Machine Learning

Assignment 2 – Lab Report

I. PROBLEM 1

In this section, we analyze the convolution kernel and loss function employed in the framework employing mathematical equations.

A. Convolutional kernel

The convolutional kernel, also known as a filter, is a small matrix of weights that slides over the input data (such as an image) to perform a convolution operation.

The convolution kernel is an important part of the neural network. It is responsible for feature extraction and dimension raising (or dimension reducing). Two 2x2 convolution kernels with a stride of 1 are used in the framework. As for the formula of the convolution operation, the general form is as follows (assuming we have a 2-dimensional input y and a 2-dimensional convolution kernel ω):

$$X_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{i+a,j+b}$$

Here i and j are the spatial locations of the output feature map, a and b are the spatial locations of the convolution kernel, m is the size of the convolution kernel, and ωab is the weight of the convolution kernel at the position a,b. $y(i+a)(j+b)$ is the value of the input feature map at position $i+a,j+b$. This formula is done by applying a convolutional kernel to each position (i,j) and summing the results to get the value of the output feature map at that position. This is the basic operation in convolutional neural networks for extracting local features from the input feature map.

The core concept of the convolution operation involves the accumulation of weighted pixels within a region that matches the size of the input image's convolution kernel. The detailed operational process is depicted in the following matrix:

input image convolution kernel feature map

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & x_{11}^c & x_{12}^c & \dots & x_{132}^c & 0 \\ 0 & x_{21}^c & x_{22}^c & \dots & x_{232}^c & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & x_{321}^c & x_{322}^c & \dots & x_{3232}^c & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w_{11}^{cn} & w_{12}^{cn} \\ w_{21}^{cn} & w_{22}^{cn} \end{bmatrix} = \begin{bmatrix} z_{11}^n & z_{12}^n & \dots & z_{133}^n \\ z_{21}^n & z_{22}^n & \dots & z_{233}^n \\ \dots & \dots & \dots & \dots \\ z_{331}^n & z_{332}^n & \dots & z_{3333}^n \end{bmatrix}$$

The equation above provides an example of a 2x2 convolution kernel and a 5x5 input image. The framework comprises three convolutional layers. The initial two have a padding size, while the final layer does not contain any padding. In the input image, x represents the pixel and c represents the channel. The convolution kernel, w represents the weight and c represents the corresponding image channel, while n

represents the number of kernels. The feature map contains z as the feature map value and n as the feature map number.

Taking z_{11}^{n} as an example, the calculation process is shown in the below equation:

$$z_{11}^{n} = w_{11}^{cn}x_{11}^{c} + w_{12}^{cn}w_{12}^{cn} + w_{12}^{cn}w_{12}^{cn} + w_{22}^{cn}x_{22}^{c}$$

B . Loss function

The loss function used in this framework is the cross-entropy loss function, the equation is shown as

$$L = -\sum_{c=1}^{M} y_c \cdot log(\hat{y}_c)$$

where M represents the number of categories in the current dataset, y(c) represents the label of class c (y(c) is 1 if data belongs to class i, and 0 otherwise), and $\hat{y}$(c) represents the confidence of class c in the model prediction results.

Moreover, according to the source code in PyTorch, the 'F.cross_entropy()' function is implemented by combining the 'log_softmax' and the 'nll_loss'. Log-Softmax is to perform an extra log operation after Softmax, thus its equation is shown as:

$$\text{LogSoftmax}(x)_i = \log\left(\frac{\exp(y_k)}{\sum_j \exp(y_j)}\right)$$

Here y represents the prediction results of the model, the numerator of the fraction represents the Euler exponent of the k-th y, and the denominator represents the sum of all Euler exponent of y.

The equation of 'nll_loss' (negative log-likelihood) is shown as equation:

$$\ell(x,t) = -\sum_{i=0}^{N} (xt)_i$$

Here: x represents the input values, which are typically the predicted probabilities of each class from your model; t represents the true labels, usually in a one-hot encoded format; N represents the number of labels or classes; $(xt)_i$represents the t-th element in the i-th input.

In the context of machine learning, this loss function is often used in conjunction with softmax activation in the output layer, which gives you a probability distribution over classes. The NLL loss then measures how good these predicted probabilities are to the true labels. The lower the NLL loss, the better your model's predictions.

Therefore, the equation you provided is indeed the formula for the cross-entropy loss function as implemented in PyTorch:

$$L(y,t) = nll(logsoftmax(y),t) = -\sum_{i=0}^{N} [\log(\frac{e^{y_k}}{\sum_j e^{y_j}})t]$$

II. PROBLEM 2

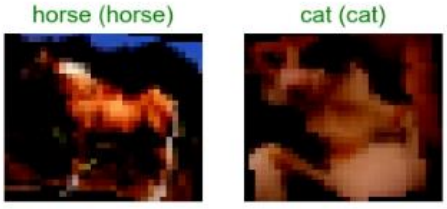
The framework is trained on the CIFAR-10 dataset. The batch size was set to 10 and it trained 300 epochs. The learning rate was initially set to 0.0001, and the momentum was set to 0.9, which can assist the optimizer in maintaining direction throughout the optimization process and preventing repeated oscillations during the gradient descent process. The final accuracy performance of the classification is 78% (7808/10000), with a loss of 0.648678.

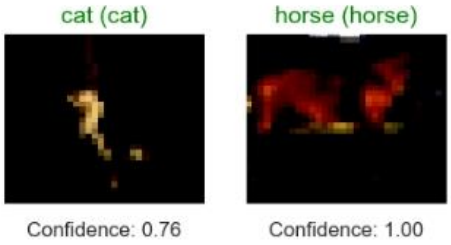
```
Test Loss: 0.648678

Test Accuracy of airplane: 82% (824/1000)
Test Accuracy of automobile: 88% (880/1000)
Test Accuracy of  bird: 67% (671/1000)
Test Accuracy of   cat: 59% (592/1000)
Test Accuracy of  deer: 79% (793/1000)
Test Accuracy of   dog: 67% (678/1000)
Test Accuracy of  frog: 83% (839/1000)
Test Accuracy of horse: 82% (822/1000)
Test Accuracy of  ship: 89% (894/1000)
Test Accuracy of truck: 81% (815/1000)

Test Accuracy (Overall): 78% (7808/10000)
```

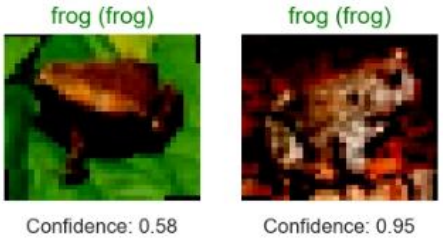
We can observe from the screenshot that the ship has the highest accuracy at 89%, while the cat has the lowest at 59%. Additionally, the accuracy of deer, dog, and bird is comparatively low, all below 80%. On the other hand, the accuracy of frogs, horses, trucks, automobiles, and airplanes is relatively high, all above 80%.

Furthermore, to enhance the clarity of the model's performance, we have chosen some well-classified and misclassified examples.

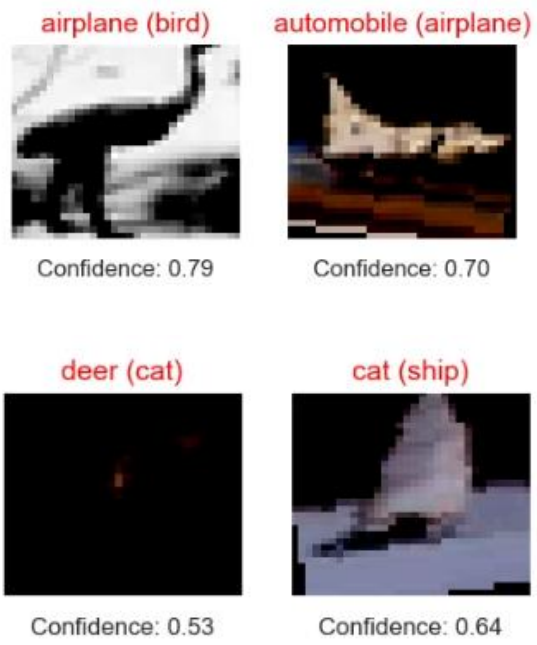
These are some well-classified examples with the corresponding confidence rate:





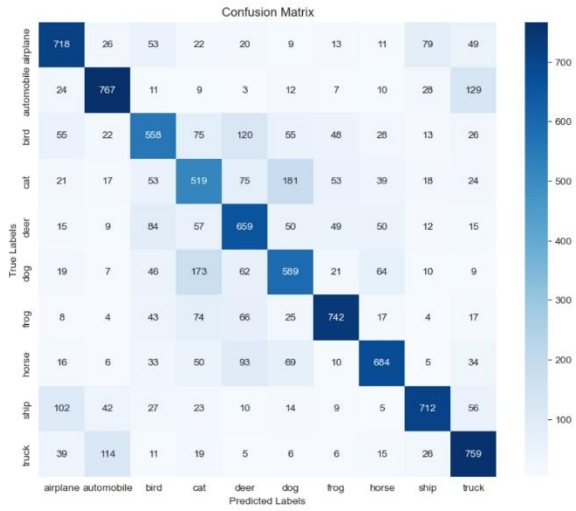


This is a misclassified example, although the accuracy of not low, it could also result in misleading classification.



The training and validation metrics and the loss and accuracy confusion matrix are shown below to illustrate the trend of these two measurements.

It is noteworthy that the accuracy rate shows an upward trend as the number of training epochs increases, as demonstrated by the metric chart. However, beyond the halfway point of 150 epochs, the upward trend becomes less apparent.



III. PROBLEM 3

In this section, two methods for improvement will be explained. The first method involves adjusting parameters based on the source code to achieve the accuracy demonstrated and imporove the CNN's performance. The second method is to construct a ResNet to reduce the model size while maintaining the current accuracy. Additionally, various other existing methods for CIFAR classification will be compared to gain relevant insights.

A. First optimization

Introducing Batch Normalization to normalize each mini-batch of data helps reduce internal covariate drift and stabilize the model during training, while also tackling the vanishing gradient problem. Additionally, batch processing has a regularization effect, reducing dependence on other regularization techniques to a certain extent.

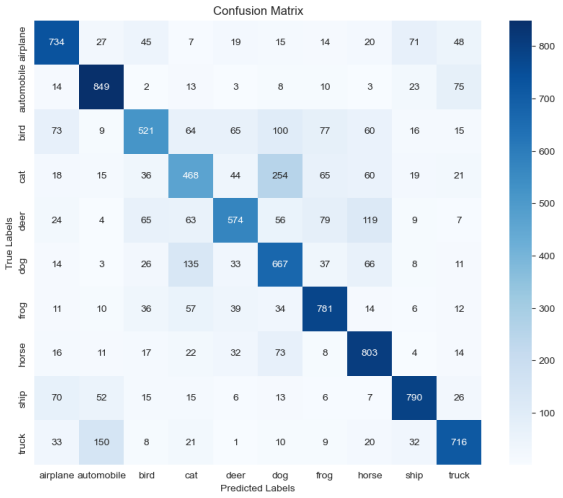
Secondly, the model's computational burden is reduced by decreasing the convolution kernel size and implementing a 1x1 Convolutional Kernel to lower the dimension of the channel.

Additionally, the model includes a nonlinear activation function and introduces nonlinear mapping to enhance its expression capability. After optimization, the model's parameters increased to 75186, and its estimated total size is 0.07 MB.

The overall precision on the test set is 68%, showing a slight improvement compared to the baseline model. The classification accuracy for the cat and bird categories has slightly improved, although the initial classification accuracy remains higher.

However, there is hardly any progress in the automobile, truck, and ship categories.

```
Test Loss: 0.912624

Test Accuracy of airplane: 73% (734/1000)
Test Accuracy of automobile: 78% (789/1000)
Test Accuracy of  bird: 52% (527/1000)
Test Accuracy of   cat: 51% (512/1000)
Test Accuracy of  deer: 62% (626/1000)
Test Accuracy of   dog: 58% (583/1000)
Test Accuracy of  frog: 76% (768/1000)
Test Accuracy of horse: 74% (748/1000)
Test Accuracy of  ship: 80% (809/1000)
Test Accuracy of truck: 72% (721/1000)

Test Accuracy (Overall): 68% (6817/10000)
```



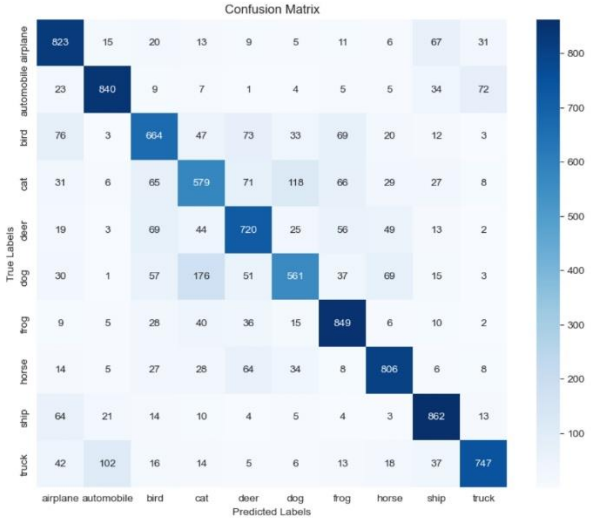
B. Further fine-tuning parameters

Realizing that the model's structure is relatively simple, I conducted additional optimization and parameter adjustments, along with some architecture modifications.

Based on the initial code, the accuracy was improved to 78.08% by modifying the convolution kernel's size and adjusting the learning rate to 0.003. Additionally, a Dropout layer was included to regularize and prevent model overfitting. Dropout is a technique for avoiding overfitting by randomly setting some output values to zero while training the neural network. This reduces the interdependence between network layers and enhances the model's generalization capabilities.

```python
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import lr_scheduler

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        self.fc2 = nn.Linear(500, 10)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = Net()
if train_on_gpu:
    model.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.003)
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

Furthermore, the validation loss and accuracy metrics and the accuracy confusion matrix are shown below to demonstrate the improvement of the exercise.



C. ResNet

In this section, we use ResNet which can improve the accuracy performance and reduce the size of the model effectively. Kaiming He et al. devised a "skip connection" (or "shortcut connection") framework that endows this network with enhanced identity mapping ability, leading to increased network depth and improved network performance [1].

This article's model improved the ResNet model by reducing its size. The input for the network is a 32x32 image, with the average value subtracted from all points after preprocessing. By implementing residual connectivity, the neural network can utilize input features directly without requiring an additional feature extraction layer, thereby decreasing the number of parameters and computational effort.

The construction of ResNet implemented in this report, involves the following steps:

(1). Define the network structure: ResNet18 was chosen as the appropriate structure for feature extraction and information transfer, given the characteristics of the CIFAR-10 dataset.

(2). Adjusting network parameters: This was accomplished by creating a new fully connected layer on the pre-trained ResNet18 model. The input characteristic quantity of this layer is the same as the original fully connected layer, while its output characteristic quantity is 10, aligned with the 10 categories of the CIFAR-10 dataset.

(3). Train the network model: The network model is trained by clearing the optimizer gradient, generating an output via the model, calculating

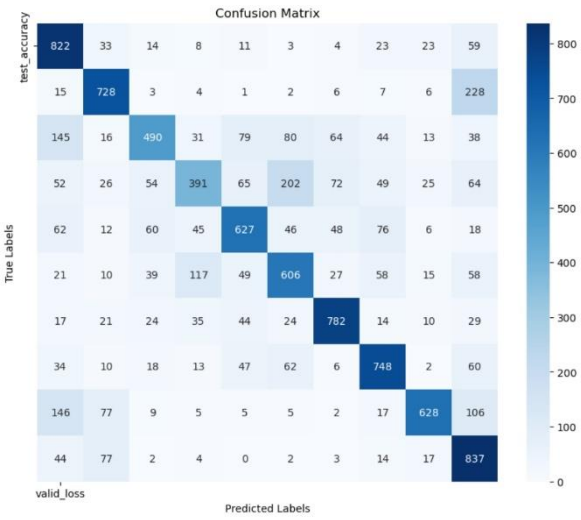
the loss, backpropagating the loss, and calculating the gradient. Finally, the model's parameters are updated using the optimizer.

(4). Test the network model: Use the test set after completing the training process. To test the network model, use the test set after completing the training process. To test the network model, use the test set after completing the training process. Evaluate the model's ability to generalize and calculate its accuracy on new data.

After processing only ten epochs, the model achieved a notably high accuracy rate. The specific accuracy rate is displayed in the screenshot.

```
Epoch 1, Loss: 0.0373, Validation Accuracy: 44.13%
Epoch 2, Loss: 0.0539, Validation Accuracy: 48.01%
Epoch 3, Loss: 0.0264, Validation Accuracy: 59.86%
Epoch 4, Loss: 0.0259, Validation Accuracy: 63.35%
Epoch 5, Loss: 0.0377, Validation Accuracy: 57.78%
Epoch 6, Loss: 0.0386, Validation Accuracy: 66.46%
Epoch 7, Loss: 0.0264, Validation Accuracy: 67.49%
Epoch 8, Loss: 0.0200, Validation Accuracy: 69.21%
Epoch 9, Loss: 0.0975, Validation Accuracy: 58.59%
Epoch 10, Loss: 0.0215, Validation Accuracy: 67.09%
Test Accuracy: 66.53%
```

What's more, the training and validation metrics and the loss and accuracy confusion matrix are shown below.



C. Other improved methods

The next discovery was other methods used for classification. By comparing their advantages and disadvantages, I found that classification research can have many other solutions and that multiple methods can be combined to achieve lighter and more accurate results.

In this paper (D. Yeboah, M. S. A. Sanoussi, and G. K. Agordzo,2021), they study deep learning-based image classification using TensorFlow GPU[2]. Similarly, the CIFAR-10 data set was used. Although more time was consumed in the training process and more losses were incurred when classifying images, very good accuracy was achieved. The test accuracy is 89.59%. The results reveal state-of-the-art accuracy and efficiency for deep learning-based image classification using TensorFlow GPUs. Methods used to improve performance.

This article (M. S and E. Karthikeyan,2022) also performs classification by extracting features from the input images, and they use CIFAR 10, CIFAR 100, and MNIST datasets to obtain the best results. The extracted information is

processed with the help of a Deep Neural Network (DNN) and image classification is done using SoftMax classifier. The combination of DNN and SoftMax classifier provides efficient image classification, with Training and Validation Accuracy reaching 0.9 and 0.85 respectively, and Training and Validation Loss reduced to 0.06 and 0.05 respectively.

Conclusion:

To demonstrate the effectiveness of our approach, we fine-tuned parameters and trained the ResNet18 model on the CIFAR-10 dataset. Our method was then compared to the latest TensorFlow, DNN, and SoftMax classic classification networks from existing papers. Test results were obtained and compared to our method, with the comparison results presented in Table 1.

Method	Accuracy
Origin	67%
First optimization	68%
Fine-tuning	78%
ResNet	67%
TensorFlow	90%
DNN	90%
SoftMax	85%

Table 1

Through a detailed analysis of experiments and comparisons, our network's classification performance is clearly demonstrated.

References:

[1] K. M. He, X. Y. Zhang, S. Q. Ren, et al. Deep residual learning for image recognition[C]. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition, 2016: 770-778.

[2] D. Yeboah, M. S. A. Sanoussi and G. K. Agordzo, "Image Classification Using TensorFlow GPU," 2021 International Conference on ICT for Smart Society (ICISS), Bandung, Indonesia, 2021, pp. 1-5, doi: 10.1109/ICISS53185.2021.9532500.

[3] M. S and E. Karthikeyan, "Classification of Image using Deep Neural Networks and SoftMax Classifier with CIFAR datasets," 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2022, pp. 1132-1135, doi: 10.1109/ICICCS53718.2022.9788359.

Appendix:

Source code after fine-tuning parameters

```
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import lr_scheduler

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        self.fc2 = nn.Linear(500, 10)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = Net()
if train_on_gpu:
    model.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.003)
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

Source code for ResNet:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models

# Check if GPU is available
train_on_gpu = torch.cuda.is_available()

# Load pre-trained ResNet model
model = models.resnet18(pretrained=True)

# Modify the classifier for CIFAR-10
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)  # 10 output classes for CIFAR-10

# Move the model to GPU if available
if train_on_gpu:
    model.cuda()

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Training the model
n_epochs = 10
for epoch in range(1, n_epochs+1):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    valid_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in valid_loader:
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()

            output = model(data)
            loss = criterion(output, target)
            valid_loss += loss.item()

            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

    valid_loss /= len(valid_loader.dataset)
    accuracy = correct / total

    print('Epoch {}, Loss: {:.4f}, Validation Accuracy: {:.2f}%'.format(epoch, valid_loss, accuracy * 100))

# Test the model
model.eval()
test_correct = 0
test_total = 0
with torch.no_grad():
    for data, target in test_loader:
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()

        output = model(data)
        _, predicted = torch.max(output.data, 1)
        test_total += target.size(0)
        test_correct += (predicted == target).sum().item()

test_accuracy = test_correct / test_total
print('Test Accuracy: {:.2f}%'.format(test_accuracy * 100))
```

```
Epoch 1, Loss: 0.0373, Validation Accuracy: 44.13%
Epoch 2, Loss: 0.0539, Validation Accuracy: 48.01%
Epoch 3, Loss: 0.0264, Validation Accuracy: 59.86%
Epoch 4, Loss: 0.0259, Validation Accuracy: 63.35%
Epoch 5, Loss: 0.0377, Validation Accuracy: 57.78%
Epoch 6, Loss: 0.0386, Validation Accuracy: 66.46%
Epoch 7, Loss: 0.0264, Validation Accuracy: 67.49%
Epoch 8, Loss: 0.0260, Validation Accuracy: 69.21%
Epoch 9, Loss: 0.0975, Validation Accuracy: 58.59%
Epoch 10, Loss: 0.0215, Validation Accuracy: 67.09%
Test Accuracy: 66.53%
```