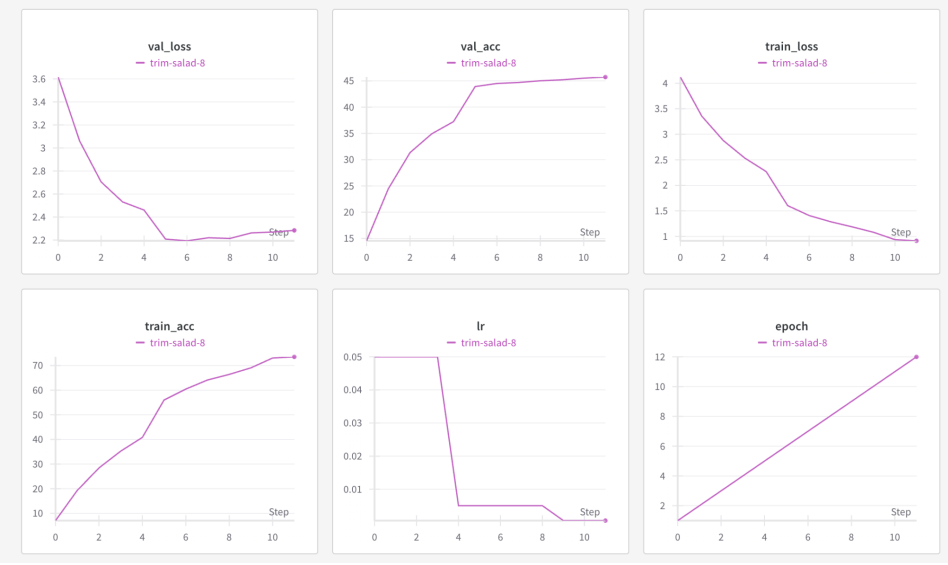
Kailen Richards
Deep Learning

AI statement

I used AI to correct my grammar mistakes in the report (Grammarly).

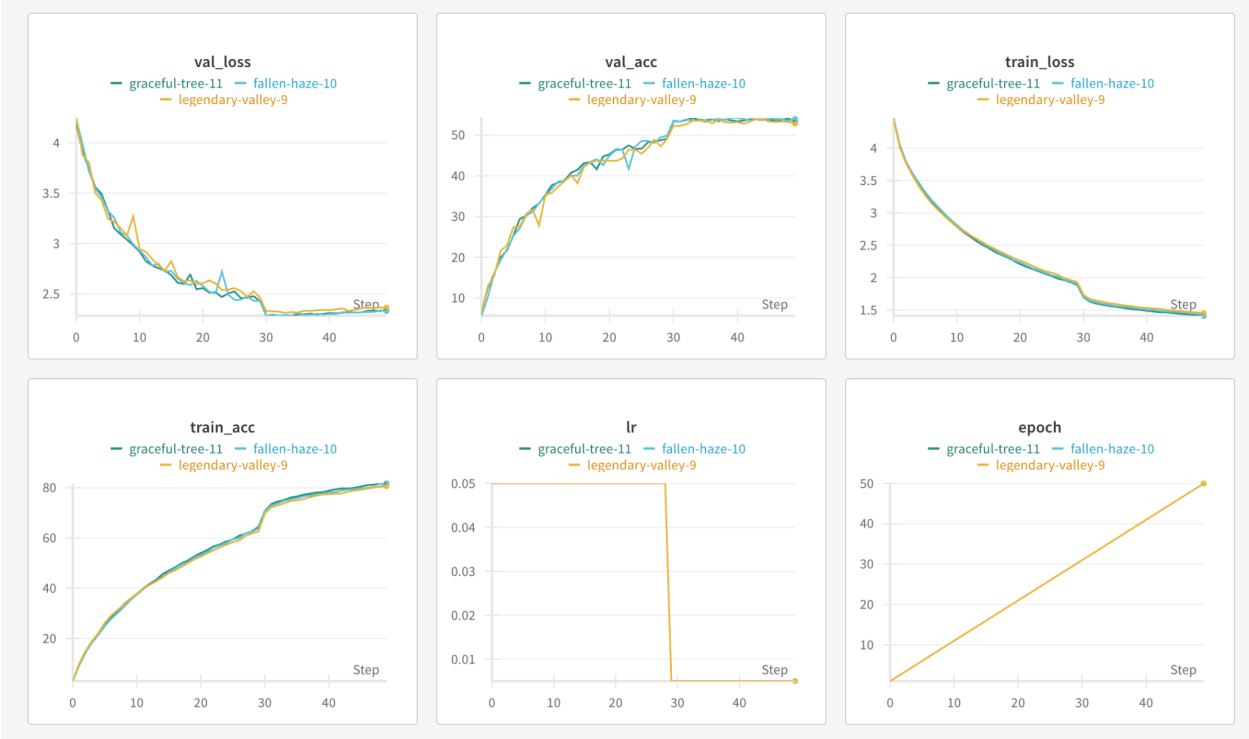
I used Deep seek to help solve an issue regarding the scoring in Kaggle, because of resizing the images for EfficientNetV2-S to 224x224, I had to modify the eval_ood.py to adjust for that.

I used a simple and straightforward CNN for the first model to classify the CIFAR 100 dataset. For the architecture, I chose to start with three convolutional layers, each with a ReLU activation function based on the nature of the data. The first layer has three input channels for the RGB images and an output of 32, following the next layer includes an input of 32 and an output of 64, followed by the third layer, which has an input of 64 nodes and an output of 128. These layers are followed by a max pooling layer with a stride of 2, which is followed by two fully connected layers. After the first fully connected layer, a dropout rate of .25 is included to prevent overfitting. The output of the final fully connected is 100 to match the classification of the images for the dataset. As for Hyperparameter tuning, I started with the initial values from the template to get a baseline but continued to adjust to having a batch size of 128 and trained for 12 epochs. In addition, it included a weight decay of 1e-4 for L2 regularization. As for transformation, only pixel values were normalized. This model is relatively shallow for the dataset, but that is by design of the midterm. The ineffectiveness of the model can be seen in the results, where the gap between validation loss and training loss is relatively high(1 vs. 2.3), and the gap between validation accuracy and training accuracy is very large (70% vs. 45%): these are significant signs of overfitting the training dataset. One improvement for the training dataset could have been data augmentation forms or a higher dropout rate between the fully connected layers.

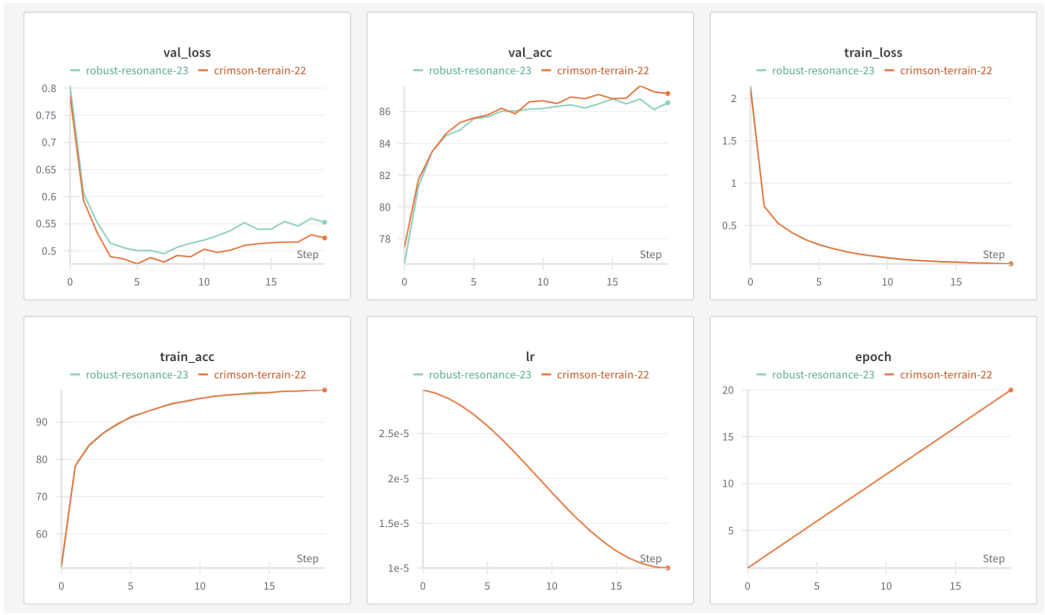


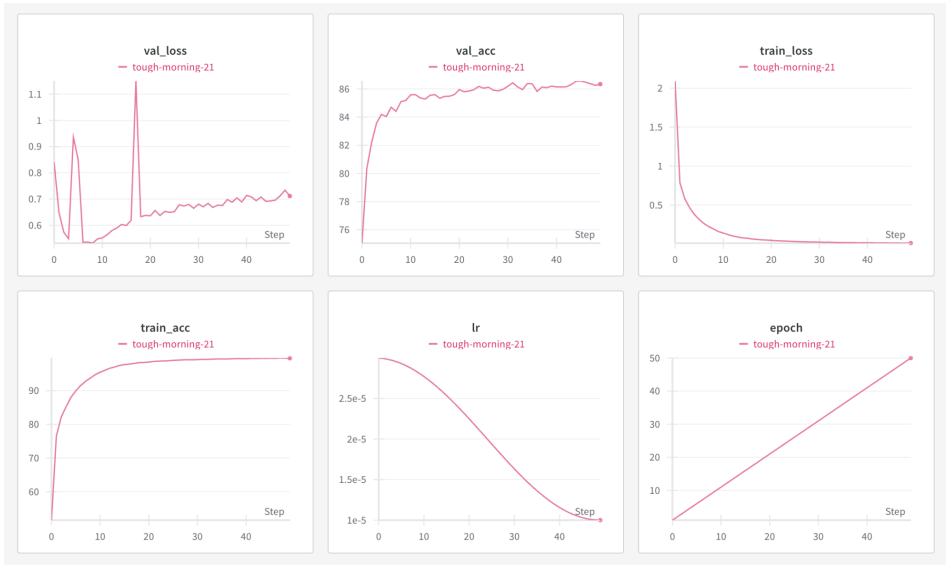
For the second model, I decided to go with the mobilenetv2 architecture; it is designed to be efficient but also accurate, especially for image classification. I went with Mobilenet for the second task after reading about its success compared to transformer models, so I hoped that even with minor modifications and an untrained model, it would bring success. The first change for the first layer is changing the input layer to have three input channels and 32 outputs to match the dataset's images and changing the final layer to have an output of 100 classes to fit the Cifar100 dataset. In terms of hyperparameter tuning after many iterations, I went with a learning rate of.05, 50 epochs, 128 batch size, and a weight decay of 4e-5. For the learning rate, I went with .05 because the default optimizer for Mobilenetv2 is RSMprop, which allows the learning rate to remain stable for each epoch. As for augmentation, I added random flip, crop, and color jitter to combat overfitting. Over the 50 epochs, at around step 30 is when the model training loss and validation loss intersect, and even though training accuracy at the same step is higher than validation accuracy, my strategies for combating overfitting compared to the initial simple model worked to an extent.

.



For the final model, I decided to use EfficientNetv2, which has been pre-trained on the Imagenet dataset. In addition, I used EfficientNetv2 because of the speed and accuracy it showed on the imagenet dataset compared to previous iterations of the model, and it proved to have a similar effect on the CIFAR-100 dataset. When looking at the variations of the model, I decided to use Efficientnetv2-s, which is designed for smaller datasets. Initially, the first modification I made was to freeze the first five feature blocks; this prevented catastrophic forgetting of features like edges and textures common among image datasets. Keeping the other layers unfrozen allows the data to be better fine-tuned for the dataset. In addition, the final layer is simplified to 100 classes for the data, and a dropout of 0.2 is added for regularization. For training, I used AdamW with a weight decay of 0.1 and different learning rates; for the feature extraction layers, it is lower and higher for the classification layers. I also included a learning rate scheduler of CosineAnnealingLR for a smooth decay in the learning rate. For hyperparameters, I decided to use a batch size of 128 and a base learning rate of 3e-4, and for the feature extraction layers, I used a learning rate of 3e-5. For data augmentation, I resize the images to 224X224 to better fit the first layer of Efficientnetv2, random flip, and random rotation. The results of this model were pleasing. At the same time, there are slight signs of overfitting with longer training runs; in shorter iterations of training the model, the validation loss and training loss are relatively close, and the gap in the accuracy of training and validation is expected. One improvement I would make is to add early stopping; without it, I wasted so much time and computing waiting for the model to finish. In addition, something that may be hindering the model's accuracy in the new data is freezing the initial layers; something that could have been done is gradually unfreezing the initial layers to allow for better fine-tuning of the initial features.


val_loss
robust-resonance-23
crimson-terrain-22
Step
val_acc
robust-resonance-23
crimson-terrain-22
Step
train_loss
robust-resonance-23
crimson-terrain-22
Step
train_acc
robust-resonance-23
crimson-terrain-22
Step
lr
robust-resonance-23
crimson-terrain-22
Step
epoch
robust-resonance-23
crimson-terrain-22
Step


val_loss
tough-morning-21
Step
val_acc
tough-morning-21
Step
train_loss
tough-morning-21
Step
train_acc
tough-morning-21
Step
lr
tough-morning-21
Step
epoch
tough-morning-21
Step