todebug3

November 27, 2025

[268]:
```
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)
```

[269]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
else:
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[270]:
```
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
```

[271]:
```
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)}
```

[272]:
```python
from sklearn.model_selection import train_test_split
from torchvision import transforms
from torch.utils.data import Subset



# Charger le dataset SANS transformation d'abord
dataset = PKLDataset(
                "ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl",
)

loader = DataLoader(dataset, batch_size=64, shuffle=False)

d_mean = torch.zeros(3)
d_std = torch.zeros(3)
nb_samples = 0.0

for images, _ in loader:
    batch_samples = images.size(0)

    d_mean += images.mean(dim=[0,2,3]) * batch_samples
    d_std += images.std(dim=[0,2,3]) * batch_samples
    nb_samples += batch_samples

d_mean /= nb_samples
d_std /= nb_samples

d_mean = d_mean.tolist()
d_std = d_std.tolist()

print("Mean:", d_mean)
print("Std:", d_std)
```

```
Mean: [0.21014535427093506, 0.005330359563231468, 0.2285669893026352]
Std: [0.18871904909610748, 0.01642582379281521, 0.16962255537509918]
```

[273]:
```python
# Définir les transformations
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5), # La rétine n'a pas de sens gauche/
 ↪droite
    transforms.RandomVerticalFlip(p=0.5),   # Ni de haut/bas strict
    transforms.RandomRotation(180),
    transforms.RandomAdjustSharpness(sharpness_factor=2, p=1.0),         # La␣
 ↪rotation est cruciale pour l'œil
```

```
    transforms.Normalize(mean=d_mean, std=d_std),
])

transform_val = transforms.Compose([
    transforms.Normalize(mean=d_mean, std=d_std),
])
```

[274]:
```
class TransformSubset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, idx):
        image, label = self.subset[idx]
        if self.transform:
            image = self.transform(image)
        return image, label

    def __len__(self):
        return len(self.subset)
```

[275]:
```
from torch.utils.data import WeightedRandomSampler
import torch
import numpy as np

# 1. Votre Split existant (inchangé)
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
)

# 2. Préparation du Sampler (NOUVEAU BLOC)
# On récupère uniquement les labels qui sont dans le set d'entrainement
y_train = labels[train_idx].reshape(-1) # reshape pour être sûr d'avoir (N,) et␣
 ↪pas (N,1)

# Compter combien d'exemples il y a par classe dans le train
class_counts = np.bincount(y_train)

# Calculer le poids de chaque classe (Inverse Frequency)
# Moins la classe est fréquente, plus le poids est grand
class_weights = 1. / class_counts
```

```
# Assigner un poids à chaque ÉCHANTILLON individuel du train
# Si l'image 1 est de classe 0, elle prend le poids de la classe 0, etc.
samples_weights = class_weights[y_train]
samples_weights = torch.from_numpy(samples_weights).double()

# Créer le sampler
sampler = WeightedRandomSampler(
    weights=samples_weights,
    num_samples=len(samples_weights),
    replacement=True # CRUCIAL : permet de re-piocher les images rares␣
 ↪plusieurs fois par epoch
)

# 3. Création des Subsets et Transforms (inchangé)
train_data = Subset(dataset, train_idx)
valid_data = Subset(dataset, valid_idx)

train_data = TransformSubset(train_data, transform=transform_train)
valid_data = TransformSubset(valid_data, transform=transform_val)

# 4. DataLoaders (MODIFIÉ)
train_loader = DataLoader(
    train_data,
    batch_size=64,
    sampler=sampler,  # <--- On ajoute le sampler ici
    shuffle=False     # <--- OBLIGATOIRE : shuffle doit être False quand on␣
 ↪utilise un sampler
)

# Le valid_loader reste classique (on ne veut pas de sampler pour la validation)
valid_loader = DataLoader(valid_data, batch_size=128, shuffle=False)
```

[276]:
```
import numpy as np

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

from sklearn.utils.class_weight import compute_class_weight

weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(labels),
    y=labels
)
class_weights = torch.tensor(weights, dtype=torch.float32).to(device)

loss_fn = nn.MSELoss()
```

```
test_loss_fn = nn.MSELoss(reduction='sum')

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

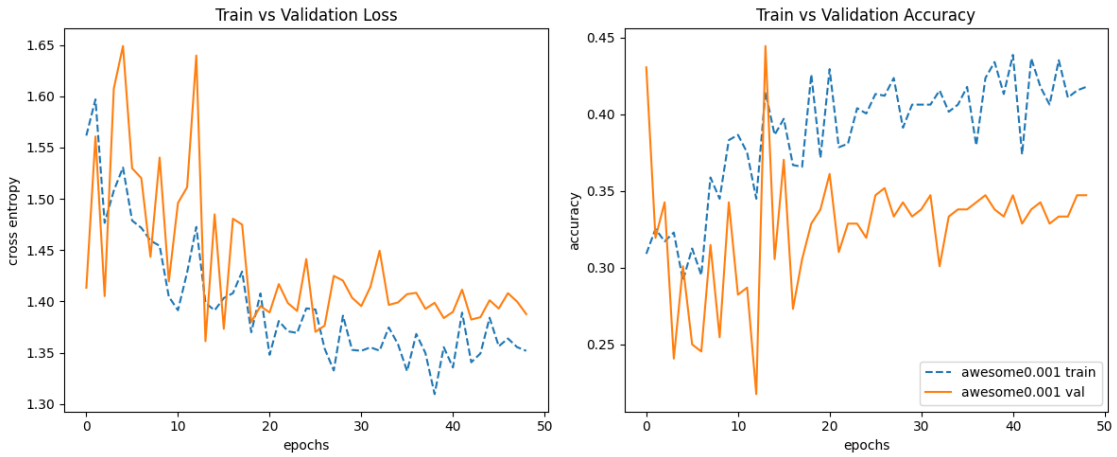
[277]:
```
def train(model, train_loader, optimizer, epoch):
    """Perform one epoch of training."""
    model.train()

    for batch_idx, (inputs, target) in enumerate(train_loader):
        inputs, target = inputs.to(device), target.float().view(-1, 1).
 ↪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()
            ))
```

[278]:
```
def test(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    test_size = 0

    with torch.no_grad():
```

```
        for inputs, target in test_loader:
            inputs = inputs.to(device)
            # Conversion pour la loss
            target_float = target.float().view(-1, 1).to(device)
            target = target.to(device) # On garde les entiers pour l'accuracy

            output = model(inputs)

            # Calcul de la loss (Somme des erreurs au carré)
            test_loss += test_loss_fn(output, target_float).item()

            # 3. Calcul de l'Accuracy pour la régression
            # On arrondit (round), on sature entre 0 et 4 (clamp)
            pred = torch.round(output).clamp(0, 4).long()

            # On compare la prédiction arrondie avec la cible originale (target
 ↪entier)
            # pred.view_as(target) assure que les dimensions collent
            correct += pred.flatten().eq(target).sum().item()

            test_size += target.size(0)

    test_loss /= test_size
    accuracy = correct / test_size
    print(f'Test set: Average loss: {test_loss:.4f}, Accuracy: {correct}/
 ↪{test_size} ({100.*accuracy:.0f}%)')
    return test_loss, accuracy
```

[279]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

for filename in os.listdir(savedir):
    if filename.endswith('.pkl'):
        with open(os.path.join(savedir, filename), 'rb') as fin:
            results = pickle.load(fin)

            label = filename[:-4]  # nom sans .pkl

            # ----- Courbes de LOSS -----
            ax1.plot(results['train_loss'], '--', label=f'{label} train')
            ax1.plot(results['val_loss'], '-',  label=f'{label} val')
            ax1.set_ylabel('cross entropy')
            ax1.set_xlabel('epochs')
            ax1.set_title('Train vs Validation Loss')

            # ----- Courbes d'ACCURACY -----
            ax2.plot(results['train_acc'], '--', label=f'{label} train')
            ax2.plot(results['val_acc'], '-',  label=f'{label} val')
```

```
            ax2.set_ylabel('accuracy')
            ax2.set_xlabel('epochs')
            ax2.set_title('Train vs Validation Accuracy')

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



[280]:
```
class CNNNet(nn.Module):
    def __init__(self, num_classes=5):
        super().__init__()
        self.act = nn.ReLU()
        self.drop = nn.Dropout(0.5) # Dropout fort pour éviter le par cœur vu␣
↪qu'on augmente les filtres

        # Bloc 1 : Extraction de features bas niveau (bords, contrastes)
        # On passe de 3 à 64 filtres directement pour capter plus de nuances de␣
↪couleurs
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        # Bloc 2 : Extraction mi-niveau
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(128)

        # Bloc 3 : Features complexes
        self.conv5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
```

```
        self.bn5 = nn.BatchNorm2d(256)

        # Classifieur
        # Après 3 max_pool (divisé par 2 trois fois): 28 -> 14 -> 7 -> 3
        self.flatten_dim = 256 * 3 * 3

        self.fc1 = nn.Linear(self.flatten_dim, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        # Bloc 1 (28x28)
        x = self.act(self.bn1(self.conv1(x)))
        x = self.act(self.bn2(self.conv2(x)))
        x = F.max_pool2d(x, 2) # -> 14x14

        # Bloc 2 (14x14)
        x = self.act(self.bn3(self.conv3(x)))
        x = self.act(self.bn4(self.conv4(x)))
        x = F.max_pool2d(x, 2) # -> 7x7

        # Bloc 3 (7x7)
        x = self.act(self.bn5(self.conv5(x)))
        x = F.max_pool2d(x, 2) # -> 3x3

        x = x.view(x.size(0), -1)
        x = self.drop(self.act(self.fc1(x)))
        x = self.fc2(x)
        return x
```

[281]:
```
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, weight_decay=1e-4)

for epoch in range(50):
    train(model, subset_loader, optimizer, epoch)
    loss, acc = test(model, subset_loader)
    print(loss, acc)
```

```
Train Epoch: 0 [0/50 (0%)]      Loss: 2.224972

/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/torch/nn/modules/loss.py:634: UserWarning: Using a target size
(torch.Size([10, 1])) that is different to the input size (torch.Size([10, 5])).
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)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[281], line 9
      7 for epoch in range(50):
      8     train(model, subset_loader, optimizer, epoch)
----> 9     loss, acc = test(model, subset_loader)
     10     print(loss, acc)

Cell In[278], line 25, in test(model, test_loader)
     21         pred = torch.round(output).clamp(0, 4).long()
     23         # On compare la prédiction arrondie avec la cible originale␣
 ↪(target entier)
     24         # pred.view_as(target) assure que les dimensions collent
---> 25         correct += pred.flatten().eq(target).sum().item()
     27         test_size += target.size(0)
     29 test_loss /= test_size

RuntimeError: The size of tensor a (50) must match the size of tensor b (10) at␣
 ↪non-singleton dimension 0
```

[ ]:
```python
# TRAINING
model = CNNNet().to(device)
lr=0.001
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=0.5, patience=4
)

results = {'name':'awesome', 'lr': lr, 'train_loss': [],
    'train_acc': [],
    'val_loss': [],
    'val_acc': []
    }
savefile = os.path.join(savedir, results['name']+str(results['lr'])+'.pkl' )

for epoch in range(1, 50):
    train(model, train_loader, optimizer, epoch)

    train_loss, train_acc = test(model, train_loader)
    val_loss,  val_acc  = test(model, valid_loader)

    scheduler.step(val_acc)

    results['train_loss'].append(train_loss)
    results['train_acc'].append(train_acc)
```

```
    results['val_loss'].append(val_loss)
    results['val_acc'].append(val_acc)

    with open(savefile, 'wb') as fout:
        pickle.dump(results, fout)
```

```
Train Epoch: 1 [0/864 (0%)]     Loss: 1.628957
Train Epoch: 1 [640/864 (71%)]  Loss: 1.597916
Test set: Average loss: 1.5617, Accuracy: 267/864 (31%)

Test set: Average loss: 1.4133, Accuracy: 93/216 (43%)

Train Epoch: 2 [0/864 (0%)]     Loss: 1.624724
Train Epoch: 2 [640/864 (71%)]  Loss: 1.703280
Test set: Average loss: 1.5970, Accuracy: 281/864 (33%)

Test set: Average loss: 1.5608, Accuracy: 69/216 (32%)

Train Epoch: 3 [0/864 (0%)]     Loss: 1.721866
Train Epoch: 3 [640/864 (71%)]  Loss: 1.597077
Test set: Average loss: 1.4766, Accuracy: 274/864 (32%)

Test set: Average loss: 1.4052, Accuracy: 74/216 (34%)

Train Epoch: 4 [0/864 (0%)]     Loss: 1.568465
Train Epoch: 4 [640/864 (71%)]  Loss: 1.488764
Test set: Average loss: 1.5080, Accuracy: 279/864 (32%)

Test set: Average loss: 1.6076, Accuracy: 52/216 (24%)

Train Epoch: 5 [0/864 (0%)]     Loss: 1.464779
Train Epoch: 5 [640/864 (71%)]  Loss: 1.470476
Test set: Average loss: 1.5306, Accuracy: 252/864 (29%)

Test set: Average loss: 1.6491, Accuracy: 65/216 (30%)

Train Epoch: 6 [0/864 (0%)]     Loss: 1.477125
Train Epoch: 6 [640/864 (71%)]  Loss: 1.499448
Test set: Average loss: 1.4788, Accuracy: 270/864 (31%)

Test set: Average loss: 1.5298, Accuracy: 54/216 (25%)

Train Epoch: 7 [0/864 (0%)]     Loss: 1.564198
Train Epoch: 7 [640/864 (71%)]  Loss: 1.661984
Test set: Average loss: 1.4717, Accuracy: 255/864 (30%)

Test set: Average loss: 1.5202, Accuracy: 53/216 (25%)
```

```
Train Epoch: 8 [0/864 (0%)]     Loss: 1.453879
Train Epoch: 8 [640/864 (71%)]  Loss: 1.443181
Test set: Average loss: 1.4593, Accuracy: 310/864 (36%)

Test set: Average loss: 1.4435, Accuracy: 68/216 (31%)

Train Epoch: 9 [0/864 (0%)]     Loss: 1.528321
Train Epoch: 9 [640/864 (71%)]  Loss: 1.503724
Test set: Average loss: 1.4542, Accuracy: 298/864 (34%)

Test set: Average loss: 1.5402, Accuracy: 55/216 (25%)

Train Epoch: 10 [0/864 (0%)]    Loss: 1.486253
Train Epoch: 10 [640/864 (71%)] Loss: 1.464042
Test set: Average loss: 1.4046, Accuracy: 331/864 (38%)

Test set: Average loss: 1.4195, Accuracy: 74/216 (34%)

Train Epoch: 11 [0/864 (0%)]    Loss: 1.410367
Train Epoch: 11 [640/864 (71%)] Loss: 1.457101
Test set: Average loss: 1.3914, Accuracy: 334/864 (39%)

Test set: Average loss: 1.4960, Accuracy: 61/216 (28%)

Train Epoch: 12 [0/864 (0%)]    Loss: 1.483926
Train Epoch: 12 [640/864 (71%)] Loss: 1.439698
Test set: Average loss: 1.4283, Accuracy: 324/864 (38%)

Test set: Average loss: 1.5114, Accuracy: 62/216 (29%)

Train Epoch: 13 [0/864 (0%)]    Loss: 1.363933
Train Epoch: 13 [640/864 (71%)] Loss: 1.533469
Test set: Average loss: 1.4727, Accuracy: 298/864 (34%)

Test set: Average loss: 1.6398, Accuracy: 47/216 (22%)

Train Epoch: 14 [0/864 (0%)]    Loss: 1.353062
Train Epoch: 14 [640/864 (71%)] Loss: 1.461062
Test set: Average loss: 1.3985, Accuracy: 358/864 (41%)

Test set: Average loss: 1.3612, Accuracy: 96/216 (44%)

Train Epoch: 15 [0/864 (0%)]    Loss: 1.437970
Train Epoch: 15 [640/864 (71%)] Loss: 1.436915
Test set: Average loss: 1.3913, Accuracy: 334/864 (39%)

Test set: Average loss: 1.4849, Accuracy: 66/216 (31%)
```

```
Train Epoch: 16 [0/864 (0%)]    Loss: 1.351175
Train Epoch: 16 [640/864 (71%)] Loss: 1.496634
Test set: Average loss: 1.4033, Accuracy: 343/864 (40%)

Test set: Average loss: 1.3733, Accuracy: 80/216 (37%)

Train Epoch: 17 [0/864 (0%)]    Loss: 1.426815
Train Epoch: 17 [640/864 (71%)] Loss: 1.349279
Test set: Average loss: 1.4081, Accuracy: 317/864 (37%)

Test set: Average loss: 1.4806, Accuracy: 59/216 (27%)

Train Epoch: 18 [0/864 (0%)]    Loss: 1.427200
Train Epoch: 18 [640/864 (71%)] Loss: 1.456698
Test set: Average loss: 1.4291, Accuracy: 316/864 (37%)

Test set: Average loss: 1.4749, Accuracy: 66/216 (31%)

Train Epoch: 19 [0/864 (0%)]    Loss: 1.537862
Train Epoch: 19 [640/864 (71%)] Loss: 1.432366
Test set: Average loss: 1.3699, Accuracy: 368/864 (43%)

Test set: Average loss: 1.3800, Accuracy: 71/216 (33%)

Train Epoch: 20 [0/864 (0%)]    Loss: 1.415628
Train Epoch: 20 [640/864 (71%)] Loss: 1.491726
Test set: Average loss: 1.4078, Accuracy: 321/864 (37%)

Test set: Average loss: 1.3954, Accuracy: 73/216 (34%)

Train Epoch: 21 [0/864 (0%)]    Loss: 1.295838
Train Epoch: 21 [640/864 (71%)] Loss: 1.391419
Test set: Average loss: 1.3479, Accuracy: 371/864 (43%)

Test set: Average loss: 1.3893, Accuracy: 78/216 (36%)

Train Epoch: 22 [0/864 (0%)]    Loss: 1.403609
Train Epoch: 22 [640/864 (71%)] Loss: 1.452295
Test set: Average loss: 1.3809, Accuracy: 327/864 (38%)

Test set: Average loss: 1.4169, Accuracy: 67/216 (31%)

Train Epoch: 23 [0/864 (0%)]    Loss: 1.323037
Train Epoch: 23 [640/864 (71%)] Loss: 1.464933
Test set: Average loss: 1.3708, Accuracy: 329/864 (38%)

Test set: Average loss: 1.3983, Accuracy: 71/216 (33%)
```

```
Train Epoch: 24 [0/864 (0%)]    Loss: 1.438230
Train Epoch: 24 [640/864 (71%)] Loss: 1.524597
Test set: Average loss: 1.3692, Accuracy: 349/864 (40%)

Test set: Average loss: 1.3907, Accuracy: 71/216 (33%)

Train Epoch: 25 [0/864 (0%)]    Loss: 1.308583
Train Epoch: 25 [640/864 (71%)] Loss: 1.230843
Test set: Average loss: 1.3932, Accuracy: 346/864 (40%)

Test set: Average loss: 1.4413, Accuracy: 69/216 (32%)

Train Epoch: 26 [0/864 (0%)]    Loss: 1.310491
Train Epoch: 26 [640/864 (71%)] Loss: 1.371282
Test set: Average loss: 1.3921, Accuracy: 357/864 (41%)

Test set: Average loss: 1.3703, Accuracy: 75/216 (35%)

Train Epoch: 27 [0/864 (0%)]    Loss: 1.502572
Train Epoch: 27 [640/864 (71%)] Loss: 1.516015
Test set: Average loss: 1.3540, Accuracy: 356/864 (41%)

Test set: Average loss: 1.3764, Accuracy: 76/216 (35%)

Train Epoch: 28 [0/864 (0%)]    Loss: 1.428820
Train Epoch: 28 [640/864 (71%)] Loss: 1.338486
Test set: Average loss: 1.3327, Accuracy: 366/864 (42%)

Test set: Average loss: 1.4249, Accuracy: 72/216 (33%)

Train Epoch: 29 [0/864 (0%)]    Loss: 1.534838
Train Epoch: 29 [640/864 (71%)] Loss: 1.281116
Test set: Average loss: 1.3861, Accuracy: 338/864 (39%)

Test set: Average loss: 1.4203, Accuracy: 74/216 (34%)

Train Epoch: 30 [0/864 (0%)]    Loss: 1.271923
Train Epoch: 30 [640/864 (71%)] Loss: 1.496226
Test set: Average loss: 1.3527, Accuracy: 351/864 (41%)

Test set: Average loss: 1.4036, Accuracy: 72/216 (33%)

Train Epoch: 31 [0/864 (0%)]    Loss: 1.308937
Train Epoch: 31 [640/864 (71%)] Loss: 1.267458
Test set: Average loss: 1.3518, Accuracy: 351/864 (41%)

Test set: Average loss: 1.3954, Accuracy: 73/216 (34%)
```

```
Train Epoch: 32 [0/864 (0%)]    Loss: 1.357844
Train Epoch: 32 [640/864 (71%)] Loss: 1.388824
Test set: Average loss: 1.3552, Accuracy: 351/864 (41%)

Test set: Average loss: 1.4142, Accuracy: 75/216 (35%)

Train Epoch: 33 [0/864 (0%)]    Loss: 1.261323
Train Epoch: 33 [640/864 (71%)] Loss: 1.411366
Test set: Average loss: 1.3519, Accuracy: 359/864 (42%)

Test set: Average loss: 1.4494, Accuracy: 65/216 (30%)

Train Epoch: 34 [0/864 (0%)]    Loss: 1.349386
Train Epoch: 34 [640/864 (71%)] Loss: 1.394161
Test set: Average loss: 1.3746, Accuracy: 347/864 (40%)

Test set: Average loss: 1.3966, Accuracy: 72/216 (33%)

Train Epoch: 35 [0/864 (0%)]    Loss: 1.423384
Train Epoch: 35 [640/864 (71%)] Loss: 1.499015
Test set: Average loss: 1.3584, Accuracy: 351/864 (41%)

Test set: Average loss: 1.3990, Accuracy: 73/216 (34%)

Train Epoch: 36 [0/864 (0%)]    Loss: 1.531783
Train Epoch: 36 [640/864 (71%)] Loss: 1.275522
Test set: Average loss: 1.3320, Accuracy: 361/864 (42%)

Test set: Average loss: 1.4071, Accuracy: 73/216 (34%)

Train Epoch: 37 [0/864 (0%)]    Loss: 1.326586
Train Epoch: 37 [640/864 (71%)] Loss: 1.394779
Test set: Average loss: 1.3683, Accuracy: 328/864 (38%)

Test set: Average loss: 1.4084, Accuracy: 74/216 (34%)

Train Epoch: 38 [0/864 (0%)]    Loss: 1.395474
Train Epoch: 38 [640/864 (71%)] Loss: 1.397625
Test set: Average loss: 1.3496, Accuracy: 366/864 (42%)

Test set: Average loss: 1.3927, Accuracy: 75/216 (35%)

Train Epoch: 39 [0/864 (0%)]    Loss: 1.378523
Train Epoch: 39 [640/864 (71%)] Loss: 1.279923
Test set: Average loss: 1.3096, Accuracy: 375/864 (43%)

Test set: Average loss: 1.3986, Accuracy: 73/216 (34%)
```

```
Train Epoch: 40 [0/864 (0%)]    Loss: 1.353968
Train Epoch: 40 [640/864 (71%)] Loss: 1.447603
Test set: Average loss: 1.3554, Accuracy: 357/864 (41%)

Test set: Average loss: 1.3838, Accuracy: 72/216 (33%)

Train Epoch: 41 [0/864 (0%)]    Loss: 1.430371
Train Epoch: 41 [640/864 (71%)] Loss: 1.390781
Test set: Average loss: 1.3355, Accuracy: 379/864 (44%)

Test set: Average loss: 1.3898, Accuracy: 75/216 (35%)

Train Epoch: 42 [0/864 (0%)]    Loss: 1.496687
Train Epoch: 42 [640/864 (71%)] Loss: 1.359934
Test set: Average loss: 1.3891, Accuracy: 323/864 (37%)

Test set: Average loss: 1.4115, Accuracy: 71/216 (33%)

Train Epoch: 43 [0/864 (0%)]    Loss: 1.316384
Train Epoch: 43 [640/864 (71%)] Loss: 1.374001
Test set: Average loss: 1.3406, Accuracy: 377/864 (44%)

Test set: Average loss: 1.3823, Accuracy: 73/216 (34%)

Train Epoch: 44 [0/864 (0%)]    Loss: 1.599025
Train Epoch: 44 [640/864 (71%)] Loss: 1.494026
Test set: Average loss: 1.3492, Accuracy: 361/864 (42%)

Test set: Average loss: 1.3846, Accuracy: 74/216 (34%)

Train Epoch: 45 [0/864 (0%)]    Loss: 1.417726
Train Epoch: 45 [640/864 (71%)] Loss: 1.294543
Test set: Average loss: 1.3840, Accuracy: 351/864 (41%)

Test set: Average loss: 1.4011, Accuracy: 71/216 (33%)

Train Epoch: 46 [0/864 (0%)]    Loss: 1.385288
Train Epoch: 46 [640/864 (71%)] Loss: 1.332602
Test set: Average loss: 1.3562, Accuracy: 376/864 (44%)

Test set: Average loss: 1.3931, Accuracy: 72/216 (33%)

Train Epoch: 47 [0/864 (0%)]    Loss: 1.397649
Train Epoch: 47 [640/864 (71%)] Loss: 1.531821
Test set: Average loss: 1.3638, Accuracy: 355/864 (41%)

Test set: Average loss: 1.4080, Accuracy: 72/216 (33%)
```

```
Train Epoch: 48 [0/864 (0%)]    Loss: 1.570987
Train Epoch: 48 [640/864 (71%)] Loss: 1.387809
Test set: Average loss: 1.3554, Accuracy: 359/864 (42%)

Test set: Average loss: 1.3998, Accuracy: 75/216 (35%)

Train Epoch: 49 [0/864 (0%)]    Loss: 1.364528
Train Epoch: 49 [640/864 (71%)] Loss: 1.393537
Test set: Average loss: 1.3518, Accuracy: 361/864 (42%)

Test set: Average loss: 1.3876, Accuracy: 75/216 (35%)
```