DogEmotion

March 5, 2024

[83]:
```
#!pip install torch
```

[84]:
```
#pip install --upgrade torch torchvision
#!pip install scikit-image
```

[1]:
```
#pip install librosa
import torch                                          # imports the core PyTorch␣
 ↪library
import torch.nn as nn                                 # torch.nn library is a␣
 ↪high-level interface for building and training neural networks
import torch.optim as optim                           # Needed for optimizer
from torch.utils.data import DataLoader               # Needed for loading data
from torchvision import datasets, transforms          # Needed for making sure␣
 ↪data is properly formatted
import torch.nn.functional as F                       # Needed for functional␣
 ↪equation in forward loop
import numpy as np                                    # Needed for outputting␣
 ↪weights and biases
from torchvision.datasets import VisionDataset        # VisionDataset is is␣
 ↪designed to be a base class for datasets in computer vision tasks
from PIL import Image                                 # Needed for manipulating␣
 ↪png image files
import os                                             # Needed for operating␣
 ↪system for dealing with dir of training and test png files
from torchvision import transforms
from skimage import exposure
import matplotlib.pyplot as plt
from torch.utils.data import Subset
from sklearn.model_selection import train_test_split
from torchvision.datasets.vision import VisionDataset
from torch.utils.data import Subset
from sklearn.model_selection import train_test_split
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
#import librosa
#import librosa.display
```

```
import matplotlib.pyplot as plt
#from python_speech_features import mfcc
import numpy as np
from matplotlib.backends.backend_pdf import PdfPages
#import pygame
import random
#from skimage import exposure


# Set the device
device = "mps" if torch.backends.mps.is_available() else "cpu"
# Check PyTorch has access to MPS (Metal Performance Shader, Apple's GPU␣
 ↪architecture)
print(f"Is MPS (Metal Performance Shader) built? {torch.backends.mps.
 ↪is_built()}")
```

```
Is MPS (Metal Performance Shader) built? True
```

[2]:
```
class DogEmotions(VisionDataset):
    def __init__(self, root, split='train', transform=None,␣
 ↪target_transform=None):
        super(DogEmotions, self).__init__(root, transform=transform,␣
 ↪target_transform=target_transform)

        self.split = split
        self.data_folder = "training" if split == 'train' else "testing"
        self.images_folder = os.path.join(self.root, self.data_folder)

        self.image_paths = self._get_image_paths()
        self.mean = np.array([0.46175721, 0.42624477, 0.3737414 ])
        self.std = np.array([0.24006152, 0.2344247,  0.22821001])

        #Calculated mean: [0.46175721, 0.42624477, 0.3737414 ]
        #Calculated std: [0.24006152, 0.2344247,  0.22821001]

         # Add a check to ensure transform is not None
        if transform is None:
            self.transform = transforms.Compose([
                #transforms.CenterCrop((244, 244)),
                transforms.Resize((244, 244)),
                transforms.ToTensor(),
                #transforms.Resize((488, 488)),
                 #transforms.Lambda(lambda x: F.equalize_hist(x)),  # Adding␣
 ↪histogram equalization
                #transforms.Lambda(lambda x: Image.fromarray(self.
 ↪equalize_hist(np.array(x)))),
                #Commented out normalization for now
```

```
                #transforms.Normalize(mean=self.mean, std=self.std),
            ])
        else:
            self.transform = transform

    def equalize_hist(self, img):
        img_eq = exposure.equalize_hist(img)
        return img_eq

    def _get_image_paths(self):
        image_paths = []
        #print("Getting image paths...")
        #print(f"Checking folder: {self.images_folder}")

        for digit_folder in os.listdir(self.images_folder):
            digit_folder_path = os.path.join(self.images_folder, digit_folder)

            # Check if it's a directory before listing its contents
            if os.path.isdir(digit_folder_path):
                for image_name in os.listdir(digit_folder_path):
                    image_path = os.path.join(digit_folder_path, image_name)

                    # Skip files with the '.DS_Store' extension
                    if not image_path.endswith('.DS_Store'):
                        image_paths.append((image_path, int(digit_folder)))

        return image_paths

    def _split_dataset(self, dataset, train_size=0.70, val_size=0.15,␣
↪test_size=0.15, random_state=None):
        # Extract images and labels from the dataset
        images, labels = zip(*dataset)

        # First, split into train and temp sets
        train_images, temp_images, train_labels, temp_labels = train_test_split(
            images, labels, test_size=(val_size + test_size),␣
↪random_state=random_state
        )

        # Second, split temp set into validation and test sets
        val_images, test_images, val_labels, test_labels = train_test_split(
            temp_images, temp_labels, test_size=test_size / (val_size +␣
↪test_size), random_state=random_state
        )

        train_dataset = list(zip(train_images, train_labels))
        val_dataset = list(zip(val_images, val_labels))
```

```
        test_dataset = list(zip(test_images, test_labels))

        return train_dataset, val_dataset, test_dataset

    def get_datasets(self):
        if self.split not in ['train', 'val', 'test']:
            raise ValueError("Invalid split. Use 'train', 'val', or 'test'.")

        dataset = self._get_image_paths()
        train_dataset, val_dataset, test_dataset = self._split_dataset(dataset)

        return train_dataset, val_dataset, test_dataset
    def __getitem__(self, index):
        if index < 0 or index >= len(self.image_paths):
            raise IndexError("Index out of range")

        image_path, label = self.image_paths[index]

    # Skip files with the '.DS_Store' extension
        while image_path.endswith('.DS_Store'):
            index += 1
            if index >= len(self.image_paths):
                raise IndexError("Index out of range")
            image_path, label = self.image_paths[index]

    # Open and read the image
        print(f"Opening image: {image_path}")

        with Image.open(image_path) as image:
        # Convert image to RGB, regardless of its mode
            image = image.convert('RGB')
            print(f"Image shape before transformation: {np.array(image).
↪shape}", flush=True)

        # Ensure the image has 3 channels
            if image.mode != 'RGB':
                image = image.convert('RGB')

            if self.transform is not None:
                image = self.transform(image)
            print(f"Image shape after transformation: {image.shape}",␣
↪flush=True)

        return image, label

    def __len__(self):
        return len(self.image_paths)
```

```
    def custom_collate(self, batch):
    # Unpack the batch
        image_paths, labels = zip(*batch)

    # Load images and apply transformations
        image_tensors = [self.transform(Image.open(img_path).convert('RGB'))␣
 ↪for img_path in image_paths]

    # Convert labels to PyTorch tensor
        label_tensor = torch.tensor(labels, dtype=torch.long)

        return torch.stack(image_tensors), label_tensor
```

[3]:
```
import numpy as np
from PIL import Image
import os

# Assuming you have a folder containing images
folder_path = '/Users/peternoble/Downloads/DogEmotions2/training/0'
#folder_path = '/Users/peternoble/Desktop/Dog_Emotions/training/2'

# Get all image paths in the folder
image_paths = [os.path.join(folder_path, filename) for filename in os.
 ↪listdir(folder_path) if filename.endswith(('.png', '.jpg', '.jpeg'))]

# Calculate mean and std
mean = np.zeros(3)
std = np.zeros(3)

for img_path in image_paths:
    img = np.array(Image.open(img_path).convert('RGB')) / 255.0
    mean += np.mean(img, axis=(0, 1))
    std += np.std(img, axis=(0, 1))

mean /= len(image_paths)
std /= len(image_paths)

print("Calculated mean:", mean)
print("Calculated std:", std)
```

```
Calculated mean: [0.45628292 0.42031068 0.3604396 ]
Calculated std: [0.2422242  0.23606527 0.2257146 ]
```

[4]:
```
# Function to print class distribution
def print_class_distribution(loader, name):
    labels = [label for _, label in loader.dataset]
```

```
    print(f"{name} set size: {len(loader.dataset)}")
    for class_label in range(3):  # Assuming you have 4 classes
        count = labels.count(class_label)
        print(f"Class {class_label}: {count} examples in the {name} set")

# Create separate instances for training, validation, and test datasets
root_directory = '/Users/peternoble/Downloads/DogEmotions2'
#root_directory = '/Users/peternoble/Desktop/Dog_Emotions'
DogEmotions_instance = DogEmotions(root_directory, split='train')
train_dataset, val_dataset, test_dataset = DogEmotions_instance.get_datasets()

# Create data loaders
# Create data loaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,␣
 ↪shuffle=True, collate_fn=DogEmotions_instance.custom_collate)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32,␣
 ↪shuffle=False, collate_fn=DogEmotions_instance.custom_collate)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32,␣
 ↪shuffle=False, collate_fn=DogEmotions_instance.custom_collate)

# Print dataset sizes
print_class_distribution(train_loader, "Train")
print_class_distribution(val_loader, "Validation")
print_class_distribution(test_loader, "Test")
```

```
Train set size: 2017
Class 0: 342 examples in the Train set
Class 1: 1074 examples in the Train set
Class 2: 601 examples in the Train set
Validation set size: 432
Class 0: 76 examples in the Validation set
Class 1: 226 examples in the Validation set
Class 2: 130 examples in the Validation set
Test set size: 433
Class 0: 73 examples in the Test set
Class 1: 225 examples in the Test set
Class 2: 135 examples in the Test set
```

[5]:
```
# Set the path where you want to save the images. Useful for visualizing the␣
 ↪actual images used for validation
output_path = '/Users/peternoble/Desktop/results'
# Ensure the output directory exists
os.makedirs(output_path, exist_ok=True)
# Variables to store all images and labels
all_images = []
all_labels = []
```

```
# Iterate through the validation loader and concatenate images and labels
for batch_idx, (images, labels) in enumerate(val_loader):
    # Check for NaN or infinity values
    if torch.isnan(images).any() or torch.isinf(images).any():
        print(f"Found invalid values in batch {batch_idx}. Skipping...")
        continue

    # Clamp image values to [0, 1]
    images = torch.clamp(images, 0, 1)

    all_images.append(images)
    all_labels.append(labels)

# Concatenate the lists to get all images and labels
all_images = torch.cat(all_images, dim=0)
all_labels = torch.cat(all_labels, dim=0)

# Define a function to save images
def save_images(images, labels, output_path):
    for i in range(len(images)):
        image, label = images[i], labels[i]
        # Assuming the images are in the range [0, 1], and using torchvision to␣
 ↪convert to PIL
        image_pil = transforms.ToPILImage()(image)
        image_pil.save(os.path.join(output_path, f"image_{i}_label_{label}.
 ↪png"))

# Save all equalized images from the validation dataset
save_images(all_images, all_labels, output_path)
```

[6]:
```
class CustomCNN(nn.Module):
    def __init__(self, num_classes=3):
        super(CustomCNN, self).__init__()

        # Layer 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=2,␣
 ↪stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 2
        self.conv2 = nn.Conv2d(32, 32, kernel_size=4, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=4, stride=2)
```

```
        # Layer 3
        self.conv3 = nn.Conv2d(32, 16, kernel_size=2, padding=1)
        self.bn3 = nn.BatchNorm2d(16)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 4
        self.conv4 = nn.Conv2d(16, 8, kernel_size=2, padding=1)
        self.bn4 = nn.BatchNorm2d(8)
        self.relu4 = nn.ReLU()
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 5
        self.conv5 = nn.Conv2d(8, 4, kernel_size=2, padding=1)
        self.bn5 = nn.BatchNorm2d(4)
        self.relu5 = nn.ReLU()
        self.pool5 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 6
        self.conv6 = nn.Conv2d(4, 4, kernel_size=2, padding=1)
        self.bn6 = nn.BatchNorm2d(4)
        self.relu6 = nn.ReLU()
        self.pool6 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 7
        self.conv7 = nn.Conv2d(4, 16, kernel_size=2, padding=1)
        self.bn7 = nn.BatchNorm2d(16)
        self.relu7 = nn.ReLU()
        self.pool7 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Layer 8
        self.conv8 = nn.Conv2d(16, 32, kernel_size=2, padding=1)
        self.bn8 = nn.BatchNorm2d(32)
        self.relu8 = nn.ReLU()
        self.pool8 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Adjusted fully connected layer
        #self.fc = nn.Linear(1952, num_classes)  # Adjust the size based on␣
↪your model architecture
        #self.fc = nn.Linear(128, num_classes)  # Adjust the size based on your␣
↪model architecture
        self.fc = nn.Linear(32, num_classes)  # Adjust the size based on your␣
↪model architecture

    def forward(self, x):
        x = self.pool1(self.relu1(self.bn1(self.conv1(x))))
        x = self.pool2(self.relu2(self.bn2(self.conv2(x))))
```

```
        x = self.pool3(self.relu3(self.bn3(self.conv3(x))))
        x = self.pool4(self.relu4(self.bn4(self.conv4(x))))
        x = self.pool5(self.relu5(self.bn5(self.conv5(x))))
        x = self.pool6(self.relu6(self.bn6(self.conv6(x))))
        x = self.pool7(self.relu7(self.bn7(self.conv7(x))))
        x = self.pool8(self.relu8(self.bn8(self.conv8(x))))

        # Flatten the output before passing it through the fully connected layer
        x = x.view(x.size(0), -1)

        # Fully connected layer
        x = self.fc(x)

        return x

# Create an instance of the CustomCNN model
model = CustomCNN()

# Print the model architecture
print(model)
```

```
CustomCNN(
  (conv1): Conv2d(3, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=4, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3): Conv2d(32, 16, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu3): ReLU()
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv4): Conv2d(16, 8, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn4): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu4): ReLU()
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv5): Conv2d(8, 4, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
  (relu5): ReLU()
  (pool5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv6): Conv2d(4, 4, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn6): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu6): ReLU()
  (pool6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv7): Conv2d(4, 16, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn7): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu7): ReLU()
  (pool7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv8): Conv2d(16, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (bn8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu8): ReLU()
  (pool8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

[7]:
```
# Assuming sugarcane_loader is your DataLoader
for inputs, labels in val_loader:
    print("Input shape:", inputs.shape)
    break
```

```
Input shape: torch.Size([32, 3, 244, 244])
```

[ ]:
```
```

[8]:
```
#Instantiate your model, optimizer, and criterion
model = CustomCNN()  # Create an instance of your CustomModel
#optimizer = optim.Adam(model.parameters(), lr=0.001)
# Assuming you have a validation DataLoader named val_loader
# Specify the weight decay (regularization strength)
weight_decay = 0.01
#loaded_model = CustomCNN()
#loaded_model.load_state_dict(torch.load('complex_model_95_8.pth'))

# Define your optimizer and pass the weight_decay parameter L2 regular
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=weight_decay)
#optimizer = optim.Adam(model.parameters(), lr=0.001)
# Define RMSprop optimizer
#optimizer = optim.RMSprop(model.parameters(), lr=0.001, alpha=0.9)
```

```
criterion = nn.CrossEntropyLoss()

num_epochs = 50

class EarlyStopping:
    def __init__(self, patience=5, delta=0, verbose=False):
        self.patience = patience
        self.delta = delta
        self.verbose = verbose
        self.counter = 0
        self.best_score = None
        self.early_stop = False

    def __call__(self, val_loss, model):
        score = -val_loss

        if self.best_score is None:
            self.best_score = score
        elif score < self.best_score + self.delta:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.counter = 0

        if self.verbose:
            print(f'EarlyStopping counter: {self.counter} out of {self.
 ↪patience}')

        return self.early_stop

# Create an instance of EarlyStopping before the training loop
early_stopping = EarlyStopping(patience=15, verbose=True)

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Initialize variables for accuracy calculation
correct_train = 0
total_train = 0
correct_val = 0
total_val = 0
```

```
for epoch in range(num_epochs):
    model.train()
    running_train_loss = 0.0

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_train_loss += loss.item()

        _, predicted_train = outputs.max(1)
        total_train += labels.size(0)
        correct_train += predicted_train.eq(labels).sum().item()

    average_train_loss = running_train_loss / len(train_loader)
    train_losses.append(average_train_loss)
    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy)

    model.eval()
    running_val_loss = 0.0

    # Reset variables for accuracy calculation
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for val_inputs, val_labels in val_loader:
            val_outputs = model(val_inputs)
            val_loss = criterion(val_outputs, val_labels)
            running_val_loss += val_loss.item()

            _, predicted_val = val_outputs.max(1)
            total_val += val_labels.size(0)
            correct_val += predicted_val.eq(val_labels).sum().item()

    average_val_loss = running_val_loss / len(val_loader)
    val_losses.append(average_val_loss)
    val_accuracy = correct_val / total_val
    val_accuracies.append(val_accuracy)

    # Call early_stopping within the loop
    if early_stopping(average_val_loss, model):
        print("Early stopping")
        break
```

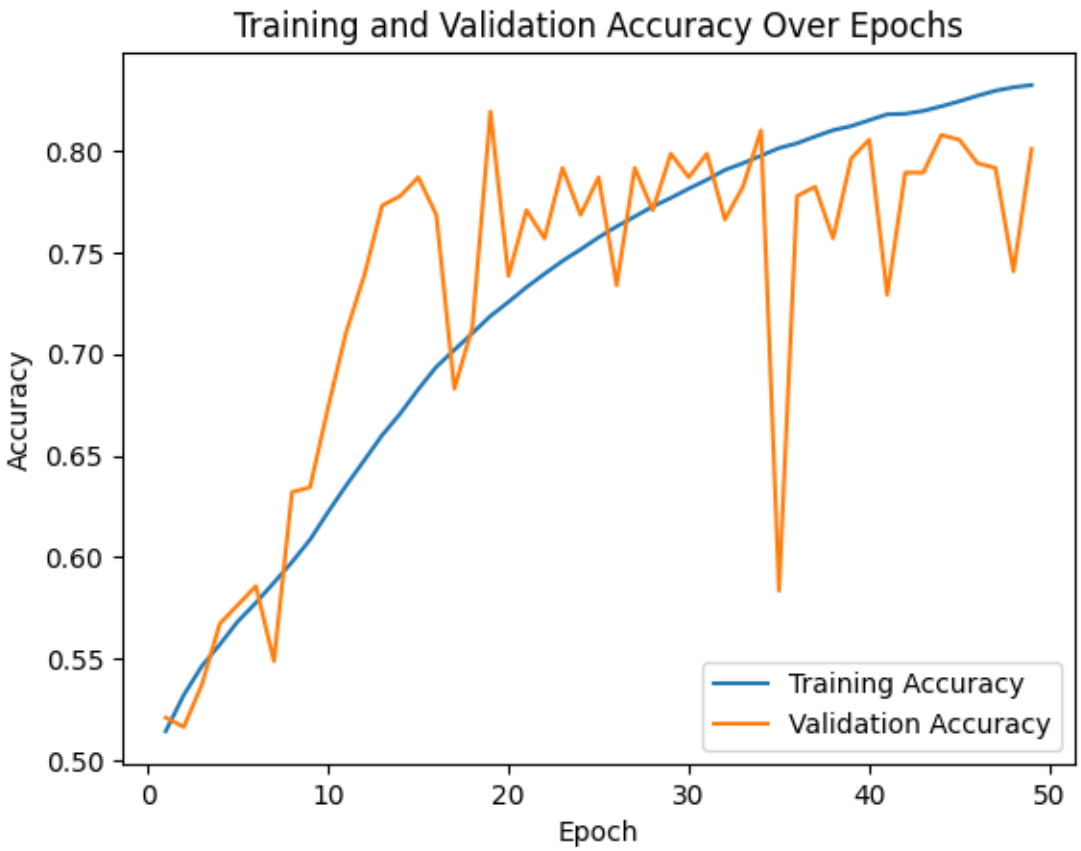
```
    print(f"Epoch {epoch + 1}/{num_epochs}, Training Loss: {average_train_loss:.
↪4f}, Training Accuracy: {train_accuracy:.4f}, Validation Loss: {average_val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")

# Plotting the accuracy curve
epochs = range(1, len(train_accuracies) + 1)
plt.plot(epochs, train_accuracies, label='Training Accuracy')
plt.plot(epochs, val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy Over Epochs')
plt.legend()
plt.show()

# Plotting the loss curve
epochs = range(1, len(train_losses) + 1)
plt.plot(epochs, train_losses, label='Training Loss')
plt.plot(epochs, val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.legend()
plt.show()
```

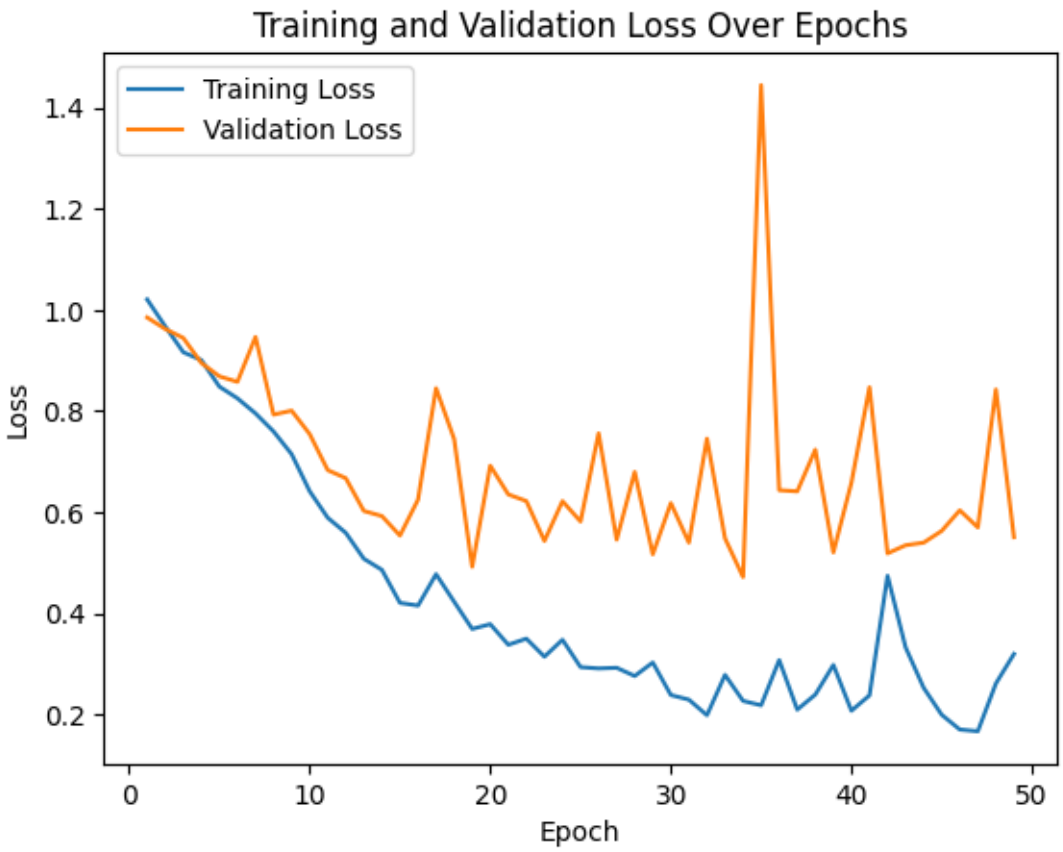
```
EarlyStopping counter: 0 out of 15
Epoch 1/50, Training Loss: 1.0206, Training Accuracy: 0.5141, Validation Loss:
0.9850, Validation Accuracy: 0.5208
EarlyStopping counter: 0 out of 15
Epoch 2/50, Training Loss: 0.9675, Training Accuracy: 0.5320, Validation Loss:
0.9623, Validation Accuracy: 0.5162
EarlyStopping counter: 0 out of 15
Epoch 3/50, Training Loss: 0.9165, Training Accuracy: 0.5462, Validation Loss:
0.9446, Validation Accuracy: 0.5370
EarlyStopping counter: 0 out of 15
Epoch 4/50, Training Loss: 0.9008, Training Accuracy: 0.5569, Validation Loss:
0.8952, Validation Accuracy: 0.5671
EarlyStopping counter: 0 out of 15
Epoch 5/50, Training Loss: 0.8484, Training Accuracy: 0.5683, Validation Loss:
0.8687, Validation Accuracy: 0.5764
EarlyStopping counter: 0 out of 15
Epoch 6/50, Training Loss: 0.8253, Training Accuracy: 0.5774, Validation Loss:
0.8579, Validation Accuracy: 0.5856
EarlyStopping counter: 1 out of 15
Epoch 7/50, Training Loss: 0.7956, Training Accuracy: 0.5874, Validation Loss:
0.9465, Validation Accuracy: 0.5486
EarlyStopping counter: 0 out of 15
Epoch 8/50, Training Loss: 0.7599, Training Accuracy: 0.5975, Validation Loss:
```

```
0.7931, Validation Accuracy: 0.6319
EarlyStopping counter: 1 out of 15
Epoch 9/50, Training Loss: 0.7151, Training Accuracy: 0.6085, Validation Loss:
0.8010, Validation Accuracy: 0.6343
EarlyStopping counter: 0 out of 15
Epoch 10/50, Training Loss: 0.6419, Training Accuracy: 0.6222, Validation Loss:
0.7548, Validation Accuracy: 0.6736
EarlyStopping counter: 0 out of 15
Epoch 11/50, Training Loss: 0.5894, Training Accuracy: 0.6351, Validation Loss:
0.6835, Validation Accuracy: 0.7106
EarlyStopping counter: 0 out of 15
Epoch 12/50, Training Loss: 0.5596, Training Accuracy: 0.6475, Validation Loss:
0.6676, Validation Accuracy: 0.7384
EarlyStopping counter: 0 out of 15
Epoch 13/50, Training Loss: 0.5084, Training Accuracy: 0.6600, Validation Loss:
0.6028, Validation Accuracy: 0.7731
EarlyStopping counter: 0 out of 15
Epoch 14/50, Training Loss: 0.4865, Training Accuracy: 0.6706, Validation Loss:
0.5928, Validation Accuracy: 0.7778
EarlyStopping counter: 0 out of 15
Epoch 15/50, Training Loss: 0.4213, Training Accuracy: 0.6826, Validation Loss:
0.5543, Validation Accuracy: 0.7870
EarlyStopping counter: 1 out of 15
Epoch 16/50, Training Loss: 0.4160, Training Accuracy: 0.6936, Validation Loss:
0.6252, Validation Accuracy: 0.7685
EarlyStopping counter: 2 out of 15
Epoch 17/50, Training Loss: 0.4779, Training Accuracy: 0.7020, Validation Loss:
0.8452, Validation Accuracy: 0.6829
EarlyStopping counter: 3 out of 15
Epoch 18/50, Training Loss: 0.4236, Training Accuracy: 0.7102, Validation Loss:
0.7450, Validation Accuracy: 0.7130
EarlyStopping counter: 0 out of 15
Epoch 19/50, Training Loss: 0.3698, Training Accuracy: 0.7187, Validation Loss:
0.4927, Validation Accuracy: 0.8194
EarlyStopping counter: 1 out of 15
Epoch 20/50, Training Loss: 0.3792, Training Accuracy: 0.7256, Validation Loss:
0.6922, Validation Accuracy: 0.7384
EarlyStopping counter: 2 out of 15
Epoch 21/50, Training Loss: 0.3385, Training Accuracy: 0.7329, Validation Loss:
0.6355, Validation Accuracy: 0.7708
EarlyStopping counter: 3 out of 15
Epoch 22/50, Training Loss: 0.3506, Training Accuracy: 0.7395, Validation Loss:
0.6224, Validation Accuracy: 0.7569
EarlyStopping counter: 4 out of 15
Epoch 23/50, Training Loss: 0.3150, Training Accuracy: 0.7458, Validation Loss:
0.5436, Validation Accuracy: 0.7917
EarlyStopping counter: 5 out of 15
Epoch 24/50, Training Loss: 0.3484, Training Accuracy: 0.7515, Validation Loss:
```

```
0.6222, Validation Accuracy: 0.7685
EarlyStopping counter: 6 out of 15
Epoch 25/50, Training Loss: 0.2944, Training Accuracy: 0.7574, Validation Loss:
0.5820, Validation Accuracy: 0.7870
EarlyStopping counter: 7 out of 15
Epoch 26/50, Training Loss: 0.2918, Training Accuracy: 0.7628, Validation Loss:
0.7563, Validation Accuracy: 0.7338
EarlyStopping counter: 8 out of 15
Epoch 27/50, Training Loss: 0.2931, Training Accuracy: 0.7677, Validation Loss:
0.5461, Validation Accuracy: 0.7917
EarlyStopping counter: 9 out of 15
Epoch 28/50, Training Loss: 0.2768, Training Accuracy: 0.7726, Validation Loss:
0.6800, Validation Accuracy: 0.7708
EarlyStopping counter: 10 out of 15
Epoch 29/50, Training Loss: 0.3035, Training Accuracy: 0.7769, Validation Loss:
0.5172, Validation Accuracy: 0.7986
EarlyStopping counter: 11 out of 15
Epoch 30/50, Training Loss: 0.2393, Training Accuracy: 0.7815, Validation Loss:
0.6183, Validation Accuracy: 0.7870
EarlyStopping counter: 12 out of 15
Epoch 31/50, Training Loss: 0.2301, Training Accuracy: 0.7858, Validation Loss:
0.5403, Validation Accuracy: 0.7986
EarlyStopping counter: 13 out of 15
Epoch 32/50, Training Loss: 0.1993, Training Accuracy: 0.7906, Validation Loss:
0.7458, Validation Accuracy: 0.7662
EarlyStopping counter: 14 out of 15
Epoch 33/50, Training Loss: 0.2791, Training Accuracy: 0.7939, Validation Loss:
0.5488, Validation Accuracy: 0.7824
EarlyStopping counter: 0 out of 15
Epoch 34/50, Training Loss: 0.2277, Training Accuracy: 0.7977, Validation Loss:
0.4723, Validation Accuracy: 0.8102
EarlyStopping counter: 1 out of 15
Epoch 35/50, Training Loss: 0.2190, Training Accuracy: 0.8014, Validation Loss:
1.4440, Validation Accuracy: 0.5833
EarlyStopping counter: 2 out of 15
Epoch 36/50, Training Loss: 0.3083, Training Accuracy: 0.8038, Validation Loss:
0.6439, Validation Accuracy: 0.7778
EarlyStopping counter: 3 out of 15
Epoch 37/50, Training Loss: 0.2104, Training Accuracy: 0.8071, Validation Loss:
0.6416, Validation Accuracy: 0.7824
EarlyStopping counter: 4 out of 15
Epoch 38/50, Training Loss: 0.2401, Training Accuracy: 0.8103, Validation Loss:
0.7239, Validation Accuracy: 0.7569
EarlyStopping counter: 5 out of 15
Epoch 39/50, Training Loss: 0.2986, Training Accuracy: 0.8122, Validation Loss:
0.5209, Validation Accuracy: 0.7963
EarlyStopping counter: 6 out of 15
Epoch 40/50, Training Loss: 0.2083, Training Accuracy: 0.8152, Validation Loss:
```

```
0.6588, Validation Accuracy: 0.8056
EarlyStopping counter: 7 out of 15
Epoch 41/50, Training Loss: 0.2386, Training Accuracy: 0.8181, Validation Loss:
0.8471, Validation Accuracy: 0.7292
EarlyStopping counter: 8 out of 15
Epoch 42/50, Training Loss: 0.4748, Training Accuracy: 0.8183, Validation Loss:
0.5191, Validation Accuracy: 0.7894
EarlyStopping counter: 9 out of 15
Epoch 43/50, Training Loss: 0.3334, Training Accuracy: 0.8198, Validation Loss:
0.5351, Validation Accuracy: 0.7894
EarlyStopping counter: 10 out of 15
Epoch 44/50, Training Loss: 0.2535, Training Accuracy: 0.8220, Validation Loss:
0.5402, Validation Accuracy: 0.8079
EarlyStopping counter: 11 out of 15
Epoch 45/50, Training Loss: 0.2002, Training Accuracy: 0.8245, Validation Loss:
0.5635, Validation Accuracy: 0.8056
EarlyStopping counter: 12 out of 15
Epoch 46/50, Training Loss: 0.1711, Training Accuracy: 0.8272, Validation Loss:
0.6041, Validation Accuracy: 0.7940
EarlyStopping counter: 13 out of 15
Epoch 47/50, Training Loss: 0.1678, Training Accuracy: 0.8297, Validation Loss:
0.5703, Validation Accuracy: 0.7917
EarlyStopping counter: 14 out of 15
Epoch 48/50, Training Loss: 0.2621, Training Accuracy: 0.8314, Validation Loss:
0.8428, Validation Accuracy: 0.7407
EarlyStopping counter: 15 out of 15
Early stopping
```


Training and Validation Accuracy Over Epochs
Accuracy
Epoch
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0
10
20
30
40
50
Training Accuracy
Validation Accuracy



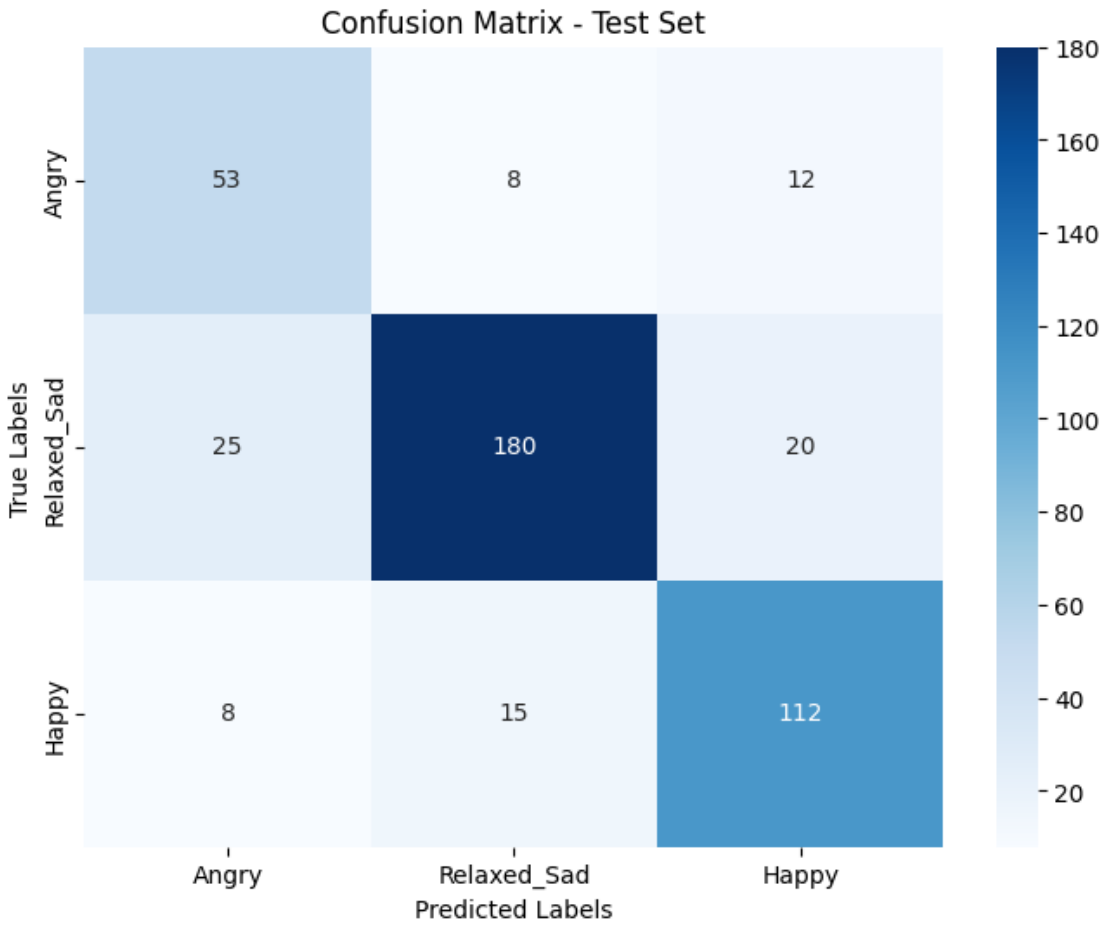
[9]:
```
# Save the model state_dict
torch.save(model.state_dict(), 'dog_emotions_model.pth')
```

[10]:
```
# Assuming you have a validation DataLoader named val_loader
loaded_model = CustomCNN()
loaded_model.load_state_dict(torch.load('dog_emotions_model.pth'))

def evaluate_model(loader, custom_labels, dataset_name):
    model.eval()  # Set the model to evaluation mode
    all_predictions = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in loader:
            try:
                inputs = inputs.view(inputs.size(0), 3, 244, 244)  # Ensure the␣
 ↪input size is correct
                outputs = model(inputs)
                predictions = torch.argmax(outputs, dim=1)
```

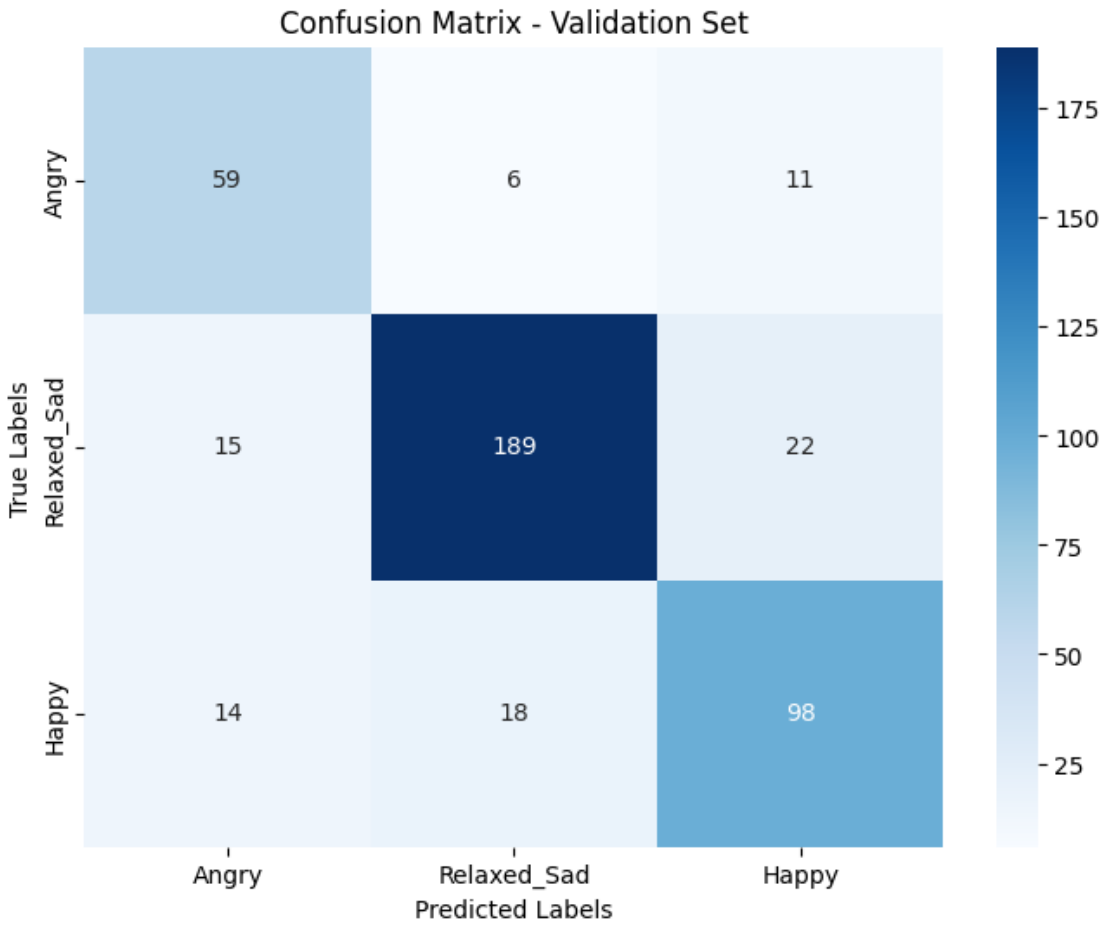
```
                all_predictions.extend(predictions.tolist())
                all_labels.extend(labels.tolist())  # Populate the true labels
            except Exception as e:
                print(f"An error occurred: {e}")

    # Compute accuracy for the dataset
    if len(all_labels) > 0:
        accuracy = sum(p == l for p, l in zip(all_predictions, all_labels)) /␣
 ↪len(all_labels)
        print(f"{dataset_name} Accuracy: {accuracy:.1%}")
    else:
        print(f"No labels available for computing {dataset_name} accuracy.")

    # Analyze classification breakdown for the dataset
    for class_label, custom_label in enumerate(custom_labels):
        correct = sum(p == class_label and l == class_label for p, l in␣
 ↪zip(all_predictions, all_labels))
        total_in_set = all_labels.count(class_label)

        if total_in_set == 0:
            print(f"{custom_label}: No examples in the {dataset_name} set")
        else:
            percentage = correct / total_in_set if total_in_set != 0 else 0.0
            print(f"{custom_label}: Correctly classified {correct}/
 ↪{total_in_set} ({percentage:.1%})")

    # Return the lists of all_labels and all_predictions
    return all_labels, all_predictions

# Define custom labels
custom_labels = ['Angry', 'Relaxed_Sad', 'Happy']

# Evaluate on the test set
all_labels_test, all_predictions_test = evaluate_model(test_loader,␣
 ↪custom_labels, "\nTest")

# Compute confusion matrix for the test set
conf_matrix_test = confusion_matrix(all_labels_test, all_predictions_test)

# Plot the confusion matrix for the test set
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=custom_labels, yticklabels=custom_labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Test Set')
plt.show()
```

```
# Generate and print classification report for the test set
class_report_test = classification_report(all_labels_test,␣
 ↪all_predictions_test, target_names=custom_labels)
print("Classification Report - Test Set:\n", class_report_test)

# Evaluate on the validation set
all_labels_val, all_predictions_val = evaluate_model(val_loader, custom_labels,␣
 ↪"Validation")

# Compute confusion matrix for the validation set
conf_matrix_val = confusion_matrix(all_labels_val, all_predictions_val)

# Plot the confusion matrix for the validation set
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_val, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=custom_labels, yticklabels=custom_labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Validation Set')
plt.show()

# Generate and print classification report for the validation set
class_report_val = classification_report(all_labels_val, all_predictions_val,␣
 ↪target_names=custom_labels)
print("Classification Report - Validation Set:\n", class_report_val)
```

```
Test Accuracy: 79.7%
Angry: Correctly classified 53/73 (72.6%)
Relaxed_Sad: Correctly classified 180/225 (80.0%)
Happy: Correctly classified 112/135 (83.0%)
```




```
Classification Report - Test Set:
              precision    recall  f1-score   support

       Angry       0.62      0.73      0.67        73
 Relaxed_Sad       0.89      0.80      0.84       225
       Happy       0.78      0.83      0.80       135

    accuracy                           0.80       433
   macro avg       0.76      0.79      0.77       433
weighted avg       0.81      0.80      0.80       433

Validation Accuracy: 80.1%
Angry: Correctly classified 59/76 (77.6%)
Relaxed_Sad: Correctly classified 189/226 (83.6%)
Happy: Correctly classified 98/130 (75.4%)
```




```
Classification Report - Validation Set:
              precision    recall  f1-score   support

       Angry       0.67      0.78      0.72        76
 Relaxed_Sad       0.89      0.84      0.86       226
       Happy       0.75      0.75      0.75       130

    accuracy                           0.80       432
   macro avg       0.77      0.79      0.78       432
weighted avg       0.81      0.80      0.80       432
```

[11]:
```
import torch

# Load the model checkpoint
checkpoint = torch.load("/Users/peternoble/Desktop/dog_emotions_model.pth",␣
 ↪map_location=torch.device('cpu'))
```

```
# Inspect the contents of the checkpoint
print(checkpoint.keys())
```

```
odict_keys(['conv1.weight', 'conv1.bias', 'bn1.weight', 'bn1.bias',
'bn1.running_mean', 'bn1.running_var', 'bn1.num_batches_tracked',
'conv2.weight', 'conv2.bias', 'bn2.weight', 'bn2.bias', 'bn2.running_mean',
'bn2.running_var', 'bn2.num_batches_tracked', 'conv3.weight', 'conv3.bias',
'bn3.weight', 'bn3.bias', 'bn3.running_mean', 'bn3.running_var',
'bn3.num_batches_tracked', 'conv4.weight', 'conv4.bias', 'bn4.weight',
'bn4.bias', 'bn4.running_mean', 'bn4.running_var', 'bn4.num_batches_tracked',
'conv5.weight', 'conv5.bias', 'bn5.weight', 'bn5.bias', 'bn5.running_mean',
'bn5.running_var', 'bn5.num_batches_tracked', 'conv6.weight', 'conv6.bias',
'bn6.weight', 'bn6.bias', 'bn6.running_mean', 'bn6.running_var',
'bn6.num_batches_tracked', 'conv7.weight', 'conv7.bias', 'bn7.weight',
'bn7.bias', 'bn7.running_mean', 'bn7.running_var', 'bn7.num_batches_tracked',
'conv8.weight', 'conv8.bias', 'bn8.weight', 'bn8.bias', 'bn8.running_mean',
'bn8.running_var', 'bn8.num_batches_tracked', 'fc.weight', 'fc.bias'])
```

[12]:
```
import torch

# Load the model checkpoint
checkpoint_path = "/Users/peternoble/Desktop/dog_emotions_model.pth"
model = CustomCNN()
model.load_state_dict(torch.load(checkpoint_path))
model.eval()
# Access the parameters of the BatchNorm2d layer
bn_key = 'bn8.'
bn_params = {key[len(bn_key):]: value for key, value in model.state_dict().
 ↪items() if bn_key in key}

# Access scale (weight) and shift (bias) parameters
gamma = bn_params['weight']
beta = bn_params['bias']

# Access running mean and running variance parameters
running_mean = bn_params['running_mean']
running_var = bn_params['running_var']

# Access num_batches_tracked parameter
num_batches_tracked = bn_params['num_batches_tracked']

# Print or use the parameters as needed
print("Weight (Gamma):", gamma)
print("Bias (Beta):", beta)
print("Running Mean:", running_mean)
print("Running Variance:", running_var)
print("Num Batches Tracked:", num_batches_tracked)
```

```
# Open a text file in write mode
with open("batch_norm_parameters.txt", "w") as file:
    # Write the parameters to the file
    file.write(f"Weight (Gamma): {gamma}\n")
    file.write(f"Bias (Beta): {beta}\n")
    file.write(f"Running Mean: {running_mean}\n")
    file.write(f"Running Variance: {running_var}\n")
    file.write(f"Num Batches Tracked: {num_batches_tracked}\n")

# Print a message indicating the file has been saved
print("Batch normalization parameters saved to batch_norm_parameters.txt")
```

```
Weight (Gamma): tensor([0.2421, 0.6517, 0.7137, 0.0070, 0.5303, 0.4873, 0.5686,
0.6924, 0.4553,
        0.2300, 0.4664, 0.4022, 0.6846, 0.4057, 0.4098, 0.4579, 0.5266, 0.3157,
        0.6251, 0.6455, 0.6788, 0.1144, 0.4966, 0.2436, 0.1986, 0.6828, 0.6356,
        0.7407, 0.4212, 0.0177, 0.2577, 0.6711])
Bias (Beta): tensor([ 0.0348, -0.0064,  0.1010,  0.0049,  0.0627,  0.0596,
-0.0276, -0.0086,
         0.0428,  0.0331,  0.0551,  0.0368,  0.0053, -0.0107,  0.0417,  0.0454,
        -0.0292,  0.0129,  0.0153,  0.0727,  0.1081,  0.0189,  0.0004,  0.0586,
         0.0368, -0.0470, -0.0058,  0.1157, -0.0035,  0.0041,  0.0381, -0.0142])
Running Mean: tensor([-0.0302, -0.0463,  0.0630, -0.0057, -0.0142, -0.0358,
0.0263,  0.0369,
        -0.0872,  0.0343,  0.1079,  0.1314,  0.0927,  0.0914,  0.0720,  0.0742,
         0.0306,  0.0671,  0.0330,  0.1344,  0.0251,  0.0659,  0.0907,  0.0799,
        -0.0031,  0.0405,  0.0918,  0.0798,  0.0465, -0.0216,  0.0548,  0.0631])
Running Variance: tensor([0.0081, 0.0311, 0.0183, 0.0003, 0.0220, 0.0146,
0.0143, 0.0170, 0.0150,
        0.0094, 0.0202, 0.0170, 0.0534, 0.0372, 0.0119, 0.0153, 0.0107, 0.0132,
        0.0186, 0.0246, 0.0237, 0.0034, 0.0383, 0.0116, 0.0043, 0.0189, 0.0475,
        0.0196, 0.0323, 0.0005, 0.0056, 0.0209])
Num Batches Tracked: tensor(3136)
Batch normalization parameters saved to batch_norm_parameters.txt
```

[ ]:

[44]:
```
model.eval()  # Set the model to evaluation mode
params = list(loaded_model.parameters())
#print(params[0].shape)
#print(params[1].shape)
#print(params[2].shape)
#print(params[3].shape)
print(params)
```

```
[Parameter containing:
tensor([[[[-1.2862e-01, -7.4132e-02],
```

```
  [-5.1123e-02, -2.4877e-03]],

 [[-1.9497e-01,  4.1154e-02],
  [-1.8533e-01, -3.4249e-02]],

 [[ 1.6030e-01,  2.4662e-01],
  [-8.4199e-02, -2.5276e-01]]],


[[[-6.9489e-03, -1.1550e-01],
  [ 1.4490e-01,  1.5836e-01]],

 [[ 1.6745e-01, -6.6469e-02],
  [ 9.3122e-02, -6.3083e-02]],

 [[ 2.7279e-01, -5.6057e-02],
  [-2.7427e-01,  2.2126e-01]]],


[[[-2.0808e-01,  2.3376e-01],
  [ 1.7773e-01,  1.2671e-01]],

 [[-5.4855e-02, -1.8969e-01],
  [-2.1482e-01,  1.1987e-01]],

 [[-2.3454e-01, -1.4718e-01],
  [ 1.3466e-01,  2.5079e-01]]],


[[[ 1.0878e-01, -2.1913e-01],
  [-2.0339e-01,  3.4536e-03]],

 [[ 1.4468e-01, -2.6049e-01],
  [-1.4162e-01, -2.0044e-01]],

 [[-6.1424e-02, -1.1785e-01],
  [ 1.1778e-01, -5.3533e-02]]],


[[[ 4.3818e-02,  1.9947e-01],
  [ 1.1587e-01,  2.6819e-01]],

 [[ 2.8754e-01,  2.1529e-01],
  [ 2.8586e-01, -1.3636e-01]],

 [[-1.5417e-01, -2.6618e-01],
  [ 1.6691e-01,  1.4499e-01]]],
```

```
[[[-2.1729e-01, -1.9851e-01],
  [-2.4967e-01, -1.3278e-01]],

 [[-2.1608e-01,  2.6612e-01],
  [-1.5269e-01, -9.3307e-02]],

 [[-1.8551e-01, -7.0948e-02],
  [ 2.7045e-01,  7.6375e-02]]],


[[[-2.2386e-01, -1.2155e-01],
  [-1.8215e-01, -1.0601e-02]],

 [[-2.4377e-02, -9.8831e-02],
  [-1.6564e-01,  1.3581e-01]],

 [[ 3.0330e-02, -2.6251e-01],
  [-2.7283e-01,  2.9806e-02]]],


[[[ 1.2161e-04,  5.3137e-02],
  [-1.9250e-01,  2.4779e-01]],

 [[ 8.2522e-02, -2.2375e-01],
  [ 4.1588e-02,  1.3680e-01]],

 [[ 2.3286e-01,  2.5193e-01],
  [-1.4712e-01, -5.6759e-02]]],


[[[ 1.2833e-01, -2.1341e-01],
  [-7.9442e-02, -1.9326e-01]],

 [[-1.1788e-01,  9.0007e-02],
  [-1.2410e-01, -2.1450e-01]],

 [[-2.0532e-01,  5.7660e-02],
  [-2.0129e-01,  2.7650e-01]]],


[[[-1.7102e-01,  2.8426e-01],
  [-1.6310e-01, -4.2167e-02]],

 [[-2.2540e-01, -7.3587e-02],
  [ 2.5221e-01,  2.5372e-01]],

 [[-2.1754e-01, -2.4622e-01],
```

```
  [-2.6112e-01, -1.1207e-01]]],


[[[ 1.6128e-01,  2.4160e-01],
  [-2.8101e-01, -2.0124e-01]],

 [[ 2.9695e-02,  6.4366e-02],
  [-1.1276e-01,  3.9310e-02]],

 [[-2.3562e-01, -4.5487e-02],
  [-1.2769e-01, -1.2402e-01]]],


[[[ 1.9316e-01, -7.1542e-02],
  [ 1.7880e-02,  1.2100e-01]],

 [[ 7.3645e-02, -4.1847e-02],
  [-1.3371e-01,  1.4042e-01]],

 [[-9.7902e-02,  1.7557e-01],
  [-1.2173e-01,  1.2777e-01]]],


[[[-1.0646e-01,  1.7544e-01],
  [-1.9008e-01, -2.8179e-01]],

 [[-2.1405e-01,  2.8287e-01],
  [-9.1039e-02,  2.7296e-01]],

 [[ 1.0285e-01,  2.2912e-01],
  [ 1.6586e-01, -1.3051e-01]]],


[[[ 1.1123e-01,  9.8354e-03],
  [ 1.6477e-01, -1.3836e-01]],

 [[ 2.2818e-01,  1.8638e-01],
  [-1.5850e-01, -6.5581e-02]],

 [[-2.1204e-01,  1.7747e-01],
  [-2.3936e-01, -5.2997e-02]]],


[[[ 2.0355e-01,  1.2741e-01],
  [-1.1312e-01, -1.8448e-01]],

 [[ 1.8240e-01, -1.8318e-02],
  [-4.4789e-02,  1.0897e-01]],
```

```
 [[ 4.9810e-02,  2.0592e-01],
  [ 2.2145e-01,  5.8419e-02]]],


[[[-2.4172e-01, -1.0820e-01],
  [ 1.2886e-01, -6.8874e-02]],

 [[ 2.2063e-01,  2.3657e-01],
  [ 1.4910e-01, -3.2015e-02]],

 [[-2.2320e-01,  1.2945e-01],
  [-2.0851e-01,  2.4967e-01]]],


[[[-2.5639e-01, -7.5443e-02],
  [-3.2454e-02, -6.6399e-02]],

 [[ 1.1102e-01, -1.1391e-01],
  [ 2.6951e-01, -1.9989e-01]],

 [[ 1.3029e-01,  2.1191e-01],
  [-2.1165e-01,  3.9331e-02]]],


[[[-3.3680e-02,  1.1208e-01],
  [ 2.2649e-01,  1.4871e-02]],

 [[ 8.5198e-02, -2.2584e-01],
  [ 5.9034e-02,  1.3105e-01]],

 [[ 8.8506e-02,  2.2061e-01],
  [-1.3456e-01, -1.0704e-01]]],


[[[-1.5140e-01, -8.7042e-02],
  [ 1.8381e-01, -1.5470e-01]],

 [[-2.1762e-01, -1.3052e-02],
  [-2.1713e-02, -1.2686e-01]],

 [[ 4.3823e-02,  1.2352e-01],
  [-1.2777e-01, -1.4085e-01]]],


[[[-2.0174e-01,  1.8290e-01],
  [ 2.5212e-01,  2.8757e-01]],
```

```
 [[ 2.6915e-02, -1.7166e-01],
  [-2.2028e-01, -4.0050e-03]],

 [[ 1.4545e-01, -3.1423e-02],
  [-2.6597e-01, -2.6357e-01]]],


[[[-2.0283e-01, -7.1709e-03],
  [-2.7133e-02,  1.8765e-01]],

 [[ 1.2536e-01,  4.8427e-02],
  [-9.4222e-02,  2.2514e-01]],

 [[ 1.3544e-01, -2.4359e-01],
  [ 1.7904e-01, -1.9357e-01]]],


[[[-2.3608e-01, -5.6042e-02],
  [-2.7965e-01, -9.9222e-02]],

 [[ 1.6958e-01,  1.8307e-01],
  [-4.6652e-03, -1.8752e-01]],

 [[ 2.2848e-01,  2.6443e-01],
  [-1.9063e-01,  6.9474e-03]]],


[[[-2.1291e-01, -1.3518e-01],
  [ 1.2607e-01, -1.8231e-01]],

 [[ 1.4098e-01, -8.0658e-02],
  [ 1.9839e-01, -1.5396e-01]],

 [[ 6.1994e-02, -1.8228e-01],
  [ 1.2293e-01, -8.0927e-02]]],


[[[ 1.6856e-01,  1.0812e-01],
  [-4.2583e-02, -1.4939e-01]],

 [[-1.8132e-01,  6.6400e-02],
  [ 1.4434e-01,  3.4804e-02]],

 [[-1.9545e-01, -2.1538e-01],
  [ 3.1109e-02, -2.7817e-03]]],


[[[ 1.9476e-01,  1.6451e-01],
```

```
   [ 1.1195e-01, -1.8705e-01]],

  [[ 9.2237e-02,  2.8184e-01],
   [ 1.6060e-01,  1.5068e-01]],

  [[-1.8709e-01,  2.1163e-01],
   [ 1.0004e-01, -1.1521e-01]]],


[[[-4.3349e-02, -2.7923e-01],
   [-7.0851e-02, -2.3322e-01]],

  [[-2.5382e-01,  2.5598e-01],
   [ 2.3160e-01,  1.6760e-01]],

  [[-2.4309e-01, -1.4582e-01],
   [-9.0942e-03, -1.2669e-01]]],


[[[-1.9262e-01,  7.4707e-02],
   [-2.5990e-01,  2.7291e-01]],

  [[-9.0592e-03,  4.2963e-02],
   [ 1.5653e-01, -1.1706e-01]],

  [[-2.4442e-01,  1.5953e-01],
   [ 6.6715e-02, -1.4256e-01]]],


[[[-1.0135e-01, -1.5981e-02],
   [-1.6047e-01,  3.3011e-02]],

  [[-1.7852e-01, -1.5567e-01],
   [-2.2787e-01,  9.6782e-02]],

  [[-1.5605e-01,  1.5343e-01],
   [-1.3718e-01, -2.8206e-01]]],


[[[ 2.7258e-01,  2.5404e-01],
   [ 1.7902e-01, -1.0906e-01]],

  [[-7.8910e-03, -1.3714e-01],
   [-2.7044e-01, -4.5800e-02]],

  [[ 2.5189e-01, -1.1355e-01],
   [ 1.6623e-01, -1.5228e-01]]],
```

```
        [[[ 1.2579e-01,  1.0626e-01],
          [-1.4654e-01, -2.1371e-01]],

         [[ 2.2543e-01, -4.2494e-02],
          [ 4.6772e-02,  1.0567e-01]],

         [[ 2.0623e-01, -1.3304e-01],
          [-8.9317e-02, -6.8539e-02]]],


        [[[-1.6130e-01, -2.5655e-01],
          [-9.5789e-02, -2.5457e-01]],

         [[-9.1106e-02,  2.0298e-01],
          [ 6.5842e-02,  2.7852e-01]],

         [[-4.5266e-02,  1.7823e-01],
          [ 4.2380e-02, -3.0429e-02]]],


        [[[ 4.9580e-02,  8.8007e-03],
          [ 8.2871e-02, -2.2294e-01]],

         [[-2.1053e-01, -7.7271e-02],
          [ 1.5591e-01,  1.0724e-01]],

         [[ 4.9694e-02, -1.7419e-01],
          [ 1.7130e-01, -6.8518e-02]]]], requires_grad=True), Parameter
containing:
tensor([-0.0531, -0.0504, -0.2108,  0.0900, -0.0814,  0.0250, -0.2178, -0.0996,
        -0.0092, -0.0616,  0.1462,  0.0564, -0.0808,  0.2129,  0.0332,  0.0256,
         0.0877, -0.2677,  0.2567,  0.0722,  0.2111, -0.1957, -0.0385, -0.1379,
        -0.2726,  0.2358,  0.0549,  0.0798, -0.2663, -0.1065, -0.2588, -0.1908],
       requires_grad=True), Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.], requires_grad=True), Parameter
containing:
tensor([[[[-1.8821e-03, -2.6503e-02, -1.4605e-02, -1.1147e-02],
          [-4.4179e-02, -1.2621e-02,  3.5566e-02,  9.7917e-03],
          [-3.8456e-03, -3.3261e-02,  3.0807e-03, -1.8286e-02],
          [ 5.7011e-03,  3.7161e-02, -3.1304e-02,  1.4198e-02]],

         [[-2.2320e-03, -2.2274e-02, -3.0260e-02, -3.7559e-02],
```

```
  [ 1.2292e-02, -8.2912e-03, -4.4935e-03,  4.0923e-02],
  [-7.9708e-03, -2.4131e-03, -1.4872e-02,  3.9768e-02],
  [ 6.3704e-03,  4.1147e-02,  1.3146e-02,  8.1047e-03]],

 [[ 1.4908e-02,  1.4298e-02,  1.8941e-02,  1.7881e-02],
  [-1.8374e-02,  7.9587e-03,  3.2842e-02,  2.3291e-02],
  [ 1.1876e-02, -2.6629e-02,  2.6711e-03,  4.4016e-02],
  [ 2.9481e-02,  4.1546e-03, -1.1571e-02,  8.2745e-03]],

 …,

 [[-4.0160e-02,  4.3214e-02,  3.5191e-02, -4.0726e-02],
  [-1.5793e-02, -4.3340e-02,  9.0029e-03,  1.0825e-02],
  [-8.4129e-03,  2.6540e-02,  2.4381e-02,  3.1115e-02],
  [ 3.8425e-02,  2.0936e-02, -2.2889e-03,  1.0093e-02]],

 [[-1.0760e-02, -3.4893e-02, -2.1125e-02,  2.1105e-02],
  [ 3.4920e-02, -3.9565e-02,  3.7951e-03, -1.4339e-04],
  [ 4.2798e-02, -1.7031e-02, -2.3830e-02, -8.9394e-03],
  [ 2.9233e-02, -4.1438e-02, -1.8542e-02,  2.5852e-02]],

 [[ 3.5524e-02,  1.8349e-02,  2.6059e-02,  3.5193e-02],
  [ 3.5271e-02, -3.3437e-02, -1.0150e-02, -4.0052e-02],
  [ 4.3020e-02,  9.3571e-03, -2.3200e-02,  3.4985e-02],
  [ 7.0088e-04,  3.0139e-02, -4.7072e-03,  2.4650e-02]]],


[[[ 2.5779e-02, -3.5073e-03, -2.3780e-02, -1.5004e-02],
  [ 2.7978e-02,  3.6263e-02,  3.9753e-02, -2.6761e-02],
  [-2.6799e-02, -3.5306e-02,  1.0705e-02, -1.9302e-02],
  [ 8.1113e-03,  3.8871e-02,  3.9333e-02,  2.4703e-02]],

 [[-3.3475e-02,  3.8349e-03,  1.3924e-02, -3.1114e-04],
  [-7.7809e-03,  1.2071e-02, -3.4567e-02,  9.9812e-03],
  [-2.5696e-02,  6.0663e-03, -2.2994e-02,  3.6700e-02],
  [ 4.3189e-02,  2.8507e-02, -2.8797e-03,  9.6976e-04]],

 [[ 3.9534e-02, -1.2386e-02, -3.7731e-02, -8.1753e-03],
  [ 3.7966e-02, -2.6073e-02, -3.4783e-02,  2.6277e-03],
  [-2.9951e-02,  2.0356e-03,  4.1576e-02,  3.7215e-02],
  [ 8.2428e-03, -7.5800e-03, -1.5049e-02, -4.4231e-03]],

 …,

 [[-3.3264e-02, -2.6514e-02, -4.2748e-02,  2.2853e-02],
  [ 1.3812e-02,  3.3721e-03,  3.9921e-02,  2.9521e-02],
  [-9.1015e-03, -1.8394e-02, -2.4041e-02,  3.4095e-02],
  [-1.0337e-02, -4.3500e-02, -2.1646e-02,  4.2764e-02]],
```

```
 [[-3.2199e-02, -5.6661e-03,  2.4847e-03, -4.7200e-04],
  [ 1.0435e-02, -3.3871e-02, -1.6805e-03, -1.6118e-02],
  [ 3.0614e-02, -1.7132e-02, -2.9147e-02, -1.2166e-03],
  [ 1.7846e-02,  4.4077e-02,  4.1343e-02, -1.5975e-02]],

 [[-1.6011e-02,  2.4876e-02, -8.6224e-03,  1.3252e-02],
  [ 2.1853e-02, -4.0665e-02,  1.0038e-02, -3.3669e-02],
  [-3.4468e-02, -1.2963e-02,  3.9079e-02,  8.1945e-03],
  [ 3.8276e-02,  3.1483e-02, -2.0975e-02, -3.6671e-02]]],


[[[ 2.2293e-02,  3.8807e-02,  3.4638e-02,  3.3681e-02],
  [-3.9232e-02, -3.6450e-02,  2.8204e-02,  9.3512e-03],
  [-1.1353e-03, -1.9759e-02, -4.1681e-02, -4.2930e-02],
  [-2.3496e-02,  3.3512e-02,  1.7050e-02, -9.7387e-03]],

 [[-3.5777e-02, -1.7874e-02,  7.1799e-03, -3.1641e-02],
  [-2.4128e-02, -3.3190e-02, -3.6361e-02,  7.2805e-03],
  [-3.8583e-02,  3.5761e-02,  2.1184e-02, -2.6158e-02],
  [ 2.1530e-02, -1.4546e-02, -5.6179e-03,  3.6653e-02]],

 [[-2.3061e-03,  9.6271e-03, -3.3166e-02, -2.9849e-02],
  [ 3.6056e-02, -1.9608e-02, -1.1501e-02, -1.5256e-02],
  [-3.0135e-02, -2.5017e-03,  4.8839e-03,  1.5415e-02],
  [ 1.6104e-02,  4.5368e-03, -2.8377e-02,  3.8510e-02]],

 …,

 [[ 2.1365e-02, -1.5487e-02,  2.7149e-02,  3.0793e-02],
  [ 3.9325e-02, -2.7265e-03,  9.7416e-03, -4.3246e-04],
  [ 4.3651e-02, -9.7338e-04,  6.0543e-04,  1.3959e-02],
  [ 3.2616e-02, -3.6341e-02, -1.8818e-02, -3.0872e-02]],

 [[-4.3870e-02, -7.0506e-03, -1.1327e-02,  2.5699e-02],
  [-4.4092e-02, -4.0536e-03, -2.3633e-02, -5.7115e-03],
  [ 4.2301e-02,  3.1068e-02, -1.8547e-02,  2.1884e-02],
  [ 6.2363e-03, -1.0552e-02,  2.7235e-02,  3.6391e-02]],

 [[ 3.7304e-02, -9.4773e-03, -4.3119e-02,  2.4861e-02],
  [-1.9702e-02, -2.8824e-02,  2.1394e-02,  3.5128e-02],
  [-2.4368e-02,  3.3196e-02, -3.8285e-02,  1.5408e-02],
  [-2.3132e-02,  2.9357e-03,  1.5180e-02, -4.3116e-02]]],


…,
```

```
[[[-2.7558e-02,  2.9652e-02, -3.6566e-02, -7.4468e-03],
  [ 7.0401e-03,  3.7015e-02, -4.4187e-02,  1.0558e-03],
  [ 2.3466e-03,  4.2496e-02,  2.0574e-02,  2.1758e-02],
  [ 1.2174e-03, -1.6117e-02,  1.3316e-02,  9.5944e-03]],

 [[ 1.5543e-03, -3.7210e-02,  1.7885e-02,  1.6541e-02],
  [ 3.7919e-02,  3.7750e-02, -3.2786e-02, -3.8893e-02],
  [-1.3230e-02, -2.3674e-02, -3.0933e-02,  1.6964e-02],
  [-5.6389e-03, -4.2468e-02,  3.8340e-02, -1.2611e-03]],

 [[-2.8050e-02,  3.0551e-02,  4.7398e-03,  3.1007e-03],
  [ 3.0581e-02,  3.6264e-03, -1.0506e-02, -2.4749e-02],
  [ 2.5384e-02,  2.1755e-02,  1.0290e-02, -3.5370e-02],
  [-2.8222e-02,  1.4070e-02, -3.3722e-02, -2.9810e-02]],

 …,

 [[ 3.0115e-02,  4.1796e-02, -3.9204e-02,  2.5803e-02],
  [ 1.9174e-02,  1.6647e-02, -9.6432e-03,  7.0725e-03],
  [ 2.4131e-02,  1.0208e-02, -3.3396e-02,  9.0159e-03],
  [-4.1594e-02, -6.3820e-03,  6.8906e-03, -2.0750e-02]],

 [[ 1.1418e-02, -2.6633e-02,  2.6686e-02,  3.7205e-02],
  [ 2.8501e-02, -1.3784e-02,  6.5233e-03, -3.2934e-02],
  [-2.1707e-02, -4.0225e-02, -2.1609e-02, -3.0288e-02],
  [-2.6565e-02, -2.3614e-02,  3.8589e-02,  2.1033e-02]],

 [[ 8.2600e-03,  2.1860e-03, -1.4169e-02, -9.8505e-03],
  [ 2.5326e-02,  1.5508e-03,  8.9214e-03, -1.8561e-02],
  [ 2.3431e-02, -1.3489e-02,  5.6862e-03,  4.7615e-03],
  [ 6.0466e-03,  3.2949e-02,  1.9107e-02,  3.8783e-02]]],


[[[-2.8833e-02,  1.6196e-02,  3.4690e-02,  5.0575e-03],
  [ 2.9514e-02, -1.0021e-02,  5.7561e-03,  5.1865e-04],
  [-3.8744e-02, -3.1062e-03, -2.8801e-02, -3.7845e-02],
  [-3.2894e-02, -1.8415e-02,  1.6492e-02,  8.9802e-03]],

 [[ 4.3694e-03, -8.5728e-03,  1.4606e-02, -3.9464e-02],
  [ 1.1575e-02,  1.8579e-02,  1.4812e-02,  4.1376e-02],
  [ 2.2529e-02,  2.5798e-02, -2.9106e-02, -2.8786e-02],
  [ 1.8064e-02, -3.9078e-02,  2.9468e-02,  2.5870e-02]],

 [[-2.8994e-02, -3.9664e-02,  6.7804e-03,  3.6112e-02],
  [ 3.7025e-02,  1.3950e-02,  2.2106e-02, -3.9447e-02],
  [-4.3603e-02, -8.1671e-04,  6.9148e-03,  3.9349e-02],
  [-2.6072e-02, -4.2814e-02,  7.1259e-03,  1.2973e-02]],
```

```
…,

 [[ 4.0241e-02, -2.4249e-02,  3.9319e-02, -1.5380e-02],
  [ 3.0221e-02,  1.5771e-02,  3.9111e-02,  2.7271e-02],
  [-3.0093e-02, -3.9932e-02, -1.1610e-02, -3.2664e-02],
  [-1.4736e-02,  3.6823e-03, -7.9107e-03,  4.3803e-02]],

 [[-4.1503e-02, -1.3731e-02,  4.0309e-02,  4.4953e-03],
  [-2.4680e-02,  2.5947e-02,  3.9358e-02, -3.4756e-02],
  [-4.3580e-02,  2.0161e-02,  4.1084e-02,  3.0454e-02],
  [ 2.3876e-03,  1.1666e-02,  4.3246e-03,  3.7567e-02]],

 [[ 6.6418e-03,  1.5001e-02,  1.1115e-02, -1.1592e-02],
  [ 2.9918e-04,  3.1330e-02, -1.2373e-03,  2.5205e-02],
  [-5.6090e-04, -1.7289e-02, -3.0567e-02,  2.5893e-03],
  [-1.7045e-02,  3.6678e-02,  2.5376e-03, -1.4118e-02]]],


[[[-6.6070e-03, -5.5165e-05,  1.2819e-02, -1.6554e-02],
  [ 3.6802e-02,  1.8771e-02, -3.4272e-02, -2.3605e-02],
  [ 2.6347e-02, -5.9776e-03, -7.4485e-03, -7.4394e-03],
  [ 3.4274e-02,  3.3746e-02,  3.8357e-02, -1.8506e-02]],

 [[ 4.0224e-02, -2.1919e-03, -2.8842e-02,  3.6720e-02],
  [-2.7114e-02, -3.2736e-02,  2.1980e-02,  1.5536e-03],
  [ 1.1152e-02, -2.4586e-02,  4.2935e-02,  1.8182e-02],
  [ 3.4576e-02,  2.9374e-02, -3.2962e-02, -2.6529e-02]],

 [[ 6.8270e-03, -1.0075e-03, -1.3838e-02,  2.7986e-02],
  [-2.1465e-02, -2.5866e-02,  2.0787e-03, -3.9265e-02],
  [ 2.1506e-02, -4.2432e-02, -2.2735e-02,  9.8403e-04],
  [-1.6872e-02,  4.2932e-03, -2.1585e-02,  3.3299e-03]],

 …,

 [[ 4.1811e-02,  3.2271e-02, -9.6462e-03, -2.7675e-02],
  [-8.4016e-04, -2.4699e-03, -1.5867e-02,  4.2870e-02],
  [-2.1763e-02,  3.5990e-02, -1.6936e-02, -8.3683e-03],
  [-1.3952e-02,  4.2594e-02,  3.0001e-02,  1.8287e-02]],

 [[ 2.2226e-02, -1.3275e-03,  2.2636e-02, -2.3606e-02],
  [-1.9255e-02,  1.3936e-02, -1.9148e-02, -2.4174e-02],
  [-3.0222e-02, -7.3415e-03, -3.0313e-02,  5.0275e-04],
  [-2.1575e-02, -2.5969e-02,  3.5347e-02, -1.2657e-04]],

 [[-3.1929e-02,  2.8433e-03, -3.6847e-02,  4.4070e-02],
  [-1.4485e-02,  3.5303e-02,  4.1365e-02,  3.0136e-02],
  [-2.7794e-02,  3.2234e-02,  8.4246e-03, -4.7168e-03],
```

```
          [ 1.1879e-02, -2.0200e-02, -1.1834e-03, -3.2851e-02]]]],
       requires_grad=True), Parameter containing:
tensor([-0.0042, -0.0433, -0.0228, -0.0375, -0.0166, -0.0137,  0.0043, -0.0016,
         0.0299, -0.0056,  0.0437, -0.0050, -0.0318,  0.0006,  0.0143,  0.0240,
        -0.0298, -0.0373, -0.0024,  0.0247, -0.0047, -0.0223, -0.0149, -0.0269,
         0.0315, -0.0181, -0.0272, -0.0400,  0.0433, -0.0103,  0.0263, -0.0152],
       requires_grad=True), Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.], requires_grad=True), Parameter
containing:
tensor([[[[-0.0491,  0.0383],
          [-0.0774, -0.0269]],

         [[ 0.0558, -0.0473],
          [-0.0298,  0.0117]],

         [[-0.0198, -0.0142],
          [-0.0305, -0.0248]],

         …,

         [[-0.0743,  0.0710],
          [-0.0033,  0.0211]],

         [[ 0.0585, -0.0783],
          [ 0.0708, -0.0363]],

         [[-0.0073, -0.0693],
          [-0.0422, -0.0274]]],


        [[[ 0.0732, -0.0297],
          [-0.0174, -0.0371]],

         [[-0.0576,  0.0037],
          [-0.0752,  0.0446]],

         [[ 0.0613, -0.0884],
          [ 0.0740,  0.0328]],

         …,

         [[-0.0781, -0.0098],
          [-0.0761, -0.0442]],
```

```
 [[-0.0207, -0.0471],
  [-0.0288,  0.0243]],

 [[-0.0485,  0.0699],
  [-0.0861, -0.0582]]],


[[[ 0.0279, -0.0029],
  [-0.0402, -0.0643]],

 [[-0.0703, -0.0646],
  [-0.0310,  0.0110]],

 [[ 0.0008,  0.0034],
  [ 0.0133,  0.0874]],

 …,

 [[ 0.0148,  0.0431],
  [-0.0291,  0.0161]],

 [[ 0.0041, -0.0538],
  [-0.0635,  0.0516]],

 [[ 0.0025, -0.0287],
  [ 0.0114, -0.0826]]],


…,


[[[-0.0709, -0.0635],
  [-0.0583, -0.0709]],

 [[ 0.0474,  0.0723],
  [ 0.0628,  0.0471]],

 [[ 0.0396,  0.0851],
  [-0.0213, -0.0207]],

 …,

 [[-0.0676,  0.0263],
  [ 0.0135, -0.0059]],

 [[ 0.0825, -0.0018],
  [-0.0809,  0.0212]],
```

```
          [[ 0.0569, -0.0625],
           [ 0.0447,  0.0003]]],


         [[[-0.0382, -0.0505],
           [ 0.0814, -0.0810]],

          [[ 0.0216, -0.0858],
           [-0.0877,  0.0697]],

          [[-0.0653,  0.0040],
           [ 0.0627,  0.0719]],

          …,

          [[-0.0374, -0.0619],
           [ 0.0445,  0.0572]],

          [[-0.0611,  0.0542],
           [-0.0805, -0.0594]],

          [[ 0.0572, -0.0261],
           [ 0.0069,  0.0876]]],


         [[[-0.0662, -0.0327],
           [-0.0620, -0.0848]],

          [[ 0.0600,  0.0298],
           [ 0.0123,  0.0856]],

          [[-0.0442,  0.0755],
           [ 0.0676, -0.0704]],

          …,

          [[ 0.0515,  0.0488],
           [ 0.0166, -0.0157]],

          [[-0.0225,  0.0625],
           [-0.0046, -0.0614]],

          [[ 0.0729,  0.0858],
           [-0.0705,  0.0084]]]], requires_grad=True), Parameter containing:
tensor([-0.0433,  0.0724,  0.0668, -0.0426, -0.0225,  0.0478, -0.0668,  0.0240,
        -0.0324, -0.0628, -0.0444,  0.0542,  0.0696,  0.0170, -0.0550,  0.0432],
       requires_grad=True), Parameter containing:
```

```
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       requires_grad=True), Parameter containing:
tensor([[[[ 0.0920, -0.0141],
          [-0.0002,  0.1142]],

         [[ 0.0978,  0.0426],
          [-0.1034,  0.0326]],

         [[-0.0218,  0.1014],
          [-0.0388,  0.0764]],

         [[ 0.0364,  0.0307],
          [ 0.1028, -0.0171]],

         [[-0.0443,  0.1244],
          [-0.1220, -0.0173]],

         [[ 0.0199, -0.1139],
          [-0.1153,  0.0392]],

         [[-0.0671, -0.0281],
          [ 0.1133,  0.0624]],

         [[-0.0877, -0.0288],
          [ 0.0133,  0.0054]],

         [[-0.0483, -0.0793],
          [ 0.0021,  0.0099]],

         [[-0.0997, -0.0436],
          [-0.0764, -0.0513]],

         [[ 0.0733,  0.0800],
          [-0.0892, -0.0692]],

         [[ 0.0071, -0.0313],
          [-0.0482, -0.0122]],

         [[-0.0470,  0.0772],
          [ 0.0524, -0.0664]],

         [[ 0.0839,  0.1000],
          [ 0.1126,  0.0281]],

         [[-0.1235, -0.0217],
          [-0.0930, -0.1038]],
```

```
 [[ 0.0713,  0.1143],
  [-0.0167, -0.0514]]],


[[[-0.0364, -0.0562],
  [ 0.0537, -0.0290]],

 [[ 0.1080,  0.0569],
  [ 0.0350, -0.0127]],

 [[-0.0477,  0.0534],
  [-0.0487,  0.0979]],

 [[-0.0284, -0.0928],
  [ 0.0140,  0.0256]],

 [[ 0.1249,  0.0551],
  [-0.1195, -0.0502]],

 [[ 0.0838, -0.0820],
  [ 0.0162, -0.0099]],

 [[ 0.0844,  0.0154],
  [ 0.0240,  0.0394]],

 [[-0.0898,  0.0594],
  [-0.0330, -0.0483]],

 [[-0.0221, -0.0102],
  [ 0.0648,  0.0740]],

 [[-0.0899, -0.0847],
  [ 0.0796, -0.0879]],

 [[-0.0340, -0.0032],
  [-0.0332,  0.0264]],

 [[ 0.0817,  0.0061],
  [-0.0760, -0.1087]],

 [[-0.0155, -0.0552],
  [ 0.0206,  0.0962]],

 [[ 0.1121, -0.0595],
  [-0.0732, -0.0277]],

 [[-0.1211, -0.0657],
```

```
   [ 0.0405, -0.1238]],

  [[-0.0567,  0.0360],
   [-0.0537,  0.0829]]],


[[[-0.0811, -0.0972],
   [-0.0329, -0.0900]],

  [[ 0.0224, -0.0119],
   [-0.0304, -0.0847]],

  [[-0.0262, -0.0070],
   [ 0.0270, -0.0191]],

  [[-0.0200, -0.1148],
   [-0.0734,  0.0374]],

  [[ 0.1197, -0.0690],
   [ 0.1020,  0.0031]],

  [[-0.0949,  0.0613],
   [ 0.0613,  0.0556]],

  [[-0.0798, -0.0711],
   [-0.0554,  0.0732]],

  [[-0.0417, -0.0129],
   [ 0.0588, -0.0898]],

  [[-0.0052,  0.1018],
   [ 0.0219,  0.0461]],

  [[ 0.0557,  0.1167],
   [-0.1091,  0.0291]],

  [[-0.0194, -0.0649],
   [ 0.1220, -0.0931]],

  [[-0.0324,  0.0565],
   [-0.0983,  0.0652]],

  [[-0.1082, -0.0331],
   [ 0.1033,  0.0810]],

  [[-0.0302, -0.0511],
   [-0.0033,  0.0640]],
```

```
 [[-0.0915,  0.0729],
  [-0.0018, -0.0222]],

 [[ 0.0166,  0.0516],
  [-0.0789,  0.0306]]],


[[[ 0.0667, -0.0305],
  [ 0.0417, -0.1027]],

 [[ 0.1136,  0.0431],
  [ 0.0547, -0.0753]],

 [[ 0.0631, -0.1193],
  [ 0.0045,  0.1245]],

 [[ 0.0405,  0.0315],
  [-0.0746,  0.0031]],

 [[-0.0754, -0.1227],
  [ 0.0884, -0.0034]],

 [[ 0.0747,  0.0789],
  [ 0.0034, -0.0467]],

 [[ 0.0355, -0.1016],
  [-0.1126, -0.0923]],

 [[ 0.1210,  0.0412],
  [ 0.0033,  0.1173]],

 [[-0.0736, -0.1150],
  [ 0.1046,  0.0564]],

 [[ 0.0876,  0.0422],
  [ 0.0555, -0.0213]],

 [[ 0.0809, -0.0919],
  [ 0.0173,  0.0974]],

 [[ 0.1139,  0.0660],
  [-0.0055,  0.0169]],

 [[-0.0868, -0.0484],
  [ 0.0778, -0.0177]],

 [[-0.0891, -0.1247],
  [ 0.0119, -0.0715]],
```

```
 [[-0.0656,  0.1051],
  [-0.0659, -0.0726]],

 [[ 0.0233, -0.0708],
  [-0.1159, -0.1183]]],


[[[ 0.0893,  0.0819],
  [-0.1065,  0.0317]],

 [[-0.0094, -0.1124],
  [ 0.0750, -0.1013]],

 [[-0.0301,  0.0713],
  [-0.0309,  0.0433]],

 [[ 0.0721, -0.0851],
  [ 0.0233, -0.1157]],

 [[-0.1098, -0.0669],
  [-0.0772, -0.0496]],

 [[-0.0066,  0.0560],
  [ 0.1040, -0.1040]],

 [[-0.0186, -0.0783],
  [ 0.0014,  0.1028]],

 [[-0.0041, -0.0916],
  [ 0.0201, -0.1206]],

 [[ 0.0084,  0.0025],
  [ 0.0838, -0.0537]],

 [[-0.0963,  0.0813],
  [ 0.1058,  0.0695]],

 [[-0.0378,  0.0817],
  [ 0.0745, -0.1059]],

 [[-0.0624,  0.0739],
  [-0.0617, -0.0287]],

 [[-0.0080, -0.0092],
  [ 0.0557, -0.1059]],

 [[ 0.0687,  0.0635],
```

```
   [-0.0178,  0.0223]],

  [[ 0.0604,  0.1172],
   [ 0.0266, -0.1099]],

  [[ 0.0977, -0.1062],
   [-0.0036, -0.0555]]],


 [[[ 0.0574, -0.1095],
   [-0.0155, -0.0921]],

  [[-0.0522, -0.0826],
   [ 0.0120, -0.0643]],

  [[-0.1029, -0.0479],
   [-0.0657,  0.0286]],

  [[ 0.0791,  0.0018],
   [-0.0957,  0.0840]],

  [[-0.0737, -0.0551],
   [-0.0055,  0.0164]],

  [[-0.0074,  0.0767],
   [ 0.0075,  0.0755]],

  [[ 0.0266, -0.0167],
   [ 0.0912, -0.0733]],

  [[-0.0235,  0.1086],
   [-0.0732, -0.0199]],

  [[ 0.0162, -0.1048],
   [ 0.0851, -0.0853]],

  [[ 0.0568,  0.0060],
   [-0.0927, -0.0094]],

  [[ 0.0531,  0.0126],
   [ 0.0594, -0.0040]],

  [[-0.0316,  0.0177],
   [ 0.0915, -0.0287]],

  [[ 0.0932, -0.0092],
   [ 0.0643,  0.0855]],
```

```
 [[ 0.0686,  0.0664],
  [ 0.1153, -0.0850]],

 [[-0.0610, -0.0388],
  [ 0.0545,  0.0426]],

 [[ 0.0984, -0.0987],
  [ 0.0384,  0.0060]]],


[[[-0.0235,  0.0470],
  [ 0.0439,  0.0397]],

 [[-0.0089,  0.0608],
  [-0.0563,  0.1188]],

 [[ 0.0349, -0.0981],
  [ 0.1030, -0.0278]],

 [[-0.0686,  0.0994],
  [-0.0128, -0.1169]],

 [[-0.0822,  0.0931],
  [-0.0360,  0.0071]],

 [[ 0.0181, -0.0757],
  [-0.0816, -0.0766]],

 [[ 0.1005, -0.0678],
  [-0.0545, -0.0928]],

 [[ 0.0526, -0.0143],
  [-0.1201, -0.0050]],

 [[-0.1070, -0.0466],
  [ 0.0783, -0.0169]],

 [[ 0.0405,  0.1167],
  [-0.1086,  0.1066]],

 [[ 0.1061, -0.1076],
  [ 0.0065, -0.0970]],

 [[ 0.0732, -0.0442],
  [-0.0550,  0.1158]],

 [[-0.1073, -0.0961],
  [ 0.0583,  0.0406]],
```

```
 [[-0.0215, -0.0782],
  [-0.0259,  0.0023]],

 [[ 0.0124,  0.0494],
  [ 0.0344,  0.0212]],

 [[-0.0835, -0.0995],
  [-0.1003, -0.0734]]],


[[[ 0.0706,  0.0297],
  [-0.0355,  0.0817]],

 [[ 0.0393,  0.0551],
  [ 0.1248,  0.0284]],

 [[-0.0253,  0.0616],
  [ 0.0607, -0.1098]],

 [[ 0.0795,  0.0323],
  [ 0.1182, -0.0809]],

 [[-0.0777, -0.0393],
  [ 0.0620,  0.0689]],

 [[ 0.0867,  0.0530],
  [-0.1235,  0.1030]],

 [[ 0.0845, -0.0586],
  [ 0.0165, -0.0008]],

 [[ 0.0321, -0.1242],
  [ 0.0552, -0.0123]],

 [[-0.0162, -0.0435],
  [ 0.0607,  0.0548]],

 [[-0.0842,  0.0055],
  [-0.0508,  0.1180]],

 [[-0.0818,  0.0193],
  [ 0.0588,  0.0087]],

 [[ 0.0131,  0.1222],
  [ 0.0413,  0.0198]],

 [[ 0.0076,  0.0863],
```

```
          [-0.1200,  0.0230]],

         [[ 0.0703, -0.0070],
          [-0.1171, -0.0445]],

         [[ 0.1069, -0.0389],
          [ 0.0413, -0.0197]],

         [[ 0.1000, -0.0844],
          [-0.0107,  0.0859]]]], requires_grad=True), Parameter containing:
tensor([-0.0425,  0.0606,  0.0307, -0.0505,  0.1075,  0.0153, -0.0777,  0.0429],
       requires_grad=True), Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1.], requires_grad=True), Parameter
containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0.], requires_grad=True), Parameter
containing:
tensor([[[[-0.1239, -0.0352],
          [ 0.1126,  0.0106]],

         [[ 0.0582, -0.0870],
          [ 0.0994,  0.0143]],

         [[ 0.0283, -0.0152],
          [ 0.0632,  0.0542]],

         [[-0.1178,  0.0831],
          [-0.1337,  0.0017]],

         [[ 0.1320,  0.0926],
          [-0.1362, -0.1180]],

         [[ 0.1466,  0.0107],
          [ 0.0587, -0.0234]],

         [[-0.0814,  0.0444],
          [-0.1296, -0.1608]],

         [[ 0.1047,  0.0511],
          [ 0.0835,  0.1240]]],


        [[[ 0.1440, -0.0846],
          [-0.1398, -0.0229]],

         [[-0.0405, -0.0957],
          [ 0.1701,  0.0623]],

         [[-0.0651, -0.1374],
```

```
  [-0.1696,  0.0014]],

 [[ 0.0105, -0.1269],
  [-0.0258, -0.1617]],

 [[ 0.1317, -0.1598],
  [-0.0302, -0.0241]],

 [[-0.1645,  0.1217],
  [ 0.1543,  0.1433]],

 [[ 0.1146,  0.1312],
  [ 0.0665, -0.1525]],

 [[-0.0067,  0.0873],
  [-0.1518, -0.0823]]],


[[[-0.1024, -0.1212],
  [ 0.1754, -0.0161]],

 [[ 0.1295,  0.0267],
  [ 0.1747, -0.1685]],

 [[ 0.1225, -0.1028],
  [ 0.0324, -0.0781]],

 [[-0.0017,  0.0039],
  [-0.1121,  0.1235]],

 [[-0.1337, -0.0239],
  [-0.0337,  0.0478]],

 [[-0.0075, -0.0696],
  [-0.0090,  0.0956]],

 [[ 0.0095,  0.1765],
  [-0.0046, -0.0013]],

 [[-0.0543,  0.1631],
  [-0.1583,  0.0581]]],


[[[ 0.0543,  0.0290],
  [ 0.0722,  0.1368]],

 [[-0.0974,  0.0886],
  [-0.1519,  0.0513]],
```

```
          [[ 0.1430,  0.1340],
           [ 0.1302, -0.1472]],

          [[-0.0009,  0.1479],
           [ 0.1691, -0.0541]],

          [[ 0.1326,  0.1179],
           [-0.1532,  0.0328]],

          [[-0.0442,  0.0092],
           [-0.1744,  0.0385]],

          [[ 0.0549,  0.1651],
           [ 0.1029, -0.1504]],

          [[-0.0797,  0.1561],
           [ 0.1639,  0.0340]]]], requires_grad=True), Parameter containing:
tensor([-0.0946, -0.1454, -0.1503, -0.0977], requires_grad=True), Parameter
containing:
tensor([1., 1., 1., 1.], requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0.], requires_grad=True), Parameter containing:
tensor([[[[-0.0260, -0.1938],
          [ 0.1492, -0.1937]],

          [[-0.1937, -0.2335],
           [-0.0790, -0.2050]],

          [[-0.1191,  0.0566],
           [ 0.1748, -0.1578]],

          [[ 0.0443,  0.1054],
           [-0.1397,  0.0068]]],


         [[[-0.0760,  0.2171],
           [ 0.0721,  0.0171]],

          [[-0.1652, -0.0693],
           [ 0.0912,  0.1203]],

          [[ 0.1186, -0.0949],
           [-0.1725,  0.2460]],

          [[-0.0707,  0.0737],
           [-0.0439, -0.0455]]],
```

```
         [[[ 0.1877, -0.1522],
           [ 0.0335,  0.1243]],

          [[-0.2161,  0.2186],
           [ 0.0470,  0.0101]],

          [[ 0.1779,  0.2365],
           [ 0.0642,  0.0538]],

          [[ 0.1631,  0.2415],
           [-0.0873,  0.0808]]],


         [[[-0.0464, -0.0822],
           [-0.0116,  0.1124]],

          [[ 0.1264,  0.1688],
           [-0.1984,  0.0143]],

          [[-0.1420,  0.1063],
           [ 0.2455, -0.2075]],

          [[-0.1444,  0.2029],
           [ 0.2134,  0.0022]]]], requires_grad=True), Parameter containing:
tensor([-0.1008,  0.1889,  0.2498, -0.2343], requires_grad=True), Parameter
containing:
tensor([1., 1., 1., 1.], requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0.], requires_grad=True), Parameter containing:
tensor([[[[-0.2468, -0.0179],
           [-0.1312, -0.1056]],

          [[-0.1961,  0.1454],
           [ 0.0656,  0.1806]],

          [[-0.0144, -0.0598],
           [-0.0484,  0.0911]],

          [[ 0.0329,  0.2083],
           [-0.0098,  0.0545]]],


         [[[ 0.0669,  0.1772],
           [ 0.0166, -0.1220]],

          [[-0.2018, -0.0592],
           [-0.2101,  0.2421]],

          [[ 0.0023, -0.0750],
```

```
  [ 0.0533,  0.1641]],

 [[-0.1024, -0.1197],
  [ 0.0189, -0.1066]]],


[[[-0.1069, -0.2034],
  [ 0.2436, -0.0571]],

 [[ 0.0420,  0.2361],
  [ 0.0843, -0.0043]],

 [[ 0.1187,  0.0571],
  [-0.0123,  0.1614]],

 [[-0.1846,  0.0731],
  [-0.2365, -0.0778]]],


[[[-0.0406, -0.1225],
  [ 0.2488, -0.1023]],

 [[ 0.0382, -0.1514],
  [ 0.2279,  0.1228]],

 [[ 0.2262,  0.2264],
  [ 0.1205,  0.1001]],

 [[ 0.1163,  0.0037],
  [-0.0667, -0.2498]]],


[[[ 0.2022, -0.2256],
  [ 0.2211,  0.0150]],

 [[-0.0745,  0.2323],
  [ 0.1020, -0.1251]],

 [[ 0.2078,  0.0309],
  [ 0.0876,  0.2294]],

 [[-0.1752, -0.2159],
  [-0.1269,  0.1105]]],


[[[ 0.2217,  0.0882],
  [ 0.0835, -0.0904]],
```

```
 [[ 0.1742,  0.1831],
  [-0.0198, -0.0144]],

 [[ 0.1564,  0.0672],
  [-0.2438, -0.0149]],

 [[-0.0629, -0.0589],
  [ 0.2281,  0.0221]]],


[[[-0.1930,  0.2462],
  [-0.1253, -0.2015]],

 [[-0.2261,  0.0336],
  [-0.0981, -0.2244]],

 [[ 0.2244,  0.0897],
  [ 0.1812, -0.1056]],

 [[ 0.0468,  0.0580],
  [ 0.2498, -0.1336]]],


[[[-0.0381,  0.2368],
  [-0.0915, -0.1547]],

 [[ 0.0419,  0.1763],
  [ 0.0672, -0.1973]],

 [[-0.0728, -0.2054],
  [ 0.1223,  0.0063]],

 [[ 0.0017,  0.1819],
  [-0.2331,  0.1266]]],


[[[-0.0580,  0.0524],
  [-0.1952,  0.0700]],

 [[-0.1865, -0.1874],
  [ 0.0739,  0.0838]],

 [[-0.2275,  0.0898],
  [ 0.1531, -0.0221]],

 [[ 0.0145,  0.1056],
  [-0.1403,  0.1891]]],
```

```
[[[ 0.1517,  0.0975],
  [-0.1995,  0.1280]],

 [[-0.2208,  0.1931],
  [-0.0486,  0.2003]],

 [[-0.1006, -0.0337],
  [-0.1611,  0.0694]],

 [[ 0.0638, -0.0690],
  [ 0.0539,  0.1127]]],


[[[ 0.1240,  0.0173],
  [ 0.2406,  0.0880]],

 [[-0.1694, -0.0928],
  [ 0.1042,  0.0026]],

 [[ 0.1895, -0.1462],
  [-0.1147,  0.2277]],

 [[ 0.0930,  0.0809],
  [-0.2189, -0.1731]]],


[[[-0.1293,  0.1637],
  [ 0.0361,  0.2496]],

 [[ 0.0270, -0.2194],
  [-0.2376, -0.0140]],

 [[ 0.0299, -0.1374],
  [ 0.1165, -0.0853]],

 [[ 0.1628,  0.1057],
  [ 0.0685,  0.0710]]],


[[[ 0.2311,  0.0298],
  [ 0.1745, -0.1091]],

 [[-0.2440,  0.1742],
  [-0.0782,  0.1282]],

 [[-0.1949, -0.1376],
  [ 0.0006, -0.1126]],
```

```
          [[-0.1061,  0.0230],
           [-0.0938,  0.1393]]],


         [[[-0.0883, -0.2143],
           [ 0.0252,  0.2117]],

          [[ 0.0179, -0.0074],
           [-0.1026, -0.1070]],

          [[-0.0488, -0.2463],
           [ 0.0623, -0.2159]],

          [[ 0.1133, -0.2122],
           [-0.1411,  0.0975]]],


         [[[-0.1297,  0.2443],
           [-0.0795, -0.0198]],

          [[-0.0014, -0.0174],
           [ 0.0297, -0.1768]],

          [[ 0.1411,  0.2431],
           [-0.1526,  0.2026]],

          [[ 0.0358,  0.0790],
           [-0.2351,  0.0078]]],


         [[[ 0.0891, -0.1433],
           [-0.2287, -0.2208]],

          [[ 0.1590, -0.2327],
           [-0.0806,  0.0681]],

          [[ 0.1365, -0.1808],
           [ 0.1257,  0.0140]],

          [[ 0.0243, -0.1252],
           [ 0.1053,  0.0787]]]], requires_grad=True), Parameter containing:
tensor([-0.1928,  0.1554, -0.2456,  0.1150,  0.0580,  0.2027, -0.0175,  0.1295,
        -0.1584, -0.0337,  0.2052,  0.1824, -0.1235,  0.0302, -0.0593,  0.1607],
       requires_grad=True), Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
```

```
       requires_grad=True), Parameter containing:
tensor([[[[ 1.1280e-01, -5.2593e-02],
          [-1.1609e-01, -5.8777e-02]],

         [[ 5.2258e-02, -1.0155e-04],
          [ 1.1952e-01,  9.3650e-02]],

         [[-4.4220e-02, -7.6557e-02],
          [-5.1574e-02,  7.6384e-02]],

         …,

         [[ 7.2023e-02, -6.2295e-02],
          [-7.7447e-02, -1.0506e-01]],

         [[ 6.9390e-02, -8.0843e-02],
          [-9.0433e-02,  8.5940e-02]],

         [[ 6.5370e-02,  7.7417e-02],
          [-6.2363e-02,  4.1309e-02]]],


        [[[-1.1692e-01,  7.7472e-02],
          [-7.7869e-03,  4.5505e-02]],

         [[-2.6251e-03, -7.3469e-02],
          [-4.8637e-02,  7.4574e-02]],

         [[-3.4029e-02, -6.6289e-02],
          [-6.6113e-02, -1.1643e-01]],

         …,

         [[-1.0676e-01,  7.4640e-02],
          [-1.1241e-01, -7.4916e-02]],

         [[-4.7767e-02, -1.1577e-01],
          [-6.2865e-02, -1.1264e-01]],

         [[ 2.3347e-02, -1.4986e-02],
          [-1.4783e-02,  2.9482e-03]]],


        [[[-4.4619e-02, -1.0601e-01],
          [ 4.3669e-02, -1.0844e-01]],

         [[ 5.6741e-02,  3.9506e-02],
          [-8.2560e-02,  3.7844e-02]],
```

```
 [[-3.5769e-03,  1.7048e-02],
  [-5.9970e-02, -4.8634e-02]],

 …,

 [[-1.1612e-01, -1.0835e-01],
  [-4.3397e-02,  4.9614e-02]],

 [[ 4.8310e-02,  1.6811e-02],
  [-1.0129e-01,  2.1608e-02]],

 [[ 1.0677e-01,  6.4486e-02],
  [ 5.2116e-02,  6.0106e-02]]],


…,


[[[ 4.5496e-02, -5.6733e-02],
  [-7.1043e-02,  3.9830e-02]],

 [[-4.2695e-02,  1.0915e-01],
  [-4.0224e-02, -1.1946e-01]],

 [[ 7.5964e-02, -6.5788e-02],
  [-1.7758e-02,  3.7943e-02]],

 …,

 [[ 6.2899e-02, -4.8216e-02],
  [ 6.7866e-02,  4.9915e-02]],

 [[ 1.7982e-02,  5.0470e-02],
  [ 1.1583e-01,  1.1016e-01]],

 [[ 7.4664e-02, -6.8089e-02],
  [-3.1619e-02, -4.8846e-02]]],


[[[-1.0006e-01,  4.9130e-02],
  [ 3.8260e-02,  1.5754e-02]],

 [[-9.1751e-02, -5.8711e-04],
  [-4.4704e-02,  6.6619e-03]],

 [[-9.4467e-02, -9.8491e-02],
  [ 5.1666e-02,  8.9268e-02]],
```

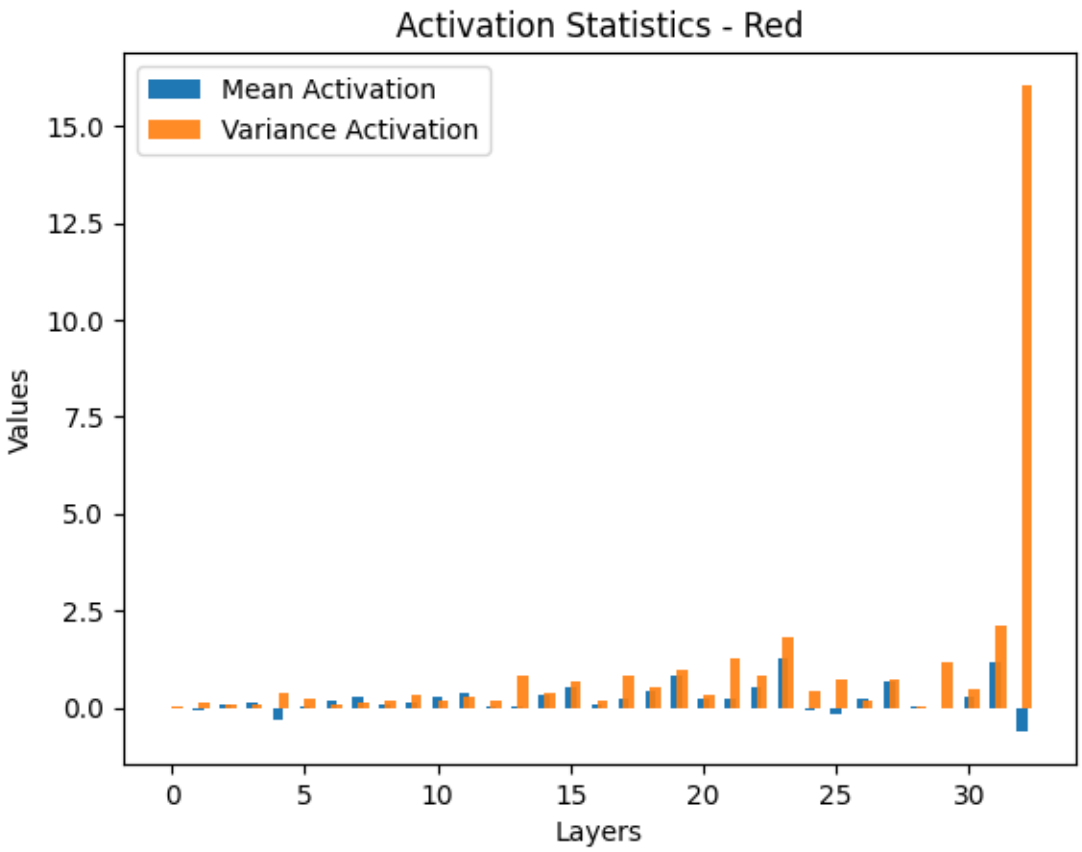
```
         …,

         [[-3.0491e-02, -8.7127e-02],
          [ 1.0346e-01, -5.3048e-02]],

         [[-8.9332e-02, -3.2832e-02],
          [ 8.9348e-02, -4.8782e-02]],

         [[ 5.3209e-02, -3.2534e-02],
          [-1.0836e-01, -1.0179e-01]]],


        [[[-9.7108e-02,  2.6516e-02],
          [-1.0874e-02,  3.7890e-02]],

         [[ 7.4632e-03, -9.8909e-04],
          [ 6.8768e-02,  1.5045e-02]],

         [[ 1.1077e-01,  7.5691e-02],
          [ 8.9206e-02,  7.6032e-02]],

         …,

         [[-5.2698e-02, -3.9539e-03],
          [ 1.2392e-02,  1.2389e-02]],

         [[ 2.7753e-03,  5.7053e-02],
          [ 1.2266e-01, -7.7410e-03]],

         [[-2.7858e-02, -6.0071e-02],
          [-6.6844e-03, -6.3582e-02]]]], requires_grad=True), Parameter
containing:
tensor([-0.0253, -0.0693,  0.0853,  0.0640, -0.0497, -0.0265,  0.0616, -0.0349,
         0.0508,  0.1229, -0.0942, -0.0379, -0.0829,  0.0190,  0.0947,  0.0906,
        -0.1139,  0.1186, -0.1201, -0.0389, -0.0823,  0.0021, -0.0324,  0.0809,
        -0.0470,  0.0364,  0.0802, -0.0522, -0.0928, -0.0831,  0.1237, -0.1230],
       requires_grad=True), Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       requires_grad=True), Parameter containing:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.], requires_grad=True), Parameter
containing:
tensor([[ 0.1279, -0.0852, -0.0384, -0.1443, -0.1168,  0.1147, -0.0911, -0.0422,
         -0.1284, -0.1658,  0.0098, -0.1626,  0.1005,  0.1368,  0.1744,  0.1207,
         -0.1194, -0.0186,  0.1184,  0.0338, -0.1316, -0.1348, -0.0763,  0.0233,
```

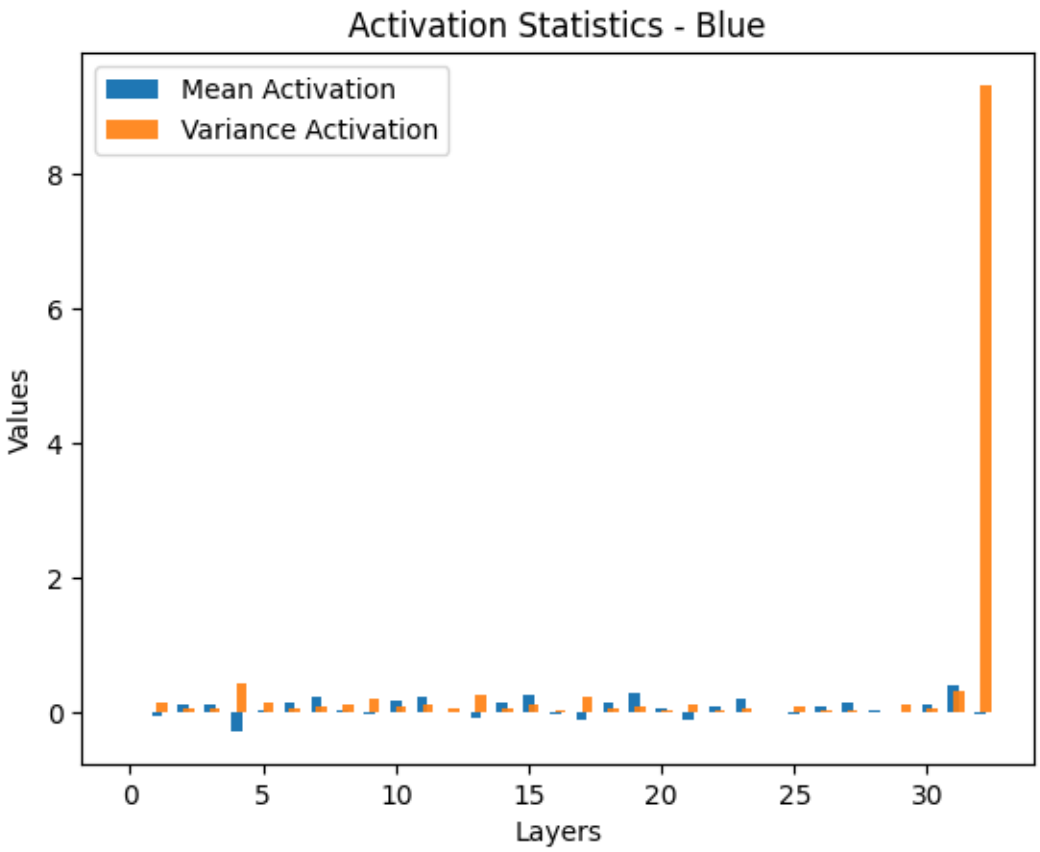
```
         -0.1742,  0.1765, -0.0800, -0.0212, -0.1347, -0.1106, -0.1325,
-0.0279],
        [ 0.1639, -0.0667, -0.0638, -0.1626, -0.0790, -0.0536, -0.0444,  0.1027,
         -0.0711,  0.0073,  0.0908, -0.1124, -0.0091,  0.0547,  0.0101,  0.1431,
          0.0517,  0.1645, -0.0012, -0.0658,  0.1529,  0.0161, -0.0023,  0.0076,
         -0.0888,  0.0094,  0.0333, -0.1504,  0.1637,  0.0849,  0.0558,
0.0491],
        [-0.0427,  0.0627, -0.0581, -0.0607, -0.1728, -0.1463, -0.1696,  0.1424,
         -0.0516, -0.0263, -0.0831, -0.0285, -0.0265, -0.0447, -0.1740, -0.0067,
          0.1098,  0.0687,  0.0706,  0.1605,  0.0362,  0.1031, -0.0675,  0.1608,
         -0.1339, -0.0781, -0.0820, -0.0548, -0.0697,  0.1049, -0.0175,
-0.1188]],
       requires_grad=True), Parameter containing:
tensor([ 0.1006, -0.1043,  0.1062], requires_grad=True)]
```

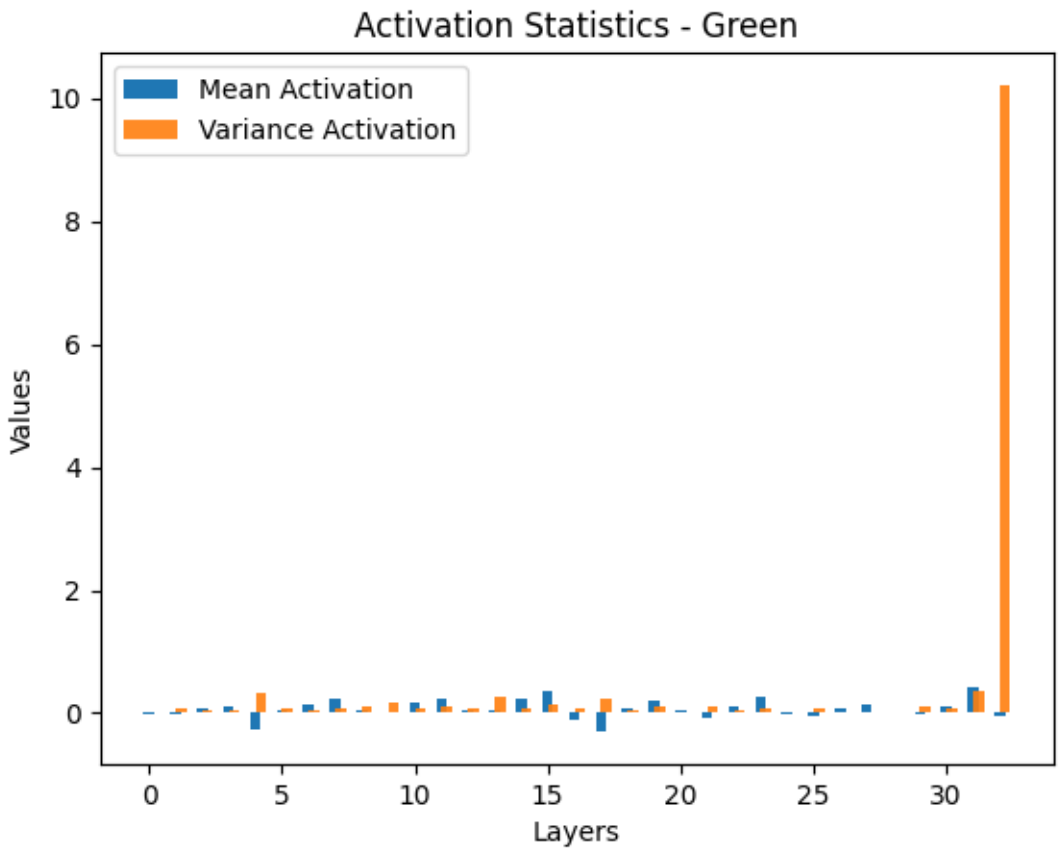
[10]:
```python
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
import random
from torchvision import transforms
from PIL import Image

# Assuming you have already defined the CustomCNN class and loaded the model
# Also, define the 'model' and 'transform' variables before this code snippet

# Create a PdfPages object to store the plots
pdf_pages = PdfPages('activation_plots.pdf')

# List of directories to investigate
directory_paths = [
    '/Users/peternoble/Downloads/DogEmotions/training/0',
    '/Users/peternoble/Downloads/DogEmotions/training/1',
    '/Users/peternoble/Downloads/DogEmotions/training/2',
    #'/Users/peternoble/Downloads/DogEmotions/training/3',
 ]
#folder_path = '/Users/peternoble/Downloads/DogEmotions/training/0'

# Custom labels for each directory
#custom_labels = ['Angry', 'Sad', 'Relaxed', 'Happy']
custom_labels = ['Red', 'Blue', 'Green']

# Loop through each directory
for i, directory_path in enumerate(directory_paths):
    # Choose a random image file from the list
    random_image_file = random.choice(os.listdir(directory_path))

    # Construct the full path to the randomly chosen image file
    random_image_path = os.path.join(directory_path, random_image_file)
```

```
# Open the image using PIL
random_image = Image.open(random_image_path)

# Define the transformation for the image
transform = transforms.Compose([
    #transforms.Resize((488, 488)),
    transforms.Resize((244, 244)),
    transforms.ToTensor(),
])

# Apply the transformation to the image
input_image = transform(random_image).unsqueeze(0)  # Add batch dimension

# Create a new figure for each plot
fig, ax = plt.subplots()

# Create a list to store the activation statistics
activation_stats = []

# Define a hook to store the activation statistics at each layer
def hook_fn(module, input, output):
    mean_activation = output.mean().item()
    var_activation = output.var().item()
    activation_stats.append((mean_activation, var_activation))

# Register the hook to each layer in your model
hooks = []
for layer in model.children():
    hook = layer.register_forward_hook(hook_fn)
    hooks.append(hook)

# Forward pass to obtain activation statistics
with torch.no_grad():
    model(input_image)

# Remove the hooks
for hook in hooks:
    hook.remove()

# Visualize activation statistics
layer_names = [f'Layer {i+1}' for i in range(len(activation_stats))]
mean_activations, var_activations = zip(*activation_stats)

# Create separate x-values for mean and variance
x_mean = [i for i in range(len(mean_activations))]
```

```
    x_var = [i + 0.2 for i in range(len(var_activations))]  # Add a small␣
↪offset for variance

    # Plot the activation statistics
    ax.bar(x_mean, mean_activations, width=0.4, label='Mean Activation')
    ax.bar(x_var, var_activations, width=0.4, label='Variance Activation',␣
↪alpha=0.9)
    ax.set_xlabel('Layers')
    ax.set_ylabel('Values')
    ax.set_title(f'Activation Statistics - {custom_labels[i]}')
    ax.legend()

    # Save the current plot to the PDF
    pdf_pages.savefig(fig)

    # Show the current plot on the screen
    plt.show()

    # Close the current plot
    plt.close()

# Close the PdfPages object
pdf_pages.close()
```


Activation Statistics - Red
Mean Activation
Variance Activation
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Values
0
5
10
15
20
25
30
Layers


Activation Statistics - Blue
Mean Activation
Variance Activation
Values
0
2
4
6
8
0
5
10
15
20
25
30
Layers



[40]:
```
# Assuming you have a validation DataLoader named val_loader
loaded_model = CustomCNN()

loaded_model.load_state_dict(torch.load('/Users/peternoble/Desktop/
 ↪dog_emotions_model.pth'))

# Set the model to evaluation mode
loaded_model.eval()

# Print and save the weights and biases
for name, param in loaded_model.named_parameters():
    if 'weight' in name or 'bias' in name:
        print(f"Parameter: {name}, Shape: {param.shape}")
        print("Values:")
        print(param.data)
        print("\n")

        # Flatten and save the parameters to text files
        flattened_params = param.data.flatten()
```

```
        np.savetxt(f'{name}.txt', flattened_params)
```

```
Parameter: conv1.weight, Shape: torch.Size([32, 3, 2, 2])
Values:
tensor([[[[-1.2862e-01, -7.4132e-02],
          [-5.1123e-02, -2.4877e-03]],

         [[-1.9497e-01,  4.1154e-02],
          [-1.8533e-01, -3.4249e-02]],

         [[ 1.6030e-01,  2.4662e-01],
          [-8.4199e-02, -2.5276e-01]]],


        [[[-6.9489e-03, -1.1550e-01],
          [ 1.4490e-01,  1.5836e-01]],

         [[ 1.6745e-01, -6.6469e-02],
          [ 9.3122e-02, -6.3083e-02]],

         [[ 2.7279e-01, -5.6057e-02],
          [-2.7427e-01,  2.2126e-01]]],


        [[[-2.0808e-01,  2.3376e-01],
          [ 1.7773e-01,  1.2671e-01]],

         [[-5.4855e-02, -1.8969e-01],
          [-2.1482e-01,  1.1987e-01]],

         [[-2.3454e-01, -1.4718e-01],
          [ 1.3466e-01,  2.5079e-01]]],


        [[[ 1.0878e-01, -2.1913e-01],
          [-2.0339e-01,  3.4536e-03]],

         [[ 1.4468e-01, -2.6049e-01],
          [-1.4162e-01, -2.0044e-01]],

         [[-6.1424e-02, -1.1785e-01],
          [ 1.1778e-01, -5.3533e-02]]],


        [[[ 4.3818e-02,  1.9947e-01],
          [ 1.1587e-01,  2.6819e-01]],

         [[ 2.8754e-01,  2.1529e-01],
```

```
  [ 2.8586e-01, -1.3636e-01]],

 [[-1.5417e-01, -2.6618e-01],
  [ 1.6691e-01,  1.4499e-01]]],


[[[-2.1729e-01, -1.9851e-01],
  [-2.4967e-01, -1.3278e-01]],

 [[-2.1608e-01,  2.6612e-01],
  [-1.5269e-01, -9.3307e-02]],

 [[-1.8551e-01, -7.0948e-02],
  [ 2.7045e-01,  7.6375e-02]]],


[[[-2.2386e-01, -1.2155e-01],
  [-1.8215e-01, -1.0601e-02]],

 [[-2.4377e-02, -9.8831e-02],
  [-1.6564e-01,  1.3581e-01]],

 [[ 3.0330e-02, -2.6251e-01],
  [-2.7283e-01,  2.9806e-02]]],


[[[ 1.2161e-04,  5.3137e-02],
  [-1.9250e-01,  2.4779e-01]],

 [[ 8.2522e-02, -2.2375e-01],
  [ 4.1588e-02,  1.3680e-01]],

 [[ 2.3286e-01,  2.5193e-01],
  [-1.4712e-01, -5.6759e-02]]],


[[[ 1.2833e-01, -2.1341e-01],
  [-7.9442e-02, -1.9326e-01]],

 [[-1.1788e-01,  9.0007e-02],
  [-1.2410e-01, -2.1450e-01]],

 [[-2.0532e-01,  5.7660e-02],
  [-2.0129e-01,  2.7650e-01]]],


[[[-1.7102e-01,  2.8426e-01],
  [-1.6310e-01, -4.2167e-02]],
```

```
 [[-2.2540e-01, -7.3587e-02],
  [ 2.5221e-01,  2.5372e-01]],

 [[-2.1754e-01, -2.4622e-01],
  [-2.6112e-01, -1.1207e-01]]],


[[[ 1.6128e-01,  2.4160e-01],
  [-2.8101e-01, -2.0124e-01]],

 [[ 2.9695e-02,  6.4366e-02],
  [-1.1276e-01,  3.9310e-02]],

 [[-2.3562e-01, -4.5487e-02],
  [-1.2769e-01, -1.2402e-01]]],


[[[ 1.9316e-01, -7.1542e-02],
  [ 1.7880e-02,  1.2100e-01]],

 [[ 7.3645e-02, -4.1847e-02],
  [-1.3371e-01,  1.4042e-01]],

 [[-9.7902e-02,  1.7557e-01],
  [-1.2173e-01,  1.2777e-01]]],


[[[-1.0646e-01,  1.7544e-01],
  [-1.9008e-01, -2.8179e-01]],

 [[-2.1405e-01,  2.8287e-01],
  [-9.1039e-02,  2.7296e-01]],

 [[ 1.0285e-01,  2.2912e-01],
  [ 1.6586e-01, -1.3051e-01]]],


[[[ 1.1123e-01,  9.8354e-03],
  [ 1.6477e-01, -1.3836e-01]],

 [[ 2.2818e-01,  1.8638e-01],
  [-1.5850e-01, -6.5581e-02]],

 [[-2.1204e-01,  1.7747e-01],
  [-2.3936e-01, -5.2997e-02]]],
```

```
[[[ 2.0355e-01,  1.2741e-01],
  [-1.1312e-01, -1.8448e-01]],

 [[ 1.8240e-01, -1.8318e-02],
  [-4.4789e-02,  1.0897e-01]],

 [[ 4.9810e-02,  2.0592e-01],
  [ 2.2145e-01,  5.8419e-02]]],


[[[-2.4172e-01, -1.0820e-01],
  [ 1.2886e-01, -6.8874e-02]],

 [[ 2.2063e-01,  2.3657e-01],
  [ 1.4910e-01, -3.2015e-02]],

 [[-2.2320e-01,  1.2945e-01],
  [-2.0851e-01,  2.4967e-01]]],


[[[-2.5639e-01, -7.5443e-02],
  [-3.2454e-02, -6.6399e-02]],

 [[ 1.1102e-01, -1.1391e-01],
  [ 2.6951e-01, -1.9989e-01]],

 [[ 1.3029e-01,  2.1191e-01],
  [-2.1165e-01,  3.9331e-02]]],


[[[-3.3680e-02,  1.1208e-01],
  [ 2.2649e-01,  1.4871e-02]],

 [[ 8.5198e-02, -2.2584e-01],
  [ 5.9034e-02,  1.3105e-01]],

 [[ 8.8506e-02,  2.2061e-01],
  [-1.3456e-01, -1.0704e-01]]],


[[[-1.5140e-01, -8.7042e-02],
  [ 1.8381e-01, -1.5470e-01]],

 [[-2.1762e-01, -1.3052e-02],
  [-2.1713e-02, -1.2686e-01]],

 [[ 4.3823e-02,  1.2352e-01],
  [-1.2777e-01, -1.4085e-01]]],
```

```
[[[-2.0174e-01,  1.8290e-01],
  [ 2.5212e-01,  2.8757e-01]],

 [[ 2.6915e-02, -1.7166e-01],
  [-2.2028e-01, -4.0050e-03]],

 [[ 1.4545e-01, -3.1423e-02],
  [-2.6597e-01, -2.6357e-01]]],


[[[-2.0283e-01, -7.1709e-03],
  [-2.7133e-02,  1.8765e-01]],

 [[ 1.2536e-01,  4.8427e-02],
  [-9.4222e-02,  2.2514e-01]],

 [[ 1.3544e-01, -2.4359e-01],
  [ 1.7904e-01, -1.9357e-01]]],


[[[-2.3608e-01, -5.6042e-02],
  [-2.7965e-01, -9.9222e-02]],

 [[ 1.6958e-01,  1.8307e-01],
  [-4.6652e-03, -1.8752e-01]],

 [[ 2.2848e-01,  2.6443e-01],
  [-1.9063e-01,  6.9474e-03]]],


[[[-2.1291e-01, -1.3518e-01],
  [ 1.2607e-01, -1.8231e-01]],

 [[ 1.4098e-01, -8.0658e-02],
  [ 1.9839e-01, -1.5396e-01]],

 [[ 6.1994e-02, -1.8228e-01],
  [ 1.2293e-01, -8.0927e-02]]],


[[[ 1.6856e-01,  1.0812e-01],
  [-4.2583e-02, -1.4939e-01]],

 [[-1.8132e-01,  6.6400e-02],
  [ 1.4434e-01,  3.4804e-02]],
```

```
 [[-1.9545e-01, -2.1538e-01],
  [ 3.1109e-02, -2.7817e-03]]],


[[[ 1.9476e-01,  1.6451e-01],
  [ 1.1195e-01, -1.8705e-01]],

 [[ 9.2237e-02,  2.8184e-01],
  [ 1.6060e-01,  1.5068e-01]],

 [[-1.8709e-01,  2.1163e-01],
  [ 1.0004e-01, -1.1521e-01]]],


[[[-4.3349e-02, -2.7923e-01],
  [-7.0851e-02, -2.3322e-01]],

 [[-2.5382e-01,  2.5598e-01],
  [ 2.3160e-01,  1.6760e-01]],

 [[-2.4309e-01, -1.4582e-01],
  [-9.0942e-03, -1.2669e-01]]],


[[[-1.9262e-01,  7.4707e-02],
  [-2.5990e-01,  2.7291e-01]],

 [[-9.0592e-03,  4.2963e-02],
  [ 1.5653e-01, -1.1706e-01]],

 [[-2.4442e-01,  1.5953e-01],
  [ 6.6715e-02, -1.4256e-01]]],


[[[-1.0135e-01, -1.5981e-02],
  [-1.6047e-01,  3.3011e-02]],

 [[-1.7852e-01, -1.5567e-01],
  [-2.2787e-01,  9.6782e-02]],

 [[-1.5605e-01,  1.5343e-01],
  [-1.3718e-01, -2.8206e-01]]],


[[[ 2.7258e-01,  2.5404e-01],
  [ 1.7902e-01, -1.0906e-01]],

 [[-7.8910e-03, -1.3714e-01],
```

```
          [-2.7044e-01, -4.5800e-02]],

         [[ 2.5189e-01, -1.1355e-01],
          [ 1.6623e-01, -1.5228e-01]]],


        [[[ 1.2579e-01,  1.0626e-01],
          [-1.4654e-01, -2.1371e-01]],

         [[ 2.2543e-01, -4.2494e-02],
          [ 4.6772e-02,  1.0567e-01]],

         [[ 2.0623e-01, -1.3304e-01],
          [-8.9317e-02, -6.8539e-02]]],


        [[[-1.6130e-01, -2.5655e-01],
          [-9.5789e-02, -2.5457e-01]],

         [[-9.1106e-02,  2.0298e-01],
          [ 6.5842e-02,  2.7852e-01]],

         [[-4.5266e-02,  1.7823e-01],
          [ 4.2380e-02, -3.0429e-02]]],


        [[[ 4.9580e-02,  8.8007e-03],
          [ 8.2871e-02, -2.2294e-01]],

         [[-2.1053e-01, -7.7271e-02],
          [ 1.5591e-01,  1.0724e-01]],

         [[ 4.9694e-02, -1.7419e-01],
          [ 1.7130e-01, -6.8518e-02]]]])


Parameter: conv1.bias, Shape: torch.Size([32])
Values:
tensor([-0.0531, -0.0504, -0.2108,  0.0900, -0.0814,  0.0250, -0.2178, -0.0996,
        -0.0092, -0.0616,  0.1462,  0.0564, -0.0808,  0.2129,  0.0332,  0.0256,
         0.0877, -0.2677,  0.2567,  0.0722,  0.2111, -0.1957, -0.0385, -0.1379,
        -0.2726,  0.2358,  0.0549,  0.0798, -0.2663, -0.1065, -0.2588, -0.1908])


Parameter: bn1.weight, Shape: torch.Size([32])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
Parameter: bn1.bias, Shape: torch.Size([32])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.])


Parameter: conv2.weight, Shape: torch.Size([32, 32, 4, 4])
Values:
tensor([[[[-1.8821e-03, -2.6503e-02, -1.4605e-02, -1.1147e-02],
          [-4.4179e-02, -1.2621e-02,  3.5566e-02,  9.7917e-03],
          [-3.8456e-03, -3.3261e-02,  3.0807e-03, -1.8286e-02],
          [ 5.7011e-03,  3.7161e-02, -3.1304e-02,  1.4198e-02]],

         [[-2.2320e-03, -2.2274e-02, -3.0260e-02, -3.7559e-02],
          [ 1.2292e-02, -8.2912e-03, -4.4935e-03,  4.0923e-02],
          [-7.9708e-03, -2.4131e-03, -1.4872e-02,  3.9768e-02],
          [ 6.3704e-03,  4.1147e-02,  1.3146e-02,  8.1047e-03]],

         [[ 1.4908e-02,  1.4298e-02,  1.8941e-02,  1.7881e-02],
          [-1.8374e-02,  7.9587e-03,  3.2842e-02,  2.3291e-02],
          [ 1.1876e-02, -2.6629e-02,  2.6711e-03,  4.4016e-02],
          [ 2.9481e-02,  4.1546e-03, -1.1571e-02,  8.2745e-03]],

         …,

         [[-4.0160e-02,  4.3214e-02,  3.5191e-02, -4.0726e-02],
          [-1.5793e-02, -4.3340e-02,  9.0029e-03,  1.0825e-02],
          [-8.4129e-03,  2.6540e-02,  2.4381e-02,  3.1115e-02],
          [ 3.8425e-02,  2.0936e-02, -2.2889e-03,  1.0093e-02]],

         [[-1.0760e-02, -3.4893e-02, -2.1125e-02,  2.1105e-02],
          [ 3.4920e-02, -3.9565e-02,  3.7951e-03, -1.4339e-04],
          [ 4.2798e-02, -1.7031e-02, -2.3830e-02, -8.9394e-03],
          [ 2.9233e-02, -4.1438e-02, -1.8542e-02,  2.5852e-02]],

         [[ 3.5524e-02,  1.8349e-02,  2.6059e-02,  3.5193e-02],
          [ 3.5271e-02, -3.3437e-02, -1.0150e-02, -4.0052e-02],
          [ 4.3020e-02,  9.3571e-03, -2.3200e-02,  3.4985e-02],
          [ 7.0088e-04,  3.0139e-02, -4.7072e-03,  2.4650e-02]]],


        [[[ 2.5779e-02, -3.5073e-03, -2.3780e-02, -1.5004e-02],
          [ 2.7978e-02,  3.6263e-02,  3.9753e-02, -2.6761e-02],
          [-2.6799e-02, -3.5306e-02,  1.0705e-02, -1.9302e-02],
          [ 8.1113e-03,  3.8871e-02,  3.9333e-02,  2.4703e-02]],
```

```
 [[-3.3475e-02,  3.8349e-03,  1.3924e-02, -3.1114e-04],
  [-7.7809e-03,  1.2071e-02, -3.4567e-02,  9.9812e-03],
  [-2.5696e-02,  6.0663e-03, -2.2994e-02,  3.6700e-02],
  [ 4.3189e-02,  2.8507e-02, -2.8797e-03,  9.6976e-04]],

 [[ 3.9534e-02, -1.2386e-02, -3.7731e-02, -8.1753e-03],
  [ 3.7966e-02, -2.6073e-02, -3.4783e-02,  2.6277e-03],
  [-2.9951e-02,  2.0356e-03,  4.1576e-02,  3.7215e-02],
  [ 8.2428e-03, -7.5800e-03, -1.5049e-02, -4.4231e-03]],

 …,

 [[-3.3264e-02, -2.6514e-02, -4.2748e-02,  2.2853e-02],
  [ 1.3812e-02,  3.3721e-03,  3.9921e-02,  2.9521e-02],
  [-9.1015e-03, -1.8394e-02, -2.4041e-02,  3.4095e-02],
  [-1.0337e-02, -4.3500e-02, -2.1646e-02,  4.2764e-02]],

 [[-3.2199e-02, -5.6661e-03,  2.4847e-03, -4.7200e-04],
  [ 1.0435e-02, -3.3871e-02, -1.6805e-03, -1.6118e-02],
  [ 3.0614e-02, -1.7132e-02, -2.9147e-02, -1.2166e-03],
  [ 1.7846e-02,  4.4077e-02,  4.1343e-02, -1.5975e-02]],

 [[-1.6011e-02,  2.4876e-02, -8.6224e-03,  1.3252e-02],
  [ 2.1853e-02, -4.0665e-02,  1.0038e-02, -3.3669e-02],
  [-3.4468e-02, -1.2963e-02,  3.9079e-02,  8.1945e-03],
  [ 3.8276e-02,  3.1483e-02, -2.0975e-02, -3.6671e-02]]],


[[[ 2.2293e-02,  3.8807e-02,  3.4638e-02,  3.3681e-02],
  [-3.9232e-02, -3.6450e-02,  2.8204e-02,  9.3512e-03],
  [-1.1353e-03, -1.9759e-02, -4.1681e-02, -4.2930e-02],
  [-2.3496e-02,  3.3512e-02,  1.7050e-02, -9.7387e-03]],

 [[-3.5777e-02, -1.7874e-02,  7.1799e-03, -3.1641e-02],
  [-2.4128e-02, -3.3190e-02, -3.6361e-02,  7.2805e-03],
  [-3.8583e-02,  3.5761e-02,  2.1184e-02, -2.6158e-02],
  [ 2.1530e-02, -1.4546e-02, -5.6179e-03,  3.6653e-02]],

 [[-2.3061e-03,  9.6271e-03, -3.3166e-02, -2.9849e-02],
  [ 3.6056e-02, -1.9608e-02, -1.1501e-02, -1.5256e-02],
  [-3.0135e-02, -2.5017e-03,  4.8839e-03,  1.5415e-02],
  [ 1.6104e-02,  4.5368e-03, -2.8377e-02,  3.8510e-02]],

 …,

 [[ 2.1365e-02, -1.5487e-02,  2.7149e-02,  3.0793e-02],
  [ 3.9325e-02, -2.7265e-03,  9.7416e-03, -4.3246e-04],
```

```
  [ 4.3651e-02, -9.7338e-04,  6.0543e-04,  1.3959e-02],
  [ 3.2616e-02, -3.6341e-02, -1.8818e-02, -3.0872e-02]],

 [[-4.3870e-02, -7.0506e-03, -1.1327e-02,  2.5699e-02],
  [-4.4092e-02, -4.0536e-03, -2.3633e-02, -5.7115e-03],
  [ 4.2301e-02,  3.1068e-02, -1.8547e-02,  2.1884e-02],
  [ 6.2363e-03, -1.0552e-02,  2.7235e-02,  3.6391e-02]],

 [[ 3.7304e-02, -9.4773e-03, -4.3119e-02,  2.4861e-02],
  [-1.9702e-02, -2.8824e-02,  2.1394e-02,  3.5128e-02],
  [-2.4368e-02,  3.3196e-02, -3.8285e-02,  1.5408e-02],
  [-2.3132e-02,  2.9357e-03,  1.5180e-02, -4.3116e-02]]],


…,


[[[-2.7558e-02,  2.9652e-02, -3.6566e-02, -7.4468e-03],
  [ 7.0401e-03,  3.7015e-02, -4.4187e-02,  1.0558e-03],
  [ 2.3466e-03,  4.2496e-02,  2.0574e-02,  2.1758e-02],
  [ 1.2174e-03, -1.6117e-02,  1.3316e-02,  9.5944e-03]],

 [[ 1.5543e-03, -3.7210e-02,  1.7885e-02,  1.6541e-02],
  [ 3.7919e-02,  3.7750e-02, -3.2786e-02, -3.8893e-02],
  [-1.3230e-02, -2.3674e-02, -3.0933e-02,  1.6964e-02],
  [-5.6389e-03, -4.2468e-02,  3.8340e-02, -1.2611e-03]],

 [[-2.8050e-02,  3.0551e-02,  4.7398e-03,  3.1007e-03],
  [ 3.0581e-02,  3.6264e-03, -1.0506e-02, -2.4749e-02],
  [ 2.5384e-02,  2.1755e-02,  1.0290e-02, -3.5370e-02],
  [-2.8222e-02,  1.4070e-02, -3.3722e-02, -2.9810e-02]],

 …,

 [[ 3.0115e-02,  4.1796e-02, -3.9204e-02,  2.5803e-02],
  [ 1.9174e-02,  1.6647e-02, -9.6432e-03,  7.0725e-03],
  [ 2.4131e-02,  1.0208e-02, -3.3396e-02,  9.0159e-03],
  [-4.1594e-02, -6.3820e-03,  6.8906e-03, -2.0750e-02]],

 [[ 1.1418e-02, -2.6633e-02,  2.6686e-02,  3.7205e-02],
  [ 2.8501e-02, -1.3784e-02,  6.5233e-03, -3.2934e-02],
  [-2.1707e-02, -4.0225e-02, -2.1609e-02, -3.0288e-02],
  [-2.6565e-02, -2.3614e-02,  3.8589e-02,  2.1033e-02]],

 [[ 8.2600e-03,  2.1860e-03, -1.4169e-02, -9.8505e-03],
  [ 2.5326e-02,  1.5508e-03,  8.9214e-03, -1.8561e-02],
  [ 2.3431e-02, -1.3489e-02,  5.6862e-03,  4.7615e-03],
  [ 6.0466e-03,  3.2949e-02,  1.9107e-02,  3.8783e-02]]],
```

```
[[[-2.8833e-02,  1.6196e-02,  3.4690e-02,  5.0575e-03],
  [ 2.9514e-02, -1.0021e-02,  5.7561e-03,  5.1865e-04],
  [-3.8744e-02, -3.1062e-03, -2.8801e-02, -3.7845e-02],
  [-3.2894e-02, -1.8415e-02,  1.6492e-02,  8.9802e-03]],

 [[ 4.3694e-03, -8.5728e-03,  1.4606e-02, -3.9464e-02],
  [ 1.1575e-02,  1.8579e-02,  1.4812e-02,  4.1376e-02],
  [ 2.2529e-02,  2.5798e-02, -2.9106e-02, -2.8786e-02],
  [ 1.8064e-02, -3.9078e-02,  2.9468e-02,  2.5870e-02]],

 [[-2.8994e-02, -3.9664e-02,  6.7804e-03,  3.6112e-02],
  [ 3.7025e-02,  1.3950e-02,  2.2106e-02, -3.9447e-02],
  [-4.3603e-02, -8.1671e-04,  6.9148e-03,  3.9349e-02],
  [-2.6072e-02, -4.2814e-02,  7.1259e-03,  1.2973e-02]],

 …,

 [[ 4.0241e-02, -2.4249e-02,  3.9319e-02, -1.5380e-02],
  [ 3.0221e-02,  1.5771e-02,  3.9111e-02,  2.7271e-02],
  [-3.0093e-02, -3.9932e-02, -1.1610e-02, -3.2664e-02],
  [-1.4736e-02,  3.6823e-03, -7.9107e-03,  4.3803e-02]],

 [[-4.1503e-02, -1.3731e-02,  4.0309e-02,  4.4953e-03],
  [-2.4680e-02,  2.5947e-02,  3.9358e-02, -3.4756e-02],
  [-4.3580e-02,  2.0161e-02,  4.1084e-02,  3.0454e-02],
  [ 2.3876e-03,  1.1666e-02,  4.3246e-03,  3.7567e-02]],

 [[ 6.6418e-03,  1.5001e-02,  1.1115e-02, -1.1592e-02],
  [ 2.9918e-04,  3.1330e-02, -1.2373e-03,  2.5205e-02],
  [-5.6090e-04, -1.7289e-02, -3.0567e-02,  2.5893e-03],
  [-1.7045e-02,  3.6678e-02,  2.5376e-03, -1.4118e-02]]],


[[[-6.6070e-03, -5.5165e-05,  1.2819e-02, -1.6554e-02],
  [ 3.6802e-02,  1.8771e-02, -3.4272e-02, -2.3605e-02],
  [ 2.6347e-02, -5.9776e-03, -7.4485e-03, -7.4394e-03],
  [ 3.4274e-02,  3.3746e-02,  3.8357e-02, -1.8506e-02]],

 [[ 4.0224e-02, -2.1919e-03, -2.8842e-02,  3.6720e-02],
  [-2.7114e-02, -3.2736e-02,  2.1980e-02,  1.5536e-03],
  [ 1.1152e-02, -2.4586e-02,  4.2935e-02,  1.8182e-02],
  [ 3.4576e-02,  2.9374e-02, -3.2962e-02, -2.6529e-02]],

 [[ 6.8270e-03, -1.0075e-03, -1.3838e-02,  2.7986e-02],
  [-2.1465e-02, -2.5866e-02,  2.0787e-03, -3.9265e-02],
  [ 2.1506e-02, -4.2432e-02, -2.2735e-02,  9.8403e-04],
```

```
          [-1.6872e-02,  4.2932e-03, -2.1585e-02,  3.3299e-03]],

         …,

         [[ 4.1811e-02,  3.2271e-02, -9.6462e-03, -2.7675e-02],
          [-8.4016e-04, -2.4699e-03, -1.5867e-02,  4.2870e-02],
          [-2.1763e-02,  3.5990e-02, -1.6936e-02, -8.3683e-03],
          [-1.3952e-02,  4.2594e-02,  3.0001e-02,  1.8287e-02]],

         [[ 2.2226e-02, -1.3275e-03,  2.2636e-02, -2.3606e-02],
          [-1.9255e-02,  1.3936e-02, -1.9148e-02, -2.4174e-02],
          [-3.0222e-02, -7.3415e-03, -3.0313e-02,  5.0275e-04],
          [-2.1575e-02, -2.5969e-02,  3.5347e-02, -1.2657e-04]],

         [[-3.1929e-02,  2.8433e-03, -3.6847e-02,  4.4070e-02],
          [-1.4485e-02,  3.5303e-02,  4.1365e-02,  3.0136e-02],
          [-2.7794e-02,  3.2234e-02,  8.4246e-03, -4.7168e-03],
          [ 1.1879e-02, -2.0200e-02, -1.1834e-03, -3.2851e-02]]]])


Parameter: conv2.bias, Shape: torch.Size([32])
Values:
tensor([-0.0042, -0.0433, -0.0228, -0.0375, -0.0166, -0.0137,  0.0043, -0.0016,
         0.0299, -0.0056,  0.0437, -0.0050, -0.0318,  0.0006,  0.0143,  0.0240,
        -0.0298, -0.0373, -0.0024,  0.0247, -0.0047, -0.0223, -0.0149, -0.0269,
         0.0315, -0.0181, -0.0272, -0.0400,  0.0433, -0.0103,  0.0263, -0.0152])


Parameter: bn2.weight, Shape: torch.Size([32])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])


Parameter: bn2.bias, Shape: torch.Size([32])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.])


Parameter: conv3.weight, Shape: torch.Size([16, 32, 2, 2])
Values:
tensor([[[[-0.0491,  0.0383],
          [-0.0774, -0.0269]],

         [[ 0.0558, -0.0473],
          [-0.0298,  0.0117]],
```

```
 [[-0.0198, -0.0142],
  [-0.0305, -0.0248]],

 …,

 [[-0.0743,  0.0710],
  [-0.0033,  0.0211]],

 [[ 0.0585, -0.0783],
  [ 0.0708, -0.0363]],

 [[-0.0073, -0.0693],
  [-0.0422, -0.0274]]],


[[[ 0.0732, -0.0297],
  [-0.0174, -0.0371]],

 [[-0.0576,  0.0037],
  [-0.0752,  0.0446]],

 [[ 0.0613, -0.0884],
  [ 0.0740,  0.0328]],

 …,

 [[-0.0781, -0.0098],
  [-0.0761, -0.0442]],

 [[-0.0207, -0.0471],
  [-0.0288,  0.0243]],

 [[-0.0485,  0.0699],
  [-0.0861, -0.0582]]],


[[[ 0.0279, -0.0029],
  [-0.0402, -0.0643]],

 [[-0.0703, -0.0646],
  [-0.0310,  0.0110]],

 [[ 0.0008,  0.0034],
  [ 0.0133,  0.0874]],

 …,
```

```
 [[ 0.0148,  0.0431],
  [-0.0291,  0.0161]],

 [[ 0.0041, -0.0538],
  [-0.0635,  0.0516]],

 [[ 0.0025, -0.0287],
  [ 0.0114, -0.0826]]],


…,


[[[-0.0709, -0.0635],
  [-0.0583, -0.0709]],

 [[ 0.0474,  0.0723],
  [ 0.0628,  0.0471]],

 [[ 0.0396,  0.0851],
  [-0.0213, -0.0207]],

 …,

 [[-0.0676,  0.0263],
  [ 0.0135, -0.0059]],

 [[ 0.0825, -0.0018],
  [-0.0809,  0.0212]],

 [[ 0.0569, -0.0625],
  [ 0.0447,  0.0003]]],


[[[-0.0382, -0.0505],
  [ 0.0814, -0.0810]],

 [[ 0.0216, -0.0858],
  [-0.0877,  0.0697]],

 [[-0.0653,  0.0040],
  [ 0.0627,  0.0719]],

 …,

 [[-0.0374, -0.0619],
  [ 0.0445,  0.0572]],
```

```
          [[-0.0611,  0.0542],
           [-0.0805, -0.0594]],

          [[ 0.0572, -0.0261],
           [ 0.0069,  0.0876]]],


         [[[-0.0662, -0.0327],
           [-0.0620, -0.0848]],

          [[ 0.0600,  0.0298],
           [ 0.0123,  0.0856]],

          [[-0.0442,  0.0755],
           [ 0.0676, -0.0704]],

          …,

          [[ 0.0515,  0.0488],
           [ 0.0166, -0.0157]],

          [[-0.0225,  0.0625],
           [-0.0046, -0.0614]],

          [[ 0.0729,  0.0858],
           [-0.0705,  0.0084]]]])


Parameter: conv3.bias, Shape: torch.Size([16])
Values:
tensor([-0.0433,  0.0724,  0.0668, -0.0426, -0.0225,  0.0478, -0.0668,  0.0240,
        -0.0324, -0.0628, -0.0444,  0.0542,  0.0696,  0.0170, -0.0550,  0.0432])


Parameter: bn3.weight, Shape: torch.Size([16])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])


Parameter: bn3.bias, Shape: torch.Size([16])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])


Parameter: conv4.weight, Shape: torch.Size([8, 16, 2, 2])
Values:
tensor([[[[ 0.0920, -0.0141],
          [-0.0002,  0.1142]],
```

```
 [[ 0.0978,  0.0426],
  [-0.1034,  0.0326]],

 [[-0.0218,  0.1014],
  [-0.0388,  0.0764]],

 [[ 0.0364,  0.0307],
  [ 0.1028, -0.0171]],

 [[-0.0443,  0.1244],
  [-0.1220, -0.0173]],

 [[ 0.0199, -0.1139],
  [-0.1153,  0.0392]],

 [[-0.0671, -0.0281],
  [ 0.1133,  0.0624]],

 [[-0.0877, -0.0288],
  [ 0.0133,  0.0054]],

 [[-0.0483, -0.0793],
  [ 0.0021,  0.0099]],

 [[-0.0997, -0.0436],
  [-0.0764, -0.0513]],

 [[ 0.0733,  0.0800],
  [-0.0892, -0.0692]],

 [[ 0.0071, -0.0313],
  [-0.0482, -0.0122]],

 [[-0.0470,  0.0772],
  [ 0.0524, -0.0664]],

 [[ 0.0839,  0.1000],
  [ 0.1126,  0.0281]],

 [[-0.1235, -0.0217],
  [-0.0930, -0.1038]],

 [[ 0.0713,  0.1143],
  [-0.0167, -0.0514]]],


[[[-0.0364, -0.0562],
```

```
 [ 0.0537, -0.0290]],

[[ 0.1080,  0.0569],
 [ 0.0350, -0.0127]],

[[-0.0477,  0.0534],
 [-0.0487,  0.0979]],

[[-0.0284, -0.0928],
 [ 0.0140,  0.0256]],

[[ 0.1249,  0.0551],
 [-0.1195, -0.0502]],

[[ 0.0838, -0.0820],
 [ 0.0162, -0.0099]],

[[ 0.0844,  0.0154],
 [ 0.0240,  0.0394]],

[[-0.0898,  0.0594],
 [-0.0330, -0.0483]],

[[-0.0221, -0.0102],
 [ 0.0648,  0.0740]],

[[-0.0899, -0.0847],
 [ 0.0796, -0.0879]],

[[-0.0340, -0.0032],
 [-0.0332,  0.0264]],

[[ 0.0817,  0.0061],
 [-0.0760, -0.1087]],

[[-0.0155, -0.0552],
 [ 0.0206,  0.0962]],

[[ 0.1121, -0.0595],
 [-0.0732, -0.0277]],

[[-0.1211, -0.0657],
 [ 0.0405, -0.1238]],

[[-0.0567,  0.0360],
 [-0.0537,  0.0829]]],
```

```
[[[-0.0811, -0.0972],
  [-0.0329, -0.0900]],

 [[ 0.0224, -0.0119],
  [-0.0304, -0.0847]],

 [[-0.0262, -0.0070],
  [ 0.0270, -0.0191]],

 [[-0.0200, -0.1148],
  [-0.0734,  0.0374]],

 [[ 0.1197, -0.0690],
  [ 0.1020,  0.0031]],

 [[-0.0949,  0.0613],
  [ 0.0613,  0.0556]],

 [[-0.0798, -0.0711],
  [-0.0554,  0.0732]],

 [[-0.0417, -0.0129],
  [ 0.0588, -0.0898]],

 [[-0.0052,  0.1018],
  [ 0.0219,  0.0461]],

 [[ 0.0557,  0.1167],
  [-0.1091,  0.0291]],

 [[-0.0194, -0.0649],
  [ 0.1220, -0.0931]],

 [[-0.0324,  0.0565],
  [-0.0983,  0.0652]],

 [[-0.1082, -0.0331],
  [ 0.1033,  0.0810]],

 [[-0.0302, -0.0511],
  [-0.0033,  0.0640]],

 [[-0.0915,  0.0729],
  [-0.0018, -0.0222]],

 [[ 0.0166,  0.0516],
  [-0.0789,  0.0306]]],
```

```
[[[ 0.0667, -0.0305],
  [ 0.0417, -0.1027]],

 [[ 0.1136,  0.0431],
  [ 0.0547, -0.0753]],

 [[ 0.0631, -0.1193],
  [ 0.0045,  0.1245]],

 [[ 0.0405,  0.0315],
  [-0.0746,  0.0031]],

 [[-0.0754, -0.1227],
  [ 0.0884, -0.0034]],

 [[ 0.0747,  0.0789],
  [ 0.0034, -0.0467]],

 [[ 0.0355, -0.1016],
  [-0.1126, -0.0923]],

 [[ 0.1210,  0.0412],
  [ 0.0033,  0.1173]],

 [[-0.0736, -0.1150],
  [ 0.1046,  0.0564]],

 [[ 0.0876,  0.0422],
  [ 0.0555, -0.0213]],

 [[ 0.0809, -0.0919],
  [ 0.0173,  0.0974]],

 [[ 0.1139,  0.0660],
  [-0.0055,  0.0169]],

 [[-0.0868, -0.0484],
  [ 0.0778, -0.0177]],

 [[-0.0891, -0.1247],
  [ 0.0119, -0.0715]],

 [[-0.0656,  0.1051],
  [-0.0659, -0.0726]],

 [[ 0.0233, -0.0708],
  [-0.1159, -0.1183]]],
```

```
[[[ 0.0893,  0.0819],
  [-0.1065,  0.0317]],

 [[-0.0094, -0.1124],
  [ 0.0750, -0.1013]],

 [[-0.0301,  0.0713],
  [-0.0309,  0.0433]],

 [[ 0.0721, -0.0851],
  [ 0.0233, -0.1157]],

 [[-0.1098, -0.0669],
  [-0.0772, -0.0496]],

 [[-0.0066,  0.0560],
  [ 0.1040, -0.1040]],

 [[-0.0186, -0.0783],
  [ 0.0014,  0.1028]],

 [[-0.0041, -0.0916],
  [ 0.0201, -0.1206]],

 [[ 0.0084,  0.0025],
  [ 0.0838, -0.0537]],

 [[-0.0963,  0.0813],
  [ 0.1058,  0.0695]],

 [[-0.0378,  0.0817],
  [ 0.0745, -0.1059]],

 [[-0.0624,  0.0739],
  [-0.0617, -0.0287]],

 [[-0.0080, -0.0092],
  [ 0.0557, -0.1059]],

 [[ 0.0687,  0.0635],
  [-0.0178,  0.0223]],

 [[ 0.0604,  0.1172],
  [ 0.0266, -0.1099]],

 [[ 0.0977, -0.1062],
```

```
  [-0.0036, -0.0555]]],


[[[ 0.0574, -0.1095],
  [-0.0155, -0.0921]],

 [[-0.0522, -0.0826],
  [ 0.0120, -0.0643]],

 [[-0.1029, -0.0479],
  [-0.0657,  0.0286]],

 [[ 0.0791,  0.0018],
  [-0.0957,  0.0840]],

 [[-0.0737, -0.0551],
  [-0.0055,  0.0164]],

 [[-0.0074,  0.0767],
  [ 0.0075,  0.0755]],

 [[ 0.0266, -0.0167],
  [ 0.0912, -0.0733]],

 [[-0.0235,  0.1086],
  [-0.0732, -0.0199]],

 [[ 0.0162, -0.1048],
  [ 0.0851, -0.0853]],

 [[ 0.0568,  0.0060],
  [-0.0927, -0.0094]],

 [[ 0.0531,  0.0126],
  [ 0.0594, -0.0040]],

 [[-0.0316,  0.0177],
  [ 0.0915, -0.0287]],

 [[ 0.0932, -0.0092],
  [ 0.0643,  0.0855]],

 [[ 0.0686,  0.0664],
  [ 0.1153, -0.0850]],

 [[-0.0610, -0.0388],
  [ 0.0545,  0.0426]],
```

```
  [[ 0.0984, -0.0987],
   [ 0.0384,  0.0060]]],


 [[[-0.0235,  0.0470],
   [ 0.0439,  0.0397]],

  [[-0.0089,  0.0608],
   [-0.0563,  0.1188]],

  [[ 0.0349, -0.0981],
   [ 0.1030, -0.0278]],

  [[-0.0686,  0.0994],
   [-0.0128, -0.1169]],

  [[-0.0822,  0.0931],
   [-0.0360,  0.0071]],

  [[ 0.0181, -0.0757],
   [-0.0816, -0.0766]],

  [[ 0.1005, -0.0678],
   [-0.0545, -0.0928]],

  [[ 0.0526, -0.0143],
   [-0.1201, -0.0050]],

  [[-0.1070, -0.0466],
   [ 0.0783, -0.0169]],

  [[ 0.0405,  0.1167],
   [-0.1086,  0.1066]],

  [[ 0.1061, -0.1076],
   [ 0.0065, -0.0970]],

  [[ 0.0732, -0.0442],
   [-0.0550,  0.1158]],

  [[-0.1073, -0.0961],
   [ 0.0583,  0.0406]],

  [[-0.0215, -0.0782],
   [-0.0259,  0.0023]],

  [[ 0.0124,  0.0494],
   [ 0.0344,  0.0212]],
```

```
  [[-0.0835, -0.0995],
   [-0.1003, -0.0734]]],


[[[ 0.0706,  0.0297],
  [-0.0355,  0.0817]],

 [[ 0.0393,  0.0551],
  [ 0.1248,  0.0284]],

 [[-0.0253,  0.0616],
  [ 0.0607, -0.1098]],

 [[ 0.0795,  0.0323],
  [ 0.1182, -0.0809]],

 [[-0.0777, -0.0393],
  [ 0.0620,  0.0689]],

 [[ 0.0867,  0.0530],
  [-0.1235,  0.1030]],

 [[ 0.0845, -0.0586],
  [ 0.0165, -0.0008]],

 [[ 0.0321, -0.1242],
  [ 0.0552, -0.0123]],

 [[-0.0162, -0.0435],
  [ 0.0607,  0.0548]],

 [[-0.0842,  0.0055],
  [-0.0508,  0.1180]],

 [[-0.0818,  0.0193],
  [ 0.0588,  0.0087]],

 [[ 0.0131,  0.1222],
  [ 0.0413,  0.0198]],

 [[ 0.0076,  0.0863],
  [-0.1200,  0.0230]],

 [[ 0.0703, -0.0070],
  [-0.1171, -0.0445]],

 [[ 0.1069, -0.0389],
```

```
          [ 0.0413, -0.0197]],

         [[ 0.1000, -0.0844],
          [-0.0107,  0.0859]]]])


Parameter: conv4.bias, Shape: torch.Size([8])
Values:
tensor([-0.0425,  0.0606,  0.0307, -0.0505,  0.1075,  0.0153, -0.0777,  0.0429])


Parameter: bn4.weight, Shape: torch.Size([8])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1.])


Parameter: bn4.bias, Shape: torch.Size([8])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0.])


Parameter: conv5.weight, Shape: torch.Size([4, 8, 2, 2])
Values:
tensor([[[[-0.1239, -0.0352],
          [ 0.1126,  0.0106]],

         [[ 0.0582, -0.0870],
          [ 0.0994,  0.0143]],

         [[ 0.0283, -0.0152],
          [ 0.0632,  0.0542]],

         [[-0.1178,  0.0831],
          [-0.1337,  0.0017]],

         [[ 0.1320,  0.0926],
          [-0.1362, -0.1180]],

         [[ 0.1466,  0.0107],
          [ 0.0587, -0.0234]],

         [[-0.0814,  0.0444],
          [-0.1296, -0.1608]],

         [[ 0.1047,  0.0511],
          [ 0.0835,  0.1240]]],
```

```
[[[ 0.1440, -0.0846],
  [-0.1398, -0.0229]],

 [[-0.0405, -0.0957],
  [ 0.1701,  0.0623]],

 [[-0.0651, -0.1374],
  [-0.1696,  0.0014]],

 [[ 0.0105, -0.1269],
  [-0.0258, -0.1617]],

 [[ 0.1317, -0.1598],
  [-0.0302, -0.0241]],

 [[-0.1645,  0.1217],
  [ 0.1543,  0.1433]],

 [[ 0.1146,  0.1312],
  [ 0.0665, -0.1525]],

 [[-0.0067,  0.0873],
  [-0.1518, -0.0823]]],


[[[-0.1024, -0.1212],
  [ 0.1754, -0.0161]],

 [[ 0.1295,  0.0267],
  [ 0.1747, -0.1685]],

 [[ 0.1225, -0.1028],
  [ 0.0324, -0.0781]],

 [[-0.0017,  0.0039],
  [-0.1121,  0.1235]],

 [[-0.1337, -0.0239],
  [-0.0337,  0.0478]],

 [[-0.0075, -0.0696],
  [-0.0090,  0.0956]],

 [[ 0.0095,  0.1765],
  [-0.0046, -0.0013]],

 [[-0.0543,  0.1631],
  [-0.1583,  0.0581]]],
```

```
        [[[ 0.0543,  0.0290],
          [ 0.0722,  0.1368]],

         [[-0.0974,  0.0886],
          [-0.1519,  0.0513]],

         [[ 0.1430,  0.1340],
          [ 0.1302, -0.1472]],

         [[-0.0009,  0.1479],
          [ 0.1691, -0.0541]],

         [[ 0.1326,  0.1179],
          [-0.1532,  0.0328]],

         [[-0.0442,  0.0092],
          [-0.1744,  0.0385]],

         [[ 0.0549,  0.1651],
          [ 0.1029, -0.1504]],

         [[-0.0797,  0.1561],
          [ 0.1639,  0.0340]]]])


Parameter: conv5.bias, Shape: torch.Size([4])
Values:
tensor([-0.0946, -0.1454, -0.1503, -0.0977])


Parameter: bn5.weight, Shape: torch.Size([4])
Values:
tensor([1., 1., 1., 1.])


Parameter: bn5.bias, Shape: torch.Size([4])
Values:
tensor([0., 0., 0., 0.])


Parameter: conv6.weight, Shape: torch.Size([4, 4, 2, 2])
Values:
tensor([[[[-0.0260, -0.1938],
          [ 0.1492, -0.1937]],

         [[-0.1937, -0.2335],
```

```
  [-0.0790, -0.2050]],

 [[-0.1191,  0.0566],
  [ 0.1748, -0.1578]],

 [[ 0.0443,  0.1054],
  [-0.1397,  0.0068]]],


[[[-0.0760,  0.2171],
  [ 0.0721,  0.0171]],

 [[-0.1652, -0.0693],
  [ 0.0912,  0.1203]],

 [[ 0.1186, -0.0949],
  [-0.1725,  0.2460]],

 [[-0.0707,  0.0737],
  [-0.0439, -0.0455]]],


[[[ 0.1877, -0.1522],
  [ 0.0335,  0.1243]],

 [[-0.2161,  0.2186],
  [ 0.0470,  0.0101]],

 [[ 0.1779,  0.2365],
  [ 0.0642,  0.0538]],

 [[ 0.1631,  0.2415],
  [-0.0873,  0.0808]]],


[[[-0.0464, -0.0822],
  [-0.0116,  0.1124]],

 [[ 0.1264,  0.1688],
  [-0.1984,  0.0143]],

 [[-0.1420,  0.1063],
  [ 0.2455, -0.2075]],

 [[-0.1444,  0.2029],
  [ 0.2134,  0.0022]]]])
```

```
Parameter: conv6.bias, Shape: torch.Size([4])
Values:
tensor([-0.1008,  0.1889,  0.2498, -0.2343])


Parameter: bn6.weight, Shape: torch.Size([4])
Values:
tensor([1., 1., 1., 1.])


Parameter: bn6.bias, Shape: torch.Size([4])
Values:
tensor([0., 0., 0., 0.])


Parameter: conv7.weight, Shape: torch.Size([16, 4, 2, 2])
Values:
tensor([[[[-0.2468, -0.0179],
          [-0.1312, -0.1056]],

         [[-0.1961,  0.1454],
          [ 0.0656,  0.1806]],

         [[-0.0144, -0.0598],
          [-0.0484,  0.0911]],

         [[ 0.0329,  0.2083],
          [-0.0098,  0.0545]]],


        [[[ 0.0669,  0.1772],
          [ 0.0166, -0.1220]],

         [[-0.2018, -0.0592],
          [-0.2101,  0.2421]],

         [[ 0.0023, -0.0750],
          [ 0.0533,  0.1641]],

         [[-0.1024, -0.1197],
          [ 0.0189, -0.1066]]],


        [[[-0.1069, -0.2034],
          [ 0.2436, -0.0571]],

         [[ 0.0420,  0.2361],
          [ 0.0843, -0.0043]],
```

```
 [[ 0.1187,  0.0571],
  [-0.0123,  0.1614]],

 [[-0.1846,  0.0731],
  [-0.2365, -0.0778]]],


[[[-0.0406, -0.1225],
  [ 0.2488, -0.1023]],

 [[ 0.0382, -0.1514],
  [ 0.2279,  0.1228]],

 [[ 0.2262,  0.2264],
  [ 0.1205,  0.1001]],

 [[ 0.1163,  0.0037],
  [-0.0667, -0.2498]]],


[[[ 0.2022, -0.2256],
  [ 0.2211,  0.0150]],

 [[-0.0745,  0.2323],
  [ 0.1020, -0.1251]],

 [[ 0.2078,  0.0309],
  [ 0.0876,  0.2294]],

 [[-0.1752, -0.2159],
  [-0.1269,  0.1105]]],


[[[ 0.2217,  0.0882],
  [ 0.0835, -0.0904]],

 [[ 0.1742,  0.1831],
  [-0.0198, -0.0144]],

 [[ 0.1564,  0.0672],
  [-0.2438, -0.0149]],

 [[-0.0629, -0.0589],
  [ 0.2281,  0.0221]]],


[[[-0.1930,  0.2462],
```

```
   [-0.1253, -0.2015]],

  [[-0.2261,  0.0336],
   [-0.0981, -0.2244]],

  [[ 0.2244,  0.0897],
   [ 0.1812, -0.1056]],

  [[ 0.0468,  0.0580],
   [ 0.2498, -0.1336]]],


 [[[-0.0381,  0.2368],
   [-0.0915, -0.1547]],

  [[ 0.0419,  0.1763],
   [ 0.0672, -0.1973]],

  [[-0.0728, -0.2054],
   [ 0.1223,  0.0063]],

  [[ 0.0017,  0.1819],
   [-0.2331,  0.1266]]],


 [[[-0.0580,  0.0524],
   [-0.1952,  0.0700]],

  [[-0.1865, -0.1874],
   [ 0.0739,  0.0838]],

  [[-0.2275,  0.0898],
   [ 0.1531, -0.0221]],

  [[ 0.0145,  0.1056],
   [-0.1403,  0.1891]]],


 [[[ 0.1517,  0.0975],
   [-0.1995,  0.1280]],

  [[-0.2208,  0.1931],
   [-0.0486,  0.2003]],

  [[-0.1006, -0.0337],
   [-0.1611,  0.0694]],

  [[ 0.0638, -0.0690],
```

```
   [ 0.0539,  0.1127]]],


[[[ 0.1240,  0.0173],
  [ 0.2406,  0.0880]],

 [[-0.1694, -0.0928],
  [ 0.1042,  0.0026]],

 [[ 0.1895, -0.1462],
  [-0.1147,  0.2277]],

 [[ 0.0930,  0.0809],
  [-0.2189, -0.1731]]],


[[[-0.1293,  0.1637],
  [ 0.0361,  0.2496]],

 [[ 0.0270, -0.2194],
  [-0.2376, -0.0140]],

 [[ 0.0299, -0.1374],
  [ 0.1165, -0.0853]],

 [[ 0.1628,  0.1057],
  [ 0.0685,  0.0710]]],


[[[ 0.2311,  0.0298],
  [ 0.1745, -0.1091]],

 [[-0.2440,  0.1742],
  [-0.0782,  0.1282]],

 [[-0.1949, -0.1376],
  [ 0.0006, -0.1126]],

 [[-0.1061,  0.0230],
  [-0.0938,  0.1393]]],


[[[-0.0883, -0.2143],
  [ 0.0252,  0.2117]],

 [[ 0.0179, -0.0074],
  [-0.1026, -0.1070]],
```

```
          [[-0.0488, -0.2463],
           [ 0.0623, -0.2159]],

          [[ 0.1133, -0.2122],
           [-0.1411,  0.0975]]],


         [[[-0.1297,  0.2443],
           [-0.0795, -0.0198]],

          [[-0.0014, -0.0174],
           [ 0.0297, -0.1768]],

          [[ 0.1411,  0.2431],
           [-0.1526,  0.2026]],

          [[ 0.0358,  0.0790],
           [-0.2351,  0.0078]]],


         [[[ 0.0891, -0.1433],
           [-0.2287, -0.2208]],

          [[ 0.1590, -0.2327],
           [-0.0806,  0.0681]],

          [[ 0.1365, -0.1808],
           [ 0.1257,  0.0140]],

          [[ 0.0243, -0.1252],
           [ 0.1053,  0.0787]]]])


Parameter: conv7.bias, Shape: torch.Size([16])
Values:
tensor([-0.1928,  0.1554, -0.2456,  0.1150,  0.0580,  0.2027, -0.0175,  0.1295,
        -0.1584, -0.0337,  0.2052,  0.1824, -0.1235,  0.0302, -0.0593,  0.1607])


Parameter: bn7.weight, Shape: torch.Size([16])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])


Parameter: bn7.bias, Shape: torch.Size([16])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
Parameter: conv8.weight, Shape: torch.Size([32, 16, 2, 2])
Values:
tensor([[[[ 1.1280e-01, -5.2593e-02],
          [-1.1609e-01, -5.8777e-02]],

         [[ 5.2258e-02, -1.0155e-04],
          [ 1.1952e-01,  9.3650e-02]],

         [[-4.4220e-02, -7.6557e-02],
          [-5.1574e-02,  7.6384e-02]],

         …,

         [[ 7.2023e-02, -6.2295e-02],
          [-7.7447e-02, -1.0506e-01]],

         [[ 6.9390e-02, -8.0843e-02],
          [-9.0433e-02,  8.5940e-02]],

         [[ 6.5370e-02,  7.7417e-02],
          [-6.2363e-02,  4.1309e-02]]],


        [[[-1.1692e-01,  7.7472e-02],
          [-7.7869e-03,  4.5505e-02]],

         [[-2.6251e-03, -7.3469e-02],
          [-4.8637e-02,  7.4574e-02]],

         [[-3.4029e-02, -6.6289e-02],
          [-6.6113e-02, -1.1643e-01]],

         …,

         [[-1.0676e-01,  7.4640e-02],
          [-1.1241e-01, -7.4916e-02]],

         [[-4.7767e-02, -1.1577e-01],
          [-6.2865e-02, -1.1264e-01]],

         [[ 2.3347e-02, -1.4986e-02],
          [-1.4783e-02,  2.9482e-03]]],


        [[[-4.4619e-02, -1.0601e-01],
          [ 4.3669e-02, -1.0844e-01]],
```

```
 [[ 5.6741e-02,  3.9506e-02],
  [-8.2560e-02,  3.7844e-02]],

 [[-3.5769e-03,  1.7048e-02],
  [-5.9970e-02, -4.8634e-02]],

 …,

 [[-1.1612e-01, -1.0835e-01],
  [-4.3397e-02,  4.9614e-02]],

 [[ 4.8310e-02,  1.6811e-02],
  [-1.0129e-01,  2.1608e-02]],

 [[ 1.0677e-01,  6.4486e-02],
  [ 5.2116e-02,  6.0106e-02]]],


…,


[[[ 4.5496e-02, -5.6733e-02],
  [-7.1043e-02,  3.9830e-02]],

 [[-4.2695e-02,  1.0915e-01],
  [-4.0224e-02, -1.1946e-01]],

 [[ 7.5964e-02, -6.5788e-02],
  [-1.7758e-02,  3.7943e-02]],

 …,

 [[ 6.2899e-02, -4.8216e-02],
  [ 6.7866e-02,  4.9915e-02]],

 [[ 1.7982e-02,  5.0470e-02],
  [ 1.1583e-01,  1.1016e-01]],

 [[ 7.4664e-02, -6.8089e-02],
  [-3.1619e-02, -4.8846e-02]]],


[[[-1.0006e-01,  4.9130e-02],
  [ 3.8260e-02,  1.5754e-02]],

 [[-9.1751e-02, -5.8711e-04],
  [-4.4704e-02,  6.6619e-03]],
```

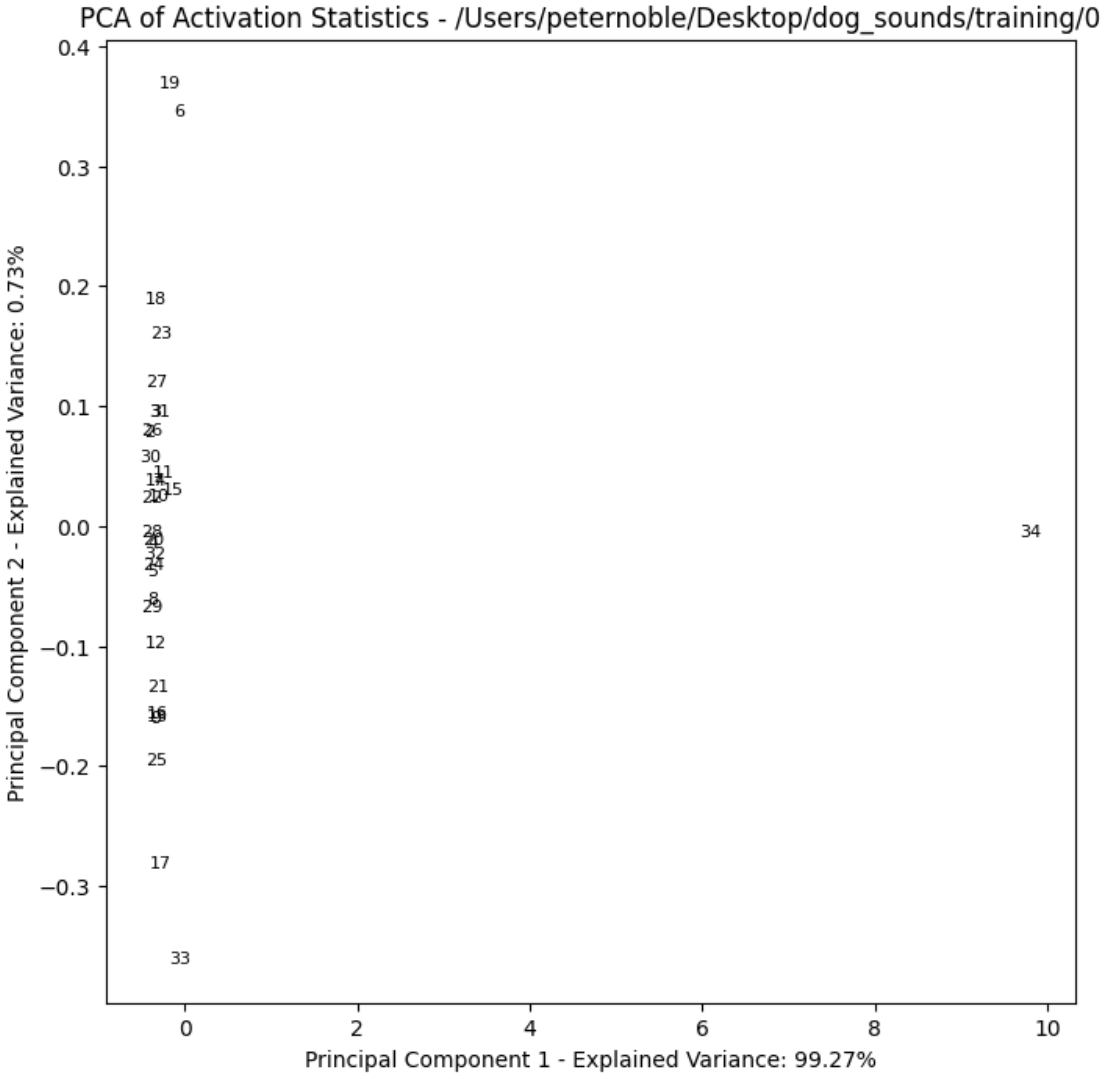
```
          [[-9.4467e-02, -9.8491e-02],
           [ 5.1666e-02,  8.9268e-02]],

          …,

          [[-3.0491e-02, -8.7127e-02],
           [ 1.0346e-01, -5.3048e-02]],

          [[-8.9332e-02, -3.2832e-02],
           [ 8.9348e-02, -4.8782e-02]],

          [[ 5.3209e-02, -3.2534e-02],
           [-1.0836e-01, -1.0179e-01]]],


         [[[-9.7108e-02,  2.6516e-02],
           [-1.0874e-02,  3.7890e-02]],

          [[ 7.4632e-03, -9.8909e-04],
           [ 6.8768e-02,  1.5045e-02]],

          [[ 1.1077e-01,  7.5691e-02],
           [ 8.9206e-02,  7.6032e-02]],

          …,

          [[-5.2698e-02, -3.9539e-03],
           [ 1.2392e-02,  1.2389e-02]],

          [[ 2.7753e-03,  5.7053e-02],
           [ 1.2266e-01, -7.7410e-03]],

          [[-2.7858e-02, -6.0071e-02],
           [-6.6844e-03, -6.3582e-02]]]])


Parameter: conv8.bias, Shape: torch.Size([32])
Values:
tensor([-0.0253, -0.0693,  0.0853,  0.0640, -0.0497, -0.0265,  0.0616, -0.0349,
         0.0508,  0.1229, -0.0942, -0.0379, -0.0829,  0.0190,  0.0947,  0.0906,
        -0.1139,  0.1186, -0.1201, -0.0389, -0.0823,  0.0021, -0.0324,  0.0809,
        -0.0470,  0.0364,  0.0802, -0.0522, -0.0928, -0.0831,  0.1237, -0.1230])


Parameter: bn8.weight, Shape: torch.Size([32])
Values:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

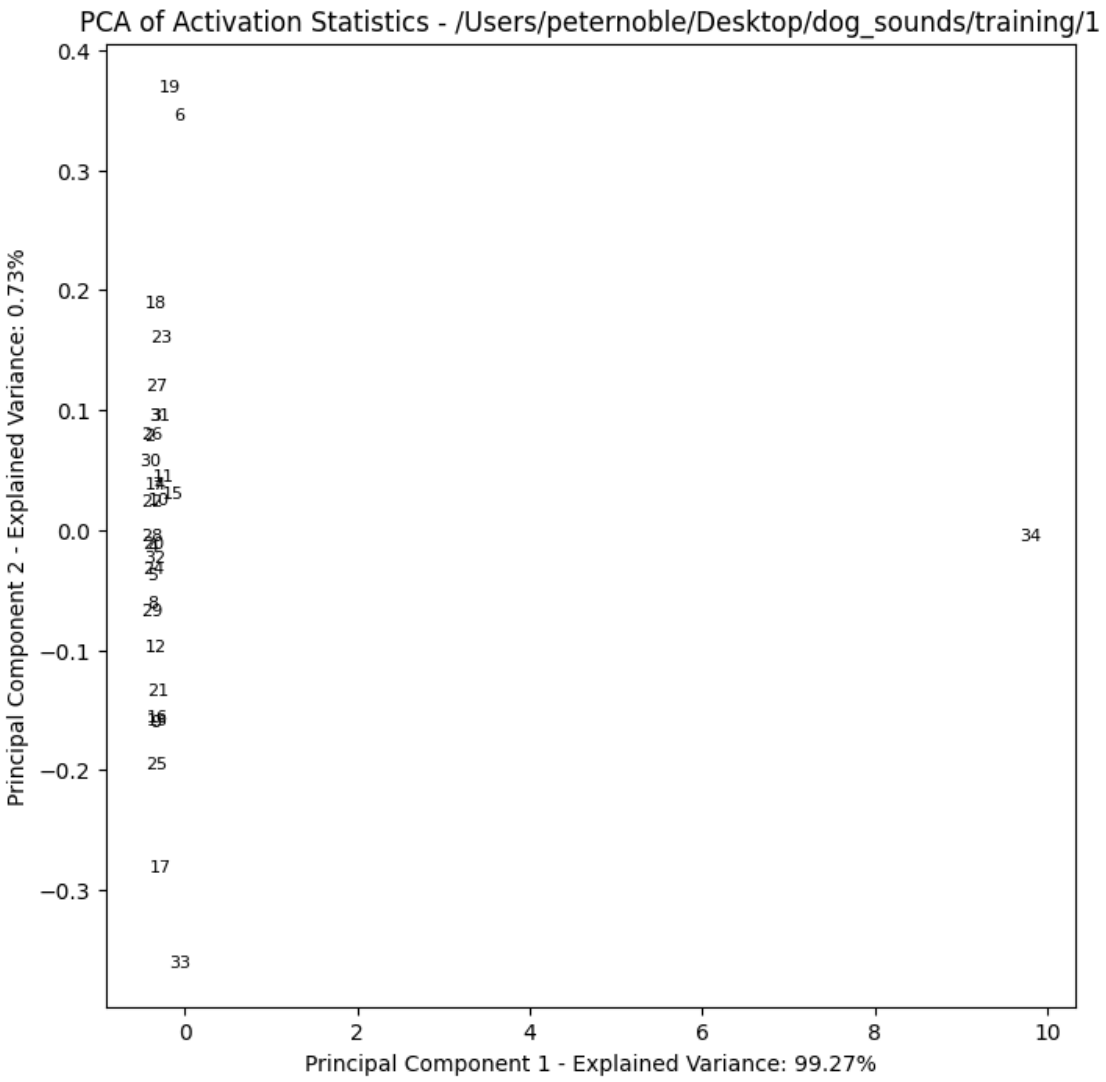
```
Parameter: bn8.bias, Shape: torch.Size([32])
Values:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0.])


Parameter: fc.weight, Shape: torch.Size([3, 32])
Values:
tensor([[ 0.1279, -0.0852, -0.0384, -0.1443, -0.1168,  0.1147, -0.0911, -0.0422,
         -0.1284, -0.1658,  0.0098, -0.1626,  0.1005,  0.1368,  0.1744,  0.1207,
         -0.1194, -0.0186,  0.1184,  0.0338, -0.1316, -0.1348, -0.0763,  0.0233,
         -0.1742,  0.1765, -0.0800, -0.0212, -0.1347, -0.1106, -0.1325,
-0.0279],
        [ 0.1639, -0.0667, -0.0638, -0.1626, -0.0790, -0.0536, -0.0444,  0.1027,
         -0.0711,  0.0073,  0.0908, -0.1124, -0.0091,  0.0547,  0.0101,  0.1431,
          0.0517,  0.1645, -0.0012, -0.0658,  0.1529,  0.0161, -0.0023,  0.0076,
         -0.0888,  0.0094,  0.0333, -0.1504,  0.1637,  0.0849,  0.0558,
0.0491],
        [-0.0427,  0.0627, -0.0581, -0.0607, -0.1728, -0.1463, -0.1696,  0.1424,
         -0.0516, -0.0263, -0.0831, -0.0285, -0.0265, -0.0447, -0.1740, -0.0067,
          0.1098,  0.0687,  0.0706,  0.1605,  0.0362,  0.1031, -0.0675,  0.1608,
         -0.1339, -0.0781, -0.0820, -0.0548, -0.0697,  0.1049, -0.0175,
-0.1188]])


Parameter: fc.bias, Shape: torch.Size([3])
Values:
tensor([ 0.1006, -0.1043,  0.1062])
```

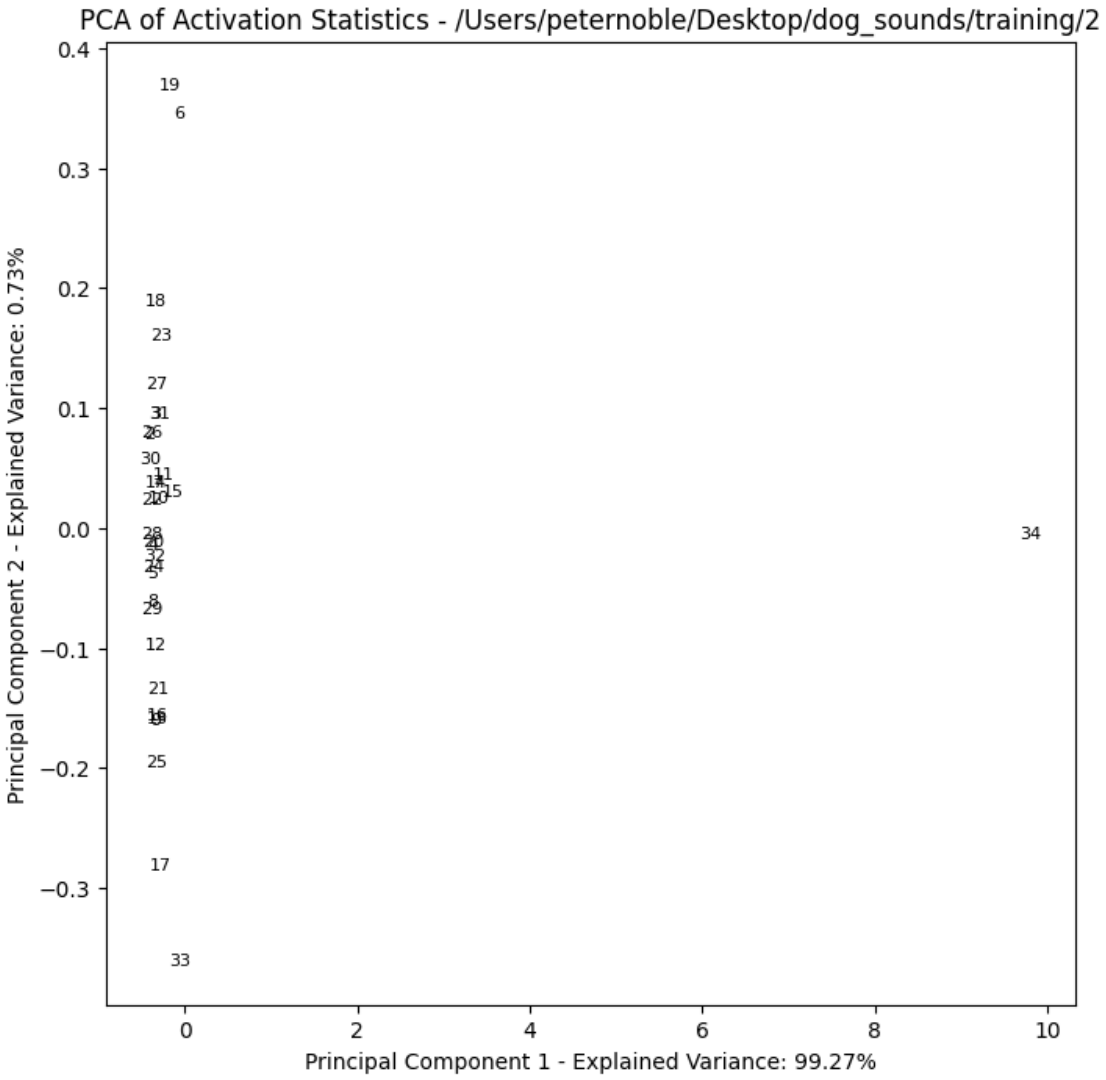
[11]:
```python
# CustomCNN class definition goes here
from sklearn.decomposition import PCA

# Function to visualize PCA results with labels and explained variance
def visualize_pca(mean_activations, var_activations, layer_numbers,␣
 ↪directory_path):
    # Combine mean and variance into a single matrix
    activations_matrix = list(zip(mean_activations, var_activations))

    # Apply PCA
    pca = PCA(n_components=2)
    activations_pca = pca.fit_transform(activations_matrix)
```

```
    # Get the contribution of explained variances
    contribution_pc1 = pca.explained_variance_ratio_[0] * 100
    contribution_pc2 = pca.explained_variance_ratio_[1] * 100

    # Visualize PCA results with dots labeled by layer numbers
    plt.figure(figsize=(8, 8))

    # Annotate explained variance on axes
 #   plt.text(activations_pca[:, 0].max() + 0.1, 0, f'PC1 ({contribution_pc1:.
 ↪2f}%)', fontsize=10, ha='left', va='center')
 #   plt.text(0, activations_pca[:, 1].max() + 0.1, f'PC2 ({contribution_pc2:.
 ↪2f}%)', fontsize=10, ha='center', va='bottom')

    # Label dots with layer numbers
    for i, (x, y) in enumerate(activations_pca):
        plt.scatter(x, y, alpha=0)  # Make the dot invisible
        plt.text(x, y, str(layer_numbers[i]), fontsize=8, ha='center',␣
 ↪va='center')

    plt.title(f'PCA of Activation Statistics - {directory_path}')
    plt.xlabel('Principal Component 1 - Explained Variance: {:.2f}%'.
 ↪format(contribution_pc1))
    plt.ylabel('Principal Component 2 - Explained Variance: {:.2f}%'.
 ↪format(contribution_pc2))
    plt.show()

# List of directory paths
directory_paths = [
    '/Users/peternoble/Desktop/dog_sounds/training/0',
    '/Users/peternoble/Desktop/dog_sounds/training/1',
    '/Users/peternoble/Desktop/dog_sounds/training/2',
    #'/Users/peternoble/Desktop/dog_sounds/training/3',
]

# Initialize lists to accumulate activation statistics for all datasets
all_mean_activations = []
all_var_activations = []

# Initialize layer numbers
layer_numbers = []

for directory_path in directory_paths:
    model = CustomCNN()

    # ... (rest of the code remains unchanged)

    # Accumulate activation statistics for the current dataset
```

```
    all_mean_activations.extend(mean_activations)
    all_var_activations.extend(var_activations)

    # Add layer numbers for the current dataset
    layer_numbers.extend([i + 2 for i in range(len(mean_activations))])

    # Visualize PCA of activation statistics for each dataset
    visualize_pca(mean_activations, var_activations, layer_numbers,␣
↪directory_path)
```




PCA of Activation Statistics - /Users/peternoble/Desktop/dog_sounds/training/1
Principal Component 2 - Explained Variance: 0.73%
Principal Component 1 - Explained Variance: 99.27%
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10
19
6
18
23
27
30
34
12
21
25
17
33



[12]:
```python
# Create a list to store the feature maps
feature_maps = []

# Define a hook to store the feature maps at each layer
def hook_fn(module, input, output):
    feature_maps.append(output)

# Register the hook to each layer in your model
hooks = []
for layer in model.children():
    hook = layer.register_forward_hook(hook_fn)
    hooks.append(hook)
```

```python
# Forward pass to obtain feature maps
with torch.no_grad():
    model(input_image)

# Remove the hooks
for hook in hooks:
    hook.remove()

# Print the shape of the feature maps
for i, feature_map in enumerate(feature_maps):
    print(f'Layer {i + 1} - Shape: {feature_map.shape}')

# Visualize the feature maps from the 2nd layer
layer_to_visualize = 1  # 0-based index, corresponds to the 2nd layer
feature_map_normalized = feature_maps[layer_to_visualize].squeeze().numpy()

# Visualize the feature map
#plt.figure(figsize=(8, 8))
#plt.imshow(feature_map_normalized, cmap='viridis')
#plt.title(f'Layer {layer_to_visualize + 1} - Feature Map')
#plt.axis('off')
#plt.show()
```

```
Layer 1 - Shape: torch.Size([1, 32, 245, 245])
Layer 2 - Shape: torch.Size([1, 32, 245, 245])
Layer 3 - Shape: torch.Size([1, 32, 245, 245])
Layer 4 - Shape: torch.Size([1, 32, 122, 122])
Layer 5 - Shape: torch.Size([1, 32, 121, 121])
Layer 6 - Shape: torch.Size([1, 32, 121, 121])
Layer 7 - Shape: torch.Size([1, 32, 121, 121])
Layer 8 - Shape: torch.Size([1, 32, 59, 59])
Layer 9 - Shape: torch.Size([1, 16, 60, 60])
Layer 10 - Shape: torch.Size([1, 16, 60, 60])
Layer 11 - Shape: torch.Size([1, 16, 60, 60])
Layer 12 - Shape: torch.Size([1, 16, 30, 30])
Layer 13 - Shape: torch.Size([1, 8, 31, 31])
Layer 14 - Shape: torch.Size([1, 8, 31, 31])
Layer 15 - Shape: torch.Size([1, 8, 31, 31])
Layer 16 - Shape: torch.Size([1, 8, 15, 15])
Layer 17 - Shape: torch.Size([1, 4, 16, 16])
Layer 18 - Shape: torch.Size([1, 4, 16, 16])
Layer 19 - Shape: torch.Size([1, 4, 16, 16])
Layer 20 - Shape: torch.Size([1, 4, 8, 8])
Layer 21 - Shape: torch.Size([1, 4, 9, 9])
Layer 22 - Shape: torch.Size([1, 4, 9, 9])
Layer 23 - Shape: torch.Size([1, 4, 9, 9])
Layer 24 - Shape: torch.Size([1, 4, 4, 4])
Layer 25 - Shape: torch.Size([1, 16, 5, 5])
```

```
Layer 26 - Shape: torch.Size([1, 16, 5, 5])
Layer 27 - Shape: torch.Size([1, 16, 5, 5])
Layer 28 - Shape: torch.Size([1, 16, 2, 2])
Layer 29 - Shape: torch.Size([1, 32, 3, 3])
Layer 30 - Shape: torch.Size([1, 32, 3, 3])
Layer 31 - Shape: torch.Size([1, 32, 3, 3])
Layer 32 - Shape: torch.Size([1, 32, 1, 1])
Layer 33 - Shape: torch.Size([1, 3])
```

[2]:
```python
import torch

print(torch.__version__)
```

```
2.2.1
```

[ ]:
```python

```