Problem 1. Write a program to perform the convolution operation of A and B using a stride of S = 2 and padding of P = 1. You may use `torch.nn.functional.conv2d.`

In [35]:
```python
import torch

# Function to perform the convolution operation
def convolution(input, filter, stride=2, padding=1):
    # Perform the convolution operation
    output = torch.nn.functional.conv2d(input, filter, stride=stride, padding=padding)
    return output

# Define input and kernel
A = torch.tensor(([[[[9, 4, 5],
                     [3, 6, 8],
                     [8, 1, 9]]]]), dtype=torch.float32)

# Kernel B: 1 output channel, 1 input channel, height 3, width 3
B = 1/8 * torch.tensor(([[[[0, 1, 0],
                           [1, 4, 1],
                           [0, 1, 0]]]]), dtype=torch.float32)

print("Convolution Result:\n", convolution(A, B))
```

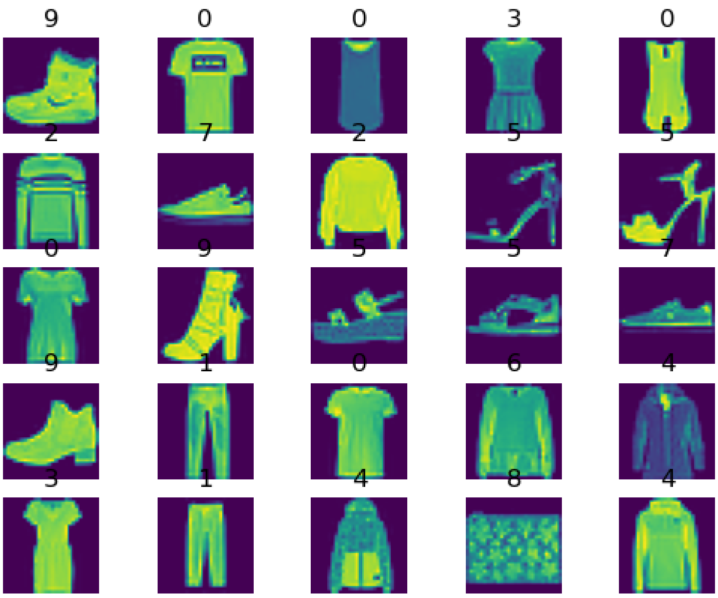
```
Convolution Result:
 tensor([[[[5.3750, 4.0000],
          [4.5000, 5.6250]]]])
```

Problem 2. CNN Training

(a) Plot the first 25 images from the training dataset. How many images does the dataset have (in both the training and test dataset)? What is the size of each image? Also, print the list of class names, which is stored in train dataset.classes.

In [36]:
```python
import torch
import torchvision.datasets as db
import matplotlib.pyplot as plt
import torchvision.transforms as transforms

# Load the Fashion MNIST dataset
train_dataset = db.FashionMNIST(root='.', train=True, download=True)
test_dataset = db.FashionMNIST(root='.', train=False, download=True)

# Check the size of the datasets
print(f'Train dataset size: {len(train_dataset)}')
print(f'Test dataset size: {len(test_dataset)}')

for i in range(25):
    img, lbl = train_dataset[i]
    plt.subplot(5, 5, i+1)
    plt.imshow(img)
    plt.title(lbl)
    plt.axis(False)

print(f'List of classes is {train_dataset.classes}')
print(f'The size of each image is {train_dataset[1][0].size}')
```

```
Train dataset size: 60000
Test dataset size: 10000
List of classes is ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag',
'Ankle boot']
The size of each image is (28, 28)
```



(b) Develop a “sequential” CNN model

In [39]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# Define the Sequential CNN model
class SequentialCNN(nn.Module):
    def __init__(self):
        super(SequentialCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)  # 1 input cha
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(32 * 7 * 7, 10)  # Ensure this matches the output from the flatten layer

    def forward(self, x):
        x = self.conv1(x)   # Output shape: [batch_size, 16, 28, 28]
        x = F.relu(x)
        x = self.pool1(x)   # Output shape: [batch_size, 16, 14, 14]
        x = self.conv2(x)   # Output shape: [batch_size, 32, 14, 14]
        x = F.relu(x)
        x = self.pool2(x)   # Output shape: [batch_size, 32, 7, 7]
        x = self.flatten(x) # Output shape: [batch_size, 32 * 7 * 7] == [batch_size, 1568]
        x = self.fc(x)      # Input shape must match 1568, output shape: [batch_size, 10]
        return F.log_softmax(x, dim=1)  # Return log probabilities

model = SequentialCNN()
print(model)
```

```
SequentialCNN(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc): Linear(in_features=1568, out_features=10, bias=True)
)
```

In [47]:
```python
class SequentialCNN(nn.Module):
    def __init__(self):
        super(SequentialCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(32 * 7 * 7, 10)  # 32 channels * 7 * 7 spatial size after pooling

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool1(x)
        x = self.conv2(x)
```

```
        x = F.relu(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = self.fc(x)
        return F.log_softmax(x, dim=1)

model = SequentialCNN()
```

(c) Train the model using a mini-batch size of 32 for 10 epochs to minimize the cross entropy loss with an SGD optimizer.

In [ ]:
```
import torch.optim as optim
from torch.utils.data import DataLoader

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = db.FashionMNIST(root='.', train=True, download=True, transform=transform)
test_dataset = db.FashionMNIST(root='.', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Example of getting one batch
for images, labels in train_loader:
    # print(f'Batch shape: {images.size}')  # Should be [32, 1, 28, 28]
    output = model(images)
    break

# Set loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(10):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch + 1}/10], Loss: {running_loss / len(train_dataset):.4f}')

# Evaluation on the test set
model.eval()
test_loss = 0.0
correct = 0
total = 0
all_labels = []
all_preds = []

with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_labels.extend(labels.numpy())
        all_preds.extend(predicted.numpy())

# Calculate accuracy
accuracy = 100 * correct / total
print(f'Test Accuracy: {accuracy:.2f}%')
print(f'Test Loss: {test_loss / len(test_dataset):.4f}')
```

```
Epoch [1/10], Loss: 0.0071
Epoch [2/10], Loss: 0.0070
Epoch [3/10], Loss: 0.0068
Epoch [4/10], Loss: 0.0068
Epoch [5/10], Loss: 0.0067
Epoch [6/10], Loss: 0.0066
Epoch [7/10], Loss: 0.0065
Epoch [8/10], Loss: 0.0064
Epoch [9/10], Loss: 0.0063
Epoch [10/10], Loss: 0.0063
Test Accuracy: 91.04%
Test Loss: 0.0082
```

In [64]:

```python
import numpy as np

num_classes = len(set(all_labels))

# Initiate Confusion Matrix
cm = np.zeros((num_classes, num_classes), dtype=int)

for true, pred in zip(all_labels, all_preds):
    cm[true][pred] += 1

print("Confusion Matrix:")
print(cm)

# Calculate per-class accuracy
class_accuracy = cm.diagonal() / cm.sum(axis=1)
worst_classes = class_accuracy.argsort()[:2]  # Get indices of the two worst classes
print(f'\nTwo classes with worst prediction accuracy: {worst_classes}')
```

```
Confusion Matrix:
[[864   1  17  20   3   0  84   0  11   0]
 [  0 989   0   6   2   0   2   0   1   0]
 [ 13   1 861  12  62   0  48   0   3   0]
 [ 13  11   8 928  15   0  23   0   2   0]
 [  2   1  46  31 877   0  42   0   1   0]
 [  0   0   0   1   0 978   0  14   0   7]
 [104   3  67  40  81   0 693   0  12   0]
 [  0   0   0   0   0  10   0 959   0  31]
 [  2   1   1   3   2   1   1   5 984   0]
 [  0   0   0   0   0   6   0  23   0 971]]

Two classes with worst prediction accuracy: [6 2]
```

In [ ]: