Fashion-MNIST Image Classification

Yuan Gao

Abstract

In this project, I compare two supervised learning models on the Fashion-MNIST dataset: an MLP Classifier and a Convolutional Neural Network. The MLP must flatten each image into a long vector, while the CNN keeps the 2-D structure. I train both models and test them on the same dataset to see how they perform. The results show that the MLP reaches 89.04% accuracy, and the CNN reaches 92.01% accuracy. This project helps me understand how different model structures affect image classification performance.

Introduction

The goal of this project is to classify clothing images from the Fashion-MNIST dataset. This dataset contains 70,000 grayscale images of size 28×28. There are ten categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. Each image has one correct label. Professor explained that MLPs flatten the image, which removes the spatial patterns inside it. CNNs keep the image shape and usually perform better on image tasks. Because of this, I train both models and compare their performance to helps me understand why CNNs are more effective for image data.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Data

This project uses dataset from Zalando. Each image already has a label from 0 to 9 that indicates which clothing class it belongs to. There are 60,000 images for training and 10,000 images for testing. To prepare the data, the dataset is loaded and the pixel values are normalized to be between 0 and 1 by dividing by 255.0.

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# normalize
x_train = x_train / 255.0
x_test = x_test / 255.0
```

The CNN part of the project is implemented in PyTorch, so after normalization the NumPy arrays are converted into PyTorch tensors. A channel dimension is added so that each image has shape (1, 28, 28), which is the standard format for convolutional layers (channels, height, width). The tensors are then wrapped in a TensorDataset and a DataLoader is used to create mini-batches with shape (batch_size, 1, 28, 28) for training and testing.

```
# convert numpy arrays to PyTorch tensors with channel dimension (N, 1, 28, 28)
X_train_torch = torch.tensor(x_train, dtype=torch.float32).unsqueeze(1)
X_test_torch = torch.tensor(x_test, dtype=torch.float32).unsqueeze(1)

y_train_torch = torch.tensor(y_train, dtype=torch.long)
y_test_torch = torch.tensor(y_test, dtype=torch.long)

train_ds = TensorDataset(X_train_torch, y_train_torch)
test_ds = TensorDataset(X_test_torch, y_test_torch)

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=64, shuffle=False)
```

Modeling

Neural Network (MLP Classifier) is a feed-forward neural network from scikit-learn. Because it cannot use the 2-D structure of images, each 28×28 image must be flattened into a 784-dimensional vector before training.

```
# reshape images for scikit-learn (flatten)
X_train_flat = x_train.reshape(len(x_train), 28 * 28)
X_test_flat = x_test.reshape(len(x_test), 28 * 28)

print("Flattened shape:", X_train_flat.shape)
```

```
Flattened shape: (60000, 784)
```

The MLP is defined and trained with two hidden layers (256, 128), ReLU activation, the Adam optimizer, small L2 regularization (alpha = 1e-4), and a larger max_iter = 200 for better optimization.

```
# train MLPClassifier
mlp = MLPClassifier(
    hidden_layer_sizes=(256, 128),   # deeper network
    activation='relu',
    solver='adam',
    alpha=1e-4,                      # L2 regularization
    learning_rate_init=0.001,
    max_iter=200,                    # train longer (very important)
    random_state=0
)

mlp.fit(X_train_flat, y_train)

y_pred_basic = mlp.predict(X_test_flat)
test_acc_basic = accuracy_score(y_test, y_pred_basic)

print("MLPClassifier test accuracy:", test_acc_basic)
```

This larger network improves accuracy because it can learn more complex patterns than the small, single-layer MLP. However, the MLP still treats every pixel as an independent input feature and does not know that nearby pixels form meaningful shapes or edges.

Convolutional Neural Network (CNN) keeps the image in its original 2-D form with shape (1, 28, 28) and uses convolution layers to learn spatial patterns directly. I use a LeNet-style CNN implemented in PyTorch.

The first convolution layer takes the single-channel image (1, 28, 28) and uses 32 filters of size 5 x 5 with padding 2. The output shape becomes (32, 28, 28). Then a 2 x 2 max-pooling layer with stride 2 reduces it to (32, 14, 14).

The second convolution layer starts from (32, 14, 14) and uses 64 filters of size 5 x 5 (no padding). The output shape becomes (64, 10, 10). Another 2 x 2 max-pooling layer with stride 2 reduces this to (64, 5, 5).

After these two convolution-and-pooling blocks, the feature maps (64, 5, 5) are flattened into a vector of size 64 x 5 x 5 = 1600. Although this vector is flattened, it keeps meaningful spatial information because the earlier convolution and pooling layers have already learned and preserved important patterns. This vector goes through two fully connected layers: first from 1600 to 256, then from 256 to 128.

ReLU activations are applied after each hidden layer (after both convolution layers and both fully connected layers). A dropout layer with probability 0.5 is placed after the first fully connected layer to reduce overfitting by randomly dropping some activations during training. The final linear layer maps 128 to 10 and outputs 10 logits, one for each Fashion-MNIST class.

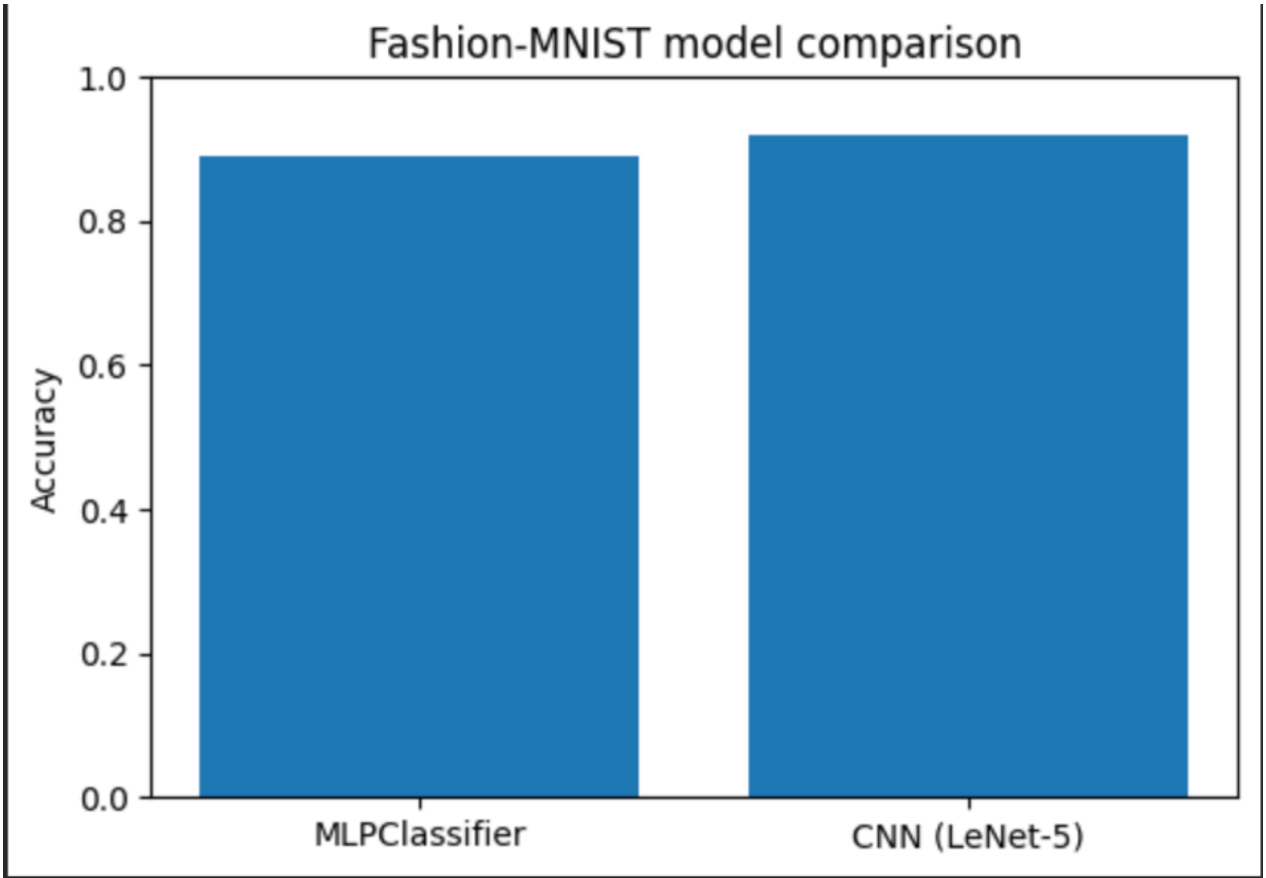
```python
class LeNet5(nn.Module):
    def __init__(self, num_classes=10):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(64 * 5 * 5, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, num_classes)

        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)          # flatten
        x = F.relu(self.fc1(x))
        x = self.dropout(x)                # dropout after first FC
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

This CNN is trained for 15 epochs using the Adam optimizer with learning rate 0.001 and cross-entropy loss. In each epoch, the model iterates over all mini-batches from the training DataLoader, computes the loss, backpropagates the gradients, and updates the weights. The average training loss for each epoch is stored and printed to monitor convergence.

```
model_cnn = LeNet5(num_classes=10)
optimizer = torch.optim.Adam(model_cnn.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

n_epochs = 15
losses_cnn = []

for epoch in range(n_epochs):
    model_cnn.train()
    running_loss = 0.0

    for Xb, yb in train_loader:
        optimizer.zero_grad()
        outputs = model_cnn(Xb)
        loss = criterion(outputs, yb)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader)
    losses_cnn.append(avg_loss)
    print(f"epoch {epoch}, loss {avg_loss:.4f}")
```

Results

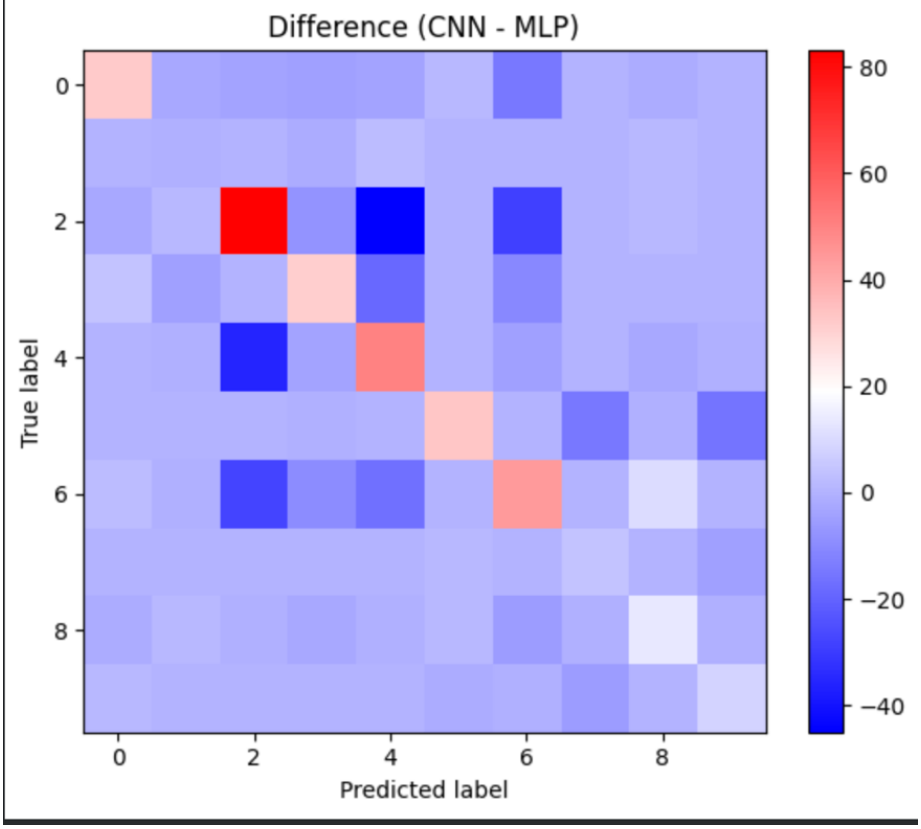
The MLP Classifier reaches a test accuracy of 0.8904, while the CNN (LeNet-5) achieves a higher accuracy of 0.9201. The bar chart below summarizes the performance




```
MLPClassifier accuracy: 0.8904
CNN accuracy: 0.9201
```

In the difference matrix (CNN − MLP), each number tells how many more images the CNN placed in that cell compared to the MLP. Positive values on the diagonal, such as 32 at (0,0) and 8 at (9,9), mean the CNN correctly classified 32 more T-shirts and 8 more ankle boots than the MLP. Negative values off the diagonal show where the MLP handled certain confusions slightly better, for example, the −45 at (2,4) means the MLP made 45 fewer mistakes than the CNN, and −28 at (6,2) shows fewer errors. The color plot simply visualizes these numbers: red areas mark positive entries where the CNN improves over the MLP, and blue areas mark negative entries where the MLP performs a bit better.

```
Difference matrix (CNN - MLP):
[[ 32  -3  -4  -5  -4   1 -15   0  -2   0]
 [  0  -1   0  -2   2   0   0   0   1   0]
 [ -3   1  83  -8 -45   0 -29   0   1   0]
 [  4  -5   0  31 -19   0 -11   0   0   0]
 [  0  -1 -36  -4  50   0  -5   0  -3  -1]
 [  0   0   0  -1   0  33   0 -15  -1 -16]
 [  2  -1 -28 -10 -17   0  44   0  10   0]
 [  0   0   0   0   0   1   0   4   0  -5]
 [ -2   1  -1  -3  -1   1  -6  -1  13  -1]
 [  1   0   0   0   0  -2  -1  -6   0   8]]
```



Discussion

The results demonstrate that the CNN performs better than the MLP. This makes sense for me to understand their, the MLP has trouble learning important visual patterns and edges. This causes mistakes when classes look similar. The CNN learn local patterns. Because of this, the CNN achieves higher accuracy. Training time also differs. The MLP trains a bit faster, while the CNN takes much longer. But the accuracy gain from the CNN is meaningful, so the extra time is worth it. These results support professor's explanation that Convolution helps the model understand shape and structure, which is important for visual data.

Conclusion

This project was my first step in comparing different models on a real dataset. I learned how to build and train a basic MLP and a simple CNN, and how to compare with this same dataset. I also saw that the choice of model can change performance more than small parameter tweaks. In the future, I can reuse this setup to test other architectures or small changes, such as data augmentation or deeper networks, and see how much they really help. This project also gives me a template for future work: define a clear task, compare models fairly, and focus on understanding not just which one wins, but why and when to use them.