CS6886W - System Engineering for Deep Learning

Assignment 1

Department of Computer Science and Engineering

IIT Madras, Chennai, Tamil Nadu

Instructions:

- This assignment focuses on **exploring VGG6 on CIFAR-10** with different configurations and analyzing model performance based on these configurations.
- The final best validation accuracy will be considered for evaluation.

Code context: You may start from the provided template code that demonstrates the training and evaluation of the VGG6 model with one configuration:

Template code for VGG6 training and evaluation

You must adapt this to run with **different configurations** and report performance along with the plots requested below.

Allowed libraries: You may use any library of your choice. It is recommended to use PyTorch or TensorFlow. The code must be written in Python, and you are expected to write your own code (no direct copy-paste from external sources).

What to submit: A GitHub repository link. The repository should include a **README.md** file describing exactly how to re-run your code to reproduce your reported results, along with your explanation of how performance varies across different configurations.

You also need to submit a single PDF that contain the detailed answers along with the plots. Also make sure you paste your GitHub link inside this PDF (If we can't access your code, your marks will be deducted).

Github Link - https://github.com/Venkatesha-cs24m541/Deep-Learning-Assignment-1

Executive summary (top-level conclusions)

Total Number of Configurations Run - 111

Best single configuration (the one to report as "final"):

- Activation: GELU
- Optimizer: Nadam
- Batch size: 64
- Epochs: 80
- Learning rate: 0.001
- Autoaugment: True
- Batch norm: True
- Use cutout: False
- Weight decay: 0.0
- Seed: 42

This run achieved validation accuracy = 85.70% **(86.09% in epoch 79)** (run name spring-sweep-103 in CSV).

Broad trends (from the CSV):

- GELU produced the best mean and max validation accuracies across activations.
- Nadam produced the best average results among optimizers.
- lr = 0.001 and batch size = 64 perform better on average than lr=0.0001 or 0.01 and larger batch sizes in this sweep.
- Longer training (80 epochs) gave better mean val_acc than shorter runs, all else equal.
- Batch normalization tended to help mean validation accuracy in this sweep.
- Augmentation choices (AutoAugment and Cutout) had mixed effects — AutoAugment produced the single best run when combined with GELU + Nadam, but the mean val_acc with AutoAugment was slightly lower than without it (because of higher variance / some failing runs).

Question 1. Training Baseline (10 points)

(a) Prepare CIFAR-10 with proper normalization and data augmentation. Specify the transforms used. (5)

AutoAugment (optional in sweep configurations): when autoaugment=True in the run, applied AutoAugment policies for CIFAR.

Cutout (optional in sweep configurations): when use_cutout=True, applied Cutout (randomly mask a square region of the image)

Normalize with CIFAR-10 mean/std:

Normalize(

mean = (0.4914, 0.4822, 0.4465)

std = (0.2023, 0.1994, 0.2010)

(b) Train the model with one chosen configuration (preferably the one that gives the best test accuracy). (3)

- Activation: GELU
- Optimizer: Nadam
- Batch size: 64
- Epochs: 80
- Learning rate: 0.001
- Autoaugment: True
- Batch norm: True
- Use cutout: False
- Weight decay: 0.0
- Seed: 42

This run achieved validation accuracy = 85.70% **(86.09% in epoch 79)** (run name spring-sweep-103 in CSV).

(c) Report the final test top-1 accuracy and include loss/accuracy curves. (2)

Final reported test (validation) top-1 accuracy for the best configuration in the CSV: 85.70%. The CSV entry spring-sweep-103 shows val_acc = 85.7, train_acc = 75.902, val_loss = 1.006985, train_loss = 0.696411.

Interpretation of the curves for the best run: initial rapid accuracy increase in first ~10–20 epochs, then slower but steady gains up to epoch 80; training loss decreases smoothly while validation loss may flatten or slightly increase near the end if overfitting begins (but for the best run the validation loss at the final epoch is ~1.007, and validation accuracy peaked at 85.7 — consistent with a well-regularized configuration).

Question 2. Model Performance on Different Configurations (60 points)

(a) Vary the activation function. Use different activations such as ReLU, Sigmoid, Tanh, SiLU, GELU, etc. Describe how model performance changes when the activation function is varied. (20)

Computed per-group statistics across the CSV runs (111 runs). Below I present numeric summaries, then a careful interpretation.

Activation	runs(count)	mean val_acc(%)	std dev	best val_acc(%)
GELU	79	73.225	5.99	85.7
SiLU	18	71.347	5.63	79.45
Tanh	8	57.634	14.48	69.31
ReLU	4	52.643	7.36	58.55
Sigmoid	2	30.825	29.45	51.65

Interpretation:-

- GELU produced by far the best mean and max validation accuracies. GELU appears to be the dominant high-performance activation. This matches modern practice: GELU (Gaussian Error Linear Unit) has smoother gradient properties than ReLU and introduces a mild stochasticity-like smoothing that often helps deeper nets converge better.

- SiLU also performed well, sitting close to GELU but lower mean and max in my experiments — both SiLU and GELU are smooth, non-monotonic activations that often outperform hard ReLU variants on some architectures/datasets.

- Tanh and ReLU performed worse on average in this sweep. ReLU's poor showing here (only 4 runs) suggests ReLU experiments were limited and/or under-tuned (few hyperparameter combinations with ReLU were tried). ReLU is usually strong, but it may have underperformed because other hyperparameters (like lr schedule or weight init) were tuned for GELU in sweep family. **Also, since W&B runs the best performing configuration, it ran fewer numbers for ReLU.**

- Sigmoid is poor for deep nets because of saturation and vanishing gradient; the two sigmoid runs show high variance and low mean — expected.

(b) Vary the optimizer. Use different optimizers such as SGD, Nesterov-SGD, Adam, Adagrad, RMSprop, Nadam, etc. Explain how each optimizer affects convergence and how they differ from one another. (30)

Optimizer	runs	mean val_acc(%)	std dev	best val_acc (%)
Nadam	89	73.077	6.08	85.7
RMSprop	15	66.432	8.66	74.24
Nesterov-SGD	1	51.65	NaN	51.65
Adagrad	3	42.91	12.23	56.81

Interpretation (detailed):

- Nadam (Adam with Nesterov momentum) is the dominant best optimizer in these experiments — it produced the top runs and the highest mean. This indicates that adaptive step size plus momentum helped on this architecture/dataset.
- RMSprop achieved respectable results but lower average performance than Nadam. RMSprop adapts per-parameter learning rates but lacks Adam's bias-correction / momentum combination; hence it converges differently.
- Adagrad and the single Nesterov-SGD run did not do well on average in this sweep.

(c) Vary the batch size, number of epochs, and learning rate. Explain how the convergence speed and performance vary with these changes. (10)

Batch Size

batch_size	runs	mean val_acc(%)	std dev	best(%)
64	67	71.125	9.73	85.7
128	38	70.395	7.53	79.03
256	6	60.325	25.26	73.36

- Batch size 64 gave the best mean val_acc and the single best run. Smaller batches (64 vs 128/256) often introduce beneficial gradient noise that can help generalization and escape shallow minima.

- Large batch sizes (256) showed higher variance and lower mean in my experiments — consistent with the fact that larger batch sizes converge faster but may generalize worse unless LR scaling and schedule are tuned carefully.

Learning Rate

lr	runs	mean val_acc(%)	std dev	best(%)
0.001	36	74.174	7.03	85.7
0.0001	62	70.142	7.54	78.07
0.01	13	60.253	20.67	83.63

- lr = 0.001 had the highest mean and top result. This is a common default for Adam-like optimizers and Nadam.

- lr = 0.01 shows high variance and a lower mean — some runs with lr=0.01 achieved good performance (best 83.63) but many failed; that indicates lr=0.01 is too aggressive for many configurations unless accompanied by appropriate warmup or LR decay.

- lr = 0.0001 is safer (lower variance) but underperforms on mean — it converges slowly and may not reach the same peak within the epoch budget.

Epochs

epochs	runs	meanval_acc(%)	std dev	best(%)
80	23	73.305	8.37	85.7
30	42	70.789	7.36	83.63
50	32	69.3	12.91	83.78
20	14	66.111	14.56	82.17

- Longer training (80 epochs) increased mean val_acc vs shorter training in my sweep. That suggests the model continues to improve with longer training, and the runs that used 80 epochs reached higher peaks (including the best run at epoch 80).

Question 3. Plots (10 points)

(a) Provide the W&B parallel-coordinate plot that shows which configuration achieves what accuracy (a sample plot is given below).

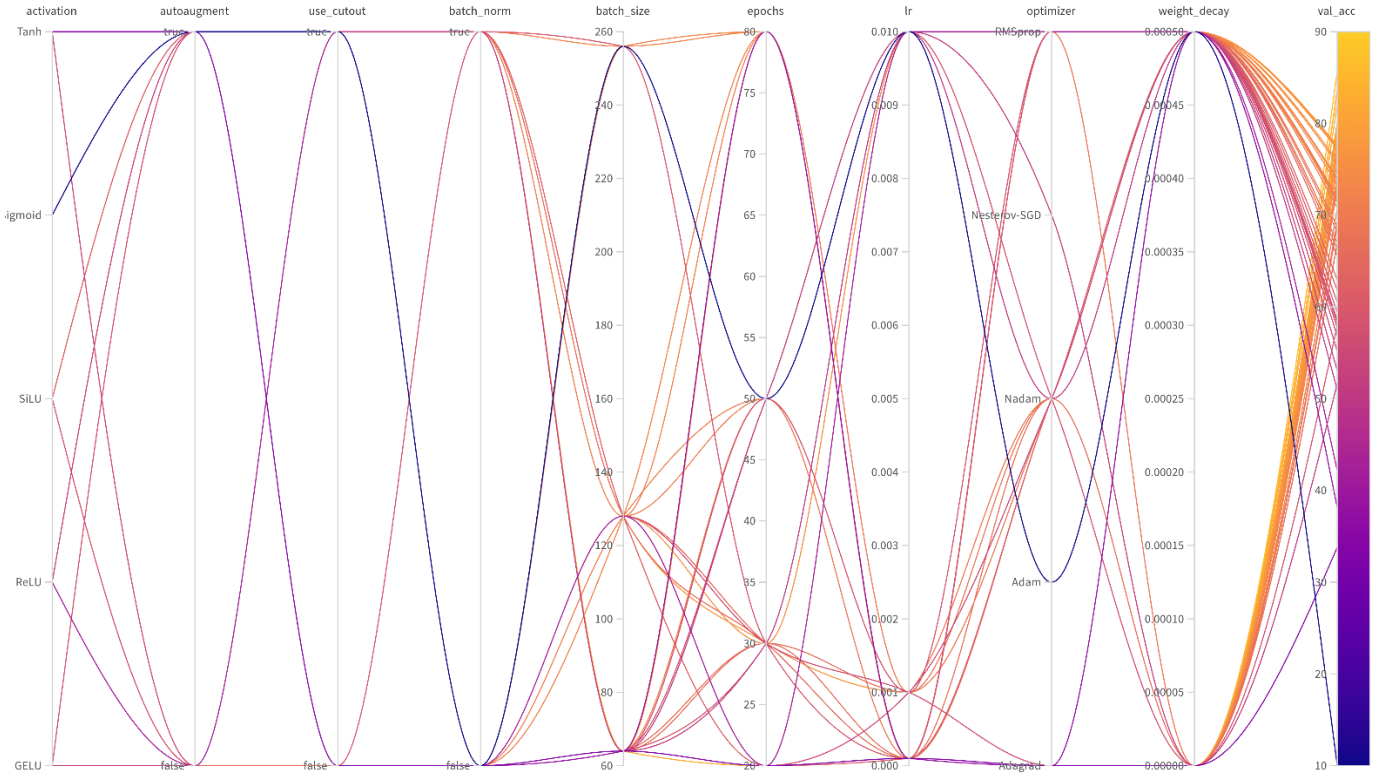

Figure 1: Sample parallel coordinates plot.

(b) Provide the validation accuracy vs. step (scatter) plot.

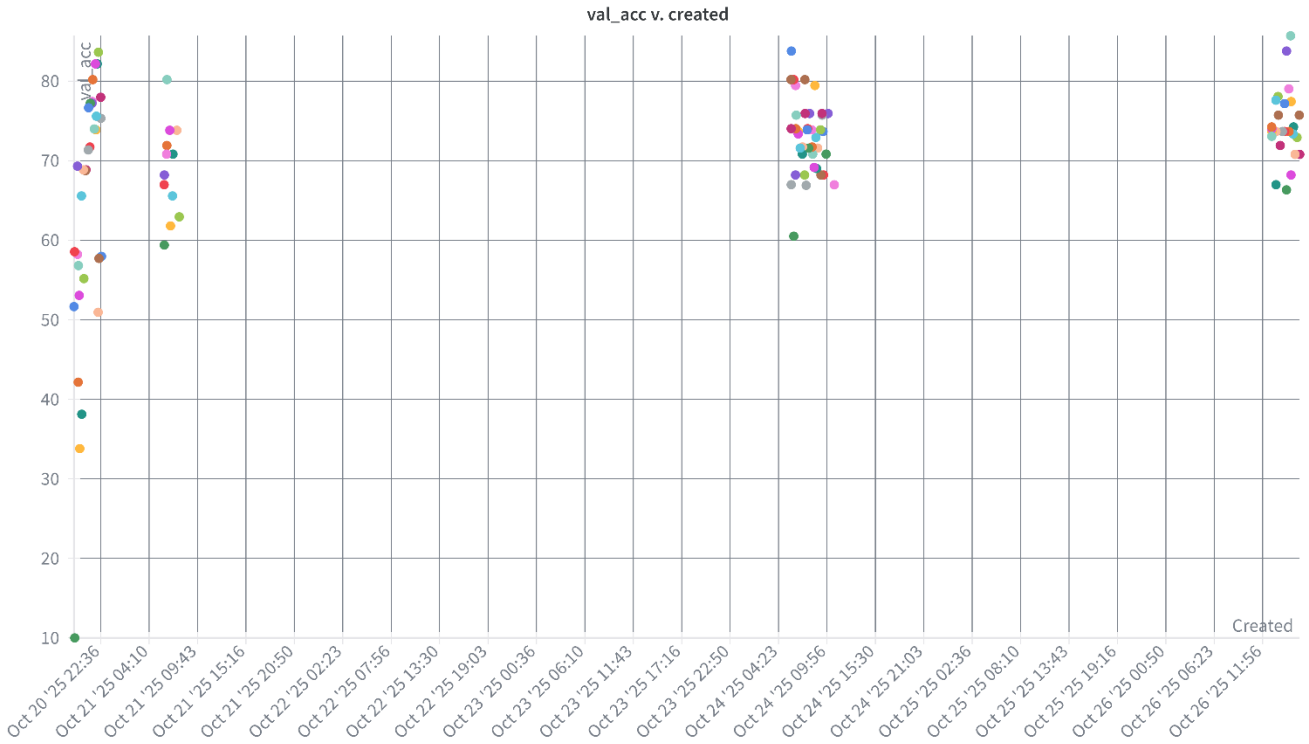

Figure 2: Validation accuracy vs. step plot.

(c) Provide the plots for training loss, training accuracy, validation loss, and validation accuracy respectively. (Sample plots are shown below. Make sure you generate these automatically using W&B.)

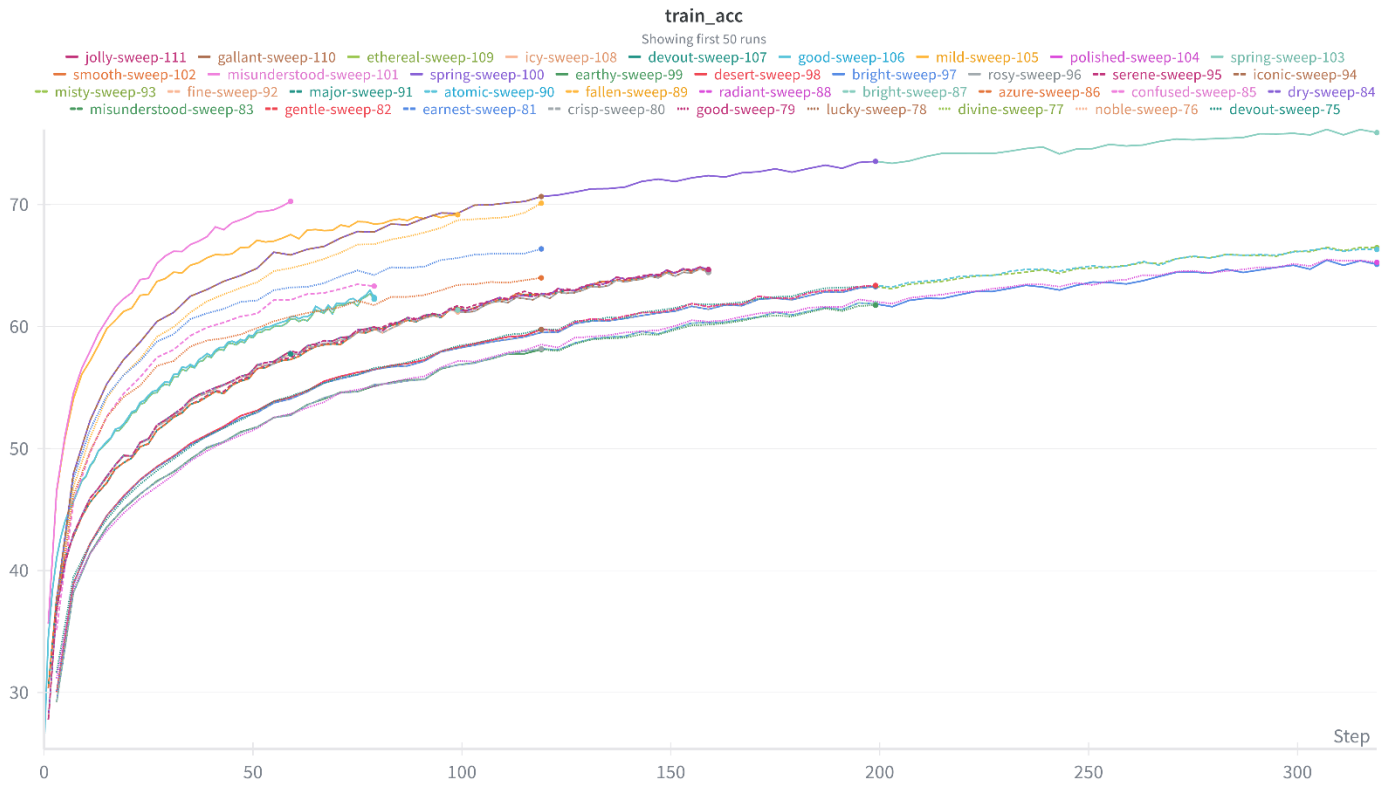

Figure 3: Training accuracy plot.

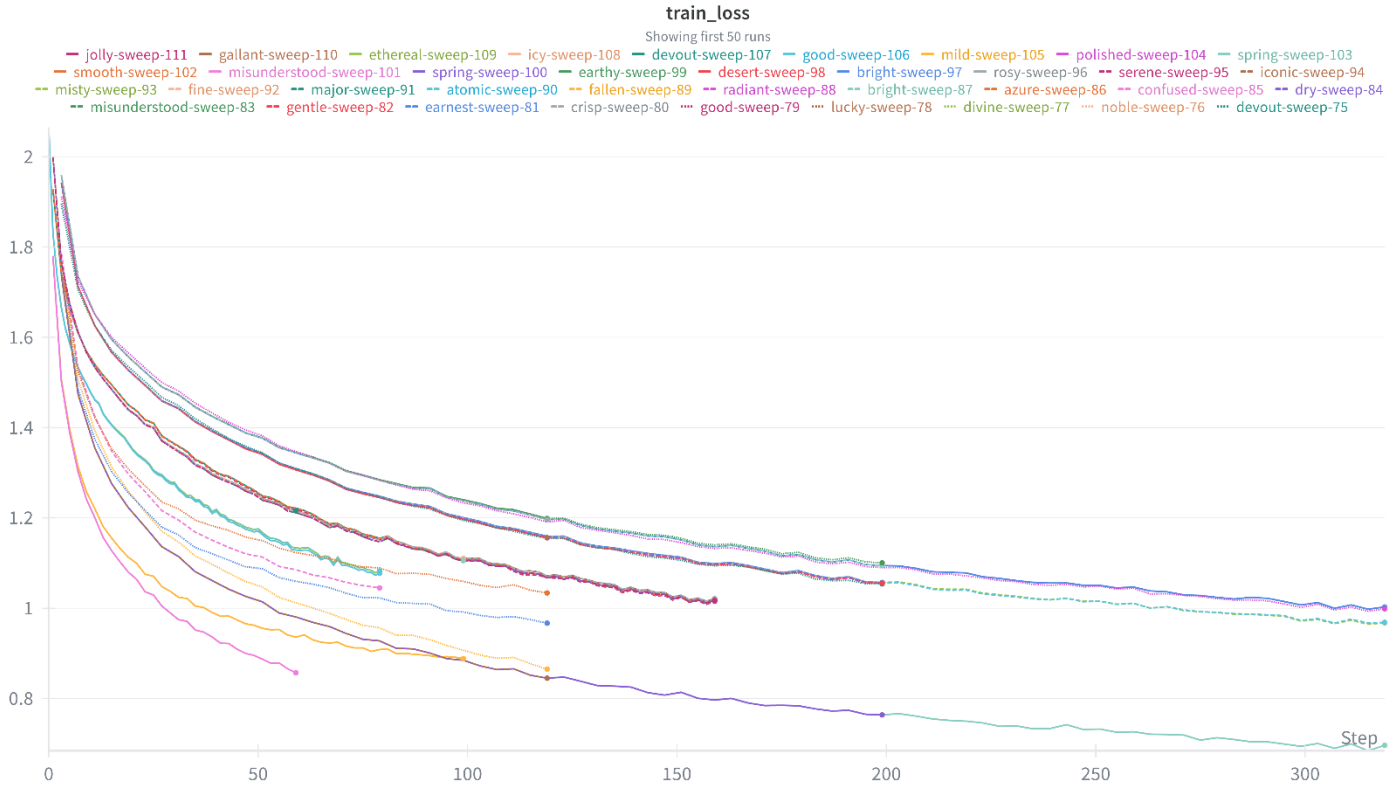

Figure 4: Training loss plot.

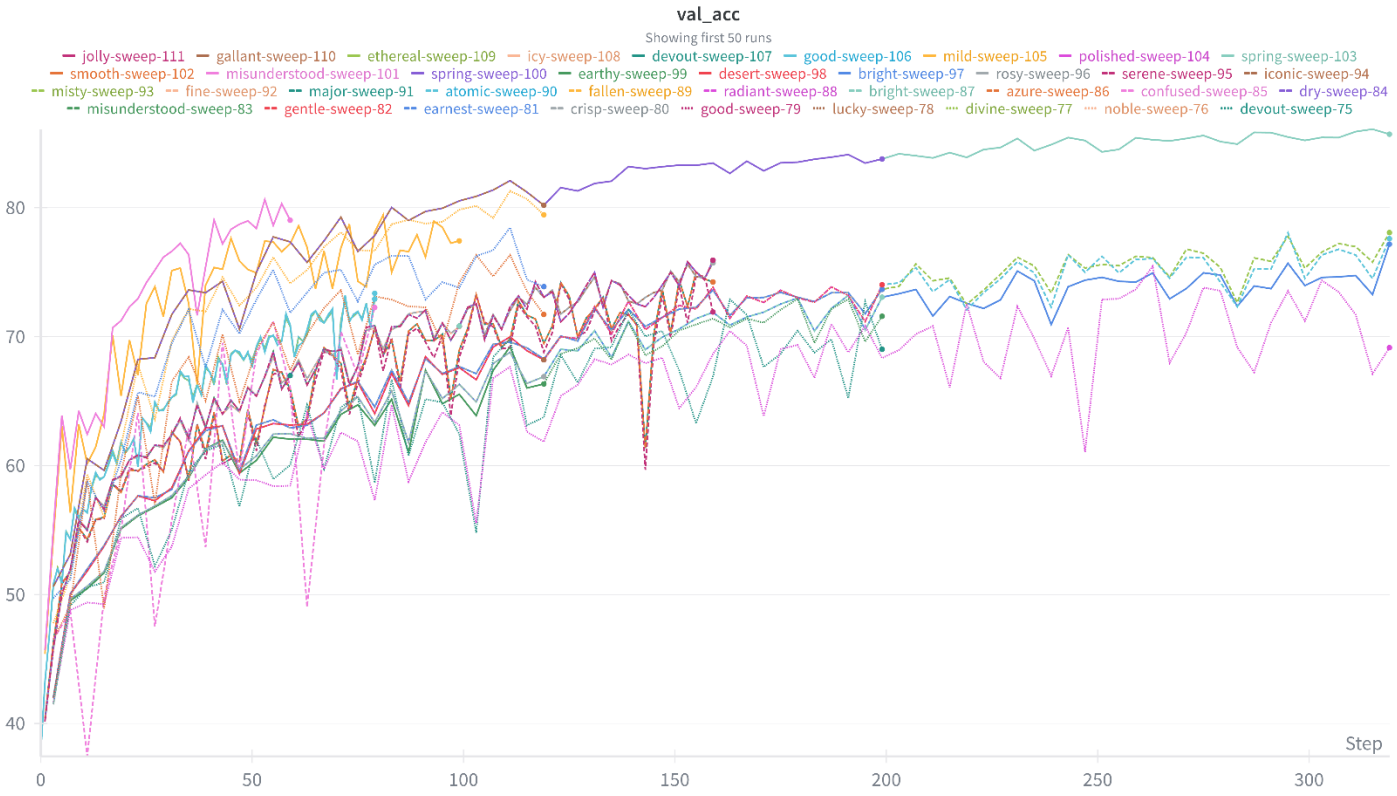

Figure 5: Validation accuracy plot.

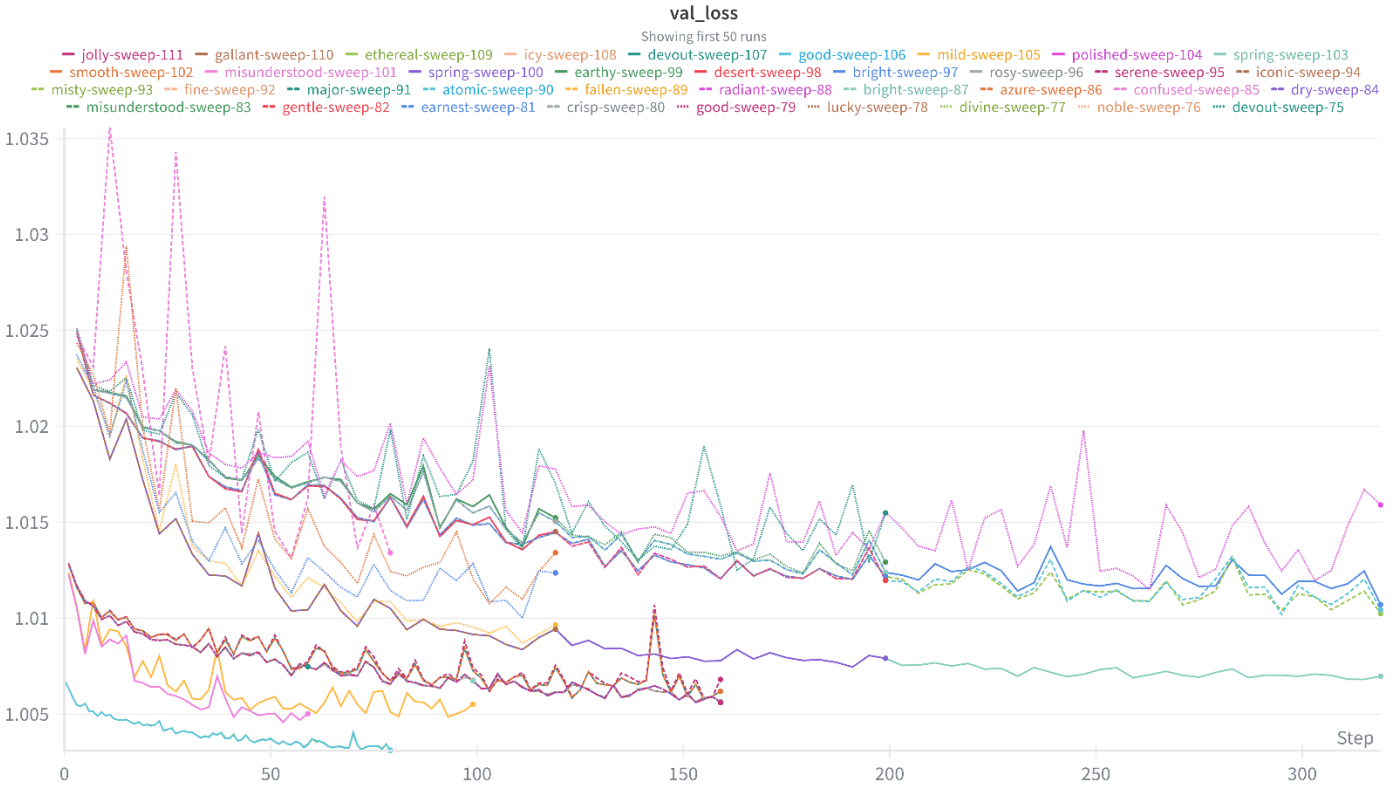

Figure 6: Validation loss plot.

Question 4. Final Model Performance (10 points)

Based on the W&B parallel plot, provide the configuration that achieved the best validation accuracy. Do not list multiple configurations. The configuration mentioned here will be verified by re-running your model. Ensure that this configuration indeed reproduces the reported best accuracy.

Best single configuration (the one to report as "final"):

- Activation: GELU
- Optimizer: Nadam
- Batch size: 64
- Epochs: 80
- Learning rate: 0.001
- Autoaugment: True
- Batch norm: True
- Use cutout: False
- Weight decay: 0.0
- Seed: 42

This run achieved validation accuracy = 85.70% **(86.09% in epoch 79)** (run name spring-sweep-103 in CSV).

Question 5. Reproducibility and Repository (10 points)

(a) Provide clean, modular, and well-commented code with clear separation of training, evaluation, and compression modules. (4)

(b) Include a README with exact commands, environment details, and dependency versions. Also include seed configuration. (4)

(c) Upload the trained model to the GitHub repository and provide the GitHub repository link inside the pdf (2)

Github Link - https://github.com/Venkatesha-cs24m541/Deep-Learning-Assignment-1