Assignment-2
Training and Visualizing a CNN on CIFAR

1. Model Architecture:

The convolutional neural network (CNN) consists of three convolutional layers followed by fully connected layers.

- **Conv1:** 32 filters, 3×3 kernel, padding=1
- **Conv2:** 64 filters, 3×3 kernel, padding=1
- **Conv3:** 128 filters, 3×3 kernel, padding=1
- **MaxPooling:** 2×2 after each convolution
- **Dropout:** 0.25 after flattening
- **Fully Connected Layer 1:** 256 neurons
- **Fully Connected Layer 2:** 10 output neurons (CIFAR-10 classes)
- **Activation Function:** ReLU

After three pooling operations the spatial size reduces from 32×32 to 4×4 before entering the fully connected layers.

Training Setup:

- Optimizer: Adam
- Loss Function: CrossEntropyLoss
- Learning Rate: 0.001
- Batch Size: 20
- Number of Epochs: 20
- Regularization: Dropout (p = 0.25)
- Data Augmentation: None

The dataset was split into training (80%) and validation (20%) sets

2. Training Results:

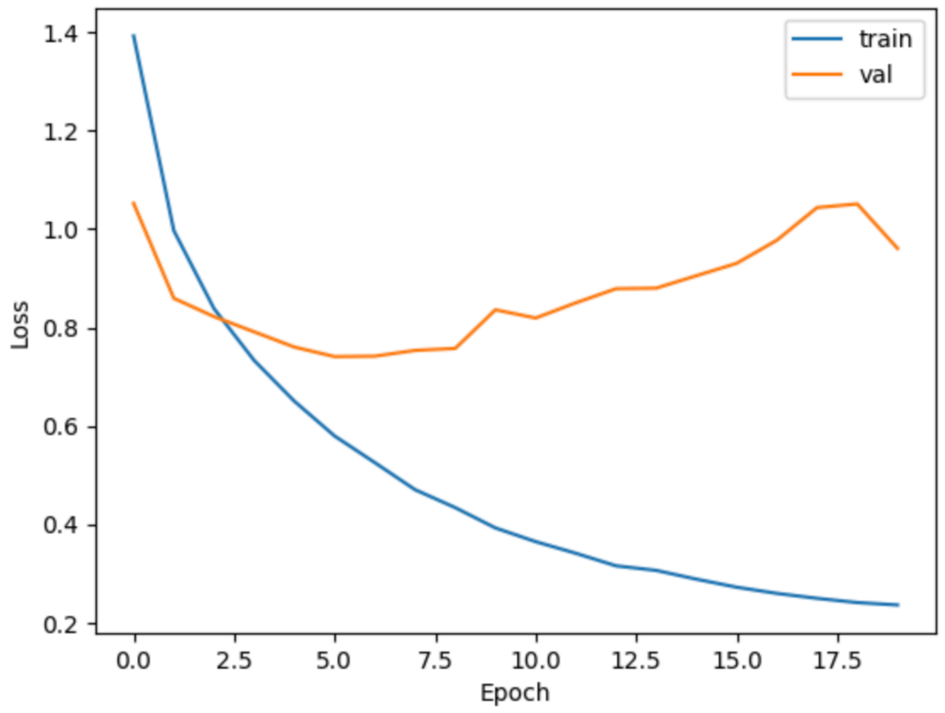
Final Test Accuracy: **74.29%**

Observations

- Training loss decreases consistently across epochs.
- Validation loss decreases initially but begins increasing after approximately epoch 5.

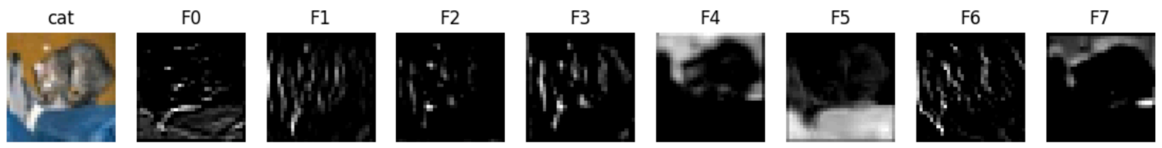
- This indicates overfitting, where the model continues improving on training data but begins to generalize less effectively on validation data

Loss Curves

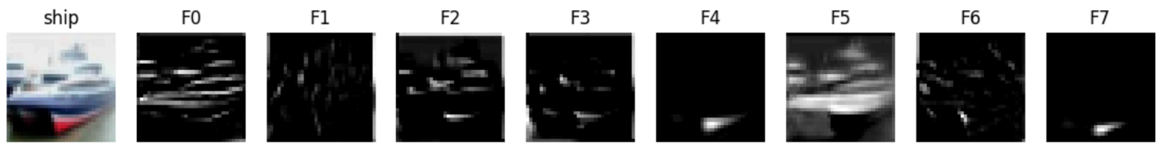


3. Feature Map Visualization (Early Layer):

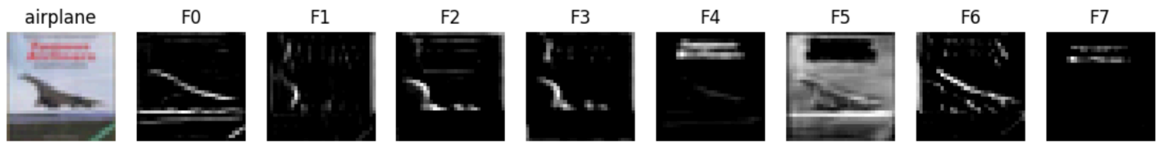
Feature Image 0



Feature Image 1



Feature Image 2



Observations

The first-layer filters detect low-level visual patterns such as:

- Edges (horizontal and vertical)
- Color contrasts
- Simple textures
- Object boundaries

Different feature maps respond differently to the same input image:

- Some highlight object contours.
- Some emphasize background regions.
- Some respond strongly to color variations.

This behavior is expected because early convolutional layers typically learn general low-level features rather than high level.

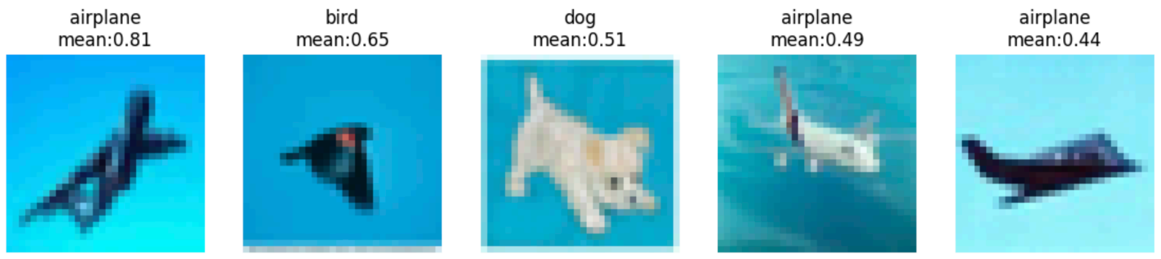
4. Maximally Activating Images:

Activation was defined as: The mean value of the 2D feature map after ReLU for a given filter.

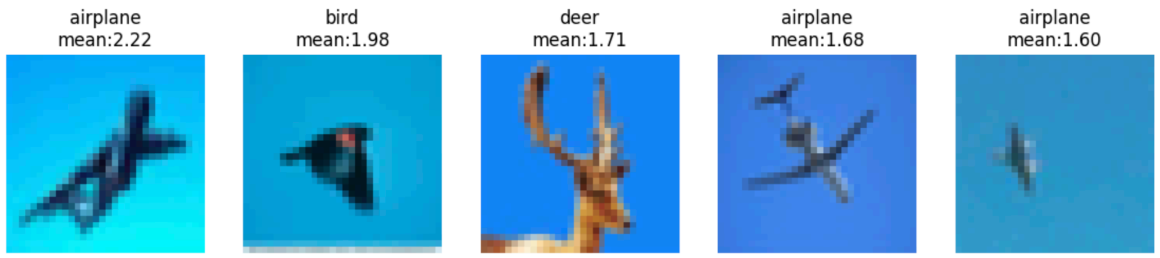
This reduces each feature map to a single scalar representing overall response strength.

Three filters from the first convolutional layer were selected:

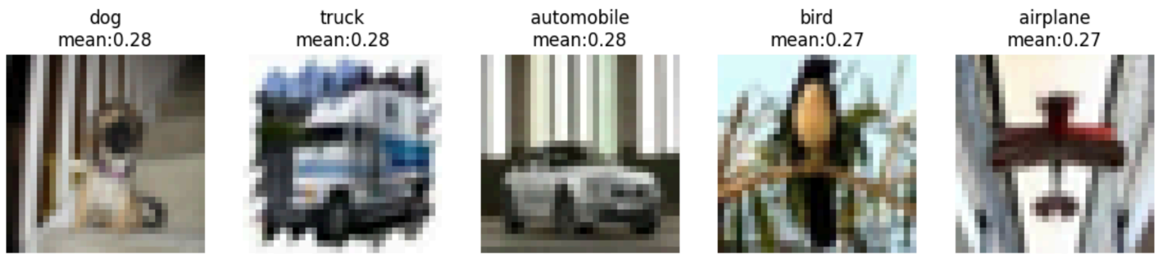
Filter 0



Filter 5:



Filter 10.



Observations

- Filters 0 and 5 produced very similar top 5 activating images.
- These images share common visual characteristics such as
    - Strong edges
    - High contrast regions
    - Similar background textures

This suggests that filters 0 and 5 detect similar low level visual features.

Filter 10 produced a different set of images indicating it responds to a different pattern possibly a different edge or color.

5. Brief Discussion and Reflection:

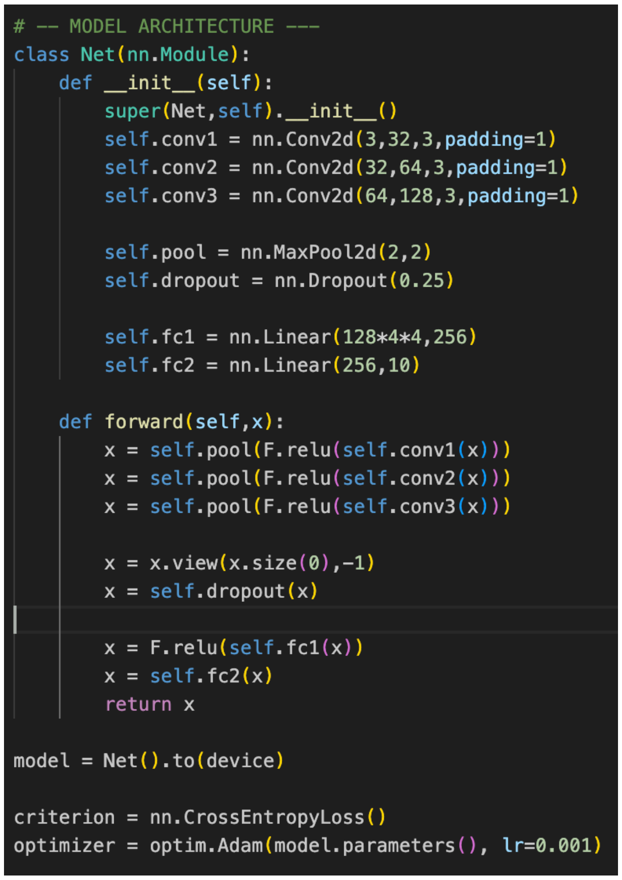
The CNN achieved a final test accuracy of 74.29% of CIFAR-10 images. The training and validation graphs show that after a few epochs the model starts to overfit. This means it keeps improving on the training data but it does not improve as much on new data. Stopping training earlier or using stronger regularization could help fix this.

The feature map images show that the first convolution layer learns simple patterns like edges, lines, colors, and textures. Different feature maps focus on different parts of the same image.

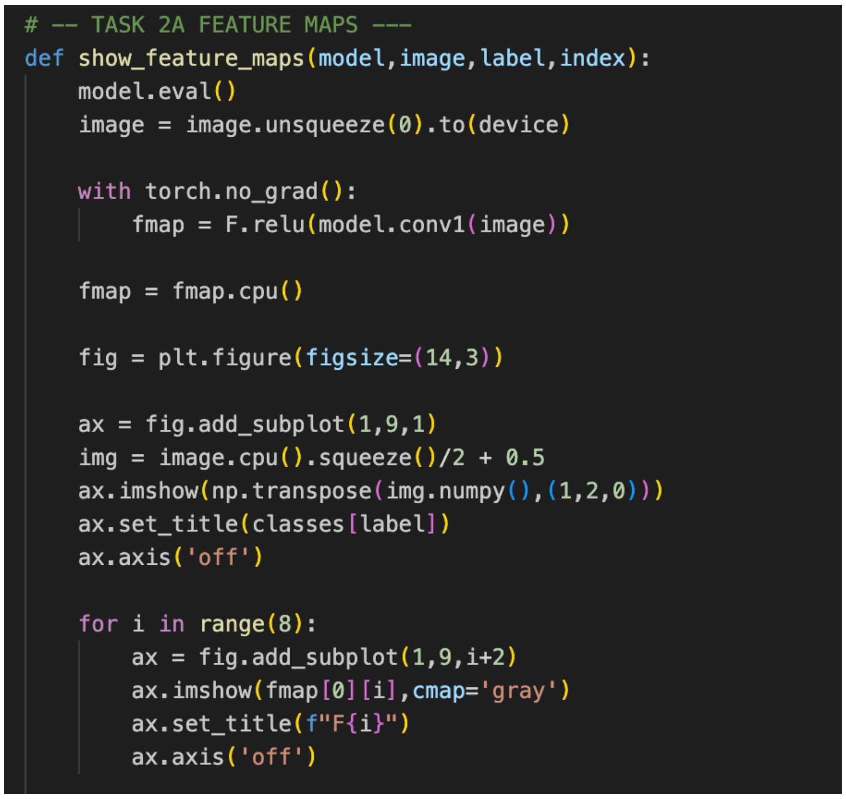
The top activating images show that the filters respond to general patterns, not specific objects. Early layers look for basic shapes and edges instead of recognizing full objects.

Overall, this model shows how a CNN learns step by step starting from simple patterns and building toward more complex understanding of images.

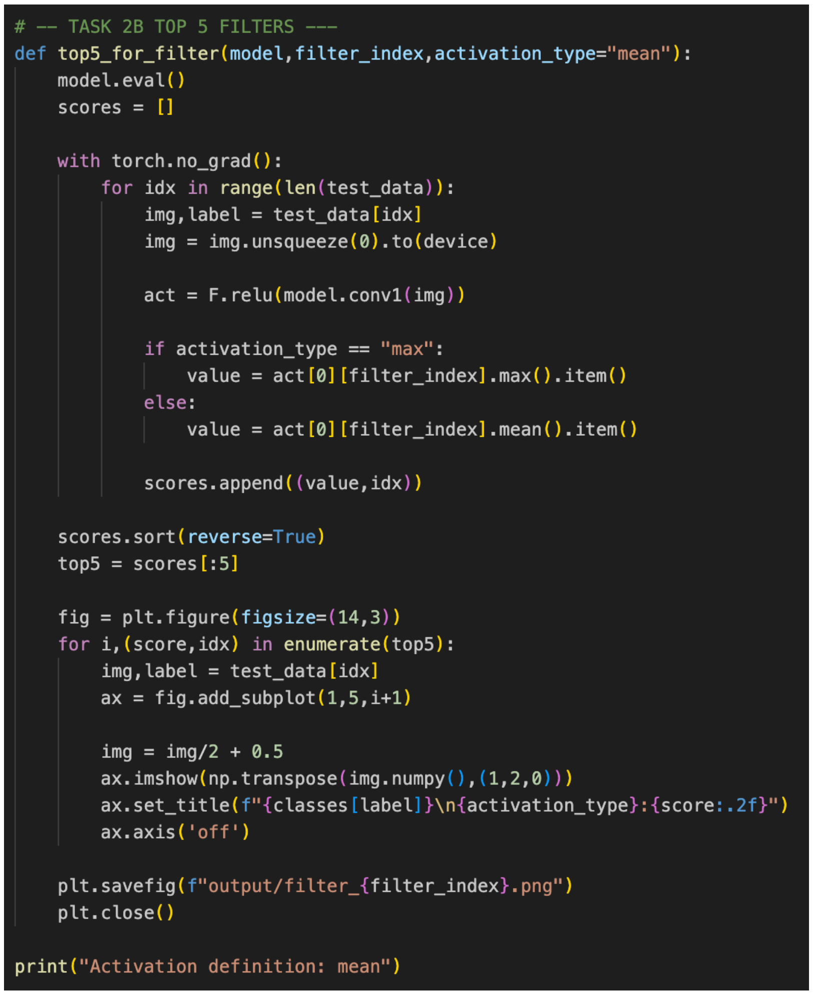
Source Code:



```python
# -- MODEL ARCHITECTURE ---
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.conv1 = nn.Conv2d(3,32,3,padding=1)
        self.conv2 = nn.Conv2d(32,64,3,padding=1)
        self.conv3 = nn.Conv2d(64,128,3,padding=1)

        self.pool = nn.MaxPool2d(2,2)
        self.dropout = nn.Dropout(0.25)

        self.fc1 = nn.Linear(128*4*4,256)
        self.fc2 = nn.Linear(256,10)

    def forward(self,x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        x = x.view(x.size(0),-1)
        x = self.dropout(x)

        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = Net().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

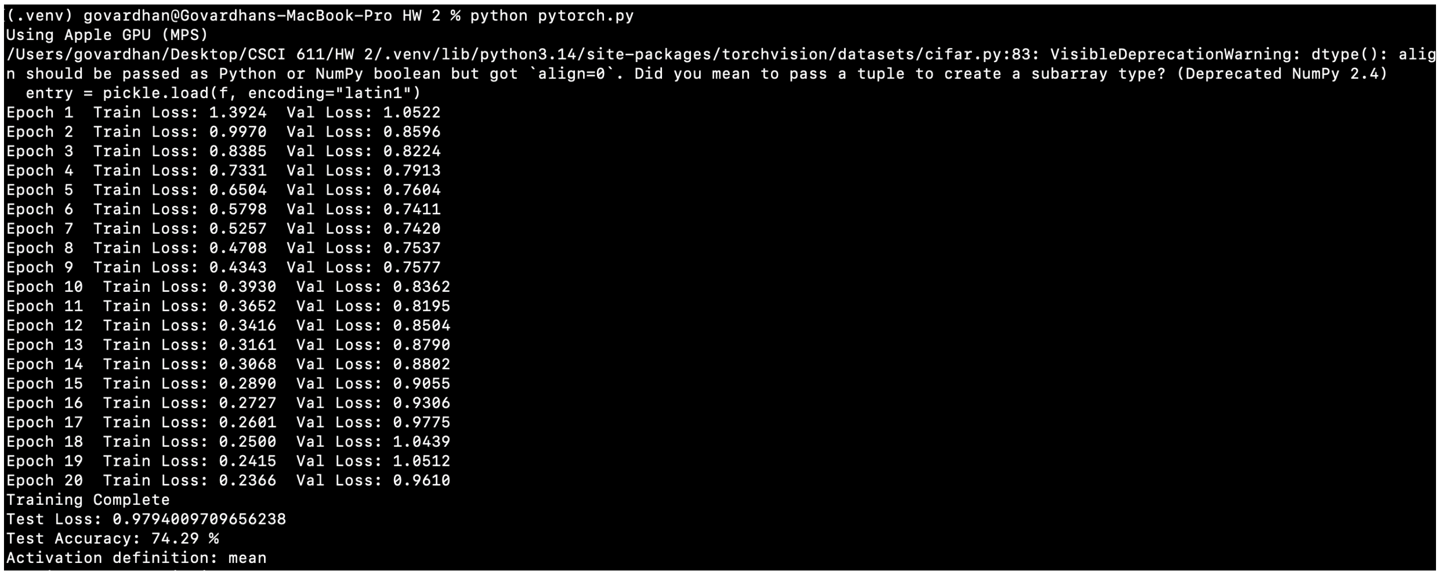


```python
# -- TASK 2A FEATURE MAPS ---
def show_feature_maps(model,image,label,index):
    model.eval()
    image = image.unsqueeze(0).to(device)

    with torch.no_grad():
        fmap = F.relu(model.conv1(image))

    fmap = fmap.cpu()

    fig = plt.figure(figsize=(14,3))

    ax = fig.add_subplot(1,9,1)
    img = image.cpu().squeeze()/2 + 0.5
    ax.imshow(np.transpose(img.numpy(),(1,2,0)))
    ax.set_title(classes[label])
    ax.axis('off')

    for i in range(8):
        ax = fig.add_subplot(1,9,i+2)
        ax.imshow(fmap[0][i],cmap='gray')
        ax.set_title(f"F{i}")
        ax.axis('off')
```



```python
# -- TASK 2B TOP 5 FILTERS ---
def top5_for_filter(model,filter_index,activation_type="mean"):
    model.eval()
    scores = []

    with torch.no_grad():
        for idx in range(len(test_data)):
            img,label = test_data[idx]
            img = img.unsqueeze(0).to(device)

            act = F.relu(model.conv1(img))

            if activation_type == "max":
                value = act[0][filter_index].max().item()
            else:
                value = act[0][filter_index].mean().item()

            scores.append((value,idx))

    scores.sort(reverse=True)
    top5 = scores[:5]

    fig = plt.figure(figsize=(14,3))
    for i,(score,idx) in enumerate(top5):
        img,label = test_data[idx]
        ax = fig.add_subplot(1,5,i+1)

        img = img/2 + 0.5
        ax.imshow(np.transpose(img.numpy(),(1,2,0)))
        ax.set_title(f"{classes[label]}\n{activation_type}:{score:.2f}")
        ax.axis('off')

    plt.savefig(f"output/filter_{filter_index}.png")
    plt.close()

print("Activation definition: mean")
```

Execution:



```
(.venv) govardhan@Govardhans-MacBook-Pro HW 2 % python pytorch.py
Using Apple GPU (MPS)
/Users/govardhan/Desktop/CSCI 611/HW 2/.venv/lib/python3.14/site-packages/torchvision/datasets/cifar.py:83: VisibleDeprecationWarning: dtype(): alig
n should be passed as Python or NumPy boolean but got `align=0`. Did you mean to pass a tuple to create a subarray type? (Deprecated NumPy 2.4)
  entry = pickle.load(f, encoding="latin1")
Epoch 1  Train Loss: 1.3924  Val Loss: 1.0522
Epoch 2  Train Loss: 0.9970  Val Loss: 0.8596
Epoch 3  Train Loss: 0.8385  Val Loss: 0.8224
Epoch 4  Train Loss: 0.7331  Val Loss: 0.7913
Epoch 5  Train Loss: 0.6504  Val Loss: 0.7604
Epoch 6  Train Loss: 0.5798  Val Loss: 0.7411
Epoch 7  Train Loss: 0.5257  Val Loss: 0.7420
Epoch 8  Train Loss: 0.4708  Val Loss: 0.7537
Epoch 9  Train Loss: 0.4343  Val Loss: 0.7577
Epoch 10  Train Loss: 0.3930  Val Loss: 0.8362
Epoch 11  Train Loss: 0.3652  Val Loss: 0.8195
Epoch 12  Train Loss: 0.3416  Val Loss: 0.8504
Epoch 13  Train Loss: 0.3161  Val Loss: 0.8790
Epoch 14  Train Loss: 0.3068  Val Loss: 0.8802
Epoch 15  Train Loss: 0.2890  Val Loss: 0.9055
Epoch 16  Train Loss: 0.2727  Val Loss: 0.9306
Epoch 17  Train Loss: 0.2601  Val Loss: 0.9775
Epoch 18  Train Loss: 0.2500  Val Loss: 1.0439
Epoch 19  Train Loss: 0.2415  Val Loss: 1.0512
Epoch 20  Train Loss: 0.2366  Val Loss: 0.9610
Training Complete
Test Loss: 0.9794009709656238
Test Accuracy: 74.29 %
Activation definition: mean
```