monCNN

November 26, 2025

[73]:
```python
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)
```

[74]:
```python
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
else:
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[75]:
```python
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)
```

```python
    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
```

[76]:
```python
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)}
```

[77]:
```
dataset = PKLDataset("ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
 ↪pkl")

train_size = int(0.8 * len(dataset))
valid_size = len(dataset) - train_size

train_data, valid_data = random_split(dataset, [train_size, valid_size])

batch_size = 32
batch_size_eval = 512

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

[78]:
```
loss_fn = nn.CrossEntropyLoss()
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)
```

[79]:
```
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()
            ))
```

[80]:
```
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
```

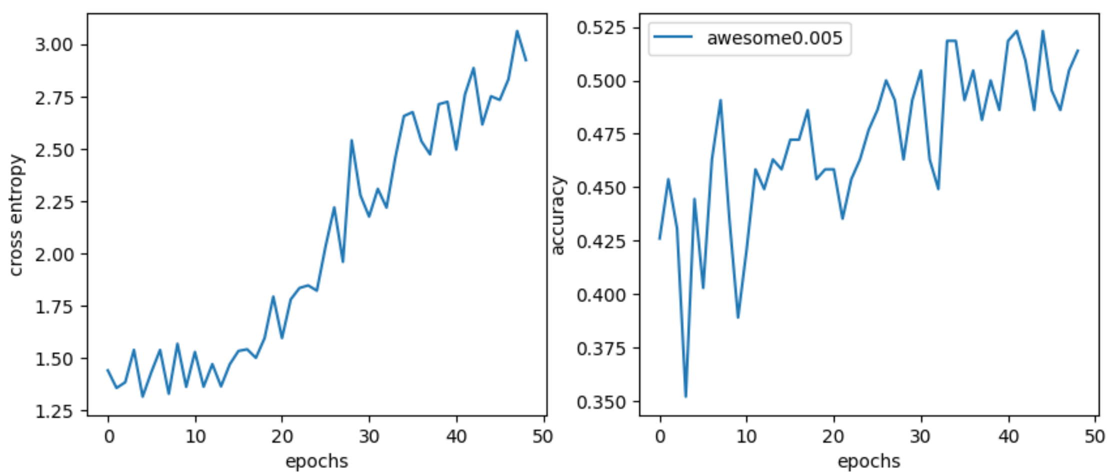
[81]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

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()
```

[81]: <matplotlib.legend.Legend at 0x12dfbee90>



[82]:
```
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*7,512)
        self.bn5 = nn.BatchNorm1d(512)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(512,5)

    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
```

[84]:

```
# TRAINING
model = CNNNet().to(device)

lr = 0.005
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, 25):
    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: 1.583290
Train Epoch: 1 [320/864 (37%)]  Loss: 1.522830
Train Epoch: 1 [640/864 (74%)]  Loss: 1.362087
Test set: Average loss: 1.3419, Accuracy: 96/216 (44%)
```

```
Train Epoch: 2 [0/864 (0%)]     Loss: 1.068226
Train Epoch: 2 [320/864 (37%)]  Loss: 1.202092
Train Epoch: 2 [640/864 (74%)]  Loss: 1.200186
Test set: Average loss: 1.4055, Accuracy: 105/216 (49%)

Train Epoch: 3 [0/864 (0%)]     Loss: 0.993706
Train Epoch: 3 [320/864 (37%)]  Loss: 1.304744
Train Epoch: 3 [640/864 (74%)]  Loss: 1.339173
Test set: Average loss: 1.5271, Accuracy: 73/216 (34%)

Train Epoch: 4 [0/864 (0%)]     Loss: 1.215430
Train Epoch: 4 [320/864 (37%)]  Loss: 1.433880
Train Epoch: 4 [640/864 (74%)]  Loss: 1.141655
Test set: Average loss: 1.4244, Accuracy: 95/216 (44%)

Train Epoch: 5 [0/864 (0%)]     Loss: 1.092364
Train Epoch: 5 [320/864 (37%)]  Loss: 1.428706
Train Epoch: 5 [640/864 (74%)]  Loss: 1.051724
Test set: Average loss: 1.3907, Accuracy: 96/216 (44%)

Train Epoch: 6 [0/864 (0%)]     Loss: 1.153327
Train Epoch: 6 [320/864 (37%)]  Loss: 1.410729
Train Epoch: 6 [640/864 (74%)]  Loss: 1.397132
Test set: Average loss: 1.4144, Accuracy: 93/216 (43%)

Train Epoch: 7 [0/864 (0%)]     Loss: 1.139777
Train Epoch: 7 [320/864 (37%)]  Loss: 0.963008
Train Epoch: 7 [640/864 (74%)]  Loss: 1.362018
Test set: Average loss: 1.8668, Accuracy: 102/216 (47%)

Train Epoch: 8 [0/864 (0%)]     Loss: 1.036156
Train Epoch: 8 [320/864 (37%)]  Loss: 1.674001
Train Epoch: 8 [640/864 (74%)]  Loss: 1.182488
Test set: Average loss: 1.4237, Accuracy: 113/216 (52%)

Train Epoch: 9 [0/864 (0%)]     Loss: 1.212816
Train Epoch: 9 [320/864 (37%)]  Loss: 1.294547
Train Epoch: 9 [640/864 (74%)]  Loss: 1.250975
Test set: Average loss: 1.3487, Accuracy: 104/216 (48%)

Train Epoch: 10 [0/864 (0%)]    Loss: 1.022856
Train Epoch: 10 [320/864 (37%)] Loss: 1.017178
Train Epoch: 10 [640/864 (74%)] Loss: 1.199377
Test set: Average loss: 1.3019, Accuracy: 101/216 (47%)

Train Epoch: 11 [0/864 (0%)]    Loss: 0.934561
Train Epoch: 11 [320/864 (37%)] Loss: 0.873475
```

```
Train Epoch: 11 [640/864 (74%)] Loss: 1.248896
Test set: Average loss: 1.3386, Accuracy: 96/216 (44%)

Train Epoch: 12 [0/864 (0%)]    Loss: 1.164412
Train Epoch: 12 [320/864 (37%)] Loss: 1.023077
Train Epoch: 12 [640/864 (74%)] Loss: 1.098583
Test set: Average loss: 1.5238, Accuracy: 89/216 (41%)

Train Epoch: 13 [0/864 (0%)]    Loss: 0.922669
Train Epoch: 13 [320/864 (37%)] Loss: 1.109569
Train Epoch: 13 [640/864 (74%)] Loss: 0.880644
Test set: Average loss: 1.3623, Accuracy: 101/216 (47%)

Train Epoch: 14 [0/864 (0%)]    Loss: 1.126411
Train Epoch: 14 [320/864 (37%)] Loss: 1.036827
Train Epoch: 14 [640/864 (74%)] Loss: 0.882298
Test set: Average loss: 1.8276, Accuracy: 108/216 (50%)

Train Epoch: 15 [0/864 (0%)]    Loss: 0.939078
Train Epoch: 15 [320/864 (37%)] Loss: 1.051155
Train Epoch: 15 [640/864 (74%)] Loss: 1.366753
Test set: Average loss: 1.3547, Accuracy: 104/216 (48%)

Train Epoch: 16 [0/864 (0%)]    Loss: 0.759326
Train Epoch: 16 [320/864 (37%)] Loss: 1.055355
Train Epoch: 16 [640/864 (74%)] Loss: 1.309602
Test set: Average loss: 1.5599, Accuracy: 98/216 (45%)

Train Epoch: 17 [0/864 (0%)]    Loss: 1.005337
Train Epoch: 17 [320/864 (37%)] Loss: 1.083367
Train Epoch: 17 [640/864 (74%)] Loss: 0.875198
Test set: Average loss: 1.7869, Accuracy: 101/216 (47%)

Train Epoch: 18 [0/864 (0%)]    Loss: 0.897862
Train Epoch: 18 [320/864 (37%)] Loss: 0.844346
Train Epoch: 18 [640/864 (74%)] Loss: 0.916109
Test set: Average loss: 1.4525, Accuracy: 109/216 (50%)

Train Epoch: 19 [0/864 (0%)]    Loss: 0.911923
Train Epoch: 19 [320/864 (37%)] Loss: 0.780164
Train Epoch: 19 [640/864 (74%)] Loss: 0.915069
Test set: Average loss: 1.5945, Accuracy: 103/216 (48%)

Train Epoch: 20 [0/864 (0%)]    Loss: 1.072943
Train Epoch: 20 [320/864 (37%)] Loss: 0.647714
Train Epoch: 20 [640/864 (74%)] Loss: 0.858513
Test set: Average loss: 1.7827, Accuracy: 106/216 (49%)
```

```
Train Epoch: 21 [0/864 (0%)]    Loss: 0.572355
Train Epoch: 21 [320/864 (37%)] Loss: 0.799647
Train Epoch: 21 [640/864 (74%)] Loss: 0.916828
Test set: Average loss: 1.8948, Accuracy: 103/216 (48%)

Train Epoch: 22 [0/864 (0%)]    Loss: 0.803661
Train Epoch: 22 [320/864 (37%)] Loss: 0.439807
Train Epoch: 22 [640/864 (74%)] Loss: 0.811048
Test set: Average loss: 1.7197, Accuracy: 89/216 (41%)

Train Epoch: 23 [0/864 (0%)]    Loss: 0.882532
Train Epoch: 23 [320/864 (37%)] Loss: 0.754777
Train Epoch: 23 [640/864 (74%)] Loss: 0.745789
Test set: Average loss: 1.8489, Accuracy: 102/216 (47%)

Train Epoch: 24 [0/864 (0%)]    Loss: 0.466339
Train Epoch: 24 [320/864 (37%)] Loss: 0.671240
Train Epoch: 24 [640/864 (74%)] Loss: 0.692526
Test set: Average loss: 1.9320, Accuracy: 98/216 (45%)
```