moncnn2

November 26, 2025

[306]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader, random_split
import os

np.random.seed(0)
```

[307]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
else:
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[308]:
```
import numpy as np
from skimage.transform import warp_polar

def to_polar(img, output_shape=(32, 64)):
    # img: numpy (28,28,3) or grayscale
    # convert to grayscale first
    if img.ndim == 3:
        img_gray = img.mean(axis=2)
    else:
        img_gray = img

    polar = warp_polar(
```

```
        img_gray,
        output_shape=output_shape,
        scaling='linear',
        center=None
    )
    return polar.astype(np.float32)
```

[309]:
```
class PKLDataset(Dataset):
    def __init__(self, path, transform=None):
        with open(path, "rb") as f:
            data = pickle.load(f)

        self.images = data["images"]        # shape: (N, 28, 28, 3)
        self.labels = data["labels"].reshape(-1)   # shape: (N,) instead of␣
 ↪(N,1)
        self.transform = transform

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

    def __getitem__(self, idx):
        img = self.images[idx]              # numpy array (28,28,3)
        label = int(self.labels[idx])       # convert to Python int

        # Convert to tensor and permute to (C, H, W)
        img = torch.tensor(img, dtype=torch.float32).permute(2, 0, 1) / 255.0


        if self.transform:
            img = self.transform(img)

        return img, label
```

[310]:
```
import pickle

with open("ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl", "rb")␣
 ↪as f:
    data = pickle.load(f)

print(type(data))
print(len(data) if hasattr(data, "__len__") else "no len")
print(data)
```

```
<class 'dict'>
2
{'images': array([[[[ 6,  4,  0],
         [ 9,  5,  0],
         [ 8,  4,  0],
```

```
  …,
  [ 9,  6,  0],
  [ 9,  6,  0],
  [ 7,  4,  0]],

 [[11,  6,  0],
  [ 4,  4,  0],
  [ 3,  3,  0],
  …,
  [ 9,  6,  0],
  [ 6,  4,  0],
  [ 4,  2,  0]],

 [[11,  6,  0],
  [ 4,  4,  0],
  [ 3,  3,  0],
  …,
  [ 6,  4,  0],
  [ 6,  4,  0],
  [ 4,  2,  0]],

 …,

 [[ 1,  1,  0],
  [ 0,  0,  0],
  [ 0,  0,  1],
  …,
  [ 5,  4,  0],
  [ 6,  5,  0],
  [ 6,  5,  0]],

 [[ 3,  1,  1],
  [ 0,  0,  0],
  [ 0,  0,  1],
  …,
  [ 6,  5,  0],
  [ 6,  5,  0],
  [ 7,  6,  0]],

 [[10,  2,  2],
  [ 0,  0,  1],
  [ 0,  0,  1],
  …,
  [ 6,  5,  0],
  [ 7,  6,  0],
  [ 7,  6,  0]]],
```

```
[[[11,  9,  0],
  [ 9,  7,  0],
  [ 9,  7,  0],
  …,
  [ 0,  0,  1],
  [ 0,  0,  1],
  [ 0,  0,  1]],

 [[12,  9,  0],
  [11,  7,  0],
  [ 9,  6,  0],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 0,  0,  1]],

 [[ 9,  7,  0],
  [12,  8,  0],
  [10,  6,  0],
  …,
  [ 0,  0,  1],
  [ 0,  0,  1],
  [ 0,  0,  0]],

 …,

 [[ 0,  0,  3],
  [ 0,  0,  3],
  [ 0,  0,  4],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 0,  0,  0]],

 [[ 0,  0,  2],
  [ 0,  0,  2],
  [ 0,  0,  5],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 1,  0,  0]],

 [[ 0,  0,  1],
  [ 0,  0,  2],
  [ 0,  0,  4],
  …,
  [ 0,  0,  1],
  [ 1,  0,  0],
```

```
  [ 1,  0,  0]]],


 [[[18, 12,  0],
   [12,  9,  0],
   [17, 11,  0],
   …,
   [ 0,  0,  3],
   [ 5,  0,  2],
   [ 5,  0,  2]],

  [[15, 11,  0],
   [10,  9,  0],
   [10,  8,  0],
   …,
   [ 2,  0,  2],
   [ 7,  0,  1],
   [ 8,  0,  1]],

  [[17, 10,  0],
   [ 7,  6,  0],
   [ 4,  4,  0],
   …,
   [ 0,  0,  2],
   [ 5,  0,  1],
   [ 8,  0,  1]],

  …,

  [[ 2,  0,  0],
   [ 1,  0,  0],
   [ 0,  0,  0],
   …,
   [ 0,  0,  1],
   [ 0,  0,  0],
   [ 0,  0,  1]],

  [[ 2,  0,  0],
   [ 3,  1,  0],
   [ 0,  0,  0],
   …,
   [ 0,  0,  0],
   [ 0,  0,  1],
   [ 1,  0,  0]],

  [[ 3,  0,  0],
   [ 2,  0,  0],
   [ 2,  1,  0],
```

```
  …,
   [ 0,  0,  1],
   [ 1,  0,  0],
   [ 1,  0,  0]]],


…,


[[[56,  8, 20],
   [19,  0, 11],
   [ 0,  0,  1],
   …,
   [ 6,  3,  0],
   [11,  7,  0],
   [16,  8,  0]],

  [[19,  0, 11],
   [ 5,  0,  7],
   [ 0,  0,  3],
   …,
   [ 5,  3,  0],
   [ 5,  3,  0],
   [ 8,  4,  0]],

  [[ 0,  0,  0],
   [ 0,  0,  2],
   [ 1,  0,  6],
   …,
   [ 7,  4,  0],
   [ 5,  3,  0],
   [ 4,  2,  0]],

  …,

  [[ 4,  3,  0],
   [ 4,  2,  0],
   [ 4,  2,  0],
   …,
   [ 0,  0,  1],
   [ 1,  0,  0],
   [ 0,  0,  0]],

  [[ 1,  1,  0],
   [ 1,  1,  0],
   [ 4,  2,  0],
   …,
   [ 0,  0,  0],
```

```
  [ 3,  1,  0],
  [ 1,  1,  0]],

 [[ 1,  1,  0],
  [ 1,  1,  0],
  [ 0,  0,  0],
  …,
  [ 1,  0,  0],
  [ 3,  2,  0],
  [ 3,  3,  0]]],


[[[ 9,  6,  0],
  [ 8,  6,  0],
  [ 6,  4,  0],
  …,
  [ 3,  2,  0],
  [ 3,  2,  0],
  [ 0,  0,  0]],

 [[ 6,  6,  0],
  [ 4,  4,  0],
  [ 6,  4,  0],
  …,
  [ 3,  2,  0],
  [ 1,  0,  0],
  [ 2,  0,  0]],

 [[ 4,  4,  0],
  [ 6,  4,  0],
  [ 6,  4,  0],
  …,
  [ 1,  0,  0],
  [ 2,  0,  0],
  [ 3,  0,  0]],

 …,

 [[ 3,  3,  0],
  [ 3,  3,  0],
  [ 3,  1,  0],
  …,
  [ 0,  0,  1],
  [ 0,  0,  0],
  [ 0,  0,  0]],

 [[ 3,  3,  0],
  [ 3,  3,  0],
```

```
  [ 4,  2,  0],
  …,
  [ 2,  1,  0],
  [ 2,  1,  0],
  [ 1,  1,  0]],

 [[ 3,  3,  0],
  [ 3,  3,  0],
  [ 4,  2,  0],
  …,
  [ 2,  1,  0],
  [ 2,  1,  0],
  [ 1,  1,  0]]],


[[[ 6,  3,  0],
  [ 3,  2,  0],
  [ 2,  0,  2],
  …,
  [ 7,  6,  0],
  [ 7,  6,  0],
  [ 6,  6,  0]],

 [[ 4,  2,  0],
  [ 1,  0,  1],
  [ 2,  0,  2],
  …,
  [ 8,  5,  0],
  [ 7,  5,  0],
  [ 6,  4,  0]],

 [[ 2,  0,  0],
  [ 0,  0,  1],
  [ 0,  0,  3],
  …,
  [ 5,  3,  0],
  [ 7,  4,  0],
  [ 7,  4,  0]],

 …,

 [[ 4,  1,  0],
  [ 2,  0,  0],
  [ 1,  0,  0],
  …,
  [ 4,  2,  0],
  [ 6,  4,  0],
  [ 6,  4,  0]],
```

```
        [[ 7,  3,  0],
         [ 7,  4,  0],
         [ 6,  2,  0],
         …,
         [ 6,  4,  0],
         [ 7,  4,  0],
         [ 7,  4,  0]],

        [[11,  6,  0],
         [10,  5,  0],
         [12,  5,  0],
         …,
         [ 7,  4,  0],
         [ 7,  4,  0],
         [ 7,  5,  0]]]], shape=(1080, 28, 28, 3), dtype=uint8), 'labels':
array([[0],
       [0],
       [0],
       …,
       [2],
       [2],
       [3]], shape=(1080, 1), dtype=uint8)}
```

[311]:

```
from sklearn.model_selection import train_test_split

dataset = PKLDataset("ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
 ↪pkl")

labels = dataset.labels
indices = np.arange(len(dataset))

train_idx, valid_idx = train_test_split(
    indices,
    test_size=0.2,
    random_state=42,
    stratify=labels
)

from torch.utils.data import Subset

train_data = Subset(dataset, train_idx)
valid_data = Subset(dataset, valid_idx)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_data, batch_size=64)
```

[312]:
```python
import numpy as np

labels = dataset.labels
classes, counts = np.unique(labels, return_counts=True)

weights = 1.0 / counts
weights = weights / weights.sum()  # normalisation

class_weights = torch.tensor(weights, dtype=torch.float32, device=device)

loss_fn = nn.CrossEntropyLoss(weight=class_weights)

test_loss_fn = nn.CrossEntropyLoss(reduction='sum')

# spot to save your learning curves, and potentially checkpoint your models
savedir = 'results'
if not os.path.exists(savedir):
    os.makedirs(savedir)
```

[313]:
```python
def train(model, train_loader, optimizer, epoch):
    """Perform one epoch of training."""
    model.train()

    for batch_idx, (inputs, target) in enumerate(train_loader):
        inputs, target = inputs.to(device), target.to(device)

        # 1) Reset gradients
        optimizer.zero_grad()

        # 2) Forward pass
        output = model(inputs)

        # 3) Compute loss
        loss = loss_fn(output, target)

        # 4) Backpropagation
        loss.backward()

        # 5) Update weights
        optimizer.step()

        # Logging
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch,
                batch_idx * len(inputs),
                len(train_loader.dataset),
```

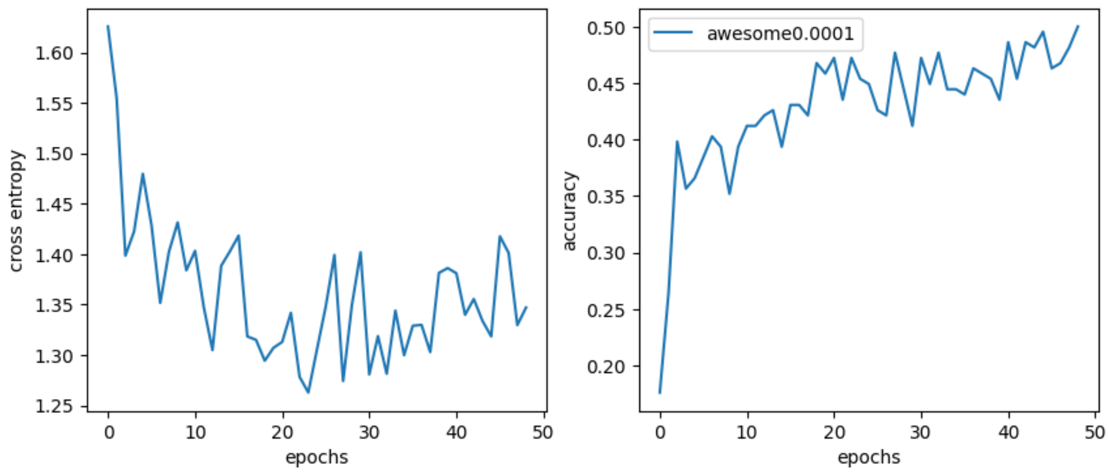
```
                100. * batch_idx / len(train_loader),
                loss.item()
            ))
```

[314]:
```
def test(model, test_loader):
    """Evaluate the model by doing one pass over a dataset"""
    model.eval()

    test_loss = 0   # total loss over test set
    correct = 0     # total number of correct test predictions
    test_size = 0   # number of test samples used

    with torch.no_grad():  # no backprop, faster evaluation
        for inputs, target in test_loader:
            inputs, target = inputs.to(device), target.to(device)

            # Forward pass
            output = model(inputs)

            # Accumulate loss (sum, not mean)
            loss = test_loss_fn(output, target)  # already reduction='sum'
            test_loss += loss.item()

            # Predictions
            pred = output.argmax(dim=1)  # index of highest logit
            correct += (pred == target).sum().item()

            # Keep track of sample count
            test_size += target.size(0)

    # Final metrics
    test_loss /= test_size
    accuracy = correct / test_size

    print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, test_size, 100. * accuracy))

    return test_loss, accuracy
```

[315]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

for filename in os.listdir(savedir):
    if filename.endswith('.pkl'):
        with open(os.path.join(savedir, filename),'rb') as fin:
            results = pickle.load(fin)
            ax1.plot(results['loss'])
            ax1.set_ylabel('cross entropy')
```

```
            ax1.set_xlabel('epochs')

            ax2.plot(results['accuracy'], label = filename[:-4])
            ax2.set_ylabel('accuracy')
            ax2.set_xlabel('epochs')

plt.legend()
```

[315]: <matplotlib.legend.Legend at 0x138f85450>



[316]:
```
class CNNNet(nn.Module):

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

        #block1
        self.conv1 = nn.Conv2d(3,32,3,padding = 1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32,32,3,padding = 1)
        self.bn2 = nn.BatchNorm2d(32)

        #block2
        self.conv3 = nn.Conv2d(32,64,3,padding = 1)
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64,64,3,padding = 1)
        self.bn4 = nn.BatchNorm2d(64)

        #fully connected layer for readou
        self.fc1 = nn.Linear(64*7*7,256)
        self.bn5 = nn.BatchNorm1d(256)
```

```
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(256,5)

    def forward(self, x):
      x = F.relu(self.bn1(self.conv1(x)))
      x = F.relu(self.bn2(self.conv2(x)))
      x = F.max_pool2d(x,2)
      x = F.relu(self.bn3(self.conv3(x)))
      x = F.relu(self.bn4(self.conv4(x)))
      x = F.max_pool2d(x,2)

      x = x.view(x.size(0),-1)
      x = F.relu(self.bn5(self.fc1(x)))
      x = self.dropout(x)
      x = self.fc2(x)
      return x
```

[319]:

```
subset = Subset(train_data, list(range(50)))
subset_loader = DataLoader(subset, batch_size=10, shuffle=True)
model = CNNNet().to(device)
lr = 0.0001
optimizer = optim.Adam(model.parameters(), lr=lr)

for epoch in range(20):
    train(model, subset_loader, optimizer, epoch)
    loss, acc = test(model, subset_loader)
    print(loss, acc)
```

```
Train Epoch: 0 [0/50 (0%)]      Loss: 1.679744
Test set: Average loss: 1.5890, Accuracy: 11/50 (22%)

1.589010944366455 0.22
Train Epoch: 1 [0/50 (0%)]      Loss: 1.447933
Test set: Average loss: 1.5950, Accuracy: 11/50 (22%)

1.5950431632995605 0.22
Train Epoch: 2 [0/50 (0%)]      Loss: 1.448872
Test set: Average loss: 1.6099, Accuracy: 11/50 (22%)

1.6098548126220704 0.22
Train Epoch: 3 [0/50 (0%)]      Loss: 1.117808
Test set: Average loss: 1.6205, Accuracy: 11/50 (22%)

1.6204501342773439 0.22
Train Epoch: 4 [0/50 (0%)]      Loss: 0.863511
Test set: Average loss: 1.6052, Accuracy: 12/50 (24%)

1.6052461624145509 0.24
```

```
Train Epoch: 5 [0/50 (0%)]      Loss: 0.862253
Test set: Average loss: 1.5675, Accuracy: 14/50 (28%)

1.5675125312805176 0.28
Train Epoch: 6 [0/50 (0%)]      Loss: 0.901393
Test set: Average loss: 1.4751, Accuracy: 23/50 (46%)

1.4750551414489745 0.46
Train Epoch: 7 [0/50 (0%)]      Loss: 0.837255
Test set: Average loss: 1.3419, Accuracy: 29/50 (58%)

1.3418502044677734 0.58
Train Epoch: 8 [0/50 (0%)]      Loss: 0.805337
Test set: Average loss: 1.1806, Accuracy: 34/50 (68%)

1.1806473541259765 0.68
Train Epoch: 9 [0/50 (0%)]      Loss: 0.365438
Test set: Average loss: 0.9891, Accuracy: 41/50 (82%)

0.989075927734375 0.82
Train Epoch: 10 [0/50 (0%)]     Loss: 0.517152
Test set: Average loss: 0.7992, Accuracy: 45/50 (90%)

0.7992279815673828 0.9
Train Epoch: 11 [0/50 (0%)]     Loss: 0.247045
Test set: Average loss: 0.6466, Accuracy: 46/50 (92%)

0.6465871906280518 0.92
Train Epoch: 12 [0/50 (0%)]     Loss: 0.453022
Test set: Average loss: 0.5364, Accuracy: 48/50 (96%)

0.5363518619537353 0.96
Train Epoch: 13 [0/50 (0%)]     Loss: 0.363443
Test set: Average loss: 0.4607, Accuracy: 49/50 (98%)

0.46067741870880125 0.98
Train Epoch: 14 [0/50 (0%)]     Loss: 0.793414
Test set: Average loss: 0.4081, Accuracy: 49/50 (98%)

0.4080589437484741 0.98
Train Epoch: 15 [0/50 (0%)]     Loss: 0.415299
Test set: Average loss: 0.3853, Accuracy: 48/50 (96%)

0.3852595043182373 0.96
Train Epoch: 16 [0/50 (0%)]     Loss: 0.393859
Test set: Average loss: 0.3632, Accuracy: 48/50 (96%)

0.36317144393920897 0.96
```

```
Train Epoch: 17 [0/50 (0%)]     Loss: 0.342084
Test set: Average loss: 0.3359, Accuracy: 49/50 (98%)

0.3359062099456787 0.98
Train Epoch: 18 [0/50 (0%)]     Loss: 0.288677
Test set: Average loss: 0.2998, Accuracy: 50/50 (100%)

0.29979662895202636 1.0
Train Epoch: 19 [0/50 (0%)]     Loss: 0.314176
Test set: Average loss: 0.2781, Accuracy: 50/50 (100%)

0.2780665445327759 1.0
```

[ ]:
```python
# TRAINING
model = CNNNet().to(device)

lr = 0.0001
optimizer = optim.Adam(model.parameters(), lr=lr)

results = {'name':'awesome', 'lr': lr, 'loss': [], 'accuracy':[]}
savefile = os.path.join(savedir, results['name']+str(results['lr'])+'.pkl' )

for epoch in range(1, 50):
    train(model, train_loader, optimizer, epoch)
    loss, acc = test(model, valid_loader)

    # save results
    results['loss'].append(loss)
    results['accuracy'].append(acc)
    with open(savefile, 'wb') as fout:
        pickle.dump(results, fout)
```

```
Train Epoch: 1 [0/864 (0%)]     Loss: 1.782380
Train Epoch: 1 [320/864 (37%)]  Loss: 1.583471
Train Epoch: 1 [640/864 (74%)]  Loss: 1.656232
Test set: Average loss: 1.6642, Accuracy: 26/216 (12%)

Train Epoch: 2 [0/864 (0%)]     Loss: 1.463709
Train Epoch: 2 [320/864 (37%)]  Loss: 1.602349
Train Epoch: 2 [640/864 (74%)]  Loss: 1.498432
Test set: Average loss: 1.5990, Accuracy: 53/216 (25%)

Train Epoch: 3 [0/864 (0%)]     Loss: 1.506461
Train Epoch: 3 [320/864 (37%)]  Loss: 1.437162
Train Epoch: 3 [640/864 (74%)]  Loss: 1.404151
Test set: Average loss: 1.4472, Accuracy: 75/216 (35%)

Train Epoch: 4 [0/864 (0%)]     Loss: 1.308002
```

```
Train Epoch: 4 [320/864 (37%)]  Loss: 1.509690
Train Epoch: 4 [640/864 (74%)]  Loss: 1.321391
Test set: Average loss: 1.4004, Accuracy: 83/216 (38%)

Train Epoch: 5 [0/864 (0%)]     Loss: 1.361703
Train Epoch: 5 [320/864 (37%)]  Loss: 1.574397
Train Epoch: 5 [640/864 (74%)]  Loss: 1.684576
Test set: Average loss: 1.4007, Accuracy: 82/216 (38%)

Train Epoch: 6 [0/864 (0%)]     Loss: 1.281565
Train Epoch: 6 [320/864 (37%)]  Loss: 1.547100
Train Epoch: 6 [640/864 (74%)]  Loss: 1.471929
Test set: Average loss: 1.3742, Accuracy: 87/216 (40%)

Train Epoch: 7 [0/864 (0%)]     Loss: 1.610609
Train Epoch: 7 [320/864 (37%)]  Loss: 1.346078
Train Epoch: 7 [640/864 (74%)]  Loss: 1.310871
Test set: Average loss: 1.4498, Accuracy: 79/216 (37%)

Train Epoch: 8 [0/864 (0%)]     Loss: 1.394938
Train Epoch: 8 [320/864 (37%)]  Loss: 1.342161
Train Epoch: 8 [640/864 (74%)]  Loss: 1.191340
Test set: Average loss: 1.3669, Accuracy: 87/216 (40%)

Train Epoch: 9 [0/864 (0%)]     Loss: 1.251976
Train Epoch: 9 [320/864 (37%)]  Loss: 1.267059
Train Epoch: 9 [640/864 (74%)]  Loss: 1.287991
Test set: Average loss: 1.3759, Accuracy: 99/216 (46%)

Train Epoch: 10 [0/864 (0%)]    Loss: 1.356218
Train Epoch: 10 [320/864 (37%)] Loss: 1.444301
Train Epoch: 10 [640/864 (74%)] Loss: 1.438623
Test set: Average loss: 1.4142, Accuracy: 89/216 (41%)

Train Epoch: 11 [0/864 (0%)]    Loss: 1.120535
Train Epoch: 11 [320/864 (37%)] Loss: 1.346429
Train Epoch: 11 [640/864 (74%)] Loss: 1.307619
Test set: Average loss: 1.3955, Accuracy: 90/216 (42%)

Train Epoch: 12 [0/864 (0%)]    Loss: 1.331817
Train Epoch: 12 [320/864 (37%)] Loss: 1.464857
Train Epoch: 12 [640/864 (74%)] Loss: 1.166912
Test set: Average loss: 1.4305, Accuracy: 85/216 (39%)

Train Epoch: 13 [0/864 (0%)]    Loss: 1.317523
Train Epoch: 13 [320/864 (37%)] Loss: 1.150822
Train Epoch: 13 [640/864 (74%)] Loss: 1.137262
Test set: Average loss: 1.3658, Accuracy: 89/216 (41%)
```

```
Train Epoch: 14 [0/864 (0%)]    Loss: 1.263249
Train Epoch: 14 [320/864 (37%)] Loss: 1.230830
Train Epoch: 14 [640/864 (74%)] Loss: 1.369614
Test set: Average loss: 1.4082, Accuracy: 89/216 (41%)

Train Epoch: 15 [0/864 (0%)]    Loss: 1.116665
Train Epoch: 15 [320/864 (37%)] Loss: 1.396810
Train Epoch: 15 [640/864 (74%)] Loss: 1.440533
Test set: Average loss: 1.3493, Accuracy: 93/216 (43%)

Train Epoch: 16 [0/864 (0%)]    Loss: 1.181443
Train Epoch: 16 [320/864 (37%)] Loss: 1.140962
Train Epoch: 16 [640/864 (74%)] Loss: 1.228503
Test set: Average loss: 1.3151, Accuracy: 91/216 (42%)

Train Epoch: 17 [0/864 (0%)]    Loss: 1.047197
Train Epoch: 17 [320/864 (37%)] Loss: 1.319429
Train Epoch: 17 [640/864 (74%)] Loss: 1.153058
Test set: Average loss: 1.3305, Accuracy: 93/216 (43%)

Train Epoch: 18 [0/864 (0%)]    Loss: 1.241787
Train Epoch: 18 [320/864 (37%)] Loss: 1.292457
Train Epoch: 18 [640/864 (74%)] Loss: 1.164752
Test set: Average loss: 1.3246, Accuracy: 88/216 (41%)

Train Epoch: 19 [0/864 (0%)]    Loss: 0.979050
Train Epoch: 19 [320/864 (37%)] Loss: 1.026454
Train Epoch: 19 [640/864 (74%)] Loss: 1.172203
Test set: Average loss: 1.2981, Accuracy: 94/216 (44%)

Train Epoch: 20 [0/864 (0%)]    Loss: 0.846902
Train Epoch: 20 [320/864 (37%)] Loss: 1.213814
Train Epoch: 20 [640/864 (74%)] Loss: 0.893829
Test set: Average loss: 1.3352, Accuracy: 88/216 (41%)

Train Epoch: 21 [0/864 (0%)]    Loss: 1.024848
Train Epoch: 21 [320/864 (37%)] Loss: 1.160144
Train Epoch: 21 [640/864 (74%)] Loss: 1.101253
Test set: Average loss: 1.3436, Accuracy: 88/216 (41%)

Train Epoch: 22 [0/864 (0%)]    Loss: 0.811015
Train Epoch: 22 [320/864 (37%)] Loss: 1.078108
Train Epoch: 22 [640/864 (74%)] Loss: 0.858267
Test set: Average loss: 1.3396, Accuracy: 102/216 (47%)

Train Epoch: 23 [0/864 (0%)]    Loss: 0.909721
Train Epoch: 23 [320/864 (37%)] Loss: 0.994284
```

```
Train Epoch: 23 [640/864 (74%)] Loss: 0.732774
Test set: Average loss: 1.3089, Accuracy: 92/216 (43%)

Train Epoch: 24 [0/864 (0%)]    Loss: 0.918355
Train Epoch: 24 [320/864 (37%)] Loss: 1.070239
Train Epoch: 24 [640/864 (74%)] Loss: 1.022037
Test set: Average loss: 1.3410, Accuracy: 90/216 (42%)

Train Epoch: 25 [0/864 (0%)]    Loss: 0.959832
Train Epoch: 25 [320/864 (37%)] Loss: 1.076749
Train Epoch: 25 [640/864 (74%)] Loss: 1.081821
Test set: Average loss: 1.2732, Accuracy: 103/216 (48%)

Train Epoch: 26 [0/864 (0%)]    Loss: 0.733921
Train Epoch: 26 [320/864 (37%)] Loss: 1.212125
Train Epoch: 26 [640/864 (74%)] Loss: 0.920557
Test set: Average loss: 1.3264, Accuracy: 97/216 (45%)

Train Epoch: 27 [0/864 (0%)]    Loss: 1.042933
Train Epoch: 27 [320/864 (37%)] Loss: 1.253436
Train Epoch: 27 [640/864 (74%)] Loss: 0.715091
Test set: Average loss: 1.3251, Accuracy: 92/216 (43%)

Train Epoch: 28 [0/864 (0%)]    Loss: 0.942986
Train Epoch: 28 [320/864 (37%)] Loss: 1.044843
Train Epoch: 28 [640/864 (74%)] Loss: 0.861113
Test set: Average loss: 1.2775, Accuracy: 101/216 (47%)

Train Epoch: 29 [0/864 (0%)]    Loss: 0.980809
Train Epoch: 29 [320/864 (37%)] Loss: 0.561878
Train Epoch: 29 [640/864 (74%)] Loss: 0.900598
Test set: Average loss: 1.4219, Accuracy: 95/216 (44%)

Train Epoch: 30 [0/864 (0%)]    Loss: 0.792306
Train Epoch: 30 [320/864 (37%)] Loss: 0.877965
Train Epoch: 30 [640/864 (74%)] Loss: 0.766107
Test set: Average loss: 1.2880, Accuracy: 99/216 (46%)

Train Epoch: 31 [0/864 (0%)]    Loss: 0.794105
Train Epoch: 31 [320/864 (37%)] Loss: 0.780075
Train Epoch: 31 [640/864 (74%)] Loss: 0.638629
Test set: Average loss: 1.3212, Accuracy: 96/216 (44%)

Train Epoch: 32 [0/864 (0%)]    Loss: 0.595142
Train Epoch: 32 [320/864 (37%)] Loss: 0.700272
Train Epoch: 32 [640/864 (74%)] Loss: 0.634284
Test set: Average loss: 1.3125, Accuracy: 99/216 (46%)
```

```
Train Epoch: 33 [0/864 (0%)]    Loss: 0.738566
Train Epoch: 33 [320/864 (37%)] Loss: 0.908737
Train Epoch: 33 [640/864 (74%)] Loss: 0.779309
Test set: Average loss: 1.2546, Accuracy: 100/216 (46%)

Train Epoch: 34 [0/864 (0%)]    Loss: 0.850645
Train Epoch: 34 [320/864 (37%)] Loss: 0.685012
Train Epoch: 34 [640/864 (74%)] Loss: 0.543319
Test set: Average loss: 1.3981, Accuracy: 100/216 (46%)

Train Epoch: 35 [0/864 (0%)]    Loss: 0.622933
Train Epoch: 35 [320/864 (37%)] Loss: 0.680022
Train Epoch: 35 [640/864 (74%)] Loss: 0.811215
Test set: Average loss: 1.2980, Accuracy: 98/216 (45%)

Train Epoch: 36 [0/864 (0%)]    Loss: 0.757709
Train Epoch: 36 [320/864 (37%)] Loss: 0.748167
Train Epoch: 36 [640/864 (74%)] Loss: 0.711941
Test set: Average loss: 1.3192, Accuracy: 95/216 (44%)

Train Epoch: 37 [0/864 (0%)]    Loss: 0.849804
Train Epoch: 37 [320/864 (37%)] Loss: 0.724919
Train Epoch: 37 [640/864 (74%)] Loss: 0.474969
Test set: Average loss: 1.3149, Accuracy: 97/216 (45%)

Train Epoch: 38 [0/864 (0%)]    Loss: 0.486010
Train Epoch: 38 [320/864 (37%)] Loss: 0.450690
Train Epoch: 38 [640/864 (74%)] Loss: 0.594306
Test set: Average loss: 1.3089, Accuracy: 98/216 (45%)

Train Epoch: 39 [0/864 (0%)]    Loss: 0.457597
Train Epoch: 39 [320/864 (37%)] Loss: 0.698482
Train Epoch: 39 [640/864 (74%)] Loss: 0.490885
Test set: Average loss: 1.3778, Accuracy: 101/216 (47%)

Train Epoch: 40 [0/864 (0%)]    Loss: 0.647184
Train Epoch: 40 [320/864 (37%)] Loss: 0.410051
Train Epoch: 40 [640/864 (74%)] Loss: 0.755910
Test set: Average loss: 1.3432, Accuracy: 96/216 (44%)

Train Epoch: 41 [0/864 (0%)]    Loss: 0.391717
Train Epoch: 41 [320/864 (37%)] Loss: 0.592973
Train Epoch: 41 [640/864 (74%)] Loss: 0.555214
Test set: Average loss: 1.3061, Accuracy: 99/216 (46%)

Train Epoch: 42 [0/864 (0%)]    Loss: 0.426634
Train Epoch: 42 [320/864 (37%)] Loss: 0.579080
Train Epoch: 42 [640/864 (74%)] Loss: 0.736073
```

```
Test set: Average loss: 1.3414, Accuracy: 95/216 (44%)

Train Epoch: 43 [0/864 (0%)]    Loss: 0.624328
Train Epoch: 43 [320/864 (37%)] Loss: 0.523391
Train Epoch: 43 [640/864 (74%)] Loss: 0.661240
Test set: Average loss: 1.3553, Accuracy: 95/216 (44%)

Train Epoch: 44 [0/864 (0%)]    Loss: 0.460886
Train Epoch: 44 [320/864 (37%)] Loss: 0.396313
Train Epoch: 44 [640/864 (74%)] Loss: 0.554027
Test set: Average loss: 1.3688, Accuracy: 93/216 (43%)

Train Epoch: 45 [0/864 (0%)]    Loss: 0.665699
Train Epoch: 45 [320/864 (37%)] Loss: 0.475768
Train Epoch: 45 [640/864 (74%)] Loss: 0.509555
Test set: Average loss: 1.3367, Accuracy: 97/216 (45%)

Train Epoch: 46 [0/864 (0%)]    Loss: 0.565105
Train Epoch: 46 [320/864 (37%)] Loss: 0.424564
Train Epoch: 46 [640/864 (74%)] Loss: 0.330786
Test set: Average loss: 1.3625, Accuracy: 103/216 (48%)

Train Epoch: 47 [0/864 (0%)]    Loss: 0.407636
Train Epoch: 47 [320/864 (37%)] Loss: 0.553786
Train Epoch: 47 [640/864 (74%)] Loss: 0.618660
Test set: Average loss: 1.3915, Accuracy: 106/216 (49%)

Train Epoch: 48 [0/864 (0%)]    Loss: 0.606045
Train Epoch: 48 [320/864 (37%)] Loss: 0.422372
Train Epoch: 48 [640/864 (74%)] Loss: 0.392674
Test set: Average loss: 1.3827, Accuracy: 97/216 (45%)

Train Epoch: 49 [0/864 (0%)]    Loss: 0.456831
Train Epoch: 49 [320/864 (37%)] Loss: 0.488723
Train Epoch: 49 [640/864 (74%)] Loss: 0.337330
Test set: Average loss: 1.4333, Accuracy: 100/216 (46%)
```