Report Title：	**Convolutional Neural Networks**
Author Name：	**王小慧**
Student ID：	**21307110415**
Institute：	**信息科学与工程学院**
Report Date：	**2024/11/28**

Contents

1 Introduction

In this experiment, we investigate the performance of three convolutional neural network (CNN) architectures—LeNet, AlexNet, and a custom-designed CNN—on the Fashion-MNIST dataset. LeNet and AlexNet showcase significant advancements in network design and performance for image classification.

We also introduce a custom CNN which is designed to strike a balance between computational efficiency and classification accuracy. By comparing these three models, we aim to highlight the evolution of CNN architectures and evaluate the potential of a task-specific design for optimal performance.

1.1 Convolutional Neural Networks (CNNs)

CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images. A CNN consists of several layers, including convolutional layers, pooling layers, and fully connected layers.

1.1.1 Convolutional Layers

These layers perform convolution operations to extract local features from the image. The convolution operation can be mathematically expressed as:

$$y(i,j) = (x * w)(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{n} x(i+m, j+n) \cdot w(m,n) \tag{1}$$

where $x(i,j)$ is the input image, $w(m,n)$ is the filter (or kernel), $y(i,j)$ is the output feature map,

The convolution operation slides the filter over the image to produce feature maps that highlight certain patterns, such as edges or textures.

1.1.2 Pooling Layers

Pooling layers (such as max pooling) downsample the feature maps, reducing their spatial dimensions while retaining important features. Max pooling can be expressed as:

$$y(i,j) = \max\left(x(i,j), x(i+1,j), \ldots, x(i+k, j+k)\right) \tag{2}$$

where a local region is reduced to the maximum value, thus providing a form of translation invariance.

The key advantage of CNNs is their ability to learn features at multiple levels of abstraction. Early layers in a CNN typically learn simple patterns such as edges and textures, while deeper layers combine these patterns to form more complex representations, such as object parts or entire objects.

1.2 LeNet

LeNet[1], introduced by Yann LeCun in 1994, was one of the first CNN architectures, developed primarily for handwritten digit recognition. Its layered structure effectively addresses challenges faced by traditional multi-layer perceptrons (MLPs), including inefficient handling of spatial correlations in images and excessive model size for large inputs. LeNet uses convolutional and pooling layers to retain spatial hierarchies while reducing the parameter count. Key components:

- Convolutional layers to extract spatial features.
- Subsampling layers for dimensionality reduction.
- Fully connected layers for classification.

1.3 AlexNet

AlexNet[2], the winner of the 2012 ImageNet competition, revolutionized CNN design by employing innovations such as ReLU activations, dropout regularization, and multi-GPU training. It tackled large-scale image classification tasks and significantly reduced error rates. Key architectural differences from LeNet include:

- Larger input image resolution.
- Deeper structure with larger convolutional filters in the initial layers.
- Use of ReLU and Dropout.

1.4 Batch Normalization

Batch Normalization (BatchNorm) is a widely used technique in deep learning to improve training stability and accelerate convergence. It addresses issues such as internal covariate shift, where the distribution of input features to a layer changes during training, by normalizing inputs within mini-batches.

For each feature in a mini-batch, BatchNorm normalizes the input values by subtracting the mini-batch mean and dividing by the mini-batch standard deviation:

$$\hat{x}_i = \frac{x_i - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}} \tag{3}$$

where x_i is the input feature for a given mini-batch, μ_{batch} is the mean of the mini-batch, σ^2_{batch} is the variance of the mini-batch, ϵ is a small constant added for numerical stability.

2 Experimental Setup

We first used LeNet and AlexNet to evaluate the effectiveness of convolutional neural networks (CNNs) on the Fashion-MNIST dataset. These experiments provided a baseline for comparison with a custom-designed CNN model.

2.1 LeNet

2.1.1 Model Architecture

The LeNet architecture consists of the following layers:

```
class LeNet(nn.Module):
def __init__(self):
    super(LeNet, self).__init__()
    self.net = nn.Sequential(
        nn.Conv2d(1,6,kernel_size=5,padding=2), nn.Sigmoid(),
        nn.AvgPool2d(kernel_size=2,stride=2),
        nn.Conv2d(6,16,kernel_size=5), nn.Sigmoid(),
        nn.AvgPool2d(kernel_size=2, stride=2),
        nn.Flatten(),
        nn.Linear(16*5*5,120), nn.Sigmoid(),
        nn.Linear(120,84), nn.Sigmoid(),
        nn.Linear(84,10))

def forward(self, x):
    x = self.net(x)
    return x
```

2.1.2 Parameter Configuration

- **Optimizer and Learning Rate Decay:** Adam + StepLR
- **Loss Function:** Cross-Entropy Loss
- **Other Hyperparameters:** Batch Size = 64, Epochs = 30, Learning Rate = 0.005

2.1.3 Result

LeNet's relatively simple architecture ensures faster training on the Fashion-MNIST dataset while maintaining a reasonable level of accuracy, which is **89.9%**. LeNet easily outperformed the Multi-Layer Perceptron (MLP) model, showcasing its ability to effectively capture spatial hierarchies and reduce computational complexity compared to fully connected layers. This highlights the advantages of convolutional layers in processing image data. Training and testing losses are shown in Figure 1.

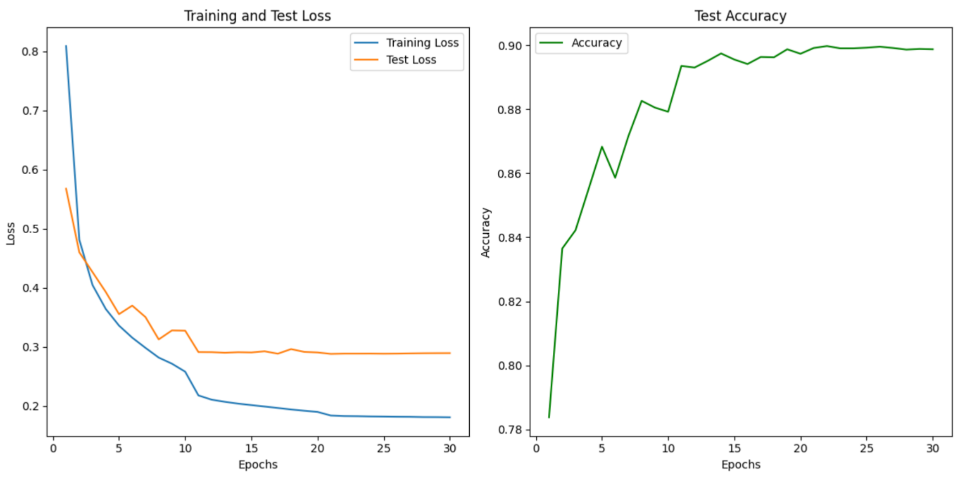

Figure 1 Training and Testing Losses for LeNet

2.2 AlexNet

2.2.1 Model Architecture

The original AlexNet design assumes large input sizes, such as 224×224, which are computationally expensive for datasets like Fashion-MNIST. Using the original parameters required resizing the dataset, leading to significantly longer running times without proportional improvement in accuracy. By adapting the kernel sizes, pooling configurations, and feature map sizes, we preserved AlexNet's hierarchical feature extraction capabilities while optimizing it for the smaller input size.

```python
class AlexNet(nn.Module):
    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=6, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 1 * 1, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(1024, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

2.2.2 Parameter Configuration

- **Optimizer and Learning Rate Decay:** Adam + StepLR
- **Loss Function:** Cross-Entropy Loss
- **Data Augmentation:** Random Rotation and Random Horizontal Flip.
- **Other Hyperparameters:** Batch Size = 64, Epochs = 20, Learning Rate = 0.001

2.2.3 Result

The larger number of parameters in AlexNet allowed it to achieve better feature representation compared to LeNet, which achieved **91.2%** in accuracy. But it required significantly higher computational resources than LeNet. Training and testing losses are shown in Figure 2.

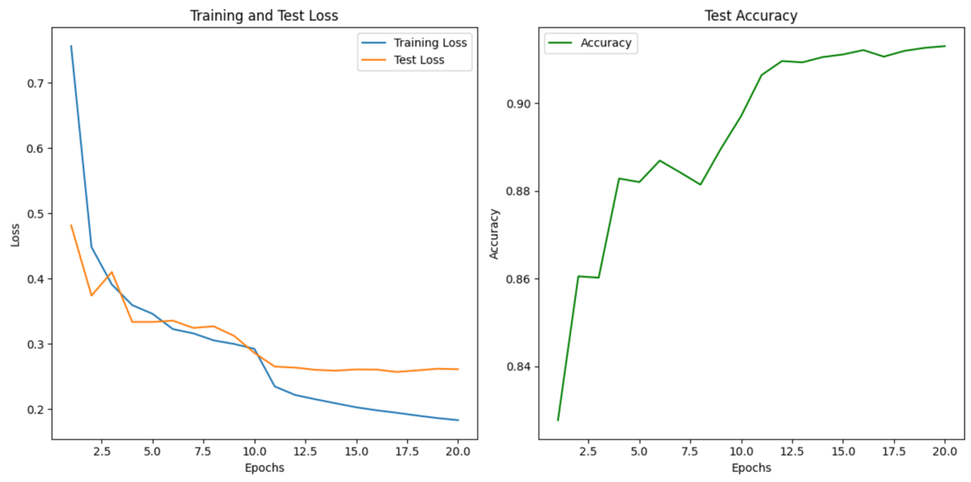

Figure 2 Training and Testing Losses for AlexNet

3 Experimental Procedure

In this section, we explore a custom CNN model specifically designed for the Fashion-MNIST dataset. The architecture of this model strikes a balance between simplicity and complexity, offering a network that is deeper than LeNet but more computationally efficient than AlexNet. This custom model also incorporates several modern techniques to improve performance and prevent overfitting, such as Batch Normalization and Dropout. This model ensures better performance while maintains a manageable computational cost.

3.1 Data Preprocessing

To mitigate overfitting and enhance the generalization capability of our models, we employed data augmentation techniques, including Random Rotation and Random Horizontal Flip. We also applied data normalization to standardize pixel values across the dataset. Normalization accelerates the convergence of the optimization process and ensures a consistent input distribution for the network layers.

```
    train_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.RandomRotation(10),
        transforms.RandomHorizontalFlip(),
        transforms.Normalize((0.5,), (0.5,))
    ])
    test_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])
```

3.2 Model Architecture

There are two different blocks in the custom CNN:

- **Feature Extractor:** Two convolutional layers with a max pooling layer and batch normalization, followed by ReLU activations.
- **Classifier:** Fully connected layers for classification, with a dropout layer to prevent overfitting.

Stacking two convolutional layers allows the network to extract more complex and hierarchical features from the input. The first convolution focuses on low-level patterns (e.g., edges, corners), while the second builds on these to capture higher-order relationships.

```python
class ConvolutionalNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=1, padding=1)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu1 = nn.ReLU()

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.relu2 = nn.ReLU()

        self.fc1 = nn.Linear(128 * 8 * 8, 512)
        self.relu4 = nn.ReLU()
        self.drop = nn.Dropout()
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.pool1(x)
        x = self.bn1(x)
        x = self.relu1(x)

        x = self.conv3(x)
        x = self.conv4(x)
        x = self.pool2(x)
        x = self.bn2(x)
        x = self.relu2(x)

        x = x.view(-1, 128 * 8 * 8)
        x = self.fc1(x)
        x = self.relu4(x)
        x = self.drop(x)
        x = self.fc2(x)

        return x
```

3.3 Parameter Configuration

- **Optimizer and Learning Rate Decay:** Adam + StepLR
- **Loss Function:** Cross-Entropy Loss
- **Other Hyperparameters:** Batch Size = 64, Epochs = 15, Learning Rate = 0.003

4 Results and Analysis

The custom CNN achieved a test accuracy of **93.3%**, as shown in Figure 3. This result represents an improvement over previous experiments and highlights the model's ability to effectively learn and generalize from the Fashion-MNIST dataset.

However, a slight degree of overfitting was observed, despite the minor overfitting, the model's performance is well-optimized for the dataset.

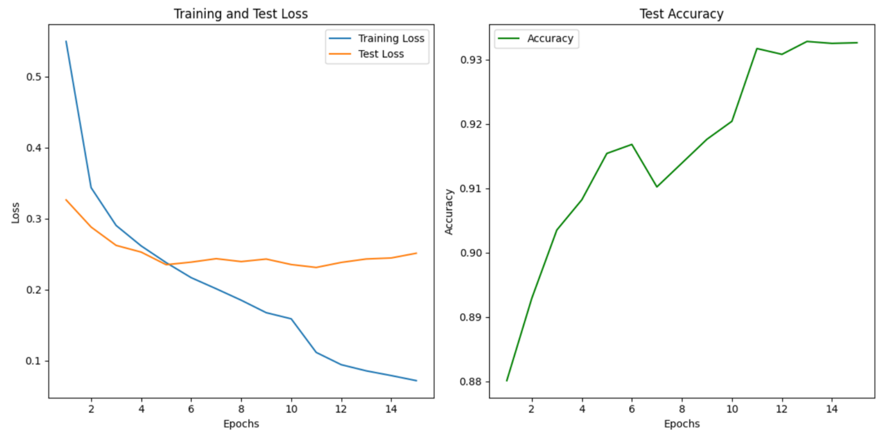

Figure 3 Training and Testing Losses for Custom CNN

4.1 Impact of BatchNorm

We conducted an additional experiment to explore the performance of the model without the BatchNorm layer while keeping the exact same settings. The results revealed that the absence of BatchNorm led to slower convergence during training and a drop in test accuracy to **86.9%**, as shown in Figure 4. This model also exhibited highly unstable learning behavior with erratic loss values.

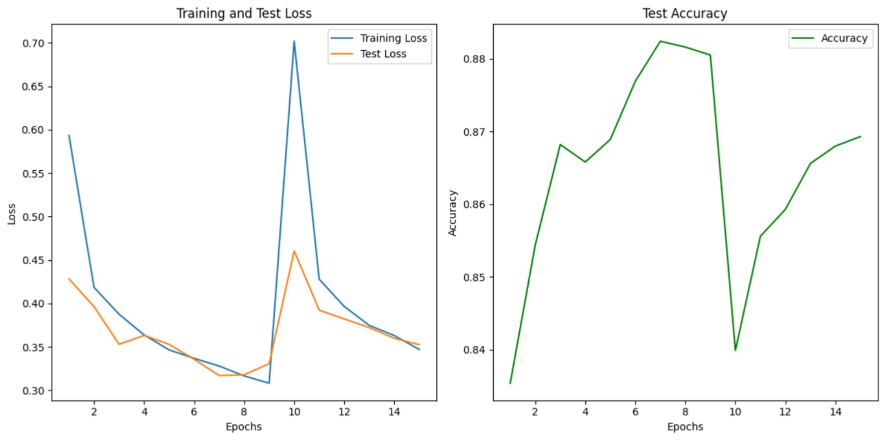

Figure 4 Training and Testing Losses for Custom CNN without BatchNorm: demonstrating highly unstable learning behavior under the same settings

To further analyze the impact, we revised certain hyperparameters, mainly reducing the learning rate to **0.001**, to accommodate the model's slower and less stable convergence. These adjustments resulted in a marked improvement, achieving a test accuracy of **92.6%**, as shown in Figure 5. However, this performance still lagged behind the model with BatchNorm, which achieved **93.3%** under the original settings.

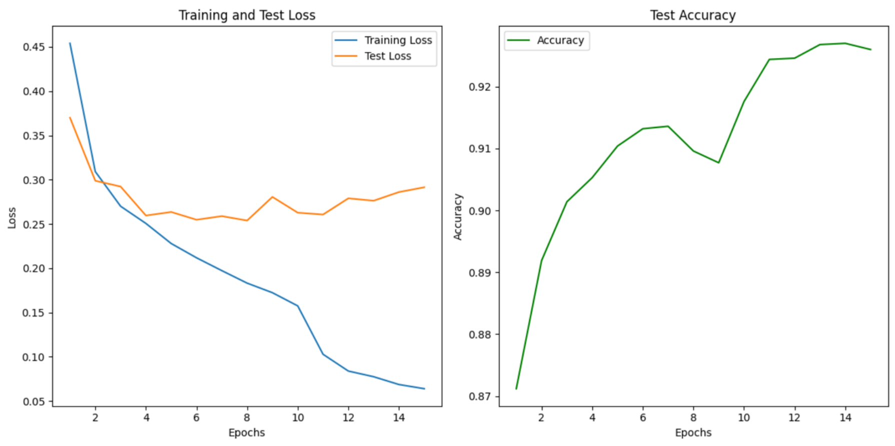

Figure 5 Training and Testing Losses for Custom CNN without BatchNorm: adjusted hyperparameters to improve performance, but showing signs of over-fitting

These findings underscore the advantages of incorporating BatchNorm into the architecture. The layer not only accelerates convergence by normalizing the inputs to each layer but also acts as a form of regularization, mitigating the risk of overfitting and enabling the model to generalize better to unseen data. In contrast, the absence of BatchNorm demands careful tuning of hyperparameters and longer training durations to achieve comparable results, adding to the computational and design complexity.

5 Conclusion

In this experiment, we explored the performance of three convolutional neural networks: LeNet, AlexNet, and a custom-designed CNN, on the Fashion-MNIST dataset. The results demonstrated the excellent capability of CNNs for image classification tasks.

This study underscores the importance of network depth and architectural design in achieving high classification performance. While deeper networks generally perform better by capturing more complex features, careful consideration of computational efficiency and regularization techniques like BatchNorm is essential to optimize performance and generalization. Overall, the results reaffirm the power of CNNs as a foundational approach for image classification tasks and emphasize the need for thoughtful model design tailored to specific datasets and tasks.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.