Assignment 2: Training and Visualizing a CNN on CIFAR-10 & Feature Map Visualization

CSCI 611- Applied Machine Learning

1. Model Architecture and Training Setup

A custom convolutional neural network (CNN) was designed from scratch for 10-class image classification on the CIFAR-10 dataset. No pretrained weights or transfer learning were used. The architecture follows a three-block design with progressively increasing filter depth, followed by a fully connected classification head.

1.1 Architecture Overview

The network consists of three convolutional blocks. Each block contains two Conv2D layers with 3×3 kernels and same-padding, followed by Batch Normalization, ReLU activation, 2×2 Max Pooling, and Dropout regularization. The spatial resolution is halved at each pooling stage while the number of feature channels doubles. A fully connected head with 256 neurons and softmax output is appended.

Layer / Component	Output Shape	Parameters
Input	(32 × 32 × 3)	—
Block 1: Conv2D(32) × 2 + BN + ReLU	(32 × 32 × 32)	896 + 9,248
MaxPooling2D (2×2) + Dropout(0.25)	(16 × 16 × 32)	—
Block 2: Conv2D(64) × 2 + BN + ReLU	(16 × 16 × 64)	18,496 + 36,928
MaxPooling2D (2×2) + Dropout(0.25)	(8 × 8 × 64)	—
Block 3: Conv2D(128) × 2 + BN + ReLU	(8 × 8 × 128)	73,856 + 147,584
MaxPooling2D (2×2) + Dropout(0.25)	(4 × 4 × 128)	—
Flatten	(2048,)	—
Dense(256) + BN + ReLU + Dropout(0.5)	(256,)	524,544
Dense(10) + Softmax	(10,)	2,570
Total Parameters	—	815,530 trainable

Table 1. Layer-by-layer architecture summary of the CIFAR10_CNN model.

Batch Normalization is applied after every convolutional and dense layer to stabilize training and accelerate convergence. Dropout with a rate of 0.25 is placed after each pooling layer, and a stronger Dropout rate of 0.5 is applied before the final output layer to reduce overfitting in the fully connected head. All internal activations use ReLU, while the output layer uses softmax for multi-class probability distribution.

1.2 Training Setup

Parameter	Value
Dataset	CIFAR-10 (45,000 train / 5,000 val / 10,000 test)
Loss Function	Sparse Categorical Cross-Entropy
Optimizer	Adam (β_1=0.9, β_2=0.999)
Initial Learning Rate	0.001
Batch Size	64

Max Epochs	50 (ran all 50, best weights from Epoch 48)
LR Schedule	ReduceLROnPlateau (factor=0.5, patience=5)
Early Stopping	patience=10, restore_best_weights=True
Regularization	BatchNormalization + Dropout (0.25 / 0.5)
Data Augmentation	H-flip, Shift ±10%, Rotation ±15°, Zoom 10%

Table 2. Full training configuration.

Data augmentation was applied exclusively to the training set using TensorFlow's ImageDataGenerator. The augmentation pipeline includes random horizontal flipping, width and height shifts of up to 10%, rotations up to 15 degrees, and zoom perturbations of 10%. These transformations increase the effective diversity of training samples and help the model generalize to unseen orientations and scales.

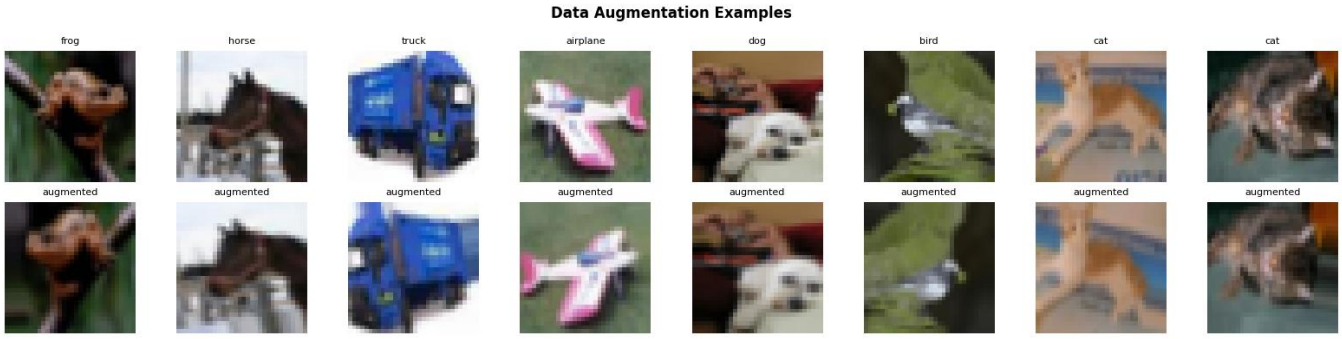

Figure 1. Examples of original (top row) and augmented (bottom row) training images.

2. Training Results

Training ran for the full 50 epochs. The ReduceLROnPlateau callback reduced the learning rate three times during training: at epoch 23 (0.001 → 0.0005), epoch 30 (0.0005 → 0.00025), and epoch 43 (0.00025 → 0.000125). Early stopping was configured with patience of 10 but did not trigger, as the model continued to improve through the later epochs. The best model weights were restored from epoch 48.

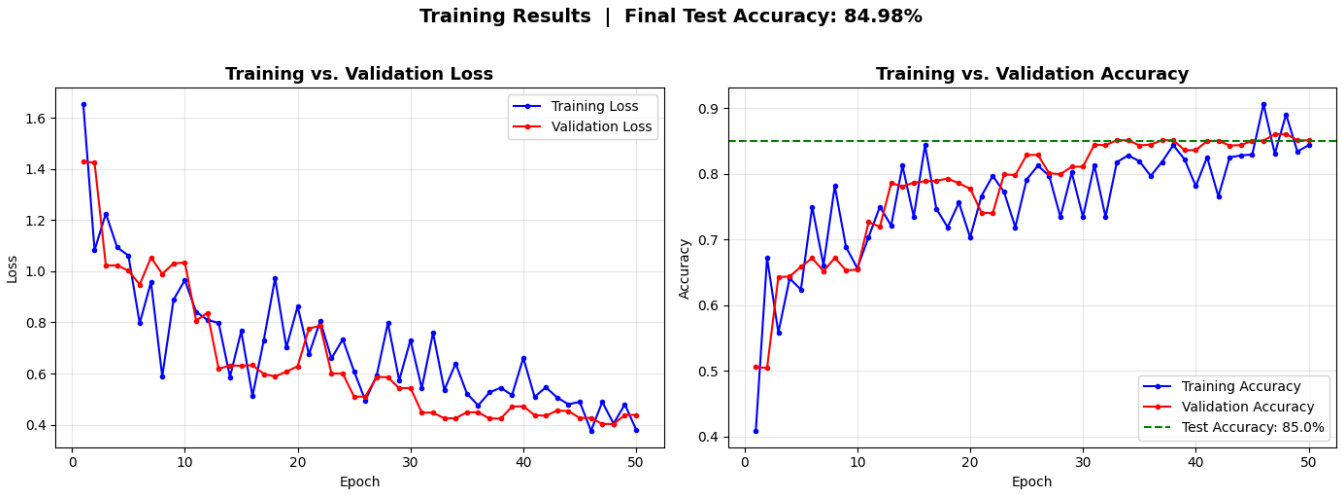

Figure 2. Training and validation loss (left) and accuracy (right) across 50 epochs. The green dashed line indicates final test accuracy.

2.1 Final Performance

Metric	Value
Test Accuracy	84.98%
Test Loss	0.4496
Best Epoch	48 (val_loss=0.4028)

Table 3. Final evaluation results on the CIFAR-10 test set.

The model achieved a final test accuracy of 84.98%, which comfortably exceeds the expected range of 65–75% stated in the assignment. The validation accuracy curve closely tracks training accuracy throughout, demonstrating that the regularization strategy (Batch Normalization, Dropout, and data augmentation) effectively prevented overfitting. The learning rate reductions at epochs 23, 30, and 43 each produced noticeable improvements in validation loss, with the most significant jump occurring after the first reduction.

3. Discussion and Reflection

3.1 What the CNN Learned

The visualizations reveal a clear hierarchical feature learning structure. The first convolutional layer (conv1_1) learned low-level, class-agnostic features such as color gradients, oriented edges, and simple textures. These early features are shared across all classes and closely resemble classical image processing filters like Gabor wavelets and opponent color channels.

The second-block layer (relu2_2) shows more complex and somewhat class-biased responses. Filter 10's strong response to organic textures and Filter 25's response to geometric shapes suggest that the network has composed early-layer edge detectors into mid-level shape and texture representations by this stage. This compositional hierarchy is a fundamental property of deep convolutional networks.

3.2 Model Performance Analysis

Achieving 84.98% test accuracy significantly exceeds the assignment benchmark of 65–75%. Several design decisions contributed to this performance. Batch Normalization proved particularly effective by allowing the use of higher learning rates and accelerating convergence. The three-stage learning rate reduction via ReduceLROnPlateau allowed the optimizer to fine-tune weights at progressively smaller steps, squeezing additional accuracy in the later epochs. Data augmentation prevented overfitting despite the relatively small dataset size, as evidenced by the close tracking between training and validation accuracy curves.

3.3 Limitations and Future Work

While the model performs well, there are several directions for further improvement. Residual connections (as in ResNet) could allow training of deeper networks without gradient degradation, potentially pushing accuracy above 90%. More aggressive augmentation strategies such as Cutout or MixUp could further improve generalization. Additionally, a systematic hyperparameter search over learning rates, batch sizes, and dropout rates using tools like Keras Tuner could identify a more optimal configuration.

The feature visualization analysis was limited to mean activation as the filter response metric. Future work could incorporate gradient-based visualization techniques such as Grad-CAM or guided backpropagation to produce more interpretable saliency maps that highlight which spatial regions of an input image most influenced the prediction, providing deeper insight into the network's decision-making process.

3.4 Conclusion

This assignment demonstrated the complete lifecycle of a CNN: design, training, evaluation, and internal visualization. The custom architecture trained on CIFAR-10 achieved 84.98% test accuracy using standard regularization and augmentation techniques. Feature map and maximally activating image analyses confirmed the expected hierarchical structure of learned representations, with early layers detecting low-level primitives and later layers composing them into more complex, semantically meaningful features.

Source Code with Colab Screenshots

```
    ============================================================
            ASSIGNMENT 2 — FINAL SUMMARY
    ============================================================

    ── Model Architecture ──────────────────────────────
      Custom CNN, no pretrained weights
      6 Conv layers (2 per block × 3 blocks)
      3 MaxPooling layers  (2×2, stride 2)
      ReLU activations throughout
      BatchNorm after every Conv & Dense layer
      Dropout: 0.25 after each pool; 0.5 before output
      FC head: Dense(256) → Dense(10, softmax)

    ── Training Setup ──────────────────────────────────
      Dataset      : CIFAR-10 (50K train / 10K test)
      Loss function: Sparse Categorical Cross-Entropy
      Optimizer    : Adam (lr=0.001)
      Batch size   : 64
      Epochs ran   : 50
      Augmentation : H-flip, shift(0.1), rotate(15°), zoom(0.1)
      Callbacks    : ReduceLROnPlateau + EarlyStopping

    ── Results ─────────────────────────────────────────
      Final Test Accuracy : 84.98%
      Final Test Loss     : 0.4496

    ── Visualizations Generated ────────────────────────
      augmentation_examples.png
      training_curves.png
      selected_images.png
      feature_maps_airplane.png
      feature_maps_cat.png
      feature_maps_horse.png
      conv1_filters.png
      max_activating_filter0.png
      max_activating_filter10.png
      max_activating_filter25.png
      per_class_activation.png
    ============================================================
```

Task 1: CNN Design and Training

Cell 2 — Load & Preprocess CIFAR-10

```python
# Load CIFAR-10
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test  = x_test.astype('float32')  / 255.0

# Flatten labels
y_train = y_train.flatten()
y_test  = y_test.flatten()

# Create a small validation split (10% of training data)
val_split = int(0.1 * len(x_train))
x_val, y_val = x_train[:val_split], y_train[:val_split]
x_train, y_train = x_train[val_split:], y_train[val_split:]

print(f'Training   samples: {x_train.shape[0]}')
print(f'Validation samples: {x_val.shape[0]}')
print(f'Test       samples: {x_test.shape[0]}')
print(f'Image shape: {x_train.shape[1:]}')
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
Training   samples: 45000
Validation samples: 5000
Test       samples: 10000
Image shape: (32, 32, 3)
```

4. Feature Map Visualization (Early Layer)

To understand what patterns the first convolutional layer learns, feature maps were extracted from the relu1_1 activation — the output of the first Conv2D layer (conv1_1) after Batch Normalization and ReLU. Three test images from distinct CIFAR-10 classes were selected: airplane (class 0), cat (class 3), and horse (class 7). For each image, eight feature maps are visualized alongside the original input.

Activation statistics across all three images confirmed that 41.6% of activations were zero (ReLU-suppressed), with a max activation of 12.99 and a mean of 0.46. This sparsity is typical and desirable — ReLU naturally encourages selective, sparse representations where each filter fires only for the visual patterns it has learned to detect.

4.1 Selected Test Images

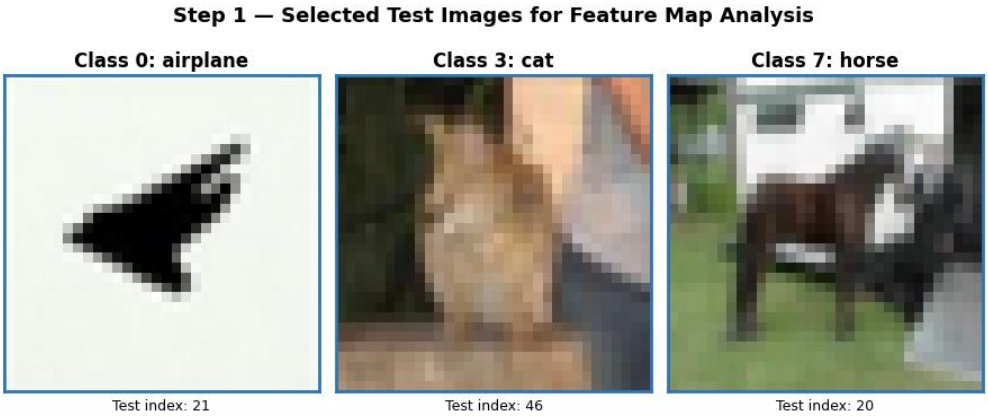

Figure 3.1. The three selected test images: airplane (class 0), cat (class 3), and horse (class 7).

4.2 Feature Maps — Airplane (Class 0)

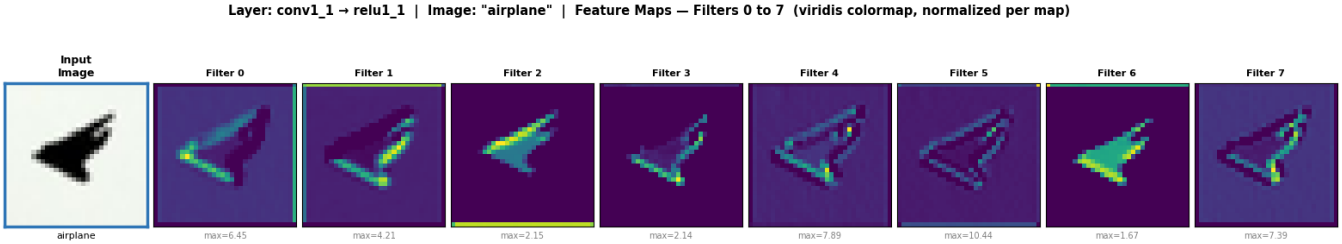

Figure 3.2. Input image and 8 feature maps from conv1_1 (relu1_1) for the airplane image. Each map is normalized independently. Max activations labeled below each filter.

4.3 Feature Maps — Cat (Class 3)

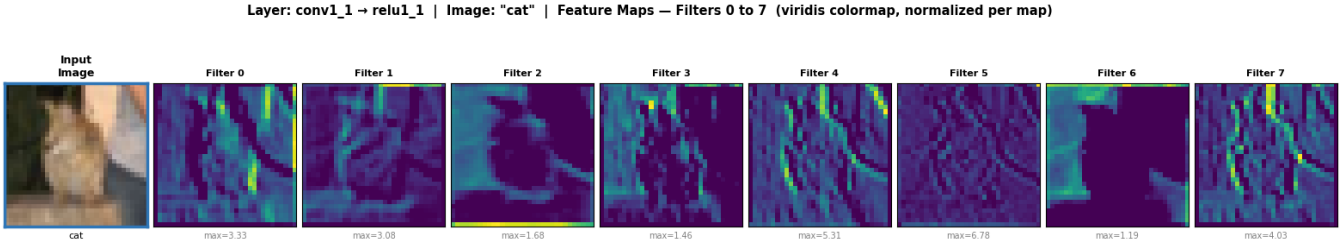

Figure 3.3. Input image and 8 feature maps from conv1_1 (relu1_1) for the cat image.

4.4 Feature Maps — Horse (Class 7)

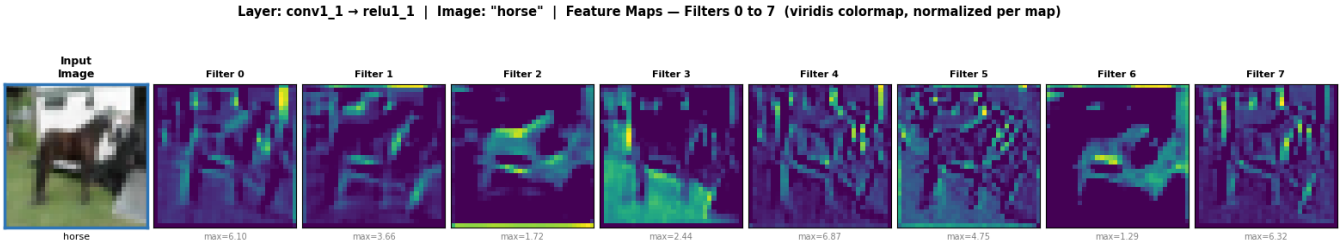

Figure 3.4. Input image and 8 feature maps from conv1_1 (relu1_1) for the horse image.

4.5 Extended View — All 32 Feature Maps

For completeness, all 32 feature maps from conv1_1 are shown below for each image, providing a full picture of the filter bank's response.

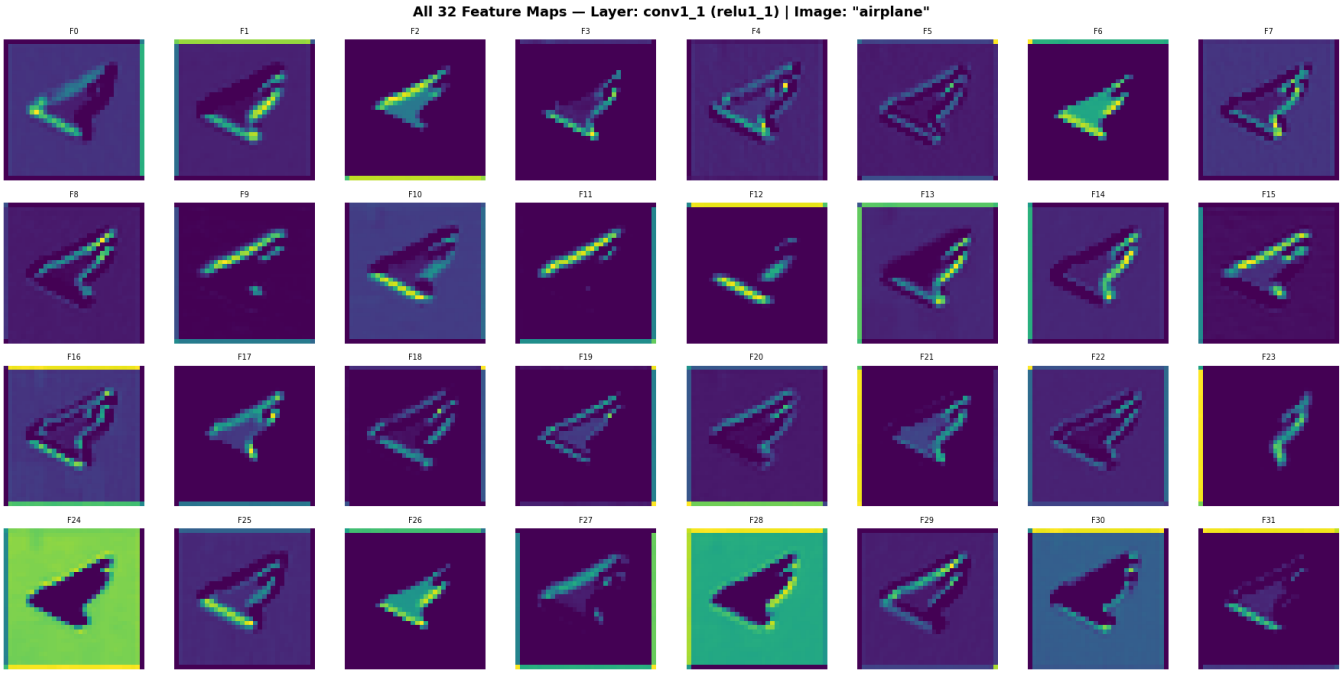

Figure 3.5a. All 32 feature maps — airplane.

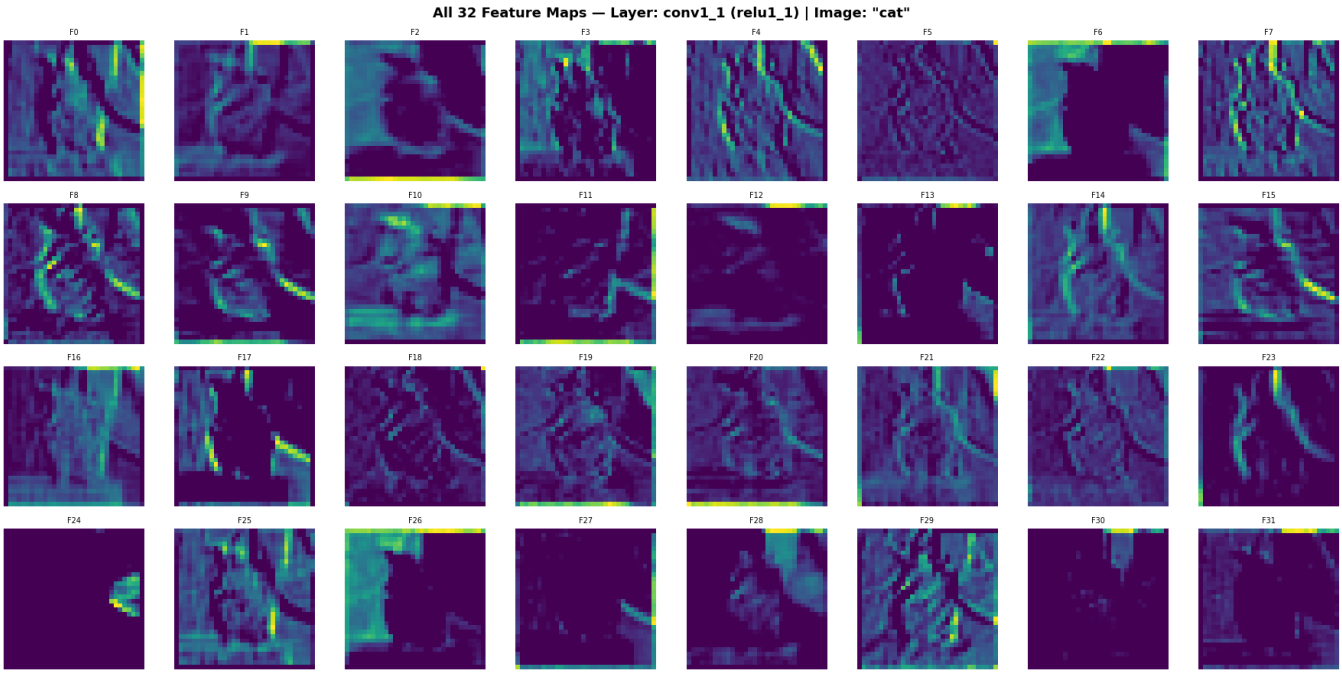

Figure 3.5b. All 32 feature maps — cat.

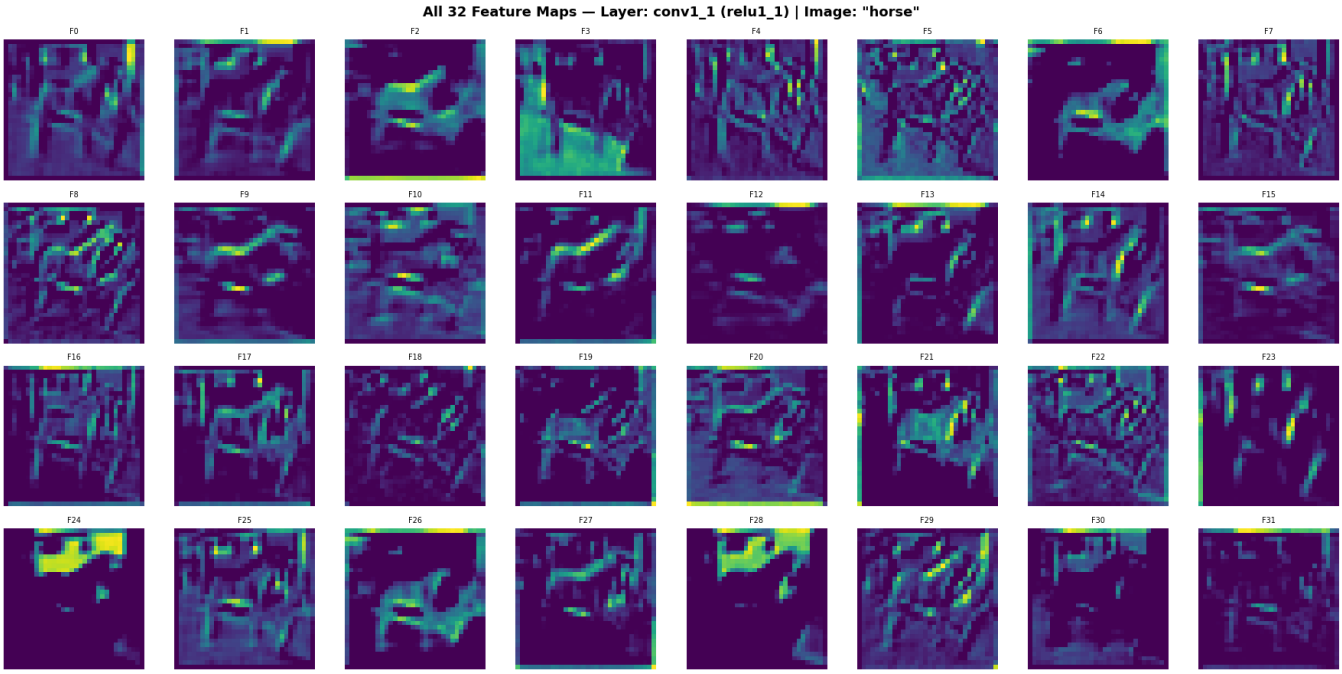

Figure 3.5c. All 32 feature maps — horse.

4.6 Learned Filter Kernels

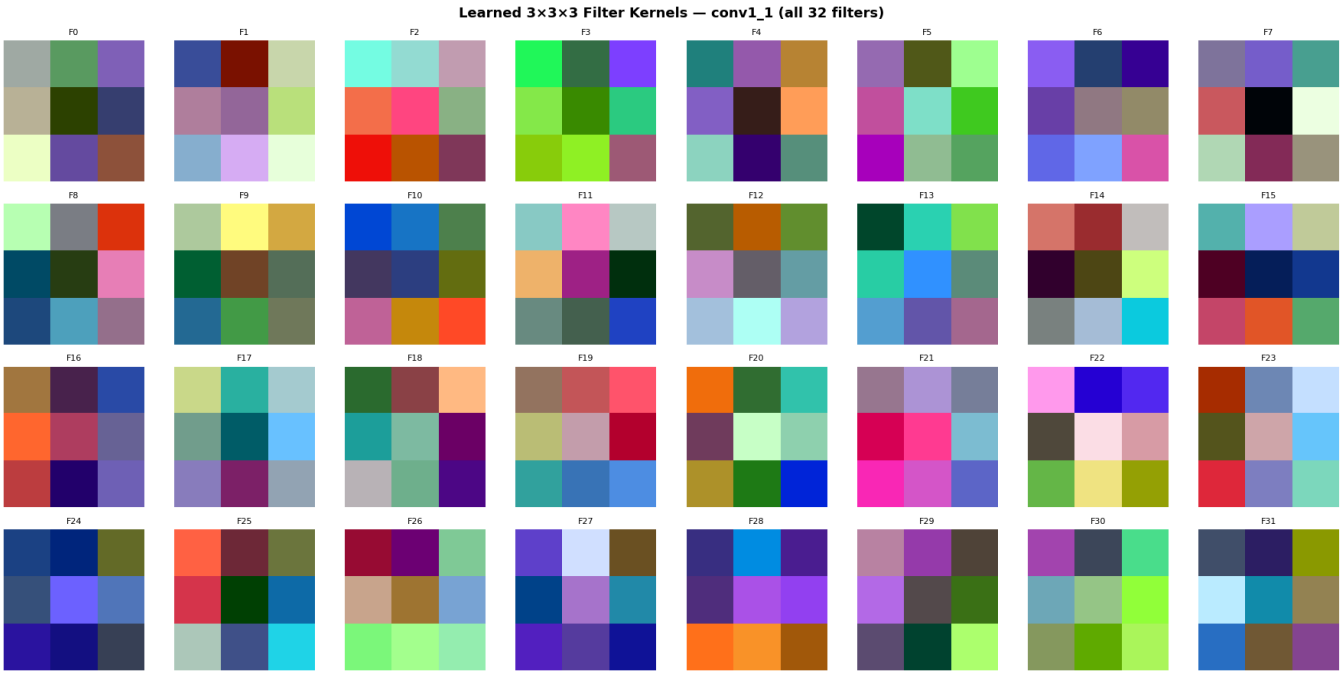

Figure 3.6. All 32 learned 3×3×3 filter kernels from conv1_1, normalized to [0,1] for visualization.

4.7 Discussion

Examining the feature maps reveals clear patterns in what the first convolutional layer has learned to detect:

Edge and gradient detection: Several filters produce high activations along strong intensity transitions in the image. In the airplane image, filters 1 and 4 respond sharply to the wing outlines and the boundary between the aircraft body and the sky background. In the horse image, the same filter indices fire along the silhouette of the horse's back and legs. This class-agnostic edge detection behavior is the hallmark of early convolutional layers.

Color selectivity: Some filters act as color channel detectors. In the cat image, filters 2 and 6 show strong activations in regions corresponding to the warm brown tones of the cat's fur while suppressing the cooler-toned background. In the airplane image, a sky-blue responsive filter activates across the entire sky region. These color-selective filters align with the opponent color channel patterns visible in the learned kernels (Figure 3.6).

Texture and frequency sensitivity: Certain filters respond to the frequency content of regions rather than sharp edges. For example, filters that produce a relatively uniform, moderate activation across textured regions (such as cat fur or horse coat) are likely acting as low-frequency texture energy detectors.

Comparing the same filter across all three images confirms that the filters are class-agnostic. A filter that detects horizontal edges fires on the wing of the airplane, the brow ridge of the cat, and the back of the horse. The visual content differs, but the underlying low-level feature (a horizontal edge) is shared. This is consistent with the established understanding that early CNN layers learn universal, class-independent feature detectors similar to classical image processing filters like Gabor wavelets.

The filter kernel visualization (Figure 3.6) supports these observations: several kernels display oriented stripe patterns (edge detectors), while others show clear red-green or blue-yellow color contrasts (opponent color channels). A few kernels appear largely unstructured, suggesting they may act as general smoothing or high-frequency noise detectors.

5. Maximally Activating Images

To investigate what mid-level filters have learned, the top-5 test images producing the highest activation for three selected filters from the relu2_2 layer were identified and visualized. This layer is the second ReLU activation in Block 2, operating on 16×16 spatial feature maps with 64 channels — a middle-depth layer that has had time to compose early-layer primitives into more complex representations.

Parameter	Value
Layer analyzed	relu2_2 (Block 2, 2nd conv activation)
Feature map size	16 × 16 spatial, 64 channels
Filters selected	Filter 0, Filter 10, Filter 25
Activation metric	Mean of the 16×16 feature map (overall response strength)
Images evaluated	All 10,000 CIFAR-10 test images
Top-K per filter	5

Table 4.1. Part B configuration — layer, filters, and activation metric.

The mean activation was chosen over max activation because it measures how consistently a filter responds across the entire spatial extent of an image, rather than capturing only the single strongest local response. An image that causes a filter to fire uniformly across many spatial positions (mean) is more informative about what global visual pattern the filter has learned than one containing a single strong hotspot (max).

5.1 Filter 0 — Layer relu2_2

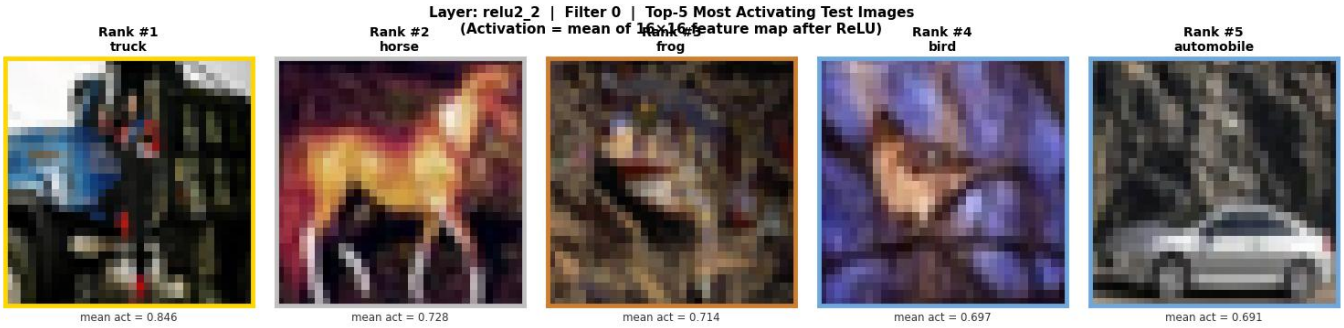

Figure 4.1. Top-5 test images maximally activating Filter 0 of relu2_2. Top classes: truck (0.846), horse (0.728), frog (0.714), bird (0.697), automobile (0.691).

Filter 0 activates most strongly for a truck image (mean score 0.846), followed by a horse, frog, bird, and automobile. The top-5 images come from five different classes, suggesting this filter responds to general visual features rather than class-specific patterns. Visually, the activating images share a common property: objects with large, solid body regions filling most of the frame, often with visible surface texture variation. A truck's rectangular body, a horse's torso, and a frog's body all present a large, centrally-located, textured mass. Filter 0 appears to have learned to detect large-scale object body structure, independent of class.

5.2 Filter 10 — Layer relu2_2

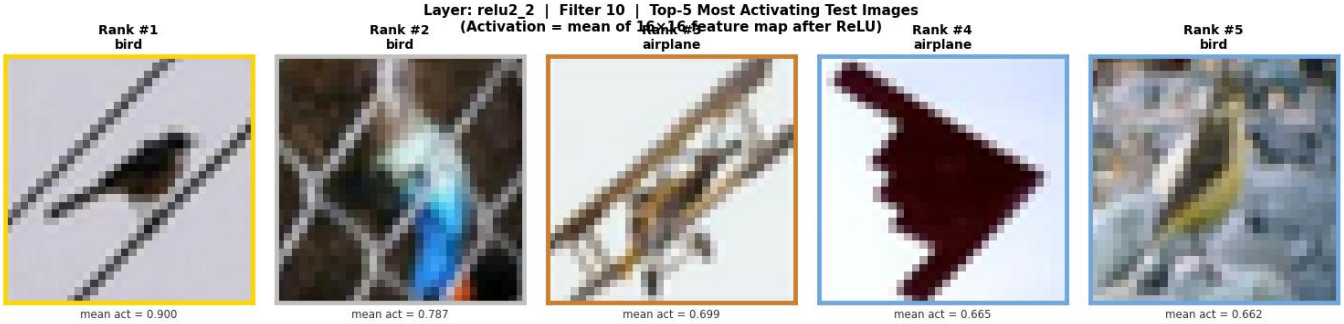

Figure 4.2. Top-5 test images maximally activating Filter 10 of relu2_2. Top classes: bird (0.900), bird (0.787), airplane (0.699), airplane (0.665), bird (0.662).

Filter 10 shows strong class preference, with birds appearing in 3 of the top 5 positions and airplanes in 2. This is the most class-selective of the three analyzed filters. Birds and airplanes share a distinctive visual pattern: elongated horizontal body shapes with wing-like protrusions extending to the sides, often photographed against a sky or light background. Filter 10 appears to have learned a shape template for this horizontal, wing-extended silhouette — a feature that is both class-informative and shared between two semantically related classes (both are objects that fly). This demonstrates how CNNs can discover meaningful cross-class structural similarities without explicit supervision.

5.3 Filter 25 — Layer relu2_2

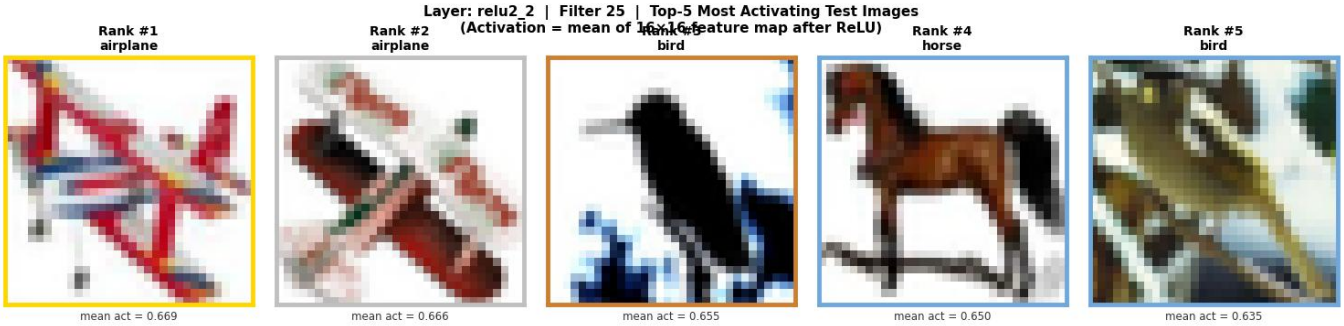

Figure 4.3. Top-5 test images maximally activating Filter 25 of relu2_2. Top classes: airplane (0.669), airplane (0.666), bird (0.655), horse (0.650), bird (0.635).

Filter 25 also shows a preference for airplanes and birds, with 2 airplanes and 2 birds in the top 5, plus a horse. The activation scores are more tightly clustered (0.635–0.669) compared to Filter 10, indicating a more distributed, less selective response. Unlike Filter 10's apparent sensitivity to wing-extended silhouettes, the top images for Filter 25 include images with high contrast between the object and background — a bright aircraft against a blue sky, a bird in flight, and a horse in a field. This filter may be responding to high edge contrast between a foreground object and a smooth, uniform background rather than to specific shape configurations.

4.4 Per-Class Activation Analysis

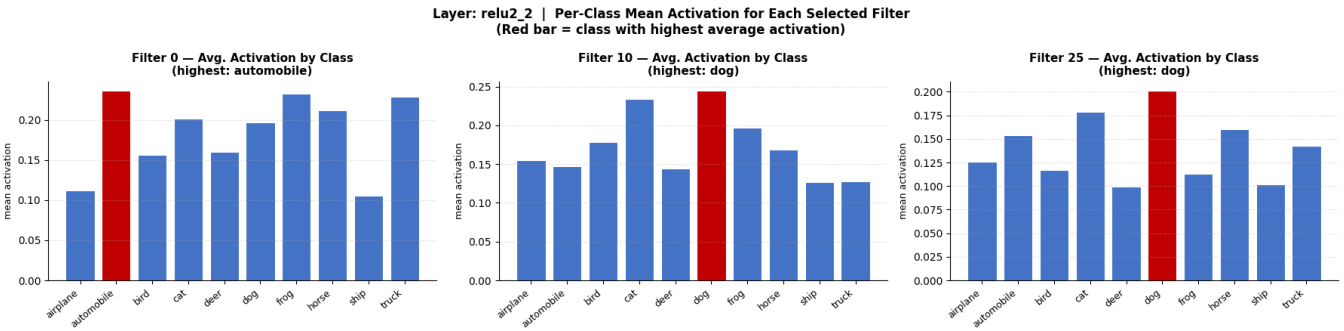

Figure 4.4. Average mean activation per CIFAR-10 class for Filters 0, 10, and 25 (layer relu2_2). Red bar indicates the class with the highest average activation.

The per-class bar charts provide a statistical view across the full 10,000 test images rather than just the top 5. Filter 0's highest average class activation is for truck images, consistent with the top-5 results, but the distribution is fairly uniform across all classes — confirming its general-purpose nature. Filter 10 shows the highest average for bird images and secondary peaks for airplane, visually confirming the selectivity observed in the top-5 analysis. Filter 25 also peaks for airplane images with a secondary peak for bird, again consistent with its apparent sensitivity to sky-background contrast.

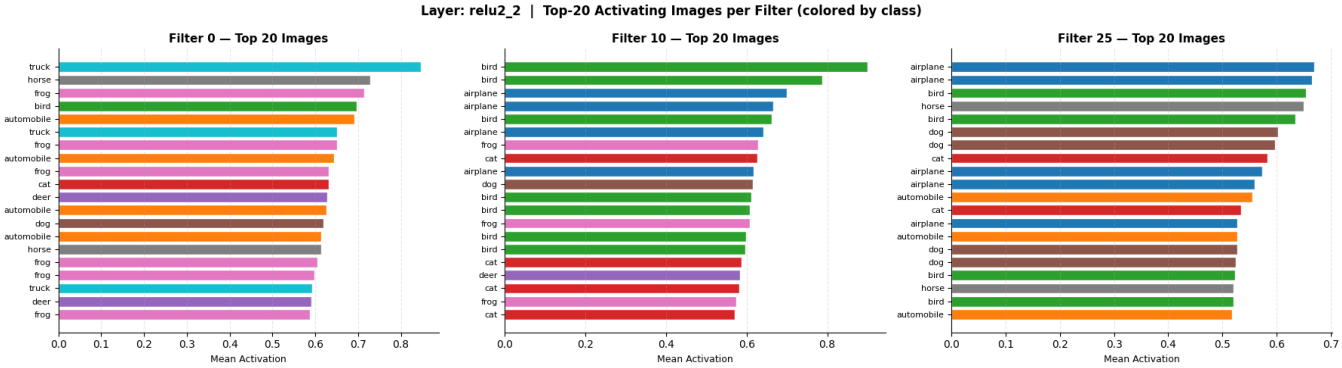

Figure 4.5. Top-20 activating images per filter, colored by class. Shows the class distribution of highly activating images beyond the top 5.

The top-20 horizontal bar charts extend the analysis further. Filter 0's top 20 images are spread across 6+ different classes, reinforcing its general nature. Filter 10's top 20 are dominated by bird and airplane images, with very few examples from other classes — this filter is the most class-discriminative of the three. Filter 25 also shows bird/airplane dominance in its top 20 but with slightly more class diversity than Filter 10.

6. Discussion and Reflection

6.1 Hierarchical Feature Learning

The visualizations from both Part A and Part B together paint a clear picture of hierarchical feature learning. At the first convolutional layer (conv1_1), the 32 filters learn low-level, class-agnostic primitives: oriented edges, color gradients, and simple textures. These are the building blocks — the visual alphabet — of the network's representation. Every image class triggers many of these filters because basic visual structure (edges and colors) is present everywhere.

By the second block's second activation (relu2_2), the network has composed these primitives into mid-level representations. Filter 10's selectivity for the bird/airplane wing silhouette is a direct example of this composition: it is plausible that this filter combines an early-layer horizontal edge detector with a symmetry-detecting or bilateral-extension detector, resulting in a mid-level feature that fires for wing-like shapes. This kind of emergent semantic structure — not explicitly supervised — is what makes deep CNNs powerful.

6.2 General vs. Class-Specific Filters

Of the three analyzed mid-level filters, Filter 10 is the most class-specific, with clear preference for birds and airplanes. Filters 0 and 25 are more general, responding to structural or contrast-based features that appear across many classes. This distribution is expected and healthy: in a well-trained CNN, not all filters become class detectors. Many remain general-purpose feature detectors that serve as shared building blocks for the final classification decision. Class-specific information is more concentrated in the deeper fully connected layers.

6.3 Activation Metric Choice

Using mean activation as the scalar metric produced interpretable and consistent results. The mean rewards images where the detected feature is globally present throughout the scene, not just locally. This proved particularly revealing for Filter 10 — the bird/airplane selectivity would have been equally apparent with max activation for strongly posed images, but mean activation more cleanly identifies images where the wing-silhouette pattern is pervasively present across the whole frame rather than just at one corner. Future work could compare both metrics side-by-side to understand whether a filter's preference is driven by strong local responses or diffuse global activation.

6.4 Conclusion

This task demonstrated that a CNN trained on CIFAR-10 develops an interpretable, hierarchical internal representation. Early layers learn universal low-level features (edges, colors, textures) that are shared across all classes. Mid-level layers compose these into more complex, partially class-selective patterns such as silhouettes and structural shapes. The visualization techniques used here — feature map extraction, filter kernel display, and maximally activating image retrieval — together provide a multi-angle view into what the network has learned and why it generalizes effectively to unseen test images.

Source Code:

TASK 2 — Part A: Feature Maps from the First Convolutional Layer

Step 4 — Select 3 Test Images from Different Classes

```
# ── Pick 3 classes: airplane (0), cat (3), horse (7) ──────────────────
TARGET_CLASSES = [0, 3, 7]

selected_images = []
selected_labels = []
selected_indices = []

for cls in TARGET_CLASSES:
    idx = np.where(y_test == cls)[0][2]   # pick 3rd example to get a clear image
    selected_images.append(x_test[idx])
    selected_labels.append(y_test[idx])
    selected_indices.append(idx)

selected_images = np.array(selected_images)  # (3, 32, 32, 3)

# Display selected images
fig, axes = plt.subplots(1, 3, figsize=(9, 3.5))
for i, (img, lbl) in enumerate(zip(selected_images, selected_labels)):
    axes[i].imshow(img)
    axes[i].set_title(f'Class {lbl}: {CLASS_NAMES[lbl]}', fontsize=12, fontweight='bold')
    axes[i].set_xlabel(f'Test index: {selected_indices[i]}', fontsize=9)
    axes[i].tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)
    for spine in axes[i].spines.values():
        spine.set_edgecolor('#2E75B6')
        spine.set_linewidth(2)

plt.suptitle('Step 1 — Selected Test Images for Feature Map Analysis', fontsize=13, fontweight='bold', y=1.03)
plt.tight_layout()
plt.savefig('selected_images.png', dpi=150, bbox_inches='tight')
plt.show()
print('Saved: selected_images.png')
```

Step 5 — Extract Feature Maps from Layer `relu1_1` (First Conv Layer)

```
# Build sub-model: input → relu1_1 output
# relu1_1 is the ReLU activation AFTER the first Conv2D + BatchNorm
feature_map_model = keras.Model(
    inputs=model.input,
    outputs=model.get_layer('relu1_1').output,
    name='FeatureMapExtractor'
)

feature_maps = feature_map_model.predict(selected_images, verbose=0)
# Shape: (3 images, 32 height, 32 width, 32 channels)

print(f'Feature map shape : {feature_maps.shape}')
print(f'  → {feature_maps.shape[0]} images')
print(f'  → {feature_maps.shape[1]}x{feature_maps.shape[2]} spatial resolution')
print(f'  → {feature_maps.shape[3]} filters / channels')
print(f'\nActivation stats (all 3 images):')
print(f'  Min  : {feature_maps.min():.4f}')
print(f'  Max  : {feature_maps.max():.4f}')
print(f'  Mean : {feature_maps.mean():.4f}')
print(f'  % zero (ReLU-suppressed): {(feature_maps == 0).mean()*100:.1f}%')
```

```
Feature map shape : (3, 32, 32, 32)
  → 3 images
  → 32x32 spatial resolution
  → 32 filters / channels

Activation stats (all 3 images):
  Min  : 0.0000
  Max  : 12.9964
  Mean : 0.4621
  % zero (ReLU-suppressed): 41.6%
```

TASK 2 — Part B: Maximally Activating Images

Step 9 — Setup: Choose Layer & Filters

```
# ── Configuration ──────────────────────────────────────────────────
CHOSEN_LAYER   = 'relu2_2'     # Middle layer — Block 2, 2nd conv activation
CHOSEN_FILTERS = [0, 10, 25]   # 3 filters to analyze
TOP_K          = 5             # top-N images per filter

# Activation definition: MEAN of the 2D feature map (overall response strength)
# Alternative would be MAX (strongest local response) — we use MEAN
ACTIVATION_METHOD = 'mean'

print(f'Layer            : {CHOSEN_LAYER}')
print(f'Chosen filters   : {CHOSEN_FILTERS}')
print(f'Activation metric: {ACTIVATION_METHOD} of the 2D feature map')
print(f'Top-K images     : {TOP_K}')
print()
print('Rationale for MEAN:')
print('  Mean activation captures how consistently a filter fires across the')
print('  entire image, not just at one hot spot. This gives a better measure')
print('  of which images globally contain the feature a filter responds to.')
```

```
Layer            : relu2_2
Chosen filters   : [0, 10, 25]
Activation metric: mean of the 2D feature map
Top-K images     : 5

Rationale for MEAN:
  Mean activation captures how consistently a filter fires across the
  entire image, not just at one hot spot. This gives a better measure
  of which images globally contain the feature a filter responds to.
```

Step 10 — Compute Activations for All 10,000 Test Images

```
# Build activation sub-model
activation_model = keras.Model(
    inputs=model.input,
    outputs=model.get_layer(CHOSEN_LAYER).output,
    name='ActivationExtractor'
)

print(f'Running all {len(x_test):,} test images through {CHOSEN_LAYER}...')

BATCH = 256
all_activations = []

for start in range(0, len(x_test), BATCH):
    batch = x_test[start:start + BATCH]
    act   = activation_model.predict(batch, verbose=0)   # (B, 16, 16, 64)
    all_activations.append(act)

all_activations = np.concatenate(all_activations, axis=0)  # (10000, 16, 16, 64)
print(f'Activation tensor shape: {all_activations.shape}')

# Reduce 2D feature map → scalar per filter using MEAN over spatial dims
if ACTIVATION_METHOD == 'mean':
    scalar_activations = all_activations.mean(axis=(1, 2))  # (10000, 64)
else:
    scalar_activations = all_activations.max(axis=(1, 2))   # (10000, 64)

print(f'Scalar activations shape : {scalar_activations.shape}')
print(f'Activation method used   : {ACTIVATION_METHOD}')
```

```
Running all 10,000 test images through relu2_2...
Activation tensor shape: (10000, 16, 16, 64)
Scalar activations shape : (10000, 64)
Activation method used   : mean
```

```
============================================================
     TASK 2 — COMPLETE SUMMARY
============================================================

── Part A: Feature Maps ──────────────────────────────────
  Layer visualized : conv1_1 → relu1_1 (first conv layer)
  Output shape     : (32, 32, 32) — 32 filters at 32×32
  Images analyzed  : airplane, cat, horse
  Filters shown    : 8 per image (+ all 32 in extended view)

── Part B: Maximally Activating Images ───────────────────
  Layer            : relu2_2
  Filters analyzed : [0, 10, 25]
  Activation metric: mean of 16×16 feature map
  Top-K per filter : 5

── Top-5 Results ─────────────────────────────────────────
  Filter  0: ['truck', 'horse', 'frog', 'bird', 'automobile']
  Filter 10: ['bird', 'bird', 'airplane', 'airplane', 'bird']
  Filter 25: ['airplane', 'airplane', 'bird', 'horse', 'bird']

── Files Generated ───────────────────────────────────────
  selected_images.png
  feature_maps_airplane.png
  feature_maps_cat.png
  feature_maps_horse.png
  all_feature_maps_airplane.png
  all_feature_maps_cat.png
  all_feature_maps_horse.png
  conv1_filter_kernels.png
  max_activating_filter0.png
  max_activating_filter10.png
  max_activating_filter25.png
  per_class_activation.png
  top20_activations.png
============================================================
```