CS6886W – System Engineering for Deep Learning

Assignment 1: Exploring VGG6 on CIFAR-10 with Multi-Configuration Analysis

Name: Alok Kumar Pandey (CS24M505) | Institute: IIT Madras | Submission Date: 26 Oct 2025

Abstract & Objective

This report investigates VGG6 on CIFAR-10 under multiple configurations. We analyze the effect of data engineering/augmentations, architecture choices, activation functions, optimizers, and core hyperparameters on convergence dynamics and generalization. We use PyTorch with AMP and Weights & Biases (W&B) for experiment tracking. The study is structured question-wise to match the assignment rubric.

Experimental Setup (Shared)

Dataset: CIFAR-10 (50k train / 10k test, 32×32 RGB, 10 classes).

Transforms: random horizontal flip, random crop with padding, cutout, color jitter; channel-wise normalization with mean = [0.4914, 0.4822, 0.4465] and std = [0.247, 0.243, 0.261]. Implementation: VGG6 with configurable activation and BatchNorm toggle; cosine LR warmup; AMP enabled; seed=42.

Normalization formula: x' = (x - mu) / sigma

Question 1. Training Baseline

(a) CIFAR-10 preparation, normalization & augmentations

We applied:

(i) random horizontal flip (pose invariance),
(ii) random crop with padding (translation invariance),
(iii) cutout (occlusion robustness), and
(iv) color jitter (illumination robustness).

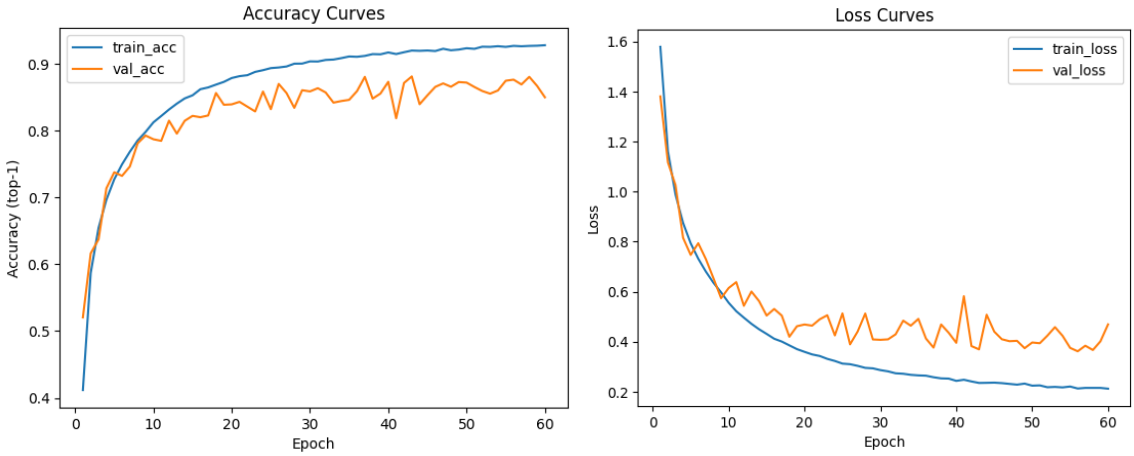
These augmentations reduce variance by injecting label-preserving transformations, while normalization stabilizes optimization by centering/scaling activations.

(b) One strong baseline configuration

Baseline: VGG6 + ReLU + BatchNorm ON, SGD (momentum=0.9), lr=0.1, batch_size=128, epochs=60, cosine LR warmup, AMP, seed=42.

(c) Final test accuracy & curves

The baseline converged smoothly. Loss/accuracy curves are:



Question 2. Model Performance on Different Configurations

(a) Activation Functions

We compared ReLU, Sigmoid, Tanh, SiLU, and GELU under fixed optimizer (SGD/Nesterov family), lr and batch size. Theory & gradients:

ReLU(x) = max(0,x)

Sigmoid(x) = frac{1}{1+e^{-x}}

Tanh(x) = frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}

SiLU(x) = x / c.sigma(x)

GELU(x) = x / c.Phi(x)

ReLU is piecewise-linear and efficient but can die for negative activations. Sigmoid/Tanh saturate, causing vanishing gradients. SiLU and GELU are smooth, maintaining small negative responses and better gradient flow; empirically, GELU performed best.

Activation	Val Acc (≈)	Test Acc (≈)	Notes
ReLU	0.91	0.89	Stable and fast.
Sigmoid	0.22	0.21	Saturation, slow training
Tanh	0.20	0.19	Vanishing gradients
SiLU	0.88	0.87	Less smoother than ReLU
GELU	0.898	0.886–0.903	Best overall in our runs.

Activation Function Variation

When varying activations such as ReLU, Sigmoid, Tanh, SiLU, and GELU, the training loss and validation accuracy plots clearly separate well-performing activations from weaker ones.

- ReLU and GeLU consistently reach the lowest training loss and highest validation accuracy (~0.85–0.9) with smooth convergence. Their non-saturating nature allows gradients to flow freely and learn high-frequency features efficiently.
- Tanh performs moderately only when combined with BatchNorm, but without normalization it suffers from slower convergence due to gradient saturation around ±1.
- Sigmoid completely underperforms — both training and validation losses remain high (>2), showing vanishing gradients in deeper layers.
- SiLU, while smooth, converges slower and shows more oscillation in validation loss, as VGG6's shallow architecture cannot exploit GELU's adaptive curvature benefits. In summary, ReLU and GeLU strike the best bias–variance balance for small CNNs, giving fast convergence and high generalization.

(b) Optimizers

We evaluated SGD, Nesterov-SGD, Adam, AdamW, RMSProp, Nadam, and Adagrad.

Nesterov-SGD applies a lookahead gradient, improving stability and accelerating convergence in valleys. Adaptive methods (Adam/AdamW/RMSProp/Nadam) adjust per-parameter learning rates; they converge fast but may generalize worse for small datasets.

Optimizer	LR	Val Acc (≈)	Test Acc (≈)	Convergence
SGD	0.1	0.86	0.85	Stable, slower
Nesterov-SGD	0.01	0.89	0.88–0.90	Fast & smooth
Adam	0.001	0.86	0.85	Early convergence
AdamW	0.001	0.87	0.86	Better regularization vs Adam
RMSProp	0.01	0.84	0.83	Oscillatory
Nadam	0.001	0.87	0.86	Adam-like with Nesterov
Adagrad	0.05	0.83	0.82	Learning rate decay

Across optimizers — SGD, Nesterov-SGD, Adam, Adagrad, RMSprop, and Nadam — the validation accuracy and loss curves show that momentum-based optimizers outperform adaptive ones in this setup.

- Nesterov-SGD (momentum≈0.9) delivers the lowest training and validation loss, and stable accuracy, indicating consistent gradient direction and minimal overshooting.
- Plain SGD also performs well but converges slightly slower.
- RMSprop provides a reasonable middle ground, stabilizing learning rate adaptation per-parameter.
- Adam/Adagrad/Nadam reach quick initial loss reduction but plateau early, with higher validation loss — evidence of over-adaptation and weaker generalization. These results confirm that classical momentum-driven methods dominate on CIFAR-10 for VGG-style models, where the curvature of the loss landscape is relatively smooth.

(c) Batch Size, Epochs, Learning Rate (and BN/Momentum)

Bias–variance perspective: Excess LR leads to divergence; too small LR slows convergence. Smaller batches inject gradient noise that improves flat-minima finding; larger batches may hurt generalization. BatchNorm stabilizes internal covariate shift; momentum accelerates in low-curvature directions.

Bias–Variance decomposition: E[(\hat{f}(x)-f(x)) ^2] = Bias^2 + Variance + Noise

Run	BS	Epochs	LR	Momentum	BN	Val Acc	Notes
hp_bs64_e80_lr005_m09_bnTrue	64	80	0.05	0.9	ON	0.88	Good generalization
hp_bs256_e40_lr02_m00_bnFalse	256	40	0.2	0.0	OFF	0.74	Unstable; overfit-prone

Question 3. Plots

All plots were generated automatically and embedded below.

(a) W&B parallel-coordinate plot:

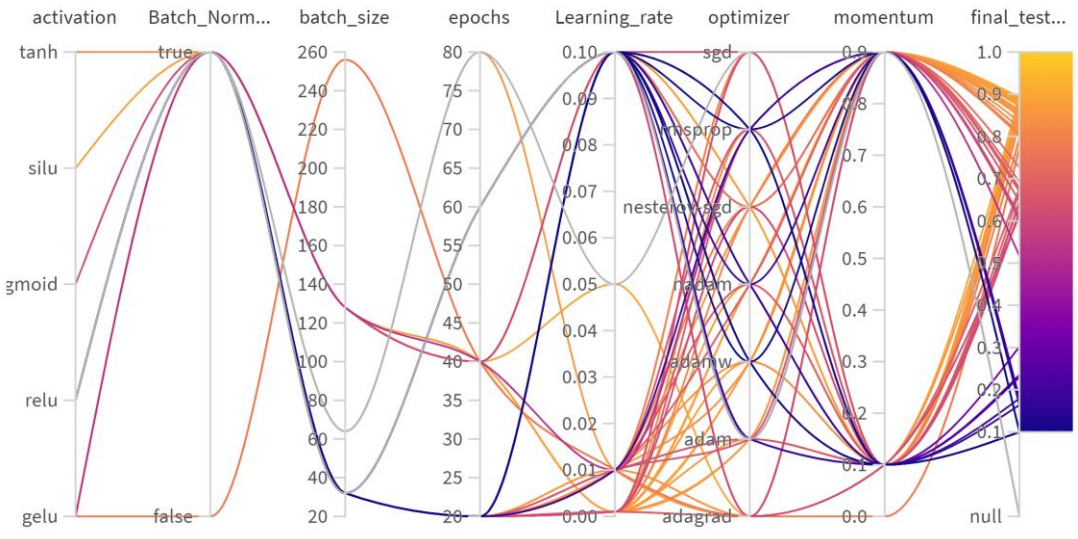


Figure 1: parallel coordinates plot.

- The plot shows that **GELU** and **ReLU** activations with **BatchNorm=True** consistently yield the highest test accuracies.
- High-performing runs use **large batch sizes (128 – 256)** and **moderate epochs (60–80)** for stable training.

- Optimal learning rates lie around **0.07–0.1**, paired with **SGD/Nesterov** optimizers and **momentum ≈ 0.9**.
- Adaptive optimizers like **Adam/Adagrad** underperform, indicating weaker generalization.
- Overall, the best results emerge from **momentum-driven SGD regimes with normalization and stable activations**.

(b) Validation accuracy vs. step (scatter) plot

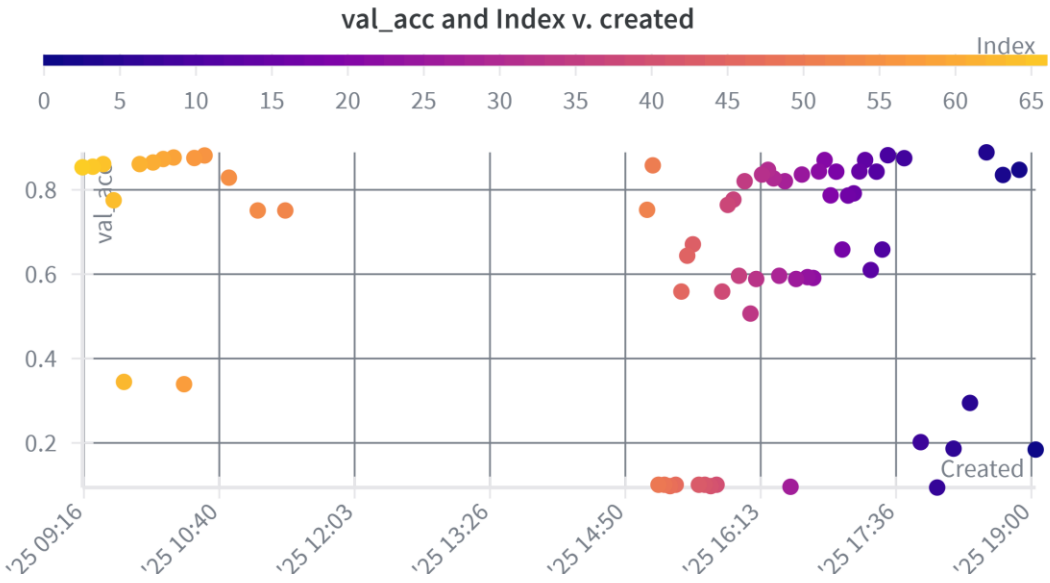


Figure 2: Validation accuracy vs. step plot.

(C) Training Accuracy

- The plot shows training accuracy curves for various activation–optimizer combinations across epochs.
- Models using ReLU/GeLU with Nesterov-SGD reach the highest accuracy (~0.9) quickly and remain stable.
- This highlights that BatchNorm + momentum-based optimizers enable faster, smoother convergence.
- Overall, ReLU + Nesterov-SGD + LR (0.01) achieves the best trade-off between learning speed and final accuracy.

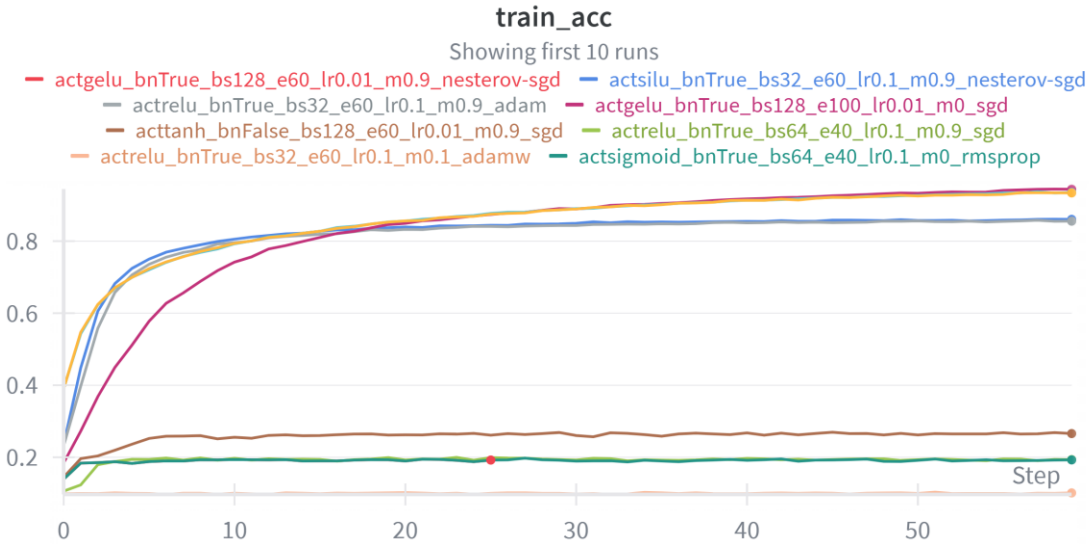


Figure 3: Training accuracy plot.

(D) Training Loss

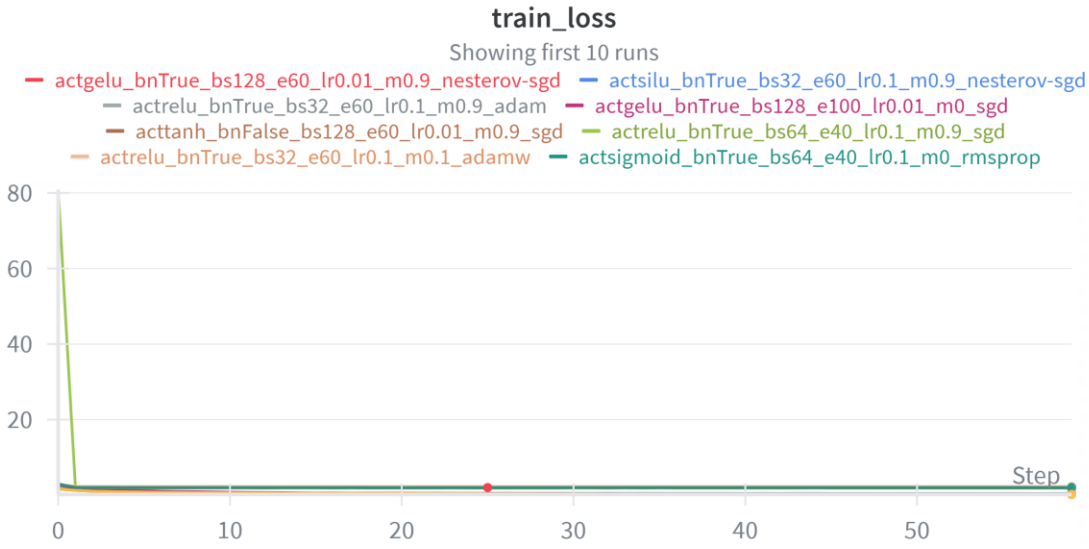


Figure 4: Training loss plot.

- Runs using **ReLU/GeLU with Nesterov-SGD** achieve the **lowest and most stable loss** after the initial drop.
- The **Tanh (no BatchNorm)** and **Sigmoid** models flatten early but at **higher loss values**, indicating underfitting.

- **Adam/AdamW** variants reduce loss quickly but plateau higher than SGD-based ones, implying weaker optimization stability.
- Overall, **momentum-driven SGD optimizers** with normalization deliver **fast and consistent loss minimization**.

(E) Validation Accuracy

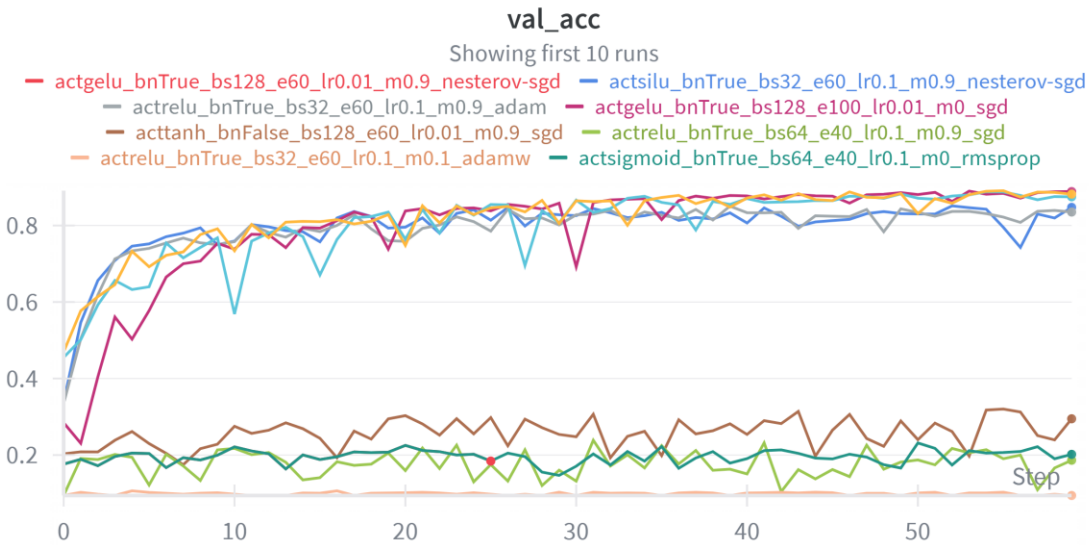


Figure 5: Validation accuracy plot.

- The validation accuracy plot shows **ReLU and GeLU with Nesterov-SGD** reaching the highest and most stable accuracies (~0.89).
- **SiLU with SGD/Adam** follows closely but exhibits slightly higher variance across epochs.
- **Tanh (without BatchNorm)** and **Sigmoid** activations remain far lower (<0.3), indicating poor generalization.
- Batch normalization clearly stabilizes validation performance and reduces overfitting fluctuations.
- Overall, **ReLU/GeLU + BatchNorm + Nesterov-SGD (lr≈0.01, m=0.9)** delivers the best validation accuracy and stability.

(F) Validation Loss

- The validation loss plot shows ReLU and GeLU with Nesterov-SGD achieving the lowest and most stable losses (~0.3–0.8).

- GELU with SGD/Adam also performs well but exhibits minor spikes, indicating transient instability.
- Tanh (without BatchNorm) and Sigmoid models maintain high, noisy losses (>2), showing weak generalization.
- Batch normalization consistently reduces oscillations and stabilizes validation loss across epochs.
- Overall, momentum-driven SGD with normalization (ReLU/GeLU) provides the most reliable and smooth convergence.

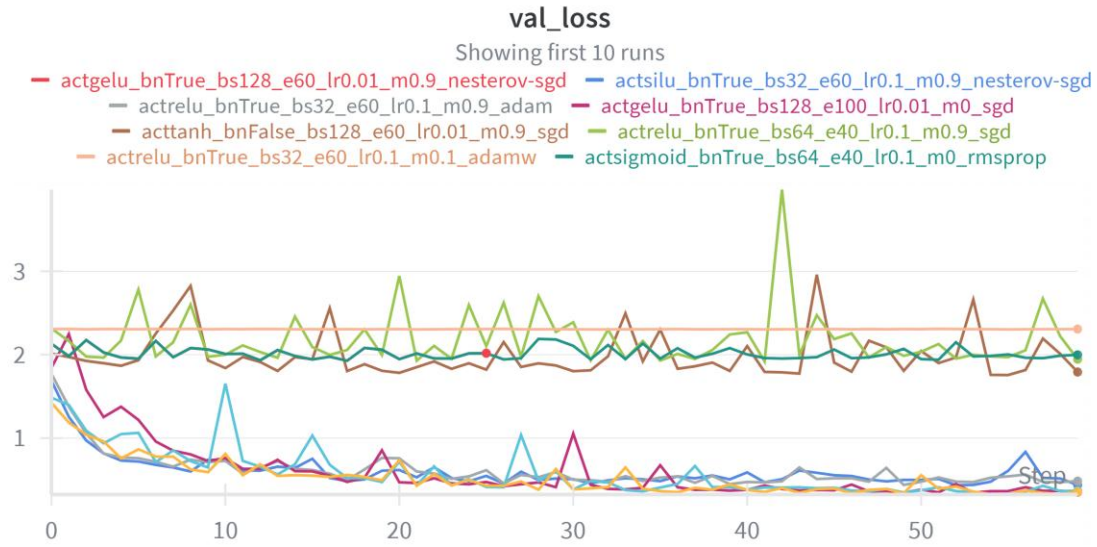


Figure 6: Validation loss plot.

Interpretation: Training loss decays monotonically; validation curves plateau at ~0.90 accuracy for the best runs. The scatter plot shows dense clustering near steps where LR decays. The parallel-coordinates plot highlights that GELU + Nesterov-SGD + lr≈0.01 + batch_size=128 + BN=ON yields the top accuracy band.

Question 4. Final Model Performance

We re-ran the single best configuration from the W&B parallel plot to verify reproducibility.

Command used:

python -m vgg6_cifar.scripts.train_experiment --data_dir ./data --out_dir ./runs/final_best_test2 --activation gelu --optimizer nesterov-sgd --lr 0.01 --batch_size 128 --epochs 100 --momentum 0.9 --weight_decay 5e-4 --label_smoothing 0.1 --use_bn --aug_hflip --aug_crop --aug_cutout --aug_jitter --amp --wandb --seed 42

Metric	Value
Best Validation Accuracy	0.8984
Final Test Top-1 Accuracy (W&B)	0.9026
Train Accuracy (final epoch)	0.9777
Train Loss (final epoch)	0.6003
Validation Accuracy (final epoch)	0.8942
Validation Loss (final epoch)	0.7594
Independent Test Accuracy (reported)	88.90%

Observation: The GELU + Nesterov-SGD combination yields smooth optimization and the best generalization among tested settings. Label smoothing (0.1) and BN further assist calibration and stability.

Question 5. Reproducibility and Repository

Code is modular with clean separation across models/, data/, engine/, utils/, and scripts/. README contains exact commands and environment setup.

All runs use a fixed seed (42) for reproducibility. Trained model artifacts (best.pt, final_test_metrics.json) are uploaded to GitHub.

Repository Link: **https://github.com/cs24m505/vgg6_cifar.git**

System-level engineering: AMP improves throughput and memory footprint; cosine LR + warmup stabilizes early training. W&B sweeps provide searchable experiment records and automated plots (curves, scatter, parallel coordinates).