1 Introduction

I. Softmax Derivative - Answer

Let us assume that $i = k$:

$$\frac{\partial \text{softmax}(x)_i}{\partial x_k} = \frac{\partial \frac{e^{x_k}}{\sum_{j=1}^{N} e^{x_j}}}{\partial x_k} = \frac{e^{x_k} \cdot \sum_{j=1}^{N} e^{x_j} - e^{x_k} \cdot e^{x_k}}{(\sum_{j=1}^{N} e^{x_j})^2} = \frac{e^{x_k} \cdot (\sum_{j=1}^{N} e^{x_j} - e^{x_k})}{(\sum_{j=1}^{N} e^{x_j})^2}$$

$$= \text{softmax}(x)_k \cdot (1 - \text{softmax}(x)_k)$$

For $i \neq k$:

$$\frac{\partial \text{softmax}(x)_i}{\partial x_k} = \frac{\partial \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}}{\partial x_k} = \frac{0 - e^{x_k} \cdot e^{x_i}}{(\sum_{j=1}^{N} e^{x_j})^2} = -\text{softmax}(x)_k \cdot \text{softmax}(x)_i$$

II. Cross-Entropy Gradient - Answer

The gradient of the cross-entropy loss is derived as follows:

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

$$= -\sum_i y_i \frac{\partial}{\partial \theta} \log(\text{softmax}(\theta)_i) = -\sum_i \frac{y_i}{\text{softmax}(\theta)_i} \cdot \frac{\partial}{\partial \theta} \text{softmax}(\theta)_i$$

Where:

$$\frac{\partial}{\partial \theta_j} \text{softmax}(\theta)_i = \begin{cases} \text{softmax}(\theta)_i \cdot (1 - \text{softmax}(\theta)_i), & \text{if } i = j \\ -\text{softmax}(\theta)_i \cdot \text{softmax}(\theta)_j, & \text{if } i \neq j \end{cases}$$

Submission instructions

Submission in pairs unless otherwise authorized

- **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 files:**
  - **As requested in Q2.a and Q3.a**
- **PY file:**
  - **As requested in Q3.a**

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

Question 2

I. Derivative Of Activation Functions (10pt)

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

In [1]:

```
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 num
erical stability
    return exp_x / exp_x.sum(dim=-1, keepdim=True)
```

In [2]:

```
def d_sigmoid(x):
    return torch.exp(-x) / (1 + torch.exp(-x))**2


def d_tanh(x):
    return 1 - torch.div((torch.exp(x) - torch.exp(-x))**2, (torch.exp(x) + torch.exp(-x
))**2)

def d_softmax(x):
    s = softmax(x)
    if len(x.shape) == 1:
```

```
        return torch.diag(s) - torch.outer(s, s)
    else:
        return torch.diag_embed(s) - torch.bmm(s.unsqueeze(-1), s.unsqueeze(1))
```

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 forward and backward passes 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 [3]:

```
import torch
import torchvision
from torch.utils.data import DataLoader

import os
import matplotlib.pyplot as plt
```

In [4]:

```
# 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 [5]:

```
# 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 [6]:

```
class FullyConnectedNetwork:
    def __init__(self, input_size, output_size, hidden_size1, activation=sigmoid, d_acti
vation=d_sigmoid, lr=0.01):
        # parameters
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size1 = hidden_size1

        # activation function and its derivative
        self.activation = activation
        self.d_activation = d_activation

        # 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):
        """Forward pass."""
        self.z1 = torch.matmul(X, self.W1) + self.b1  # Linear transformation
        self.h = self.activation(self.z1)  # Activation (hidden layer)
        self.z2 = torch.matmul(self.h, self.W2) + self.b2  # Linear transformation
        return softmax(self.z2)

    def backward(self, X, y, y_hat):
        """Backward pass to compute gradients"""
        batch_size = y.size(0)

        # Output layer gradients
        dl_dz2 = (y_hat - y) / batch_size  # Cross-entropy derivative
        dl_dW2 = torch.matmul(self.h.T, dl_dz2)
        dl_db2 = dl_dz2.sum(dim=0)

        # Hidden layer gradients
        dl_dh = torch.matmul(dl_dz2, self.W2.T)
        dl_dz1 = dl_dh * self.d_activation(self.z1)
        dl_dW1 = torch.matmul(X.T, dl_dz1)
        dl_db1 = dl_dz1.sum(dim=0)

        # Update weights and biases
        self.W2 -= self.lr * dl_dW2
        self.b2 -= self.lr * dl_db2
        self.W1 -= self.lr * dl_dW1
        self.b1 -= self.lr * dl_db1

    def train(self, X, y):
```

```
        """Train the network on a single batch."""
        y_hat = self.forward(X)  # Forward pass
        self.backward(X, y, y_hat)  # Backward pass
```

In [7]:

```
def train_model_Q2(model):
    train_losses, train_accuracy = [], []
    test_losses, test_accuracy = [], []
    for i in range(EPOCHS):
        running_loss_train = 0
        correct, total = 0, 0
        running_loss_test = 0
        correct_predictions, total_predictions = 0, 0
        for x, y in train_dataloader:
            # training data:
            x = x.view(x.size(0), -1)
            y_one_hot = one_hot(y, NUM_OF_CLASSES)
            model.train(x, y_one_hot)
            output = model.forward(x)
            loss = cross_entropy(y, output)
            running_loss_train += loss.item()
            _, predicted_train = torch.max(output.data, 1)
            total += y.size(0)
            correct += (predicted_train == y).sum().item()

        epoch_loss_train = running_loss_train / len(train_dataloader)
        epoch_accuracy_train = 100 * correct / total
        train_losses.append(epoch_loss_train)
        train_accuracy.append(epoch_accuracy_train)

        for x, y in test_dataloader:
            with torch.no_grad():
                x = x.view(x.size(0), -1)  # Flatten input if needed
                y_one_hot_test = one_hot(y, NUM_OF_CLASSES)
                output_test = model.forward(x)
                loss_test = cross_entropy(y, output_test)
                running_loss_test += loss_test.item()
                _, predicted_test = torch.max(output_test, dim=1)
                correct_predictions += (predicted_test == y).sum().item()
                total_predictions += y.size(0)

        avg_loss_test = running_loss_test / len(test_dataloader)
        accuracy_test = 100 * correct_predictions / total_predictions
        test_losses.append(avg_loss_test)
        test_accuracy.append(accuracy_test)

        print(f"Epoch {i} Train Loss: {epoch_loss_train:.4f} Train: Accuracy: {epoch_acc
uracy_train:.2f}, Test Loss: {avg_loss_test:.4f}, Test Acuracy: {accuracy_test:.2f}")
    return train_losses, train_accuracy, test_losses, test_accuracy
```

In [8]:

```
lrs = [0.01, 0.05, 0.1]
model_stats = {}
i=0
students_ids = '337604821_340915156'
base_filename = f'HW1_{students_ids}'

for lr in lrs:
    set_seed(SEED)
    model = FullyConnectedNetwork(784, 10, 128, lr=lr)
    print("new model with learning rate of:", lr)
    train_losses, train_accuracy, test_losses, test_accuracy = train_model_Q2(model)
    model_stats[lr] = (train_losses, train_accuracy, test_losses, test_accuracy)

    filename = f'{base_filename}_{lr}.pkl'
    torch.save({"W1": model.W1, "W2": model.W2, "b1": model.b1, "b2": model.b2}, filenam
e)
```

```
new model with learning rate of: 0.01
Epoch 0 Train Loss: 3.5529 Train: Accuracy: 36.81, Test Loss: 1.8211, Test Acuracy: 56.35
```

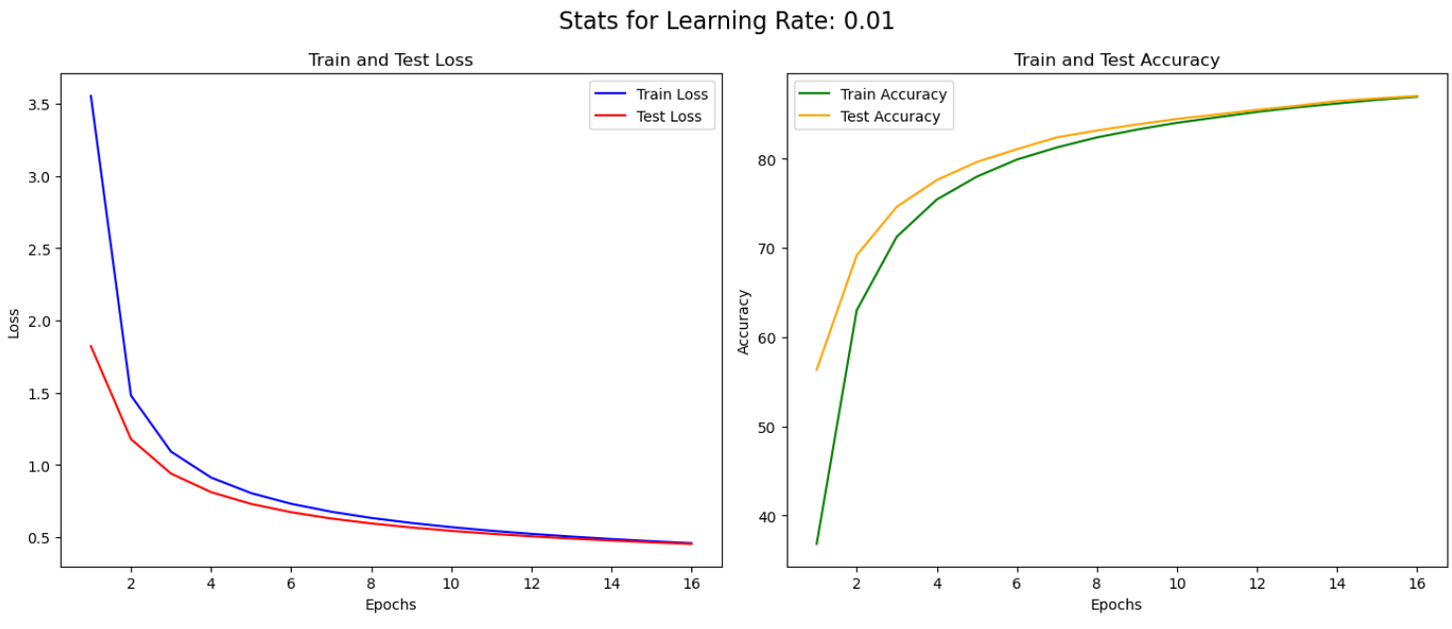
```
Epoch 1 Train Loss: 1.4797 Train: Accuracy: 62.99, Test Loss: 1.1778, Test Acuracy: 69.16
Epoch 2 Train Loss: 1.0913 Train: Accuracy: 71.23, Test Loss: 0.9392, Test Acuracy: 74.59
Epoch 3 Train Loss: 0.9118 Train: Accuracy: 75.40, Test Loss: 0.8109, Test Acuracy: 77.59
Epoch 4 Train Loss: 0.8039 Train: Accuracy: 77.97, Test Loss: 0.7291, Test Acuracy: 79.61
Epoch 5 Train Loss: 0.7300 Train: Accuracy: 79.88, Test Loss: 0.6715, Test Acuracy: 81.04
Epoch 6 Train Loss: 0.6751 Train: Accuracy: 81.24, Test Loss: 0.6282, Test Acuracy: 82.37
Epoch 7 Train Loss: 0.6321 Train: Accuracy: 82.37, Test Loss: 0.5939, Test Acuracy: 83.14
Epoch 8 Train Loss: 0.5972 Train: Accuracy: 83.24, Test Loss: 0.5659, Test Acuracy: 83.81
Epoch 9 Train Loss: 0.5680 Train: Accuracy: 84.00, Test Loss: 0.5424, Test Acuracy: 84.43
Epoch 10 Train Loss: 0.5432 Train: Accuracy: 84.62, Test Loss: 0.5222, Test Acuracy: 84.9
4
Epoch 11 Train Loss: 0.5216 Train: Accuracy: 85.22, Test Loss: 0.5047, Test Acuracy: 85.4
5
Epoch 12 Train Loss: 0.5027 Train: Accuracy: 85.73, Test Loss: 0.4892, Test Acuracy: 85.9
1
Epoch 13 Train Loss: 0.4859 Train: Accuracy: 86.16, Test Loss: 0.4754, Test Acuracy: 86.4
4
Epoch 14 Train Loss: 0.4708 Train: Accuracy: 86.58, Test Loss: 0.4631, Test Acuracy: 86.7
4
Epoch 15 Train Loss: 0.4572 Train: Accuracy: 86.92, Test Loss: 0.4519, Test Acuracy: 87.0
1
new model with learning rate of: 0.05
Epoch 0 Train Loss: 1.4990 Train: Accuracy: 65.82, Test Loss: 0.7279, Test Acuracy: 79.82
Epoch 1 Train Loss: 0.6127 Train: Accuracy: 82.78, Test Loss: 0.5406, Test Acuracy: 84.64
Epoch 2 Train Loss: 0.4838 Train: Accuracy: 86.21, Test Loss: 0.4617, Test Acuracy: 86.61
Epoch 3 Train Loss: 0.4167 Train: Accuracy: 87.99, Test Loss: 0.4148, Test Acuracy: 87.88
Epoch 4 Train Loss: 0.3731 Train: Accuracy: 89.23, Test Loss: 0.3827, Test Acuracy: 88.73
Epoch 5 Train Loss: 0.3416 Train: Accuracy: 90.06, Test Loss: 0.3588, Test Acuracy: 89.33
Epoch 6 Train Loss: 0.3174 Train: Accuracy: 90.71, Test Loss: 0.3400, Test Acuracy: 89.80
Epoch 7 Train Loss: 0.2980 Train: Accuracy: 91.27, Test Loss: 0.3246, Test Acuracy: 90.21
Epoch 8 Train Loss: 0.2820 Train: Accuracy: 91.70, Test Loss: 0.3117, Test Acuracy: 90.56
Epoch 9 Train Loss: 0.2685 Train: Accuracy: 92.12, Test Loss: 0.3008, Test Acuracy: 90.88
Epoch 10 Train Loss: 0.2569 Train: Accuracy: 92.47, Test Loss: 0.2914, Test Acuracy: 91.2
1
Epoch 11 Train Loss: 0.2468 Train: Accuracy: 92.74, Test Loss: 0.2831, Test Acuracy: 91.4
0
Epoch 12 Train Loss: 0.2378 Train: Accuracy: 93.00, Test Loss: 0.2759, Test Acuracy: 91.5
9
Epoch 13 Train Loss: 0.2298 Train: Accuracy: 93.23, Test Loss: 0.2694, Test Acuracy: 91.7
9
Epoch 14 Train Loss: 0.2226 Train: Accuracy: 93.43, Test Loss: 0.2635, Test Acuracy: 91.9
4
Epoch 15 Train Loss: 0.2160 Train: Accuracy: 93.65, Test Loss: 0.2583, Test Acuracy: 92.1
7
new model with learning rate of: 0.1
Epoch 0 Train Loss: 0.9937 Train: Accuracy: 75.44, Test Loss: 0.5417, Test Acuracy: 84.81
Epoch 1 Train Loss: 0.4247 Train: Accuracy: 87.89, Test Loss: 0.4182, Test Acuracy: 87.91
Epoch 2 Train Loss: 0.3383 Train: Accuracy: 90.20, Test Loss: 0.3616, Test Acuracy: 89.26
Epoch 3 Train Loss: 0.2921 Train: Accuracy: 91.52, Test Loss: 0.3267, Test Acuracy: 90.14
Epoch 4 Train Loss: 0.2617 Train: Accuracy: 92.41, Test Loss: 0.3025, Test Acuracy: 90.87
Epoch 5 Train Loss: 0.2396 Train: Accuracy: 93.04, Test Loss: 0.2845, Test Acuracy: 91.34
Epoch 6 Train Loss: 0.2226 Train: Accuracy: 93.55, Test Loss: 0.2705, Test Acuracy: 91.74
Epoch 7 Train Loss: 0.2088 Train: Accuracy: 94.00, Test Loss: 0.2593, Test Acuracy: 92.16
Epoch 8 Train Loss: 0.1973 Train: Accuracy: 94.31, Test Loss: 0.2499, Test Acuracy: 92.39
Epoch 9 Train Loss: 0.1875 Train: Accuracy: 94.61, Test Loss: 0.2420, Test Acuracy: 92.76
Epoch 10 Train Loss: 0.1789 Train: Accuracy: 94.91, Test Loss: 0.2353, Test Acuracy: 93.0
0
Epoch 11 Train Loss: 0.1713 Train: Accuracy: 95.17, Test Loss: 0.2294, Test Acuracy: 93.2
1
Epoch 12 Train Loss: 0.1644 Train: Accuracy: 95.40, Test Loss: 0.2243, Test Acuracy: 93.3
4
Epoch 13 Train Loss: 0.1583 Train: Accuracy: 95.59, Test Loss: 0.2197, Test Acuracy: 93.4
2
Epoch 14 Train Loss: 0.1526 Train: Accuracy: 95.78, Test Loss: 0.2156, Test Acuracy: 93.5
3
Epoch 15 Train Loss: 0.1474 Train: Accuracy: 95.93, Test Loss: 0.2119, Test Acuracy: 93.6
6
```

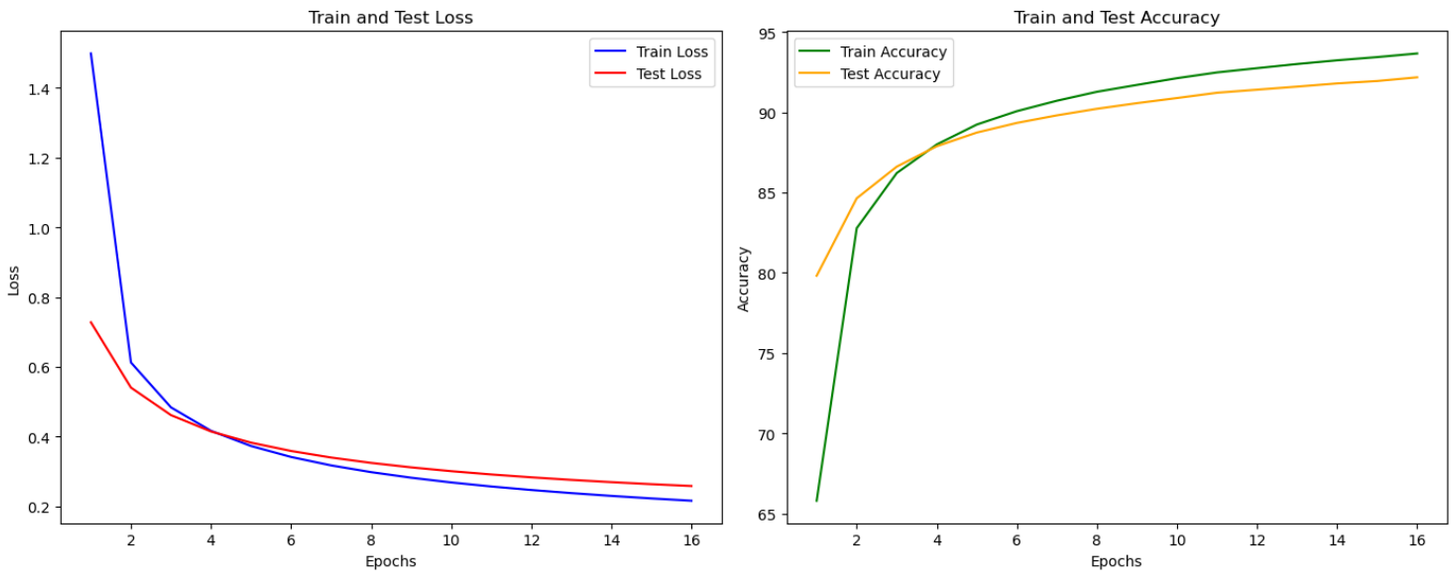
b) Train the model with various learning rates (at least 3).

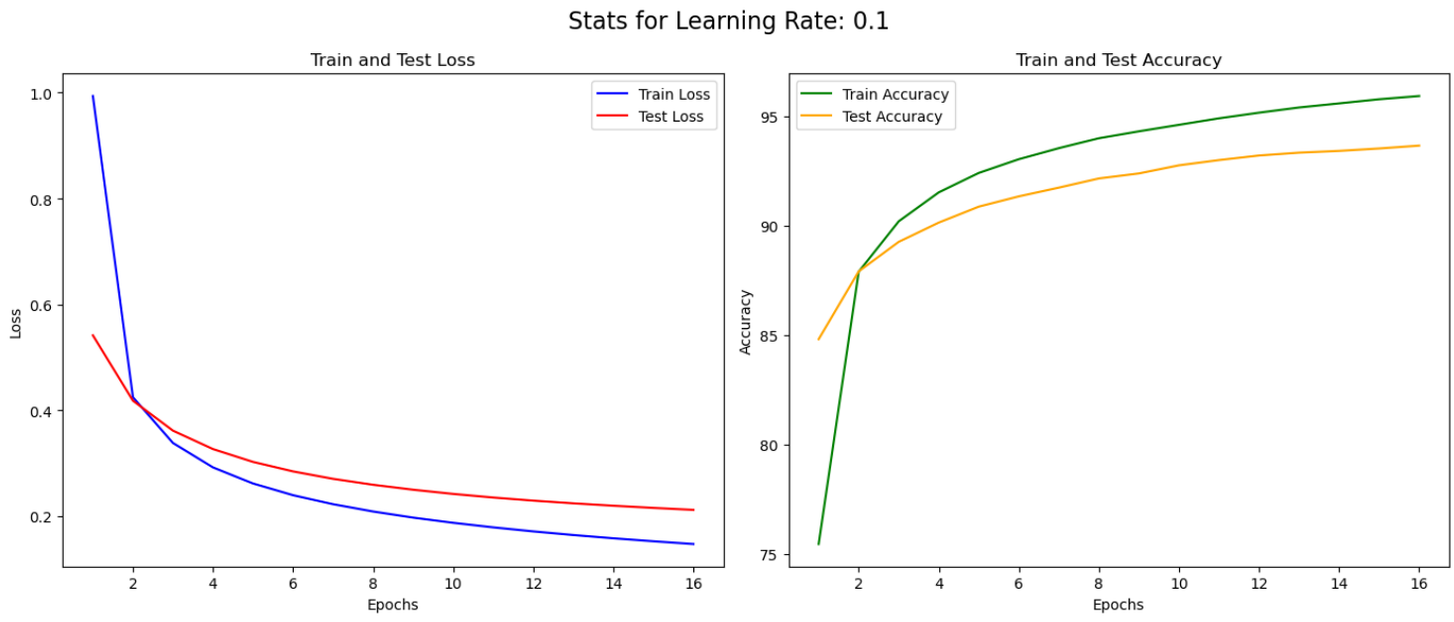
- **Plot the model's accuracy and loss (both on train and test sets) as a function of the epochs.**
- **Discuss the differences in training with different learning rates. Support your answer with plots.**

In [9]:

```
epochs = range(1, len(train_losses) + 1)
for lr, (train_losses, train_accuracy, test_losses, test_accuracy) in model_stats.items(
):
    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    axs[0].plot(epochs, train_losses, label='Train Loss', color='blue')
    axs[0].plot(epochs, test_losses, label='Test Loss', color='red')
    axs[0].set_title('Train and Test Loss')
    axs[0].set_xlabel('Epochs')
    axs[0].set_ylabel('Loss')
    axs[0].legend()

    axs[1].plot(epochs, train_accuracy, label='Train Accuracy', color='green')
    axs[1].plot(epochs, test_accuracy, label='Test Accuracy', color='orange')
    axs[1].set_title('Train and Test Accuracy')
    axs[1].set_xlabel('Epochs')
    axs[1].set_ylabel('Accuracy')
    axs[1].legend()

    fig.suptitle(f'Stats for Learning Rate: {lr}', fontsize=16)

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







As we can see, the plots show that higher learning rates lead to a faster descent in loss, enabling the model to converge more quickly - for higher learning rates we get better accuracy on test (on last epochs)

However, this comes at the cost of overfitting, as indicated by the growing gap between training and test loss (as well as accuracy) for higher learning rates. While higher learning rates achieve better test accuracy initially, the train test accuracy gap increases, usually reflecting on poorer generalization. In contrast, lower learning rates converge more slowly but maintain a smaller gap between training and test metrics (lr=0.01), which may result in more stable learning and better generalization for further epochs.

Question 3

I. Implement and Train a CNN (30pt)

As you might know, there are many dogs on campus. Sometimes, understanding the emotions of a dog can be challenging, and people might mistakenly try to pet it when it is sad or angry. As a data scientist, you have been asked to assist Technion's students. Your task is to create a "dog emotion classifier.
Your code should meet the following constraints:

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

To satisfy your boss, your model must achieve at least 70% accuracy on the test set. Your boss also emphasized that the model will be deployed on smartphones, so it should have a small number of parameters. 25% of your grade for this task will be based on the number of parameters your model uses — fewer parameters will yield a higher grade.

Stages

1. **Perform a short EDA (Exploratory Data Analysis).**
2. **Train the model and plot its accuracy and loss (for both the training and validation sets) as a function of the epochs.**
3. **Report the test set accuracy.**
4. **Discuss the progress you made and describe your final model.**
   **
**
   Your data is in `hw1_data/dog_emotion`.
   Tou can define a custom dataset (as in tutorial 3) or use `torchvision.datasets.ImageFolder`.

Submission

In addition to the code in the notebook, you should submit:

- **a `.py` file containing your model class.**
- **a `.pkl` file containing the weight of your model**

In [1]:

```
import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Subset
from model import CNN
import matplotlib.pyplot as plt
import os
from collections import Counter
```

In [2]:

```
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)

set_seed(42)
```

In [3]:

```
mus = (0.4785, 0.4509, 0.3911)
sigmas = (0.2281, 0.2223, 0.2189)

transform_train = transforms.Compose([
    transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.05),
    transforms.ToTensor(),
    transforms.Normalize(mus, sigmas)
])

transform_test = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mus, sigmas)
])

project_path = r'C:/Users/yumif/Desktop/projects/machine_learning_hw/'

train_dataset = dsets.ImageFolder(root=project_path+'ML2/HW1/hw1_data/Dog_Emotion/train',
transform=transform_train)
val_dataset = dsets.ImageFolder(root=project_path+'ML2/HW1/hw1_data/Dog_Emotion/val', tra
nsform=transform_test)
test_dataset = dsets.ImageFolder(root=project_path+'ML2/HW1/hw1_data/Dog_Emotion/test',
transform=transform_test)
```

In [4]:

```
# Hyper Parameters
num_epochs = 300
batch_size = 32
learning_rate = 0.0005
decay = 1e-4
```

```
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

1)

In [ ]:

```
print('--- Dataset size ---')
print(f"Training set size: {len(train_dataset)}")
print(f"Validation set size: {len(val_dataset)}")
print(f"Test set size: {len(test_dataset)}")

print('--- Classes ---')
print(f"Classes: {train_dataset.classes}")
print(f"Class-to-Index Mapping: {train_dataset.class_to_idx}")

train_labels = [label for _, label in train_dataset]
train_label_counts = Counter(train_labels)

plt.bar(train_dataset.classes, train_label_counts.values())
plt.xlabel('Classes')
plt.ylabel('Number of Samples')
plt.title('Class Distribution in Training Set')
plt.show()
```

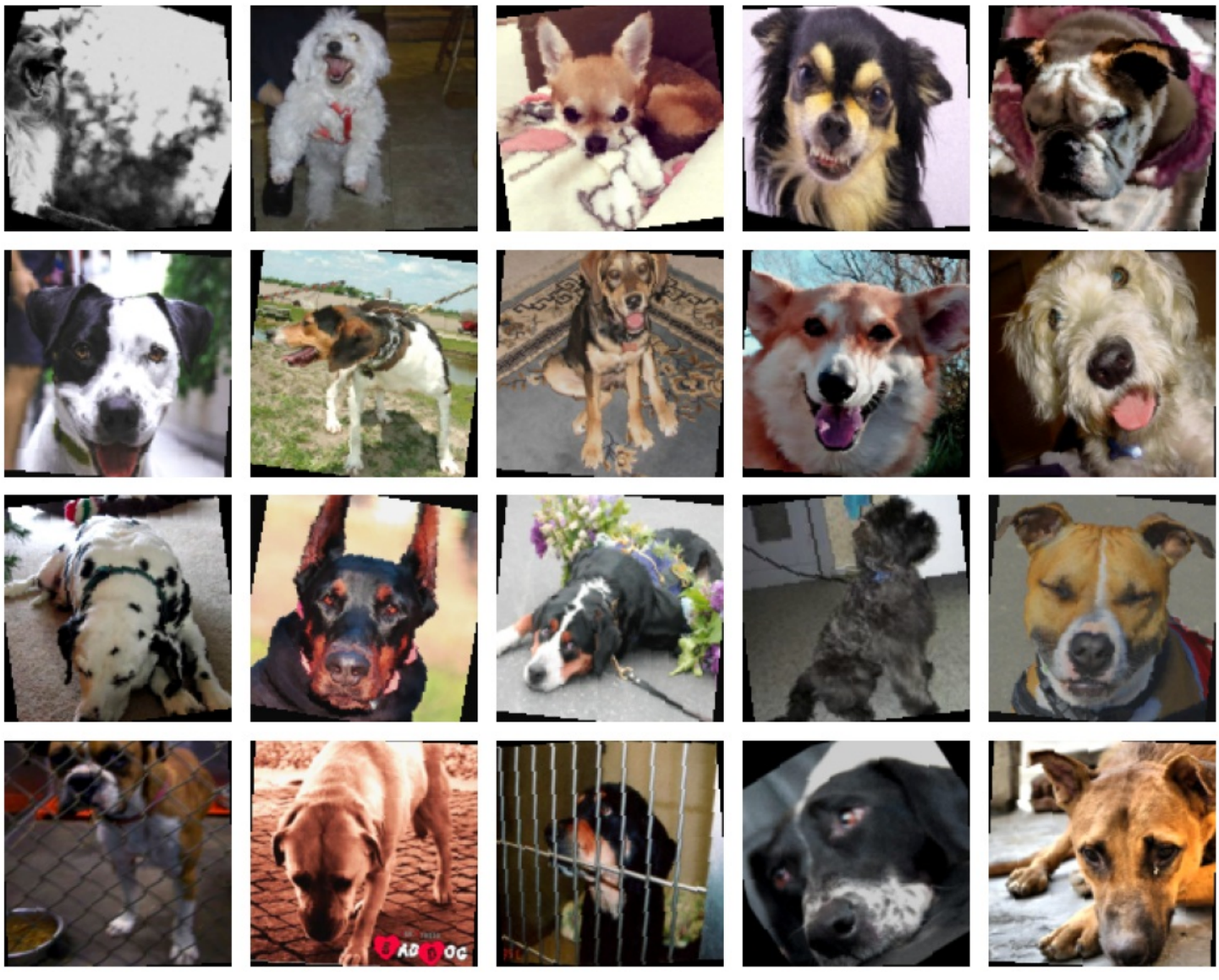
```
--- Dataset size ---
Training set size: 2800
Validation set size: 400
Test set size: 800
--- Classes ---
Classes: ['angry', 'happy', 'relaxed', 'sad']
Class-to-Index Mapping: {'angry': 0, 'happy': 1, 'relaxed': 2, 'sad': 3}
```

As we can see, the class distribution is balanced

In [ ]:

```
#
# Some pictures of training set after transformation . . .
#
```

```
class_images = {cls: [] for cls in [0, 1, 2, 3]}
num_classes = 4
num_samples = 5
mus_torch = torch.tensor(mus).view(3, 1, 1)
sigmas_torch = torch.tensor(sigmas).view(3, 1, 1)

# Take the sample images
for images, labels in train_loader:
    for img, label in zip(images, labels):
        label = label.item()  # Convert label to int
        if len(class_images[label]) < 5:
            class_images[label].append(img)

    if all(len(images) == num_samples for images in class_images.values()):
        break


fig, axes = plt.subplots(num_classes, num_samples, figsize=(num_samples * 2, num_classes
* 2))
for cls in range(num_classes):
    for sample_idx in range(num_samples):
        ax = axes[cls, sample_idx]
        img = class_images[cls][sample_idx] * sigmas_torch + mus_torch
        img = torch.clamp(img, 0, 1)
        ax.imshow(img.permute(1, 2, 0).numpy())
        ax.axis('off')
        if sample_idx == 0:
            ax.set_ylabel(f"Class {cls}", fontsize=12)
plt.tight_layout()
plt.show()
```



2)

In [11]:

```
cnn = CNN()

if torch.cuda.is_available():
    cnn = cnn.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn.parameters(), lr=learning_rate, weight_decay=decay)

# Reducing step a bit after A WHILE
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.8)

print('Number of parameters: ', sum(param.numel() for param in cnn.parameters()))
```

```
Number of parameters:  100116
```

In [ ]:

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

best_val_accuracy = 0
best_val_model_weights = None

best_test_accuracy = 0
best_test_model_weights = None
# save_interval = 5
```

In [ ]:

```
for epoch in range(num_epochs):
    running_loss = 0
    correct_train = 0
    total_train = 0

    # Training phase
    cnn.train()
    for i, (images, labels) in enumerate(train_loader):
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

        # Forward + Backward + Optimize
        outputs = cnn(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Track loss and accuracy for training
        running_loss += loss.item()
        predicted = torch.argmax(outputs, dim=1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    scheduler.step()

    # Compute training metrics
    epoch_train_loss = running_loss / len(train_loader)
    epoch_train_accuracy = 100 * correct_train / total_train

    train_losses.append(epoch_train_loss)
    train_accuracies.append(epoch_train_accuracy)

    # Validation phase
    cnn.eval()
    running_val_loss = 0
```

```
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for images, labels in val_loader:
            if torch.cuda.is_available():
                images = images.cuda()
                labels = labels.cuda()

            outputs = cnn(images)
            loss = criterion(outputs, labels)

            running_val_loss += loss.item()
            predicted = torch.argmax(outputs, dim=1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    # Compute validation metrics
    epoch_val_loss = running_val_loss / len(val_loader)
    epoch_val_accuracy = 100 * correct_val / total_val

    val_losses.append(epoch_val_loss)
    val_accuracies.append(epoch_val_accuracy)

    # Track the best model with highest val accuracy
    if epoch_val_accuracy > best_val_accuracy:
        best_val_accuracy = epoch_val_accuracy
        best_val_model_weights = cnn.state_dict()

        torch.save(best_val_model_weights, project_path + 'ML2/HW1/models/best_val_model
.pkl')
        print(f'Best Val Model Saved at Epoch {epoch+1} with Val Accuracy: {best_val_accu
racy:.3f}%')

    # Print epoch results
    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_train_loss:.3f}, Train Ac
curacy: {epoch_train_accuracy:.3f}%')
    print(f'Epoch [{epoch+1}/{num_epochs}], Val Loss: {epoch_val_loss:.3f}, Val Accuracy
: {epoch_val_accuracy:.3f}%')

    # Checking test every couple of epochs or so to see whats up
    if epoch >= 200 and epoch % 5 == 0:
        cnn.eval()
        correct_test = 0
        total_test = 0

        with torch.no_grad():
            for images, labels in test_loader:
                if torch.cuda.is_available():
                    images = images.cuda()
                    labels = labels.cuda()

                outputs = cnn(images)
                predicted = torch.argmax(outputs, dim=1)
                total_test += labels.size(0)
                correct_test += (predicted == labels).sum().item()

        test_accuracy = 100 * correct_test / total_test

        if test_accuracy > best_test_accuracy:
            best_test_accuracy = test_accuracy
            best_test_model_weights = cnn.state_dict()

            torch.save(best_test_model_weights, project_path + 'ML2/HW1/models/best_test
_model.pkl')
            print(f'Best Test Model Saved at Epoch {epoch+1} with Test Accuracy: {best_t
est_accuracy:.3f}%')

        print(f'Test Accuracy at Epoch {epoch+1}: {test_accuracy:.3f}%')
```

```
Best Val Model Saved at Epoch 1 with Val Accuracy: 30.000%
Epoch [1/300], Train Loss: 1.370, Train Accuracy: 28.643%
```

```
Epoch [1/300], Val Loss: 1.359, Val Accuracy: 30.000%
Epoch [2/300], Train Loss: 1.353, Train Accuracy: 31.071%
Epoch [2/300], Val Loss: 1.356, Val Accuracy: 29.750%
Best Val Model Saved at Epoch 3 with Val Accuracy: 31.000%
Epoch [3/300], Train Loss: 1.340, Train Accuracy: 32.929%
Epoch [3/300], Val Loss: 1.337, Val Accuracy: 31.000%
Best Val Model Saved at Epoch 4 with Val Accuracy: 32.250%
Epoch [4/300], Train Loss: 1.325, Train Accuracy: 34.179%
Epoch [4/300], Val Loss: 1.333, Val Accuracy: 32.250%
Best Val Model Saved at Epoch 5 with Val Accuracy: 32.500%
Epoch [5/300], Train Loss: 1.315, Train Accuracy: 33.286%
Epoch [5/300], Val Loss: 1.323, Val Accuracy: 32.500%
Best Val Model Saved at Epoch 6 with Val Accuracy: 33.000%
Epoch [6/300], Train Loss: 1.320, Train Accuracy: 33.286%
Epoch [6/300], Val Loss: 1.318, Val Accuracy: 33.000%
Epoch [7/300], Train Loss: 1.310, Train Accuracy: 34.393%
Epoch [7/300], Val Loss: 1.330, Val Accuracy: 31.250%
Epoch [8/300], Train Loss: 1.307, Train Accuracy: 34.893%
Epoch [8/300], Val Loss: 1.329, Val Accuracy: 32.750%
Best Val Model Saved at Epoch 9 with Val Accuracy: 34.000%
Epoch [9/300], Train Loss: 1.306, Train Accuracy: 34.286%
Epoch [9/300], Val Loss: 1.303, Val Accuracy: 34.000%
Epoch [10/300], Train Loss: 1.298, Train Accuracy: 37.071%
Epoch [10/300], Val Loss: 1.314, Val Accuracy: 34.000%
Epoch [11/300], Train Loss: 1.297, Train Accuracy: 35.250%
Epoch [11/300], Val Loss: 1.306, Val Accuracy: 33.250%
Epoch [12/300], Train Loss: 1.300, Train Accuracy: 35.250%
Epoch [12/300], Val Loss: 1.304, Val Accuracy: 32.750%
Epoch [13/300], Train Loss: 1.293, Train Accuracy: 35.393%
Epoch [13/300], Val Loss: 1.307, Val Accuracy: 31.750%
Best Val Model Saved at Epoch 14 with Val Accuracy: 35.250%
Epoch [14/300], Train Loss: 1.290, Train Accuracy: 35.429%
Epoch [14/300], Val Loss: 1.296, Val Accuracy: 35.250%
Epoch [15/300], Train Loss: 1.285, Train Accuracy: 37.357%
Epoch [15/300], Val Loss: 1.323, Val Accuracy: 35.250%
Epoch [16/300], Train Loss: 1.282, Train Accuracy: 36.464%
Epoch [16/300], Val Loss: 1.314, Val Accuracy: 33.750%
Epoch [17/300], Train Loss: 1.287, Train Accuracy: 36.750%
Epoch [17/300], Val Loss: 1.296, Val Accuracy: 32.500%
Best Val Model Saved at Epoch 18 with Val Accuracy: 36.250%
Epoch [18/300], Train Loss: 1.282, Train Accuracy: 37.750%
Epoch [18/300], Val Loss: 1.283, Val Accuracy: 36.250%
Epoch [19/300], Train Loss: 1.277, Train Accuracy: 37.571%
Epoch [19/300], Val Loss: 1.307, Val Accuracy: 35.250%
Epoch [20/300], Train Loss: 1.280, Train Accuracy: 37.500%
Epoch [20/300], Val Loss: 1.322, Val Accuracy: 34.750%
Best Val Model Saved at Epoch 21 with Val Accuracy: 40.250%
Epoch [21/300], Train Loss: 1.278, Train Accuracy: 37.179%
Epoch [21/300], Val Loss: 1.281, Val Accuracy: 40.250%
Epoch [22/300], Train Loss: 1.273, Train Accuracy: 37.679%
Epoch [22/300], Val Loss: 1.285, Val Accuracy: 37.250%
Epoch [23/300], Train Loss: 1.266, Train Accuracy: 39.536%
Epoch [23/300], Val Loss: 1.281, Val Accuracy: 38.750%
Epoch [24/300], Train Loss: 1.272, Train Accuracy: 38.571%
Epoch [24/300], Val Loss: 1.281, Val Accuracy: 38.500%
Epoch [25/300], Train Loss: 1.262, Train Accuracy: 39.786%
Epoch [25/300], Val Loss: 1.293, Val Accuracy: 36.000%
Epoch [26/300], Train Loss: 1.253, Train Accuracy: 40.286%
Epoch [26/300], Val Loss: 1.363, Val Accuracy: 35.750%
Best Val Model Saved at Epoch 27 with Val Accuracy: 41.000%
Epoch [27/300], Train Loss: 1.252, Train Accuracy: 40.536%
Epoch [27/300], Val Loss: 1.266, Val Accuracy: 41.000%
Epoch [28/300], Train Loss: 1.255, Train Accuracy: 39.964%
Epoch [28/300], Val Loss: 1.275, Val Accuracy: 35.500%
Epoch [29/300], Train Loss: 1.254, Train Accuracy: 40.321%
Epoch [29/300], Val Loss: 1.276, Val Accuracy: 39.250%
Epoch [30/300], Train Loss: 1.251, Train Accuracy: 40.929%
Epoch [30/300], Val Loss: 1.282, Val Accuracy: 40.500%
Epoch [31/300], Train Loss: 1.249, Train Accuracy: 40.786%
Epoch [31/300], Val Loss: 1.268, Val Accuracy: 36.750%
Epoch [32/300], Train Loss: 1.234, Train Accuracy: 41.179%
Epoch [32/300], Val Loss: 1.271, Val Accuracy: 39.250%
```

```
Epoch [33/300], Train Loss: 1.240, Train Accuracy: 41.786%
Epoch [33/300], Val Loss: 1.246, Val Accuracy: 40.250%
Best Val Model Saved at Epoch 34 with Val Accuracy: 44.750%
Epoch [34/300], Train Loss: 1.224, Train Accuracy: 43.429%
Epoch [34/300], Val Loss: 1.285, Val Accuracy: 44.750%
Epoch [35/300], Train Loss: 1.230, Train Accuracy: 42.929%
Epoch [35/300], Val Loss: 1.225, Val Accuracy: 40.250%
Epoch [36/300], Train Loss: 1.233, Train Accuracy: 41.250%
Epoch [36/300], Val Loss: 1.248, Val Accuracy: 39.000%
Epoch [37/300], Train Loss: 1.224, Train Accuracy: 43.000%
Epoch [37/300], Val Loss: 1.218, Val Accuracy: 42.750%
Epoch [38/300], Train Loss: 1.221, Train Accuracy: 43.786%
Epoch [38/300], Val Loss: 1.224, Val Accuracy: 40.500%
Epoch [39/300], Train Loss: 1.212, Train Accuracy: 43.714%
Epoch [39/300], Val Loss: 1.244, Val Accuracy: 44.750%
Epoch [40/300], Train Loss: 1.211, Train Accuracy: 42.929%
Epoch [40/300], Val Loss: 1.231, Val Accuracy: 43.000%
Epoch [41/300], Train Loss: 1.207, Train Accuracy: 45.214%
Epoch [41/300], Val Loss: 1.272, Val Accuracy: 38.500%
Best Val Model Saved at Epoch 42 with Val Accuracy: 46.750%
Epoch [42/300], Train Loss: 1.196, Train Accuracy: 45.321%
Epoch [42/300], Val Loss: 1.199, Val Accuracy: 46.750%
Best Val Model Saved at Epoch 43 with Val Accuracy: 48.500%
Epoch [43/300], Train Loss: 1.198, Train Accuracy: 45.929%
Epoch [43/300], Val Loss: 1.207, Val Accuracy: 48.500%
Epoch [44/300], Train Loss: 1.182, Train Accuracy: 46.857%
Epoch [44/300], Val Loss: 1.217, Val Accuracy: 43.250%
Epoch [45/300], Train Loss: 1.192, Train Accuracy: 45.786%
Epoch [45/300], Val Loss: 1.199, Val Accuracy: 45.000%
Epoch [46/300], Train Loss: 1.187, Train Accuracy: 47.321%
Epoch [46/300], Val Loss: 1.192, Val Accuracy: 45.250%
Epoch [47/300], Train Loss: 1.190, Train Accuracy: 45.786%
Epoch [47/300], Val Loss: 1.237, Val Accuracy: 46.750%
Epoch [48/300], Train Loss: 1.174, Train Accuracy: 47.214%
Epoch [48/300], Val Loss: 1.239, Val Accuracy: 45.250%
Epoch [49/300], Train Loss: 1.171, Train Accuracy: 47.250%
Epoch [49/300], Val Loss: 1.150, Val Accuracy: 48.250%
Epoch [50/300], Train Loss: 1.156, Train Accuracy: 47.964%
Epoch [50/300], Val Loss: 1.186, Val Accuracy: 46.250%
Epoch [51/300], Train Loss: 1.164, Train Accuracy: 47.607%
Epoch [51/300], Val Loss: 1.183, Val Accuracy: 46.750%
Epoch [52/300], Train Loss: 1.150, Train Accuracy: 49.607%
Epoch [52/300], Val Loss: 1.220, Val Accuracy: 42.750%
Epoch [53/300], Train Loss: 1.143, Train Accuracy: 49.786%
Epoch [53/300], Val Loss: 1.286, Val Accuracy: 42.500%
Epoch [54/300], Train Loss: 1.159, Train Accuracy: 49.036%
Epoch [54/300], Val Loss: 1.172, Val Accuracy: 45.000%
Epoch [55/300], Train Loss: 1.136, Train Accuracy: 49.500%
Epoch [55/300], Val Loss: 1.198, Val Accuracy: 43.000%
Epoch [56/300], Train Loss: 1.144, Train Accuracy: 49.036%
Epoch [56/300], Val Loss: 1.208, Val Accuracy: 44.500%
Epoch [57/300], Train Loss: 1.112, Train Accuracy: 51.750%
Epoch [57/300], Val Loss: 1.172, Val Accuracy: 46.500%
Best Val Model Saved at Epoch 58 with Val Accuracy: 50.000%
Epoch [58/300], Train Loss: 1.124, Train Accuracy: 51.786%
Epoch [58/300], Val Loss: 1.124, Val Accuracy: 50.000%
Epoch [59/300], Train Loss: 1.118, Train Accuracy: 51.393%
Epoch [59/300], Val Loss: 1.158, Val Accuracy: 45.250%
Best Val Model Saved at Epoch 60 with Val Accuracy: 51.000%
Epoch [60/300], Train Loss: 1.104, Train Accuracy: 51.714%
Epoch [60/300], Val Loss: 1.105, Val Accuracy: 51.000%
Epoch [61/300], Train Loss: 1.098, Train Accuracy: 52.036%
Epoch [61/300], Val Loss: 1.146, Val Accuracy: 49.750%
Epoch [62/300], Train Loss: 1.096, Train Accuracy: 51.929%
Epoch [62/300], Val Loss: 1.109, Val Accuracy: 49.000%
Epoch [63/300], Train Loss: 1.095, Train Accuracy: 53.571%
Epoch [63/300], Val Loss: 1.157, Val Accuracy: 45.500%
Best Val Model Saved at Epoch 64 with Val Accuracy: 52.000%
Epoch [64/300], Train Loss: 1.089, Train Accuracy: 52.000%
Epoch [64/300], Val Loss: 1.093, Val Accuracy: 52.000%
Epoch [65/300], Train Loss: 1.098, Train Accuracy: 52.036%
Epoch [65/300], Val Loss: 1.175, Val Accuracy: 45.500%
```

```
Best Val Model Saved at Epoch 66 with Val Accuracy: 56.500%
Epoch [66/300], Train Loss: 1.094, Train Accuracy: 52.786%
Epoch [66/300], Val Loss: 1.086, Val Accuracy: 56.500%
Epoch [67/300], Train Loss: 1.092, Train Accuracy: 51.929%
Epoch [67/300], Val Loss: 1.125, Val Accuracy: 48.500%
Epoch [68/300], Train Loss: 1.081, Train Accuracy: 52.500%
Epoch [68/300], Val Loss: 1.166, Val Accuracy: 47.750%
Epoch [69/300], Train Loss: 1.075, Train Accuracy: 53.786%
Epoch [69/300], Val Loss: 1.081, Val Accuracy: 53.000%
Epoch [70/300], Train Loss: 1.074, Train Accuracy: 54.250%
Epoch [70/300], Val Loss: 1.151, Val Accuracy: 44.750%
Epoch [71/300], Train Loss: 1.069, Train Accuracy: 53.357%
Epoch [71/300], Val Loss: 1.121, Val Accuracy: 49.250%
Epoch [72/300], Train Loss: 1.057, Train Accuracy: 54.714%
Epoch [72/300], Val Loss: 1.061, Val Accuracy: 53.750%
Epoch [73/300], Train Loss: 1.044, Train Accuracy: 56.143%
Epoch [73/300], Val Loss: 1.203, Val Accuracy: 48.500%
Epoch [74/300], Train Loss: 1.063, Train Accuracy: 54.679%
Epoch [74/300], Val Loss: 1.113, Val Accuracy: 55.250%
Epoch [75/300], Train Loss: 1.051, Train Accuracy: 54.714%
Epoch [75/300], Val Loss: 1.116, Val Accuracy: 52.250%
Epoch [76/300], Train Loss: 1.047, Train Accuracy: 55.571%
Epoch [76/300], Val Loss: 1.306, Val Accuracy: 41.750%
Epoch [77/300], Train Loss: 1.033, Train Accuracy: 55.607%
Epoch [77/300], Val Loss: 1.037, Val Accuracy: 54.750%
Epoch [78/300], Train Loss: 1.042, Train Accuracy: 55.429%
Epoch [78/300], Val Loss: 1.082, Val Accuracy: 51.250%
Epoch [79/300], Train Loss: 1.040, Train Accuracy: 54.536%
Epoch [79/300], Val Loss: 1.040, Val Accuracy: 52.750%
Epoch [80/300], Train Loss: 1.041, Train Accuracy: 54.750%
Epoch [80/300], Val Loss: 1.055, Val Accuracy: 56.500%
Best Val Model Saved at Epoch 81 with Val Accuracy: 57.750%
Epoch [81/300], Train Loss: 1.050, Train Accuracy: 54.821%
Epoch [81/300], Val Loss: 1.050, Val Accuracy: 57.750%
Epoch [82/300], Train Loss: 1.032, Train Accuracy: 55.393%
Epoch [82/300], Val Loss: 1.049, Val Accuracy: 51.750%
Epoch [83/300], Train Loss: 1.029, Train Accuracy: 55.857%
Epoch [83/300], Val Loss: 1.024, Val Accuracy: 57.500%
Epoch [84/300], Train Loss: 1.028, Train Accuracy: 56.679%
Epoch [84/300], Val Loss: 1.062, Val Accuracy: 53.250%
Epoch [85/300], Train Loss: 1.021, Train Accuracy: 55.679%
Epoch [85/300], Val Loss: 1.029, Val Accuracy: 56.250%
Epoch [86/300], Train Loss: 1.011, Train Accuracy: 57.464%
Epoch [86/300], Val Loss: 1.001, Val Accuracy: 55.500%
Epoch [87/300], Train Loss: 1.006, Train Accuracy: 57.857%
Epoch [87/300], Val Loss: 1.030, Val Accuracy: 55.250%
Epoch [88/300], Train Loss: 1.006, Train Accuracy: 56.107%
Epoch [88/300], Val Loss: 1.002, Val Accuracy: 57.750%
Epoch [89/300], Train Loss: 1.003, Train Accuracy: 58.143%
Epoch [89/300], Val Loss: 1.071, Val Accuracy: 53.000%
Epoch [90/300], Train Loss: 1.019, Train Accuracy: 55.893%
Epoch [90/300], Val Loss: 1.067, Val Accuracy: 55.750%
Epoch [91/300], Train Loss: 1.003, Train Accuracy: 57.786%
Epoch [91/300], Val Loss: 1.100, Val Accuracy: 47.500%
Epoch [92/300], Train Loss: 1.016, Train Accuracy: 57.143%
Epoch [92/300], Val Loss: 1.004, Val Accuracy: 55.750%
Epoch [93/300], Train Loss: 1.001, Train Accuracy: 58.036%
Epoch [93/300], Val Loss: 1.031, Val Accuracy: 55.250%
Epoch [94/300], Train Loss: 0.995, Train Accuracy: 57.036%
Epoch [94/300], Val Loss: 1.024, Val Accuracy: 55.750%
Epoch [95/300], Train Loss: 0.997, Train Accuracy: 58.250%
Epoch [95/300], Val Loss: 1.074, Val Accuracy: 56.750%
Epoch [96/300], Train Loss: 0.990, Train Accuracy: 58.179%
Epoch [96/300], Val Loss: 1.036, Val Accuracy: 55.750%
Epoch [97/300], Train Loss: 0.970, Train Accuracy: 58.071%
Epoch [97/300], Val Loss: 1.009, Val Accuracy: 56.250%
Best Val Model Saved at Epoch 98 with Val Accuracy: 59.500%
Epoch [98/300], Train Loss: 0.988, Train Accuracy: 58.536%
Epoch [98/300], Val Loss: 0.980, Val Accuracy: 59.500%
Epoch [99/300], Train Loss: 0.959, Train Accuracy: 59.107%
Epoch [99/300], Val Loss: 1.014, Val Accuracy: 57.500%
Epoch [100/300], Train Loss: 0.953, Train Accuracy: 59.786%
```

```
Epoch [100/300], Val Loss: 0.983, Val Accuracy: 58.250%
Best Val Model Saved at Epoch 101 with Val Accuracy: 61.250%
Epoch [101/300], Train Loss: 0.957, Train Accuracy: 59.250%
Epoch [101/300], Val Loss: 0.943, Val Accuracy: 61.250%
Epoch [102/300], Train Loss: 0.961, Train Accuracy: 59.643%
Epoch [102/300], Val Loss: 1.018, Val Accuracy: 56.750%
Epoch [103/300], Train Loss: 0.965, Train Accuracy: 59.929%
Epoch [103/300], Val Loss: 0.953, Val Accuracy: 60.500%
Epoch [104/300], Train Loss: 0.951, Train Accuracy: 60.000%
Epoch [104/300], Val Loss: 1.044, Val Accuracy: 58.000%
Epoch [105/300], Train Loss: 0.954, Train Accuracy: 60.179%
Epoch [105/300], Val Loss: 0.960, Val Accuracy: 59.500%
Epoch [106/300], Train Loss: 0.938, Train Accuracy: 60.786%
Epoch [106/300], Val Loss: 0.969, Val Accuracy: 58.500%
Epoch [107/300], Train Loss: 0.938, Train Accuracy: 60.821%
Epoch [107/300], Val Loss: 0.966, Val Accuracy: 57.750%
Epoch [108/300], Train Loss: 0.933, Train Accuracy: 60.393%
Epoch [108/300], Val Loss: 0.961, Val Accuracy: 58.750%
Epoch [109/300], Train Loss: 0.933, Train Accuracy: 60.214%
Epoch [109/300], Val Loss: 0.955, Val Accuracy: 60.000%
Epoch [110/300], Train Loss: 0.952, Train Accuracy: 60.357%
Epoch [110/300], Val Loss: 0.976, Val Accuracy: 59.500%
Epoch [111/300], Train Loss: 0.931, Train Accuracy: 60.214%
Epoch [111/300], Val Loss: 0.947, Val Accuracy: 58.750%
Epoch [112/300], Train Loss: 0.953, Train Accuracy: 61.286%
Epoch [112/300], Val Loss: 1.032, Val Accuracy: 58.000%
Best Val Model Saved at Epoch 113 with Val Accuracy: 61.500%
Epoch [113/300], Train Loss: 0.939, Train Accuracy: 60.321%
Epoch [113/300], Val Loss: 0.918, Val Accuracy: 61.500%
Epoch [114/300], Train Loss: 0.947, Train Accuracy: 60.143%
Epoch [114/300], Val Loss: 0.925, Val Accuracy: 59.250%
Epoch [115/300], Train Loss: 0.932, Train Accuracy: 61.036%
Epoch [115/300], Val Loss: 1.004, Val Accuracy: 54.500%
Epoch [116/300], Train Loss: 0.933, Train Accuracy: 61.107%
Epoch [116/300], Val Loss: 0.933, Val Accuracy: 58.000%
Epoch [117/300], Train Loss: 0.919, Train Accuracy: 61.500%
Epoch [117/300], Val Loss: 0.990, Val Accuracy: 59.750%
Epoch [118/300], Train Loss: 0.924, Train Accuracy: 62.143%
Epoch [118/300], Val Loss: 0.942, Val Accuracy: 59.500%
Best Val Model Saved at Epoch 119 with Val Accuracy: 62.750%
Epoch [119/300], Train Loss: 0.917, Train Accuracy: 61.750%
Epoch [119/300], Val Loss: 0.913, Val Accuracy: 62.750%
Epoch [120/300], Train Loss: 0.898, Train Accuracy: 63.143%
Epoch [120/300], Val Loss: 0.949, Val Accuracy: 61.750%
Epoch [121/300], Train Loss: 0.924, Train Accuracy: 60.214%
Epoch [121/300], Val Loss: 0.944, Val Accuracy: 57.750%
Epoch [122/300], Train Loss: 0.893, Train Accuracy: 62.929%
Epoch [122/300], Val Loss: 0.933, Val Accuracy: 62.250%
Epoch [123/300], Train Loss: 0.908, Train Accuracy: 62.429%
Epoch [123/300], Val Loss: 0.903, Val Accuracy: 59.000%
Epoch [124/300], Train Loss: 0.891, Train Accuracy: 63.536%
Epoch [124/300], Val Loss: 0.901, Val Accuracy: 60.500%
Epoch [125/300], Train Loss: 0.888, Train Accuracy: 63.214%
Epoch [125/300], Val Loss: 0.947, Val Accuracy: 59.250%
Epoch [126/300], Train Loss: 0.881, Train Accuracy: 63.964%
Epoch [126/300], Val Loss: 0.934, Val Accuracy: 61.500%
Epoch [127/300], Train Loss: 0.914, Train Accuracy: 61.357%
Epoch [127/300], Val Loss: 0.900, Val Accuracy: 61.250%
Best Val Model Saved at Epoch 128 with Val Accuracy: 63.750%
Epoch [128/300], Train Loss: 0.904, Train Accuracy: 63.000%
Epoch [128/300], Val Loss: 0.889, Val Accuracy: 63.750%
Epoch [129/300], Train Loss: 0.910, Train Accuracy: 62.643%
Epoch [129/300], Val Loss: 0.918, Val Accuracy: 62.250%
Epoch [130/300], Train Loss: 0.882, Train Accuracy: 64.393%
Epoch [130/300], Val Loss: 0.994, Val Accuracy: 59.750%
Epoch [131/300], Train Loss: 0.889, Train Accuracy: 63.786%
Epoch [131/300], Val Loss: 0.953, Val Accuracy: 58.500%
Epoch [132/300], Train Loss: 0.872, Train Accuracy: 63.321%
Epoch [132/300], Val Loss: 0.936, Val Accuracy: 61.000%
Best Val Model Saved at Epoch 133 with Val Accuracy: 64.250%
Epoch [133/300], Train Loss: 0.888, Train Accuracy: 62.679%
Epoch [133/300], Val Loss: 0.912, Val Accuracy: 64.250%
```

```
Epoch [134/300], Train Loss: 0.889, Train Accuracy: 63.500%
Epoch [134/300], Val Loss: 0.907, Val Accuracy: 61.500%
Epoch [135/300], Train Loss: 0.885, Train Accuracy: 63.929%
Epoch [135/300], Val Loss: 0.910, Val Accuracy: 60.250%
Epoch [136/300], Train Loss: 0.873, Train Accuracy: 64.321%
Epoch [136/300], Val Loss: 1.070, Val Accuracy: 54.750%
Epoch [137/300], Train Loss: 0.890, Train Accuracy: 63.857%
Epoch [137/300], Val Loss: 0.895, Val Accuracy: 60.500%
Epoch [138/300], Train Loss: 0.876, Train Accuracy: 63.071%
Epoch [138/300], Val Loss: 0.981, Val Accuracy: 58.250%
Epoch [139/300], Train Loss: 0.864, Train Accuracy: 63.929%
Epoch [139/300], Val Loss: 0.906, Val Accuracy: 61.000%
Epoch [140/300], Train Loss: 0.870, Train Accuracy: 64.786%
Epoch [140/300], Val Loss: 0.916, Val Accuracy: 58.500%
Epoch [141/300], Train Loss: 0.874, Train Accuracy: 64.214%
Epoch [141/300], Val Loss: 0.972, Val Accuracy: 59.750%
Epoch [142/300], Train Loss: 0.872, Train Accuracy: 64.643%
Epoch [142/300], Val Loss: 0.905, Val Accuracy: 62.250%
Epoch [143/300], Train Loss: 0.877, Train Accuracy: 63.679%
Epoch [143/300], Val Loss: 0.877, Val Accuracy: 63.250%
Epoch [144/300], Train Loss: 0.866, Train Accuracy: 63.714%
Epoch [144/300], Val Loss: 0.935, Val Accuracy: 63.000%
Epoch [145/300], Train Loss: 0.882, Train Accuracy: 62.036%
Epoch [145/300], Val Loss: 0.938, Val Accuracy: 59.500%
Epoch [146/300], Train Loss: 0.858, Train Accuracy: 64.607%
Epoch [146/300], Val Loss: 0.928, Val Accuracy: 63.250%
Epoch [147/300], Train Loss: 0.870, Train Accuracy: 64.286%
Epoch [147/300], Val Loss: 0.881, Val Accuracy: 62.250%
Epoch [148/300], Train Loss: 0.878, Train Accuracy: 63.536%
Epoch [148/300], Val Loss: 0.949, Val Accuracy: 58.750%
Epoch [149/300], Train Loss: 0.853, Train Accuracy: 63.679%
Epoch [149/300], Val Loss: 0.872, Val Accuracy: 63.000%
Epoch [150/300], Train Loss: 0.870, Train Accuracy: 63.893%
Epoch [150/300], Val Loss: 0.918, Val Accuracy: 64.000%
Epoch [151/300], Train Loss: 0.834, Train Accuracy: 65.107%
Epoch [151/300], Val Loss: 0.899, Val Accuracy: 59.750%
Epoch [152/300], Train Loss: 0.864, Train Accuracy: 63.750%
Epoch [152/300], Val Loss: 0.889, Val Accuracy: 61.000%
Epoch [153/300], Train Loss: 0.856, Train Accuracy: 64.929%
Epoch [153/300], Val Loss: 1.001, Val Accuracy: 62.250%
Epoch [154/300], Train Loss: 0.846, Train Accuracy: 64.464%
Epoch [154/300], Val Loss: 0.925, Val Accuracy: 60.500%
Epoch [155/300], Train Loss: 0.848, Train Accuracy: 64.929%
Epoch [155/300], Val Loss: 0.878, Val Accuracy: 60.750%
Epoch [156/300], Train Loss: 0.849, Train Accuracy: 65.714%
Epoch [156/300], Val Loss: 0.918, Val Accuracy: 60.250%
Epoch [157/300], Train Loss: 0.848, Train Accuracy: 64.857%
Epoch [157/300], Val Loss: 0.862, Val Accuracy: 64.250%
Epoch [158/300], Train Loss: 0.847, Train Accuracy: 65.679%
Epoch [158/300], Val Loss: 0.928, Val Accuracy: 63.250%
Epoch [159/300], Train Loss: 0.839, Train Accuracy: 65.464%
Epoch [159/300], Val Loss: 0.846, Val Accuracy: 63.750%
Epoch [160/300], Train Loss: 0.841, Train Accuracy: 64.321%
Epoch [160/300], Val Loss: 0.889, Val Accuracy: 59.500%
Epoch [161/300], Train Loss: 0.831, Train Accuracy: 65.643%
Epoch [161/300], Val Loss: 0.899, Val Accuracy: 62.500%
Epoch [162/300], Train Loss: 0.843, Train Accuracy: 65.393%
Epoch [162/300], Val Loss: 0.950, Val Accuracy: 61.250%
Best Val Model Saved at Epoch 163 with Val Accuracy: 65.000%
Epoch [163/300], Train Loss: 0.853, Train Accuracy: 65.036%
Epoch [163/300], Val Loss: 0.841, Val Accuracy: 65.000%
Epoch [164/300], Train Loss: 0.817, Train Accuracy: 66.536%
Epoch [164/300], Val Loss: 0.865, Val Accuracy: 65.000%
Epoch [165/300], Train Loss: 0.836, Train Accuracy: 65.536%
Epoch [165/300], Val Loss: 0.846, Val Accuracy: 64.750%
Epoch [166/300], Train Loss: 0.822, Train Accuracy: 66.500%
Epoch [166/300], Val Loss: 0.866, Val Accuracy: 61.750%
Epoch [167/300], Train Loss: 0.835, Train Accuracy: 64.893%
Epoch [167/300], Val Loss: 0.855, Val Accuracy: 64.000%
Epoch [168/300], Train Loss: 0.825, Train Accuracy: 65.571%
Epoch [168/300], Val Loss: 0.877, Val Accuracy: 63.250%
Epoch [169/300], Train Loss: 0.846, Train Accuracy: 64.500%
```

```
Epoch [169/300], Val Loss: 0.821, Val Accuracy: 64.500%
Epoch [170/300], Train Loss: 0.816, Train Accuracy: 66.393%
Epoch [170/300], Val Loss: 0.846, Val Accuracy: 63.250%
Epoch [171/300], Train Loss: 0.819, Train Accuracy: 66.286%
Epoch [171/300], Val Loss: 0.832, Val Accuracy: 64.500%
Best Val Model Saved at Epoch 172 with Val Accuracy: 65.500%
Epoch [172/300], Train Loss: 0.816, Train Accuracy: 66.143%
Epoch [172/300], Val Loss: 0.830, Val Accuracy: 65.500%
Best Val Model Saved at Epoch 173 with Val Accuracy: 68.250%
Epoch [173/300], Train Loss: 0.822, Train Accuracy: 65.857%
Epoch [173/300], Val Loss: 0.825, Val Accuracy: 68.250%
Epoch [174/300], Train Loss: 0.825, Train Accuracy: 65.429%
Epoch [174/300], Val Loss: 0.832, Val Accuracy: 65.500%
Epoch [175/300], Train Loss: 0.816, Train Accuracy: 65.643%
Epoch [175/300], Val Loss: 0.858, Val Accuracy: 63.500%
Epoch [176/300], Train Loss: 0.833, Train Accuracy: 66.000%
Epoch [176/300], Val Loss: 0.850, Val Accuracy: 64.750%
Epoch [177/300], Train Loss: 0.824, Train Accuracy: 65.179%
Epoch [177/300], Val Loss: 0.852, Val Accuracy: 64.000%
Epoch [178/300], Train Loss: 0.799, Train Accuracy: 67.000%
Epoch [178/300], Val Loss: 0.872, Val Accuracy: 62.750%
Epoch [179/300], Train Loss: 0.815, Train Accuracy: 67.143%
Epoch [179/300], Val Loss: 0.892, Val Accuracy: 62.000%
Epoch [180/300], Train Loss: 0.815, Train Accuracy: 66.107%
Epoch [180/300], Val Loss: 0.914, Val Accuracy: 59.500%
Epoch [181/300], Train Loss: 0.833, Train Accuracy: 65.964%
Epoch [181/300], Val Loss: 0.845, Val Accuracy: 67.750%
Epoch [182/300], Train Loss: 0.799, Train Accuracy: 68.643%
Epoch [182/300], Val Loss: 0.809, Val Accuracy: 65.250%
Epoch [183/300], Train Loss: 0.815, Train Accuracy: 66.571%
Epoch [183/300], Val Loss: 0.853, Val Accuracy: 62.500%
Epoch [184/300], Train Loss: 0.802, Train Accuracy: 67.179%
Epoch [184/300], Val Loss: 0.843, Val Accuracy: 63.000%
Epoch [185/300], Train Loss: 0.805, Train Accuracy: 66.179%
Epoch [185/300], Val Loss: 0.825, Val Accuracy: 66.000%
Epoch [186/300], Train Loss: 0.781, Train Accuracy: 67.857%
Epoch [186/300], Val Loss: 0.841, Val Accuracy: 64.500%
Epoch [187/300], Train Loss: 0.800, Train Accuracy: 67.679%
Epoch [187/300], Val Loss: 0.879, Val Accuracy: 66.000%
Epoch [188/300], Train Loss: 0.796, Train Accuracy: 67.429%
Epoch [188/300], Val Loss: 0.850, Val Accuracy: 64.750%
Epoch [189/300], Train Loss: 0.798, Train Accuracy: 66.786%
Epoch [189/300], Val Loss: 0.914, Val Accuracy: 65.250%
Epoch [190/300], Train Loss: 0.782, Train Accuracy: 68.071%
Epoch [190/300], Val Loss: 0.843, Val Accuracy: 67.750%
Epoch [191/300], Train Loss: 0.784, Train Accuracy: 67.714%
Epoch [191/300], Val Loss: 0.845, Val Accuracy: 63.000%
Epoch [192/300], Train Loss: 0.791, Train Accuracy: 68.321%
Epoch [192/300], Val Loss: 0.841, Val Accuracy: 64.000%
Epoch [193/300], Train Loss: 0.790, Train Accuracy: 66.821%
Epoch [193/300], Val Loss: 0.832, Val Accuracy: 65.500%
Epoch [194/300], Train Loss: 0.780, Train Accuracy: 67.393%
Epoch [194/300], Val Loss: 0.912, Val Accuracy: 64.000%
Epoch [195/300], Train Loss: 0.802, Train Accuracy: 67.107%
Epoch [195/300], Val Loss: 0.829, Val Accuracy: 65.000%
Epoch [196/300], Train Loss: 0.792, Train Accuracy: 67.536%
Epoch [196/300], Val Loss: 0.824, Val Accuracy: 65.000%
Epoch [197/300], Train Loss: 0.786, Train Accuracy: 67.000%
Epoch [197/300], Val Loss: 0.844, Val Accuracy: 65.500%
Epoch [198/300], Train Loss: 0.806, Train Accuracy: 66.286%
Epoch [198/300], Val Loss: 0.814, Val Accuracy: 65.250%
Epoch [199/300], Train Loss: 0.789, Train Accuracy: 68.393%
Epoch [199/300], Val Loss: 0.827, Val Accuracy: 67.750%
Epoch [200/300], Train Loss: 0.757, Train Accuracy: 69.679%
Epoch [200/300], Val Loss: 0.812, Val Accuracy: 65.250%
Epoch [201/300], Train Loss: 0.797, Train Accuracy: 66.571%
Epoch [201/300], Val Loss: 0.840, Val Accuracy: 63.500%
Best Test Model Saved at Epoch 201 with Test Accuracy: 60.625%
Test Accuracy at Epoch 201: 60.625%
Epoch [202/300], Train Loss: 0.767, Train Accuracy: 67.214%
Epoch [202/300], Val Loss: 0.815, Val Accuracy: 65.750%
Epoch [203/300], Train Loss: 0.760, Train Accuracy: 68.786%
```

```
Epoch [203/300], Val Loss: 0.852, Val Accuracy: 62.500%
Epoch [204/300], Train Loss: 0.766, Train Accuracy: 69.036%
Epoch [204/300], Val Loss: 0.813, Val Accuracy: 63.750%
Epoch [205/300], Train Loss: 0.757, Train Accuracy: 69.286%
Epoch [205/300], Val Loss: 0.832, Val Accuracy: 63.250%
Epoch [206/300], Train Loss: 0.747, Train Accuracy: 69.036%
Epoch [206/300], Val Loss: 0.825, Val Accuracy: 67.250%
Best Test Model Saved at Epoch 206 with Test Accuracy: 62.375%
Test Accuracy at Epoch 206: 62.375%
Epoch [207/300], Train Loss: 0.758, Train Accuracy: 69.429%
Epoch [207/300], Val Loss: 0.805, Val Accuracy: 63.750%
Epoch [208/300], Train Loss: 0.753, Train Accuracy: 69.179%
Epoch [208/300], Val Loss: 0.811, Val Accuracy: 65.500%
Epoch [209/300], Train Loss: 0.755, Train Accuracy: 69.607%
Epoch [209/300], Val Loss: 0.864, Val Accuracy: 64.500%
Epoch [210/300], Train Loss: 0.775, Train Accuracy: 68.571%
Epoch [210/300], Val Loss: 0.813, Val Accuracy: 65.750%
Epoch [211/300], Train Loss: 0.758, Train Accuracy: 68.964%
Epoch [211/300], Val Loss: 0.824, Val Accuracy: 67.750%
Test Accuracy at Epoch 211: 61.500%
Best Val Model Saved at Epoch 212 with Val Accuracy: 68.750%
Epoch [212/300], Train Loss: 0.759, Train Accuracy: 69.071%
Epoch [212/300], Val Loss: 0.795, Val Accuracy: 68.750%
Epoch [213/300], Train Loss: 0.750, Train Accuracy: 68.857%
Epoch [213/300], Val Loss: 0.840, Val Accuracy: 65.750%
Epoch [214/300], Train Loss: 0.755, Train Accuracy: 69.500%
Epoch [214/300], Val Loss: 0.782, Val Accuracy: 67.500%
Epoch [215/300], Train Loss: 0.763, Train Accuracy: 68.607%
Epoch [215/300], Val Loss: 0.809, Val Accuracy: 66.000%
Epoch [216/300], Train Loss: 0.760, Train Accuracy: 69.643%
Epoch [216/300], Val Loss: 0.801, Val Accuracy: 67.750%
Best Test Model Saved at Epoch 216 with Test Accuracy: 62.750%
Test Accuracy at Epoch 216: 62.750%
Epoch [217/300], Train Loss: 0.760, Train Accuracy: 68.464%
Epoch [217/300], Val Loss: 0.811, Val Accuracy: 66.750%
Epoch [218/300], Train Loss: 0.741, Train Accuracy: 69.286%
Epoch [218/300], Val Loss: 0.790, Val Accuracy: 68.000%
Epoch [219/300], Train Loss: 0.735, Train Accuracy: 70.607%
Epoch [219/300], Val Loss: 0.900, Val Accuracy: 66.500%
Epoch [220/300], Train Loss: 0.738, Train Accuracy: 70.286%
Epoch [220/300], Val Loss: 0.790, Val Accuracy: 67.250%
Best Val Model Saved at Epoch 221 with Val Accuracy: 69.000%
Epoch [221/300], Train Loss: 0.758, Train Accuracy: 69.393%
Epoch [221/300], Val Loss: 0.816, Val Accuracy: 69.000%
Test Accuracy at Epoch 221: 62.000%
Epoch [222/300], Train Loss: 0.758, Train Accuracy: 68.607%
Epoch [222/300], Val Loss: 0.776, Val Accuracy: 68.000%
Epoch [223/300], Train Loss: 0.750, Train Accuracy: 68.786%
Epoch [223/300], Val Loss: 0.806, Val Accuracy: 65.250%
Epoch [224/300], Train Loss: 0.770, Train Accuracy: 68.857%
Epoch [224/300], Val Loss: 0.818, Val Accuracy: 66.750%
Epoch [225/300], Train Loss: 0.745, Train Accuracy: 69.536%
Epoch [225/300], Val Loss: 0.783, Val Accuracy: 66.750%
Epoch [226/300], Train Loss: 0.746, Train Accuracy: 71.179%
Epoch [226/300], Val Loss: 0.788, Val Accuracy: 68.500%
Best Test Model Saved at Epoch 226 with Test Accuracy: 63.125%
Test Accuracy at Epoch 226: 63.125%
Epoch [227/300], Train Loss: 0.747, Train Accuracy: 68.964%
Epoch [227/300], Val Loss: 0.787, Val Accuracy: 66.750%
Epoch [228/300], Train Loss: 0.725, Train Accuracy: 70.286%
Epoch [228/300], Val Loss: 0.829, Val Accuracy: 66.250%
Epoch [229/300], Train Loss: 0.759, Train Accuracy: 70.036%
Epoch [229/300], Val Loss: 0.801, Val Accuracy: 66.250%
Epoch [230/300], Train Loss: 0.746, Train Accuracy: 69.429%
Epoch [230/300], Val Loss: 0.771, Val Accuracy: 68.000%
Epoch [231/300], Train Loss: 0.743, Train Accuracy: 70.179%
Epoch [231/300], Val Loss: 0.813, Val Accuracy: 67.750%
Best Test Model Saved at Epoch 231 with Test Accuracy: 63.875%
Test Accuracy at Epoch 231: 63.875%
Epoch [232/300], Train Loss: 0.732, Train Accuracy: 69.964%
Epoch [232/300], Val Loss: 0.812, Val Accuracy: 65.000%
Best Val Model Saved at Epoch 233 with Val Accuracy: 70.250%
```

```
Epoch [233/300], Train Loss: 0.734, Train Accuracy: 69.107%
Epoch [233/300], Val Loss: 0.819, Val Accuracy: 70.250%
Epoch [234/300], Train Loss: 0.732, Train Accuracy: 70.071%
Epoch [234/300], Val Loss: 0.769, Val Accuracy: 68.000%
Epoch [235/300], Train Loss: 0.721, Train Accuracy: 70.929%
Epoch [235/300], Val Loss: 0.771, Val Accuracy: 67.000%
Epoch [236/300], Train Loss: 0.750, Train Accuracy: 69.714%
Epoch [236/300], Val Loss: 0.816, Val Accuracy: 66.000%
Test Accuracy at Epoch 236: 63.750%
Epoch [237/300], Train Loss: 0.742, Train Accuracy: 69.536%
Epoch [237/300], Val Loss: 0.802, Val Accuracy: 67.500%
Epoch [238/300], Train Loss: 0.716, Train Accuracy: 70.964%
Epoch [238/300], Val Loss: 0.802, Val Accuracy: 66.250%
Epoch [239/300], Train Loss: 0.737, Train Accuracy: 70.071%
Epoch [239/300], Val Loss: 0.815, Val Accuracy: 69.000%
Epoch [240/300], Train Loss: 0.731, Train Accuracy: 70.893%
Epoch [240/300], Val Loss: 0.817, Val Accuracy: 66.750%
Epoch [241/300], Train Loss: 0.750, Train Accuracy: 69.393%
Epoch [241/300], Val Loss: 0.855, Val Accuracy: 67.000%
Test Accuracy at Epoch 241: 61.750%
Epoch [242/300], Train Loss: 0.733, Train Accuracy: 70.179%
Epoch [242/300], Val Loss: 0.792, Val Accuracy: 69.500%
Epoch [243/300], Train Loss: 0.733, Train Accuracy: 70.607%
Epoch [243/300], Val Loss: 0.814, Val Accuracy: 67.750%
Epoch [244/300], Train Loss: 0.741, Train Accuracy: 69.607%
Epoch [244/300], Val Loss: 0.828, Val Accuracy: 64.750%
Epoch [245/300], Train Loss: 0.732, Train Accuracy: 69.893%
Epoch [245/300], Val Loss: 0.808, Val Accuracy: 66.500%
Epoch [246/300], Train Loss: 0.714, Train Accuracy: 71.643%
Epoch [246/300], Val Loss: 0.790, Val Accuracy: 69.000%
Test Accuracy at Epoch 246: 63.750%
Epoch [247/300], Train Loss: 0.703, Train Accuracy: 72.786%
Epoch [247/300], Val Loss: 0.816, Val Accuracy: 68.750%
Epoch [248/300], Train Loss: 0.737, Train Accuracy: 70.143%
Epoch [248/300], Val Loss: 0.836, Val Accuracy: 67.000%
Epoch [249/300], Train Loss: 0.721, Train Accuracy: 71.143%
Epoch [249/300], Val Loss: 0.840, Val Accuracy: 69.500%
Epoch [250/300], Train Loss: 0.725, Train Accuracy: 70.857%
Epoch [250/300], Val Loss: 0.805, Val Accuracy: 68.000%
Epoch [251/300], Train Loss: 0.711, Train Accuracy: 71.286%
Epoch [251/300], Val Loss: 0.805, Val Accuracy: 67.750%
Test Accuracy at Epoch 251: 62.375%
Epoch [252/300], Train Loss: 0.717, Train Accuracy: 71.500%
Epoch [252/300], Val Loss: 0.808, Val Accuracy: 68.750%
Epoch [253/300], Train Loss: 0.717, Train Accuracy: 71.679%
Epoch [253/300], Val Loss: 0.803, Val Accuracy: 66.750%
Epoch [254/300], Train Loss: 0.725, Train Accuracy: 71.107%
Epoch [254/300], Val Loss: 0.791, Val Accuracy: 67.000%
Epoch [255/300], Train Loss: 0.722, Train Accuracy: 70.893%
Epoch [255/300], Val Loss: 0.806, Val Accuracy: 67.000%
Epoch [256/300], Train Loss: 0.712, Train Accuracy: 70.786%
Epoch [256/300], Val Loss: 0.835, Val Accuracy: 68.750%
Test Accuracy at Epoch 256: 63.375%
Epoch [257/300], Train Loss: 0.713, Train Accuracy: 71.500%
Epoch [257/300], Val Loss: 0.769, Val Accuracy: 68.250%
Epoch [258/300], Train Loss: 0.687, Train Accuracy: 72.321%
Epoch [258/300], Val Loss: 0.908, Val Accuracy: 68.000%
Epoch [259/300], Train Loss: 0.708, Train Accuracy: 72.071%
Epoch [259/300], Val Loss: 0.764, Val Accuracy: 67.750%
Epoch [260/300], Train Loss: 0.728, Train Accuracy: 70.321%
Epoch [260/300], Val Loss: 0.837, Val Accuracy: 66.000%
Epoch [261/300], Train Loss: 0.697, Train Accuracy: 72.286%
Epoch [261/300], Val Loss: 0.750, Val Accuracy: 67.750%
Best Test Model Saved at Epoch 261 with Test Accuracy: 65.125%
Test Accuracy at Epoch 261: 65.125%
Epoch [262/300], Train Loss: 0.704, Train Accuracy: 71.500%
Epoch [262/300], Val Loss: 0.810, Val Accuracy: 67.250%
Epoch [263/300], Train Loss: 0.706, Train Accuracy: 71.679%
Epoch [263/300], Val Loss: 0.798, Val Accuracy: 69.250%
Epoch [264/300], Train Loss: 0.686, Train Accuracy: 72.821%
Epoch [264/300], Val Loss: 0.815, Val Accuracy: 67.750%
Epoch [265/300], Train Loss: 0.705, Train Accuracy: 71.893%
```

```
Epoch [265/300], Val Loss: 0.785, Val Accuracy: 69.250%
Epoch [266/300], Train Loss: 0.707, Train Accuracy: 70.607%
Epoch [266/300], Val Loss: 0.791, Val Accuracy: 68.500%
Test Accuracy at Epoch 266: 64.500%
Epoch [267/300], Train Loss: 0.711, Train Accuracy: 71.214%
Epoch [267/300], Val Loss: 0.783, Val Accuracy: 67.500%
Epoch [268/300], Train Loss: 0.714, Train Accuracy: 71.393%
Epoch [268/300], Val Loss: 0.814, Val Accuracy: 66.250%
Epoch [269/300], Train Loss: 0.693, Train Accuracy: 71.786%
Epoch [269/300], Val Loss: 0.780, Val Accuracy: 68.500%
Epoch [270/300], Train Loss: 0.725, Train Accuracy: 70.786%
Epoch [270/300], Val Loss: 0.774, Val Accuracy: 70.000%
Epoch [271/300], Train Loss: 0.709, Train Accuracy: 71.000%
Epoch [271/300], Val Loss: 0.850, Val Accuracy: 67.250%
Test Accuracy at Epoch 271: 63.375%
Epoch [272/300], Train Loss: 0.698, Train Accuracy: 71.179%
Epoch [272/300], Val Loss: 0.798, Val Accuracy: 66.250%
Epoch [273/300], Train Loss: 0.689, Train Accuracy: 71.750%
Epoch [273/300], Val Loss: 0.802, Val Accuracy: 67.500%
Best Val Model Saved at Epoch 274 with Val Accuracy: 72.250%
Epoch [274/300], Train Loss: 0.705, Train Accuracy: 71.607%
Epoch [274/300], Val Loss: 0.759, Val Accuracy: 72.250%
Epoch [275/300], Train Loss: 0.697, Train Accuracy: 71.714%
Epoch [275/300], Val Loss: 0.784, Val Accuracy: 68.250%
Epoch [276/300], Train Loss: 0.710, Train Accuracy: 72.214%
Epoch [276/300], Val Loss: 0.766, Val Accuracy: 70.750%
Test Accuracy at Epoch 276: 65.000%
Epoch [277/300], Train Loss: 0.719, Train Accuracy: 70.429%
Epoch [277/300], Val Loss: 0.812, Val Accuracy: 67.750%
Epoch [278/300], Train Loss: 0.684, Train Accuracy: 72.107%
Epoch [278/300], Val Loss: 0.810, Val Accuracy: 67.250%
Epoch [279/300], Train Loss: 0.691, Train Accuracy: 71.357%
Epoch [279/300], Val Loss: 0.798, Val Accuracy: 66.750%
Epoch [280/300], Train Loss: 0.695, Train Accuracy: 71.821%
Epoch [280/300], Val Loss: 0.794, Val Accuracy: 70.250%
Epoch [281/300], Train Loss: 0.695, Train Accuracy: 71.821%
Epoch [281/300], Val Loss: 0.768, Val Accuracy: 69.000%
Test Accuracy at Epoch 281: 64.500%
Epoch [282/300], Train Loss: 0.689, Train Accuracy: 73.143%
Epoch [282/300], Val Loss: 0.791, Val Accuracy: 67.750%
Epoch [283/300], Train Loss: 0.682, Train Accuracy: 72.464%
Epoch [283/300], Val Loss: 0.785, Val Accuracy: 70.500%
Epoch [284/300], Train Loss: 0.704, Train Accuracy: 71.393%
Epoch [284/300], Val Loss: 0.801, Val Accuracy: 68.500%
Epoch [285/300], Train Loss: 0.692, Train Accuracy: 71.536%
Epoch [285/300], Val Loss: 0.757, Val Accuracy: 68.750%
Epoch [286/300], Train Loss: 0.693, Train Accuracy: 72.071%
Epoch [286/300], Val Loss: 0.788, Val Accuracy: 70.250%
Test Accuracy at Epoch 286: 64.625%
Epoch [287/300], Train Loss: 0.685, Train Accuracy: 72.893%
Epoch [287/300], Val Loss: 0.780, Val Accuracy: 67.000%
Epoch [288/300], Train Loss: 0.717, Train Accuracy: 71.893%
Epoch [288/300], Val Loss: 0.800, Val Accuracy: 69.500%
Epoch [289/300], Train Loss: 0.711, Train Accuracy: 72.393%
Epoch [289/300], Val Loss: 0.789, Val Accuracy: 68.500%
Epoch [290/300], Train Loss: 0.684, Train Accuracy: 72.929%
Epoch [290/300], Val Loss: 0.779, Val Accuracy: 68.500%
Epoch [291/300], Train Loss: 0.693, Train Accuracy: 71.429%
Epoch [291/300], Val Loss: 0.795, Val Accuracy: 71.000%
Best Test Model Saved at Epoch 291 with Test Accuracy: 65.875%
Test Accuracy at Epoch 291: 65.875%
Epoch [292/300], Train Loss: 0.693, Train Accuracy: 71.714%
Epoch [292/300], Val Loss: 0.787, Val Accuracy: 68.250%
Epoch [293/300], Train Loss: 0.699, Train Accuracy: 72.250%
Epoch [293/300], Val Loss: 0.767, Val Accuracy: 69.250%
Epoch [294/300], Train Loss: 0.679, Train Accuracy: 72.750%
Epoch [294/300], Val Loss: 0.781, Val Accuracy: 69.000%
Epoch [295/300], Train Loss: 0.708, Train Accuracy: 71.214%
Epoch [295/300], Val Loss: 0.783, Val Accuracy: 68.000%
Epoch [296/300], Train Loss: 0.693, Train Accuracy: 72.750%
Epoch [296/300], Val Loss: 0.769, Val Accuracy: 68.000%
Test Accuracy at Epoch 296: 65.250%
```

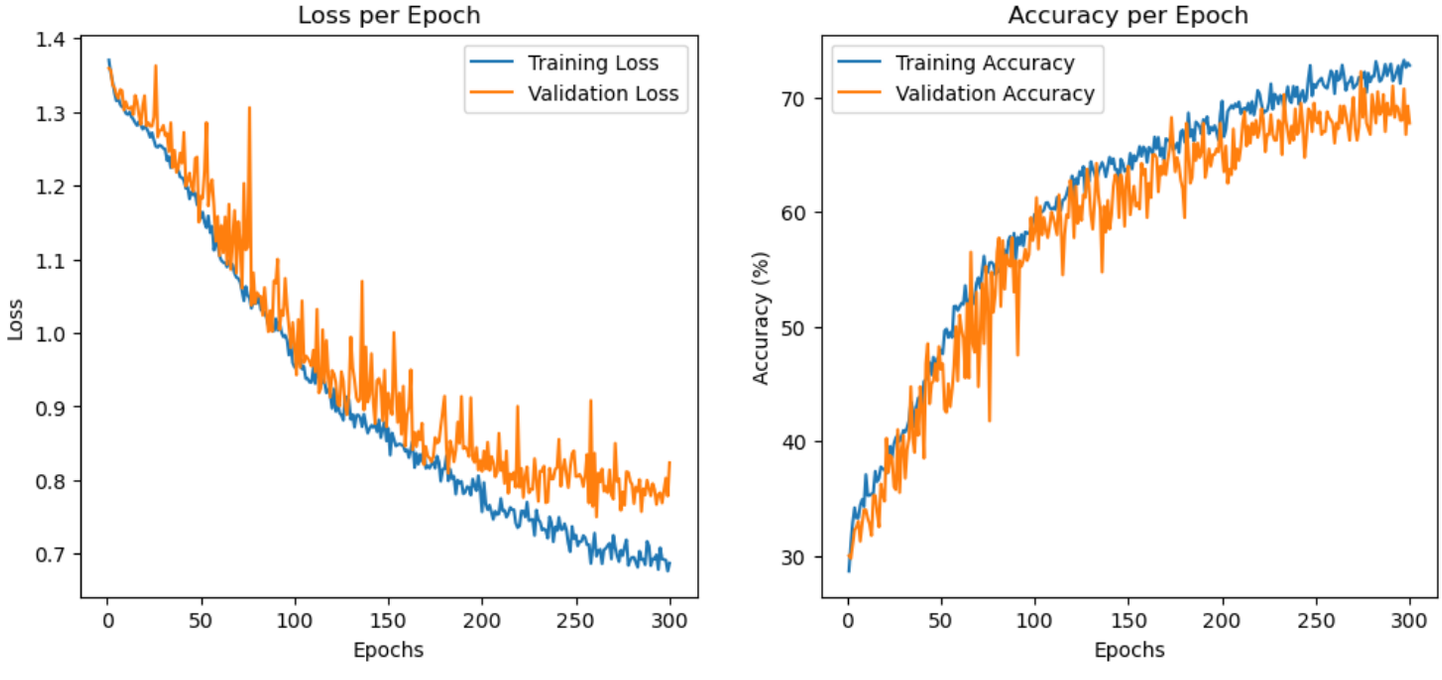
```
Epoch [297/300], Train Loss: 0.692, Train Accuracy: 73.250%
Epoch [297/300], Val Loss: 0.786, Val Accuracy: 70.750%
Epoch [298/300], Train Loss: 0.691, Train Accuracy: 72.607%
Epoch [298/300], Val Loss: 0.803, Val Accuracy: 66.750%
Epoch [299/300], Train Loss: 0.676, Train Accuracy: 73.000%
Epoch [299/300], Val Loss: 0.778, Val Accuracy: 69.250%
Epoch [300/300], Train Loss: 0.687, Train Accuracy: 72.786%
Epoch [300/300], Val Loss: 0.824, Val Accuracy: 67.750%
```

In [ ]:

```
# Saving model
i = 1
base_filename = project_path + 'ML2/HW1/models/cnn'
filename = f'{base_filename}.pkl'

while os.path.exists(filename):
    filename = f"{base_filename}_{i}.pkl"
    i += 1

torch.save(cnn.state_dict(), filename)
```

In [ ]:

```
epochs = range(1, num_epochs+1)
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label='Training Loss')
plt.plot(epochs, val_losses, label='Validation Loss')
plt.title('Loss per Epoch')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracies, label='Training Accuracy')
plt.plot(epochs, val_accuracies, label='Validation Accuracy')
plt.title('Accuracy per Epoch')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()

plt.show()
```



3)

In [11]:

```
# Loading the best validation model
# This is the one we've taken at the end

state_dict = torch.load(r'C:\Users\yumif\Desktop\projects\machine_learning_hw\ML2\HW1_FINAL\HW1_Q3_337604821_340915156.pkl')
```

In [12]:

```
cnn.load_state_dict(state_dict)
```

Out[12]:

```
<All keys matched successfully>
```

In [13]:

```
cnn.eval()

test_loss = 0.0
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

        outputs = cnn(images)

        loss = criterion(outputs, labels)
        test_loss += loss.item()  # Accumulate the test loss

        predicted = torch.argmax(outputs, dim=1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

average_test_loss = test_loss / len(test_loader)
accuracy = 100 * correct / total

print(f"Test Loss: {average_test_loss:.3f}")
print(f"Test Accuracy: {accuracy:.3f}%")
```

```
Test Loss: 0.858
Test Accuracy: 67.875%
```

4)

Through extensive trial and error, we've designed a cnn model that progressively increases the number of channels while incorporating dropout and activation functions such as Leaky ReLU and SiLU to enhance generalization and performance. The extracted features are processed through an adaptive average pooling layer, followed by fully connected layers for final classification.

After training the model for 300 epochs, we took the best model based on validation set. We've achieved a test accuracy of 67.875% with only 100,116 parameters.

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 `hw1_data/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 `hw1_data/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 their response (their output) for the image from section 5. Explain what do you see.**

In [ ]:

```
import torch
import torchvision.models as models
import torch.nn as nn
from PIL import Image
```

1.

In [ ]:

```
vgg16 = models.vgg16(pretrained=True, progress=True)
vgg16.eval()

if torch.cuda.is_available():
    vgg16 = vgg16.cuda()
```

```
C:\Users\yumif\anaconda3\envs\genenv\Lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in th
e future, please use 'weights' instead.
  warnings.warn(
C:\Users\yumif\anaconda3\envs\genenv\Lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated si
nce 0.13 and may be removed in the future. The current behavior is equivalent to passing `
weights=VGG16_Weights.IMAGENET1K_V1`. You can also use `weights=VGG16_Weights.DEFAULT` to
get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to C:\Users\yumif/.
cache\torch\hub\checkpoints\vgg16-397923af.pth
100%|██████████| 528M/528M [00:25<00:00, 21.7MB/s]
```

2.

In [ ]:

```
birds_path = project_path + '/ML2/HW1/hw1_data/birds/'

image_0 = Image.open(birds_path + 'bird_0.jpg').convert('RGB')
image_1 = Image.open(birds_path + 'bird_1.jpg').convert('RGB')

images = [image_0, image_1]
```

In [ ]:

```
fig, axes = plt.subplots(1, len(images), figsize=(12, 4))
```

```
for i, image in enumerate(images):
    axes[i].imshow(image)
    axes[i].axis('off')

plt.show()
```

3.

We've resized images to 224x224, and then normalized them accordingly

In [ ]:

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

transformed_images = torch.stack([transform(img) for img in images])
```

4.

In [ ]:

```
with torch.no_grad():
    outputs = vgg16(transformed_images)
```

The outputs are raw scores for each class (1000 classes total) for each of the two images

In [ ]:

```
print(outputs.shape)
print(outputs)
```

```
torch.Size([2, 1000])
tensor([[ 1.5175,  4.2852,  2.7133,  ...,  0.5375,  4.9361, -1.3509],
        [ 0.8642,  5.0238, -0.9463,  ...,  0.6321,  4.5410, -1.8296]])
```

We can compute probabilities with softmax and get top k indices of classes.

Overall, it seems like vgg16 predicted similar classes for the images (they are both birds, after all)

In [ ]:

```
probs = nn.functional.softmax(outputs, dim=1)
_, top5_indices = torch.topk(probs, 5)
```

In [ ]:

```
print(top5_indices)
```

```
tensor([[ 94,  95,  92,  84, 128],
        [ 90,  88,  11,  92,  95]])
```

5.

In [ ]:

```
dog_path = project_path + 'ML2/HW1/hw1_data/dogs/dog_7.jpg'
dog_image = Image.open(dog_path).convert('RGB')
```

In [ ]:

```
plt.imshow(dog_image)
plt.axis('off')
plt.show()
```

In [ ]:

```
dog_transformed = torch.stack([transform(dog_image)])

with torch.no_grad():
    output = vgg16(dog_transformed)
```

Again, the outputs are raw scores for each class

In [ ]:

```
print(output.shape)
print(output)
```

```
torch.Size([1, 1000])
tensor([[ 4.0035e-01, -4.1130e+00,  4.9246e-02, -1.8518e+00, -2.9120e+00,
         -8.6880e-01, -3.5274e+00, -1.2696e+00,  3.6082e-01, -2.4375e+00,
          2.5472e-01, -2.6453e+00, -3.2328e+00, -1.7111e+00, -2.6212e+00,
         -1.5648e+00, -2.2126e+00, -2.0455e+00, -2.3517e+00, -3.4826e+00,
         -2.5599e+00, -7.5499e-01, -3.0126e+00, -2.5906e+00, -3.2965e+00,
          1.1103e+00,  5.4938e-02, -1.1802e+00, -5.2863e-03, -3.5758e+00,
         -1.3560e+00, -2.6722e+00, -2.6543e-01, -8.0850e-01, -3.1780e-01,
         -1.2984e-01,  3.3015e-01, -3.4683e-01, -3.7454e+00, -3.1020e+00,
         -3.7873e+00, -1.7079e+00, -3.0208e+00, -1.7253e+00, -3.7442e+00,
         -2.6757e+00, -4.3063e+00, -2.0166e+00, -4.0280e+00, -9.0577e-01,
         -1.4152e+00,  7.7519e-01, -8.5202e-01, -9.1757e-01, -1.2209e+00,
         -3.6901e+00, -2.2970e+00, -1.0446e+00, -2.8337e+00, -9.3002e-01,
         -6.7811e-01, -5.8468e-01,  2.2858e+00,  7.8846e-02, -1.4821e+00,
          5.8840e-01,  9.6343e-01, -2.3578e+00, -1.4007e-01, -2.5804e+00,
         -1.6220e+00, -1.4486e+00,  2.2186e-01, -6.5374e-02, -1.4226e+00,
         -1.6618e+00, -4.3914e+00, -2.8011e+00,  3.3817e-01, -2.5319e+00,
```

```
-1.9118e+00, -4.6885e+00, -1.7180e+00, -5.6302e-01, -3.7228e+00,
-2.8670e+00, -1.8346e+00, -2.8034e+00, -3.6170e+00, -3.5858e+00,
-4.0857e+00, -4.4884e+00, -3.6489e+00, -2.2825e+00, -4.0065e+00,
-1.7943e+00, -3.7433e+00, -6.7423e-02, -3.5723e-01,  1.7904e-01,
 1.5532e-01, -1.5015e-01, -1.8795e+00, -2.3583e+00,  3.7798e+00,
-1.8526e+00,  4.4862e+00, -3.7276e+00, -4.4095e+00, -2.8296e+00,
-1.6279e-01, -1.3038e+00, -1.5364e+00, -1.2571e+00, -4.8703e-01,
-1.3976e+00,  8.4960e-01, -3.2963e+00, -9.5316e-01, -1.9447e+00,
-1.5827e+00,  3.7658e-01,  2.1389e-01, -2.2239e+00, -2.3751e+00,
-2.1485e+00, -1.2715e+00, -1.9165e+00,  1.5040e-01, -4.0364e+00,
-3.2938e+00, -3.6249e+00, -3.6635e+00, -2.4637e+00, -2.3972e+00,
-1.7109e+00, -3.0432e-01, -2.1568e+00, -4.0359e+00, -1.1134e+00,
-2.0083e+00, -2.6611e+00, -5.7677e-01,  1.4284e-01, -2.7473e+00,
-1.7001e+00, -3.0347e+00, -7.3465e-02, -8.8322e-01, -1.8968e+00,
-1.6734e+00,  4.8779e+00,  9.7723e-01, -2.5296e+00,  1.8774e-01,
-2.2806e+00, -1.4188e+00,  1.4705e+00,  5.8019e+00,  5.6569e+00,
 2.3231e+00,  2.2158e+00,  2.4229e+00,  6.3124e+00,  5.4347e+00,
 6.6222e+00,  3.8971e+00,  6.0085e+00,  5.7340e+00,  3.5970e+00,
 6.1864e+00, -1.3210e-01,  2.1845e+00,  6.3615e+00,  1.0726e+01,
 7.3447e+00,  2.3334e+00,  3.9596e+00,  4.1152e-01,  4.2976e+00,
 5.8035e+00,  2.0358e+00,  7.2489e+00,  2.3752e+00,  3.9558e+00,
 3.6398e+00,  6.8247e+00,  2.0487e+00,  6.8884e+00,  6.6339e+00,
 1.9319e+00,  6.6314e+00,  4.2832e+00,  7.8902e+00,  3.5110e+00,
 2.6487e+00,  2.4155e+00,  4.0322e+00,  3.9593e+00,  4.9091e+00,
 4.5505e+00,  2.1370e+00,  2.6200e+00,  1.0589e+00,  5.1037e-01,
 3.6212e+00,  2.5326e+00,  3.4584e+00,  4.3279e+00,  5.0677e+00,
 2.7094e+00,  1.6208e+00,  1.8490e+00,  9.5408e-01,  4.2478e+00,
 2.0521e+00,  3.3071e+00,  5.0053e-01,  1.5709e+00,  2.2267e-01,
 3.8064e+00,  4.5105e-01,  4.5593e+00,  5.8314e+00,  6.7489e+00,
 1.1915e+01,  6.5406e+00,  1.2559e+01,  2.4651e+00,  2.5881e+00,
 4.3744e+00,  7.2398e+00,  5.7350e+00,  4.4627e+00,  9.0151e+00,
 1.5998e+01,  8.3402e+00,  3.8837e+00,  4.7786e+00,  2.5715e+00,
 8.5642e+00,  5.6483e+00,  1.9106e+00,  3.9509e+00,  7.1980e+00,
 2.0212e+00,  5.0910e+00,  4.7918e+00,  8.8116e+00,  8.7733e+00,
 7.8477e+00,  1.1634e+00,  2.1229e+00,  6.5071e+00, -8.9046e-02,
 6.2886e+00,  3.8164e+00,  3.1420e+00,  2.3940e+00, -5.3425e-01,
 5.1023e+00,  5.0002e+00,  3.3225e+00,  8.4842e+00,  9.2504e+00,
-4.4559e-01, -1.3457e-01,  1.1980e+00,  4.0660e+00,  9.2529e+00,
 7.4779e+00,  8.4685e+00,  6.6892e+00,  1.0270e+01,  8.7478e+00,
 8.5004e+00,  2.7832e+00,  3.7950e+00,  4.7515e+00,  1.4026e+00,
 4.7989e+00,  2.7517e+00,  2.1687e+00, -1.6342e+00,  4.4193e-01,
 2.5099e+00,  1.3769e+00,  8.8929e-01,  8.9519e-01,  6.9108e-01,
 2.1233e+00,  9.4317e-01,  2.2837e+00,  1.4053e+00,  3.8693e+00,
 5.9871e+00,  1.9395e+00,  3.4340e+00, -1.7629e+00, -1.4787e+00,
-1.3524e+00, -2.7767e+00, -1.5645e+00, -6.4902e-01, -1.4707e+00,
-3.8231e+00, -2.3464e+00, -3.2450e+00, -3.8379e+00, -1.6469e+00,
-2.3761e+00, -2.8086e+00, -4.0485e+00, -3.2676e+00, -3.0747e+00,
-4.6908e+00, -3.8950e+00, -1.3955e+00, -2.4002e+00, -3.8613e+00,
-4.4692e+00, -2.3474e+00, -2.5539e+00, -1.8130e+00, -2.7627e+00,
-2.7869e+00, -3.1488e+00, -3.9841e-01, -3.7997e+00,  1.6719e-02,
 5.7555e-01,  2.2023e+00, -7.8203e-01, -1.9235e+00, -3.6272e+00,
 1.1283e+00, -7.7299e-01,  8.8926e-01, -1.4794e+00,  2.2816e+00,
-1.2618e+00,  4.0113e+00,  2.7094e+00, -1.3132e+00, -4.3061e-01,
-3.5157e-01, -5.3153e-01, -1.4301e+00, -6.9638e-01, -9.2200e-01,
 6.8028e-01, -2.6697e-01, -2.1289e-01,  3.2388e-01, -1.1933e+00,
-2.2466e-01,  1.1593e+00, -5.8547e-01,  1.5306e+00,  1.5291e+00,
-8.0370e-01,  2.5370e-01,  3.5318e-01, -9.5838e-01,  5.8557e-01,
-1.7546e+00, -1.8199e+00,  8.5939e-01,  2.8827e-01, -1.4276e-02,
-3.8181e-01,  4.5005e-01,  7.5550e-01,  1.2795e-01, -4.1022e+00,
-2.9398e+00, -2.3377e+00, -1.9494e+00,  1.8413e+00,  3.8057e-02,
-1.3371e+00, -9.6681e-01, -4.6528e-01, -1.3830e+00,  9.1263e-01,
-1.2933e+00, -2.6781e+00, -7.8495e-01, -7.9480e-02, -1.1779e-02,
-1.1669e+00,  3.0301e-01, -1.2092e+00, -3.8986e+00,  2.1109e+00,
-1.2557e+00, -4.2290e+00, -4.1293e+00, -1.6984e+00, -1.5915e+00,
 4.0057e-01, -5.0205e-01, -1.8374e+00, -3.8735e+00, -1.9707e+00,
-2.1326e+00, -2.5205e+00,  9.6902e-01,  1.0031e+00, -9.3364e-03,
-5.2190e-01,  4.9183e-01,  9.1549e-01,  2.7808e+00, -2.0517e-01,
-1.1192e+00,  3.2605e-01, -1.1224e+00, -1.0296e+00, -1.1591e+00,
-9.6952e-01, -8.8493e-02,  3.6420e-01, -1.4791e+00, -7.5062e-01,
-1.1011e+00,  1.5622e+00,  8.5026e-01,  3.2690e+00,  1.4070e+00,
 2.3889e+00, -9.0324e-01, -7.1599e-01, -1.0567e+00,  9.9372e-01,
-1.7339e+00, -4.8315e-01, -1.2498e+00, -1.3723e+00,  2.2911e+00,
```

```
 5.2723e-01, -1.9945e+00, -8.4928e-01, -1.4902e+00,  8.6772e-01,
-3.3229e-01, -6.9436e-01, -3.9786e-01, -1.5305e+00, -2.0735e+00,
 1.3731e+00,  2.5760e+00, -2.1731e+00, -8.1672e-01,  3.0831e-01,
 6.4184e-01,  2.4388e+00,  5.2668e-01, -9.7398e-01, -1.5362e+00,
-1.5362e+00,  1.0610e+00,  2.8047e-02,  1.4944e+00,  6.7187e-01,
 9.7895e+00, -1.9608e+00, -5.2804e-01,  1.1224e+00, -1.1738e+00,
-1.0569e+00,  4.8241e-01,  3.3056e+00, -6.6232e-01, -2.1217e+00,
-7.1735e-01, -2.2183e+00,  1.1723e+00,  8.8230e-01,  2.0458e+00,
 2.5074e+00, -8.8375e-01,  5.9225e-01, -5.6107e-01, -4.0969e-01,
-1.3250e+00, -2.2915e+00,  5.0395e-01,  2.0893e+00,  5.9577e-01,
-2.2961e-01,  1.7565e+00, -1.6191e+00, -5.0025e-01, -1.0551e-01,
-2.6214e+00, -6.7066e-01, -2.0163e+00, -2.6349e+00,  1.1818e+00,
-2.5052e+00, -7.3593e-01,  3.2266e+00, -1.3879e+00, -1.1085e+00,
-1.2920e+00, -6.6173e-01,  1.4421e+00, -6.3769e-01, -6.6071e-01,
-1.7170e-01, -1.6877e+00, -1.8121e+00,  1.6795e+00,  2.9687e+00,
 1.3191e+00, -8.6123e-01,  5.7296e-01, -2.2990e-01,  1.4225e+00,
-2.3567e+00, -1.4746e+00,  3.8239e+00,  4.4285e-01,  2.0499e+00,
-3.3588e+00, -2.7684e+00,  2.8722e-02, -5.1242e-01, -1.4942e+00,
 1.0064e+00,  1.5184e+00, -1.5297e+00, -3.1826e+00, -7.8114e-01,
-3.2414e-01, -6.8157e-01,  6.3039e+00, -2.5463e+00,  2.2543e+00,
-5.3163e-01, -1.6364e-01,  4.2944e-01,  2.4282e+00,  5.7211e-01,
-7.1587e-01, -7.1585e-01, -5.4046e-01, -6.2003e-01, -1.4564e+00,
-1.2601e+00, -1.5519e+00, -5.1525e-01,  1.5368e+00, -1.3587e+00,
-4.6566e-01, -4.5247e-01,  7.8744e-02, -1.1284e+00,  1.0266e-01,
 6.6033e-01, -6.0625e-03, -2.1979e+00, -2.6523e+00, -2.6969e+00,
-1.1205e+00,  8.6892e-01,  1.6817e+00, -1.8305e-01, -7.3337e-01,
-1.8740e+00, -1.3102e-01, -2.9910e+00,  6.4360e-01,  1.4932e+00,
 2.6771e-01, -7.2327e-01,  4.8062e-01, -7.9388e-01, -3.6058e+00,
-1.2942e+00, -2.5380e+00, -1.9033e+00,  2.5467e-01, -2.3413e+00,
-1.6171e+00, -1.3122e-01,  7.4704e-01, -3.2700e-01, -1.5167e+00,
 2.3212e+00, -9.7578e-01, -1.3774e+00, -7.2970e-01, -1.5532e+00,
 3.2273e-01,  3.8234e+00,  1.1013e-01, -2.6655e-01,  6.5046e-01,
 6.5748e-01, -5.0825e-01,  8.2120e-01,  4.4487e-01, -2.4241e+00,
-1.6327e+00, -3.1751e-01, -2.8064e-01,  2.6867e-01,  9.3150e-01,
 1.5499e+00, -2.0017e-01, -7.1968e-01, -1.0959e+00, -1.4270e+00,
 2.2669e+00, -2.7970e+00, -1.4790e+00, -1.1695e+00, -1.5102e+00,
 6.1880e-02,  3.1714e+00, -1.2976e+00, -2.6629e-01, -1.4983e+00,
-6.7314e-01, -4.8479e-01, -1.4719e-01, -3.5686e+00, -1.2298e+00,
 1.2932e+00, -7.1311e-01, -6.7403e-01, -7.2458e-01, -8.3818e-01,
 2.9855e-01, -9.2316e-03, -4.4067e-01,  1.2393e+00,  7.9860e-01,
 1.3210e+00, -2.5155e-01, -9.2063e-01, -1.2121e+00, -1.9037e+00,
-1.2682e+00,  4.0072e+00, -2.5084e+00, -2.3433e+00,  6.7093e-02,
-2.0083e+00,  1.1606e-01,  6.2965e-01, -8.4040e-01,  1.1700e+00,
-1.5486e+00,  1.7789e+00, -8.6683e-01, -1.4534e+00, -1.3901e+00,
 1.7768e-01, -2.4582e+00, -1.1892e+00, -7.6731e-01, -8.8767e-01,
 1.7585e+00, -4.4328e-01,  9.8947e-01, -2.2806e+00, -4.8222e+00,
 1.5245e+00,  9.6685e-01, -4.9907e-01, -1.2119e+00, -3.6242e-01,
 9.4137e-01,  9.1879e+00, -1.2496e+00,  2.0765e+00, -2.2874e+00,
 6.9741e-01, -6.6980e-01, -6.3125e-01,  5.3885e-01,  4.8218e-01,
-2.0449e+00, -1.4282e+00, -2.4100e+00,  6.2209e-01, -4.4338e-01,
-1.8885e+00,  7.8002e-01,  4.1581e-01,  3.3297e+00, -8.7304e-01,
 1.0745e+00, -2.3213e+00, -9.6878e-02, -1.1734e+00,  3.5548e-01,
 1.8865e+00, -1.9892e-02,  9.3203e-01, -4.5964e-01, -6.9579e-01,
-2.0082e-02, -1.4757e+00,  1.2348e+00, -2.9547e-02, -1.9872e+00,
-2.5308e+00, -4.2135e+00, -1.9686e+00, -1.0052e+00, -7.4569e-01,
 6.6476e-01, -8.6486e-01, -2.7374e-01, -1.8643e+00, -1.9154e+00,
-2.2659e+00, -1.3850e+00,  8.5823e-01, -5.4919e-01, -2.4588e+00,
-1.2200e+00,  1.4619e+00, -6.4864e-01,  1.5011e+00, -1.6672e+00,
-3.0899e-01,  3.1645e+00, -1.2796e+00,  1.7659e+00,  3.4276e-01,
-2.3527e+00, -8.3046e-01, -1.6875e+00, -1.5965e+00, -1.7752e+00,
 5.7621e-01,  1.1225e+00,  1.1708e+00,  1.6648e-01,  4.6570e-01,
-3.2884e+00,  2.2546e+00,  6.5139e-01, -1.8112e+00, -3.4190e+00,
 1.8550e+00, -7.7726e-01,  6.8618e-02,  1.5761e-01, -3.8720e-01,
-2.9089e+00,  4.6391e-02,  1.3516e-01,  1.3836e+00, -2.2687e+00,
-1.6466e+00,  3.0453e+00, -1.1677e+00,  4.7393e-01,  7.6421e-01,
-7.3481e-01, -2.0121e-01, -9.9402e-01,  4.1223e+00, -2.9876e-01,
 9.6368e-01, -1.2036e+00, -1.6921e+00, -1.5162e+00, -4.1063e-01,
 1.8517e-01, -9.2853e-01,  2.1458e-01, -8.7278e-01,  8.4206e-01,
-3.4392e+00,  1.9077e+00,  4.3422e-01, -4.3745e-01,  7.4266e-01,
 4.6953e-01, -2.3242e+00,  2.6203e+00,  1.0410e-01,  1.7162e+00,
 1.2758e+00, -3.9022e-01,  2.3084e+00,  1.1644e+00, -1.4113e+00,
 3.7589e+00, -6.2206e-01,  1.4026e+00, -1.4969e+00,  2.5885e+00,
```

```
 1.1685e+00, -1.0309e+00,  9.3325e-01,  1.4328e-01, -5.4298e-01,
 8.4379e+00, -1.6011e+00,  1.3816e+00,  1.2945e+00, -2.7408e+00,
-3.2031e-01, -1.6603e-01, -6.3910e-01,  1.3389e+00, -9.1032e-03,
-9.2166e-01, -2.7651e+00, -5.5160e-01, -1.1183e+00, -3.5783e+00,
-1.7818e+00, -8.8050e-01,  7.4880e-02, -5.7273e-01,  5.8511e-01,
 1.6944e+00,  9.7646e-01, -8.8063e-02, -2.0748e+00, -8.4993e-01,
 2.7157e+00, -2.6219e-01, -2.0245e+00, -2.4508e+00, -1.1374e-02,
-6.4763e-01,  5.9676e-01,  2.9170e+00,  1.0820e+00, -1.4410e+00,
-1.7636e-01,  1.1041e+00, -6.4023e-01, -1.1443e+00,  4.3131e-01,
-4.0222e+00, -1.2430e+00, -1.4711e+00, -3.0659e-01, -3.0701e+00,
-2.8674e+00,  1.6198e+00,  6.3498e+00, -1.7694e+00, -2.1435e+00,
-2.7234e+00, -9.7957e-01, -1.2256e+00, -8.7051e-01, -1.6692e+00,
 9.5134e-01,  5.6505e-01,  8.3395e-01,  9.1229e-01,  6.3528e-01,
-3.0068e-01, -6.4403e-01, -1.0794e+00, -1.1584e-01, -5.4221e-01,
-8.4645e-01, -1.4205e+00,  2.6445e-01, -1.6747e+00, -1.2704e-01,
 3.3672e-01,  3.5720e-01, -5.3206e-01, -1.3799e+00, -1.2570e-01,
 8.0503e-01, -2.9499e+00,  2.5546e-01, -1.8499e+00, -2.3435e+00,
 6.8368e-01,  2.7643e-02, -4.9652e-01, -9.6876e-01, -2.3406e+00,
 1.4331e+00,  7.7230e-01, -7.4475e-01, -2.2062e+00, -2.5557e+00,
-1.6322e+00, -2.1549e+00, -1.5735e+00, -2.1774e-01, -2.9364e-01,
-2.5219e+00, -2.4100e+00,  1.2446e+00, -3.0298e+00,  1.5721e+00,
-7.4346e-01,  2.4090e+00, -7.9466e-01, -1.9242e+00, -4.5362e-01,
-6.2017e-01,  1.0005e-01,  2.0310e+00, -2.1014e+00, -5.5502e-01,
-2.0964e+00,  6.7112e-01,  9.2203e-01, -4.2480e-01,  1.5174e+00,
-3.0816e-01,  9.8122e-01, -2.1736e+00, -1.0164e+00, -7.7890e-01,
-2.0957e+00,  1.6536e-02, -3.6177e+00, -2.8744e+00,  1.9016e+00,
 8.9026e-01, -6.2674e-01,  1.2634e-01, -2.0656e+00,  2.4765e-01,
-1.9339e+00, -1.1935e+00,  3.0803e-01, -2.2049e+00,  1.1407e+00,
-4.9972e-03, -9.6721e-02, -1.8153e+00, -4.4898e-01, -2.1880e+00,
-1.8105e+00, -1.8344e+00, -2.8437e+00, -1.3012e+00, -1.1333e+00,
-1.7899e+00,  4.0078e-01,  2.0709e-02, -8.4288e-01, -1.9857e+00,
-9.6393e-01,  7.2803e-01, -1.8339e+00, -2.4150e+00, -9.3527e-01,
-7.6602e-01, -3.7571e-01, -2.6334e+00, -1.1902e+00, -1.9095e+00,
-1.6741e+00, -1.6333e+00, -3.6182e+00, -2.2704e+00, -1.1094e+00,
 1.2447e+00, -1.1647e+00, -1.0970e+00, -1.5660e+00, -8.4011e-01,
 5.1283e-01, -1.4230e+00,  1.8498e+00, -9.1538e-01, -2.8664e-01,
-3.3783e+00,  1.1259e+00, -2.7461e+00, -2.4600e+00, -1.0171e+00,
-3.9045e+00, -2.3466e+00,  1.8179e+00, -1.1703e+00, -1.3252e+00,
-1.5109e+00, -1.8262e+00, -1.2398e+00, -3.5378e-01, -4.4860e-02,
 2.5351e-01, -7.8115e-01, -2.6205e+00,  1.9900e+00, -3.8988e-01]])
```

Again, we can compute probabilities with softmax and get top k indices of classes.

This time model predicted completely different top 5 classes compared to the bird images.

In [ ]:

```
probs = nn.functional.softmax(output, dim=1)
_, top5_indices = torch.topk(probs, 5)
```

In [ ]:

```
print(top5_indices)
```

```
tensor([[235, 227, 225, 174, 273]])
```

6.

In [ ]:

```
with torch.no_grad():
    feature_maps = vgg16.features[0](dog_transformed)
```

In [ ]:

```
feature_maps.shape
```

Out[ ]:

```
torch.Size([1, 64, 224, 224])
```

In [ ]:

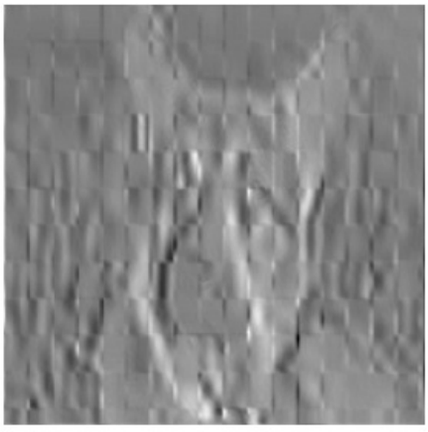
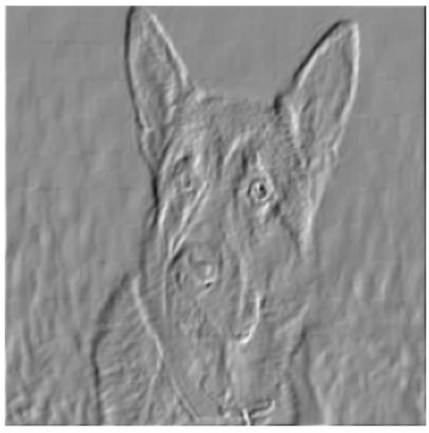
```
feature_maps = feature_maps[0, :3, :, :]
```

As we can see, each filter seems to accent on different details of the image

- **The first one seems to show rough and distinct edges**
- **The second shows slightly more complex features, like contours and textures**
- **The third filter shows even more finer details of dogs' silhouette and form**

In [ ]:

```
fig, axes = plt.subplots(1, len(feature_maps), figsize=(12, 4))

for i, map in enumerate(feature_maps):
    axes[i].imshow(map, cmap='gray')
    axes[i].axis('off')

plt.show()
```


In [ ]:

```

```