Fashion MNIST Project Report

In this project, I implemented a convolutional neural network (CNN) to classify images from the Fashion MNIST dataset. The steps involved in the project are summarized below:

Library Installation: I began by installing the necessary libraries, including PyTorch, torchvision, and other dependencies.

Setup and Configuration: I set up the device for GPU usage if available, ensuring reproducibility by seeding the random number generators.

Data Loading and Processing:

1. I loaded the Fashion MNIST training and test datasets, applying transformations to convert images into PyTorch tensor format.
2. The training data was organized into batches with a batch size of 4, which produced a shape of `(4, 1, 28, 28)` for images and a corresponding tensor for labels.

```
Batch of images shape: torch.Size([4, 1, 28, 28])
Batch of labels: tensor([1, 8, 6, 3])
```

Model Design:

1. I designed a CNN model consisting of two convolutional layers and two fully connected layers.
2. The model was trained using the Adam optimizer with a learning rate of 0.001 for 10 epochs.
3. Initial accuracy results indicated a strong performance, with a training accuracy reaching 99.33% and a validation accuracy of 91.54%.

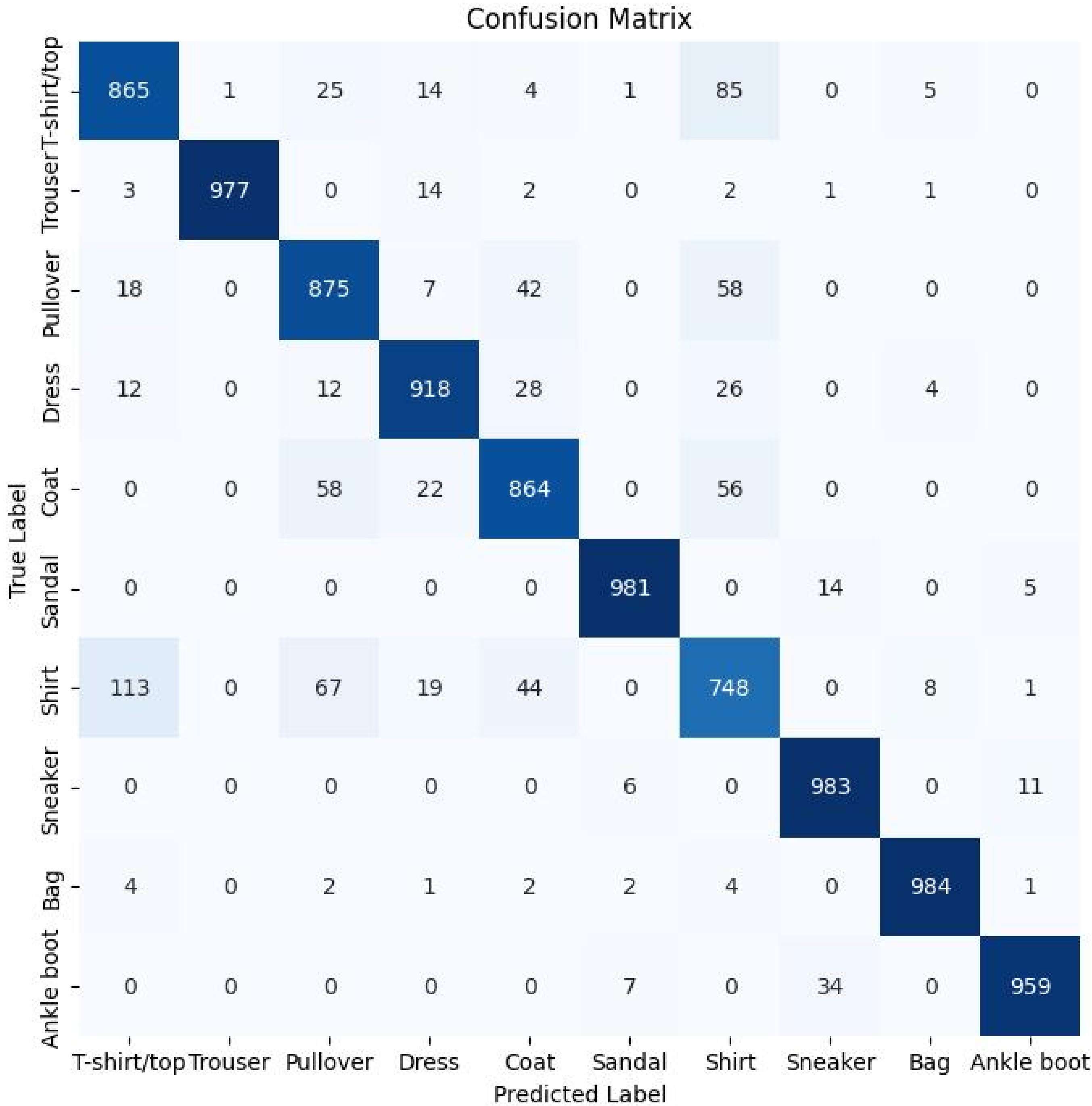
```python
[22] import torch.nn as nn
     import torch.nn.functional as F

     class BasicCNN(nn.Module):
         def __init__(self):
             super(BasicCNN, self).__init__()
             # Input: [batch_size, 1, 28, 28]
             self.conv1 = nn.Conv2d(1, 32, 3)  # Output: [batch_size, 32, 26, 26]

             # Input: [batch_size, 32, 26, 26]
             self.conv2 = nn.Conv2d(32, 64, 3) # Output: [batch_size, 64, 11, 11]

             self.fc1 = nn.Linear(64 * 5 * 5, 128)  # Flattening: [batch_size, 64*5*5]
             self.fc2 = nn.Linear(128, 10)

         def forward(self, x):
             # Input: [batch_size, 1, 28, 28]
             x = F.relu(self.conv1(x))
             # Shape: [batch_size, 32, 26, 26]
             x = F.max_pool2d(x, 2)
             # Shape: [batch_size, 32, 13, 13]

             x = F.relu(self.conv2(x))
             # Shape: [batch_size, 64, 11, 11]
             x = F.max_pool2d(x, 2)
             # Shape: [batch_size, 64, 5, 5]

             x = x.view(-1, 64 * 5 * 5) # Flattening
             x = F.relu(self.fc1(x))
             x = self.fc2(x)
             return F.log_softmax(x, dim=1)
```

```python
[24] optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
     criterion = nn.CrossEntropyLoss()
```

```
Epoch [1/10], Loss: 0.1456, Training Accuracy: 94.56%, Validation Accuracy: 90.30%
Epoch [2/10], Loss: 0.1108, Training Accuracy: 95.76%, Validation Accuracy: 90.93%
Epoch [3/10], Loss: 0.0847, Training Accuracy: 96.85%, Validation Accuracy: 90.82%
Epoch [4/10], Loss: 0.0638, Training Accuracy: 97.62%, Validation Accuracy: 91.16%
Epoch [5/10], Loss: 0.0486, Training Accuracy: 98.22%, Validation Accuracy: 91.25%
Epoch [6/10], Loss: 0.0386, Training Accuracy: 98.67%, Validation Accuracy: 91.57%
Epoch [7/10], Loss: 0.0312, Training Accuracy: 98.93%, Validation Accuracy: 91.56%
Epoch [8/10], Loss: 0.0264, Training Accuracy: 99.13%, Validation Accuracy: 91.64%
Epoch [9/10], Loss: 0.0230, Training Accuracy: 99.28%, Validation Accuracy: 91.52%
Epoch [10/10], Loss: 0.0207, Training Accuracy: 99.33%, Validation Accuracy: 91.54%
```

Evaluation:

1. I implemented a confusion matrix for a comprehensive evaluation of the model's performance across different classes.
2. The confusion matrix showed precision and recall values that varied among classes, indicating areas for improvement.



Regularization:

1. To further enhance model performance and reduce overfitting, I applied dropout regularization, which helped maintain accuracy during training.

```
NetDropout(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.25, inplace=False)
  (fc1): Linear(in_features=3136, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

2. Enhanced result after regularization

```
Epoch [1/5], Loss: 0.0195, Training Accuracy: 99.41%, Validation Accuracy: 91.54%
Epoch [2/5], Loss: 0.0195, Training Accuracy: 99.41%, Validation Accuracy: 91.54%
Epoch [3/5], Loss: 0.0195, Training Accuracy: 99.41%, Validation Accuracy: 91.54%
Epoch [4/5], Loss: 0.0195, Training Accuracy: 99.41%, Validation Accuracy: 91.54%
Epoch [5/5], Loss: 0.0195, Training Accuracy: 99.41%, Validation Accuracy: 91.54%
```

3. Overall accuracy and Loss

```
Overall Validation Loss: 0.5887
Overall Validation Accuracy: 91.54%
```

Comparison of Results

First Attempt:

- **Batch Size**: 4
- **Optimizer**: Adam
- **Model**: CNN
- **Layers**: 2
- **Learning Rate**: 0.001
- **Epochs**: 10

Results:

- Epoch [1/10]: Loss: 0.1456, Training Accuracy: 94.56%, Validation Accuracy: 90.30%
- Epoch [10/10]: Loss: 0.0207, Training Accuracy: 99.33%, Validation Accuracy: 91.54%
- **Overall Validation Loss**: 0.5887
- **Overall Validation Accuracy**: 91.54%

Second Attempt:

- **Batch Size**: 64
- **Optimizer**: SGD
- **Model**: Updated CNN model with additional layer
- **Layers**: 3
- **Learning Rate**: 0.01
- **Epochs**: 10

Results: Very poor performance.

- Loss: 2.3151, Accuracy: 8.66%

Third Attempt:

- **Batch Size**: 128
- **Optimizer**: Adam
- **Model**: Updated CNN model with additional layer
- **Layers**: 3
- **Learning Rate**: 0.01
- **Epochs**: 5
- **Dataset Type**: Numpy Array

Results: Still poor performance.

- Loss: 0.4903, Accuracy: 86.90%
- Subsequent losses and accuracies fluctuated but did not improve significantly.