Convolutional Neural Networks — CIFAR-10 Report (PyTorch)

1. Model Architecture and Training Setup

1.1 Objective

I implemented and trained a custom Convolutional Neural Network (CNN) from scratch (no pretrained networks) to classify CIFAR-10 images into 10 categories. The goal was to demonstrate a correct CNN design and a complete training pipeline by reporting: (1) final test accuracy, (2) training/validation loss curves, and (3) a clear description of the architecture and training setup.

1.2 Dataset (CIFAR-10)

CIFAR-10 is a standard benchmark dataset for small-image classification:

- 50,000 training images
- 10,000 test images
- Image size: 32 × 32
- Channels: 3 (RGB)
- Classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

A CNN is appropriate for CIFAR-10 because it learns hierarchical visual representations: early layers capture edges and color contrasts, while deeper layers combine these into textures, parts, and class-discriminative patterns.

1.3 Data pipeline

Train/validation split
I split the official CIFAR-10 training set into:

- Training: 80% (40,000 images)
- Validation: 20% (10,000 images)

The validation set was used to monitor generalization and to select the best checkpoint using minimum validation loss.

Normalization
All images were normalized using CIFAR-10's per-channel mean and standard deviation:

- Mean: (0.4914, 0.4822, 0.4465)
- Std: (0.2470, 0.2435, 0.2616)

This stabilizes training by keeping input scales consistent across channels and improving gradient behavior.

Data augmentation (training only)
To improve generalization, I applied augmentation only to training:

- RandomCrop(32, padding=4)
- RandomHorizontalFlip()

Validation and test images were not augmented to keep evaluation consistent and reproducible.

1.4 Model architecture (Custom CNN)

Requirement checklist

- Custom CNN (no pretrained model): Yes
- ≥ 3 convolution layers: Yes (3 conv layers)
- ≥ 1 pooling layer: Yes (MaxPool)
- Nonlinear activations: Yes (ReLU)
- Fully connected head: Yes (Linear layers to 10 outputs)

Architecture summary (Colab run)
The network used was:

1. Conv1: Conv2d(3 → 32, kernel=3, padding=1) → ReLU → MaxPool(2×2)
2. Conv2: Conv2d(32 → 64, kernel=3, padding=1) → ReLU → MaxPool(2×2)
3. Conv3: Conv2d(64 → 128, kernel=3, padding=1) → ReLU → MaxPool(2×2)
4. Flatten
5. Dropout(p=0.25)
6. FC1: Linear(2048 → 256) → ReLU
7. Dropout(p=0.25)
8. FC2: Linear(256 → 10) (logits)

Tensor shape flow (why flatten size = 2048)
Input: 3 × 32 × 32

- After Conv1 + Pool → 32 × 16 × 16

- After Conv2 + Pool → 64 × 8 × 8
- After Conv3 + Pool → 128 × 4 × 4
  Flatten: 128 × 4 × 4 = 2048

That is why FC1 has in_features = 2048.

1.5 Training setup

Loss function: CrossEntropyLoss
This is the standard loss for multi-class classification with 10 classes, using raw logits.

Optimizer: Adam
Adam converges quickly and works well for this CNN without extensive manual tuning.

Hyperparameters (Colab run)

- Learning rate: 0.001
- Batch size: 64
- Epochs: 25
- Device: GPU (Google Colab runtime)
- Weight decay (L2): 1e-4
- Dropout: p = 0.25

Dropout and weight decay reduce overfitting by discouraging overly large weights and preventing reliance on a small set of activations.

---

2. Training Results

2.1 Training process and checkpoint selection

For each epoch, I computed:

- average training loss over the training subset
- average validation loss over the validation subset

I saved the model whenever validation loss decreased. Final testing used the best saved checkpoint.

Best validation loss (Colab): 0.5499993113

2.2 Loss curves (training vs validation)

Figure 1 shows the training and validation loss curves.

- Training loss decreased steadily from ~1.62 to ~0.61
- Validation loss decreased from ~1.26 to ~0.55
- Validation loss stayed close to training loss and did not trend upward near the end

This indicates stable optimization and no major overfitting. Small fluctuations are expected due to minibatch noise and random augmentation.

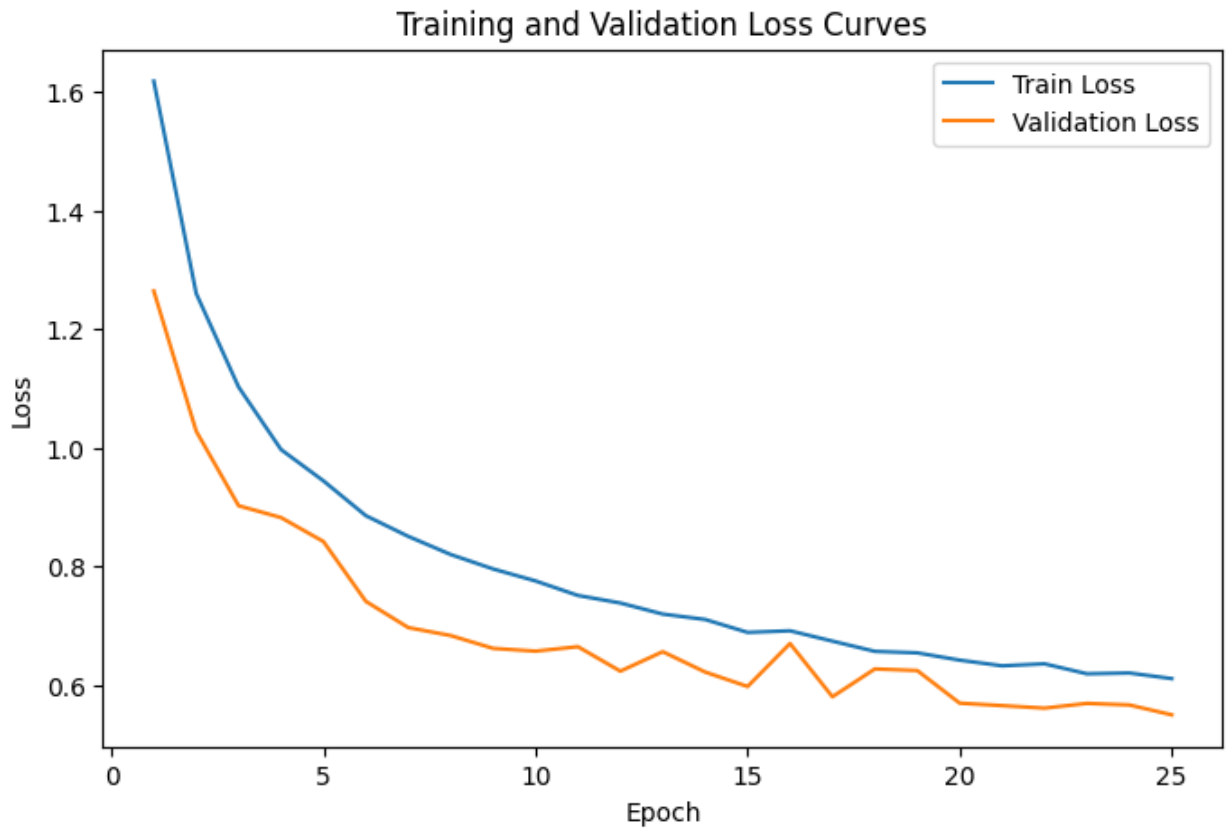


Figure 1. Training and validation loss curves (outputs/loss_curves.png).

2.3 Final test performance (Colab run)

Using the best validation checkpoint, the final test results were:

- Final test loss: 0.5531639125
- Final test accuracy (overall): 80.98% (8098 / 10000)

Per-class accuracies:

- airplane: 84.7%
- automobile: 90.7%
- bird: 66.0%
- cat: 67.5%
- deer: 79.5%
- dog: 72.3%
- frog: 87.9%
- horse: 87.6%
- ship: 89.4%
- truck: 84.2%

This exceeds the typical “sufficient” accuracy range (~65–75%) and shows the CNN learned strong discriminative features.

2.4 Colab (GPU) vs VS Code script (CPU) comparison

I trained the same model in two environments:

VS Code / Python script (CPU)

- Best validation loss: 0.5655084051
- Test loss: 0.5726488047
- Test accuracy: 80.43%
- Seed: 42 (explicit)

Google Colab notebook (GPU)

- Best validation loss: 0.5499993113
- Test loss: 0.5531639125
- Test accuracy: 80.98%

Small differences are expected because training uses random augmentation (crop/flip), data is shuffled, and GPU execution can be nondeterministic unless strict deterministic settings are enabled. Importantly, both runs are consistent (~80% accuracy), confirming the pipeline is stable.

2.5 Qualitative sample predictions

To provide qualitative evidence that the model produces reasonable predictions, I generated a grid of sample test predictions.

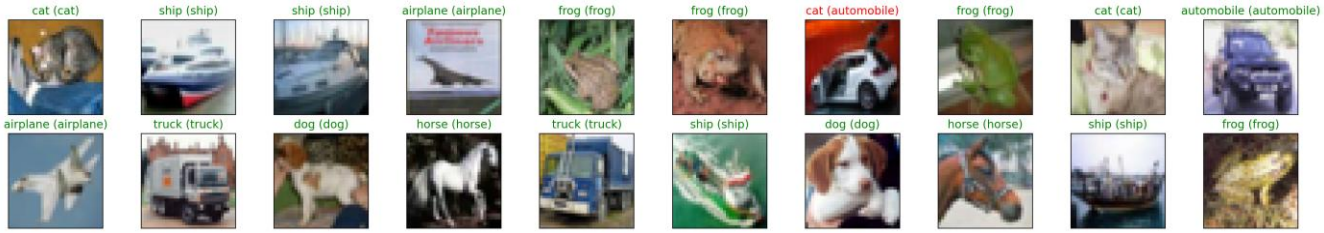


Figure 2. Sample predictions on test batch (outputs/sample_predictions.png).

---

3. Feature Map Visualization (Early Layer)

This section examines what the CNN learns internally by visualizing feature maps from the first convolution layer (Conv1).

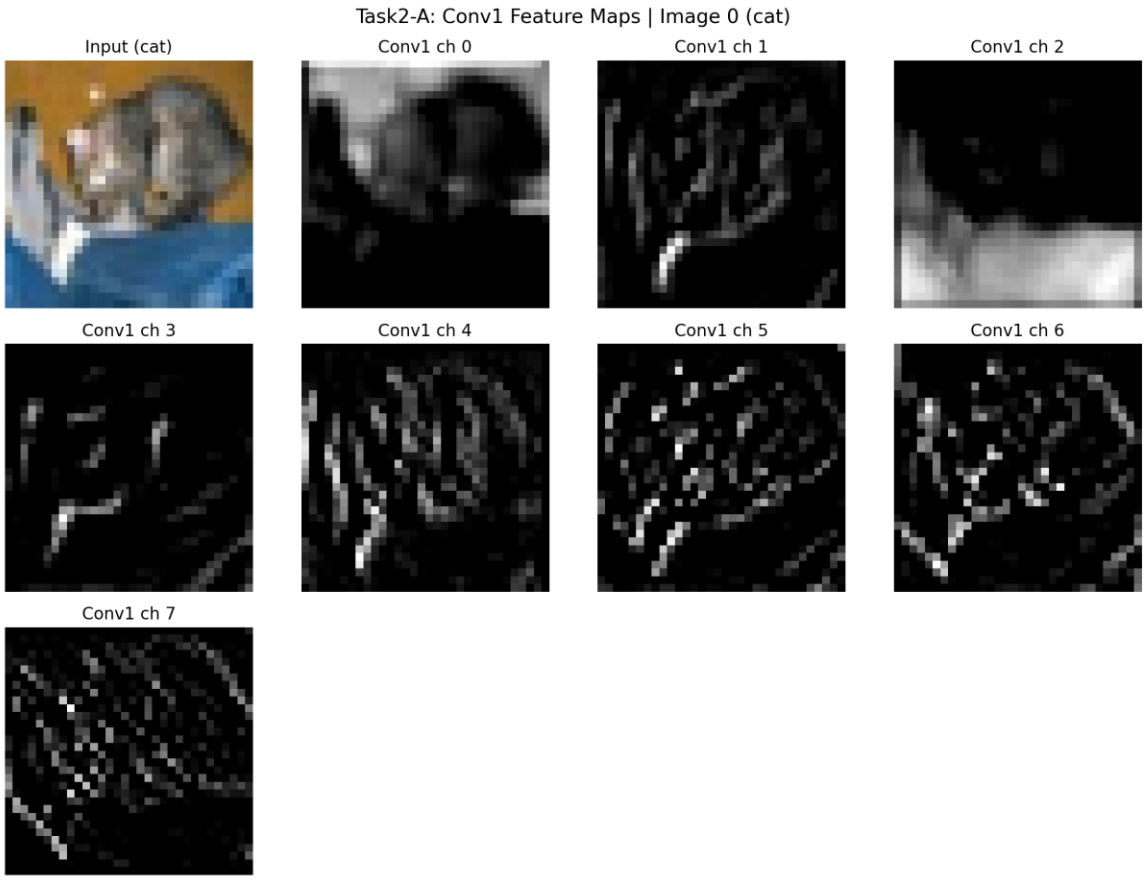
3.1 Method

I selected 3 test images from different classes. For each image:

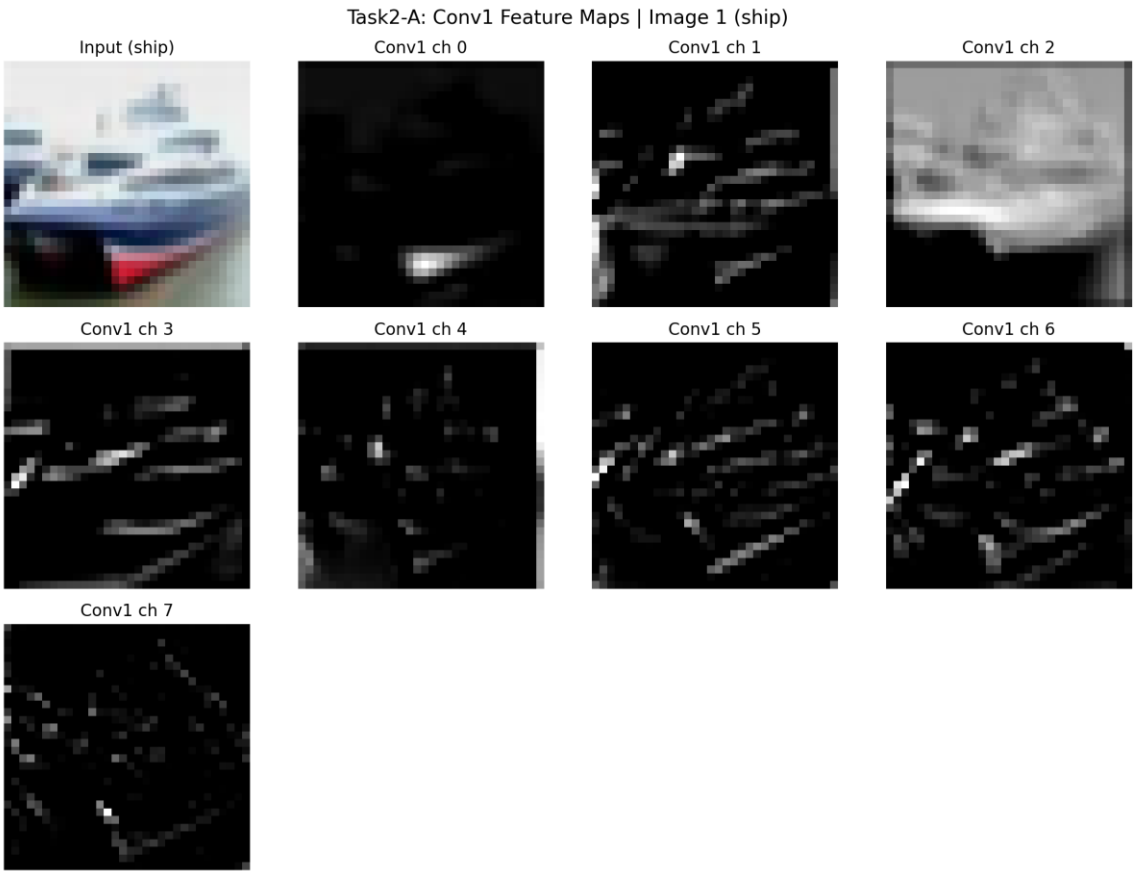
1. I passed the image through the trained network.
2. I extracted the Conv1 output after ReLU (Conv1 activations).
3. Conv1 produces 32 channels, and I visualized 8 feature maps per image to meet the requirement.

Each feature map is a 2D activation map showing where a specific filter responds strongly.

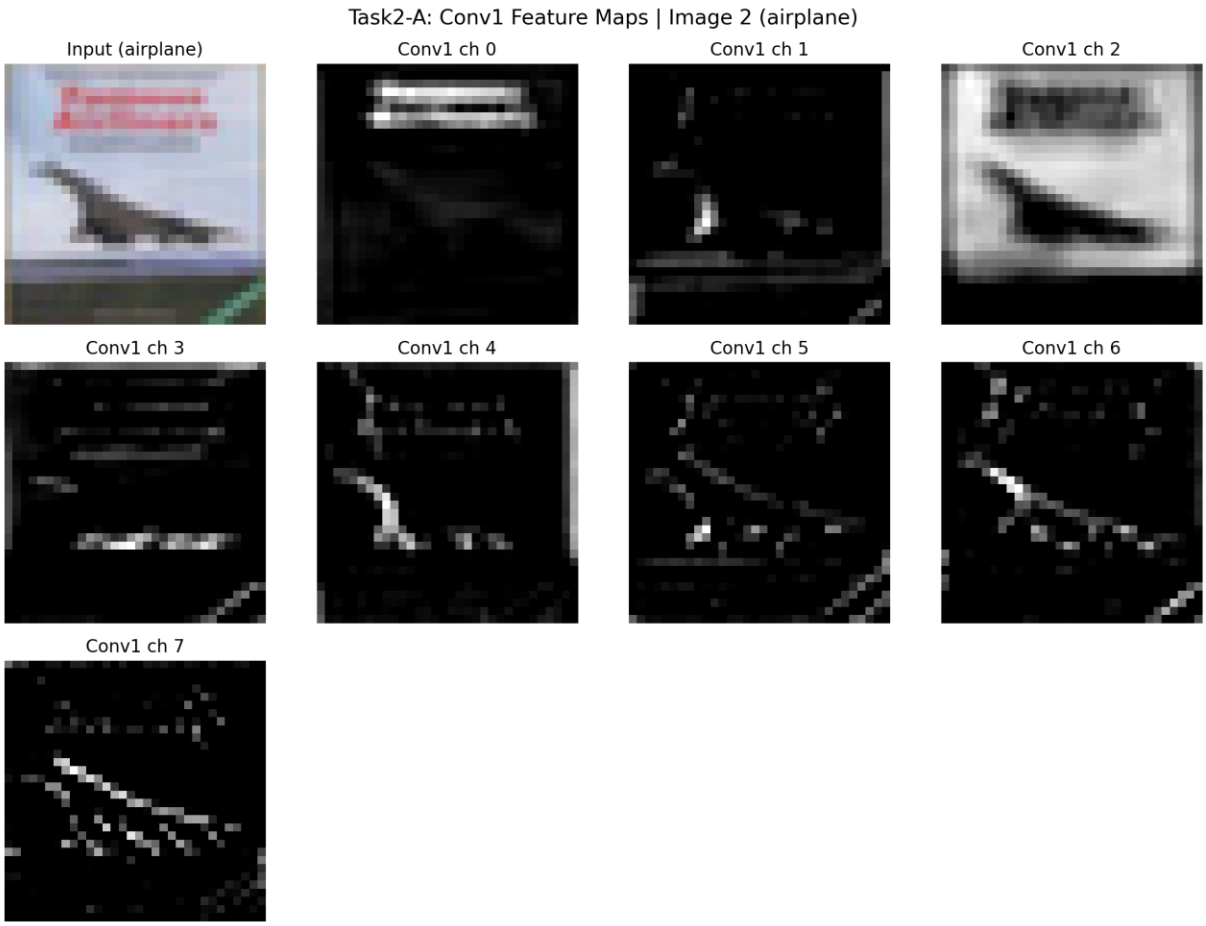
3.2 Results (Figures A1–A3)



- Figure 3. Conv1 feature maps for Image 0 (task2A_conv1_featuremaps_img0.png)



- Figure 4. Conv1 feature maps for Image 1 (task2A_conv1_featuremaps_img1.png)



- Figure 5. Conv1 feature maps for Image 2 (task2A_conv1_featuremaps_img2.png)

3.3 Interpretation (what Conv1 filters detect)

Across the three images, Conv1 feature maps showed typical early-layer CNN behavior:

- Edge detectors: Several channels activated strongly on sharp intensity transitions, producing bright contours around object boundaries (e.g., outlines of the main object).
- Texture detectors: Some channels responded to repeated local patterns (e.g., fur-like texture, grass-like noise, or repeated small gradients), producing many small activations across textured regions.
- Color/intensity contrast maps: Some channels produced broader activation blobs, separating foreground regions from background based on color or brightness differences.

3.4 How feature maps differ for the same input

For a single image, different channels reacted to different properties:

- Some channels emphasized the main object boundary,
- others emphasized background texture,
- others highlighted corners or specific localized patches.

This indicates Conv1 learns a diverse bank of low-level filters rather than a single “edge map.” These low-level features become the building blocks for more meaningful representations in deeper layers.

---

4. Maximally Activating Images

This section identifies which test images maximally activate selected filters in a deeper layer, to interpret what those filters detect.

4.1 Layer and activation definition

I selected Conv2 as the target layer.

Each Conv2 filter produces a 2D feature map after ReLU. To convert that to a single activation score per image, I used:

Activation definition (mean):
score(image, filter f) = mean(ReLU(feature_map_f))

Mean activation measures overall response strength across the feature map (not just one local peak).

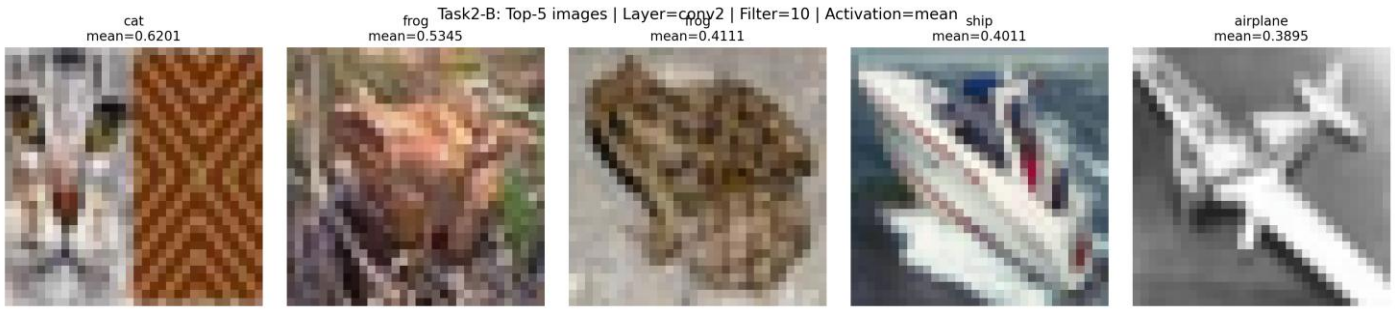
4.2 Method

For each selected Conv2 filter:

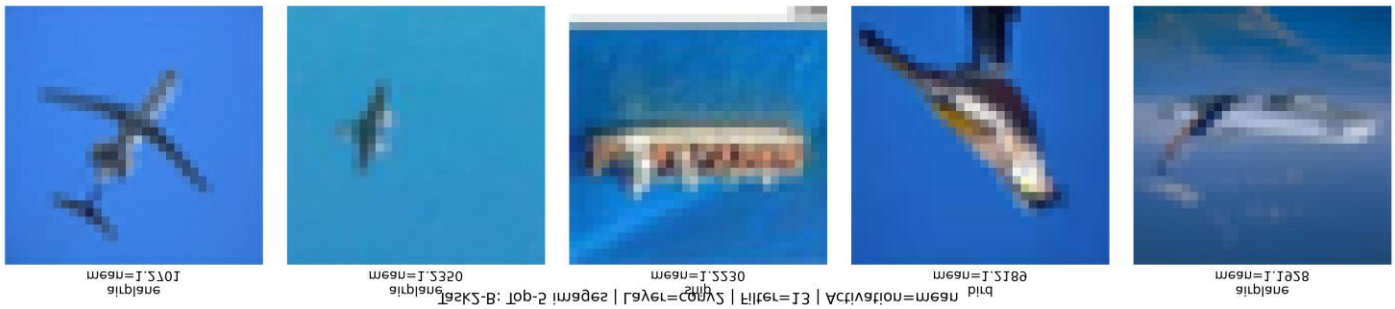
1. I passed every CIFAR-10 test image through the network up to Conv2.
2. I computed the scalar activation score using mean(ReLU(feature_map)).
3. I ranked all test images by score.
4. I selected the top 5 images and visualized them.

I used filters 13, 10, and 20 (filter 0 was excluded because it produced near-zero activations, making interpretation meaningless).

4.3 Results (Figures B1–B3)



- Figure 6. Top-5 images for Conv2 filter 13 (task2B_top5_conv2_filter13.png)



- Figure 7. Top-5 images for Conv2 filter 10 (task2B_top5_conv2_filter10.png)

- Figure 8. Top-5 images for Conv2 filter 20 (task2B_top5_conv2_filter20.png)

4.4 Discussion: common patterns and general vs class-specific behavior

Filter 13 (Conv2) — blue background + silhouette boundary
In Figure 6, the top-5 images are dominated by large smooth blue regions (sky/water) and a relatively darker foreground object (airplane/ship/bird). This suggests filter 13 responds strongly to a combination of (1) broad low-texture blue regions and (2) clean high-contrast object boundaries against that background. This filter appears to detect a general visual feature (color + contrast + boundary structure), but it is dataset-biased toward classes that frequently appear with blue backgrounds (airplane, ship, bird).

Filter 10 (Conv2) — high-frequency texture + diagonal edge structure
In Figure 7, the top-5 images share strong high-frequency texture (e.g., repeated zig-zag

patterns) and sharp angled boundaries. Frogs and cats appear alongside ship and airplane, indicating that the filter is not class-specific. The common trigger is texture density and directional edge energy (particularly diagonal/angled structures). This filter responds to general mid-level features, not a single class.

Filter 20 (Conv2) — warm color dominance + soft texture
In Figure 8, the top-5 images are primarily dogs (plus one deer) and share strong warm red/orange/brown tones with relatively smooth, soft textures (fur-like or warm indoor background regions). This suggests filter 20 responds to a general feature combining warm color dominance with low-to-mid texture structure. Because many top-5 images are dogs, this filter is class-leaning, but the evidence indicates it is still primarily reacting to visual properties (warm color + texture) rather than "dog-ness" as a semantic concept.

4.5 Summary for Task 2B

Conv2 filters respond to more structured combinations of features than Conv1. While Conv1 mostly detects edges and simple textures, Conv2 filters tend to capture patterns like specific background color fields, texture frequency, and combined boundary/texture cues. Some filters are general-purpose; others become class-leaning due to dataset correlations (background + lighting + typical scene composition).

---

5. Brief Discussion and Reflection

5.1 What I learned from Task 1 (training)

The CNN trained stably on CIFAR-10 and achieved ~81% test accuracy. The loss curves showed consistent improvement and no major overfitting, which indicates the architecture and regularization (dropout + weight decay) were appropriate for this task.

5.2 What I learned from Task 2 (interpretability)

Feature map visualizations confirmed the expected CNN hierarchy:

- Conv1 learns low-level primitives (edges, color contrasts, simple textures).
- Conv2 begins to combine primitives into mid-level patterns (texture frequency, boundary structure, background field cues).

The maximally activating image analysis also showed that filters can become "class-leaning" due to dataset bias (e.g., airplanes often appear in blue skies), even if the filter is fundamentally responding to general visual properties rather than semantic class identity.

5.3 Limitations and possible improvements

This analysis is based on a small set of selected images/filters. A more thorough study could:

- visualize more Conv2/Conv3 filters,
- compare mean vs max activation definitions,
- and examine whether certain filters correlate strongly with specific classes across the full dataset.