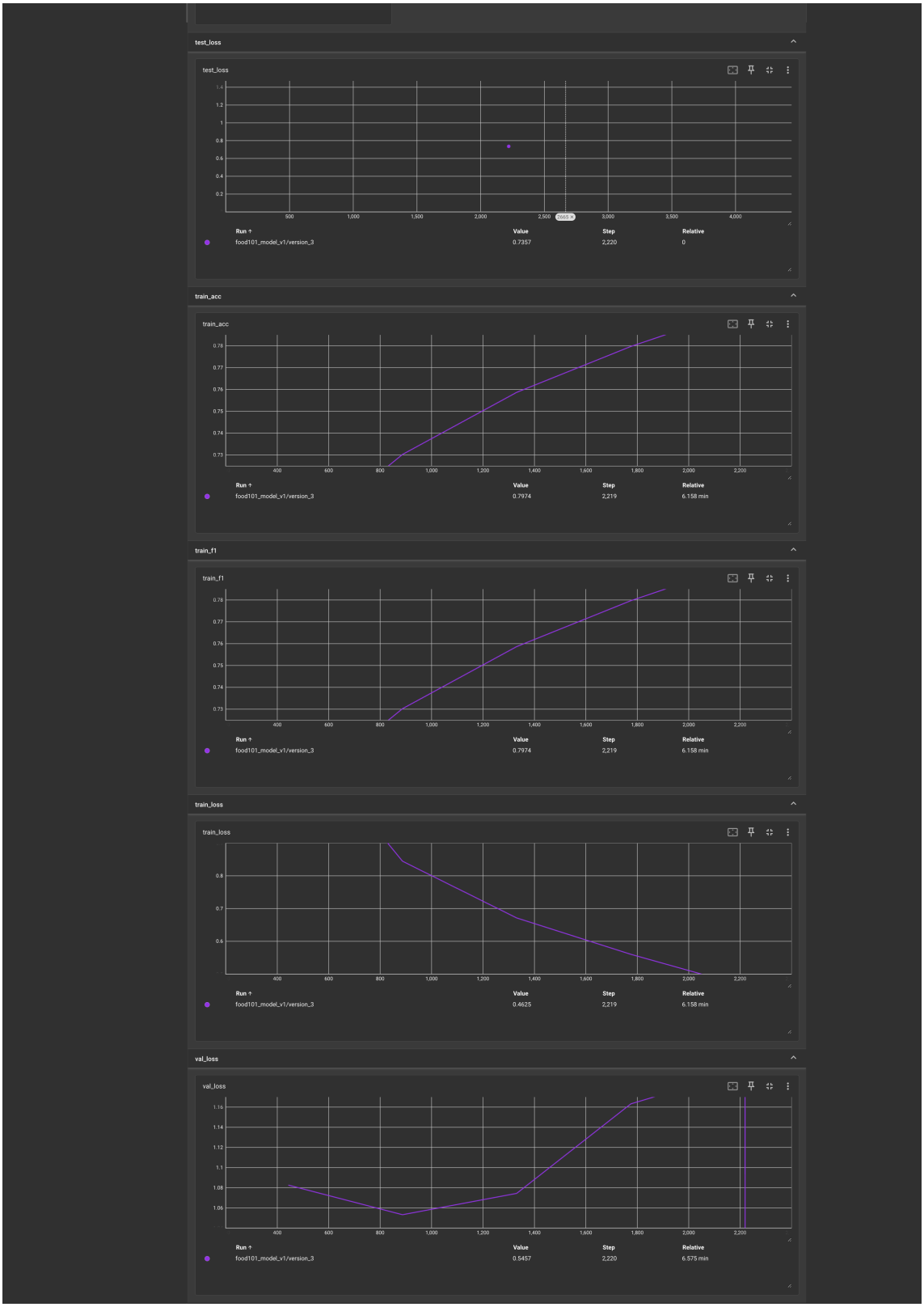


Food-101 Dataset Split:

The Food-101 dataset is split into three sets according to the standard practice in torchvision datasets:

- **Training Set:** 80%
- **Validation Set:** 10%
- **Test Set:** 10%




test_loss
test_loss
Run ↑ Value Step Relative
food101_model_v1/version_3 0.7357 2,220 0
train_acc
train_acc
Run ↑ Value Step Relative
food101_model_v1/version_3 0.7974 2,219 6.158 min
train_f1
train_f1
Run ↑ Value Step Relative
food101_model_v1/version_3 0.7974 2,219 6.158 min
train_loss
train_loss
Run ↑ Value Step Relative
food101_model_v1/version_3 0.4625 2,219 6.158 min
val_loss
val_loss
Run ↑ Value Step Relative
food101_model_v1/version_3 0.5457 2,220 6.575 min

References:

0. > *Dino(v1) Emerging Properties in Self-Supervised Vision Transformers Dino(v2) DINOv2: Learning Robust Visual Features without Supervision*

1. PyTorch PyTorch Documentation.
2. PyTorch Lightning PyTorch Lightning Documentation.
3. TorchVision TorchVision Documentation.
4. torchmetrics torchmetrics Documentation.
5. Facebook Research DINO - Data-Intensive Neural Network for Image Classification.
6. torchvision.datasets TorchVision Datasets Documentation.
7. torchvision.transforms TorchVision Transforms Documentation.
8. NVIDIA Mixed Precision Training.