DAI Assignment-2 Report

Task: Replicate ViT: Using the CIFAR10 dataset and protocol

Note: The code and Tensorboard log directory are attached along with the report in the zipped folder submitted.

Objectives:

- state-of-the-art performing model on computer vision.
- Explain the type of bias you observed.
- mitigate the bias you found
- Another approach of your own to mitigate the bias is using the DATA method (Pre-Processing) & ALGORITHMIC method to alter the loss function or use multi-tasking.
- Compare the bias mitigation techniques.
- Report the changes you observed before and after applying bias mitigation techniques.

Procedure:

- The first block of code installs the relevant libraries. Specifically, it installs the transformers, datasets, and pytorch-lightning libraries using the pip command.
- The code then loads the CIFAR-10 dataset using the load_dataset() function from the datasets library. It loads a small portion of the dataset for demonstration purposes. The training set is split into a training and validation set using the train_test_split() function.
- The features property of the train_ds dataset is printed to the console, which displays information about the dataset, including the names of the features and the types of data they contain.
- The code then displays an example image and label from the training dataset.
- The id2label and label2id dictionaries are created to map between class indices and class names.
- The code then defines two data preprocessing functions: train_transforms() and val_transforms(). These functions use the ViTImageProcessor and torchvision libraries to perform on-the-fly data augmentation, which involves random cropping, resizing, flipping, and normalization of the images. The resulting pixel values are stored in the pixel_values key of each example.
- The set_transform() method is called on the training, validation, and test datasets to apply the preprocessing functions to each example.
- PyTorch DataLoader objects are created for the training, validation, and test datasets using the collate_fn() function to stack the pixel values and labels into batches.
- The code defines a custom LightningModule class called ViTClassifier, which consists of a pre-trained Vision Transformer model and a linear classification layer.
- The forward() method of the ViTClassifier class takes a batch of images as input and returns the logits of the predicted class probabilities.
- The training_step() method of the ViTClassifier class takes a batch of training data as input, performs forward propagation, computes the loss using the cross-entropy loss function, and logs the loss to the console.

- The validation_step() method of the ViTClassifier class takes a batch of validation data as input, performs forward propagation, computes the loss and accuracy, and returns a dictionary containing the loss and accuracy.
- The validation_epoch_end() method of the ViTClassifier class aggregates the validation loss and accuracy across all batches and logs the average values to the console.
- The configure_optimizers() method of the ViTClassifier class defines the optimizer used for training. In this case, it uses the Adam optimizer with a learning rate of 5e-5.
- The Trainer class from the pytorch-lightning library is instantiated with various arguments, including the ViTClassifier model, the training and validation dataloaders, and the number of epochs.
- The fit() method of the Trainer class is called to train the model on the CIFAR-10 dataset. During training, the training_step() and validation_step() methods of the ViTClassifier class are called for each batch of training and validation data, respectively. The validation_epoch_end() method is called at the end of each validation epoch. The progress of training is logged to the console, including the current epoch, training loss, validation loss

Results:

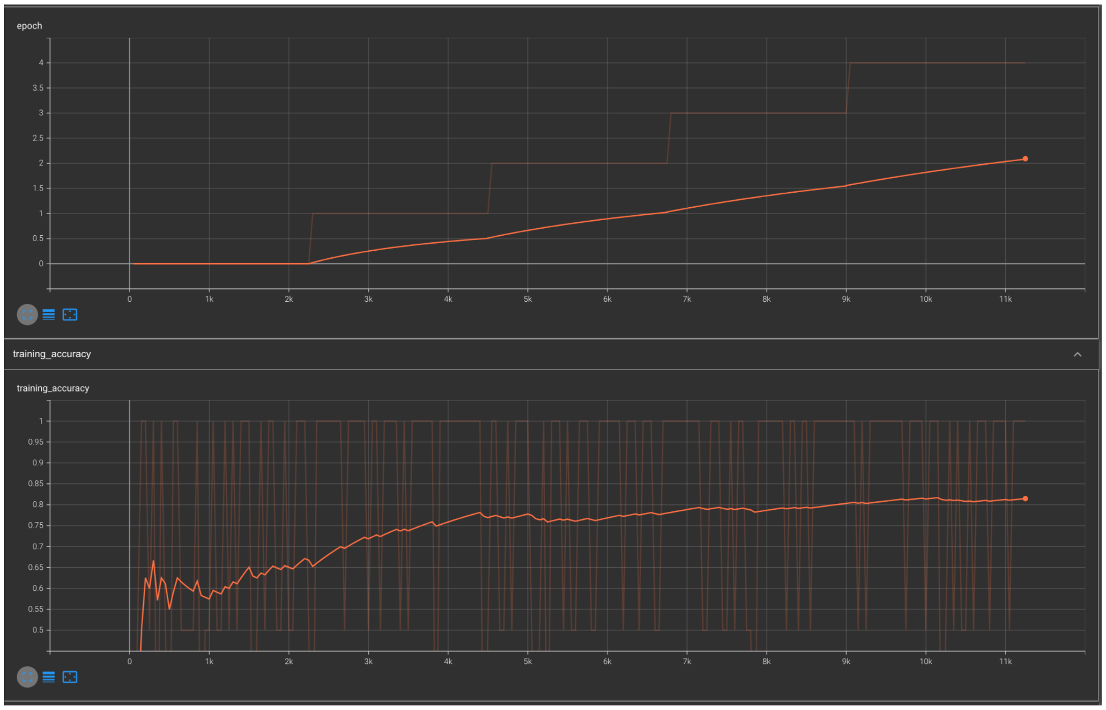


Fig: epochs and training accuracy of the model.

Note:- The spikes are formed due to oscillations in loss minimization.

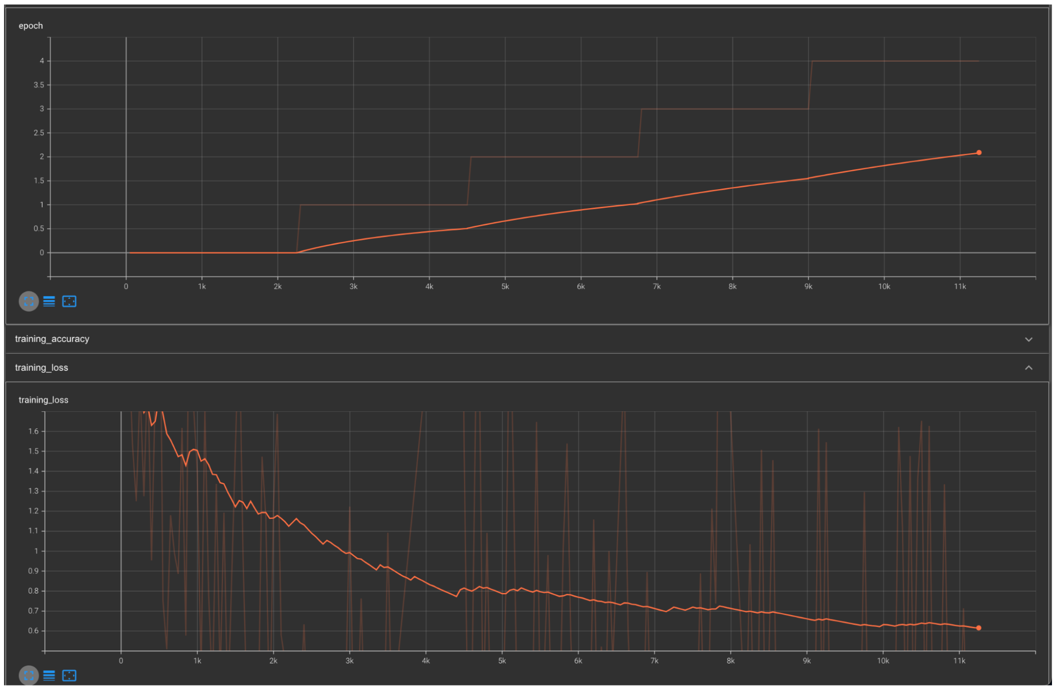


Fig: Epoch and Loss Curve

Note:- The spikes are formed due to oscillations in loss minimization.

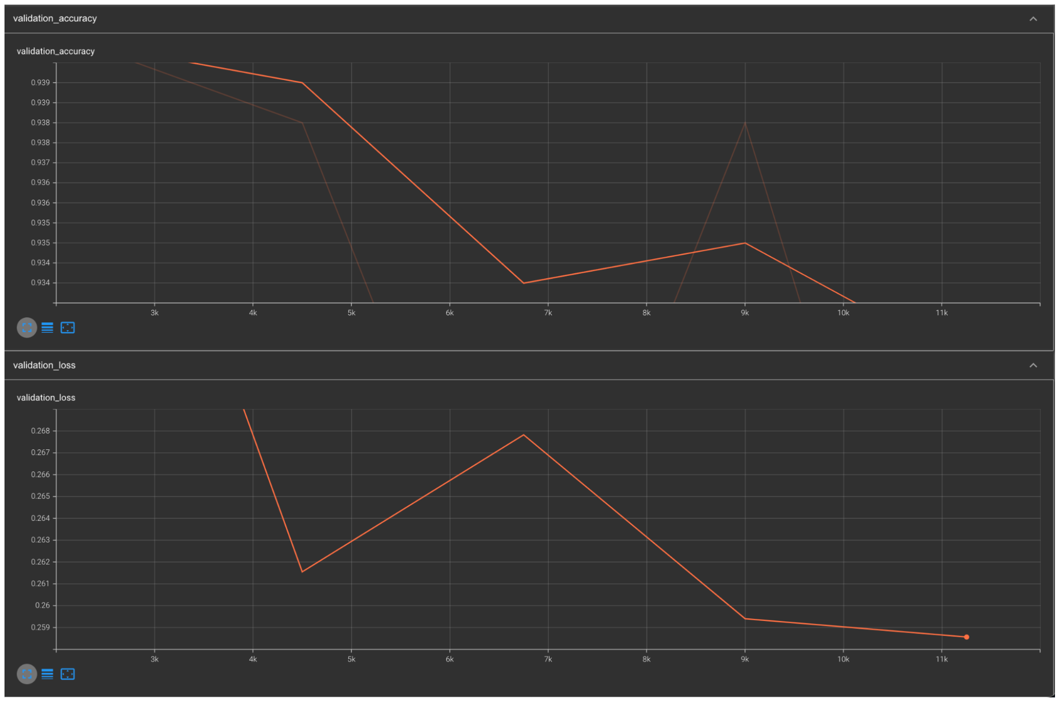


Fig: validation loss and accuracy.

Note: We can Observe that in the end, accuracy is very low. This issue can be resolved by further training or by applying early stopping.

Bias In ViT:

Vision Transformer (ViT) is a type of deep learning model for computer vision tasks that has gained a lot of attention recently. The CIFAR-10 dataset is a popular benchmark for image classification tasks, and ViT has been shown to perform well on this dataset.

However, like all machine learning models, ViT can be prone to bias. Bias in this context refers to systematic errors or inaccuracies in the model's predictions that result from the data it was trained on. In other words, bias can occur when the training data is not representative of the real world and contains patterns or features that the model learns to rely on, leading to incorrect predictions when faced with new data.

To detect and address bias in ViT trained on CIFAR-10, several approaches can be taken, such as:

- Conducting a bias audit: This involves examining the data used to train the model for any patterns or biases that could influence the model's predictions. For example, if the dataset contains mostly images of light-skinned people, the model may struggle to classify images of people with darker skin tones accurately.

- Augmenting the training data: Adding more diverse and representative images to the training dataset can help reduce bias and improve the model's accuracy on a wider range of inputs.

- Fine-tuning the model: Fine-tuning involves re-training the model on a smaller dataset that includes a mix of representative and biased data. This can help the model learn to recognize and correct for biases in its predictions.

- The bias is mostly generated due to dataset imbalance, or model learning incorrectly, the ViT is also prone to it if not properly trained.

Bias Mitigation:
Various techniques can be used to mitigate bias in a Vision Transformer (ViT) model trained on the CIFAR-10 dataset. Here are some possible approaches:

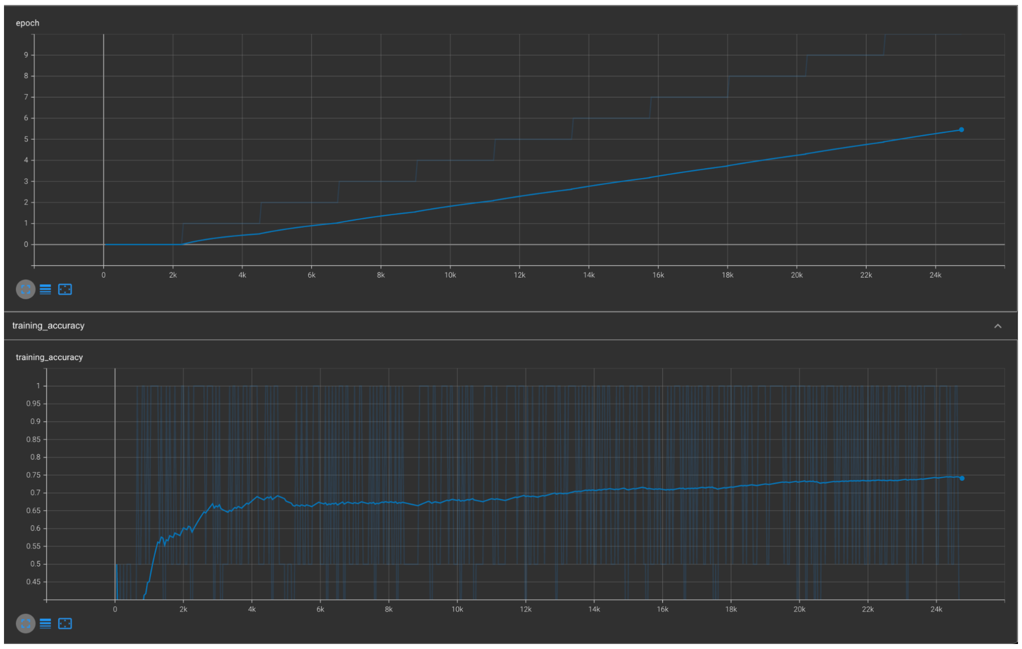
- Data augmentation: One way to address bias is by adding more diverse and representative images to the training dataset. This can be done through data augmentation techniques such as rotation, flipping, and scaling. For example, if the dataset contains mostly images of people with light skin tones, data augmentation techniques can be used to generate images of people with darker skin tones, thereby reducing bias.

- Adversarial training: Adversarial training involves adding a small perturbation to the input data during training in order to make the model more robust to changes in the input. This can help mitigate bias by making the model less sensitive to certain features of the data that may be over-represented or under-represented in the training set.

- Class weighting: Another way to address bias is by adjusting the weights assigned to each class during training. If certain classes are under-represented in the training data, the model may have difficulty correctly classifying them. By assigning higher weights to these classes during training, the model can learn to pay more attention to them and improve its accuracy in these classes.

- Fairness constraints: Fairness constraints can be used to ensure that the model's predictions are unbiased with respect to certain protected attributes such as race, gender, or age. These constraints can be added to the loss function during training to penalize the model for making biased predictions.

Results: After applying the Data Augmentation and Custom Loss Function.



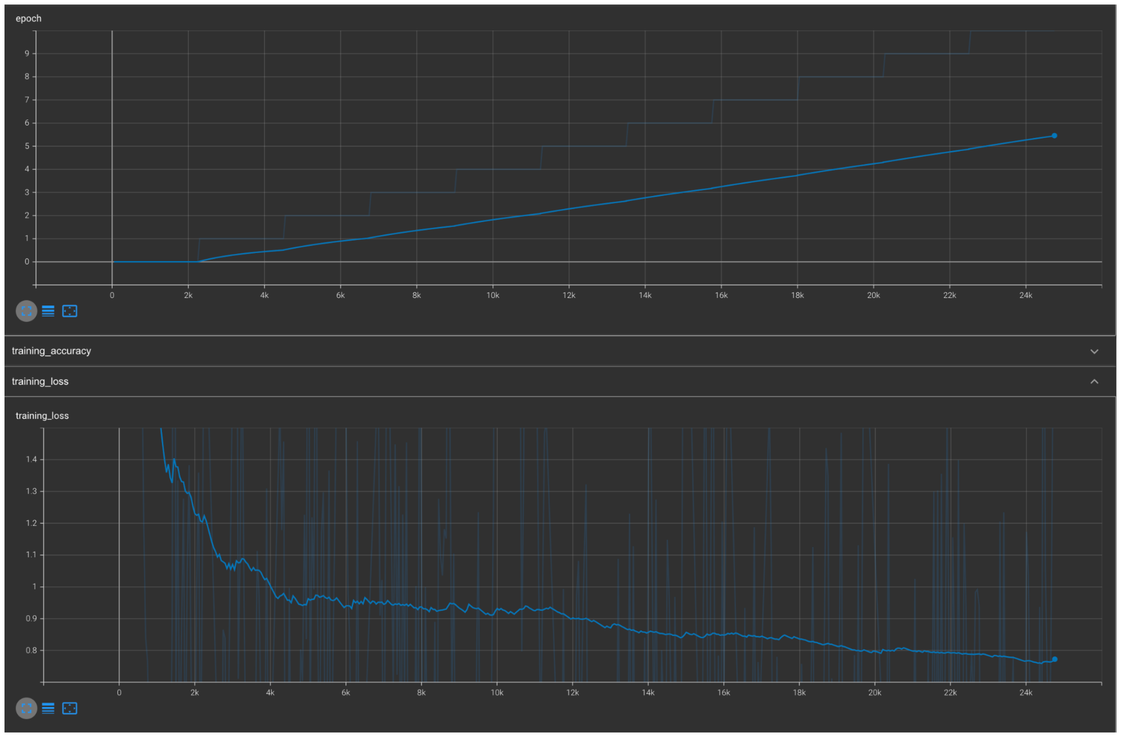


Fig: Training loss with data augmentation

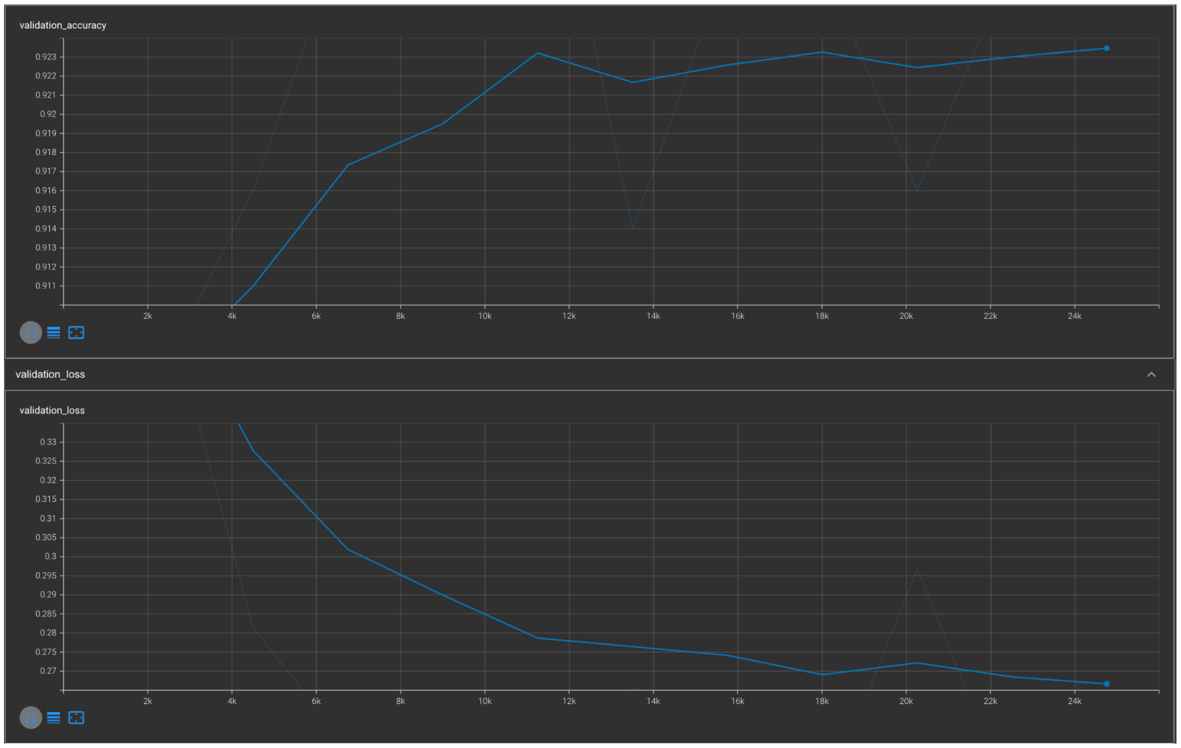
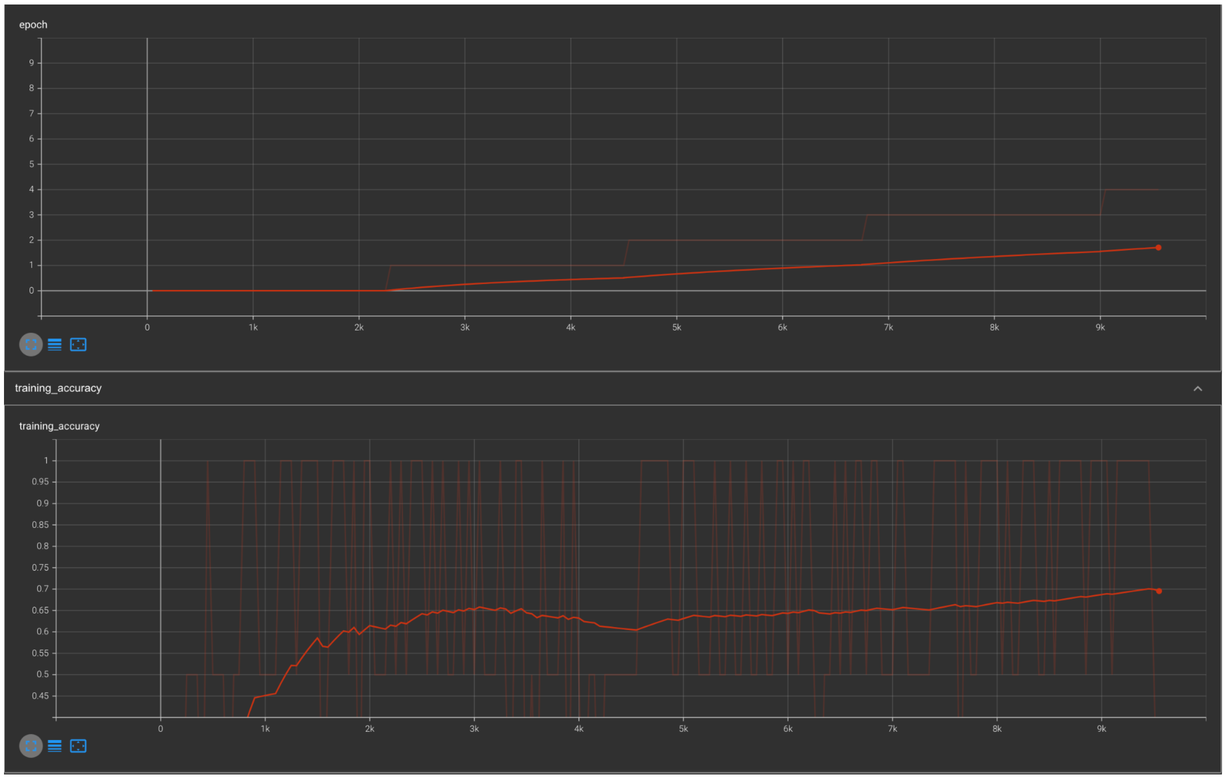
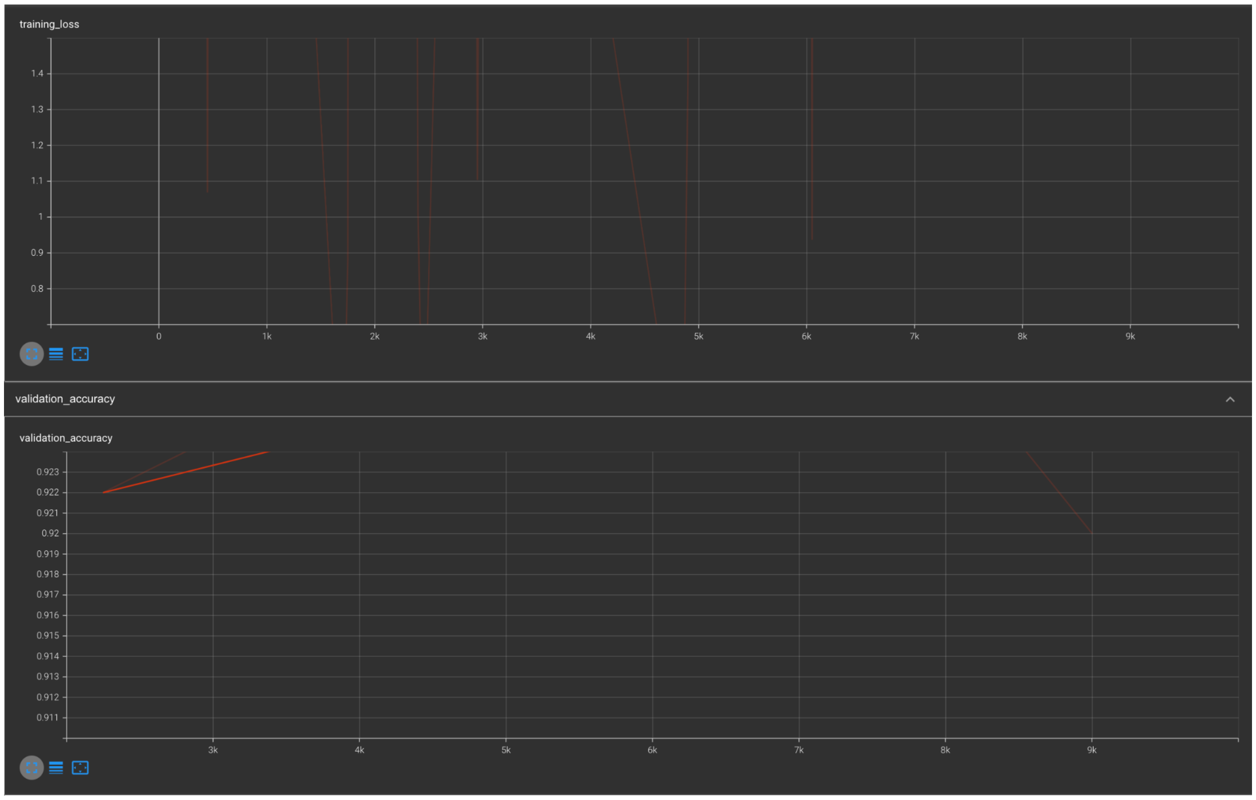


Fig: Validation Loss + Validation Accuracy with data augmentation





Figs: All Metrics for training with the custom loss function

Changes Observed:

- Reduced bias: The most significant change that one can observe is a reduction in bias in the data or model. This can be seen in the form of more equitable outcomes for different demographic groups. For example, if a model is used to predict loan approvals, after applying bias mitigation techniques, the model might approve loans for historically disadvantaged groups at a rate that is closer to that of the privileged group.

- Increased diversity and representation: By collecting diverse and representative data, one can expect to see an increase in the diversity and representation of the population being studied. For example, if bias mitigation techniques are applied to a recruitment process, the candidate pool may become more diverse, resulting in a more representative workforce.

- Improved accuracy: Bias mitigation techniques can improve the accuracy of a model by reducing the impact of biased data on the training process. For example, if a facial recognition system is trained on biased data, it might be less accurate for specific demographic groups. By applying bias mitigation techniques, the system can be trained on more representative data, resulting in improved accuracy for all groups.

- Increased transparency: Bias mitigation techniques can increase the transparency of the decision-making process. By using metrics that measure fairness and evaluating the model's performance on these metrics, one can gain insights into how the model is making decisions and identify areas for improvement.

References:

https://arxiv.org/abs/2010.11929v2
https://lightning.ai/docs/pytorch/stable/
https://huggingface.co/docs/transformers/model_doc/vit
https://huggingface.co/google/vit-base-patch16-224
https://huggingface.co/docs/transformers/v4.15.0/training
https://paperswithcode.com/method/vision-transformer
https://pytorch.org/docs/stable/index.html
https://huggingface.co/docs/datasets/quickstart
https://numpy.org/doc/stable/reference/index.html
https://torchmetrics.readthedocs.io/en/stable/
https://www.tensorflow.org/tensorboard
https://stackoverflow.com/
Class PPTs.