Project 2: Convolutional Neural Networks

CSE 849 Deep Learning (Spring 2025)

Arman Khoshnevis

March 2025

1 Fine Tuning of the CNN Model

The initial setup for hyperparameter tuning is as follows: `batch_size=64`, `lr = 1e-3`, `lr_min = 1e-5`, `weight_decay = 1e-3`, and the use of `CosineAnnealingLR` along with `RandomHorizontalFlip` and `Normalize` as data augmentation techniques. The corresponding training and validation losses, as well as accuracies, are shown in Figure 1. The following subsections describe the tuning process aimed at improving validation accuracy and reducing overfitting.

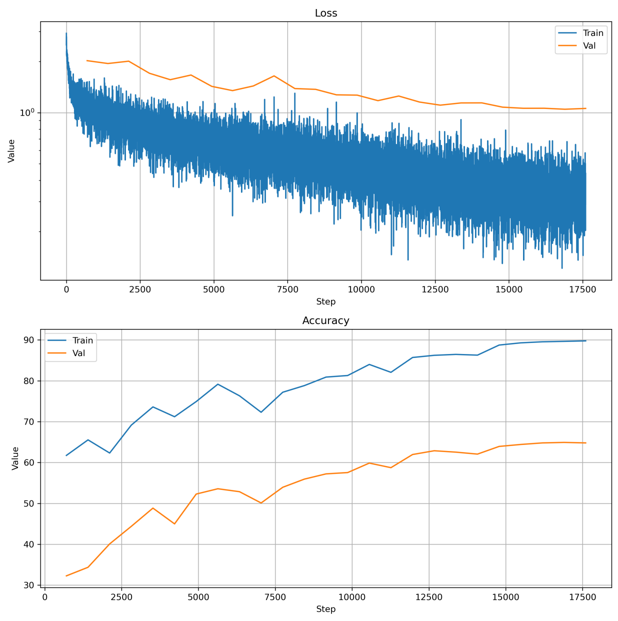


Figure 1: Caption

1.1 Batch Size

The hyper-parameter tuning begins with testing three values of batch size, 64, 32, and 16, out of which the later batch size resulted into the highest validation accuracy of 70.12% (number of epoch was also increased to 40).

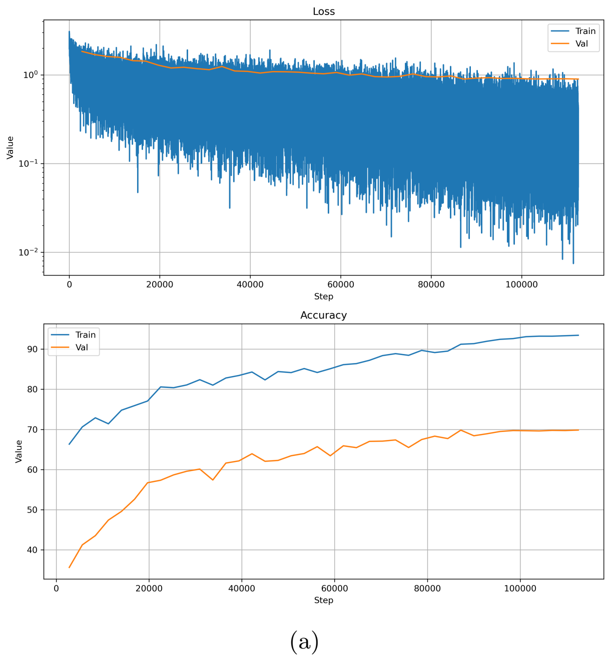
1.2 Data Augmentation

In this experiment, various data augmentation techniques were tested, including `GaussianNoise` with `sigma` values of 0.01, 0.02, 0.05, and 0.1, `RandomErasing` with probabilities of 0.2, 0.3, and 0.4 and scales of

(0.02, 0.05) and (0.02, 0.1), and `RandomRotation` with rotation angles of 5 and 10 degrees. The best-performing combination consisted of `GaussianNoise` with `sigma=0.05` and `RandomErasing` with `p=0.3` and `scale=(0.02, 0.05)`, resulting in a validation accuracy of 69.84%.

1.3 Optimizer

Two key hyperparameters of the `adam` optimizer—`lr` (learning rate) and `weight_decay` (which acts as L2 regularization applied directly to weight updates)—were tested across multiple configurations. First, using a very low learning rate (`lr=1e-4`) resulted in a validation accuracy slightly above 50% (Figurer 2), indicating that the weight updates were too small, leading to suboptimal values. Next, testing three different values of `weight_decay` did not improve validation accuracy beyond 70% (Figure 3). Finally, reducing the minimum learning rate reached by `CosineAnnealingLR` had no significant impact on the results.



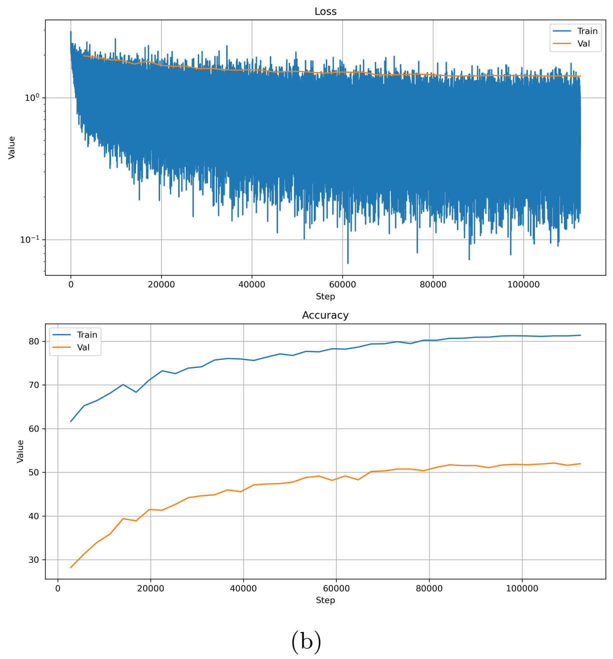
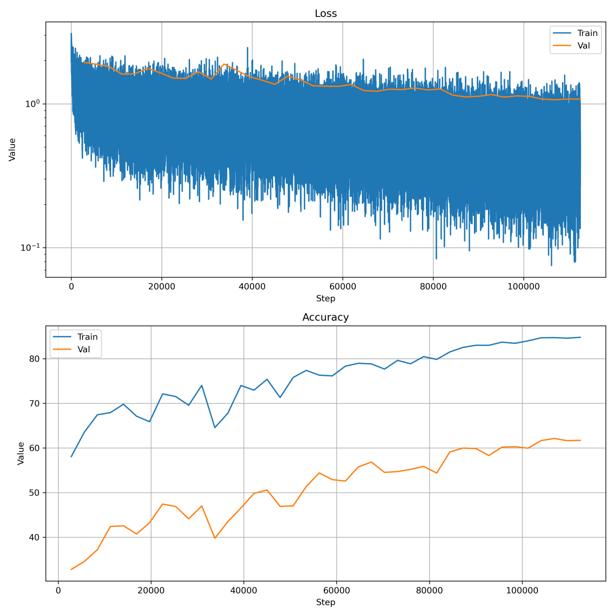
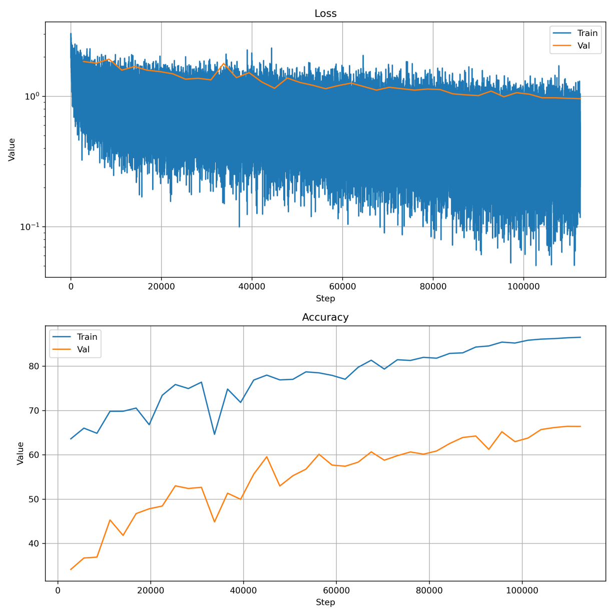


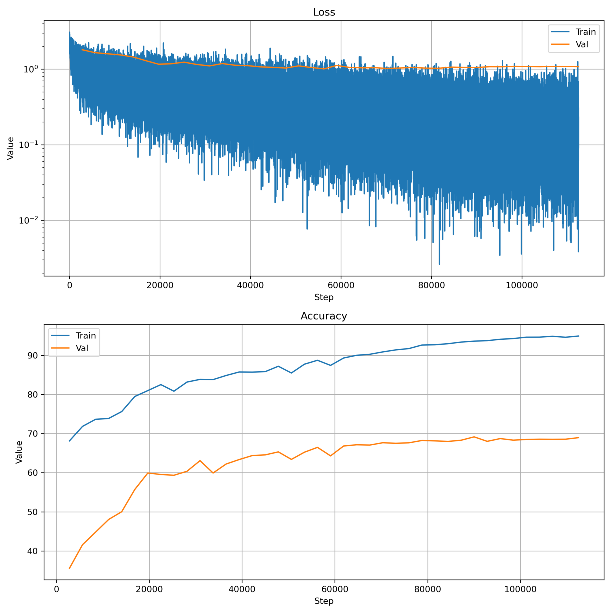
Figure 2: (a) Training and validation losses and accuracies corresponding to `lr=1e-3` and (b) `lr=1e-4`.



(a)



(b)

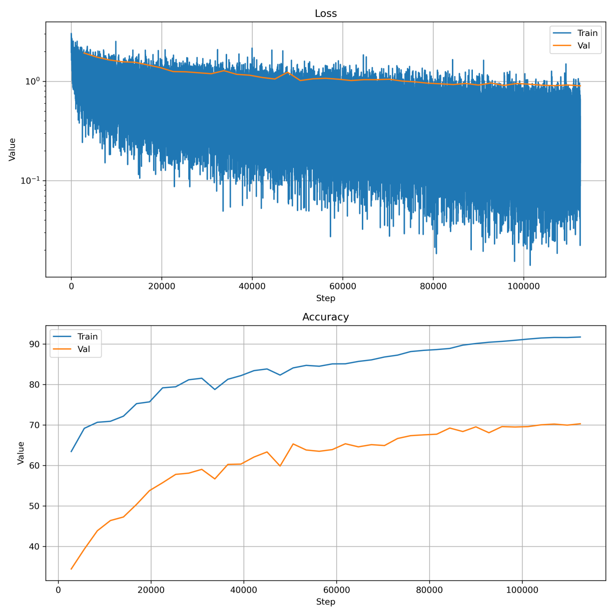


(c)

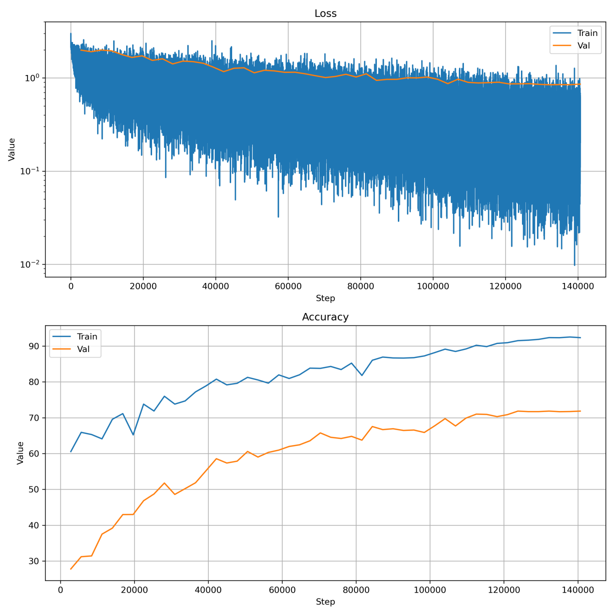
Figure 3: Training and validation losses and accuracies corresponding to (a) `weight_decay=1e-2`, (b) `weight_decay=5e-3`, and (c) `weight_decay=1e-4`.

1.4 Learning Rate Scheduler

While using the `CosineAnnealingLR` seems to be quite effective compared to `StepLR` or a constant learning rate case, using a `SequentialLR` consists of a linear warm-up step (`LinearLR`), followed by a `CosineAnnealingLR` step turns out to improve the validation accuracy up to 71.84%. After a few trial and errors, the hyper-parameters correspond to this scheduler are as follow: `warmup_epochs=10`, `start_factor=0.1`, `lr=1e-3`, `lr_min=1e-5`, `num_epoch=50`. This slight improvement is shown in Figure 4.



(a)



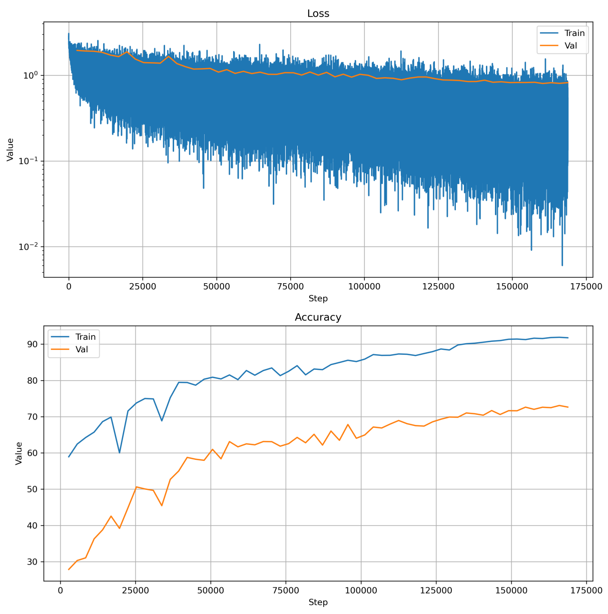
(b)

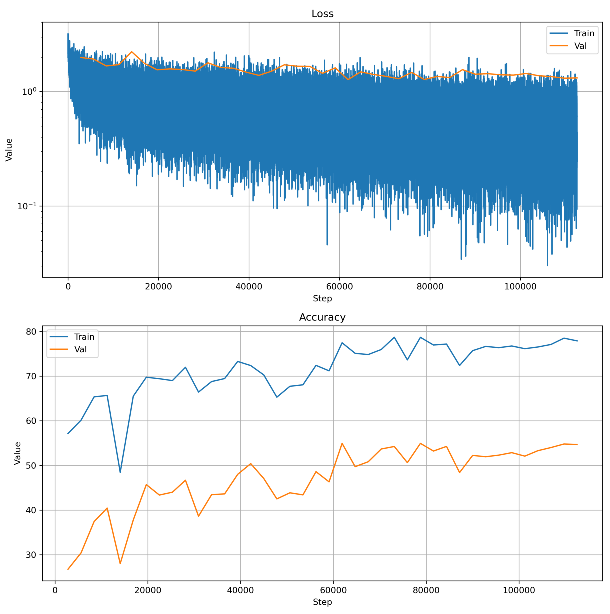
Figure 4: (a) Training and validation losses and accuracies with only `CosineAnnealingLR` scheduler(b) Training and validation losses and accuracies with a sequence of `LinearLR` followed by `CosineAnnealingLR` scheduler.

1.5 Dropout Layers

Dropout layers are commonly used in fully connected feed-forward deep neural networks for regularization, but their application in convolutional layers—particularly when batch normalization is present—is less frequent. Nonetheless, some variations of the current model incorporating dropout have been tested.

Initially, dropout layers were added after each convolutional layer, but this resulted in a noticeable decline in both training and validation accuracy, even with a minimal dropout probability (`p`) of 0.1 (Figure 5(b)). Subsequently, dropout was applied only to deeper layers. Experiments were conducted by introducing dropout in the last one, last two, and last three convolutional layers, using two different `p` values (0.1 and 0.2). The best result, a slight improvement in validation accuracy to 72.64%, was observed when a dropout layer with `p=0.1` was applied to the last three convolutional layers (Figure 5(a)). It worth mentioning that the number of epochs in these tests was increased to 60 to ensure reaching the best possible performance measures.





(a) (b)

Figure 5: (a) Improvement in the training and loss accuracy of the base model via introducing dropout layer (with `p=0.1`) after the last three convolutional layers. (b) Significant negative impact of having dropout layer (with `p=0.2`) after each convolutional layer.

1.6 Final Hyper-Parameters

Ultimately, the best combination of hyper-parameters I arrived with is as follows:

- Data Augmentation and Loading
  - `GaussianNoise(sigma=0.05, clip=False)`
  - `RandomErasing(p=0.3, scale=(0.02, 0.05))`
  - `batch_size=16`
- Model
  - Add three dropout layers after the last three convolutional layers with `p=0.1`.
- Training
  - `num_epochs=60`
  - `lr=1e-3`
  - `min_lr=1e-5`
  - `weight_decay=1e-3`
  - `LinearLR with warmup_epochs=10`
  - `CosineAnnealingLR`

Figure 6 exhibits the training and validation losses and accuracies and the best training and validation accuracies achieved were 80.53% and 73.08%, respectively.

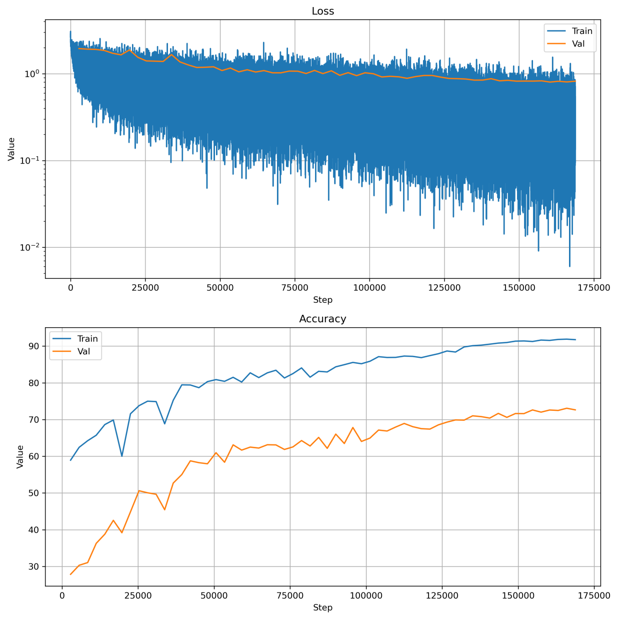


Figure 6: Losses and accuracies of the best model achieved.

2 Analyzing the Trained CNN Model

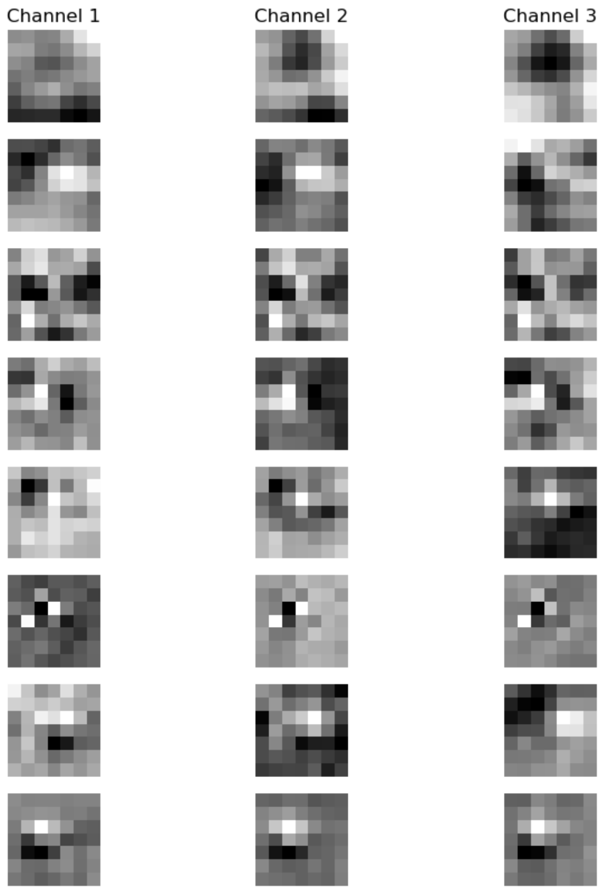
2.1 Filter Visualization

The first aim of this section is to visualize the kernel filters from the first layer of the trained CNN model. The first layer has three input channels (RBG) and 16 output channels, with a kernel size of 7×7. Weights of each kernel filter are extracted from the model first. Then, they are all normalized to the range of [0225] and saved as separate figures. Finally, in order to visualize them with `matplotlib.imshow` tool, they are normalized to the range of $[0, 1]$ and get visualized by gray scale in Figure 7. From some of the filters (for instance, filter 6 of channel 2 and 3, filter 1 and 2 of channel 3, and filter 15 of channel 1), it can be observed that basic preliminary patterns like edges have been captures. That being said, due to size of figures and visualization hurdles, the patterns cannot be rocognized by eyes easily.

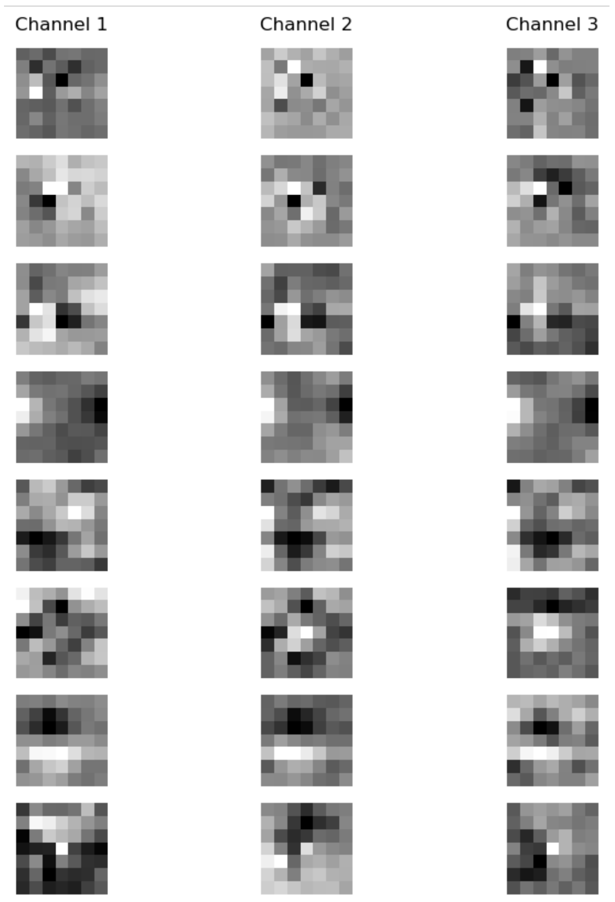
2.2 Compare the First and Last Layer; Who is Expert?

The second aim of this analysis is to understand how different filters in the convolutional layers of your CNN respond to images from different classes. By computing the norm of the feature maps produced by each filter, we can measure how strongly a filter is activated for a given image; especially since each filter in a convolutional layer detects specific features (e.g., edges, textures, patterns) in the input image.

The filters in the first layer detect low-level features that are common to all images, such as edges, corners, and simple textures. As a result, the activations of these filters are uniform across classes (Figures 8(a)-(c)). This is expected because low-level features are not specific to any particular class. The filters in the last layer detect high-level features that are specific to certain classes, such as object parts or complex patterns. As a result, the activations of these filters are class-specific (Figures 8(e)-(g)). Filters that are highly activated for a specific class are likely detecting features unique to that class.



(a)



(b)

Figure 7: Kernel filters of the first convolutional layer of the trained model with 3 and 16 input and output channels, respectively. (a) The first 8 kernel filters, and (b) the second 8 kernel filters.

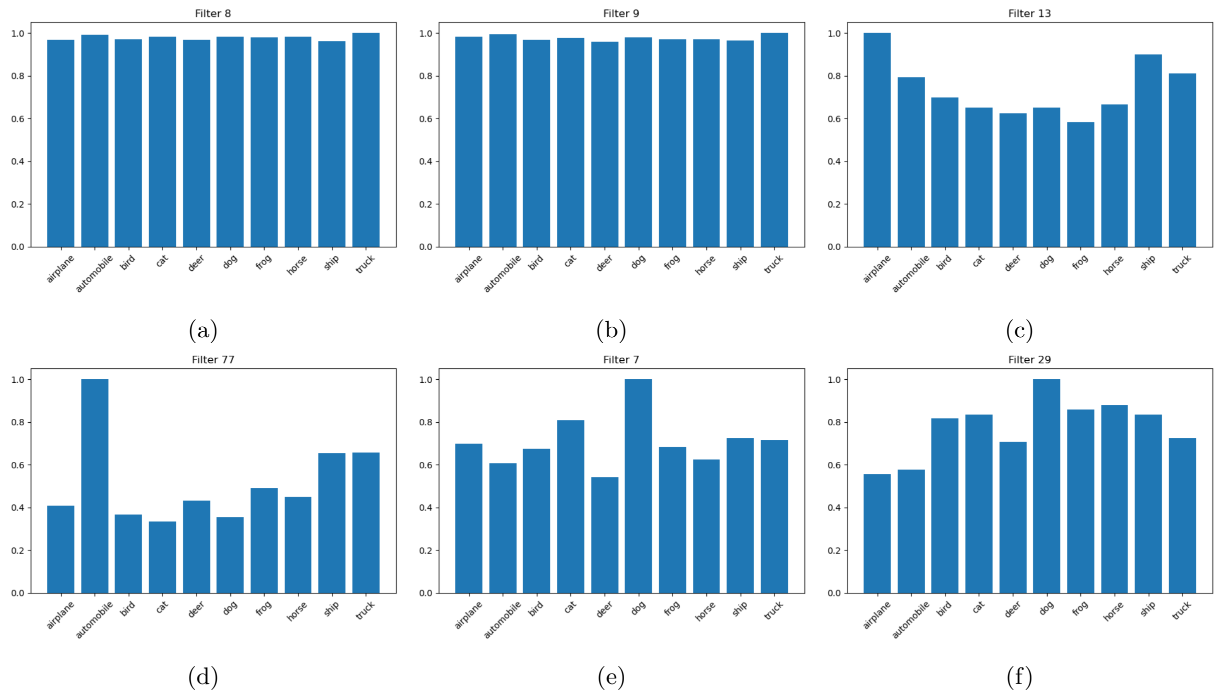


Figure 8: Channel-wise norm of the features after the first and last convolutional layer. Top row exhibits three examples of filters in the first convolutional layer where (a) and (b) have uniform norm distribution of features across various classes and (c) shows a bit of variation across different classes. The bottom row displays the same figures for three different filters. (e) and (f) seem to be more expert in detecting automobile and dog, respectively, while the last filter (g) does not seem to differentiate between classes as much as the other two filters in the same last layer.