Kabir_Notebook_6

November 27, 2024

1 Intro ML Homework 6

1.1 Name: Jaskin Kabir

1.2 Student ID: 801186717

Github: https://github.com/jaskinkabir/Intro_ML/tree/main/HM5

[63]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import torch
from torch import nn

path = 'housing.csv'
housing = pd.DataFrame(pd.read_csv(path))

varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea', 'furnishingstatus']

# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, 'no': 0, 'unfurnished': 0, 'semi-furnished': 1,␣
 ↪'furnished': 2})

# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
housing.head()

df_train, df_test = train_test_split(housing, train_size=0.8, test_size=0.2,␣
 ↪random_state=100)

Y_train_tensor_p1 = df_train.pop('price')
X_train = df_train

Y_test_tensor_p1 = df_test.pop('price')
X_test_batch = df_test
```

```
X_train.head()


b_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',␣
 ↪'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']
X_p3 = housing[b_vars]
X_train_df_p3 = X_train[b_vars]
X_test_df_p3 = X_test_batch[b_vars]

X_train_tensor_p1 = X_train_df_p3.to_numpy()
X_test_tensor_p1 = X_test_df_p3.to_numpy()

scaler = preprocessing.StandardScaler().fit(X_train_tensor_p1)
X_train_tensor_p1 = scaler.transform(X_train_tensor_p1)
X_test_tensor_p1 = scaler.transform(X_test_tensor_p1)
```

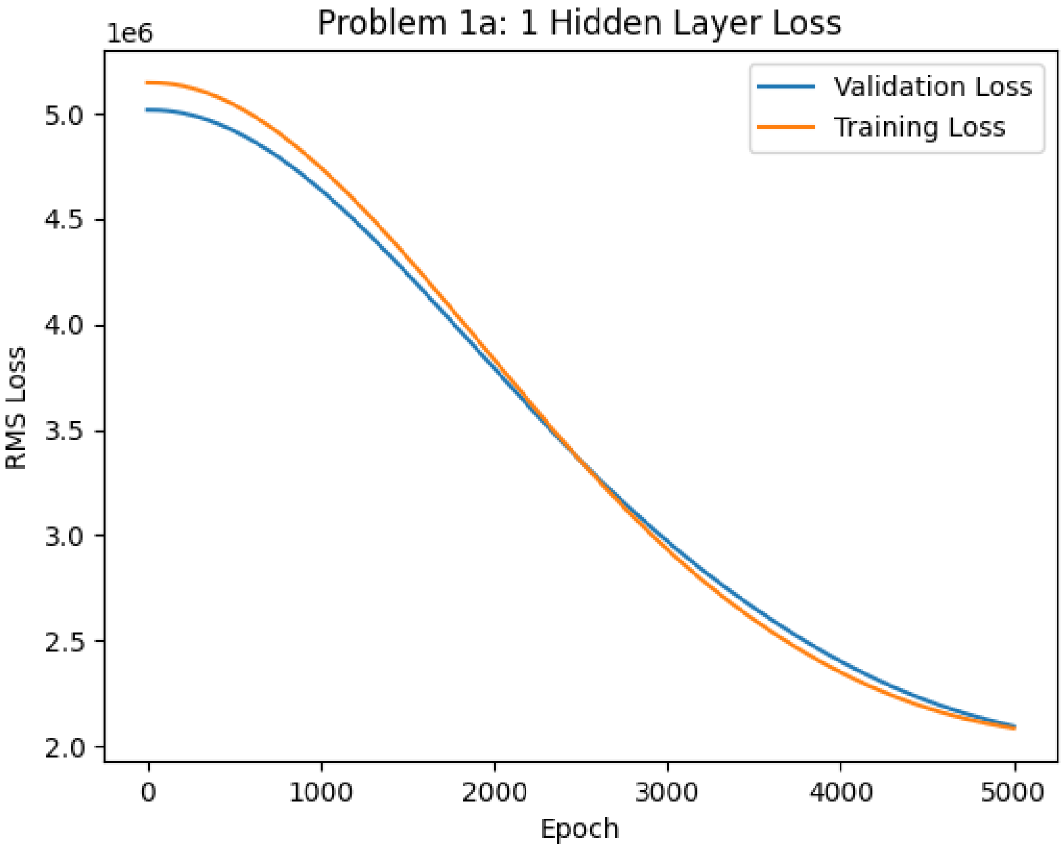
[64]:

```
from sklearn.metrics import root_mean_squared_error
import matplotlib.pyplot as plt

class Regressor(nn.Module):
    @classmethod
    def compare_results(cls, results1, results2):
        print(100 * (results1 - results2) / results1)

    def __init__(self, in_dim, out_dim, hidden_layers=[64,32], activation=nn.
 ↪Tanh,):
        super().__init__()
        self.hidden_layers = hidden_layers
        self.activation = activation
        self.input_cols = []
        self.output_cols = []

        #Error Mode is a 5 bit integer, with each bit representing a feature
        # If the bit is 1, the feature is errored
        output_features = out_dim
        input_features = in_dim



        self.stack_list = [nn.Linear(input_features, hidden_layers[0]),␣
 ↪activation()]
        for i in range(1, len(hidden_layers)):
            self.stack_list.extend([nn.Linear(hidden_layers[i-1],␣
 ↪hidden_layers[i]), activation()])
        self.stack_list.extend([nn.Linear(hidden_layers[-1], output_features)])
        self.stack = nn.Sequential(*self.stack_list)
    def train(self, epochs, X_train, X_test, Y_train, Y_test, alpha=1e-2, loss_fn=nn.MSELoss(),):

        val_hist = np.zeros(epochs)
        train_hist = np.zeros(epochs)

        optimizer = torch.optim.Adam(self.parameters(), lr=alpha)
        for i in range(epochs):
            optimizer.zero_grad()
            Y_pred = self.forward(X_train)
            loss = loss_fn(Y_pred.squeeze(), Y_train)
            loss.backward()
            optimizer.step()
            train_hist[i] = np.sqrt(loss.item())


            with torch.no_grad():
                Y_pred_val = self.forward(X_test)
                val_hist[i] = np.sqrt(loss_fn(Y_pred_val, Y_test).item())

        self.last_test = Y_test_tensor_p1
        self.last_pred = self.forward(X_test)
        self.last_score = val_hist[-1]

        self.last_epochs = epochs
        self.last_val_hist = val_hist
        self.last_train_hist = train_hist

    def plot_loss(self, title):

        plt.plot(range(self.last_epochs), self.last_val_hist, label='Validation Loss')
        plt.plot(range(self.last_epochs), self.last_train_hist, label='Training Loss')
        plt.title(title)
        plt.xlabel('Epoch')
        plt.ylabel('RMS Loss')
        plt.legend()
        plt.show()

    def forward(self, x):
        return self.stack(x)

    def print_results(self):
        if self.last_score is None:
            raise ValueError('No results to print')
        print(f'MSE: {self.last_score:.2E}')

class CustomMSELoss(nn.Module):
    def __init__(self, lambda_val=0.0):
        super(CustomMSELoss, self).__init__()
        self.lambda_val = lambda_val

    def forward(self, predictions, targets):

        m = targets.size(0)

        errors = predictions - targets
        mse_loss = (1 / (2*m)) * torch.sum(errors ** 2)




        total_loss = mse_loss

        return total_loss
```

[65]:

```
device = 'cpu'


X_train_tensor_p1 = torch.tensor(X_train_tensor_p1).to(device).float()
X_test_tensor_p1 = torch.tensor(X_test_tensor_p1).to(device).float()

Y_train_tensor_p1 = torch.tensor(Y_train_tensor_p1.to_numpy()).to(device).
 ↪float().view(-1, 1)
Y_test_tensor_p1 = torch.tensor(Y_test_tensor_p1.to_numpy()).to(device).float().
 ↪view(-1, 1)


model_1a = Regressor(
    in_dim=X_train_tensor_p1.shape[1],
    out_dim=1,
    hidden_layers=[8],
    activation=nn.ReLU,
).to(device)

model_1a.train(
    epochs=5000,
    X_train=X_train_tensor_p1,
    X_test=X_test_tensor_p1,
    Y_train=Y_train_tensor_p1,
    Y_test=Y_test_tensor_p1,
    alpha=1e-1,
    loss_fn=nn.MSELoss(),
)

model_1a.plot_loss('Problem 1a: 1 Hidden Layer Loss')
model_1a.print_results()
```

```
/home/jaskin/.local/lib/python3.11/site-packages/torch/nn/modules/loss.py:608:
UserWarning: Using a target size (torch.Size([436, 1])) that is different to the
input size (torch.Size([436])). This will likely lead to incorrect results due
to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```



```
MSE: 2.09E+06
```

[67]:
```
from sklearn.datasets import load_breast_cancer

breast = load_breast_cancer()
X_2 = breast.data
Y_2 = breast.target
```

```
X_train_2, X_test_2, Y_train_2, Y_test_2 = train_test_split(X_2, Y_2,␣
 ↪train_size=0.8, test_size=0.2, random_state=00)

scaler = preprocessing.StandardScaler().fit(X_train_2)
X_train_2 = scaler.transform(X_train_2)
X_test_2 = scaler.transform(X_test_2)
```

[68]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix, classification_report

class Classifier(nn.Module):
    @classmethod
    def compare_results(cls, results1, results2):
        print('Comparing results:')
        comparisons = {
            'accuracy': 100*(results1['accuracy'] - results2['accuracy'])/
 ↪results1['accuracy'],
            'precision': 100*(results1['precision'] - results2['precision'])/
 ↪results1['precision'],
            'recall': 100*(results1['recall'] - results2['recall'])/
 ↪results1['recall'],
            'f1': 100*(results1['f1'] - results2['f1'])/results1['f1']
        }
        for key, value in comparisons.items():
            print(f'{key}: {value} %')

    def __init__(self):
        super().__init__()

    def get_results(self, Y_test=None, Y_pred=None):
        if Y_test is None:
            Y_test = self.last_test
        if Y_pred is None:
            Y_pred = self.last_pred

        if isinstance(Y_test, torch.Tensor):
            Y_test = Y_test.cpu().detach().numpy()
        if isinstance(Y_pred, torch.Tensor):
            Y_pred = Y_pred.cpu().detach().numpy()
        results = {
            'accuracy': accuracy_score(Y_test, Y_pred),
            'precision': precision_score(Y_test, Y_pred, average='weighted'),
            'recall': recall_score(Y_test, Y_pred, average='weighted'),
            'f1': f1_score(Y_test, Y_pred, average='weighted'),
            'confusion_matrix': confusion_matrix(Y_test, Y_pred),
            'classification_report': classification_report(Y_test, Y_pred)
```

```python
        }
        self.last_results = results
        return results
    def print_results(self, results=None):
        if results is None:
            try:
                results = self.last_results
            except:
                results = self.get_results()
        for key, value in results.items():
            if key in ['confusion_matrix', 'classification_report']:
                print(f'{key.capitalize()}:\n{value}')
            else:
                print(f'{key.capitalize()}: {value}')


class LogisticClassifier(Classifier):
    def __init__(self, input_dim=0, activation=nn.ReLU, hidden_layers = [64, 32, 16], pass_through=False):
        super().__init__()
        if pass_through:
            return
        self.stack_list = [nn.Linear(input_dim, hidden_layers[0]), activation()]
        for i in range(1, len(hidden_layers)):
            self.stack_list.extend([nn.Linear(hidden_layers[i-1], hidden_layers[i]), activation()])

        self.stack_list.extend([nn.Linear(hidden_layers[-1], 1), nn.Sigmoid()])
        self.stack = nn.Sequential(*self.stack_list)

    def forward(self, x):
        return self.stack(x)

    def predict(self, x):
        with torch.no_grad():
            return self.forward(x).round()

    def train(self, epochs, X_train, X_test, Y_train, Y_test, alpha, loss_fn=nn.BCELoss(), print_epoch=500):
        optimizer = torch.optim.SGD(self.parameters(), lr=alpha)

        for epoch in range(epochs):
            optimizer.zero_grad()
            Y_pred = self.forward(X_train)
            loss = loss_fn(Y_pred, Y_train)
            loss.backward()
            optimizer.step()
            if epoch % print_epoch == 0:
                test_loss = loss_fn(self.forward(X_test), Y_test)
                print(f'Epoch {epoch}: Training Loss: {loss.item()}, Test Loss:␣
 ↪{test_loss.item()}')
        Y_pred = self.predict(X_test)
        self.last_pred = Y_pred
        self.last_test = Y_test
        return [Y_test,Y_pred]
```

[69]:

```
X_train_2 = torch.tensor(X_train_2).to(device).float()
X_test_2 = torch.tensor(X_test_2).to(device).float()
Y_train_2 = torch.tensor(Y_train_2).to(device).float().view(-1, 1)
Y_test_2 = torch.tensor(Y_test_2).to(device).float().view(-1, 1)

model_2a = LogisticClassifier(
    input_dim=X_train_2.shape[1],
    hidden_layers=[32],
    activation=nn.ReLU
).to(device)

model_2a.train(
    epochs=5000,
    X_train=X_train_2,
    X_test=X_test_2,
    Y_train=Y_train_2,
    Y_test=Y_test_2,
    alpha=1e-1,
    loss_fn=nn.BCELoss(),
    print_epoch=500
)

results_2a = model_2a.get_results()
model_2a.print_results(results_2a)
```

```
Epoch 0: Training Loss: 0.6775199174880981, Test Loss: 0.6508200168609619
Epoch 500: Training Loss: 0.05119786411523819, Test Loss: 0.08860453963279724
Epoch 1000: Training Loss: 0.03805161640048027, Test Loss: 0.09681770205497742
Epoch 1500: Training Loss: 0.029710467904806137, Test Loss: 0.10133590549230576
Epoch 2000: Training Loss: 0.023742537945508957, Test Loss: 0.09929131716489792
Epoch 2500: Training Loss: 0.018893906846642494, Test Loss: 0.09738191962242126
Epoch 3000: Training Loss: 0.015220258384943008, Test Loss: 0.09081994742155075
Epoch 3500: Training Loss: 0.012455631978809834, Test Loss: 0.08439692854881287
Epoch 4000: Training Loss: 0.010189911350607872, Test Loss: 0.07659681141376495
Epoch 4500: Training Loss: 0.008512535132467747, Test Loss: 0.06754373759031296
Accuracy: 0.9649122807017544
Precision: 0.9657164890247598
Recall: 0.9649122807017544
```

```
F1: 0.965011961722488
Confusion_matrix:
[[46  1]
 [ 3 64]]
Classification_report:
              precision    recall  f1-score   support

         0.0       0.94      0.98      0.96        47
         1.0       0.98      0.96      0.97        67

    accuracy                           0.96       114
   macro avg       0.96      0.97      0.96       114
weighted avg       0.97      0.96      0.97       114
```

[70]:

```
model_2b = LogisticClassifier(
    input_dim=X_train_2.shape[1],
    hidden_layers=[32,64,32],
    activation=nn.ReLU
).to(device)

model_2b.train(
    epochs=5000,
    X_train=X_train_2,
    X_test=X_test_2,
    Y_train=Y_train_2,
    Y_test=Y_test_2,
    alpha=5e-2,
    loss_fn=nn.BCELoss(),
    print_epoch=500
)
print('\n')
results_2b = model_2b.get_results()
model_2b.print_results(results_2b)
```

```
Epoch 0: Training Loss: 0.7006031274795532, Test Loss: 0.6969746351242065
Epoch 500: Training Loss: 0.0444357804954052, Test Loss: 0.09696567803621292
Epoch 1000: Training Loss: 0.023338234052062035, Test Loss: 0.0965108647942543
Epoch 1500: Training Loss: 0.01155084278434515, Test Loss: 0.09876886010169983
Epoch 2000: Training Loss: 0.0062743364833295345, Test Loss: 0.1080765500664711
Epoch 2500: Training Loss: 0.0038169343024492264, Test Loss: 0.11713045835494995
Epoch 3000: Training Loss: 0.0025181507226079702, Test Loss: 0.12493595480918884
Epoch 3500: Training Loss: 0.0017743278294801712, Test Loss: 0.13145829737186432
Epoch 4000: Training Loss: 0.001317627727985382, Test Loss: 0.1369219720363617
Epoch 4500: Training Loss: 0.00102024688385427, Test Loss: 0.1416374146938324


Accuracy: 0.9473684210526315
```

```
Precision: 0.9502514456074828
Recall: 0.9473684210526315
F1: 0.9476328183095101
Confusion_matrix:
[[46  1]
 [ 5 62]]
Classification_report:
              precision    recall  f1-score   support

         0.0       0.90      0.98      0.94        47
         1.0       0.98      0.93      0.95        67

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.95       114
weighted avg       0.95      0.95      0.95       114
```

[71]:
```python
# Import cifar-10 dataset
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.data import DataLoader

data_path = './data'
cifar10 = datasets.CIFAR10(data_path, train=True, download=True,␣
 ↪transform=transforms.ToTensor())

device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
Files already downloaded and verified
```

[72]:
```python
train_imgs = torch.stack([img for img, _ in cifar10], dim=3)
view = train_imgs.view(3, -1)#.to(device=device)

mean = train_imgs.view(3, -1).mean(dim=1)
std = train_imgs.view(3, -1).std(dim=1)

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,␣
 ↪transform=transform)
train_loader = DataLoader(cifar10_train, batch_size=64, shuffle=True)

cifar10_test = datasets.CIFAR10(data_path, train=False, download=True,␣
 ↪transform=transform)
```

```
test_loader = DataLoader(cifar10_test, batch_size=64, shuffle=False)
X_test_3 = torch.stack([img for img, _ in cifar10_test], dim=3).view(3, -1).
 ↪to(device=device)
Y_test_3 = torch.tensor([label for _, label in cifar10_test]).to(device=device)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[73]:

```
import time

class ImageClassifier(Classifier):
    def __init__(self, input_dim=0, output_dim = 0, activation=nn.ReLU,␣
 ↪hidden_layers = [64, 32, 16], pass_through=False):
        super().__init__()
        self.stack_list = [nn.Flatten(), nn.Linear(input_dim,␣
 ↪hidden_layers[0]), activation()]
        for i in range(1, len(hidden_layers)):
            self.stack_list.extend([nn.Linear(hidden_layers[i-1],␣
 ↪hidden_layers[i]), activation()])

        self.stack_list.extend([nn.Linear(hidden_layers[-1], output_dim), nn.
 ↪Softmax(dim=1)])
        self.stack = nn.Sequential(*self.stack_list)
    def forward(self, x):
        return self.stack(x)
    def predict(self, x):
        with torch.no_grad():
            return self.forward(x).argmax(dim=1)
    def train_model(
        self,
        epochs,
        train_loader,
        test_loader,
        alpha,
        loss_fn=nn.CrossEntropyLoss(),
        optimizer=torch.optim.SGD,
        print_epoch=10,
    ):

        optimizer = optimizer(self.parameters(), lr=alpha)
        training_time = 0
        for epoch in range(epochs):
            self.train()

            start_time = time.time()
            train_loss = 0
            for X_batch, Y_batch in train_loader:
                X_batch, Y_batch = X_batch.to(device), Y_batch.to(device)
                optimizer.zero_grad()
                Y_pred = self.forward(X_batch)
                loss = loss_fn(Y_pred, Y_batch)
                loss.backward()
                optimizer.step()

                train_loss += loss.item()
            training_time += time.time() - start_time


            self.eval()
            with torch.no_grad():
                test_loss = 0
                Y_pred_eval = []
                Y_test = []
                for X_test_batch, Y_test_batch in test_loader:
                    X_test_batch, Y_test_batch = X_test_batch.to(device),␣
 ↪Y_test_batch.to(device)

                    out = self.forward(X_test_batch)
                    test_loss += loss_fn(out, Y_test_batch).item()
                    Y_test.extend(Y_test_batch.cpu().detach().numpy())
                    Y_pred_eval.extend(out.argmax(dim=1).cpu().detach().numpy())

            accuracy = accuracy_score(Y_test, Y_pred_eval)
            if epoch % print_epoch == 0:
                print(f'Epoch {epoch}: Training Loss: {train_loss/
 ↪len(train_loader)}, Test Loss: {test_loss/len(test_loader)}, Accuracy:␣
 ↪{accuracy}')
        self.last_pred = torch.tensor(Y_pred_eval)
        self.last_test = torch.tensor(Y_test)
        print(f'\nTraining Time: {training_time} seconds\n')
```

[74]:

```
device = 'cuda'
model_3a = ImageClassifier(
    input_dim=3*32*32,
    output_dim=10,
    hidden_layers=[256],
    activation=nn.Tanh
).to(device)

model_3a.train_model(
    epochs=100,
    train_loader=train_loader,
    test_loader=test_loader,
    alpha=1e-2,
    loss_fn=nn.CrossEntropyLoss(),
    print_epoch=1
)

model_3a.get_results()
model_3a.print_results()
```

```
Epoch 0: Training Loss: 2.201410284737492, Test Loss: 2.1488690786300952,
Accuracy: 0.3354
Epoch 1: Training Loss: 2.1313129562855986, Test Loss: 2.115134690217911,
Accuracy: 0.3624
Epoch 2: Training Loss: 2.1072356914315384, Test Loss: 2.097880730963057,
Accuracy: 0.3769
Epoch 3: Training Loss: 2.0923693166364488, Test Loss: 2.086663885481039,
Accuracy: 0.3865
Epoch 4: Training Loss: 2.08120296205706, Test Loss: 2.077612098614881,
Accuracy: 0.3979
Epoch 5: Training Loss: 2.0715074409609255, Test Loss: 2.070352053945991,
Accuracy: 0.4042
Epoch 6: Training Loss: 2.0632094161589736, Test Loss: 2.064615787973829,
Accuracy: 0.4078
Epoch 7: Training Loss: 2.0561082657340846, Test Loss: 2.0597093469777685,
Accuracy: 0.4105
Epoch 8: Training Loss: 2.0497846004298275, Test Loss: 2.054938104501955,
Accuracy: 0.4137
Epoch 9: Training Loss: 2.0441966700127057, Test Loss: 2.0527782402220804,
Accuracy: 0.415
Epoch 10: Training Loss: 2.0390897987748655, Test Loss: 2.048685533225916,
Accuracy: 0.4202
Epoch 11: Training Loss: 2.034578595319977, Test Loss: 2.0454648467385845,
Accuracy: 0.4231
Epoch 12: Training Loss: 2.0302584508190984, Test Loss: 2.0433450481694218,
Accuracy: 0.4246
Epoch 13: Training Loss: 2.02627748418647, Test Loss: 2.0410740231252777,
Accuracy: 0.4278
Epoch 14: Training Loss: 2.0223188958204616, Test Loss: 2.038747470090344,
Accuracy: 0.4294
Epoch 15: Training Loss: 2.018991274144643, Test Loss: 2.037671606252148,
Accuracy: 0.4305
Epoch 16: Training Loss: 2.0157555367635642, Test Loss: 2.035060773229903,
Accuracy: 0.4328
Epoch 17: Training Loss: 2.0125163214285964, Test Loss: 2.032249483333272,
Accuracy: 0.4377
Epoch 18: Training Loss: 2.0089872252300878, Test Loss: 2.0313054520613067,
Accuracy: 0.4373
```

```
Epoch 19: Training Loss: 2.006210855053514, Test Loss: 2.030170185550763,
Accuracy: 0.4373
Epoch 20: Training Loss: 2.0031930728031853, Test Loss: 2.0274508234801565,
Accuracy: 0.4399
Epoch 21: Training Loss: 2.000550609567891, Test Loss: 2.026248433787352,
Accuracy: 0.4396
Epoch 22: Training Loss: 1.9977973473956212, Test Loss: 2.02506713502726,
Accuracy: 0.441
Epoch 23: Training Loss: 1.9950800313973975, Test Loss: 2.0247168244829603,
Accuracy: 0.4403
Epoch 24: Training Loss: 1.9924540940453024, Test Loss: 2.0225466793509805,
Accuracy: 0.4421
Epoch 25: Training Loss: 1.990202148552136, Test Loss: 2.021040697006663,
Accuracy: 0.4442
Epoch 26: Training Loss: 1.9875995176832388, Test Loss: 2.02035410009372,
Accuracy: 0.4446
Epoch 27: Training Loss: 1.9850812461370093, Test Loss: 2.018848002336587,
Accuracy: 0.4461
Epoch 28: Training Loss: 1.982587840093676, Test Loss: 2.0185956916991312,
Accuracy: 0.4447
Epoch 29: Training Loss: 1.9802616491647023, Test Loss: 2.017457759304411,
Accuracy: 0.447
Epoch 30: Training Loss: 1.977693507585989, Test Loss: 2.0163778438689604,
Accuracy: 0.4484
Epoch 31: Training Loss: 1.975706407633584, Test Loss: 2.0154311193782055,
Accuracy: 0.4482
Epoch 32: Training Loss: 1.9732406907679174, Test Loss: 2.014418176025342,
Accuracy: 0.4508
Epoch 33: Training Loss: 1.9709903735029117, Test Loss: 2.0140721251250833,
Accuracy: 0.4491
Epoch 34: Training Loss: 1.968823590394481, Test Loss: 2.0130390232535684,
Accuracy: 0.4507
Epoch 35: Training Loss: 1.9664480809665397, Test Loss: 2.0122012309967334,
Accuracy: 0.4499
Epoch 36: Training Loss: 1.964585790853671, Test Loss: 2.0116470315653805,
Accuracy: 0.4507
Epoch 37: Training Loss: 1.9623820021024445, Test Loss: 2.0096353451917124,
Accuracy: 0.4548
Epoch 38: Training Loss: 1.9603788617931668, Test Loss: 2.0097902701918486,
Accuracy: 0.454
Epoch 39: Training Loss: 1.958145472704602, Test Loss: 2.0092169615873106,
Accuracy: 0.4545
Epoch 40: Training Loss: 1.9559663636300264, Test Loss: 2.0086839882431518,
Accuracy: 0.4548
Epoch 41: Training Loss: 1.9539883107785374, Test Loss: 2.0087792797453083,
Accuracy: 0.4545
Epoch 42: Training Loss: 1.9520726197820795, Test Loss: 2.007216927352225,
Accuracy: 0.4554
```

```
Epoch 43: Training Loss: 1.9498786005522588, Test Loss: 2.0065962007850597,
Accuracy: 0.4567
Epoch 44: Training Loss: 1.9478924571705596, Test Loss: 2.0057929359423885,
Accuracy: 0.4583
Epoch 45: Training Loss: 1.9459039570425478, Test Loss: 2.0053204077823907,
Accuracy: 0.457
Epoch 46: Training Loss: 1.9441391784516746, Test Loss: 2.0043849876731823,
Accuracy: 0.4595
Epoch 47: Training Loss: 1.9421423170572656, Test Loss: 2.0044837848396058,
Accuracy: 0.4594
Epoch 48: Training Loss: 1.94016695022583, Test Loss: 2.002669891734032,
Accuracy: 0.4619
Epoch 49: Training Loss: 1.9382048354429358, Test Loss: 2.002578695868231,
Accuracy: 0.4605
Epoch 50: Training Loss: 1.9363141062924318, Test Loss: 2.0038897072433666,
Accuracy: 0.4596
Epoch 51: Training Loss: 1.9344832413946575, Test Loss: 2.0018469095230103,
Accuracy: 0.4612
Epoch 52: Training Loss: 1.9324918358832064, Test Loss: 2.0017270867232306,
Accuracy: 0.4616
Epoch 53: Training Loss: 1.9307439822675017, Test Loss: 2.001353355729656,
Accuracy: 0.4633
Epoch 54: Training Loss: 1.928962046685426, Test Loss: 2.0016647691179994,
Accuracy: 0.4608
Epoch 55: Training Loss: 1.927043501068564, Test Loss: 1.9999224835899985,
Accuracy: 0.4651
Epoch 56: Training Loss: 1.9250040407985678, Test Loss: 1.9994656105709683,
Accuracy: 0.4641
Epoch 57: Training Loss: 1.9234034970898153, Test Loss: 1.9991518092003597,
Accuracy: 0.4642
Epoch 58: Training Loss: 1.9215860725058924, Test Loss: 1.999201297000715,
Accuracy: 0.4643
Epoch 59: Training Loss: 1.9198868384446635, Test Loss: 1.9987571254657333,
Accuracy: 0.4647
Epoch 60: Training Loss: 1.9179791044396208, Test Loss: 1.9974228773906733,
Accuracy: 0.4659
Epoch 61: Training Loss: 1.916289828469991, Test Loss: 1.9975098523364705,
Accuracy: 0.4673
Epoch 62: Training Loss: 1.9146924146910762, Test Loss: 1.9978118755255536,
Accuracy: 0.4662
Epoch 63: Training Loss: 1.912617788137987, Test Loss: 1.9964036083525154,
Accuracy: 0.4676
Epoch 64: Training Loss: 1.9110569766415355, Test Loss: 1.995940009499811,
Accuracy: 0.468
Epoch 65: Training Loss: 1.9095145576750225, Test Loss: 1.9957442184922043,
Accuracy: 0.4668
Epoch 66: Training Loss: 1.9077311972218096, Test Loss: 1.995925371813926,
Accuracy: 0.469
```

```
Epoch 67: Training Loss: 1.906208941698684, Test Loss: 1.995034005231918,
Accuracy: 0.4691
Epoch 68: Training Loss: 1.9042365439712543, Test Loss: 1.9958832036158083,
Accuracy: 0.4686
Epoch 69: Training Loss: 1.9026686892180187, Test Loss: 1.994063199705379,
Accuracy: 0.4693
Epoch 70: Training Loss: 1.9011069923410635, Test Loss: 1.9941022388494698,
Accuracy: 0.4689
Epoch 71: Training Loss: 1.89944163963313, Test Loss: 1.9938461734990405,
Accuracy: 0.4708
Epoch 72: Training Loss: 1.8979422139084858, Test Loss: 1.9931916909612668,
Accuracy: 0.4695
Epoch 73: Training Loss: 1.8962264480188376, Test Loss: 1.9930588234761717,
Accuracy: 0.4729
Epoch 74: Training Loss: 1.894419810503645, Test Loss: 1.9929505951085669,
Accuracy: 0.4722
Epoch 75: Training Loss: 1.8932654946051595, Test Loss: 1.9926604366606209,
Accuracy: 0.469
Epoch 76: Training Loss: 1.8913014256740774, Test Loss: 1.9919216412647514,
Accuracy: 0.4722
Epoch 77: Training Loss: 1.8898866644600774, Test Loss: 1.9922871020189516,
Accuracy: 0.4712
Epoch 78: Training Loss: 1.8885408505759276, Test Loss: 1.9918207097205387,
Accuracy: 0.4715
Epoch 79: Training Loss: 1.8867450173553604, Test Loss: 1.9917527133492148,
Accuracy: 0.4733
Epoch 80: Training Loss: 1.885162004424483, Test Loss: 1.9901161619052765,
Accuracy: 0.4739
Epoch 81: Training Loss: 1.8836467884995443, Test Loss: 1.990831255153486,
Accuracy: 0.4749
Epoch 82: Training Loss: 1.8821341834409768, Test Loss: 1.9903806865594948,
Accuracy: 0.4735
Epoch 83: Training Loss: 1.880492230053143, Test Loss: 1.9894001423173648,
Accuracy: 0.4761
Epoch 84: Training Loss: 1.8792056093740341, Test Loss: 1.9900260746099387,
Accuracy: 0.474
Epoch 85: Training Loss: 1.877844167030071, Test Loss: 1.9894564698456199,
Accuracy: 0.4731
Epoch 86: Training Loss: 1.876429661765428, Test Loss: 1.9892730614182297,
Accuracy: 0.4771
Epoch 87: Training Loss: 1.8750434484323273, Test Loss: 1.9887491677217424,
Accuracy: 0.476
Epoch 88: Training Loss: 1.8735629942106165, Test Loss: 1.9892685428546493,
Accuracy: 0.4746
Epoch 89: Training Loss: 1.8719964249969443, Test Loss: 1.9882509078189825,
Accuracy: 0.4746
Epoch 90: Training Loss: 1.8706222033256765, Test Loss: 1.987908758175601,
Accuracy: 0.4747
```

```
Epoch 91: Training Loss: 1.869113135063435, Test Loss: 1.9885168736148033,
Accuracy: 0.476
Epoch 92: Training Loss: 1.8677857948081267, Test Loss: 1.988387291598472,
Accuracy: 0.4748
Epoch 93: Training Loss: 1.8664771082151272, Test Loss: 1.9878002739256355,
Accuracy: 0.475
Epoch 94: Training Loss: 1.8650445026509903, Test Loss: 1.9875242808821854,
Accuracy: 0.4747
Epoch 95: Training Loss: 1.8636897526433707, Test Loss: 1.9874323697606469,
Accuracy: 0.475
Epoch 96: Training Loss: 1.8621286147695673, Test Loss: 1.9872268529454613,
Accuracy: 0.4757
Epoch 97: Training Loss: 1.8607349558864408, Test Loss: 1.9873499710848377,
Accuracy: 0.4763
Epoch 98: Training Loss: 1.859359428248442, Test Loss: 1.986620311524458,
Accuracy: 0.4745
Epoch 99: Training Loss: 1.8583002029477482, Test Loss: 1.986218580015146,
Accuracy: 0.4742

Training Time: 1157.4642927646637 seconds

Accuracy: 0.4742
Precision: 0.46611456870171314
Recall: 0.4742
F1: 0.4682781879764696
Confusion_matrix:
[[587  37  61  25  20  17  24  36 137  56]
 [ 34 561  26  39  21  23  41  38  77 140]
 [105  37 289  78 141  69 143  69  42  27]
 [ 37  37  93 271  65 163 149  61  42  82]
 [ 50  25 128  58 394  57 116 111  31  30]
 [ 29  28 100 156  80 348  92  84  50  33]
 [ 11  24  58  70 127  63 563  29  25  30]
 [ 45  37  54  50  85  76  42 518  25  68]
 [ 89  58  16  24  13  28  14  20 681  57]
 [ 64 158  15  28  16  26  49  42  72 530]]
Classification_report:
              precision    recall  f1-score   support

           0       0.56      0.59      0.57      1000
           1       0.56      0.56      0.56      1000
           2       0.34      0.29      0.31      1000
           3       0.34      0.27      0.30      1000
           4       0.41      0.39      0.40      1000
           5       0.40      0.35      0.37      1000
           6       0.46      0.56      0.50      1000
           7       0.51      0.52      0.52      1000
           8       0.58      0.68      0.62      1000
           9       0.50      0.53      0.52      1000

    accuracy                           0.47     10000
   macro avg       0.47      0.47      0.47     10000
weighted avg       0.47      0.47      0.47     10000
```

[75]:

```
model_3a_results = model_3a.get_results()
model_3a.print_results(model_3a_results)
```

```
Accuracy: 0.4742
Precision: 0.46611456870171314
Recall: 0.4742
F1: 0.4682781879764696
Confusion_matrix:
[[587  37  61  25  20  17  24  36 137  56]
 [ 34 561  26  39  21  23  41  38  77 140]
 [105  37 289  78 141  69 143  69  42  27]
 [ 37  37  93 271  65 163 149  61  42  82]
 [ 50  25 128  58 394  57 116 111  31  30]
 [ 29  28 100 156  80 348  92  84  50  33]
 [ 11  24  58  70 127  63 563  29  25  30]
 [ 45  37  54  50  85  76  42 518  25  68]
 [ 89  58  16  24  13  28  14  20 681  57]
 [ 64 158  15  28  16  26  49  42  72 530]]
Classification_report:
              precision    recall  f1-score   support

           0       0.56      0.59      0.57      1000
           1       0.56      0.56      0.56      1000
           2       0.34      0.29      0.31      1000
           3       0.34      0.27      0.30      1000
           4       0.41      0.39      0.40      1000
           5       0.40      0.35      0.37      1000
           6       0.46      0.56      0.50      1000
           7       0.51      0.52      0.52      1000
           8       0.58      0.68      0.62      1000
           9       0.50      0.53      0.52      1000

    accuracy                           0.47     10000
   macro avg       0.47      0.47      0.47     10000
weighted avg       0.47      0.47      0.47     10000
```

Noticed the SGD optimizer is much slower than Adam, but yields more consistent results. To improve the model performance, the alpha was increased to 1e-2 to deal with the lower learning rate of SGD

[76]:

```
device = 'cuda'
model_3b = ImageClassifier(
    input_dim=3*32*32,
    output_dim=10,
    hidden_layers=[256,512,128],#[256,384,256],
    activation=nn.Tanh
).to(device)

model_3b.train_model(
    epochs=100,
    train_loader=train_loader,
    test_loader=test_loader,
    alpha=1e-2,
    loss_fn=nn.CrossEntropyLoss(),
    optimizer = torch.optim.SGD,
    print_epoch=1
)

model_3b_results = model_3a.get_results()
model_3b.print_results(model_3a_results)
```

```
Epoch 0: Training Loss: 2.280403497273965, Test Loss: 2.252564680804113,
Accuracy: 0.2359
Epoch 1: Training Loss: 2.228228701045141, Test Loss: 2.202385954036834,
Accuracy: 0.2707
Epoch 2: Training Loss: 2.1879612088508313, Test Loss: 2.172818032039958,
Accuracy: 0.2915
Epoch 3: Training Loss: 2.164266648804745, Test Loss: 2.152656055559778,
Accuracy: 0.316
Epoch 4: Training Loss: 2.145001944983402, Test Loss: 2.1356226805668728,
Accuracy: 0.3354
Epoch 5: Training Loss: 2.128756971615355, Test Loss: 2.1195827631434057,
Accuracy: 0.3519
Epoch 6: Training Loss: 2.113841903484081, Test Loss: 2.107796171668229,
Accuracy: 0.3608
Epoch 7: Training Loss: 2.1028579657949753, Test Loss: 2.098387276291088,
Accuracy: 0.371
Epoch 8: Training Loss: 2.0934950303848443, Test Loss: 2.090748143803542,
Accuracy: 0.3772
Epoch 9: Training Loss: 2.0848870728631765, Test Loss: 2.084034829382684,
Accuracy: 0.3808
Epoch 10: Training Loss: 2.0770044459406374, Test Loss: 2.078609412642801,
Accuracy: 0.3871
Epoch 11: Training Loss: 2.070014102532126, Test Loss: 2.0740907898374425,
Accuracy: 0.391
Epoch 12: Training Loss: 2.0636268466939707, Test Loss: 2.0721085997903423,
Accuracy: 0.3919
Epoch 13: Training Loss: 2.0574979936070457, Test Loss: 2.0637668682511445,
Accuracy: 0.3985
Epoch 14: Training Loss: 2.051519621058803, Test Loss: 2.0614985918543143,
Accuracy: 0.4013
Epoch 15: Training Loss: 2.046250664821976, Test Loss: 2.056406200311746,
Accuracy: 0.4054
Epoch 16: Training Loss: 2.0412039791836456, Test Loss: 2.053648417163047,
Accuracy: 0.4074
Epoch 17: Training Loss: 2.036012465539186, Test Loss: 2.049658063111032,
Accuracy: 0.4128
Epoch 18: Training Loss: 2.031437578561056, Test Loss: 2.0488287567333052,
Accuracy: 0.4131
Epoch 19: Training Loss: 2.0267347254411643, Test Loss: 2.0439856819286466,
Accuracy: 0.4178
Epoch 20: Training Loss: 2.022114611952506, Test Loss: 2.043041490445471,
Accuracy: 0.4191
Epoch 21: Training Loss: 2.017402906704437, Test Loss: 2.038575057011501,
Accuracy: 0.4234
Epoch 22: Training Loss: 2.0128561633322244, Test Loss: 2.0373852860396076,
Accuracy: 0.4235
Epoch 23: Training Loss: 2.0082216503675028, Test Loss: 2.0335943714068954,
Accuracy: 0.4275
Epoch 24: Training Loss: 2.0037876928553864, Test Loss: 2.030826511656403,
Accuracy: 0.4319
Epoch 25: Training Loss: 1.9994239329986865, Test Loss: 2.0289009050199183,
Accuracy: 0.4343
Epoch 26: Training Loss: 1.9945184483247644, Test Loss: 2.0266629237278253,
Accuracy: 0.4382
Epoch 27: Training Loss: 1.9897202751825533, Test Loss: 2.0271820012171555,
Accuracy: 0.437
Epoch 28: Training Loss: 1.9846469432191776, Test Loss: 2.0247295221705346,
Accuracy: 0.4392
Epoch 29: Training Loss: 1.9798921794842577, Test Loss: 2.0211990319999162,
Accuracy: 0.4406
Epoch 30: Training Loss: 1.975139911522341, Test Loss: 2.019865758859428,
Accuracy: 0.4439
Epoch 31: Training Loss: 1.970379686569009, Test Loss: 2.0177729889086096,
Accuracy: 0.4455
Epoch 32: Training Loss: 1.9653151279215313, Test Loss: 2.015972366758213,
Accuracy: 0.446
Epoch 33: Training Loss: 1.9602141008352685, Test Loss: 2.0135348678394487,
Accuracy: 0.4484
Epoch 34: Training Loss: 1.9551919712434949, Test Loss: 2.011253961332285,
Accuracy: 0.4515
Epoch 35: Training Loss: 1.950315807481556, Test Loss: 2.0147219919095374,
Accuracy: 0.4458
Epoch 36: Training Loss: 1.9453664797041423, Test Loss: 2.0082669774438164,
Accuracy: 0.4536
Epoch 37: Training Loss: 1.940512748782897, Test Loss: 2.009062681987787,
Accuracy: 0.4502
Epoch 38: Training Loss: 1.9353905344558189, Test Loss: 2.009172895152098,
Accuracy: 0.4531
Epoch 39: Training Loss: 1.9309233869128215, Test Loss: 2.0051034256151525,
Accuracy: 0.457
Epoch 40: Training Loss: 1.9258459394850085, Test Loss: 2.0054075057339515,
Accuracy: 0.4542
Epoch 41: Training Loss: 1.920948523389714, Test Loss: 2.0034787070219684,
Accuracy: 0.4572
Epoch 42: Training Loss: 1.9158197467589317, Test Loss: 2.003737722232843,
Accuracy: 0.4566
Epoch 43: Training Loss: 1.9112602746700083, Test Loss: 2.007011801573881,
Accuracy: 0.45
Epoch 44: Training Loss: 1.9068707323745084, Test Loss: 2.003478769284145,
Accuracy: 0.4543
Epoch 45: Training Loss: 1.90230433227461, Test Loss: 2.00412516609119,
Accuracy: 0.4558
Epoch 46: Training Loss: 1.8979026368816796, Test Loss: 2.0014274537942973,
Accuracy: 0.4565
Epoch 47: Training Loss: 1.8929194580868383, Test Loss: 2.005838249139725,
Accuracy: 0.4531
Epoch 48: Training Loss: 1.8888147938281983, Test Loss: 2.0037533644657985,
Accuracy: 0.4546
Epoch 49: Training Loss: 1.8840698384872787, Test Loss: 2.0042982344414777,
Accuracy: 0.4539
Epoch 50: Training Loss: 1.8803379241462863, Test Loss: 2.0060581659815115,
Accuracy: 0.4536
Epoch 51: Training Loss: 1.8755995143405007, Test Loss: 2.0026044678536192,
Accuracy: 0.4558
Epoch 52: Training Loss: 1.8712224228607723, Test Loss: 1.9998929014631137,
Accuracy: 0.4587
Epoch 53: Training Loss: 1.8668004260648547, Test Loss: 2.0050053262406853,
Accuracy: 0.4537
Epoch 54: Training Loss: 1.862304040080751, Test Loss: 1.9967357801024321,
Accuracy: 0.4616
Epoch 55: Training Loss: 1.8581476738995604, Test Loss: 2.011644670158435,
Accuracy: 0.4476
Epoch 56: Training Loss: 1.85449959600673, Test Loss: 2.0013837252452875,
Accuracy: 0.4555
Epoch 57: Training Loss: 1.8503952450154688, Test Loss: 2.005105645793259,
Accuracy: 0.4534
Epoch 58: Training Loss: 1.8461702011735237, Test Loss: 2.0121096296674885,
Accuracy: 0.4455
Epoch 59: Training Loss: 1.8425086915035687, Test Loss: 2.00076228418168,
Accuracy: 0.4561
Epoch 60: Training Loss: 1.8384573668470163, Test Loss: 1.9964367902962266,
Accuracy: 0.4612
Epoch 61: Training Loss: 1.834525834568931, Test Loss: 1.9943766305401067,
Accuracy: 0.4647
Epoch 62: Training Loss: 1.8304467589958855, Test Loss: 1.9952366450789627,
Accuracy: 0.4618
Epoch 63: Training Loss: 1.8265629633308371, Test Loss: 2.0005740686586706,
Accuracy: 0.456
Epoch 64: Training Loss: 1.822805018223765, Test Loss: 2.005883834923908,
Accuracy: 0.4491
Epoch 65: Training Loss: 1.81961776914499, Test Loss: 1.99604929632442,
Accuracy: 0.4605
Epoch 66: Training Loss: 1.8148655074331768, Test Loss: 1.9966157560895204,
Accuracy: 0.4607
Epoch 67: Training Loss: 1.8123863682417614, Test Loss: 2.001093756621051,
Accuracy: 0.4567
Epoch 68: Training Loss: 1.8089882707047036, Test Loss: 1.9976130207632756,
Accuracy: 0.4591
Epoch 69: Training Loss: 1.8050172597246097, Test Loss: 2.000473508409634,
Accuracy: 0.4585
Epoch 70: Training Loss: 1.8015150918680078, Test Loss: 1.9935544228098194,
Accuracy: 0.4642
Epoch 71: Training Loss: 1.798106653458627, Test Loss: 1.9967582954722605,
Accuracy: 0.459
Epoch 72: Training Loss: 1.7946201843373917, Test Loss: 1.9966439912273626,
Accuracy: 0.4595
Epoch 73: Training Loss: 1.7911296585941558, Test Loss: 1.9975928455401377,
Accuracy: 0.4605
Epoch 74: Training Loss: 1.7893220724352181, Test Loss: 1.9976800953506664,
Accuracy: 0.4607
Epoch 75: Training Loss: 1.7857283984913546, Test Loss: 1.997754140264669,
Accuracy: 0.461
Epoch 76: Training Loss: 1.7820604012140533, Test Loss: 2.0081516086675557,
Accuracy: 0.4476
Epoch 77: Training Loss: 1.7795437854879044, Test Loss: 2.014743470841912,
Accuracy: 0.442
Epoch 78: Training Loss: 1.7772340364468373, Test Loss: 1.9949941035288914,
Accuracy: 0.4622
Epoch 79: Training Loss: 1.774098057423711, Test Loss: 2.003993206722721,
Accuracy: 0.4537
Epoch 80: Training Loss: 1.771172871979911, Test Loss: 1.996228008513238,
Accuracy: 0.4629
Epoch 81: Training Loss: 1.7697991905614847, Test Loss: 1.998249265039043,
Accuracy: 0.4581
Epoch 82: Training Loss: 1.7668739221894834, Test Loss: 2.0033721316392255,
Accuracy: 0.4551
Epoch 83: Training Loss: 1.7646816086281292, Test Loss: 1.9957349505394129,
Accuracy: 0.4613
Epoch 84: Training Loss: 1.7610802517827515, Test Loss: 1.9970971710363012,
Accuracy: 0.4589
Epoch 85: Training Loss: 1.7594040022481738, Test Loss: 1.995714901359218,
Accuracy: 0.4591
Epoch 86: Training Loss: 1.7561884853236205, Test Loss: 2.000106747742671,
Accuracy: 0.4548
Epoch 87: Training Loss: 1.7541905739118375, Test Loss: 1.9980947082969034,
Accuracy: 0.4594
Epoch 88: Training Loss: 1.752043614302145, Test Loss: 1.9978720876061993,
Accuracy: 0.4585
Epoch 89: Training Loss: 1.7497063177016081, Test Loss: 2.04112223758819,
Accuracy: 0.4134
Epoch 90: Training Loss: 1.7477915985200105, Test Loss: 1.9977301518628552,
Accuracy: 0.4571
Epoch 91: Training Loss: 1.745197372515793, Test Loss: 2.0041657192691877,
Accuracy: 0.4528
Epoch 92: Training Loss: 1.7442166652825788, Test Loss: 1.998430759284147,
Accuracy: 0.4579
Epoch 93: Training Loss: 1.7417706339560506, Test Loss: 2.000380478087504,
Accuracy: 0.4567
Epoch 94: Training Loss: 1.7388503150561887, Test Loss: 2.007878496388721,
Accuracy: 0.4491
Epoch 95: Training Loss: 1.7377261348697535, Test Loss: 2.003134667493735,
Accuracy: 0.4536
Epoch 96: Training Loss: 1.7351144726014198, Test Loss: 2.000657352672261,
Accuracy: 0.4562
Epoch 97: Training Loss: 1.7338576975380977, Test Loss: 2.0077067985656156,
Accuracy: 0.4488
Epoch 98: Training Loss: 1.7315342722036648, Test Loss: 2.003403690969868,
Accuracy: 0.4531
Epoch 99: Training Loss: 1.730216651621377, Test Loss: 2.002178602917179,
Accuracy: 0.4532

Training Time: 1186.6000225543976 seconds

Accuracy: 0.4742
Precision: 0.46611456870171314
Recall: 0.4742
F1: 0.4682781879764696
Confusion_matrix:
[[587  37  61  25  20  17  24  36 137  56]
 [ 34 561  26  39  21  23  41  38  77 140]
 [105  37 289  78 141  69 143  69  42  27]
 [ 37  37  93 271  65 163 149  61  42  82]
 [ 50  25 128  58 394  57 116 111  31  30]
 [ 29  28 100 156  80 348  92  84  50  33]
 [ 11  24  58  70 127  63 563  29  25  30]
 [ 45  37  54  50  85  76  42 518  25  68]
 [ 89  58  16  24  13  28  14  20 681  57]
 [ 64 158  15  28  16  26  49  42  72 530]]
Classification_report:
```

```
              precision    recall  f1-score   support

           0       0.56      0.59      0.57      1000
           1       0.56      0.56      0.56      1000
           2       0.34      0.29      0.31      1000
           3       0.34      0.27      0.30      1000
           4       0.41      0.39      0.40      1000
           5       0.40      0.35      0.37      1000
           6       0.46      0.56      0.50      1000
           7       0.51      0.52      0.52      1000
           8       0.58      0.68      0.62      1000
           9       0.50      0.53      0.52      1000

    accuracy                           0.47     10000
   macro avg       0.47      0.47      0.47     10000
weighted avg       0.47      0.47      0.47     10000
```