Fashion-MNIST Image Classification

Yuan Gao
UCLA Statistics & Data Science
December 2024

Abstract

In this project I compare two supervised machine learning models on the Fashion-MNIST dataset. The first model is a basic feed-forward neural network implemented with `MLPClassifier` in scikit-learn. The second model is a convolutional neural network (CNN) in PyTorch based on LeNet-5.

Both models try to classify 28×28 grayscale clothing images into 10 classes. The CNN reaches higher test accuracy (MLP: 0.8709, CNN: 0.8953). This supports the idea from class that flattening images loses spatial information, while convolutional layers can use this structure and usually work better on images.

1 Introduction

Image classification is a common task in machine learning. Fashion-MNIST is a small image dataset that is often used in teaching and simple projects. It lets us test models quickly while still being harder than the original digit MNIST dataset.

In lecture we learned that scikit-learn has simple neural networks through `MLPClassifier`, but it does not provide CNNs. For CNNs we need a deep learning library such as PyTorch. In this project I follow that idea: I first train a basic MLP as my baseline model, and then build a LeNet-5 style CNN to see how much improvement we get.

The main question is simple: *does the CNN clearly beat the basic MLP on Fashion-MNIST, and can we explain why?*

2 Data

Fashion-MNIST contains 60,000 training images and 10,000 test images. Each image is a 28×28 grayscale picture of one clothing item. There are 10 classes (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot).

In my notebook I first load the data from Keras and normalize pixel values by dividing by 255.0. For the MLP model I flatten each image into a 784-dimensional vector:

$$\text{input size} = 28 \times 28 = 784$$

For the CNN model I keep the 2D structure and reshape each image to shape (1, 28, 28), where 1 is the number of channels (grayscale).

3 Modelling

3.1 MLPClassifier (Basic Neural Network)

The scikit-learn `MLPClassifier` is a feed-forward neural network trained with backpropagation. Because it only accepts vectors, I must flatten each 28×28 image. The network I use in this project has:

- One hidden layer with 64 units and ReLU activation.
- An output layer with 10 units (one for each class).

The forward pass can be written as

$$\hat{y} = softmax(W_2 \cdot ReLU(W_1 x + b_1) + b_2)$$

The core code in my notebook is:

```
mlp = MLPClassifier(hidden_layer_sizes=(64,),
                    activation='relu',
                    solver='adam',
                    max_iter=20,
                    verbose=False,
                    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)
```

I also compute and plot a confusion matrix for this model. In the HTML report I place the image from that plot into Figure 2 below.

3.2 Convolutional Neural Network (CNN: LeNet-5)

For the CNN I use PyTorch and follow a LeNet-5 style architecture. Convolutional layers slide small filters over the image to detect local patterns such as edges. Pooling layers reduce the spatial size and add some translation invariance.

The architecture I use is:

- Conv2d(1 → 6, kernel 5×5) + AvgPool
- Conv2d(6 → 16, kernel 5×5) + AvgPool
- Fully connected: 16×5×5 → 120
- Fully connected: 120 → 84
- Fully connected: 84 → 10

The model is defined in my notebook as:

```
class LeNet5(nn.Module):
    def __init__(self, num_classes=10):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, padding=2)
        self.pool1 = nn.AvgPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.pool2 = nn.AvgPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, num_classes)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

I train this model for 5 epochs using the Adam optimizer and cross-entropy loss. After training, I compute predictions on the test set and build a confusion matrix in the notebook, which I use as Figure 3 in this report.

4 Results

On the Fashion-MNIST test set I get the following accuracies:

- **MLPClassifier:** 0.8709
- **CNN (LeNet-5):** 0.8953

Model accuracy comparison

Figure 1: Test accuracy comparison between MLPClassifier and CNN.

MLP confusion matrix

Figure 2: Confusion matrix for the MLPClassifier. (In practice I replace this image with the confusion matrix plot exported from my notebook.)

CNN confusion matrix

Figure 3: Confusion matrix for the CNN (LeNet-5). (Also replaced with the real confusion matrix plot from my notebook.)

5 Discussion

The basic MLP must flatten each 28×28 image into a long vector. This step destroys the 2D structure of the picture. Because of this, the model does not know which pixels are neighbors, and it has more trouble learning shape information. From the confusion matrix we can see that the MLP often confuses classes that look similar, such as T-shirt/top and Shirt.

The CNN keeps the 2D layout of the image. Convolutional layers learn filters that detect local edges and patterns. Pooling layers summarize those patterns and make the model more stable to small shifts. As a result, the CNN reaches a higher accuracy and its confusion matrix looks cleaner, with stronger values on the diagonal and fewer large off-diagonal entries.

These results match what we learned in class: for image data, CNNs usually perform much better than simple feed-forward networks, especially when the images contain local structure that is important for the classification task.

6 Conclusion

In this project I compared two neural network approaches for Fashion-MNIST image classification. A basic MLP implemented with scikit-learn reached a test accuracy of 0.8709, while a LeNet-5 CNN in PyTorch achieved 0.8953. The CNN clearly gave better performance.

The main reason is that the CNN uses convolution to keep spatial relationships between pixels, while the MLP fully ignores that structure. This experiment supports the idea that for most image tasks, CNNs are a more appropriate choice than simple fully connected networks.

For future work I could try deeper CNN architectures, data augmentation, and regularization methods such as dropout and batch normalization to improve the model further.

7 References

1. Fashion-MNIST Dataset, Zalando Research.
2. scikit-learn Documentation: MLPClassifier.
3. PyTorch Documentation.
4. UCLA course materials and project template.