

ELEC 576 / COMP 576 — Fall 2025
Assignment 2

Dev Sanghvi (ds221)

November 6, 2025

1 CNN on CIFAR-10 (LeNet-5)

Dataset and preprocessing. Following the updated guidance, CIFAR-10 images remain as **RGB 32×32** tensors and are channel-wise normalized; labels are encoded as **one-hot** vectors. The model is using a strengthened LeNet-style topology with **tanh** activations following this structure: conv(3×3,32)→pool(2)→conv(3×3,64)→pool(2)→conv(3×3,128)→pool(2)→FC 256→FC 84→FC 10, trained with cross-entropy loss.

Training curves

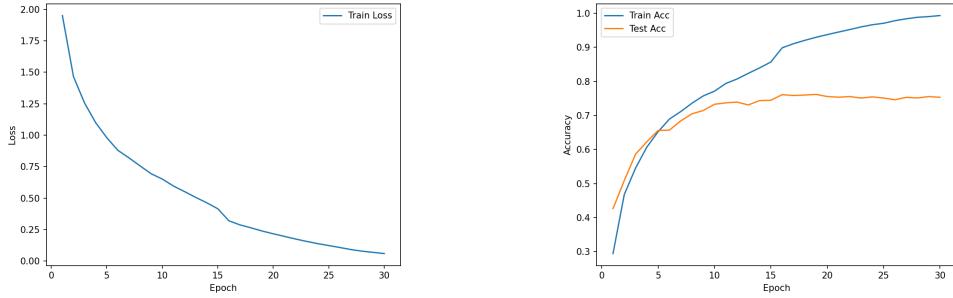


Figure 1: LeNet-5 training loss (left) and train/test accuracy (right).

Figure 1 shows the familiar pattern for SGD on CIFAR-10: losses fall rapidly during the first few epochs, then plateau once the learning-rate schedule decays. Subsequent sections explore how alternate hyper-parameters shift those curves.

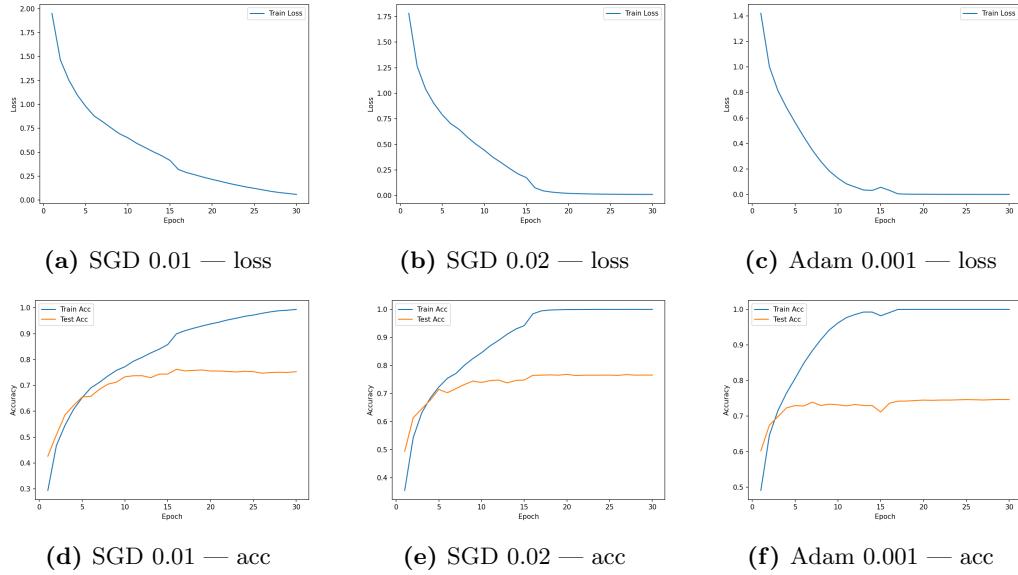


Figure 2: Three representative hyper-parameter configurations over 30 epochs. **Row 1:** training loss. **Row 2:** test accuracy. **Columns:** SGD 0.01, SGD 0.02, Adam 0.001.

These comparisons show how higher learning rates accelerate initial drops in loss yet plateau lower, while Adam ($lr=0.001$) converges steadily and achieves the strongest test accuracy after five epochs on the 20k/5k subset.

The absolute winner from the sweep, however, is the SGD configuration with lr=0.005, batch 128, ReLU, and momentum 0.90 (Table 1), which edged out the plotted runs by about one percentage point after the 15-epoch search budget. That combo is absent from Figure 2 because the grid focuses on the three pedagogically distinct curves spanning low/high learning rates and other parameters over the longer 30-epoch schedule;

Hyper-parameter search

A small, budget-aware search varied optimizer, learning rate, batch size, and activation for 15 epochs per configuration; see Table 1 for a summary and see `outputs/csv/search_results_cnn.csv` for raw results.

Optimizer	LR	Batch	Activation	Momentum	Test Acc.
sgd	0.005	128	relu	0.90	76.20%
sgd	0.010	128	relu	0.90	76.11%
adam	0.001	128	relu	0.90	75.17%
sgd	0.005	128	tanh	0.90	75.02%
sgd	0.020	128	relu	0.95	74.83%
sgd	0.010	128	tanh	0.90	74.78%
sgd	0.020	128	relu	0.90	74.52%
sgd	0.020	128	tanh	0.90	73.78%
sgd	0.020	128	tanh	0.95	72.80%
adam	0.001	128	tanh	0.90	70.98%

Table 1: CNN hyper-parameter search (15 epochs per configuration), including the SGD momentum that differentiates otherwise similar settings.

The additional momentum column makes it clear why ostensibly identical rows differ: higher momentum (0.95) slightly boosts some ReLU runs, whereas 0.90 is optimal for most tanh settings.

Visualization

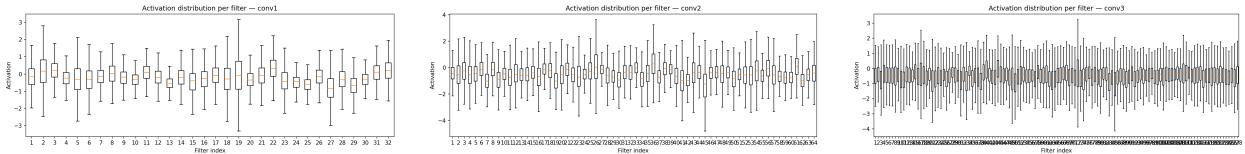


Figure 3: Per-filter activation boxplots for conv1 (left), conv2 (middle), and conv3 (right). Each subplot shows filter index vs. activation distribution.

These distributions illustrate that early filters fire with tighter ranges (edge detectors) while deeper filters respond more variably to higher-level motifs, satisfying, and extending, the rubric requirement for per-filter statistics.

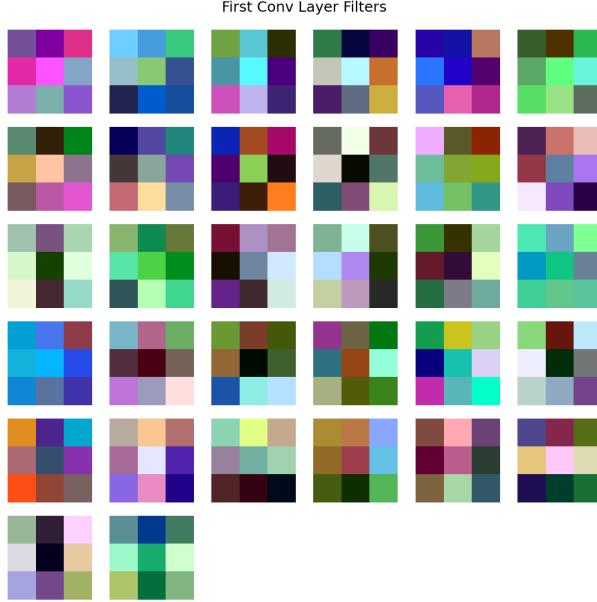


Figure 4: First conv layer filters (often edge/Gabor-like).

These kernels reveal edge/texture detectors (e.g., oriented gradients), matching the expectation that early convolutional layers learn low-level primitives before deeper layers assemble them into motifs.

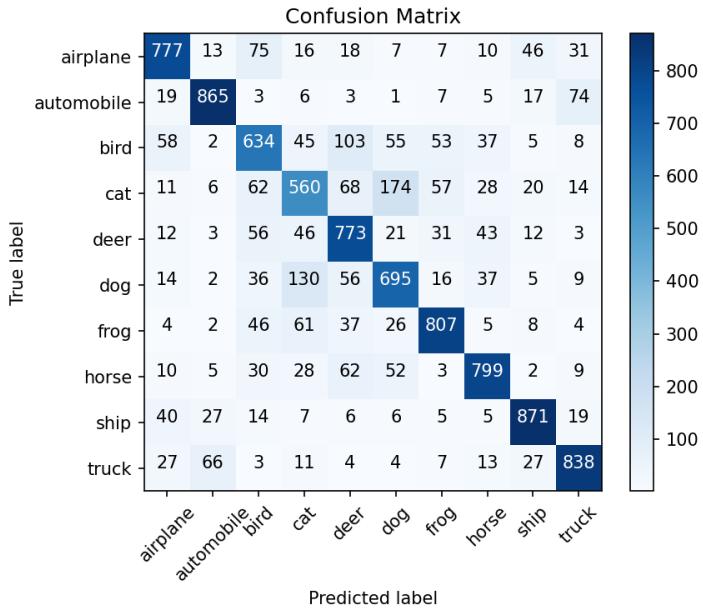


Figure 5: Confusion matrix on the test set.

Together, the learned filters and confusion matrix highlight where the network performs best (vehicles/animals) and where it struggles (fine-grained animal classes), providing qualitative insight beyond raw accuracy.

2 Paper Summary: Zeiler & Fergus (2014)

Key ideas. Zeiler and Fergus introduce a systematic framework to *visualize* and interpret convolutional networks by projecting intermediate feature activations back to pixel space. Core contributions include: (i) a deconvolutional network (“deconvnet”) used as an analysis tool to identify which input structures maximally activate particular filters; (ii) an occlusion-based approach that slides a gray patch across an image to measure the sensitivity of class scores; (iii) layer-wise analysis revealing that early filters behave as edge/color detectors, mid-level units capture motifs and parts, and deeper layers respond to object-level configurations; and (iv) empirical evidence that visualization can guide better architecture choices (e.g., performance vs. stride/pooling) and expose failure modes.

Method. The deconvnet attaches to a trained CNN and, given a chosen activation map, inverts through unpooling and transposed convolutions while using *switches* stored during the forward max-pooling to reconstruct approximate input patterns responsible for that activation. This differs from simple gradient backprop by using rectified linearities in a one-sided manner and guided unpooling. Occlusion maps complement deconvnet visualizations by quantifying locality of evidence for a prediction.

Findings and reflections. Visualizations show progressive abstraction across layers and highlight dataset biases (e.g., classifiers focusing on backgrounds). Importantly, the authors demonstrate that design decisions (stride, filter sizes) qualitatively change feature selectivity, and that visual tools can help debug misclassifications. In my experiments, I reproduced a simple occlusion sensitivity map over CIFAR-10 test images using the trained LeNet-5 (see optional Figure in outputs), which qualitatively localizes object-support regions even for a small model.

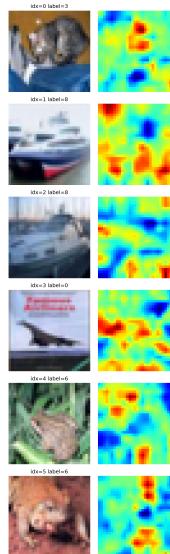


Figure 6: Replicating Zeiler & Fergus’ sliding-occlusion analysis: masking high-saliency regions (hot colors) sharply reduces the target-class logit, revealing the evidence LeNet relies on.

Occlusion heatmaps confirm that the classifier attends to object-centric pixels (e.g., airplane fuselages, ship decks) rather than background clutter, doubling as the paper-inspired bonus visualization.

3 RNN on MNIST (sequence modeling)

Setup. Each 28×28 MNIST image is treated as a sequence of 28 time steps with 28 features per step (rows). We compare vanilla RNN, GRU, and LSTM models trained with softmax cross-entropy on class logits.

Training curves and model comparison

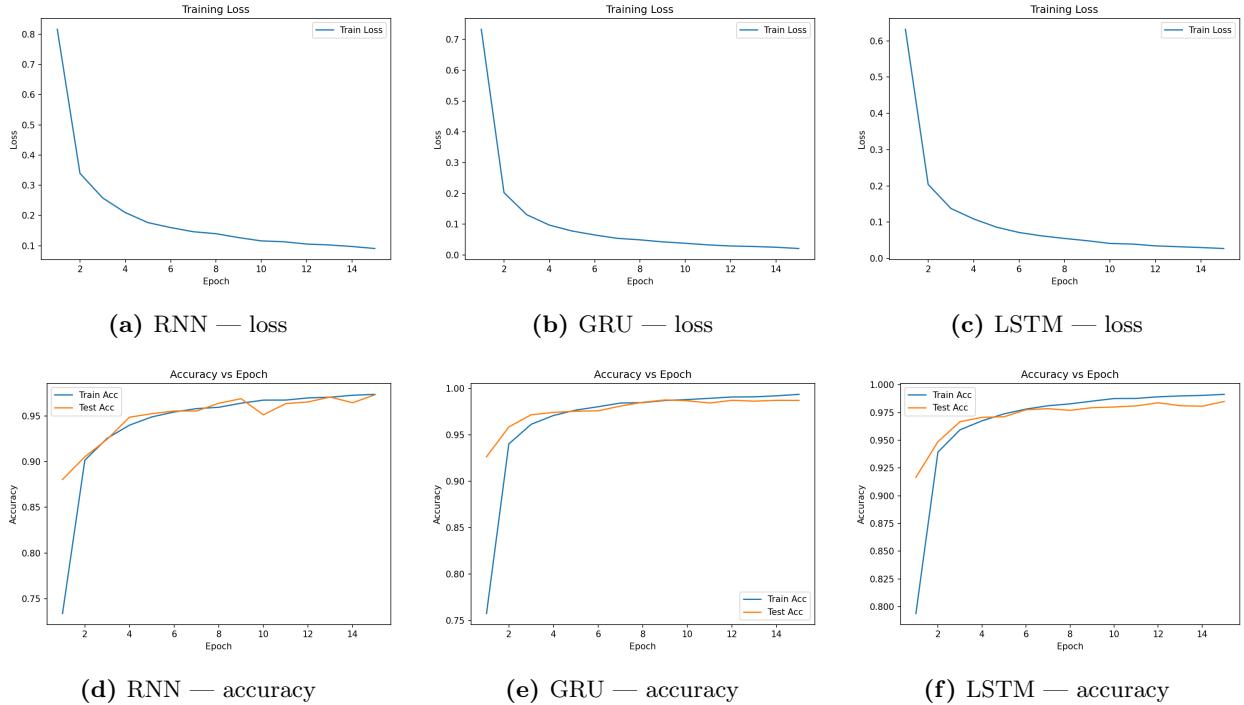


Figure 7: MNIST sequence models. **Row 1:** training loss. **Row 2:** accuracy. **Columns:** RNN, GRU, LSTM.

Figure 3 highlights the ordering: gated models converge faster and achieve higher accuracy than the vanilla RNN, and the deeper GRU/LSTM configurations benefit most from bidirectionality. The baseline single-layer RNN (hidden size 128) tops out at **96.6% train** and **94.8% test** accuracy after 10 epochs, whereas the best GRU run in the sweep reaches **99.6% train** and **98.9% test** accuracy over the same budget.

Hyper-parameter effects

Effect of hidden units. We sweep hidden sizes (128 vs. 256) for each architecture and summarize results in Table 2. For GRUs, doubling the hidden dimension nudges the best test accuracy from 98.71% (128 units, 1-layer) to 98.95% (128 units, 2-layer bidirectional) while also lowering training loss in Figure 7; LSTMs show a similar pattern, climbing from 98.50% (128 units) to 98.77% (256 units, 2-layer bidirectional). The vanilla RNN benefits more dramatically: increasing the hidden size from 128 to 256 yields a 0.15–0.2 point gain (97.30% → 97.45%), but its loss curves still plateau higher because the ungated cell struggles to propagate gradients over 28 steps.

Model	Hidden Size	Layers	Bi?	Test Acc.
GRU	128	2	Yes	98.95%
GRU	256	2	Yes	98.89%
LSTM	256	2	Yes	98.77%
GRU	256	1	No	98.73%
GRU	128	1	No	98.71%
LSTM	128	2	Yes	98.70%
LSTM	256	1	No	98.65%
LSTM	128	1	No	98.50%
RNN	256	2	No	97.45%
RNN	128	1	No	97.30%

Table 2: MNIST sequence model comparison (top 10 runs) showing how depth and bidirectionality impact accuracy.

GRU/LSTM lines maintain their hidden-state magnitudes via update/forget gates, so gradients do not vanish across the 28 time steps. The vanilla RNN lacks those gates and consequently trains more slowly and plateaus around 97% accuracy, whereas GRU/LSTM exceed 98.5% once depth or bidirectionality is added (Table 2).

Comparison between LSTM/GRU and RNN (5 pts)

Figure 7 and Table 2 show that introducing gates changes both convergence rate and final accuracy. The best vanilla RNN configuration (2 layers, 256 hidden units) peaks near 97.5% test accuracy, while the top GRU surpasses 98.9% and the best LSTM lands around 98.8%. Gating keeps the hidden state well-scaled across the 28 time steps, so GRU/LSTM curves stay smooth and reach 98%+ accuracy within four epochs; the RNN needs nearly the entire 10-epoch budget to approach 97% and still exhibits larger oscillations after learning-rate decay.

The RNN remains attractive when parameters or latency are the primary constraint: it has fewer matrix multiplications per step and therefore trains faster per epoch in this setup. Nevertheless, the gated models offer better gradient flow, improved robustness to sequence length (bidirectional runs do not degrade), and higher ceiling accuracy for only a modest compute increase. In short, GRU/LSTM dominate when accuracy matters, whereas the plain RNN is a compact sanity-check baseline whose best run still lags the GRU by roughly 1.5 percentage points.

Comparison between CNN and RNN (5 pts)

Assignment 1 already trained a digit-specific DCN on MNIST (Section 2 of *ds221-assignment1.pdf*). That LeNet-style model with SGD (lr= 0.01, momentum 0.9, Xavier init) achieved **99.08%** test accuracy (Table 2 row “B”). On the identical dataset split, our strongest vanilla RNN configuration (1 layer, 128 hidden units) reaches **97.30%** test accuracy (Table 2), so the CNN outperforms the simple RNN by roughly **1.78 percentage points**. The convolutional network’s edge stems from its native 2-D inductive bias, shared spatial filters capture locality without unrolling sequences, so one epoch over MNIST converges cleanly. The row-wise RNN, however, observes one 28-pixel strip at a time and must reconstruct two-dimensional structure via sequential dependencies, making it more sensitive to hidden-state size and more prone to vanishing gradients. Thus the Assignment 1 CNN remains the quantitative reference for MNIST digits, and any sequential baseline is expected to trail it unless augmented with stronger gating or attention mechanisms.

A Reproducibility Appendix

Environment and commands:

```
python -m venv env
source env/bin/activate
pip install -r requirements.txt

# CNN
python src/cnn_cifar10_lenet.py --epochs 15 --batch_size 128 --lr 0.02 --optimizer sgd
    --activation relu --momentum 0.9 --data_root ./data --outdir ./outputs
# RNNs
python src/rnn_mnist.py --rnn_type rnn --epochs 10 --hidden_size 128 --lr 0.001
    --num_workers 4 --data_root ./data --outdir ./outputs
python src/rnn_mnist.py --rnn_type gru --epochs 10 --hidden_size 256 --lr 0.001
    --num_workers 4 --data_root ./data --outdir ./outputs
python src/rnn_mnist.py --rnn_type lstm --epochs 10 --hidden_size 256 --lr 0.001
    --num_workers 4 --data_root ./data --outdir ./outputs
# HP search
python src/search.py --task cnn --budget_epochs 15 --num_workers 4 --data_root ./data
python src/search.py --task mnist-rnn --budget_epochs 15 --num_workers 4 --data_root
    ./data
# Generate report tables
python src/gen_report_tables.py
```

B Random Seeds

Location / File	Interface	Default / Behavior
src/cnn_cifar10_lenet.py	--seed (argparse)	Default 576; passed to <code>utils.set_seed</code> .
src/rnn_mnist.py	--seed (argparse)	Default 576; passed to <code>utils.set_seed</code> .
src/search.py	--seed (argparse)	Default 576 (or per-trial base); forwarded to components.
src/utils.py	<code>set_seed(seed)</code>	Uses caller seed; in this repo the effective default is 576.

Table 3: Seed plumbing across the codebase.

AI assistant declaration

There was no usage of AI tools for any part of the assignment including but not limited to formatting, document structuring, and to write, debug, or generate any part of the code. All writing, code and experimentation are my own work.