Deep Learning Course

Assignment 2

Assignment Goals

- Design and implementation of CNNs.
- CNN visualization.
- Implementation of ResNet.

In this assignment, you will be asked to learn CNN models for an image dataset. Different experiments will help you achieve a better understanding of CNNs.

Dataset

The dataset consists of around 9K images (some grayscale and some RGB) belonging to 101 classes. The shape of each image is (64,64,3). Every image is labeled with one of the classes. The image file is contained in the folder named after the class name.

Requirements

1. **(40 points) Implement and improve a CNN model.**

   (a) We are aiming to learn a CNN on the given dataset. Download the dataset, and use PyTorch to implement LeNet5 to classify instances. Use a one-hot encoding for labels. Split the dataset into training (90 percent) and validation (10 percent) and report the model loss (cross-entropy) and accuracy on both training and validation sets. (20 points)

   The LeNet5 configuration is:

   - Convolutional layer (kernel size 5 x 5, 32 filters, stride 1 x 1 and followed by ReLU)
   - Max Pooling layer with size 4 and stride 4 x 4
   - Convolutional layer (kernel size 5 x 5, 64 filters, stride 1 x 1 and followed by ReLU)
   - Max Pooling layer with size 4 and stride 4 x 4
   - Fully Connected ReLU layer that has 1021 neurons
   - Fully Connected ReLU layer with 84 neurons
   - Fully Connected Softmax layer that has input 84 and output which is equal to the number of classes (one node for each of the classes).

   (b) Try to improve model accuracy on the validation dataset by tuning the model hyperparameters. You can use any improvement methods you prefer. You are expected to reach at least 65 percent accuracy on validation set. (20 points)

   Here are some improvement methods you can use, of course you can use others which are not mentioned here:

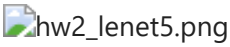
- Dropout
- L1, L2 regularization
- Try improved initialization (e.g., Xavier initialier)
- Batch Normalization

The grading of part (b) is based on the correctness of your implementation (5 points) and the performance of your improvement on the validation set. The validation accuracy and corresponding score is:

- 65% (5 points)
- 67% (8 points)
- 69% (12 points)
- 71% (15 points)

Structure of LENET-5

This following LENET-5 structure is for 10-class dataset. Therefore, the layer size is not exactly the same as ours.



1. **(20 points) Visualize layer activation**

   There are several approaches to understand and visualize convolutional Networks, including visualizing the activations and layers weights. The most straight-forward visualization technique is to show the activations of the network during the forward pass. The second most common strategy is to visualize the weights. For more information we recommend the course notes on "Visualizing what ConvNets learn". More advanced techniques can be found in "Visualizing and Understanding Convolutional Networks" paper by Matthew D.Zeiler and Rob Fergus.

   Please visualize the layer activation of **the first conv layer** and **the second conv layer** of your above CNN model (after completing Q1), on the following 2 images:

   - accordion/image_0001
   - camera/image_0001

   Visualizing a CNN layer activation means to visualize the result of the activation layer as an image. Specifically, the activation of the first conv layer is the output of the first (conv + ReLU) layer during forward propagation. Since we have 32 filters in the first conv layer, you should draw 32 activation images for the first conv layer. Please display multiple images side by side in a row to make your output more readable (Hint: matplotlib.pyplot.subplot).

1. **(40 points) ResNet Implementation**

   Use PyTorch to implement ResNet 18 to classify the given dataset. Same as above, please use a one-hot encoding for labels, split the dataset into training (90 percent) and validation (10 percent) and report the model loss (cross-entropy) and accuracy on both

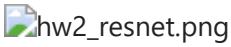
training and validation sets. See the paper Deep Residual Learning for Image Recognition for detailed introduction of ResNet.

The grading of this part is mainly based on the implementation and performance on validation set. If you need more resources to complete the training, consider using Google Colab.

The ResNet 18 configuration is:

- conv_1 (kernel size 7 x 7, 64 filters, stride 2 x 2)
- conv_2 (max pooling layer with size 3 x 3, followed by 2 blocks.Each block contains two conv layers. Each conv layer has kernel size 3 x 3, 64 filters, stride 2 x 2)
- conv_3 (2 blocks, each contains 2 conv layers with kernel size 3*3, 128 filters)
- conv_4 (2 blocks, each contains 2 conv layers with kernel size 3*3, 256 filters)
- conv_5 (2 blocks, each contains 2 conv layers with kernel size 3*3, 512 filters)

A block has the structure:



Submission Notes

Please use Jupyter Notebook. The notebook should include the final code, results and your answers. You should submit your Notebook in (.pdf or .html) and .ipynb format. (penalty 10 points)

Your Implementation

In [1]:
```
# You can use the following helper functions

from typing import Any
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import os
from torchvision.io import read_image
from torchvision import transforms
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import torch
from torch import nn
from tqdm import tqdm
from matplotlib import pyplot as plt
import torch.optim as optim
import torch.nn.functional as F
from torch.optim.lr_scheduler import MultiStepLR as MultiStepLR
from PIL import Image
import torchvision.transforms.functional as TF
```

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

In [3]:
```
class ImageDataset( Dataset ):

    def __init__(self, is_val= False, transform = None) -> None:
```

```
        if is_val:
            self.df = pd.read_csv( 'validation.csv', index_col=0 )
        else:
            self.df = pd.read_csv( 'train.csv', index_col= 0 )

        self.cls_names = self.df['cls_name'].unique().tolist()
        self.df['label'] = self.df['cls_name'].apply( self.cls_names.index )

        self.transform = transform


    def get_num_classes(self):
        return len( self.cls_names )

    def __len__(self):
        return len( self.df )

    def __getitem__(self, index):
        path = self.df.iloc[index]['path']

        path = os.path.join(os.getcwd(), path) # to be commented
        # # print(path)

        img = read_image( path ).type( torch.float32 )

        target = self.df.iloc[index]['label']

        if self.transform is not None:
            img = self.transform( img )

        target = torch.tensor( target )


        return img/255 , target

def collate_fn( batch ):
    imgs, targets = [], []

    for img, target in batch:
        imgs.append( img )
        targets.append( target )

    imgs = torch.stack( imgs, dim= 0 )
    targets = torch.stack( targets, dim= 0 )
    return imgs, targets
```

In [11]:

```
num_epochs = 75
batch_size = 64
learning_rate = 0.001
weight_decay = 0.01
momentum = 0.9
TOTAL_CLASS = 101
```

In [5]:

```
transform = transforms.Compose([
    transforms.RandomVerticalFlip( .5 ),
    transforms.Normalize( (0.485, 0.456, 0.406), (0.229, 0.224, 0.225) ),
])

train_dataset = ImageDataset( is_val = False, transform = transform )
val_dataset = ImageDataset( is_val = True )

train_dataloader = DataLoader( train_dataset, batch_size = batch_size, shuffle= Tru
val_dataloader = DataLoader( val_dataset, batch_size = batch_size, shuffle= True, c
```

In [6]:
```
def init_weights( m ):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            m.bias.data.fill_(0.01)
```

Implement and improve a CNN model

In [7]:
```
# implement your Lenet5 here
class Lenet(nn.Module):
  def __init__(self):
    super(Lenet,self).__init__()
    self.conv1 = nn.Conv2d(3,32,5)
    self.conv2 = nn.Conv2d(32,64,5)

    self.maxpool = nn.MaxPool2d(4,stride=4)
    self.relu = nn.ReLU(inplace=True)

    self.linear = nn.Sequential(
        nn.Flatten(),
        nn.Linear(2*2*64, 1021),
        nn.ReLU(inplace=True),

        nn.Linear(1021,84),
        nn.ReLU(inplace=True),
        nn.Linear(84,TOTAL_CLASS),
    )
  def forward(self, x):
    x = self.conv1(x)
    # 64 -> 60
    x = self.relu(x)
    x = self.maxpool(x)
    # 60 -> 15
    x = self.conv2(x)
    # 15 -> 11
    x = self.relu(x)
    x = self.maxpool(x)
    # 11 -> 2

    x = self.linear(x)
    return x
model = Lenet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_de
```

In [8]:
```
#train
for epoch in range(num_epochs):
    model.train()  # Set the model to training mode
    running_loss = 0.0
    for inputs, labels in train_dataloader:
        # print(labels)
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
```

```
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    # Calculate average training loss for the epoch
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in train_dataloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs

        print('Accuracy of the network on the training images: {} %'.format(100 * c
    epoch_loss = running_loss / len(train_dataset)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}")


    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in val_dataloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs

        print('Accuracy of the network on the validation images: {} %'.format(100 *
```

```
Accuracy of the network on the training images: 23.268091681689416 %
Epoch [1/30], Loss: 0.0636
Accuracy of the network on the validation images: 19.319429198682766 %
Accuracy of the network on the training images: 29.693535925830542 %
Epoch [2/30], Loss: 0.0549
Accuracy of the network on the validation images: 27.003293084522504 %
Accuracy of the network on the training images: 32.34612413082668 %
Epoch [3/30], Loss: 0.0508
Accuracy of the network on the validation images: 30.076838638858398 %
Accuracy of the network on the training images: 34.12310069533866 %
Epoch [4/30], Loss: 0.0479
Accuracy of the network on the validation images: 30.515916575192097 %
Accuracy of the network on the training images: 37.67705382436261 %
Epoch [5/30], Loss: 0.0461
Accuracy of the network on the validation images: 33.80900109769484 %
Accuracy of the network on the training images: 36.33788308009271 %
Epoch [6/30], Loss: 0.0441
Accuracy of the network on the validation images: 33.80900109769484 %
Accuracy of the network on the training images: 40.36827195467422 %
Epoch [7/30], Loss: 0.0430
Accuracy of the network on the validation images: 38.52908891328211 %
Accuracy of the network on the training images: 42.1967550862735 %
Epoch [8/30], Loss: 0.0410
Accuracy of the network on the validation images: 36.333699231613615 %
Accuracy of the network on the training images: 41.334020087561164 %
Epoch [9/30], Loss: 0.0403
Accuracy of the network on the validation images: 38.52908891328211 %
Accuracy of the network on the training images: 42.518671130569146 %
Epoch [10/30], Loss: 0.0389
Accuracy of the network on the validation images: 35.12623490669594 %
Accuracy of the network on the training images: 43.56167911408705 %
Epoch [11/30], Loss: 0.0384
Accuracy of the network on the validation images: 40.834248079034026 %
Accuracy of the network on the training images: 45.19701261910894 %
Epoch [12/30], Loss: 0.0373
Accuracy of the network on the validation images: 38.199780461031835 %
Accuracy of the network on the training images: 44.14112799381921 %
Epoch [13/30], Loss: 0.0367
Accuracy of the network on the validation images: 38.090010976948406 %
Accuracy of the network on the training images: 47.321658511460214 %
Epoch [14/30], Loss: 0.0360
Accuracy of the network on the validation images: 39.62678375411635 %
Accuracy of the network on the training images: 49.27890806077775 %
Epoch [15/30], Loss: 0.0353
Accuracy of the network on the validation images: 41.1635565312843 %
Accuracy of the network on the training images: 49.07288179242853 %
Epoch [16/30], Loss: 0.0348
Accuracy of the network on the validation images: 38.63885839736553 %
Accuracy of the network on the training images: 49.56219417975792 %
Epoch [17/30], Loss: 0.0341
Accuracy of the network on the validation images: 41.1635565312843 %
Accuracy of the network on the training images: 50.296162760752 %
Epoch [18/30], Loss: 0.0337
Accuracy of the network on the validation images: 40.61470911086718 %
Accuracy of the network on the training images: 51.648210146793716 %
Epoch [19/30], Loss: 0.0331
Accuracy of the network on the validation images: 41.27332601536773 %
Accuracy of the network on the training images: 51.30054081895442 %
Epoch [20/30], Loss: 0.0327
Accuracy of the network on the validation images: 40.834248079034026 %
Accuracy of the network on the training images: 49.24027813546227 %
Epoch [21/30], Loss: 0.0328
Accuracy of the network on the validation images: 41.93194291986828 %
Accuracy of the network on the training images: 50.489312387329385 %
```

```
Epoch [22/30], Loss: 0.0319
Accuracy of the network on the validation images: 38.748627881448954 %
Accuracy of the network on the training images: 53.19340715941283 %
Epoch [23/30], Loss: 0.0318
Accuracy of the network on the validation images: 40.28540065861691 %
Accuracy of the network on the training images: 50.965748132886944 %
Epoch [24/30], Loss: 0.0314
Accuracy of the network on the validation images: 40.39517014270033 %
Accuracy of the network on the training images: 52.98738089106361 %
Epoch [25/30], Loss: 0.0314
Accuracy of the network on the validation images: 42.70032930845225 %
Accuracy of the network on the training images: 53.180530517640996 %
Epoch [26/30], Loss: 0.0307
Accuracy of the network on the validation images: 44.6761800219539 %
Accuracy of the network on the training images: 56.54133402008756 %
Epoch [27/30], Loss: 0.0305
Accuracy of the network on the validation images: 46.212952799121844 %
Accuracy of the network on the training images: 54.42956476950811 %
Epoch [28/30], Loss: 0.0299
Accuracy of the network on the validation images: 45.334796926454445 %
Accuracy of the network on the training images: 52.31779551892866 %
Epoch [29/30], Loss: 0.0298
Accuracy of the network on the validation images: 41.38309549945115 %
Accuracy of the network on the training images: 54.82874066443472 %
Epoch [30/30], Loss: 0.0297
Accuracy of the network on the validation images: 44.2371020856202 %
```

Methods I used to improve Lenet:

1. I added padding to every conv layer to make the dimension consistent before and after conv layer.
2. I added BatchNorm to every conv layer
3. I change the fully connected layer to (2*2*64 -> 256 -> 128 -> TOTAL_CLASS)
4. I added three conv layer and each layer is followed by a batchNorm and ReLU
5. I added Xavier initializer to both conv layers and linear layers
6. I added Normalization to data transform
7. I added a scheduler to decrease the learning rate while training
8. I change the maxpool to size=2 and stride=2 and only use pooling layer to decrease dimension.

In [10]:
```python
# implement your Lenet5 here
class improvedLenet(nn.Module):
    def __init__(self):
        super(improvedLenet,self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3,32,5,padding='same'),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),

        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32,64,5,padding='same'),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),

        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64,128,3,padding =1),
```

```
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(128,256,3,padding =1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),

        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(256,128,3,padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),

        )
        # self.conv5 = nn.Sequential(
        #     nn.Conv2d(256,512,3,padding =1),
        #     nn.ReLU(inplace=True),
        #     nn.BatchNorm2d(512),
        # )
        # self.conv6 = nn.Sequential(
        #     nn.Conv2d(512,256,3,padding=1),
        #     nn.ReLU(inplace=True),
        #     nn.BatchNorm2d(256)
        # )
        # self.conv6 = nn.Sequential(
        #     nn.Conv2d(256,256,3,padding =1),
        #     nn.ReLU(inplace=True),
        #     nn.BatchNorm2d(256),
        # )
        # self.conv7 = nn.Sequential(
        #     nn.Conv2d(256,512,3,padding =1),
        #     nn.ReLU(inplace=True),
        #     nn.BatchNorm2d(512)
        # )
        # self.conv8 = nn.Sequential(
        #     nn.Conv2d(512,512,3,padding=1),
        #     nn.ReLU(inplace=True),
        #     nn.BatchNorm2d(512)
        # )
        self.maxpool = nn.MaxPool2d(2,stride=2)
        self.linear = nn.Sequential(
            nn.Flatten(),
            nn.Linear(2*2*128, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256,128),
            nn.BatchNorm1d(128),
            nn.ReLU(inplace=True),
            nn.Linear(128,TOTAL_CLASS)
        )
    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = self.maxpool(x)
        x = self.conv3(x)
        x = self.maxpool(x)
        x = self.conv4(x)
        x = self.maxpool(x)
        x = self.conv5(x)
        x = self.maxpool(x)
        # x = self.conv6(x)
        x = self.linear(x)
        return x
```

```
model = improvedLenet().to(device)
model.apply(init_weights)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_de
scheduler = MultiStepLR(optimizer, [30,50], gamma=0.001)
```

In [41]:

```
#train
for epoch in range(num_epochs):
    model.train()  # Set the model to training mode
    running_loss = 0.0
    for inputs, labels in train_dataloader:
        # print(labels)
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    # Calculate average training loss for the epoch
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in train_dataloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs

        print('Accuracy of the network on the training images: {} %'.format(100 * c
    epoch_loss = running_loss / len(train_dataset)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}")


    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in val_dataloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs

        print('Accuracy of the network on the validation images: {} %'.format(100 *
```

```
Accuracy of the network on the training images: 40.47128508884883 %
Epoch [1/75], Loss: 0.0504
Accuracy of the network on the validation images: 38.748627881448954 %
Accuracy of the network on the training images: 48.01699716713881 %
Epoch [2/75], Loss: 0.0405
Accuracy of the network on the validation images: 43.578485181119646 %
Accuracy of the network on the training images: 52.56245171259336 %
Epoch [3/75], Loss: 0.0358
Accuracy of the network on the validation images: 48.957189901207464 %
Accuracy of the network on the training images: 56.631470512490345 %
Epoch [4/75], Loss: 0.0328
Accuracy of the network on the validation images: 52.25027442371021 %
Accuracy of the network on the training images: 59.41282513520474 %
Epoch [5/75], Loss: 0.0306
Accuracy of the network on the validation images: 53.567508232711305 %
Accuracy of the network on the training images: 62.12979654906 %
Epoch [6/75], Loss: 0.0286
Accuracy of the network on the validation images: 55.3238199780461 %
Accuracy of the network on the training images: 62.59335565284574 %
Epoch [7/75], Loss: 0.0274
Accuracy of the network on the validation images: 55.7628979143798 %
Accuracy of the network on the training images: 64.64074169456606 %
Epoch [8/75], Loss: 0.0264
Accuracy of the network on the validation images: 57.0801317233809 %
Accuracy of the network on the training images: 66.36621169199073 %
Epoch [9/75], Loss: 0.0253
Accuracy of the network on the validation images: 57.5192096597146 %
Accuracy of the network on the training images: 68.34921452485192 %
Epoch [10/75], Loss: 0.0244
Accuracy of the network on the validation images: 60.37321624588365 %
Accuracy of the network on the training images: 67.75688900334792 %
Epoch [11/75], Loss: 0.0236
Accuracy of the network on the validation images: 59.055982436882545 %
Accuracy of the network on the training images: 71.99330414627865 %
Epoch [12/75], Loss: 0.0228
Accuracy of the network on the validation images: 60.37321624588365 %
Accuracy of the network on the training images: 72.97192892093742 %
Epoch [13/75], Loss: 0.0218
Accuracy of the network on the validation images: 63.44676180021954 %
Accuracy of the network on the training images: 72.79165593613186 %
Epoch [14/75], Loss: 0.0214
Accuracy of the network on the validation images: 60.81229418221734 %
Accuracy of the network on the training images: 73.74452742724698 %
Epoch [15/75], Loss: 0.0202
Accuracy of the network on the validation images: 61.47091108671789 %
Accuracy of the network on the training images: 75.0450682462014 %
Epoch [16/75], Loss: 0.0198
Accuracy of the network on the validation images: 62.019758507135016 %
Accuracy of the network on the training images: 76.55163533350502 %
Epoch [17/75], Loss: 0.0190
Accuracy of the network on the validation images: 63.22722283205269 %
Accuracy of the network on the training images: 78.01957249549318 %
Epoch [18/75], Loss: 0.0184
Accuracy of the network on the validation images: 63.88583973655324 %
Accuracy of the network on the training images: 78.14833891321143 %
Epoch [19/75], Loss: 0.0176
Accuracy of the network on the validation images: 64.32491767288694 %
Accuracy of the network on the training images: 79.28148338913212 %
Epoch [20/75], Loss: 0.0170
Accuracy of the network on the validation images: 64.54445664105378 %
Accuracy of the network on the training images: 80.26010816379089 %
Epoch [21/75], Loss: 0.0164
Accuracy of the network on the validation images: 64.43468715697036 %
Accuracy of the network on the training images: 81.47051249034251 %
```

```
Epoch [22/75], Loss: 0.0159
Accuracy of the network on the validation images: 65.09330406147092 %
Accuracy of the network on the training images: 80.4661344321401 %
Epoch [23/75], Loss: 0.0154
Accuracy of the network on the validation images: 63.44676180021954 %
Accuracy of the network on the training images: 82.87406644347155 %
Epoch [24/75], Loss: 0.0150
Accuracy of the network on the validation images: 64.10537870472008 %
Accuracy of the network on the training images: 83.71104815864022 %
Epoch [25/75], Loss: 0.0143
Accuracy of the network on the validation images: 65.64215148188804 %
Accuracy of the network on the training images: 83.77543136749935 %
Epoch [26/75], Loss: 0.0141
Accuracy of the network on the validation images: 66.95938529088913 %
Accuracy of the network on the training images: 85.41076487252124 %
Epoch [27/75], Loss: 0.0135
Accuracy of the network on the validation images: 65.20307354555433 %
Accuracy of the network on the training images: 84.83131599278909 %
Epoch [28/75], Loss: 0.0134
Accuracy of the network on the validation images: 66.41053787047201 %
Accuracy of the network on the training images: 86.35075972186453 %
Epoch [29/75], Loss: 0.0129
Accuracy of the network on the validation images: 64.76399560922064 %
Accuracy of the network on the training images: 86.32500643832088 %
Epoch [30/75], Loss: 0.0124
Accuracy of the network on the validation images: 65.64215148188804 %
Accuracy of the network on the training images: 87.18774143703322 %
Epoch [31/75], Loss: 0.0122
Accuracy of the network on the validation images: 68.05708013172338 %
Accuracy of the network on the training images: 86.17048673705898 %
Epoch [32/75], Loss: 0.0120
Accuracy of the network on the validation images: 66.63007683863886 %
Accuracy of the network on the training images: 87.30363121297965 %
Epoch [33/75], Loss: 0.0121
Accuracy of the network on the validation images: 66.95938529088913 %
Accuracy of the network on the training images: 88.0247231522019 %
Epoch [34/75], Loss: 0.0117
Accuracy of the network on the validation images: 66.95938529088913 %
Accuracy of the network on the training images: 88.12773628637652 %
Epoch [35/75], Loss: 0.0113
Accuracy of the network on the validation images: 67.17892425905599 %
Accuracy of the network on the training images: 88.87458150914242 %
Epoch [36/75], Loss: 0.0108
Accuracy of the network on the validation images: 68.27661909989023 %
Accuracy of the network on the training images: 89.14499098635076 %
Epoch [37/75], Loss: 0.0110
Accuracy of the network on the validation images: 67.06915477497256 %
Accuracy of the network on the training images: 89.56992016482101 %
Epoch [38/75], Loss: 0.0109
Accuracy of the network on the validation images: 66.8496158068057 %
Accuracy of the network on the training images: 89.64718001545197 %
Epoch [39/75], Loss: 0.0109
Accuracy of the network on the validation images: 66.30076838638858 %
Accuracy of the network on the training images: 89.18362091166624 %
Epoch [40/75], Loss: 0.0103
Accuracy of the network on the validation images: 67.83754116355654 %
Accuracy of the network on the training images: 90.17512232809683 %
Epoch [41/75], Loss: 0.0103
Accuracy of the network on the validation images: 66.30076838638858 %
Accuracy of the network on the training images: 91.08936389389648 %
Epoch [42/75], Loss: 0.0100
Accuracy of the network on the validation images: 69.70362239297475 %
Accuracy of the network on the training images: 91.51429307236673 %
Epoch [43/75], Loss: 0.0098
```

```
Accuracy of the network on the validation images: 67.94731064763995 %
Accuracy of the network on the training images: 91.64305949008498 %
Epoch [44/75], Loss: 0.0098
Accuracy of the network on the validation images: 66.8496158068057 %
Accuracy of the network on the training images: 90.78032449137265 %
Epoch [45/75], Loss: 0.0095
Accuracy of the network on the validation images: 66.30076838638858 %
Accuracy of the network on the training images: 91.23100695338655 %
Epoch [46/75], Loss: 0.0097
Accuracy of the network on the validation images: 68.93523600439077 %
Accuracy of the network on the training images: 91.06361061035282 %
Epoch [47/75], Loss: 0.0099
Accuracy of the network on the validation images: 66.8496158068057 %
Accuracy of the network on the training images: 91.78470254957507 %
Epoch [48/75], Loss: 0.0093
Accuracy of the network on the validation images: 67.83754116355654 %
Accuracy of the network on the training images: 92.69894411537472 %
Epoch [49/75], Loss: 0.0091
Accuracy of the network on the validation images: 67.72777167947311 %
Accuracy of the network on the training images: 90.81895441668813 %
Epoch [50/75], Loss: 0.0092
Accuracy of the network on the validation images: 65.97145993413831 %
Accuracy of the network on the training images: 91.66881277362864 %
Epoch [51/75], Loss: 0.0089
Accuracy of the network on the validation images: 65.64215148188804 %
Accuracy of the network on the training images: 91.2567602369302 %
Epoch [52/75], Loss: 0.0095
Accuracy of the network on the validation images: 66.52030735455543 %
Accuracy of the network on the training images: 92.32552150399177 %
Epoch [53/75], Loss: 0.0088
Accuracy of the network on the validation images: 68.38638858397366 %
Accuracy of the network on the training images: 92.32552150399177 %
Epoch [54/75], Loss: 0.0088
Accuracy of the network on the validation images: 69.37431394072448 %
Accuracy of the network on the training images: 91.91346896729333 %
Epoch [55/75], Loss: 0.0090
Accuracy of the network on the validation images: 65.97145993413831 %
Accuracy of the network on the training images: 93.48441926345609 %
Epoch [56/75], Loss: 0.0087
Accuracy of the network on the validation images: 70.25246981339188 %
Accuracy of the network on the training images: 92.85346381663662 %
Epoch [57/75], Loss: 0.0087
Accuracy of the network on the validation images: 69.81339187705818 %
Accuracy of the network on the training images: 92.57017769765645 %
Epoch [58/75], Loss: 0.0087
Accuracy of the network on the validation images: 67.94731064763995 %
Accuracy of the network on the training images: 92.9436003090394 %
Epoch [59/75], Loss: 0.0086
Accuracy of the network on the validation images: 68.82546652030736 %
Accuracy of the network on the training images: 93.11099665207314 %
Epoch [60/75], Loss: 0.0085
Accuracy of the network on the validation images: 69.37431394072448 %
Accuracy of the network on the training images: 93.07236672675766 %
Epoch [61/75], Loss: 0.0086
Accuracy of the network on the validation images: 67.94731064763995 %
Accuracy of the network on the training images: 92.26113829513262 %
Epoch [62/75], Loss: 0.0084
Accuracy of the network on the validation images: 67.06915477497256 %
Accuracy of the network on the training images: 93.57455575585887 %
Epoch [63/75], Loss: 0.0084
Accuracy of the network on the validation images: 66.63007683863886 %
Accuracy of the network on the training images: 93.99948493432913 %
Epoch [64/75], Loss: 0.0085
Accuracy of the network on the validation images: 67.50823271130626 %
```

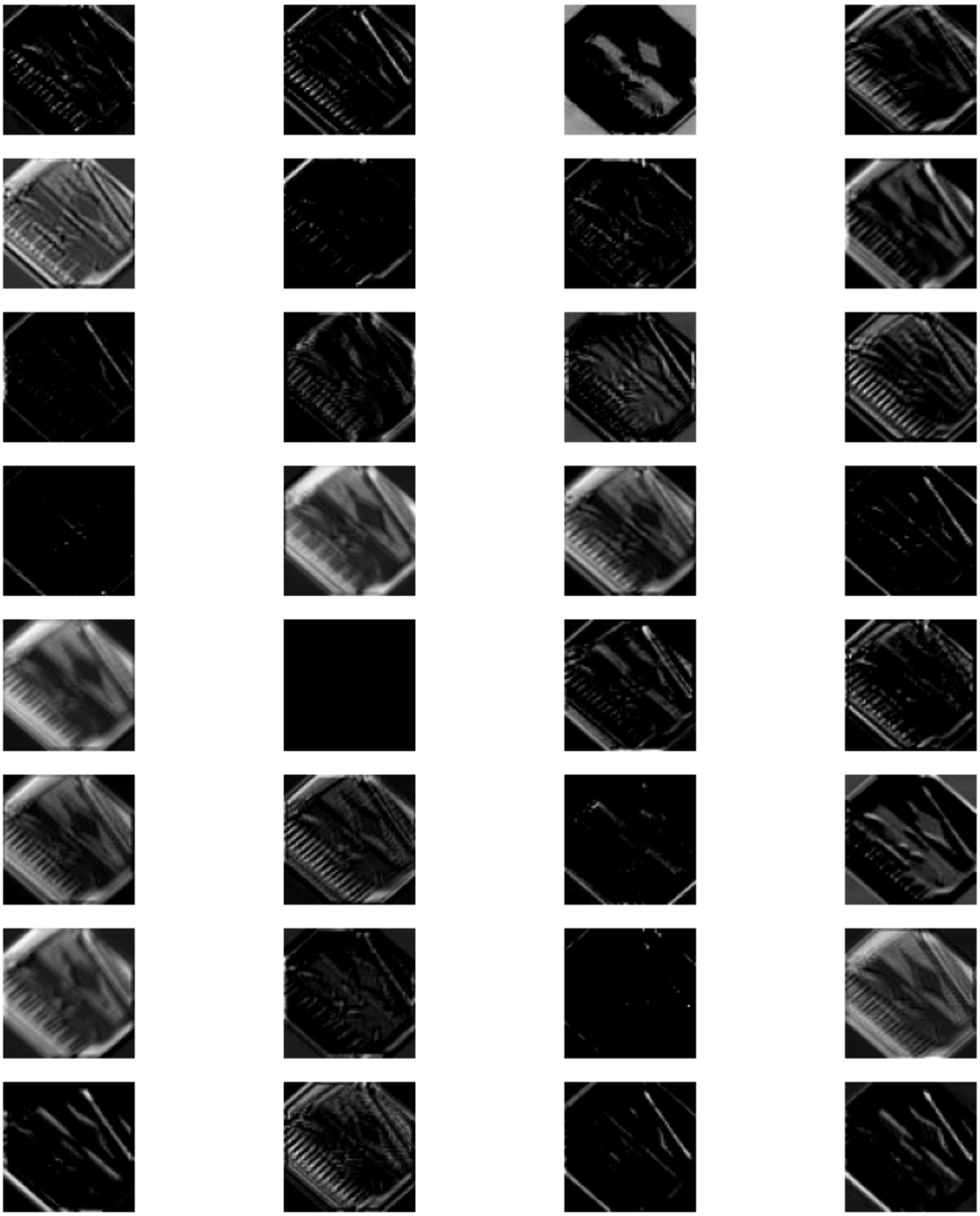
```
Accuracy of the network on the training images: 94.55318053051764 %
Epoch [65/75], Loss: 0.0083
Accuracy of the network on the validation images: 69.37431394072448 %
Accuracy of the network on the training images: 94.24414112799381 %
Epoch [66/75], Loss: 0.0080
Accuracy of the network on the validation images: 68.71569703622393 %
Accuracy of the network on the training images: 93.65181560648983 %
Epoch [67/75], Loss: 0.0083
Accuracy of the network on the validation images: 68.05708013172338 %
Accuracy of the network on the training images: 93.96085500901366 %
Epoch [68/75], Loss: 0.0084
Accuracy of the network on the validation images: 68.82546652030736 %
Accuracy of the network on the training images: 94.08962142673191 %
Epoch [69/75], Loss: 0.0077
Accuracy of the network on the validation images: 67.17892425905599 %
Accuracy of the network on the training images: 94.45016739634303 %
Epoch [70/75], Loss: 0.0076
Accuracy of the network on the validation images: 68.71569703622393 %
Accuracy of the network on the training images: 94.43729075457121 %
Epoch [71/75], Loss: 0.0079
Accuracy of the network on the validation images: 67.72777167947311 %
Accuracy of the network on the training images: 92.27401493690445 %
Epoch [72/75], Loss: 0.0083
Accuracy of the network on the validation images: 68.71569703622393 %
Accuracy of the network on the training images: 93.70332217357713 %
Epoch [73/75], Loss: 0.0084
Accuracy of the network on the validation images: 68.16684961580681 %
Accuracy of the network on the training images: 94.11537471027556 %
Epoch [74/75], Loss: 0.0080
Accuracy of the network on the validation images: 68.05708013172338 %
Accuracy of the network on the training images: 94.06386814318826 %
Epoch [75/75], Loss: 0.0078
Accuracy of the network on the validation images: 69.0450054884742 %
```

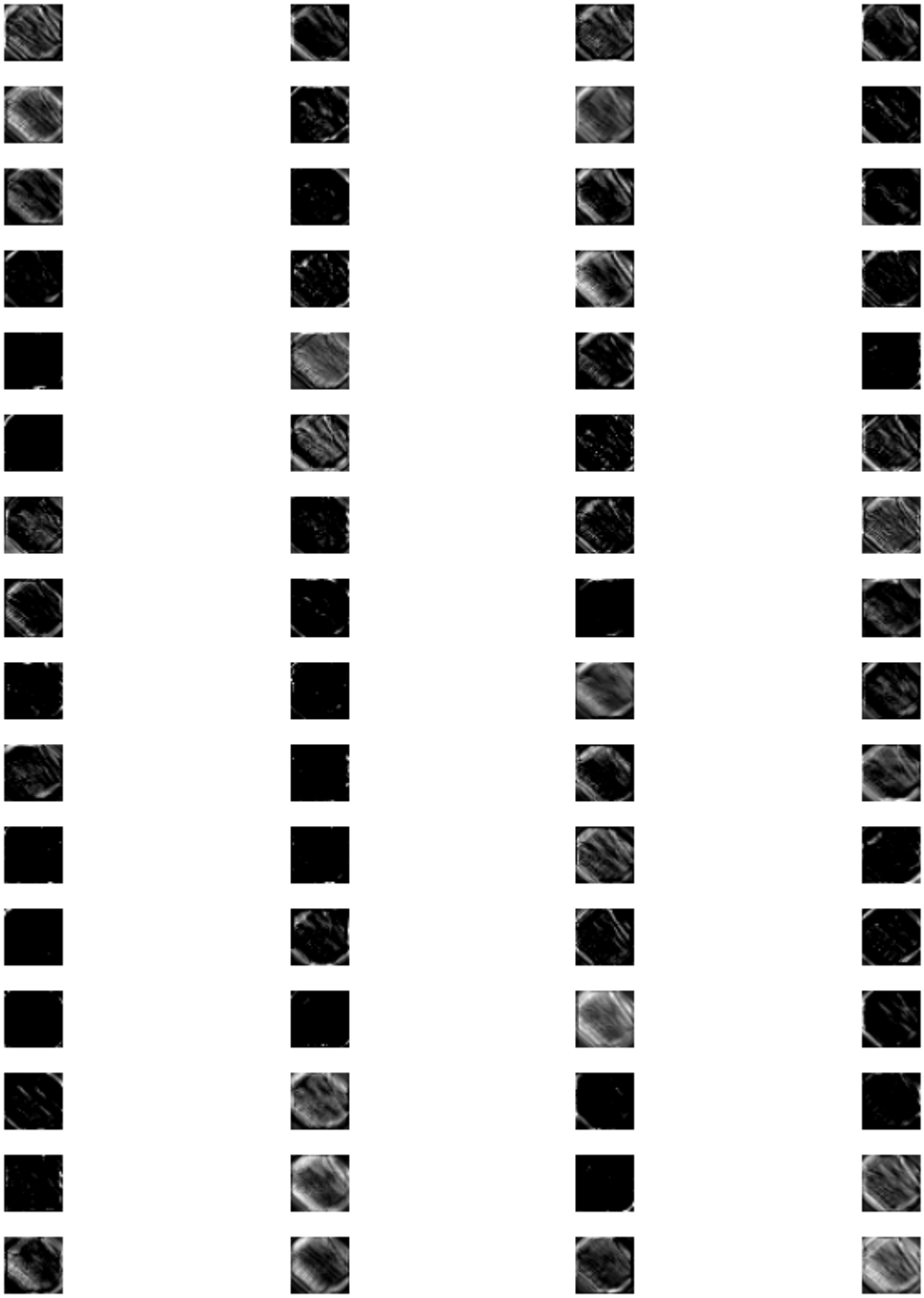
Visualize layer activation

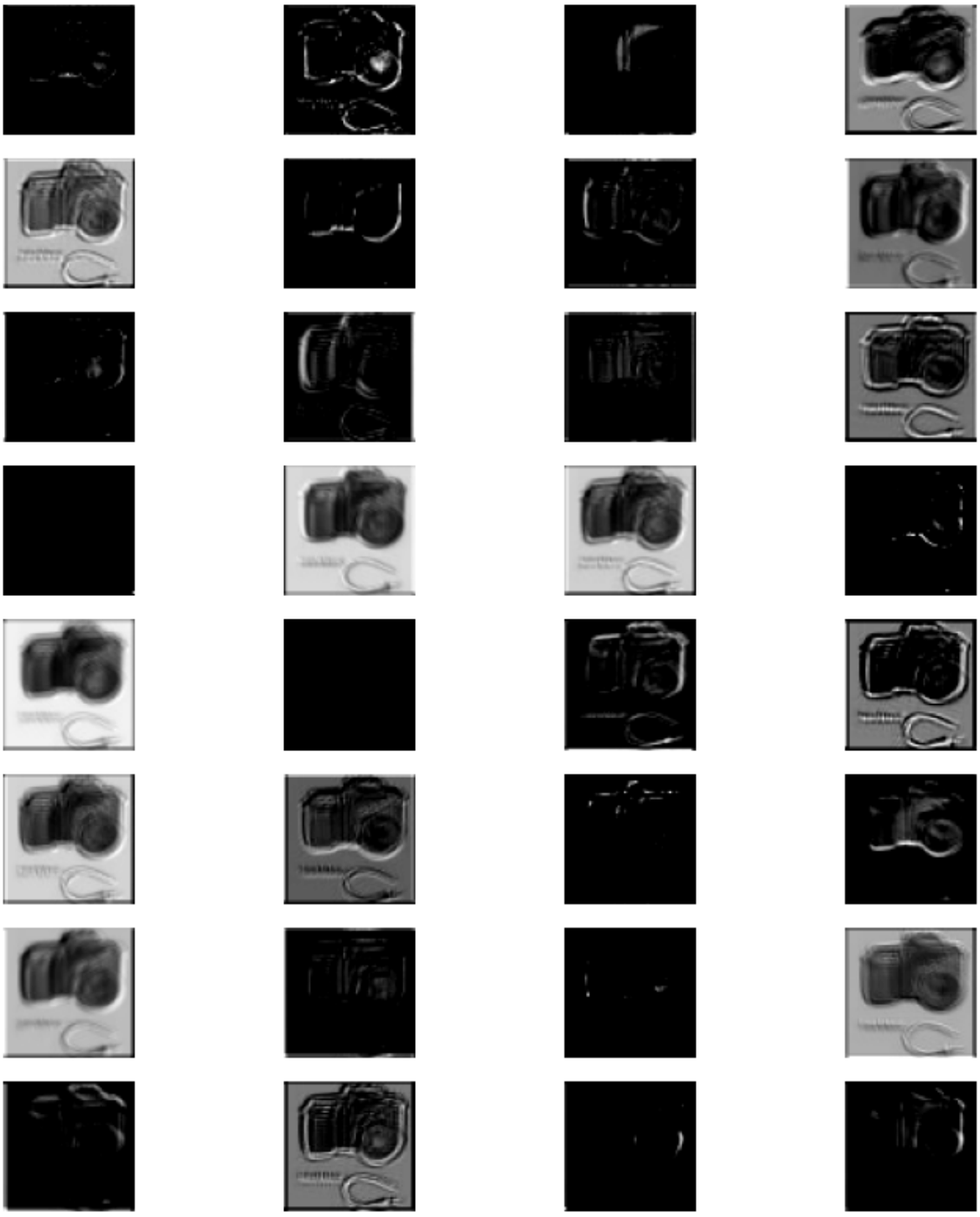
In [57]:

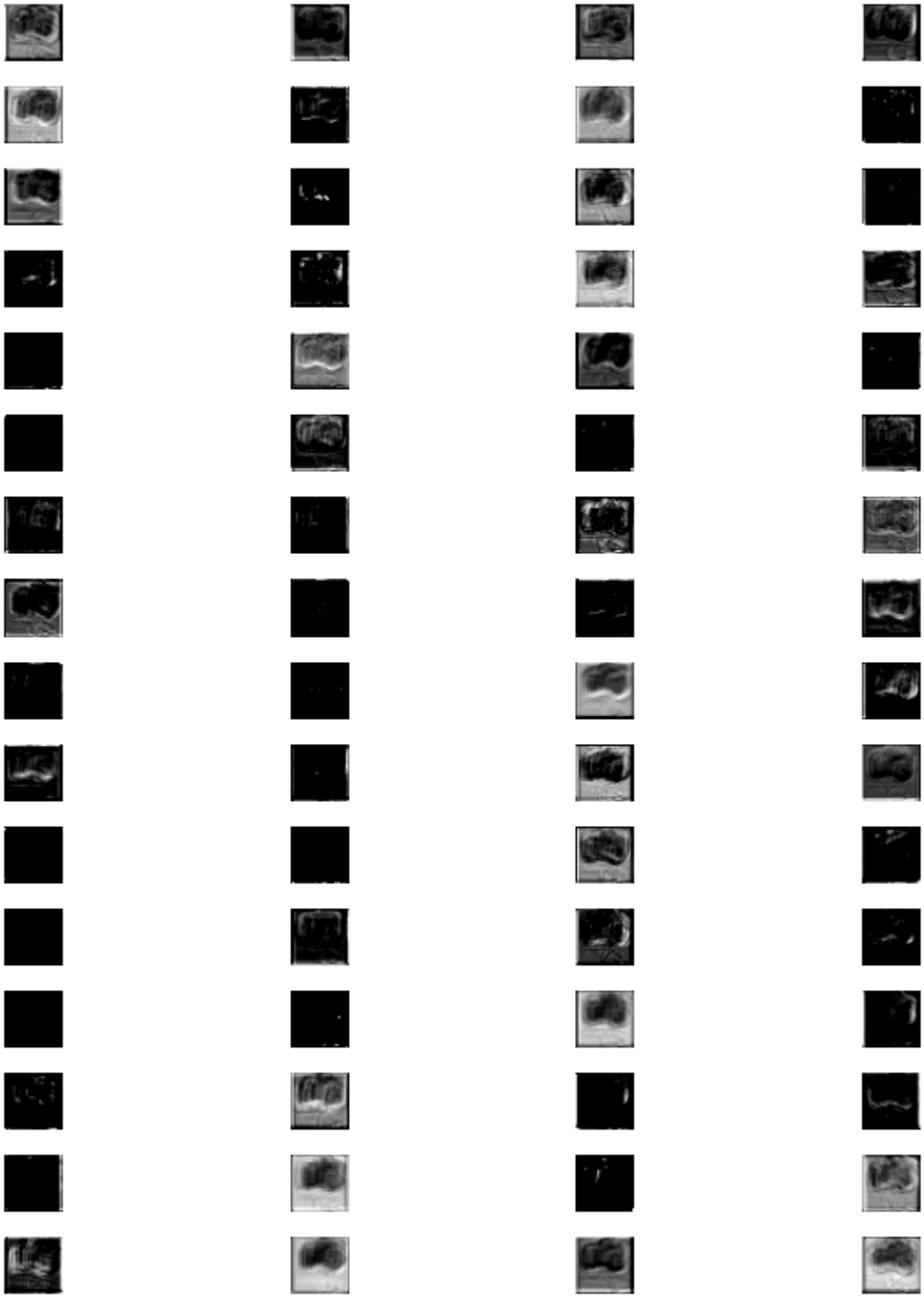
```python
def get_activation(model, input_img):
    activations = []
    hook_layer1 = model.conv1.register_forward_hook(lambda module, input, output: a
    hook_layer2 = model.conv2.register_forward_hook(lambda module, input, output: a

    with torch.no_grad():
        model.eval()
        model(input_img)

    for act in activations:
        num_channels = act.size(1)
        num_rows = (num_channels + 3) // 4  # Calculate the number of rows for subp
        plt.figure(figsize=(8, 8))
        for j in range(num_channels):
            plt.subplot(num_rows, 4, j + 1)
            plt.imshow(act[0, j].detach().numpy(), cmap='gray')
            plt.axis('off')
        plt.tight_layout()
        plt.show()

    # Remove hooks
    hook_layer1.remove()
    hook_layer2.remove()

# Assuming improvedLenet is your model class
model = improvedLenet().cpu()
```

```
# Load and preprocess images
img1 = Image.open('Dataset/train/accordion/image_0001.jpg')
img2 = Image.open('Dataset/train/camera/image_0001.jpg')
x1 = TF.to_tensor(img1).unsqueeze_(0)
x2 = TF.to_tensor(img2).unsqueeze_(0)

# Get activations for each image
get_activation(model, x1)
get_activation(model, x2)
```









ResNet Implementation

In [58]:
```
#data transform
transform = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomVerticalFlip( .5 ),
    transforms.Normalize( mean=[0.4914, 0.4822, 0.4465],
        std=[0.2023, 0.1994, 0.2010], ),
])
val_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.Normalize( mean=[0.4914, 0.4822, 0.4465],
        std=[0.2023, 0.1994, 0.2010], )
```

```
])
train_dataset = ImageDataset( is_val = False, transform = transform )
val_dataset = ImageDataset( is_val = True, transform = val_transform )

train_dataloader = DataLoader( train_dataset, batch_size = batch_size, shuffle= Tru
val_dataloader = DataLoader( val_dataset, batch_size = batch_size, shuffle= True, c
```

In [59]:
```
# implement a ResNet model here
# building network
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride = 1, downsample = None):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size = 3, stride = stride,
            nn.BatchNorm2d(out_channels),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size = 3, stride = 1, padd
            nn.BatchNorm2d(out_channels)
        )
        self.downsample = downsample
        self.relu = nn.ReLU()
        self.out_channels = out_channels

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

In [60]:
```
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes = 10):
        super(ResNet, self).__init__()
        self.inplanes = 64
        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 64, kernel_size = 7, stride = 2, padding = 3),
                        nn.BatchNorm2d(64),
                        nn.ReLU())
        self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)
        self.layer0 = self._make_layer(block, 64, layers[0], stride = 1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride = 2)
        self.layer2 = self._make_layer(block, 256, layers[2], stride = 2)
        self.layer3 = self._make_layer(block, 512, layers[3], stride = 2)
        self.avgpool = nn.AvgPool2d(7, stride=1)
        self.fc = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:

            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes
        for i in range(1, blocks):
```

```
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

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

        return x
```

In [61]:
```
num_classes = 101
num_epochs = 30
batch_size = 16
learning_rate = 0.001

model = ResNet(ResidualBlock,[2,2,2,2],num_classes).to(device)
model.apply(init_weights)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate, weight_decay =
scheduler = MultiStepLR(optimizer, [10, 15,20], gamma=0.01)
total_step = len(train_dataloader)
```

In [62]:
```
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_dataloader):
        # Move tensors to the configured device
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion  (outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print ('Epoch [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, loss.item()))

    # Validation
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in val_dataloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs
```

```
        print('Accuracy of the network on the validation images: {} %'.format(100 *
```

```
Epoch [1/30], Loss: 3.1672
Accuracy of the network on the validation images: 29.308452250274424 %
Epoch [2/30], Loss: 2.6120
Accuracy of the network on the validation images: 36.77277716794731 %
Epoch [3/30], Loss: 3.3185
Accuracy of the network on the validation images: 41.712403951701425 %
Epoch [4/30], Loss: 2.7850
Accuracy of the network on the validation images: 45.554335894621296 %
Epoch [5/30], Loss: 1.8969
Accuracy of the network on the validation images: 50.38419319429199 %
Epoch [6/30], Loss: 1.4783
Accuracy of the network on the validation images: 52.799121844127335 %
Epoch [7/30], Loss: 2.5145
Accuracy of the network on the validation images: 54.665203073545555 %
Epoch [8/30], Loss: 2.0136
Accuracy of the network on the validation images: 57.189901207464324 %
Epoch [9/30], Loss: 1.8736
Accuracy of the network on the validation images: 56.75082327113063 %
Epoch [10/30], Loss: 1.0987
Accuracy of the network on the validation images: 56.86059275521405 %
Epoch [11/30], Loss: 0.9681
Accuracy of the network on the validation images: 60.37321624588365 %
Epoch [12/30], Loss: 1.1115
Accuracy of the network on the validation images: 59.71459934138309 %
Epoch [13/30], Loss: 1.6382
Accuracy of the network on the validation images: 63.11745334796927 %
Epoch [14/30], Loss: 2.0679
Accuracy of the network on the validation images: 61.14160263446762 %
Epoch [15/30], Loss: 0.8609
Accuracy of the network on the validation images: 63.995609220636666 %
Epoch [16/30], Loss: 0.8921
Accuracy of the network on the validation images: 65.42261251372119 %
Epoch [17/30], Loss: 1.5110
Accuracy of the network on the validation images: 64.65422612513721 %
Epoch [18/30], Loss: 0.7024
Accuracy of the network on the validation images: 65.42261251372119 %
Epoch [19/30], Loss: 0.8113
Accuracy of the network on the validation images: 67.17892425905599 %
Epoch [20/30], Loss: 0.4832
Accuracy of the network on the validation images: 67.06915477497256 %
Epoch [21/30], Loss: 0.7274
Accuracy of the network on the validation images: 66.30076838638858 %
Epoch [22/30], Loss: 0.4296
Accuracy of the network on the validation images: 67.2886937431394 %
Epoch [23/30], Loss: 0.6065
Accuracy of the network on the validation images: 68.27661909989023 %
Epoch [24/30], Loss: 0.4688
Accuracy of the network on the validation images: 66.19099890230515 %
Epoch [25/30], Loss: 0.9351
Accuracy of the network on the validation images: 64.98353457738749 %
Epoch [26/30], Loss: 0.4237
Accuracy of the network on the validation images: 67.72777167947311 %
Epoch [27/30], Loss: 0.6408
Accuracy of the network on the validation images: 66.41053787047201 %
Epoch [28/30], Loss: 0.4399
Accuracy of the network on the validation images: 67.72777167947311 %
Epoch [29/30], Loss: 0.4094
Accuracy of the network on the validation images: 68.16684961580681 %
Epoch [30/30], Loss: 0.4014
Accuracy of the network on the validation images: 69.0450054884742 %
```

In [ ]: