Submission instructions

Submission must be in pairs, unless otherwise authorized.

Submit by 28/2/2024

- This notebook contains all the questions. You should follow the instructions below.
- Solutions for both theoretical and practical parts should be written in this notebook

Moodle submission

You should submit three files:

- IPYNB notebook:
  - All the wet and dry parts, including code, graphs, discussion, etc.
- PDF file:
  - Export the notebook to PDF. Make sure that all the cells are visible.
- Pickle file:
  - As requested in Q2.a

All files should be in the following format: "HW1_ID1_ID2.file"
Good Luck!

Question 1

I. Softmax Derivative (10pt)

Derive the gradients of the softmax function and demonstrate how the expression can be reformulated solely by using the softmax function, i.e., in some expression where only $softmax(x)$, but not x, is present). Recall that the softmax function is defined as follows:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

I. Softmax Derivative - Answer:

$$\frac{\partial softmax(x)_i}{\partial x_k} = \frac{\partial \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}}{\partial x_k}$$

$$when\ i = j$$

$$\frac{\partial softmax(x)_i}{\partial x_k} = softmax(x)_i \cdot (1 - softmax(x)_i)$$

$$when\ i \neq j$$

$$\frac{\partial softmax(x)_i}{\partial x_k} = -softmax(x)_i \cdot softmax(x)_k$$

II. Cross-Entropy Gradient (10pt)

Derive the gradient of cross-entropy loss with regard to the inputs of a softmax function. i.e., find the gradients with respect to the softmax input vector θ, when the prediction is denoted by $\hat{y} = softmax(\theta)$.

where y is the one-hot label vector, and $\hat{y}$ is the predicted probability vector for all classes.

$$\hat{y} = softmax(\theta)$$

Remember the cross entropy function is:

$$CE(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i)$$

II. Cross-Entropy Gradient - Answer

$$\frac{\partial CE(y,\hat{y})}{\partial \theta} = \frac{\partial CE(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} = \frac{\partial - \sum_i y_i log(\hat{y}_i)}{\partial \theta}$$

$$\begin{aligned}
\frac{\partial CE}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k} \sum_{j=1}^{n} (-y_j \log(\sigma(\theta_j))) \\
&= -\sum_{j=1}^{n} y_j \frac{\partial}{\partial \theta_k} \log(\sigma(\theta_j)) && \text{...addition rule, } -y_j \text{ is constant} \\
&= -\sum_{j=1}^{n} y_j \frac{1}{\sigma(\theta_j)} \frac{\partial}{\partial \theta_k} \sigma(\theta_j) && \text{...chain rule} \\
&= -y_k \frac{\sigma(\theta_k)(1-\sigma(\theta_k))}{\sigma(\theta_k)} + \sum_{j \neq k} y_j \frac{\sigma(\theta_k)\sigma(\theta_j)}{\sigma(\theta_j)} && \text{...consider both } j = k \text{ and } j \neq k \\
&= -y_k(1-\sigma(\theta_k)) + \sum_{j \neq k} y_j \sigma(\theta_k) \\
&= -y_k + y_k \sigma(\theta_k) + \sum_{j \neq k} y_j \sigma(\theta_k) \\
&= -y_k + \sigma(\theta_k) \sum_j y_j.
\end{aligned}$$

$$\Rightarrow \frac{\partial CE}{\partial \theta_k} = \sigma(\theta_k) - y_k$$

Question 2

I. Derivative Of Activation Functions (10pt)

The following cell contains an implementation of some activation functions. Implement the corresponding derivatives.

In [3]:
```
import torch

def sigmoid(x):
    return 1 / (1 + torch.exp(-x))

def tanh(x):
    return torch.div(torch.exp(x) - torch.exp(-x), torch.exp(x) + torch.exp(-x))


def softmax(x):
    exp_x = torch.exp(x.T - torch.max(x, dim=-1).values).T  # Subtracting max(x) for numerical stability
    return exp_x / exp_x.sum(dim=-1, keepdim=True)
```

```
c:\Users\hadar\AppData\Local\Programs\Python\Python37\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Pleas
e update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [4]:
```
def d_sigmoid(x):
    return sigmoid(x)*(1-sigmoid(x))


def d_tanh(x):
    return 1 - (tanh(x)**2)


def d_softmax(x):
    """_summary_

    Args:
```

```
        x (_type_): _description_

    Returns:
        _type_: _description_
    """
    s = softmax(x)
    batch_size, n_classes = s.shape
    # Initialize the Jacobian matrix for each sample in the batch
    jacobian_m = torch.zeros((batch_size, n_classes, n_classes))

    for i in range(batch_size):
        for j in range(n_classes):
            for k in range(n_classes):
                if j == k:
                    jacobian_m[i, j, k] = s[i, j] * (1 - s[i, j])
                else:
                    jacobian_m[i, j, k] = -s[i, j] * s[i, k]
    return jacobian_m


def cross_entropy_derivative(y_hat, y):
    return y_hat - y
```

II. Train a Fully Connected network on MNIST (30pt)

In the following exercise, you will create a classifier for the MNIST dataset. You should write your own training and evaluation code and meet the following constraints:

- You are only allowed to use torch tensor manipulations.
- You are NOT allowed to use:
  - Auto-differentiation - backward()
  - Built-in loss functions
  - Built-in activations
  - Built-in optimization
  - Built-in layers (torch.nn)

       </h4>

a) The required classifier class is defined.

- You should implement the backward pass of the model.
- Train the model and plot the model's accuracy and loss (both on train and test sets) as a function of the epochs.
- You should save the model's weights and biases. Change the student_ids to yours.

In this section, you **must** use the "set_seed" function with the given seed and **sigmoid** as an activation function.

In [6]:
```
import torch
import torchvision
from torch.utils.data import DataLoader

import os
import matplotlib.pyplot as plt
import seaborn as sns; sns.set_theme()
import torch.nn.functional as F

# Constants
SEED = 42
EPOCHS = 16
BATCH_SIZE = 32
NUM_OF_CLASSES = 10

# Setting seed
def set_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    os.environ["PYTHONHASHSEED"] = str(seed)


# Transformation for the data
transform = torchvision.transforms.Compose(
    [torchvision.transforms.ToTensor(),
     torch.flatten])

# Cross-Entropy loss implementation
def one_hot(y, num_of_classes=10):
```

```
    hot = torch.zeros((y.size()[0], num_of_classes))
    hot[torch.arange(y.size()[0]), y] = 1
    return hot

def cross_entropy(y, y_hat):
    return -torch.sum(one_hot(y) * torch.log(y_hat)) / y.size()[0]
```

In [7]:
```
# Create dataloaders
train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
                                            download=True, transform=transform)
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE)


test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
                                           download=True, transform=transform)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE)
```

In [8]:
```
class FullyConnectedNetwork:
    def __init__(self, input_size, output_size, hidden_size1, activation_func = sigmoid, lr=0.01):
        # parameters
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size1 = hidden_size1

        # activation function
        self.activation_func = activation_func

        # weights
        self.W1 = torch.randn(self.input_size, self.hidden_size1)
        self.b1 = torch.zeros(self.hidden_size1)

        self.W2 = torch.randn(self.hidden_size1, self.output_size)
        self.b2 = torch.zeros(self.output_size)

        self.lr = lr

    def forward(self, x):
        self.z1 = torch.matmul(x, self.W1) + self.b1
        self.h1 = self.activation_func(self.z1)
        self.z2 = torch.matmul(self.h1, self.W2) + self.b2
        self.y_hat = softmax(self.z2)
        return self.y_hat
```

```
    def backward(self, x, y, y_hat):
        # Ensure y is one-hot encoded to match y_hat's shape
        # Assuming y is not one-hot encoded, convert it using torch.nn.functional.one_hot
        y_one_hot = torch.nn.functional.one_hot(y, num_classes=self.output_size).to(torch.float32)
        lr = self.lr
        batch_size = y.size(0)
        # Simplified derivative for cross-entropy with softmax
        dl_dz2 =  cross_entropy_derivative(y_hat=y_hat,y=y_one_hot)
        dl_dW2 = torch.matmul(torch.t(self.h1), dl_dz2)
        dl_db2 = torch.matmul(torch.t(dl_dz2), torch.ones(batch_size))

        dl_dh = torch.matmul(dl_dz2, torch.t(self.W2))
        dl_dz1 = dl_dh * d_sigmoid(self.z1)

        dl_dW1 = torch.matmul(torch.t(x), dl_dz1)
        dl_db1 = torch.matmul(torch.t(dl_dz1), torch.ones(batch_size))


        #gradient step
        self.W1 -= lr*dl_dW1
        self.b1 -= lr*dl_db1
        self.W2 -= lr*dl_dW2
        self.b2 -= lr*dl_db2

    def train(self, X, y):
        # forward + backward pass for trainig a model
        o = self.forward(X)
        self.backward(X, y, o)
```

In [9]:
```
set_seed(SEED)
model = FullyConnectedNetwork(784, 10, 128, sigmoid, lr=0.01)
```

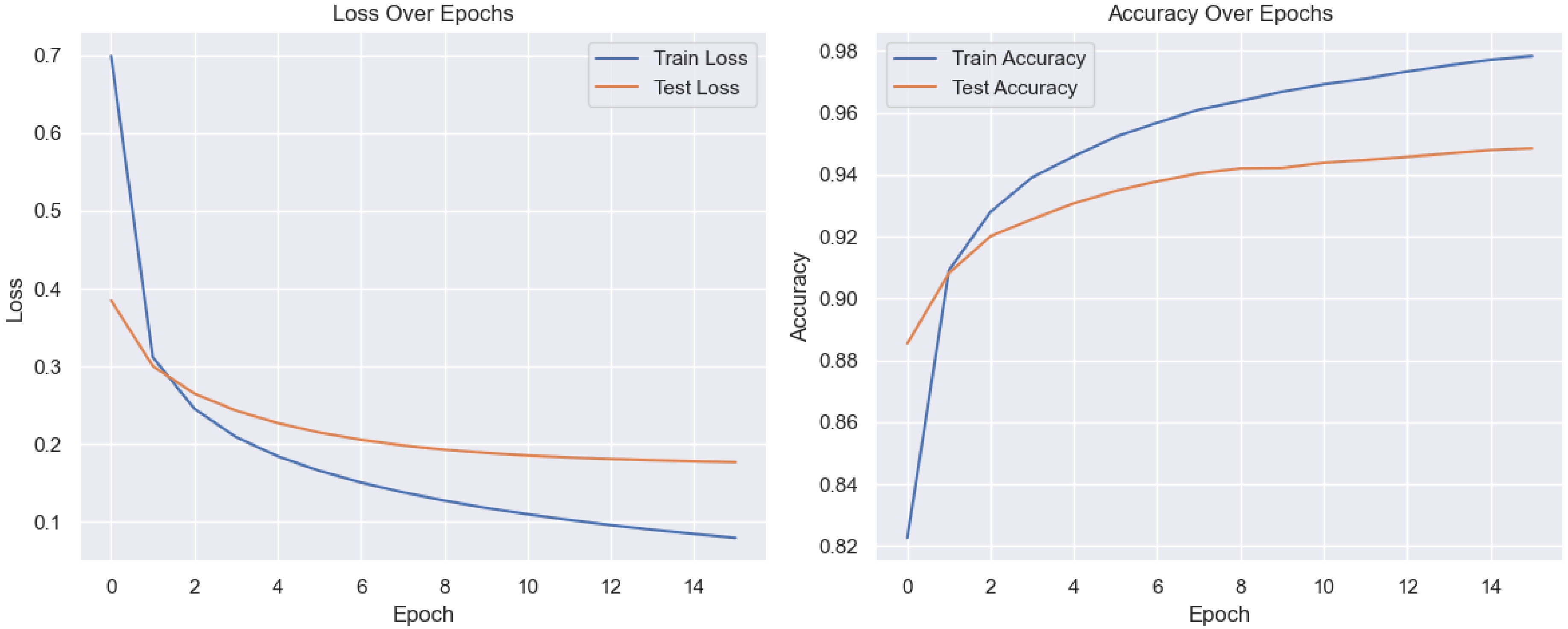
In [10]:
```
# Initialize history lists for tracking progress
history = {
    'train_loss': [],
    'train_accuracy': [],
    'test_loss': [],
    'test_accuracy': []
}

# Function to calculate metrics for a given dataloader
def calculate_metrics(dataloader, mode='train'):
```

```
    total_loss, total_correct, total_samples = 0, 0, 0
    for X_batch, y_batch in dataloader:
        y_hat = model.forward(x=X_batch)
        loss = cross_entropy(y=y_batch, y_hat=y_hat)
        _, predicted = torch.max(y_hat, 1)

        # Check if `y_batch` is one-hot encoded and convert if necessary
        if y_batch.ndimension() > 1:
            y_batch = y_batch.argmax(dim=1)

        # Calculate the number of correct predictions
        correct = (predicted == y_batch).sum().item()


        # Accumulate batch results
        total_loss += loss * len(y_batch)
        total_correct += correct
        total_samples += X_batch.size(0)

        # Backpropagation for training mode
        if mode == 'train':
            model.backward(x=X_batch, y=y_batch, y_hat=y_hat)

    # Calculate and store epoch metrics
    history[f'{mode}_loss'].append(total_loss / total_samples)
    history[f'{mode}_accuracy'].append(total_correct / total_samples)

# Function to plot the training and testing loss and accuracy
def plot_metrics(history):
    plt.figure(figsize=(12, 5))

    # Plot training and testing loss
    plt.subplot(1, 2, 1)
    plt.plot(history['train_loss'], label='Train Loss')
    plt.plot(history['test_loss'], label='Test Loss')
    plt.title('Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Plot training and testing accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history['train_accuracy'], label='Train Accuracy')
    plt.plot(history['test_accuracy'], label='Test Accuracy')
```

```
    plt.title('Accuracy Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()


# Training and testing the model
for epoch in range(EPOCHS):
    print(f'Epoch: {epoch+1} ({EPOCHS - (epoch+1)} to go)')

    calculate_metrics(train_dataloader, 'train')
    calculate_metrics(test_dataloader, 'test')
    print('\n')


plot_metrics(history)
```

```
Epoch: 1 (15 to go)

Epoch: 2 (14 to go)

Epoch: 3 (13 to go)

Epoch: 4 (12 to go)

Epoch: 5 (11 to go)

Epoch: 6 (10 to go)

Epoch: 7 (9 to go)

Epoch: 8 (8 to go)

Epoch: 9 (7 to go)

Epoch: 10 (6 to go)

Epoch: 11 (5 to go)

Epoch: 12 (4 to go)

Epoch: 13 (3 to go)

Epoch: 14 (2 to go)

Epoch: 15 (1 to go)
```

```
Epoch: 16 (0 to go)
```



In [11]:
```
students_ids = "318880754_206567067"
torch.save({"W1": model.W1, "W2": model.W2, "b1": model.b1, "b2": model.b2}, f"HW1_{students_ids}.pkl")
```

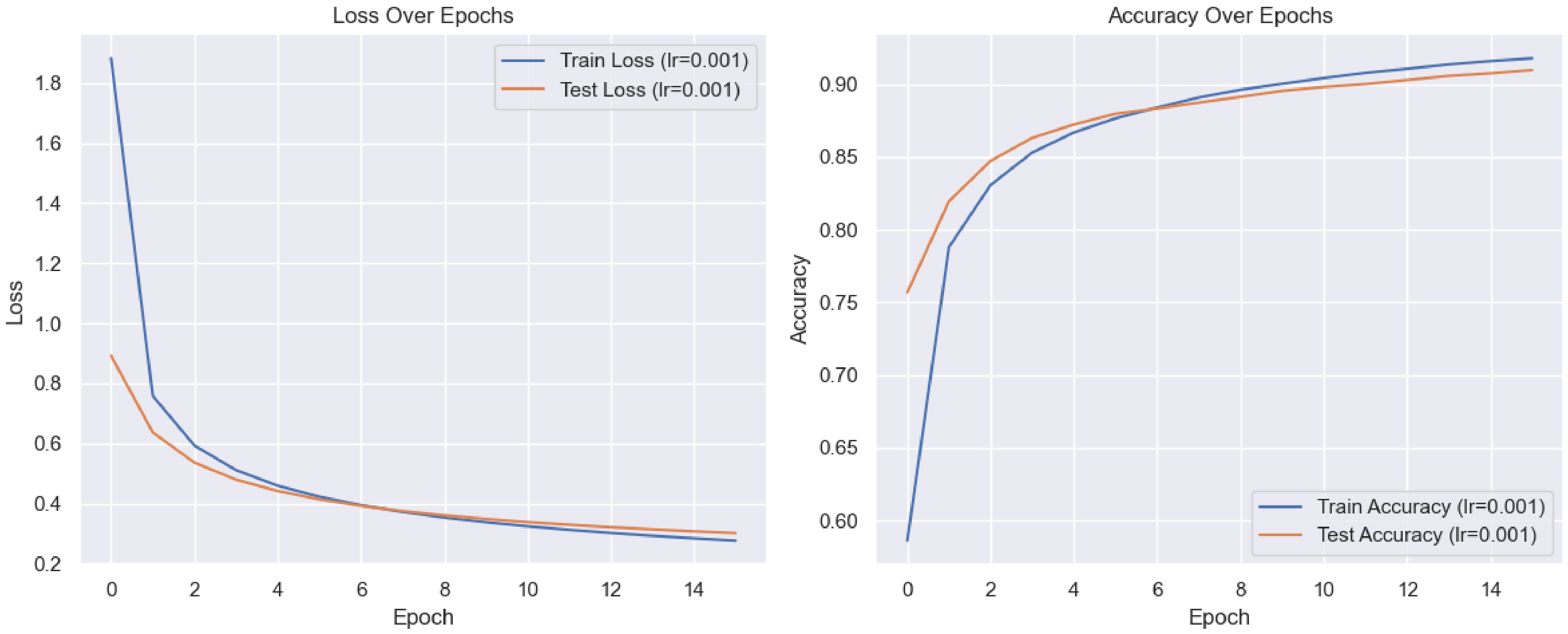
b) Train the model with various learning rates (at least 3).

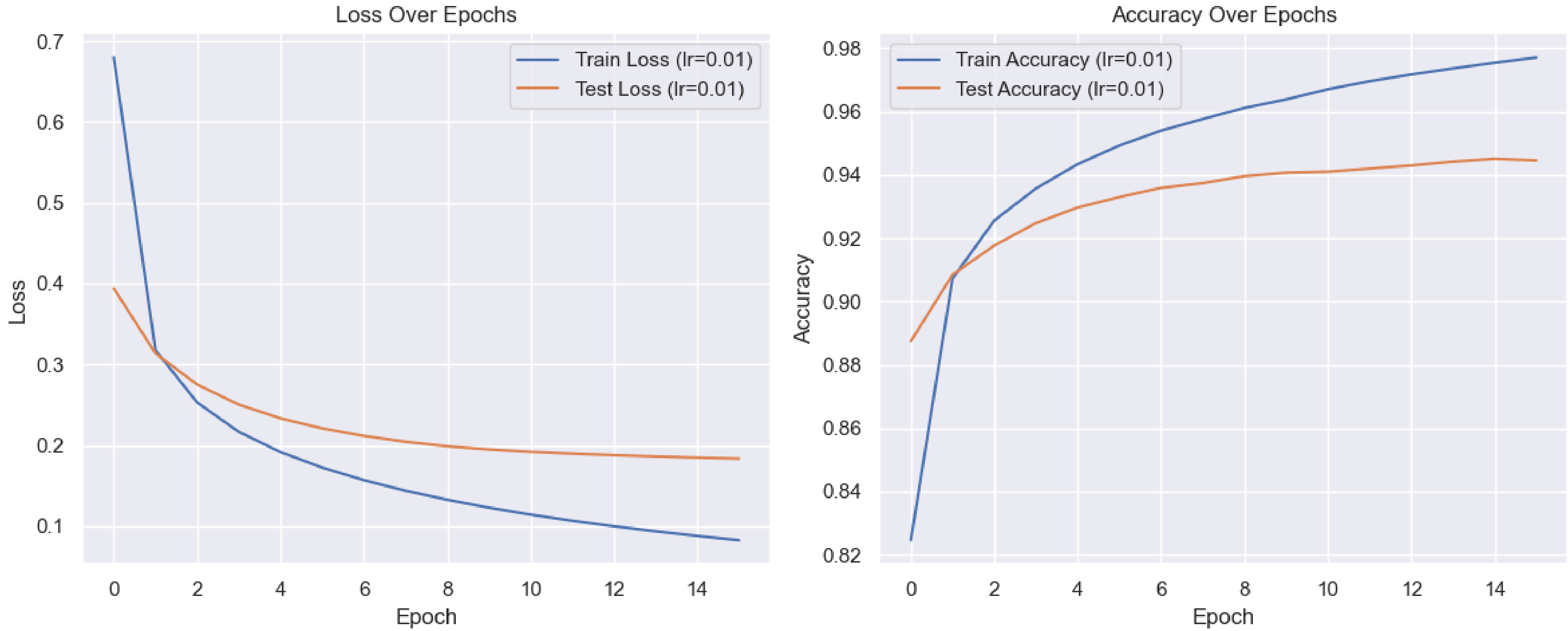
- Plot the model's accuracy and loss (both on train and test sets) as a function of the epochs.
- Discuss the differences in training with different learning rates. Support your answer with plots.

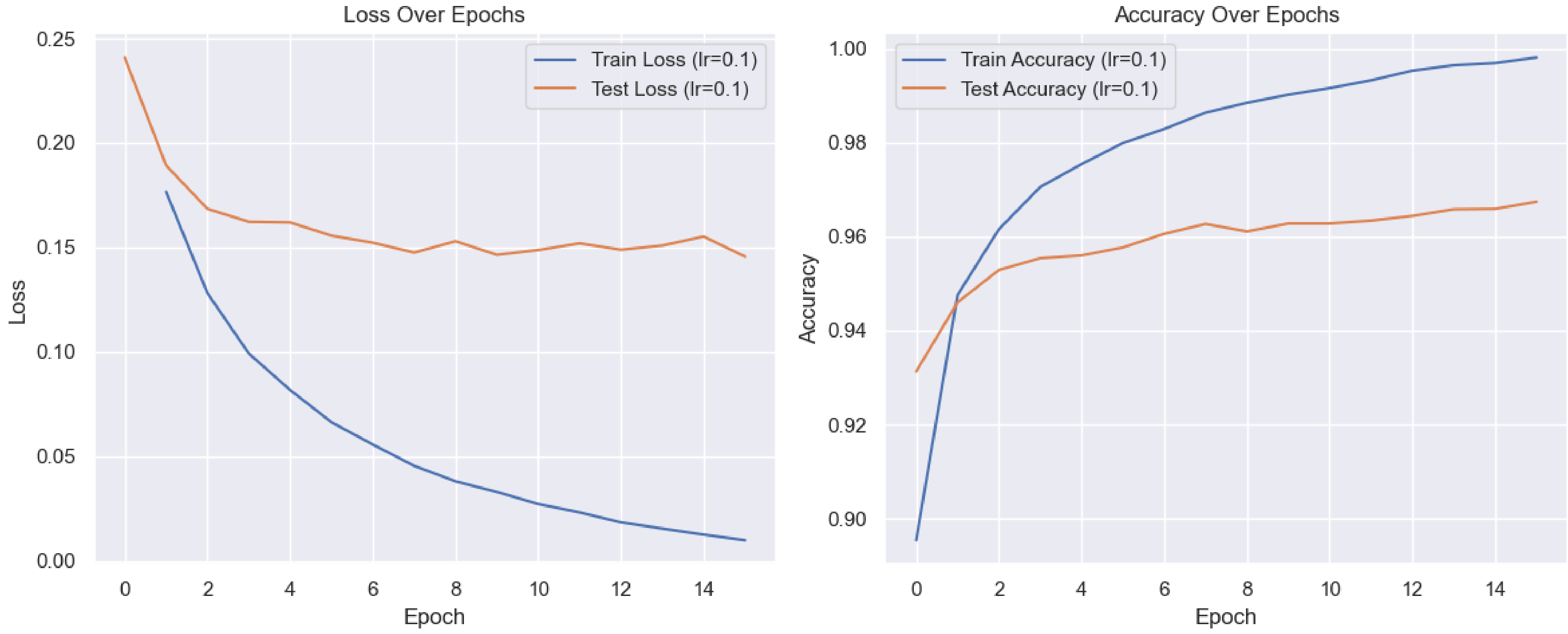
In [12]:
```
# Define the learning rates to test
learning_rates = [0.001, 0.01, 0.1]
```

```python
# Define a dictionary to hold the history for each learning rate
histories = {}

# Loop over each learning rate
for lr in learning_rates:
    # Initialize the model with the current learning rate
    model = FullyConnectedNetwork(input_size=784, output_size=10, hidden_size1=128, lr=lr)
    # Initialize the history
    history = {
        'train_loss': [],
        'train_accuracy': [],
        'test_loss': [],
        'test_accuracy': []
    }

    # Train the model
    for epoch in range(EPOCHS):
        calculate_metrics(train_dataloader, 'train')
        calculate_metrics(test_dataloader, 'test')

    # Save the history for this learning rate
    histories[lr] = history

# Now plot the accuracy and loss for each learning rate
for lr, history in histories.items():
    plt.figure(figsize=(12, 5))

    # Plot training and testing loss
    plt.subplot(1, 2, 1)
    plt.plot(history['train_loss'], label=f'Train Loss (lr={lr})')
    plt.plot(history['test_loss'], label=f'Test Loss (lr={lr})')
    plt.title('Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Plot training and testing accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history['train_accuracy'], label=f'Train Accuracy (lr={lr})')
    plt.plot(history['test_accuracy'], label=f'Test Accuracy (lr={lr})')
    plt.title('Accuracy Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
```

```
    plt.legend()

    plt.tight_layout()
    plt.show()
```




Loss Over Epochs
Train Loss (lr=0.01)
Test Loss (lr=0.01)
Loss
Epoch
Accuracy Over Epochs
Train Accuracy (lr=0.01)
Test Accuracy (lr=0.01)
Accuracy
Epoch



The plots demonstrate how different learning rates impact model training and generalization. A lower learning rate of 0.001 results in a steady but slow convergence, indicating careful updates that prevent overshooting the optimal solution with minimal overfitting, as evidenced by the close test and training loss. A tenfold increase to 0.01 accelerates convergence but starts to show signs of overfitting with a larger gap between training and test accuracy. A further increase to 0.1 leads to rapid initial improvements but ultimately results in poor generalization and potential overfitting, as the test loss plateaus and the test accuracy remains lower than for smaller learning rates. These observations underscore the trade-off between convergence speed and generalization when selecting an appropriate learning rate for neural network training.

Question 3

I. Implement and Train a CNN (30pt)

You are a data scientist at a supermarket. Your manager asked you to write a new image classifiaction algorithem for the self checkout cashiers. The images are of products from your grocery store (dataset files are attched in the Moodle). Your code and meet the following constraints:

- Your classifier must be CNN based
- You are not allowed to use any pre-trained model

  In order to satisfy your boss you have to reach 65% accuracy on the test set. You will get a bonus for your salary (and 10 points to your grade) if your model's number of paramters is less than 100K. You can reutilize code from the tutorials.

- Train the model and plot the model's accuracy and loss (both on train and validation sets) as a function of the epochs.
- Report the test set accurecy.
- Discus the progress you made and describe your final model.

In [80]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
import pandas as pd
import torch.nn.functional as F
import os
from torchvision.io import read_image
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
# Load the CSV file to inspect its content
classes_df = pd.read_csv('GroceryStoreDataset/classes.csv')
num_classes = classes_df['Coarse Class ID (int)'].nunique()
```

```
class GroceryModel(nn.Module):
    def __init__(self, num_classes=num_classes,drop_prob=0.5):
        super(GroceryModel, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.layer2 = self._make_layer(64, 128, 2, stride=1)
        self.layer3 = self._make_layer(128, 256, 2, stride=2)
        self.layer4 = self._make_layer(256, 512, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
        self.dropout = nn.Dropout(0.5)

    def _make_layer(self, in_channels, out_channels, blocks, stride):
        layers = []
        layers.append(nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1))
        layers.append(nn.BatchNorm2d(out_channels))
        layers.append(nn.ReLU())
        for _ in range(1, blocks):
            layers.append(nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1))
            layers.append(nn.BatchNorm2d(out_channels))
            layers.append(nn.ReLU())
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

In [97]:

```
class GroceryStoreDataset(Dataset):
    def __init__(self, annotations_file, root_dir, transform=None):
        with open(annotations_file, 'r') as file:
            self.img_labels = [line.strip().split(', ') for line in file.readlines()]
```

```
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        # Corrected to access list elements by index
        img_path = os.path.join(self.root_dir, self.img_labels[idx][0])
        image = Image.open(img_path).convert('RGB')  # Load as PIL Image and convert to RGB
        label = int(self.img_labels[idx][1])  # Convert label to integer
        if self.transform:
            image = self.transform(image)
        return image, label


# Define your transformations
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),  # Now it's okay to convert from PIL Image to Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Create dataset instances
train_dataset = GroceryStoreDataset(
    annotations_file='GroceryStoreDataset/train.txt',
    root_dir='GroceryStoreDataset',
    transform=transform
)

test_dataset = GroceryStoreDataset(
    annotations_file='GroceryStoreDataset/test.txt',
    root_dir='GroceryStoreDataset',
    transform=transform
)
# Create validation dataset instance
validation_dataset = GroceryStoreDataset(
    annotations_file='GroceryStoreDataset/val.txt',  # Make sure this path is correct
    root_dir='GroceryStoreDataset',  # Adjust if necessary
    transform=transform
)

# Create data loaders
```

```
validation_loader = DataLoader(validation_dataset, batch_size=128, shuffle=False)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

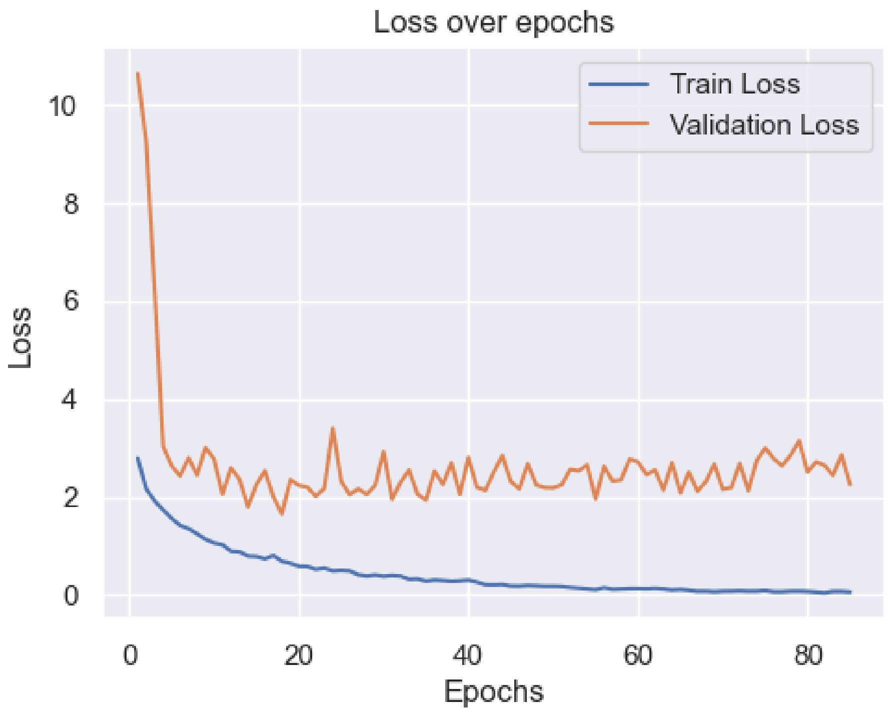
In [103…

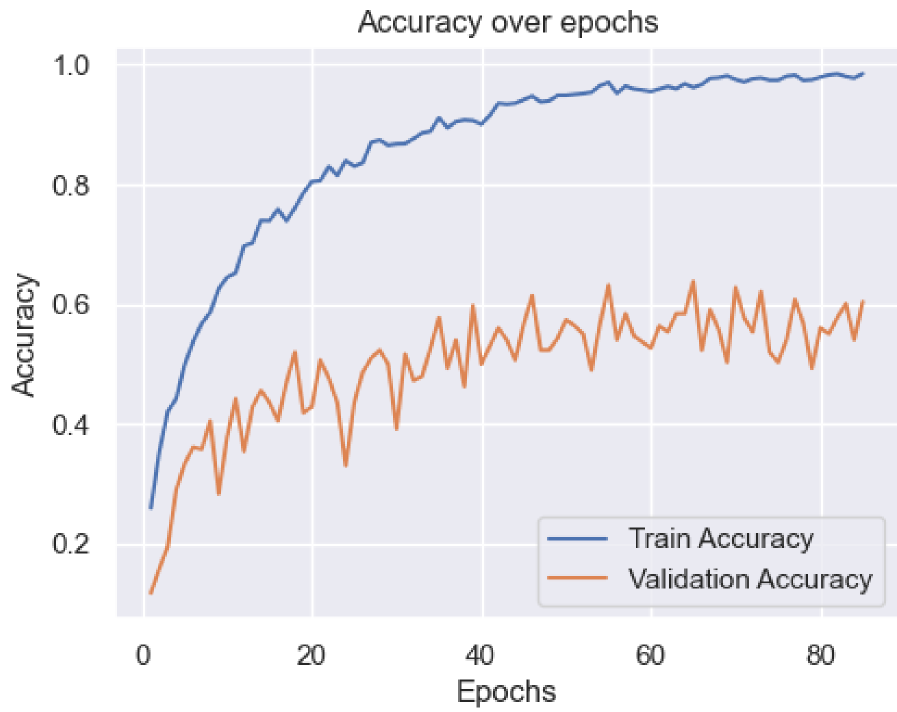
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = GroceryModel(num_classes=num_classes).to(device)
# optimizer = optim.Adam(model.parameters(), lr=0.09)
optimizer = optim.SGD(model.parameters(), lr=0.07, momentum=0.9)
criterion = nn.CrossEntropyLoss()
total_params = sum(p.numel() for p in model.parameters())
print(f'Total number of parameters: {total_params}')

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []
EPOCHS = 85

for epoch in range(EPOCHS):
    model.train()
    train_loss = 0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        correct_train += (predicted == labels).sum().item()
        total_train += labels.size(0)

    avg_train_loss = train_loss / total_train
    train_accuracy = correct_train / total_train
    train_losses.append(avg_train_loss)
    train_accuracies.append(train_accuracy)

    # Validation phase
    model.eval()
```

```python
    val_loss = 0
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for images, labels in validation_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs.data, 1)
            correct_val += (predicted == labels).sum().item()
            total_val += labels.size(0)

    avg_val_loss = val_loss / total_val
    val_accuracy = correct_val / total_val
    val_losses.append(avg_val_loss)
    val_accuracies.append(val_accuracy)

    print(f'Epoch {epoch+1}/{EPOCHS}, '
          f'Train Loss: {avg_train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, '
          f'Validation Loss: {avg_val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}')

# Plotting the training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(range(1, EPOCHS+1), train_losses, label='Train Loss')
plt.plot(range(1, EPOCHS+1), val_losses, label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plotting the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, EPOCHS+1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, EPOCHS+1), val_accuracies, label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Total number of parameters: 4681899
Epoch 1/85, Train Loss: 2.7922, Train Accuracy: 0.2602, Validation Loss: 10.6347, Validation Accuracy: 0.1182
Epoch 2/85, Train Loss: 2.1663, Train Accuracy: 0.3538, Validation Loss: 9.2328, Validation Accuracy: 0.1588
Epoch 3/85, Train Loss: 1.9211, Train Accuracy: 0.4212, Validation Loss: 6.2377, Validation Accuracy: 0.1959
Epoch 4/85, Train Loss: 1.7376, Train Accuracy: 0.4428, Validation Loss: 3.0319, Validation Accuracy: 0.2905
Epoch 5/85, Train Loss: 1.5695, Train Accuracy: 0.5000, Validation Loss: 2.6398, Validation Accuracy: 0.3345
Epoch 6/85, Train Loss: 1.4199, Train Accuracy: 0.5379, Validation Loss: 2.4291, Validation Accuracy: 0.3615
Epoch 7/85, Train Loss: 1.3546, Train Accuracy: 0.5682, Validation Loss: 2.7968, Validation Accuracy: 0.3581
Epoch 8/85, Train Loss: 1.2493, Train Accuracy: 0.5864, Validation Loss: 2.4475, Validation Accuracy: 0.4054
Epoch 9/85, Train Loss: 1.1402, Train Accuracy: 0.6261, Validation Loss: 3.0114, Validation Accuracy: 0.2838
Epoch 10/85, Train Loss: 1.0630, Train Accuracy: 0.6451, Validation Loss: 2.7731, Validation Accuracy: 0.3784
Epoch 11/85, Train Loss: 1.0269, Train Accuracy: 0.6527, Validation Loss: 2.0618, Validation Accuracy: 0.4426
Epoch 12/85, Train Loss: 0.8944, Train Accuracy: 0.6977, Validation Loss: 2.5953, Validation Accuracy: 0.3547
Epoch 13/85, Train Loss: 0.8847, Train Accuracy: 0.7027, Validation Loss: 2.3631, Validation Accuracy: 0.4291
Epoch 14/85, Train Loss: 0.7975, Train Accuracy: 0.7402, Validation Loss: 1.7976, Validation Accuracy: 0.4561
Epoch 15/85, Train Loss: 0.7880, Train Accuracy: 0.7398, Validation Loss: 2.2572, Validation Accuracy: 0.4358
Epoch 16/85, Train Loss: 0.7395, Train Accuracy: 0.7587, Validation Loss: 2.5334, Validation Accuracy: 0.4054
Epoch 17/85, Train Loss: 0.8078, Train Accuracy: 0.7394, Validation Loss: 2.0050, Validation Accuracy: 0.4696
Epoch 18/85, Train Loss: 0.6909, Train Accuracy: 0.7610, Validation Loss: 1.6525, Validation Accuracy: 0.5203
Epoch 19/85, Train Loss: 0.6491, Train Accuracy: 0.7860, Validation Loss: 2.3539, Validation Accuracy: 0.4189
Epoch 20/85, Train Loss: 0.5910, Train Accuracy: 0.8049, Validation Loss: 2.2367, Validation Accuracy: 0.4291
Epoch 21/85, Train Loss: 0.5859, Train Accuracy: 0.8064, Validation Loss: 2.2021, Validation Accuracy: 0.5068
Epoch 22/85, Train Loss: 0.5263, Train Accuracy: 0.8303, Validation Loss: 2.0170, Validation Accuracy: 0.4764
Epoch 23/85, Train Loss: 0.5520, Train Accuracy: 0.8152, Validation Loss: 2.1740, Validation Accuracy: 0.4358
Epoch 24/85, Train Loss: 0.4961, Train Accuracy: 0.8402, Validation Loss: 3.4034, Validation Accuracy: 0.3311
Epoch 25/85, Train Loss: 0.5070, Train Accuracy: 0.8303, Validation Loss: 2.3186, Validation Accuracy: 0.4358
Epoch 26/85, Train Loss: 0.4954, Train Accuracy: 0.8364, Validation Loss: 2.0484, Validation Accuracy: 0.4865
Epoch 27/85, Train Loss: 0.4163, Train Accuracy: 0.8705, Validation Loss: 2.1679, Validation Accuracy: 0.5101
Epoch 28/85, Train Loss: 0.3891, Train Accuracy: 0.8742, Validation Loss: 2.0549, Validation Accuracy: 0.5236
Epoch 29/85, Train Loss: 0.4098, Train Accuracy: 0.8655, Validation Loss: 2.2373, Validation Accuracy: 0.5000
Epoch 30/85, Train Loss: 0.3851, Train Accuracy: 0.8678, Validation Loss: 2.9254, Validation Accuracy: 0.3919
Epoch 31/85, Train Loss: 0.4010, Train Accuracy: 0.8682, Validation Loss: 1.9550, Validation Accuracy: 0.5169
Epoch 32/85, Train Loss: 0.3881, Train Accuracy: 0.8765, Validation Loss: 2.3028, Validation Accuracy: 0.4730
Epoch 33/85, Train Loss: 0.3249, Train Accuracy: 0.8860, Validation Loss: 2.5496, Validation Accuracy: 0.4797
Epoch 34/85, Train Loss: 0.3318, Train Accuracy: 0.8890, Validation Loss: 2.0584, Validation Accuracy: 0.5270
Epoch 35/85, Train Loss: 0.2850, Train Accuracy: 0.9114, Validation Loss: 1.9367, Validation Accuracy: 0.5777
Epoch 36/85, Train Loss: 0.3081, Train Accuracy: 0.8947, Validation Loss: 2.5274, Validation Accuracy: 0.4932
Epoch 37/85, Train Loss: 0.2974, Train Accuracy: 0.9049, Validation Loss: 2.2583, Validation Accuracy: 0.5405
Epoch 38/85, Train Loss: 0.2812, Train Accuracy: 0.9080, Validation Loss: 2.6984, Validation Accuracy: 0.4628
Epoch 39/85, Train Loss: 0.2906, Train Accuracy: 0.9068, Validation Loss: 2.0563, Validation Accuracy: 0.5980
Epoch 40/85, Train Loss: 0.3062, Train Accuracy: 0.9008, Validation Loss: 2.8062, Validation Accuracy: 0.5000
Epoch 41/85, Train Loss: 0.2655, Train Accuracy: 0.9155, Validation Loss: 2.2006, Validation Accuracy: 0.5304
Epoch 42/85, Train Loss: 0.2125, Train Accuracy: 0.9356, Validation Loss: 2.1376, Validation Accuracy: 0.5608
Epoch 43/85, Train Loss: 0.2090, Train Accuracy: 0.9341, Validation Loss: 2.5085, Validation Accuracy: 0.5405
```

```
Epoch 44/85, Train Loss: 0.2156, Train Accuracy: 0.9356, Validation Loss: 2.8480, Validation Accuracy: 0.5068
Epoch 45/85, Train Loss: 0.1844, Train Accuracy: 0.9417, Validation Loss: 2.3168, Validation Accuracy: 0.5676
Epoch 46/85, Train Loss: 0.1821, Train Accuracy: 0.9477, Validation Loss: 2.1646, Validation Accuracy: 0.6149
Epoch 47/85, Train Loss: 0.1952, Train Accuracy: 0.9379, Validation Loss: 2.6782, Validation Accuracy: 0.5236
Epoch 48/85, Train Loss: 0.1878, Train Accuracy: 0.9398, Validation Loss: 2.2451, Validation Accuracy: 0.5236
Epoch 49/85, Train Loss: 0.1783, Train Accuracy: 0.9489, Validation Loss: 2.1962, Validation Accuracy: 0.5439
Epoch 50/85, Train Loss: 0.1793, Train Accuracy: 0.9489, Validation Loss: 2.1887, Validation Accuracy: 0.5743
Epoch 51/85, Train Loss: 0.1742, Train Accuracy: 0.9504, Validation Loss: 2.2488, Validation Accuracy: 0.5642
Epoch 52/85, Train Loss: 0.1553, Train Accuracy: 0.9519, Validation Loss: 2.5628, Validation Accuracy: 0.5507
Epoch 53/85, Train Loss: 0.1405, Train Accuracy: 0.9542, Validation Loss: 2.5345, Validation Accuracy: 0.4899
Epoch 54/85, Train Loss: 0.1270, Train Accuracy: 0.9652, Validation Loss: 2.6544, Validation Accuracy: 0.5676
Epoch 55/85, Train Loss: 0.1109, Train Accuracy: 0.9705, Validation Loss: 1.9579, Validation Accuracy: 0.6318
Epoch 56/85, Train Loss: 0.1499, Train Accuracy: 0.9523, Validation Loss: 2.6258, Validation Accuracy: 0.5405
Epoch 57/85, Train Loss: 0.1181, Train Accuracy: 0.9644, Validation Loss: 2.3262, Validation Accuracy: 0.5845
Epoch 58/85, Train Loss: 0.1250, Train Accuracy: 0.9595, Validation Loss: 2.3479, Validation Accuracy: 0.5473
Epoch 59/85, Train Loss: 0.1327, Train Accuracy: 0.9576, Validation Loss: 2.7741, Validation Accuracy: 0.5372
Epoch 60/85, Train Loss: 0.1334, Train Accuracy: 0.9553, Validation Loss: 2.7163, Validation Accuracy: 0.5270
Epoch 61/85, Train Loss: 0.1284, Train Accuracy: 0.9595, Validation Loss: 2.4567, Validation Accuracy: 0.5642
Epoch 62/85, Train Loss: 0.1382, Train Accuracy: 0.9636, Validation Loss: 2.5498, Validation Accuracy: 0.5541
Epoch 63/85, Train Loss: 0.1250, Train Accuracy: 0.9598, Validation Loss: 2.1452, Validation Accuracy: 0.5845
Epoch 64/85, Train Loss: 0.1046, Train Accuracy: 0.9682, Validation Loss: 2.7010, Validation Accuracy: 0.5845
Epoch 65/85, Train Loss: 0.1154, Train Accuracy: 0.9621, Validation Loss: 2.0828, Validation Accuracy: 0.6385
Epoch 66/85, Train Loss: 0.0987, Train Accuracy: 0.9670, Validation Loss: 2.5095, Validation Accuracy: 0.5236
Epoch 67/85, Train Loss: 0.0782, Train Accuracy: 0.9773, Validation Loss: 2.1174, Validation Accuracy: 0.5912
Epoch 68/85, Train Loss: 0.0796, Train Accuracy: 0.9784, Validation Loss: 2.3121, Validation Accuracy: 0.5574
Epoch 69/85, Train Loss: 0.0677, Train Accuracy: 0.9814, Validation Loss: 2.6729, Validation Accuracy: 0.5034
Epoch 70/85, Train Loss: 0.0785, Train Accuracy: 0.9754, Validation Loss: 2.1596, Validation Accuracy: 0.6284
Epoch 71/85, Train Loss: 0.0809, Train Accuracy: 0.9712, Validation Loss: 2.1906, Validation Accuracy: 0.5777
Epoch 72/85, Train Loss: 0.0866, Train Accuracy: 0.9765, Validation Loss: 2.6846, Validation Accuracy: 0.5541
Epoch 73/85, Train Loss: 0.0792, Train Accuracy: 0.9780, Validation Loss: 2.1291, Validation Accuracy: 0.6216
Epoch 74/85, Train Loss: 0.0801, Train Accuracy: 0.9742, Validation Loss: 2.7425, Validation Accuracy: 0.5203
Epoch 75/85, Train Loss: 0.0940, Train Accuracy: 0.9742, Validation Loss: 3.0019, Validation Accuracy: 0.5034
Epoch 76/85, Train Loss: 0.0672, Train Accuracy: 0.9807, Validation Loss: 2.7815, Validation Accuracy: 0.5405
Epoch 77/85, Train Loss: 0.0672, Train Accuracy: 0.9826, Validation Loss: 2.6363, Validation Accuracy: 0.6081
Epoch 78/85, Train Loss: 0.0795, Train Accuracy: 0.9739, Validation Loss: 2.8557, Validation Accuracy: 0.5676
Epoch 79/85, Train Loss: 0.0796, Train Accuracy: 0.9750, Validation Loss: 3.1485, Validation Accuracy: 0.4932
Epoch 80/85, Train Loss: 0.0736, Train Accuracy: 0.9792, Validation Loss: 2.5076, Validation Accuracy: 0.5608
Epoch 81/85, Train Loss: 0.0587, Train Accuracy: 0.9830, Validation Loss: 2.7168, Validation Accuracy: 0.5507
Epoch 82/85, Train Loss: 0.0455, Train Accuracy: 0.9848, Validation Loss: 2.6451, Validation Accuracy: 0.5777
Epoch 83/85, Train Loss: 0.0735, Train Accuracy: 0.9803, Validation Loss: 2.4416, Validation Accuracy: 0.6014
Epoch 84/85, Train Loss: 0.0733, Train Accuracy: 0.9777, Validation Loss: 2.8646, Validation Accuracy: 0.5405
Epoch 85/85, Train Loss: 0.0613, Train Accuracy: 0.9852, Validation Loss: 2.2578, Validation Accuracy: 0.6047
```





After trialing different architectures and epoch durations, we settled on a ResNet-like model structure. The provided plots indicate a high learning rate, as seen by the significant initial drop in training loss that levels out, while validation loss shows considerable fluctuations. Such erratic validation accuracy suggests the learning rate may be too high, causing the model to overshoot optimal minima. A lower learning rate might smooth out validation loss and improve generalization, as evidenced by the discrepancy between training and validation accuracy.

In [104…

```
# Test phase
model.eval()
test_loss = 0
correct_test = 0
total_test = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        correct_test += (predicted == labels).sum().item()
```

```
        total_test += labels.size(0)

avg_test_loss = test_loss / total_test
test_accuracy = correct_test / total_test

print(f'Test Loss: {avg_test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')
```

```
Test Loss: 1.8870, Test Accuracy: 0.6588
```

In [105…

```
torch.save(model.state_dict(), 'CNN_model_state_dict.pth')

# when you want to load the model for inference or further training, you'll need to create an instance of the model class and the
# model = GroceryModel(num_classes=num_classes).to(device)

# # Load the state dictionary
# model.load_state_dict(torch.load(model_save_path))

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

II. Analyzing a Pre-trained CNN (Filters) (10pt)

In this part, you are going to analyze a (large) pre-trained model. Pre-trained models are quite popular these days, as big companies can train really large models on large datasets (something that personal users can't do as they lack the sufficient hardware). These pre-trained models can be used to fine-tune on other/small datasets or used as components in other tasks (like using a pre-trained classifier for object detection).

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

You can use the following transform to normalize:

`normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])` Read more here

1. Load a pre-trained VGG16 with PyTorch using torchvision.models.vgg16(pretrained=True, progress=True, **kwargs) (read more here). Don't forget to use the model in evaluation mode (`model.eval()`).

2. Load the images in the 'birds' folder and display them.

3. Pre-process the images to fit VGG16's architecture. What steps did you take?

4. Feed the images (forward pass) to the model. What are the outputs?

5. Choose an image of a dog in the 'dogs' folder, display it and feed it to network. What are the outputs?

6. For the first 3 filters in the first layer of VGG16, plot the filters, and then plot their response (their output) for the image from question 5. Explain your observations.

1

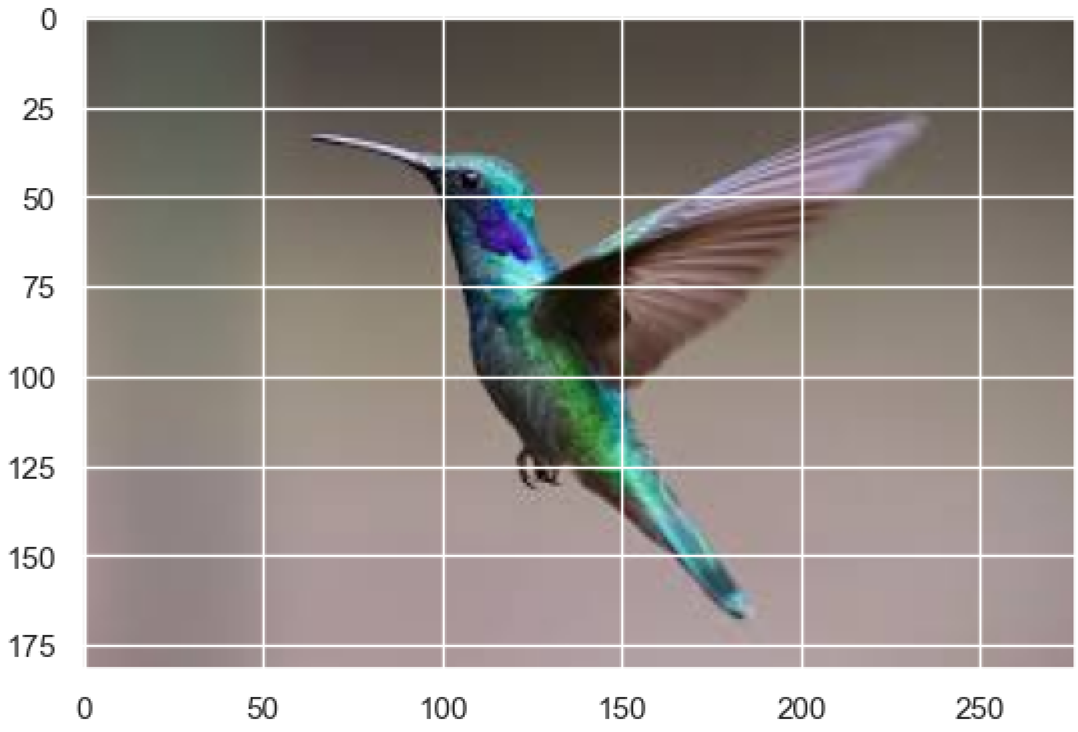
In [62]:
```
import torchvision.models as models

# Load the pre-trained VGG16 model
vgg16 = models.vgg16(pretrained=True, progress=True)

# Use the model in evaluation mode
vgg16.eval()
```

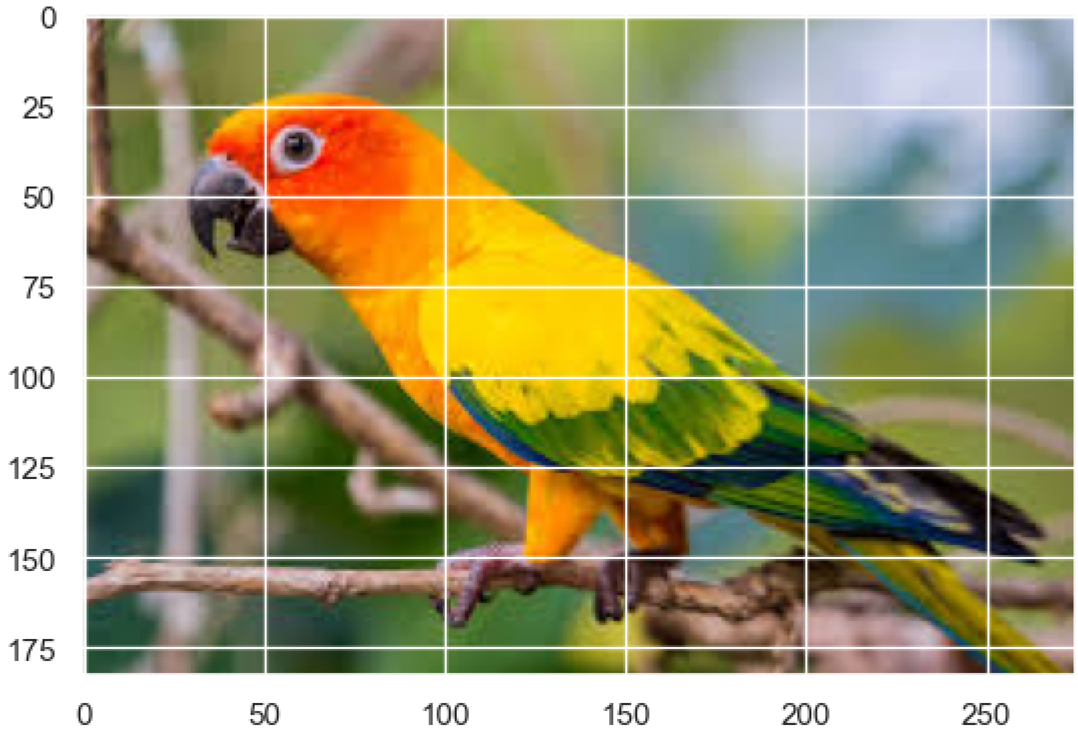
Out[62]:
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
```

```
    )
)
```

2

In [63]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Load the images
bird_images = [Image.open(f'birds/bird_{i}.jpg') for i in range(2)]

# Display the images
for img in bird_images:
    plt.imshow(img)
    plt.show()
```





3

In [64]:
```
from torchvision import transforms
import torch

# Define the transformation
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Apply the transformation to each bird image
bird_tensors = [preprocess(img) for img in bird_images]
```

steps taken:

1. resizing to 256 pixels
2. cropping to 224 pixels
3. converting to tensor
4. normalizing - By subtracting the mean and dividing by the standard deviation for each channel, the pixel values of an input image are standardized to have a mean of 0 and a standard deviation of 1. This process is known as feature scaling, and it makes the optimization landscape smoother, which is beneficial for training the model.

4

In [65]:
```
# Add an extra batch dimension and pass the image through the model
with torch.no_grad():  # We don't need gradients for evaluation
    outputs = [vgg16(img.unsqueeze(0)) for img in bird_tensors]

# The outputs are the class probabilities
for out in outputs:
    print(out)
```

```
tensor([[ 2.7060e+00,  5.0277e+00,  2.9450e+00, -2.4856e-02,  2.4988e+00,
          5.6518e+00, -3.4072e-01,  3.1121e+00,  1.6135e+00,  2.4929e+00,
          9.1234e+00,  6.1760e+00,  7.7521e+00,  6.6037e+00,  1.1655e+01,
          7.8751e+00,  1.3027e+01,  1.2279e+01,  1.1689e+01,  1.0140e+01,
          1.5392e+01,  1.3191e+01,  1.8790e+00,  5.6937e+00,  4.0444e+00,
          6.2261e-01,  4.5118e+00, -2.2118e-01, -1.2594e+00,  3.5354e+00,
          3.3018e+00,  2.1493e+00,  3.5794e+00,  2.7929e+00,  4.5813e+00,
          1.6225e+00,  5.0983e+00, -2.6581e+00,  5.2292e+00,  1.2032e+00,
          1.0537e+01,  6.8741e+00,  1.1330e+01,  8.1567e+00,  6.2877e+00,
         -1.1993e+00,  9.1111e+00,  1.1892e+01, -2.2449e-01, -8.4637e-01,
         -6.9902e-01,  4.0637e+00,  3.5153e+00,  5.4052e+00,  7.5434e-01,
          4.2801e+00, -5.1978e-01, -1.6011e+00,  3.9177e+00,  1.2785e+01,
          3.0190e+00, -1.9027e+00,  8.6624e-01,  6.0651e+00,  8.1334e+00,
          5.0426e+00,  5.2342e+00,  2.5183e+00,  3.1506e+00, -3.4604e+00,
          6.4542e-01, -1.3845e+00, -1.9206e+00, -6.3782e-02, -3.4248e+00,
         -1.4253e+00, -3.7242e+00, -1.5409e-01,  5.9037e-01, -3.5663e+00,
          1.0672e+01,  6.5711e+00,  9.7876e+00,  8.2318e+00,  1.5200e+01,
          7.9761e+00,  9.2827e+00,  2.9789e+00,  4.7227e+00,  2.8613e+00,
          3.9560e+00,  1.1881e+01,  2.1098e+01,  6.2811e+00,  2.9623e+01,
          2.7770e+01,  8.0685e+00,  9.0163e+00,  1.3781e+01,  6.9197e+00,
          1.1396e+00, -5.8499e+00, -2.4459e+00,  8.8775e+00,  9.1594e-01,
         -1.5346e+00, -5.8784e+00, -5.5321e+00, -3.6661e+00, -3.3234e+00,
          2.6554e+00,  6.7360e+00, -8.5510e-01,  1.0354e+01,  8.8651e+00,
          2.1743e+00,  2.0631e+00, -1.4991e-01, -4.2656e+00, -2.9318e+00,
          1.1919e+00, -5.8677e+00, -4.3332e-02, -1.7743e+00,  5.1624e+00,
         -2.9594e+00,  6.4178e+00,  4.7576e+00,  1.6148e+01,  1.1344e+01,
          6.0155e-01,  1.0997e+01,  4.5650e+00,  1.2298e+01,  9.4438e+00,
          9.2400e+00,  1.3863e+01,  3.5566e+00,  1.0764e+01,  7.5806e+00,
          7.2992e+00,  1.1174e+01,  1.2170e+01,  3.6213e+00,  5.3114e+00,
          6.8824e+00,  6.9955e+00,  6.4797e+00,  2.0152e+00,  3.7859e+00,
          2.9912e+00,  2.4800e+00, -3.0904e-02, -2.7580e+00, -3.4882e+00,
         -6.8325e-01, -7.1703e-01,  3.0805e+00,  5.4799e+00, -2.3527e+00,
          2.5357e+00,  1.7128e+00,  1.7733e+00,  4.4407e-01,  3.3733e+00,
         -8.5707e-02,  2.3438e+00,  3.8474e+00, -2.9558e-01,  2.7777e+00,
         -2.9028e+00,  4.6150e+00,  2.2460e+00,  1.9338e+00, -2.0236e+00,
         -1.5661e+00,  2.9898e+00,  3.1759e+00, -2.2275e+00, -3.2299e+00,
         -2.8479e+00,  7.0369e-01, -1.1546e+00,  1.7556e+00, -1.6052e+00,
         -1.8627e+00, -7.4594e-01,  3.0249e-01,  3.6990e+00,  2.8451e-01,
         -1.0840e+00,  3.3068e-01, -3.3297e+00,  8.9125e-01, -5.5091e-01,
          1.2055e+00,  4.8939e-01, -2.9445e+00,  1.6258e+00, -1.5734e-01,
         -4.1942e+00, -3.7919e+00, -4.6405e+00, -3.8639e+00, -5.0068e+00,
         -1.5891e+00, -5.0374e-01, -2.0661e+00, -2.1292e+00, -3.1712e+00,
          2.7195e+00, -2.3481e+00,  1.6751e+00,  6.8168e-01, -2.1145e-01,
          2.9977e-01, -2.8791e+00, -6.3913e-01,  1.2337e+00,  6.3574e-01,
```

```
 1.1729e-01,  2.9039e-01, -5.8024e+00, -6.4086e+00, -2.5322e+00,
 2.3667e-01, -4.6522e+00, -1.1769e+00, -6.1053e+00, -4.5517e+00,
-1.5441e+00, -2.8691e+00, -1.1825e+00, -4.3714e+00, -3.8803e+00,
 1.3965e-02, -1.9199e+00,  2.6432e+00, -2.9778e+00, -4.3019e+00,
-1.8068e+00, -3.1208e+00, -1.4559e+00, -3.3291e+00, -6.7298e+00,
-2.4960e+00, -1.8628e+00, -3.1815e+00, -3.6626e+00, -5.8714e+00,
-3.8691e+00, -6.2652e+00, -2.3043e+00,  1.0468e+00, -3.3836e+00,
-5.2593e+00, -3.2304e+00, -6.5891e+00, -6.1669e+00, -4.9897e+00,
-6.0530e+00, -5.7399e+00, -1.4829e+00, -1.2746e+00,  6.2705e-01,
 1.8608e+00, -7.3378e-01, -6.2339e-01,  2.7387e+00, -1.3526e+00,
-2.8017e+00, -1.9498e-01,  1.6610e+00,  7.4697e-01,  2.5777e-01,
-2.0066e+00, -2.3775e+00,  1.0271e+00,  1.0090e+00, -1.8340e+00,
 3.3349e+00, -1.6612e+00, -3.5347e+00, -6.5963e+00,  2.5074e+00,
 2.0892e+00, -2.7493e+00, -3.0763e+00, -2.9281e+00, -6.0168e+00,
-4.9307e+00, -6.0862e+00, -8.6254e+00, -5.8691e-01, -4.9358e+00,
-4.6437e+00, -3.4382e+00, -2.8297e+00,  1.8650e+00,  6.3187e+00,
 4.6927e+00,  1.0181e+00,  4.3559e+00,  6.6263e+00,  4.6301e+00,
-3.3388e-01,  1.3901e+00,  1.0435e+01,  6.3393e+00,  4.8505e+00,
 6.9751e+00,  4.7766e+00,  4.3545e+00,  5.2037e+00,  3.1178e+00,
 4.2596e+00,  2.5857e+00,  4.7872e+00,  8.4496e+00,  1.3240e+01,
 1.0565e+01,  1.9886e+00,  5.6778e+00,  4.4898e-01,  5.6983e+00,
 2.1667e+00,  6.9261e+00,  1.4755e+00, -1.3939e+00, -9.8346e-01,
 5.0722e+00,  9.1062e+00, -1.1072e+00, -3.3759e+00, -2.7463e-01,
 6.8155e+00, -8.8778e-01,  3.2884e+00, -3.1824e+00, -4.0212e+00,
-5.3924e+00, -5.3294e+00, -6.8722e+00, -5.9482e+00, -3.3948e+00,
-3.0205e+00, -4.3782e+00, -4.0509e+00, -6.0936e+00, -2.5837e+00,
 3.5910e-01,  1.2620e+00,  2.2922e+00,  6.2004e+00, -1.1890e+00,
 2.8418e+00,  6.9659e+00,  4.2673e+00,  1.8180e+00,  3.2161e+00,
 3.2283e+00, -1.4386e+00, -3.1202e-01,  2.5477e+00,  3.2657e+00,
 1.1684e+00, -4.9194e+00, -2.0117e+00,  8.1639e-01,  2.5428e+00,
 2.7426e+00,  2.4744e+00, -3.5479e+00, -3.3533e+00, -1.5694e+00,
 1.9105e+00,  2.0242e+00,  1.2423e+00,  1.3302e+00,  1.6454e+00,
 3.8076e+00,  3.6607e+00,  3.0422e+00,  4.3440e+00,  2.2322e+00,
-4.9890e+00, -4.7660e+00, -2.6760e+00, -5.8475e+00,  9.2803e+00,
 5.4348e+00,  3.4739e+00,  2.9258e+00, -3.0242e+00,  1.1503e+00,
 9.3860e+00, -5.7069e-01, -1.7635e-01, -6.2971e-01, -4.7393e+00,
-3.2394e-02, -7.0335e+00, -4.3045e+00, -5.6776e+00,  7.9890e+00,
-8.2250e-01, -3.1200e+00, -5.4054e+00, -5.6540e+00, -1.3460e+00,
-7.0461e+00,  2.4464e-01, -2.6471e+00, -1.6373e+00, -4.2359e+00,
-7.6694e+00,  5.4641e+00,  1.0675e+00,  4.7506e+00, -2.9691e+00,
-1.6504e+00, -8.2220e-01,  3.9693e+00, -1.8402e+00, -5.6179e+00,
-3.1778e+00,  8.5463e-01, -2.9478e+00,  2.1229e+00, -3.8117e+00,
-4.2532e+00, -1.6253e+00,  8.9575e-01, -2.6675e+00, -6.3579e-01,
-1.6157e+00, -2.1701e+00, -2.9201e+00,  6.0324e+00, -1.5434e+00,
```

```
 7.9161e-01, -2.4352e+00, -3.1334e+00, -2.1630e+00, -2.0006e+00,
 2.7481e-01, -1.6922e+00, -3.8103e+00,  1.8172e+00, -2.6482e+00,
-4.0580e+00,  2.4996e+00,  1.1639e+00, -1.9511e+00, -4.8785e+00,
-2.1391e+00,  5.5871e+00,  2.3669e+00, -4.3257e+00,  3.7471e+00,
-1.8734e+00,  2.6611e-01,  6.1741e+00,  8.5821e-01,  1.7298e-01,
-4.0857e+00, -2.2727e-01, -3.9649e+00, -6.3192e+00,  4.4074e-01,
-9.7369e-01, -4.4781e+00, -3.6623e+00,  4.3331e+00, -1.7531e+00,
 6.1025e-01, -5.9822e+00, -1.4020e+00, -4.4624e+00, -5.1599e+00,
-3.0618e+00, -2.9019e+00, -5.4981e+00, -5.5661e+00, -1.0377e+00,
-4.6095e+00, -3.5806e+00,  2.9628e-01,  4.2218e+00,  7.9149e-01,
-2.1685e-01, -5.2945e+00, -5.4298e+00, -5.5211e+00, -7.3448e-01,
-5.9911e+00, -4.7379e-01, -3.4278e+00, -8.3569e+00,  4.8405e+00,
-6.2277e+00,  1.3034e+00, -1.2998e-01,  2.4524e+00, -2.2676e-01,
 2.6935e+00,  2.0448e+00, -8.9261e-01, -8.5463e-01, -5.4697e+00,
-3.9540e+00, -4.0163e+00,  4.1226e+00, -3.6677e+00, -1.8604e+00,
-6.8692e-01, -2.7579e+00, -1.6838e+00,  1.1199e-01, -6.6841e+00,
-3.5358e+00, -3.2014e-01,  8.6686e-01,  3.5644e-01, -2.5851e+00,
-2.9370e+00,  4.6493e-02,  1.7972e+00, -4.2588e+00, -4.6804e+00,
-7.0558e-01, -8.7438e-01, -4.1794e+00,  6.7118e-01, -1.8670e+00,
-7.3924e+00, -5.7325e+00, -7.4301e+00, -1.0540e+00, -2.5684e+00,
-6.2637e+00, -2.5895e+00, -5.5976e-01,  3.0321e+00, -2.9217e+00,
 6.9808e-01, -1.6002e+00, -6.1544e+00, -3.7065e+00, -5.7382e-01,
-3.1960e+00, -2.1951e+00, -2.0840e-01, -3.1233e+00, -1.6530e+00,
-6.3659e+00, -1.1235e+00,  2.1197e+00, -1.2828e+00, -1.0387e-01,
-6.8263e+00, -5.5269e+00,  2.4010e+00,  3.5640e+00, -8.3654e-01,
-7.2922e+00, -2.5684e+00,  3.5802e+00, -3.9419e+00, -5.1608e+00,
-1.0334e+00, -5.4974e+00,  6.4311e+00, -6.7464e+00,  1.8789e+00,
-2.8786e+00, -5.1332e+00,  5.1477e-01,  1.5758e+00, -3.8244e+00,
 9.0056e-01, -5.3516e+00, -3.3967e+00, -1.1253e+00,  4.9292e+00,
-1.8054e+00, -7.5697e+00,  9.2783e+00, -2.3701e+00,  1.8560e+00,
 2.3857e+00, -4.2739e-02, -2.1616e+00, -2.4080e+00, -4.3147e+00,
-3.0811e+00,  4.4959e+00, -4.2010e+00, -3.9773e+00, -3.4150e+00,
 5.2563e+00, -6.7226e-02,  5.2734e+00, -5.7671e+00,  5.5190e+00,
-1.1654e-01,  8.5118e-01, -8.3946e+00, -3.9211e+00, -6.4477e+00,
-4.7262e+00, -7.1105e-01, -2.3222e+00,  1.2111e+00, -3.3430e+00,
-4.3087e+00,  5.3075e+00, -4.0048e-02,  4.8872e+00,  1.8737e+00,
-3.1854e-01, -4.9664e-01, -3.7796e+00,  8.8471e+00, -4.9820e+00,
-9.0817e+00, -1.8992e+00, -3.2293e+00, -8.1668e+00,  1.1453e+00,
-5.9699e-01,  1.7613e+00, -2.1807e+00,  4.8501e+00, -8.3303e+00,
-5.8746e-01, -1.4710e-01, -3.4600e+00,  1.6430e+00, -7.8296e-01,
-8.8754e+00,  4.1672e+00, -3.0753e+00, -2.8739e+00,  6.5312e+00,
 2.7030e-02, -3.9581e+00,  2.7502e+00, -2.4979e+00, -2.6989e+00,
 2.4865e+00, -1.5150e+00, -6.8781e+00, -1.0277e+00, -3.1742e+00,
-2.0325e+00, -4.7932e-01,  6.1845e-01,  8.7497e-01, -7.8974e-01,
```

```
-6.3340e-01, -8.9038e+00, -6.3630e-01, -4.2588e+00,  5.5331e+00,
-2.9313e+00,  4.0590e+00,  6.1527e+00,  1.1534e-01,  1.3335e+00,
 1.7136e+00, -4.2579e+00,  1.3799e-01,  2.4382e+00,  2.0764e-01,
-6.1960e+00,  1.7464e+00,  8.9250e+00, -2.5827e+00,  9.4377e-01,
 3.3679e+00, -3.3110e-01, -2.2322e+00, -7.5738e-01,  5.7074e-01,
-2.1493e+00, -3.4186e+00, -9.0095e+00, -1.8842e+00,  2.0152e+00,
-6.7907e+00,  1.0929e+00,  1.1742e+00,  3.5022e+00, -5.5299e+00,
-3.3999e+00,  6.1190e+00, -6.0705e+00, -5.8875e+00, -1.3884e+00,
 9.6066e-01,  2.0949e+00,  3.7641e+00, -3.6568e+00, -4.6264e+00,
-5.0100e+00, -5.3408e+00, -5.8550e+00,  2.0133e+00, -3.3132e+00,
-1.9911e-01,  1.4049e+00,  1.7435e+00,  1.6542e-01, -4.0276e-01,
-9.7542e-02, -3.3293e+00, -5.6496e+00, -2.5401e+00, -1.7660e+00,
-2.9966e+00, -2.8364e+00, -5.2894e-03,  7.7264e+00, -3.1402e+00,
 5.2657e+00, -8.4615e-01, -1.7389e+00,  1.6350e+00, -2.7008e+00,
-5.3626e-02,  3.1845e+00, -2.3966e+00,  2.9526e+00, -3.0104e+00,
-1.2806e+00, -3.1466e+00,  1.4316e+00,  1.0348e+00, -3.8082e+00,
 2.7603e+00,  3.5240e-01, -1.6157e+00, -3.8259e+00,  2.4082e+00,
-9.0315e-01, -4.4165e+00,  9.0759e-02, -1.4939e+00,  9.7819e+00,
-1.0995e+00, -6.4099e+00,  1.9400e+00, -3.1930e+00, -1.3145e-01,
-5.4178e-02, -2.3966e+00, -5.8452e+00,  4.1551e+00, -3.4640e+00,
-4.2600e+00, -2.0438e+00, -4.3000e+00, -4.1370e-01, -1.2067e+00,
-1.7653e+00, -1.9465e+00, -7.3044e-01, -3.2896e+00,  2.1125e+00,
-6.1944e+00, -1.7036e+00,  2.9732e+00, -1.5058e+00,  1.1819e-01,
-3.3261e-01, -3.7198e+00,  5.4369e+00, -5.8028e-01, -7.9914e+00,
-2.1486e+00, -2.2711e+00,  3.7136e+00,  3.4423e+00,  7.8457e+00,
-5.9366e-01, -4.5086e+00, -1.7720e+00, -6.6247e+00, -9.3438e-01,
 1.6281e+00, -8.4010e-02,  1.7060e+00, -5.6525e-01, -1.8305e-01,
 3.4350e-01, -2.5410e+00, -3.1032e+00,  4.2321e-01, -3.8093e+00,
-7.1229e+00,  2.7031e+00, -3.6285e+00, -8.1149e+00,  5.9588e+00,
-2.3342e+00,  3.4046e+00,  3.4387e+00,  1.7967e+00,  1.1832e+00,
-3.3101e+00, -3.8783e+00, -7.9458e-02,  4.4920e+00, -2.2161e+00,
-3.9812e+00,  2.8974e+00, -1.6230e+00,  4.1658e+00, -4.4391e+00,
-9.2565e+00, -1.1079e+00, -1.3185e+00,  2.7372e+00,  1.7595e+00,
-1.4654e+00, -9.7629e-01, -1.0485e+00,  2.8960e+00, -4.9112e+00,
-3.1447e+00, -2.1054e+00, -6.1244e+00, -4.3843e+00, -2.9195e+00,
 2.6360e+00,  2.2963e+00, -1.8801e-01, -2.6546e-01, -5.7274e+00,
 1.5339e+00, -3.1938e+00, -1.7766e+00, -1.4660e+00, -2.4245e+00,
 5.5873e+00,  6.3844e+00, -3.4368e+00, -4.9190e+00,  2.1576e+00,
-1.8848e+00,  3.4263e+00,  1.6714e+00, -3.1562e+00, -2.1043e+00,
 4.0085e+00, -3.1002e+00, -4.8446e+00, -3.2549e+00, -2.8487e+00,
-1.7113e+00,  3.2997e-01,  1.4841e+00, -3.5405e+00, -3.5909e+00,
-5.7853e+00, -8.9463e+00, -4.2932e+00,  4.4566e-01, -5.4219e+00,
-1.3123e+00, -8.3963e-01, -4.0968e-01, -4.1906e+00, -6.3931e+00,
-6.7352e-01, -1.7806e+00,  9.4180e-02, -7.7762e+00,  2.7370e+00,
```

```
        -3.8803e-02, -5.3958e+00, -1.6436e+00,  4.2620e+00, -4.9474e+00,
         1.6072e+00, -6.0258e+00, -2.7488e+00, -2.7232e+00, -4.5802e+00,
        -5.0621e+00, -4.0235e-01,  2.8424e+00, -3.0720e+00, -6.6609e+00,
         2.5986e+00, -9.7790e-02, -3.7758e+00,  3.0182e+00,  3.9492e+00,
        -4.0192e+00,  1.7247e+00,  4.0866e+00, -7.2800e-01,  6.4024e+00,
         1.5870e+00,  6.7544e+00,  3.0021e+00,  1.4409e+00,  2.2844e-01,
         1.6287e+00,  1.8094e+00,  8.9256e+00, -6.7623e+00, -2.9283e+00,
        -7.3177e+00, -5.1157e-01, -4.9094e+00,  1.1855e+00,  1.4836e-01,
        -3.3793e-01,  1.7244e+00, -2.1433e+00, -1.7236e+00, -7.9742e-01,
        -8.2485e-02, -2.5501e+00,  3.1594e-02, -1.4392e+00, -9.2487e-02,
        -2.9827e+00, -6.3532e+00, -6.0192e+00, -6.6066e+00, -5.6659e+00,
        -3.0570e+00, -2.1255e-01,  2.9599e+00, -1.2270e+00,  1.6740e+00,
        -2.2931e+00, -2.5518e+00, -3.2893e+00,  2.1843e+00,  4.1640e+00,
         1.1242e-01,  6.2913e+00, -5.8271e-01,  2.1894e+00,  5.4071e+00,
         6.2925e-01, -1.6989e+00,  6.4062e+00,  1.2832e+00, -5.3780e-01,
        -3.2120e+00,  1.6992e+00, -3.2394e+00, -2.8306e+00, -5.6959e+00,
        -2.8220e+00, -7.5976e+00, -3.1104e-01, -4.6680e+00, -3.9490e+00,
        -6.7029e+00,  2.7625e+00, -2.3692e+00,  1.0222e+00,  2.0405e+00,
        -1.9321e+00,  3.4153e+00, -1.9862e+00,  1.9487e-01, -1.1662e+00,
         3.5665e+00,  6.4443e-02,  2.8970e-02,  7.4477e-02, -1.7551e+00,
        -1.9575e+00,  1.7908e-01, -4.6116e+00, -2.4902e+00,  1.7637e+00,
         4.8953e+00,  3.0306e+00,  2.6262e+00,  7.4406e+00,  4.8430e+00,
        -1.8529e+00, -5.3694e+00, -3.9986e+00, -2.6678e+00,  2.4994e+00,
         3.0964e+00, -1.3039e+00,  6.9722e-01,  5.7531e+00, -1.4468e+00]])
tensor([[ 1.4935e+00,  8.1809e+00, -1.5602e+00, -3.8882e+00, -4.1595e+00,
        -1.2790e+00, -2.9763e+00,  7.4214e+00,  6.3584e+00, -4.3471e-02,
         9.5735e+00,  1.3498e+01,  9.7240e+00,  5.7353e+00,  1.0312e+01,
         8.6859e+00,  1.2129e+01,  7.4136e+00,  7.8780e+00,  5.0433e+00,
         6.7286e+00,  8.6189e+00,  5.1014e+00,  6.9557e+00,  2.1671e+00,
         3.2122e+00,  2.7483e+00,  4.0267e+00, -1.8364e-01,  2.1162e+00,
        -4.7759e-01,  6.0199e+00,  3.4347e+00, -2.1924e+00, -2.1848e+00,
         9.3482e-01,  2.7815e+00,  2.2177e+00,  2.5251e+00,  3.2634e+00,
         7.4696e+00,  2.9861e+00,  8.1169e+00,  3.6356e+00,  1.7902e+00,
         3.2610e-01,  7.4471e+00,  5.6811e+00,  3.9690e-01, -7.6875e-01,
        -1.3764e+00, -3.8673e-01,  4.2682e-01,  2.2279e+00, -2.1433e+00,
         3.0128e+00, -1.5366e-01, -1.0404e+00, -3.3835e-01,  5.4065e+00,
         6.6944e-01, -2.3057e+00, -7.1167e-01,  7.8788e-01,  4.7581e+00,
        -8.2578e-01, -5.3748e-01, -5.0771e+00, -1.4935e+00, -3.5708e+00,
        -1.2710e+00, -3.8756e-01,  6.2969e-01,  2.9161e-01, -1.9110e-01,
        -6.0237e-01, -2.6295e+00, -3.0077e+00,  1.5743e+00, -1.3132e+00,
         6.4652e+00,  3.6550e+00,  8.8073e+00,  5.5286e+00,  4.1730e+00,
         7.3926e+00,  1.0134e+01,  9.6164e+00,  1.6668e+01,  1.0589e+01,
         1.9824e+01,  1.1220e+01,  1.4033e+01,  1.0239e+01,  9.5150e+00,
         1.3107e+01,  1.5355e+01,  6.9479e+00,  3.7320e+00,  6.6572e+00,
```

```
 5.4334e+00, -1.5014e+00, -2.5244e+00,  1.0402e+00, -2.2464e+00,
 1.2953e-02, -1.0981e+00, -1.7572e+00,  1.4494e+00,  8.7363e-01,
-1.6361e+00, -7.7425e-01, -3.8261e-01,  1.1613e+00,  2.1507e+00,
 1.3578e+00, -9.0631e-01, -2.0937e+00,  1.2201e+00,  5.4790e-01,
 1.8564e+00,  1.3894e+00,  3.2685e-03, -1.3319e+00,  2.5510e+00,
-7.5087e-01, -8.7923e-01,  4.5969e+00,  7.1356e+00,  7.7944e+00,
 6.2249e+00,  5.7873e+00,  3.7561e+00,  5.4321e+00,  6.4275e+00,
 3.1212e+00,  1.2714e+01,  5.2624e+00,  7.5844e-01, -9.3431e-01,
-5.6251e-01,  1.5387e+00, -1.9819e+00,  5.1571e+00,  4.0266e+00,
 5.3241e+00,  1.0818e+00, -3.0993e+00, -3.7289e+00, -1.0060e+00,
-6.6014e-01, -9.3593e-01, -2.0761e+00, -5.7727e-01, -1.8415e+00,
-2.7793e+00, -1.8889e+00, -6.0436e-01, -8.2253e-01, -2.0373e+00,
-3.1634e+00, -1.9444e+00, -2.1446e+00, -7.8851e-01, -1.1576e+00,
 5.6524e-01, -6.5726e-01, -4.3779e-01, -2.0737e+00, -1.2615e+00,
-1.3791e+00, -1.3666e+00, -2.5850e+00, -5.9287e-01, -3.1874e+00,
-1.9779e+00, -1.7577e+00, -3.0222e+00, -1.8802e+00, -2.2290e+00,
-2.6624e-01, -1.6130e+00, -1.3775e+00, -1.9339e+00, -2.9508e+00,
-2.1685e+00,  9.9075e-01, -6.2522e-01, -1.2255e+00, -8.3613e-01,
-3.0360e+00, -4.3269e-01, -3.9051e+00, -1.4023e+00, -2.8739e+00,
-1.3888e-01, -2.2970e+00, -2.1980e+00, -1.4307e+00, -1.7730e+00,
-2.6668e+00, -2.2752e+00, -9.8888e-01, -1.8689e+00, -2.0723e+00,
 3.8864e-01, -1.2318e-01, -1.0931e-01, -1.4841e+00, -2.9640e-01,
-3.4568e+00, -1.7313e+00, -1.6692e+00, -1.3398e+00, -1.7724e-01,
 4.3367e-01, -1.3099e+00, -3.0617e+00, -6.4538e-01, -2.8278e+00,
-2.0191e+00, -2.0462e+00, -1.7072e+00,  4.6207e-01, -1.7752e+00,
-6.9506e-01, -2.6578e+00, -1.2570e+00, -1.8277e+00, -1.6801e+00,
-2.4935e+00, -1.5998e+00, -7.9478e-01, -2.4432e+00,  3.8041e-01,
-8.4071e-01, -2.4086e-01,  6.2441e-01, -3.1343e-01, -7.6865e-01,
-9.3231e-01, -8.5038e-01, -1.4181e+00, -1.9794e+00, -1.3400e+00,
-2.8369e+00, -2.3270e+00, -1.8811e+00, -3.8869e+00, -3.0877e+00,
-3.3115e+00, -2.6400e+00, -3.4742e+00, -9.8155e-01, -5.0846e-01,
-1.4715e+00, -1.4956e+00, -2.7946e+00, -3.2681e+00, -1.2690e+00,
-1.2718e+00, -2.3066e+00, -2.0228e+00, -1.2598e+00, -2.4288e+00,
-6.4971e-01, -1.1447e-02,  1.4223e-01, -3.9320e-01, -2.3149e+00,
-1.7299e+00, -6.5198e-01, -2.0442e+00, -9.2234e-01, -1.0945e+00,
-1.3692e+00, -1.5361e+00, -1.3667e-01, -1.8628e+00, -3.1465e+00,
-1.3801e+00, -1.8083e+00, -1.9276e+00, -1.2251e+00, -2.1103e+00,
-2.2405e+00, -1.6984e+00, -3.6305e+00, -1.0804e+00, -3.1727e+00,
-2.5430e+00, -1.6633e-01, -8.5100e-01, -2.5578e+00, -1.9662e+00,
-3.9294e-01,  5.6331e-01,  6.2192e-01,  2.2739e+00, -3.5322e-01,
 1.7755e+00,  5.4950e+00,  6.5106e-01,  1.5763e+00,  2.6356e+00,
-2.1446e+00,  2.3476e+00,  3.1579e+00,  8.0133e-01,  2.5046e+00,
 6.0732e+00,  3.7835e+00,  1.6328e+00,  1.9363e+00,  2.9225e-01,
 2.8643e+00,  3.5401e+00,  3.7888e+00,  1.5394e+00,  1.5120e+00,
```

```
 1.7052e+00,  2.9496e+00,  4.6578e-01,  4.2430e+00,  2.0861e-01,
 6.5218e+00,  2.8824e+00, -1.4790e+00, -1.3294e+00, -3.0792e+00,
-1.5580e+00, -1.5497e+00, -1.2619e+00, -2.5334e-01, -4.8358e-01,
 2.1066e+00, -1.2903e+00,  1.1255e+00,  3.0243e+00, -2.0388e+00,
-2.8196e+00, -5.9221e-01, -2.7530e+00, -4.7755e+00, -2.1332e+00,
-1.2795e+00, -1.8673e+00, -3.9047e+00, -2.8051e+00, -2.7228e+00,
-2.7237e+00, -3.1043e+00, -1.9358e+00, -3.3918e+00, -1.1197e+00,
-2.6282e+00,  8.6785e-01,  4.1908e-01,  6.8161e-01,  1.3691e+00,
-1.0809e+00, -1.4658e+00, -7.4220e-01, -1.7558e+00,  2.4291e+00,
 3.9458e-01, -4.7950e-01,  1.6035e+00,  9.4909e-01,  2.3398e+00,
 4.6702e+00,  4.1128e+00,  2.6551e+00,  2.5853e+00,  4.8550e+00,
 1.8447e+00,  3.1057e+00,  7.6327e+00,  4.0913e+00,  3.1149e+00,
 6.5808e+00,  3.7364e+00,  6.7353e+00,  1.0866e-01,  2.9274e+00,
-1.3657e+00, -8.9743e-01,  3.2632e+00,  1.8101e-02,  1.0164e-01,
 3.9605e+00,  2.1286e-01,  6.7662e+00,  2.8955e+00, -1.6357e+00,
 2.1868e-01, -3.3013e-01, -1.4705e+00, -5.2100e-01, -8.1258e-01,
-1.0468e+00, -2.5826e+00, -1.7332e+00, -3.0158e+00,  6.7171e-01,
 2.5240e+00, -1.7349e+00,  1.6017e+00, -1.2942e+00, -1.8032e+00,
-2.6144e+00, -1.7770e-01,  1.4100e+00, -5.5354e-01,  7.7586e-01,
-2.4235e+00,  2.7248e-01,  2.8610e+00, -1.3667e+00, -1.2865e+00,
-1.5759e+00,  1.0271e+00, -1.9385e+00, -1.5962e+00, -1.1735e+00,
-8.3842e-01, -2.5825e+00,  1.3279e+00,  3.3990e+00,  2.3667e+00,
 6.2230e-01, -1.3983e+00, -1.7049e+00, -1.7596e-01, -1.8049e-01,
-1.1243e+00, -3.5969e+00, -9.9849e-01, -7.5706e-01,  3.6686e+00,
-9.0158e-01, -5.9344e-01, -1.1867e+00, -6.6462e-01,  1.0931e+00,
-1.1140e+00, -1.9879e-01, -1.0069e+00,  3.8345e+00,  9.8219e-02,
-2.4197e-02, -1.3395e+00,  2.3268e+00, -2.6919e-02, -2.1976e+00,
-2.0686e-01, -7.3126e-01,  6.4142e-01, -1.7473e+00, -1.0063e+00,
-9.0524e-01, -1.6558e+00,  4.0580e-02,  2.5720e+00, -4.8877e-01,
-1.8051e+00, -2.2610e+00,  1.4211e+00,  4.3794e-01,  1.2025e+00,
-5.1929e-01, -2.7909e+00,  8.0065e-01, -7.4826e-01, -1.6144e+00,
-2.4477e-01, -6.1855e-01, -1.4642e+00, -1.8991e+00, -2.6557e+00,
-2.8521e+00, -7.1642e-01, -2.5627e+00, -2.5221e+00, -1.4771e+00,
-2.0941e+00, -1.1282e+00, -1.3846e+00,  2.1080e+00,  5.2607e+00,
-1.3546e+00,  2.1998e+00, -2.0105e+00, -1.6343e+00,  2.4631e-01,
-4.8034e+00,  7.4671e-01, -1.7883e+00, -3.1721e+00,  9.8224e-01,
-1.2387e+00, -3.2538e-01,  7.6947e-02, -1.5282e+00,  9.0152e-01,
-2.0266e-01,  6.1459e-01,  2.9368e-01, -1.3672e+00, -1.3752e+00,
-2.5460e+00, -1.4319e+00, -1.7641e+00, -3.4702e+00, -3.7581e-02,
-2.4008e-01, -2.2296e+00, -1.1873e+00,  3.0096e+00, -3.5554e-01,
 5.0791e-02,  5.8464e-01,  2.8766e+00, -1.7235e+00, -9.7377e-01,
-3.4368e+00, -1.3669e+00, -8.8717e-01, -1.9273e+00, -9.3264e-01,
 9.4811e-01, -1.5760e+00, -2.4945e+00, -6.6972e-01, -4.9757e-01,
-2.4769e+00, -6.0475e-01, -3.7701e+00, -1.2126e+00, -1.2802e+00,
```

```
-1.5741e+00,  6.8555e-01, -2.4579e+00, -2.2097e+00,  2.7475e-01,
-1.9666e+00, -1.0587e+00, -4.0065e+00, -2.7995e+00, -1.5348e+00,
-2.8480e+00, -4.0610e+00,  3.3957e+00, -6.2793e-01, -2.2728e+00,
-3.0490e+00,  2.5210e+00,  1.6651e+00,  9.5500e-01,  1.0272e+00,
 3.6345e-01, -1.2150e+00, -1.2564e+00,  3.7007e-01, -3.3619e+00,
-3.4813e+00, -2.4482e+00, -1.3687e+00, -6.3254e-01, -4.2495e-02,
-8.5371e-01, -1.2256e+00,  9.9990e-01, -1.2741e+00, -4.2722e-01,
-1.5715e+00, -3.0328e+00, -1.4645e+00, -1.3970e+00, -3.6226e+00,
 7.3619e-01, -1.3860e+00,  8.6706e-01,  2.9682e-01,  2.1522e-01,
-1.7887e+00, -4.4002e+00, -8.2185e-01,  3.6258e-01, -7.7260e-01,
-5.6645e-01,  2.4373e+00, -3.5422e+00, -9.2151e-01, -2.1946e+00,
-6.2224e-01,  8.9912e-01, -4.1671e+00, -2.3104e+00, -2.6086e+00,
 3.1225e+00, -4.7173e-01,  2.1958e+00, -3.4823e+00, -2.0454e+00,
-3.1181e+00,  4.4056e-01,  1.5794e+00, -1.6353e+00, -2.3115e+00,
-2.5726e-01,  2.1038e-01,  6.0893e-01,  5.5107e-01,  8.2572e-02,
-4.1070e-01,  4.8720e+00, -2.5115e+00, -1.5816e+00,  1.9511e+00,
-2.6225e-01,  5.2841e-01, -1.9020e+00,  2.2644e-01, -9.4624e-02,
 1.3160e+00, -1.7110e+00, -3.9466e+00, -2.7706e+00, -2.7793e+00,
-3.5344e+00, -2.5854e+00,  1.4754e+00,  7.8299e-02, -3.0024e+00,
-2.1665e+00,  7.9302e-03, -1.3091e-01,  1.2277e-01,  4.0973e-01,
-4.5398e+00,  1.6144e+00, -1.4243e+00,  1.6038e+00,  1.2201e+00,
 3.1762e-01, -2.8233e+00, -2.0056e+00, -3.4258e+00, -1.9563e+00,
-2.7111e+00, -1.8606e-01, -9.9659e-01, -6.1717e-01, -4.5982e-01,
-1.0084e+00, -2.2260e+00,  1.0921e-01, -7.5562e-01, -6.8808e-01,
-2.7162e-01, -4.2880e+00, -2.2314e+00, -2.9227e+00,  2.6978e+00,
 1.1059e+00, -2.5466e+00,  1.0216e+00, -2.9935e+00, -2.2293e+00,
 1.8371e+00,  1.4931e+00,  3.6358e-01, -9.0079e-01,  2.1796e-01,
-2.2632e+00,  2.3644e-01,  2.6572e+00,  1.0441e+00, -1.9459e+00,
-3.7981e+00, -4.5487e-02, -2.4361e+00, -9.6606e-01, -2.3334e-01,
-2.4672e+00, -3.2816e+00, -4.7812e+00, -2.3043e-02, -1.3119e+00,
-2.7509e+00, -3.9161e+00,  1.7876e+00,  1.5168e+00, -2.8592e+00,
 1.5273e+00,  2.2103e+00, -4.1505e+00, -3.0377e+00, -4.3755e-01,
-2.6997e-01,  2.7998e+00,  1.6347e+00, -1.1863e-01,  6.1292e-01,
-2.8808e+00, -2.6515e+00, -1.3616e+00, -1.1357e-02, -1.3816e-01,
 1.1957e+00, -3.8863e+00, -3.5024e+00, -1.0013e+00, -8.5312e-01,
-1.9346e+00,  1.3771e+00, -3.2869e+00, -3.0334e+00,  7.3280e-01,
-6.3733e-01,  2.2663e+00,  3.6300e+00,  8.4915e-01, -1.0105e+00,
 8.1648e-01, -1.2761e+00, -5.2220e-01,  1.5934e+00, -1.2207e+00,
-2.9212e+00,  1.3603e-01, -1.7387e+00,  7.9513e-01, -1.5800e+00,
 1.2751e+00, -2.9865e+00, -5.1845e-01,  2.6573e+00,  2.2930e-02,
 2.1600e+00, -1.0597e+00, -3.3113e-01,  1.1988e+00, -1.0722e+00,
-5.4212e+00, -3.4259e+00,  2.1843e+00, -1.7403e+00,  3.8942e+00,
 6.6150e-02, -1.9056e+00,  1.6188e+00, -1.7234e+00, -4.0127e+00,
-5.8606e+00,  6.8814e-01, -3.9318e+00,  1.1765e-01, -3.5673e+00,
```

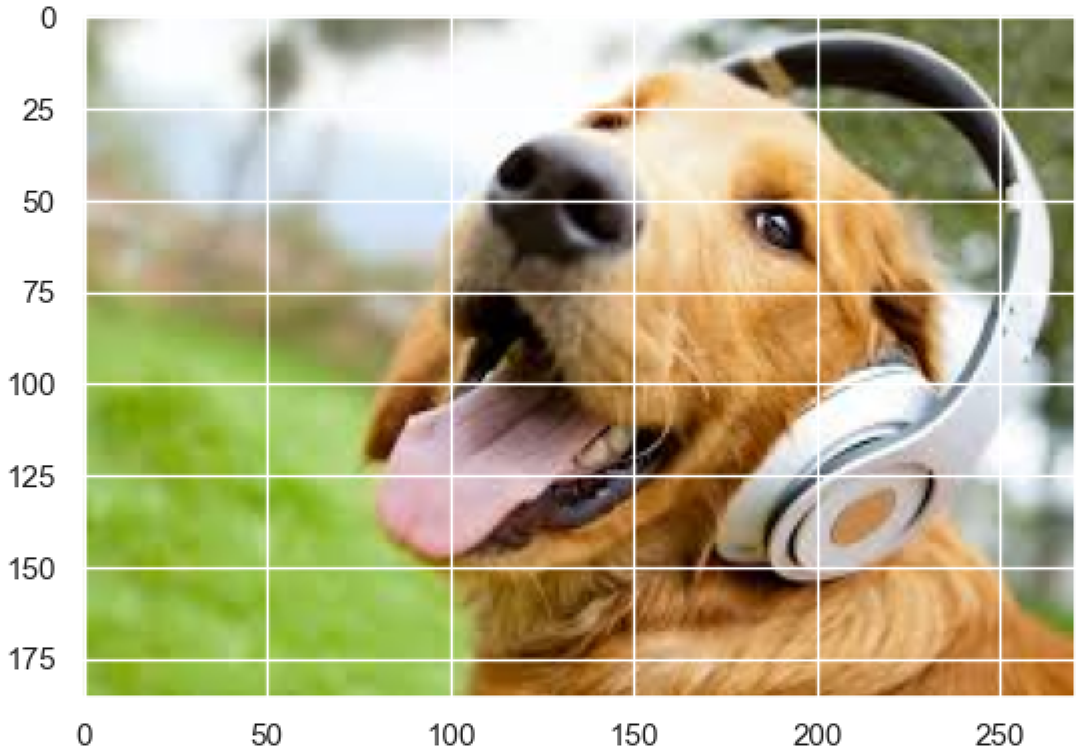
```
-2.1515e+00, -2.0618e+00, -2.6137e-01, -3.1794e+00, -1.3183e+00,
-8.3774e-01, -7.0508e-01,  1.9509e+00,  6.3378e-01,  1.8336e-01,
-4.6366e-01, -1.2181e+00, -2.9198e+00, -1.1868e+00, -4.0959e-01,
 3.8115e+00, -8.5092e-01, -1.2868e+00,  2.2569e-01,  1.7699e+00,
-6.0012e-01, -1.0585e+00,  2.7388e+00, -3.2970e-01,  1.2169e+00,
 1.3782e+00, -2.5838e+00, -1.4025e+00, -2.3370e+00, -1.2745e+00,
 1.0830e+00,  1.4448e+00, -2.6632e-01, -2.0703e+00,  9.3587e-02,
 7.7772e-01,  9.8808e-01,  1.4788e+00, -2.7539e+00, -1.4412e+00,
-5.6260e-01, -9.6553e-01, -4.3710e-01, -3.5004e+00,  5.5743e-01,
-7.4059e-01, -1.2094e+00, -3.0028e+00,  9.4261e-01, -1.1729e+00,
-4.7353e+00,  1.0392e+00, -5.1703e+00, -1.3800e+00, -4.0147e-02,
-1.1675e+00, -1.1023e+00, -1.1832e-01, -2.5542e-01, -1.6346e+00,
-5.7594e+00, -1.4082e+00, -6.3861e-01, -1.7865e+00,  5.7856e-01,
-8.8256e-01, -2.8043e-01, -7.9357e-01, -1.3129e+00, -2.4882e+00,
 3.0910e+00, -3.1280e-01, -5.7118e-01, -1.6655e+00, -1.2669e+00,
-2.0043e+00, -5.9727e-01, -6.3515e-01, -1.1479e+00, -4.5339e+00,
 1.1440e+00, -2.3087e+00, -2.8908e-01,  1.4590e+00,  3.3617e-01,
-1.3681e+00,  2.6715e+00, -3.8751e+00, -1.4517e+00,  4.9721e-01,
 2.0513e+00,  1.7686e+00,  2.3912e+00, -1.9266e+00, -1.3591e+00,
 1.4240e+00, -2.7218e+00, -2.2811e+00, -9.3026e-01, -2.8662e-01,
-4.9209e-01, -1.9487e-01,  2.1303e+00, -1.2860e+00, -2.3913e+00,
 2.1402e+00, -2.1261e+00, -2.6445e+00,  1.7726e+00, -1.7527e+00,
 1.9267e+00, -2.0134e+00, -2.6860e+00, -3.8404e+00, -2.8953e+00,
-2.2678e+00,  7.0329e-02, -1.0293e+00, -4.0639e+00,  1.3006e+00,
 2.4886e+00, -3.4082e+00,  4.3507e-01,  2.6623e+00, -3.4242e+00,
-1.0486e+00, -2.0402e+00, -3.1327e+00, -3.3686e+00, -2.5524e+00,
 1.8025e+00, -2.0149e+00, -9.3069e-01, -7.3255e-01, -9.3118e-01,
 9.8787e-01, -2.4690e+00, -9.4693e-01, -3.0550e+00,  1.2972e+00,
-2.9941e+00, -3.8010e+00,  1.3686e+00,  2.2472e+00,  3.1317e+00,
-4.8561e-01, -1.8199e-01, -8.2410e-01,  1.6897e+00, -4.0999e-01,
-2.4912e+00,  1.3842e+00,  7.9023e+00, -4.3016e+00, -2.3807e+00,
-3.7470e+00, -1.1270e+00,  2.6090e+00, -1.5285e+00,  1.2513e+00,
 4.6194e-01,  3.2569e+00, -1.4813e+00,  7.6614e-01, -1.3731e+00,
-2.0203e+00, -9.7766e-01, -2.0557e+00, -2.0352e+00,  8.1207e-01,
-6.3608e-01, -2.8280e+00, -1.3693e+00, -5.6554e-01,  4.6147e-01,
-1.0107e+00, -5.9683e-01,  8.0285e-02,  1.5136e+00,  2.8846e+00,
 3.2665e+00,  3.3104e+00,  2.5094e+00,  1.5274e+00,  1.7245e+00,
 2.7046e+00,  2.2777e+00,  2.3907e+00,  2.7192e+00,  3.5979e+00,
 6.1094e+00,  5.7764e+00,  7.3034e+00,  4.1701e+00,  3.3397e+00,
 9.5592e-01,  3.2127e+00,  5.3803e+00, -1.8774e+00, -3.2971e+00,
-4.4625e+00, -2.3031e+00, -3.4028e-01,  3.9984e-01,  9.0869e-02,
-1.4340e+00, -2.2233e-01, -3.4883e+00,  8.2020e-01, -1.7982e+00,
-1.0883e-02, -9.2006e-01,  2.0598e-01,  3.7426e+00, -2.5995e+00,
 2.8495e+00, -2.6278e+00, -1.0103e+00,  7.1214e-01, -2.1434e+00,
```

```
8.0429e-01,  1.8406e+00, -2.9729e+00, -9.3180e-01,  9.2549e-01,
2.2760e+00,  3.0720e+00,  2.7754e+00,  3.7987e+00,  6.6557e+00,
8.5728e-01,  2.8689e+00,  1.6775e+00, -2.8590e-01,  4.7430e+00,
1.0511e+00,  1.8625e+00,  2.5782e+00,  3.8515e+00, -2.1162e+00]])
```

Answer: The outputs are the class probabilities

5

In [74]:
```python
# Assume the dog image is loaded similarly to the bird images
dog_image = Image.open('dogs/dog_1.jpg')

# Display the dog image
plt.imshow(dog_image)
plt.show()

# Preprocess the image
dog_tensor = preprocess(dog_image)

# Add an extra batch dimension and pass the image through the model
with torch.no_grad():
    dog_output = vgg16(dog_tensor.unsqueeze(0))

# Output from the model
print(dog_output)
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250

```
tensor([[-9.3060e-01, -1.5955e+00, -2.9181e+00, -4.1349e+00, -2.8313e+00,
         -2.4859e+00, -3.1912e+00, -2.7256e+00, -1.1700e+00, -2.7315e+00,
         -2.8770e+00, -1.5801e+00, -4.0232e+00, -4.2405e+00, -2.6859e+00,
         -2.8089e+00, -3.9026e+00, -2.3800e+00, -2.5218e+00, -3.2893e+00,
         -2.9577e+00, -1.5593e+00, -2.3144e+00, -2.4575e+00, -3.1204e+00,
         -1.3885e+00, -1.6896e+00, -6.0798e-01, -1.2195e+00, -1.8533e+00,
         -2.4155e+00, -7.8195e-01, -5.7745e-01, -2.2821e+00, -2.4605e+00,
         -3.5580e+00, -1.9797e+00, -2.1328e+00, -3.2781e+00, -4.5888e+00,
         -3.6562e+00, -4.3453e+00, -2.5557e+00, -2.4689e+00, -4.7558e+00,
         -3.4266e+00, -4.4510e+00, -2.2288e+00, -3.7720e+00, -5.1290e+00,
         -4.6302e+00, -8.4347e-01, -7.2791e-01, -1.0448e+00, -3.1612e+00,
         -2.6060e+00, -3.5917e+00, -3.3642e+00, -3.0165e+00, -1.0052e+00,
         -1.7103e+00, -1.3729e+00, -8.7586e-01,  4.4460e-01, -8.2738e-01,
          3.1337e-02, -1.8670e+00, -3.5151e+00, -2.0490e+00, -2.9219e+00,
         -3.2173e+00, -1.9258e+00, -2.7817e+00, -1.0925e+00, -2.7280e+00,
         -2.9774e+00, -3.1106e+00, -3.3193e+00,  4.5527e-01, -2.6626e+00,
         -2.4115e+00, -4.2777e+00, -2.1560e+00, -1.7952e+00, -3.0947e+00,
         -3.5731e+00, -1.8481e+00,  3.7205e-02, -1.4454e+00, -2.6534e+00,
         -2.2799e+00, -4.3833e+00, -2.4769e+00, -3.2584e+00, -3.1906e+00,
         -2.2515e+00, -2.1849e+00,  1.9516e+00,  3.1432e-01,  1.0880e+00,
         -1.3574e+00, -9.8150e-01, -8.2496e-01, -3.1887e-01, -7.6571e-01,
         -1.4704e+00, -4.7907e-01, -1.7897e+00, -3.1342e-01,  5.3437e-01,
         -2.9351e-01,  5.7276e-01, -1.0535e+00,  1.8138e-02, -2.3122e-01,
         -2.3583e-01, -7.5521e-02, -2.3299e+00, -5.0080e-01, -2.3621e+00,
         -1.7998e+00, -8.9790e-01, -1.7945e+00, -2.0338e+00, -7.0620e-01,
          6.5285e-01, -1.4624e+00, -3.0378e-01, -8.4593e-01, -3.1592e+00,
         -8.2936e-01, -3.6102e+00, -3.0903e+00, -3.1451e+00, -1.3620e+00,
         -1.1475e+00, -2.2329e+00, -3.6127e+00, -5.6224e+00, -2.4204e+00,
         -2.5409e+00, -2.9809e+00, -2.0930e+00, -8.5930e-01, -2.6059e-01,
         -5.3039e-01, -4.5042e-01, -3.2710e+00, -2.3977e+00, -1.3893e-01,
         -7.7873e-01,  3.0370e+00,  1.2672e+00, -4.6721e-01,  4.1618e-01,
         -1.1733e+00,  2.0519e+00,  8.1341e-01,  3.0440e+00,  8.7988e+00,
          4.2781e+00,  5.6136e+00,  5.9133e+00,  7.3604e+00,  3.5719e+00,
          4.9242e+00,  7.0642e+00,  7.2929e+00,  9.8741e+00,  4.7561e+00,
          3.6507e+00,  3.4588e+00,  2.7083e+00,  6.2490e+00,  2.6695e+00,
          7.2712e+00,  5.8248e+00,  3.3746e+00,  2.9405e+00,  4.1923e+00,
          5.6326e+00,  3.0720e+00,  4.2938e+00,  2.0578e+00,  7.7446e+00,
          5.1070e+00,  4.7692e+00,  8.8516e-01,  5.2854e+00,  6.4672e+00,
          2.7412e+00,  7.4043e+00,  1.5728e+00,  5.2809e+00,  2.0127e+00,
         -1.5119e+00,  1.0866e+00,  6.9420e-01,  3.1351e-01,  1.9723e+00,
          3.5933e+00,  6.5199e-01,  2.8197e+00,  8.4160e-01, -2.3948e-01,
          5.3183e+00,  7.2238e+00,  1.2840e+01,  9.8444e+00,  8.1881e+00,
          1.5965e+00,  8.2568e+00,  4.1963e+00,  7.8992e+00,  4.4920e+00,
          7.0028e+00,  5.5798e+00,  7.1779e-01,  6.3259e+00,  6.7902e+00,
```

```
 7.4639e+00,  3.8861e+00,  6.9491e+00,  2.8928e+00,  2.7621e+00,
 4.4117e+00,  3.2255e+00,  8.5554e+00,  1.7755e+00,  3.2546e+00,
 5.3447e+00,  6.6945e+00,  4.9965e+00,  1.0188e+00,  5.1962e+00,
 5.5674e+00,  4.6137e+00,  3.9339e+00,  5.6339e+00,  3.9197e+00,
 6.9619e+00,  4.4623e+00,  1.1593e+00,  3.0996e+00,  5.2035e+00,
-1.3696e+00,  2.2082e+00,  3.9303e+00,  4.0820e+00,  2.7064e+00,
 2.6285e+00,  1.5542e+00,  2.2109e-01,  6.9662e+00, -2.1651e+00,
 3.4390e+00,  3.6005e+00,  4.1271e+00,  2.6598e+00,  1.6496e+00,
 5.5124e+00, -5.5393e-02,  2.0657e+00,  6.2261e+00,  6.7473e+00,
 3.0138e+00,  4.3878e+00,  6.5241e+00,  3.5115e-01,  1.2784e+00,
 1.4475e+00,  3.4722e+00,  1.1573e+00,  7.4964e+00,  3.5552e+00,
-7.7606e-01,  1.0034e+00,  1.6317e+00,  1.1119e-01, -4.7544e-01,
-1.4139e+00, -1.1501e+00, -3.2818e-01, -2.6088e+00, -1.7869e+00,
-2.0956e+00, -2.3662e+00, -3.5960e+00, -2.1551e+00, -5.2976e+00,
-1.4505e+00,  1.3691e-01, -1.4895e+00,  8.7173e-03,  5.8100e+00,
 1.2491e+00,  1.7415e+00,  1.9360e+00,  3.8263e-01,  1.1923e+00,
-3.1307e+00, -1.2151e+00, -2.5836e+00, -2.1586e+00, -8.1157e-01,
-3.0531e+00, -2.3877e+00, -2.3394e+00, -5.0466e+00, -2.4951e+00,
-1.0549e+00, -3.4951e+00, -3.7332e+00, -3.3120e+00, -1.5219e+00,
-2.8254e+00, -2.8234e+00, -4.8520e-01, -3.7514e+00, -3.7396e+00,
-4.2496e+00, -3.3071e+00, -3.1587e+00, -2.6859e+00, -4.4335e+00,
-2.5306e+00, -4.1923e+00, -8.9043e-01, -6.8424e-02, -1.1263e+00,
-2.3618e+00, -1.2012e-01, -1.1837e+00, -1.3155e+00, -1.8819e+00,
 5.1026e-02, -4.2363e-01,  1.3278e+00,  1.2882e+00,  3.6749e+00,
-2.8724e+00,  2.3104e+00,  1.6034e-01, -2.8725e+00, -2.4746e+00,
 1.8839e+00, -8.9901e-01, -8.4088e-01, -7.3987e-01, -1.4383e+00,
-3.1684e-01, -1.8681e+00, -1.5344e+00, -5.0096e-02, -2.2007e-01,
-3.8547e-02,  1.6397e-01, -1.3991e+00, -2.4537e-01,  1.9315e-01,
-1.0545e+00, -1.2367e+00,  3.1842e-01, -1.5420e+00,  1.2599e+00,
 6.7404e-01, -2.1672e+00, -1.2205e+00,  4.4810e-01, -1.0648e+00,
-1.9080e+00,  1.6625e-01, -4.3917e-01, -2.1533e-01, -1.8344e+00,
-3.3768e+00,  7.1183e-01, -6.1144e-01,  6.3324e-01,  7.3495e-01,
 3.5015e-01, -1.5343e+00, -1.7597e-01, -2.8379e+00, -1.5569e-01,
-2.1074e+00, -4.0176e-01,  1.2679e+00,  6.6183e-01, -2.0477e+00,
-7.2965e-01, -3.5072e+00,  6.9933e-01,  4.8573e-01, -1.8032e+00,
-2.1135e+00, -2.7777e+00, -2.0807e+00, -7.7747e-01, -1.0204e+00,
 1.9432e+00, -4.2632e-01, -2.0027e+00, -5.0677e+00, -3.7548e-01,
-1.2351e+00, -3.6770e+00,  5.3702e-01, -1.6790e+00,  2.1049e+00,
-1.4789e+00,  2.6984e-01,  4.2694e-01, -1.0628e+00,  5.7449e-01,
 2.1007e+00,  5.0149e-01,  1.1006e+00,  1.1824e+00,  1.3130e+00,
-1.6069e+00, -3.9009e-01,  5.6902e-01, -1.2953e+00, -1.5162e-01,
-2.1928e+00,  1.5395e+00,  1.4923e+00,  2.6177e+00,  1.2772e+00,
 3.8252e+00,  1.0493e-01, -1.3059e-01,  2.2733e+00,  5.3891e+00,
 6.9356e-01,  8.0560e-01, -2.7474e+00, -1.2768e+00,  9.9206e-02,
```

```
 1.0559e+00, -1.2341e+00, -2.9593e+00,  6.7124e-01,  7.8818e-01,
 1.5556e+00,  6.2402e-01,  9.6553e-01, -1.3688e+00, -2.0405e+00,
-3.5830e-01,  3.2466e-01,  1.6890e+00, -1.5276e+00, -1.2595e+00,
 1.5113e+00,  7.1576e-01, -6.6378e-02, -2.1045e+00, -2.2267e-01,
-1.9416e+00, -1.4808e-01,  1.1438e-01,  1.6410e+00, -3.0782e-01,
 1.7181e+00, -5.4052e-01, -7.8598e-01,  9.8451e-01, -1.5640e+00,
-1.5003e+00, -1.4649e+00,  3.2553e+00,  1.8074e-01, -7.9909e-01,
 1.8557e+00, -5.0550e-02, -1.6366e-01,  7.4177e-01,  1.4321e+00,
 1.4343e+00, -5.7247e-01,  1.0310e+00, -2.4788e+00, -1.8923e+00,
 5.3668e-02, -4.1693e-01,  2.7319e+00, -2.3265e-01, -7.5840e-01,
-1.2493e+00,  6.0521e-01, -1.1807e+00, -1.4301e-01,  5.1344e-01,
-2.2260e+00, -4.8687e-01, -2.2339e+00, -1.1662e+00, -2.2613e-01,
-2.3805e+00, -4.5296e-01,  2.7169e+00, -1.6289e+00,  1.0390e+00,
-9.8520e-01, -5.9597e-01,  1.3761e+00, -2.1728e-01,  1.1144e-01,
-1.0976e+00, -4.1665e-01, -1.1764e+00, -5.5894e-01,  1.6367e+00,
 2.6832e+00, -2.5923e-01, -1.2449e+00,  1.6755e+00,  8.9728e-01,
-1.2874e-01, -2.8093e-01,  3.4548e+00,  1.9890e+00, -1.3686e+00,
-3.9710e+00, -1.4016e+00, -2.6346e-01, -6.9498e-01,  2.1679e+00,
 2.8677e+00,  3.5024e+00, -2.2675e+00, -2.1190e+00,  3.3148e-01,
-1.1025e+00, -2.1279e+00,  1.3178e+00, -2.8721e+00,  2.8591e+00,
-2.5362e+00,  2.0600e-01,  9.1052e-01,  2.9386e+00,  3.9416e-01,
-9.5405e-01, -1.2496e+00, -3.4697e+00,  2.6066e-01, -1.8326e+00,
-1.4228e+00, -3.5409e-01,  2.0434e+00,  9.1595e-01, -3.9075e+00,
-2.2262e+00, -2.1168e+00, -2.2183e-01,  9.6689e-01,  1.6637e+00,
 2.7830e+00, -1.1339e+00, -2.5665e+00, -2.0430e+00, -2.7972e+00,
-2.6235e+00, -8.8409e-01,  1.8184e+00, -1.0202e+00, -1.1935e+00,
-6.9665e-01, -2.3154e-01, -3.3940e+00,  7.5382e-01,  1.9786e+00,
 1.0405e+00, -2.5131e+00,  9.8075e-01, -1.4914e-01, -2.6107e+00,
-1.5155e+00, -2.6128e+00,  5.0115e-01, -6.6098e-01,  1.9138e-01,
 1.1351e+00, -3.0742e+00,  2.9325e-01,  1.7886e+00,  1.4612e+00,
 3.8452e+00,  2.0111e-01, -1.1219e+00,  9.4350e-01,  1.6334e-01,
-1.1098e+00,  1.1990e+00, -7.8501e-01,  4.1735e-01,  9.5464e-01,
 3.1521e-01, -1.7203e-01,  8.8622e-01, -1.8079e+00, -3.5575e-01,
 1.9062e+00,  2.3260e-02,  1.6780e-01, -1.4834e+00,  3.7201e-01,
 1.4008e+00, -1.2430e+00, -9.2832e-01,  3.7552e-02, -2.0786e+00,
 2.7727e+00, -9.8709e-01, -6.4539e-02, -1.1092e+00, -2.5446e+00,
 7.9583e-01,  1.4130e+00,  6.4121e-01, -5.0632e-01, -1.2589e+00,
-1.0587e+00,  8.2938e-01, -5.3945e-01, -3.0288e+00, -7.2153e-01,
-2.0327e-01,  1.2665e+00,  3.7496e-02,  9.9236e-01, -1.5970e+00,
 4.4034e-01,  1.9778e-01, -9.7453e-01,  3.0116e+00,  2.7800e+00,
-1.5032e+00,  2.0295e+00, -9.8320e-01,  3.6147e-01,  3.0573e-04,
 1.8753e-01,  2.8472e+00, -4.7097e-01, -1.2338e+00, -1.7241e+00,
 9.7306e-02,  5.5474e-01, -1.9512e+00, -6.4605e-01,  8.7404e-01,
 5.6680e-01,  1.7672e+00, -2.3354e+00, -1.3098e+00, -1.1024e+00,
```

```
         6.5794e-01, -1.2816e+00, -7.8722e-01, -2.5556e+00,  1.9478e-01,
         5.1512e-01, -5.2475e-02,  3.0419e+00, -2.1200e+00, -3.1643e+00,
         2.4566e-01,  9.9745e-02,  5.6238e-01,  1.1762e+00, -9.5257e-02,
         1.0167e+00,  6.1751e+00, -1.1840e+00,  6.0872e+00, -1.9132e+00,
         3.6890e+00,  1.9527e-01, -1.6800e+00, -9.1282e-01,  3.0282e+00,
        -7.5298e-01,  9.0833e-01, -2.8104e+00,  9.6817e-01, -1.9182e-01,
        -4.6733e-01,  2.2941e+00,  1.6383e+00,  4.2997e+00, -2.5242e+00,
         3.2819e-01,  3.8675e-01,  4.4509e-02, -2.7034e+00,  9.3212e-01,
         3.7038e+00, -5.8790e-01,  1.2440e+00, -2.6955e-01, -3.6463e-01,
         7.4902e-01, -2.7496e+00,  1.4654e+00, -1.8086e+00, -7.5032e-01,
         5.8283e-01, -2.2962e+00, -2.3547e-01, -1.4750e+00, -8.6365e-02,
        -2.1399e+00, -5.5195e-01,  1.3737e+00, -2.1136e+00,  1.0485e+00,
         8.5159e-01, -9.0081e-01,  1.6734e+00, -4.5805e-01, -2.7525e+00,
        -1.4383e+00, -5.5515e-01, -1.9979e+00,  1.2103e+00, -1.7702e+00,
        -2.5813e+00,  2.1871e+00,  4.4523e-01,  1.7199e+00,  5.9147e-01,
        -1.0721e-01,  3.7469e-01,  1.4639e-01, -1.9179e+00, -1.0644e-01,
         1.4873e+00, -1.6455e-01,  5.1824e-01, -1.4409e+00, -1.6860e+00,
        -1.5860e+00,  5.0964e-01,  2.4429e-01, -2.3331e-01, -1.5093e+00,
         1.5499e+00,  9.9722e-01,  2.3102e+00, -1.5366e+00,  1.2790e+00,
        -3.8730e+00,  5.1147e-01,  1.2737e+00,  1.5486e+00, -1.4126e+00,
        -1.1820e+00,  2.4626e+00, -8.5809e-01, -8.8023e-01, -8.2075e-01,
         9.6245e-01,  4.9607e-01, -8.4025e-02,  4.2926e+00, -1.4070e+00,
         7.6010e-01,  6.0129e-02,  6.9413e-01,  1.2564e-01, -7.6681e-01,
         1.4212e+00, -2.0861e+00, -2.6172e+00, -9.4888e-01,  2.0844e+00,
        -3.0990e+00,  1.3166e+00,  2.0444e+00, -1.4762e+00,  1.6570e+00,
         4.0738e+00, -1.5861e+00, -7.1457e-01, -1.2817e+00, -5.8509e-01,
         1.0704e+00,  3.6180e-01, -6.9588e-01,  2.6946e+00, -1.2152e+00,
        -3.6397e-01,  1.8762e-01,  2.8496e-01, -1.0331e+00,  1.0586e+00,
         1.8153e+00,  1.2504e+00, -2.3376e+00, -3.8005e+00,  1.8799e+00,
         8.2898e+00, -9.8920e-01, -1.0479e+00,  1.9927e+00, -1.3067e-01,
        -1.8872e+00,  2.9266e+00, -1.9394e-01,  2.4415e+00,  8.8183e-01,
        -2.8045e+00, -1.7206e+00,  5.2111e-01, -1.1253e+00, -2.7224e+00,
        -4.6725e+00, -1.1395e+00, -1.1031e+00,  1.7127e+00,  1.6351e+00,
        -9.9400e-01,  2.9178e+00, -2.2974e-01, -6.5722e-01, -1.3380e+00,
         8.7284e-01, -3.0997e-01, -3.5476e+00, -3.4021e+00, -2.7301e-01,
        -2.1080e+00,  1.3839e+00,  1.0532e+00,  3.1743e+00, -1.9079e+00,
        -2.8202e-01, -9.3314e-01,  2.1625e+00,  1.2279e+00,  1.3354e+00,
        -1.3560e+00, -2.1118e+00, -4.1347e+00,  6.8902e-02, -9.8258e-01,
         2.6753e+00,  1.7758e+00,  8.2814e+00, -2.6655e+00, -1.4390e+00,
        -2.5588e-01, -2.8999e-01, -2.2689e+00, -3.6966e+00,  1.8107e+00,
         3.6999e-01,  8.6500e-01, -6.2683e-01, -5.0496e-01,  4.6390e-02,
         2.0729e+00, -1.2197e-02, -7.9384e-01, -2.6987e-01, -1.2703e+00,
         2.0691e+00, -2.7818e+00,  3.6056e-01, -2.0576e+00, -1.0955e+00,
        -1.1565e+00,  2.4733e+00,  9.3500e-02, -1.7258e+00, -6.0776e-01,
```

```
 8.7012e-01, -1.0386e+00,  1.4883e+00, -3.1953e+00, -2.7487e+00,
-5.3782e-01, -3.2006e-01, -8.0039e-01, -2.2911e+00, -4.6722e-01,
 4.0515e+00,  1.0265e+00, -8.1557e-02, -2.7833e-01, -2.0201e+00,
-2.5723e+00, -1.8507e+00, -4.3498e-01,  3.2301e+00,  3.6744e-01,
-2.7440e+00, -8.9989e-01,  3.3462e+00,  8.8289e-01,  1.4621e+00,
-7.9103e-01,  9.2202e-01, -7.6250e-01, -1.0817e+00, -1.4745e+00,
 7.5835e-01,  9.1968e-01,  1.4505e-01, -2.6657e+00, -2.4837e+00,
-1.7599e+00,  1.6049e+00,  8.9829e-03,  6.0769e-04,  1.1978e+00,
-5.7336e-01,  8.7768e-01, -1.4038e+00,  2.4647e-01,  5.5597e-01,
-7.5475e-01,  8.9554e-02, -2.0328e+00, -1.1477e+00,  3.6972e+00,
 2.6525e+00,  2.8546e+00,  3.6697e+00, -2.3912e-01,  2.5678e+00,
-4.2833e-01, -2.9124e-01,  2.0680e+00, -3.2019e-01,  1.0417e+00,
-2.6525e-02, -1.2939e+00, -5.0509e-01, -2.9430e-01, -1.6124e+00,
-2.5069e+00, -2.1197e+00, -2.2198e-01,  1.2990e+00, -1.5277e+00,
 8.3328e-01,  1.4729e+00,  2.0794e-02, -1.5356e+00, -1.1993e+00,
 1.7192e+00,  1.4288e+00, -4.2325e+00,  1.2376e+00, -1.8170e-01,
 1.2617e+00,  2.7897e-01, -1.9093e-01,  3.8230e-02,  1.1775e+00,
-1.9136e-01, -1.4290e+00, -5.9584e-01, -1.4163e-01,  9.4019e-01,
 3.8893e-01,  1.3667e+00, -1.0236e+00,  1.5870e+00, -2.3074e+00,
 1.1125e+00, -1.6120e+00,  1.7624e+00,  5.7577e-01, -2.2896e+00,
-3.8100e+00,  4.7890e-01, -8.7057e-01, -1.3675e+00, -1.8656e-01,
-2.7479e+00, -9.3618e-01,  3.0738e+00, -5.8214e-01, -3.7268e+00,
-1.0619e-01, -8.6885e-01, -1.8823e+00, -4.6552e-01, -9.5514e-01,
 5.8733e-01,  2.8836e-01, -4.1823e-01,  2.6853e+00,  3.2806e+00]])
```

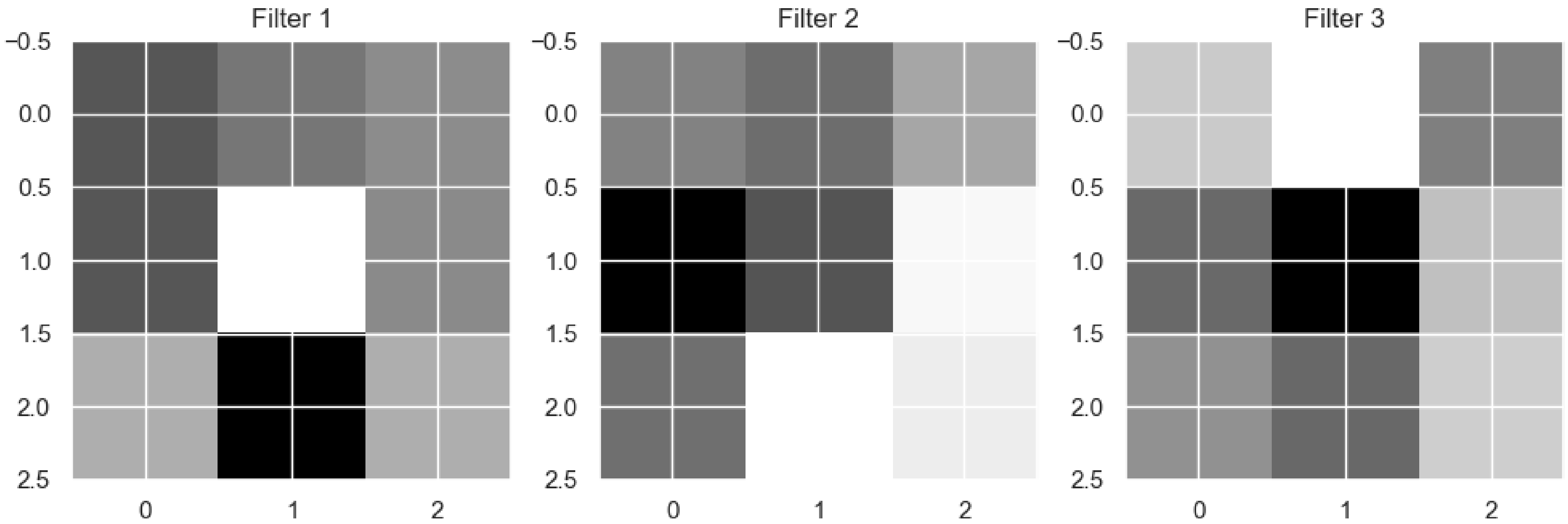
Answer: The outputs are the class probabilities

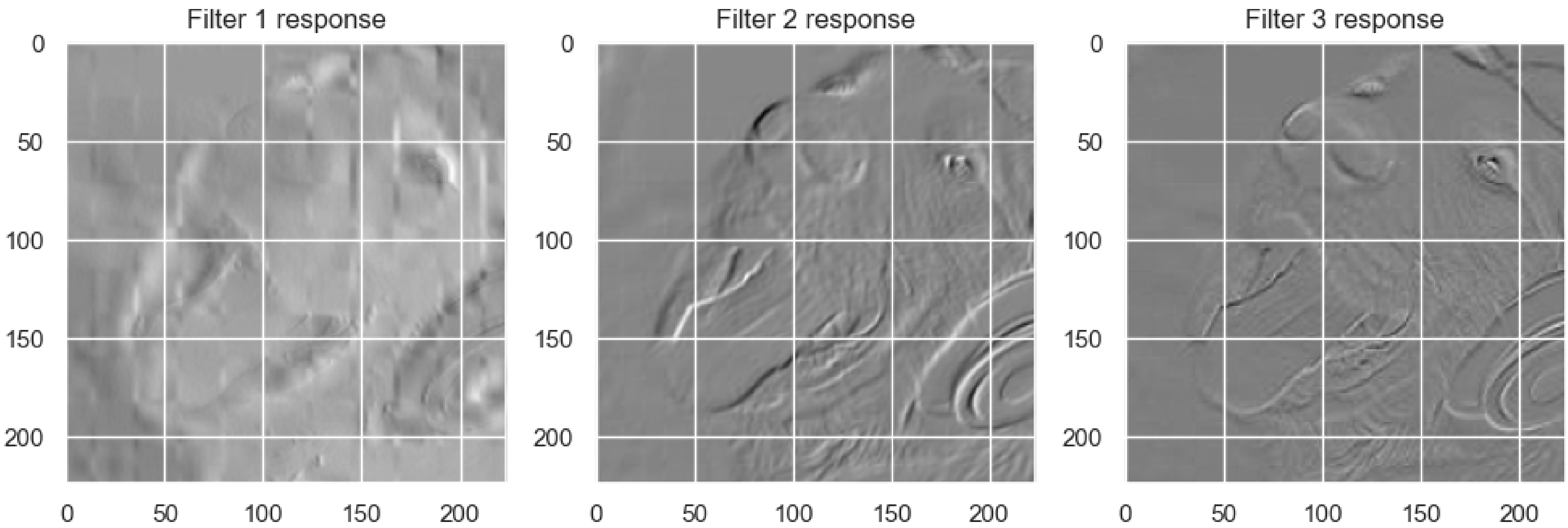
6

In [76]:

```python
# Access the filters in the first convolutional layer
filters = vgg16.features[0].weight.data.cpu().clone()

# Plot the first 3 filters
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
for i, ax in enumerate(axs.flat):
    # The filters have shape [out_channels, in_channels, kernel_height, kernel_width]
    # Since they are convolving RGB images, in_channels = 3. We'll take the mean to visualize them.
    ax.imshow(torch.mean(filters[i], dim=0), cmap='gray')
    ax.set_title(f'Filter {i+1}')
plt.show()
```

```
# Get the response from the first layer for the dog image
dog_image_prepared = dog_tensor.unsqueeze(0).to(next(vgg16.parameters()).device)
response = vgg16.features[:1](dog_image_prepared)

# Plot the response of the first 3 filters
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
for i, ax in enumerate(axs.flat):
    ax.imshow(response[0, i].cpu().detach().numpy(), cmap='gray')
    ax.set_title(f'Filter {i+1} response')
plt.show()
```





Filter 1 Response: The activation map indicates that this filter is likely sensitive to horizontal or near-horizontal edges. You can see the bright regions where the horizontal edges of the image features are located.

Filter 2 Response: This filter seems to respond to another set of features, possibly edges with a different orientation or a specific texture pattern. The activations are not as uniform as Filter 1, suggesting it might be detecting more complex features.

Filter 3 Response: The response here is more dispersed across the image, which could mean that this filter is detecting a broader range of features or perhaps is sensitive to a particular texture that is distributed more uniformly in the image.