

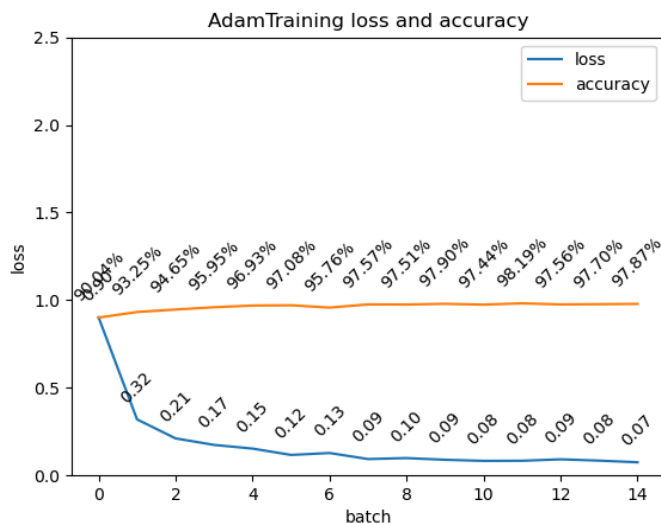
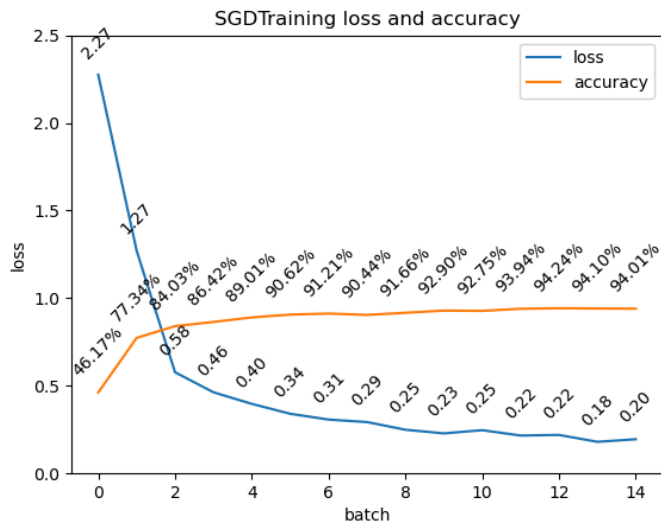
# CV assignment 3

## 1.1 Answer the questions:

		Given a <b>batch_size=16</b> , what are the shapes of the input and output for the 7 specified layers?	How many trainable parameters does each layer contain?
0	Input image: 1x28x28	input size 16*1*28*28	
1	Conv layer: <ul style="list-style-type: none"> <li>kernel_size: 5x5</li> <li>out_channels: 16</li> <li>activation: ReLU</li> </ul>	# convolutional layer 1 # output = $28 - 5 + 1 = 24$ # (16, 1, 28, 28) -> (16, 6, 24, 24)	$(1*5*5+1)*6 = 156$
2	Max pooling: <ul style="list-style-type: none"> <li>kernel_size: 2x2</li> </ul>	# pooling layer 1 (2,2) # output = $24/2 = 12$ # (16, 6, 24, 24) -> (16, 6, 12, 12)	0
3	Conv layer: <ul style="list-style-type: none"> <li>kernel_size: 3x3</li> <li>out_channels: 32</li> <li>activation: ReLU</li> </ul>	# convolutional layer 2 # output = $12 - 3 + 1 = 10$ # (16, 6, 12, 12) -> (16, 32, 10, 10)	$(6*3*3+1)*32 = 1760$
4	Max pooling: <ul style="list-style-type: none"> <li>kernel_size: 2x2</li> </ul>	#pooling layer 2 (2,2) # output = $10/2 = 5$ # (16, 32, 10, 10) -> (16, 32, 5, 5)	0
5	Conv layer: <ul style="list-style-type: none"> <li>kernel_size: 1x1</li> <li>out_channels: 16</li> <li>activation: ReLU</li> </ul>	# convolutional layer 3 # output = $5 - 1 + 1 = 5$ # (16, 32, 5, 5) -> (16, 16, 5, 5)	$(32*1*1+1)*16 = 528$
6	FC layer: <ul style="list-style-type: none"> <li>in_features: <b>16 * 5 * 5 = 400</b></li> <li>out_features: 64</li> <li>activation: ReLU</li> </ul>	FC layer1: input = (16,(16*5*5)) = (16,400) output = (16, 64)	$400*64+64 = 25664$
7	FC layer: <ul style="list-style-type: none"> <li>out_features: <b>10</b></li> <li>activation: None</li> </ul>	FC layer2: input =(16, 64) output = (16, 10) (label have 10)	$64*10+10 = 650$

## 1.2 Training a CNN to Recognize Hand-written Digits in the MNIST Dataset.

b. Experiment with different optimizers such as Adam in addition to the provided SGD optimizer, and **discuss** their impact on the final results. (1 point)



SGD Optimizer: Final loss: 0.20 Final accuracy: 94.01%

Adam Optimizer: Final loss: 0.07 Final accuracy: 97.87%

Adam computes adaptive learning rates for each parameter. It is generally considered to be more effective than SGD for many deep learning tasks.

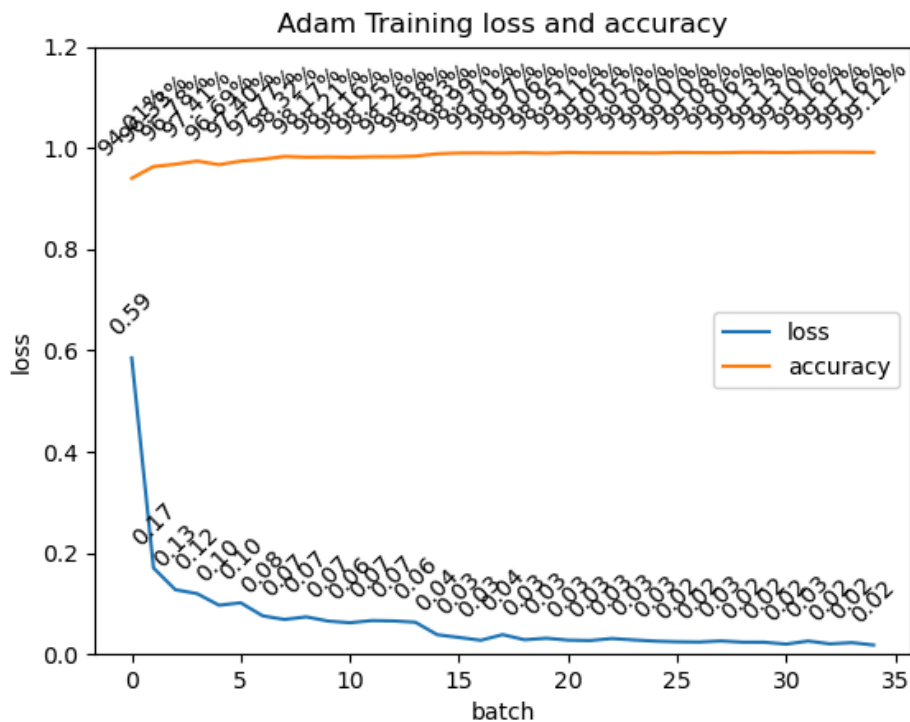
The Adam optimizer outperforms the SGD optimizer in this case. **The final loss is lower, and the final accuracy is higher** when using the Adam optimizer.

This is due to the adaptive nature of the Adam optimizer, which adjusts the learning rate for each parameter based on the estimates of the first and second moments of the gradients.

This can lead to faster convergence and better performance, especially for complex models or tasks with noisy gradients.

d: Set appropriate parameters (e.g., modify the optimizer, learning rate, epochs, etc.) to ensure that the model achieves an accuracy higher than 95% on the test set. Print the training accuracy, testing accuracy and learning rate every 1000 mini-batches. Save the model to the file `model1.pth`. (2 points)

the final result reached 99.12%



screenshot

```

98
99 if drawfigure == True:
100     plt.plot(loss_list, label='loss')
101     plt.plot(accuracy_list, label='accuracy')
102     plt.xlabel('batch')
103     plt.ylabel('loss')
104     plt.ylim(0, 1.2)
105     plt.title(table_name+' Training loss and accuracy')
106     plt.legend()
107     for i, acc in enumerate(accuracy_list):
108         plt.annotate(
109             f'{acc*100:.2f}%',
110             (i, acc),
111             textcoords="offset points",
112             xytext=(0,10),
113             ha='center',
114             rotation=45
115         )
116     for i, loss in enumerate(loss_list):
117         plt.annotate(

```

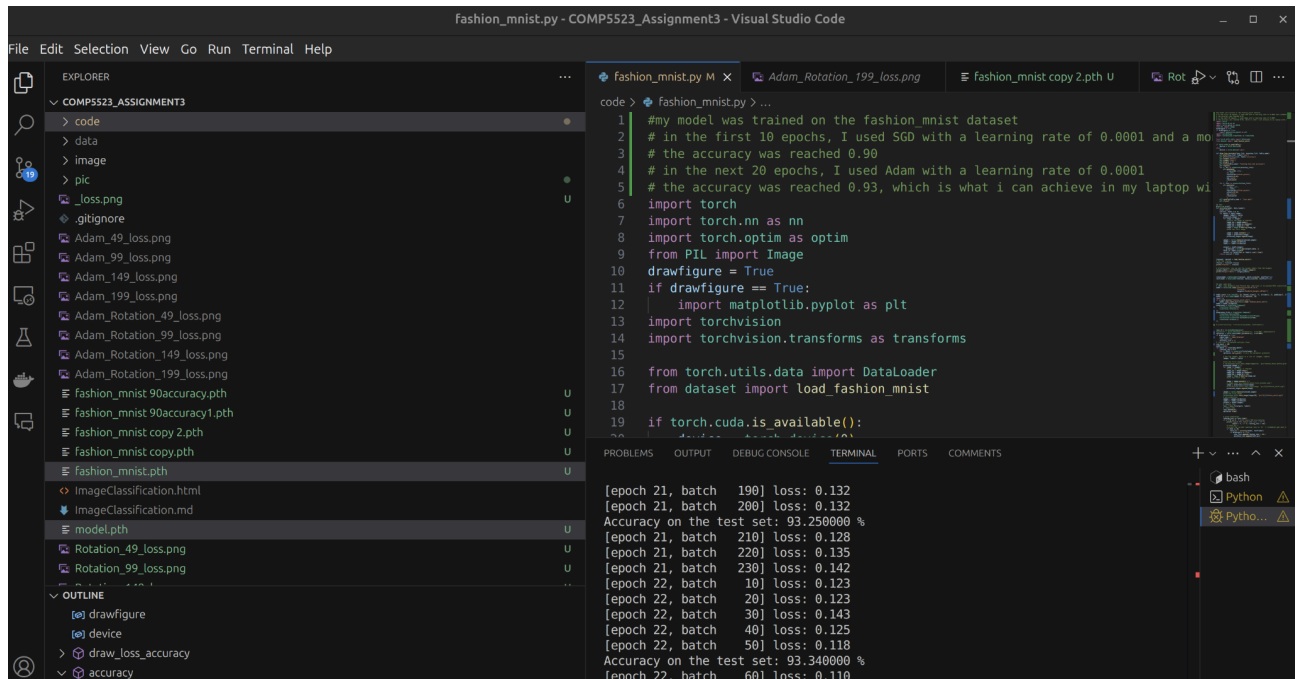
```

GroundTruth: 7 - seven 2 - two 1 - one 0 - zero 4 - four 1 - one 4 - four 9 - nine
Prediction: 7 - seven 2 - two 1 - one 0 - zero 4 - four 1 - one 4 - four 9 - nine
MESA-LOADER: failed to open radeonsi: /usr/lib/dri/radeonsi.dri.so: cannot open shared object file
: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\${ORIGIN}/dri:/usr/lib/d
ri, suffix_dri)
failed to load driver: radeonsi
MESA-LOADER: failed to open radeonsi: /usr/lib/dri/radeonsi.dri.so: cannot open shared object file
: No such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\${ORIGIN}/dri:/usr/lib/d
ri, suffix_dri)
failed to load driver: radeonsi
MESA-LOADER: failed to open swrast: /usr/lib/dri/swrast.dri.so: cannot open shared object file: No
such file or directory (search paths /usr/lib/x86_64-linux-gnu/dri:\${ORIGIN}/dri:/usr/lib/dri,
suffix_dri)
Accuracy on the test set: 99.030000 %
(cva3) (base) lzq@lqzq-ASUS-TUF-Gaming-A15-FA507XV-FA507XV:~/workspace/COMP5523_Assignment3$

```

## Bonus

my model was trained on the fashion\_mnist dataset, in the first 10 epochs, I used SGD with a learning rate of 0.0001 and a momentum of 0.9, the accuracy was reached 0.90, in the next 20 epochs, I used Adam with a learning rate of 0.0001, the accuracy was reached 0.93



```
code > fashion_mnist.py > ...
1 #my model was trained on the fashion_mnist dataset
2 # in the first 10 epochs, I used SGD with a learning rate of 0.0001 and a mo
3 # the accuracy was reached 0.90
4 # in the next 20 epochs, I used Adam with a learning rate of 0.0001
5 # the accuracy was reached 0.93, which is what i can achieve in my laptop wi
6 import torch
7 import torch.nn as nn
8 import torch.optim as optim
9 from PIL import Image
10 drawfigure = True
11 if drawfigure == True:
12     import matplotlib.pyplot as plt
13 import torchvision
14 import torchvision.transforms as transforms
15
16 from torch.utils.data import DataLoader
17 from dataset import load_fashion_mnist
18
19 if torch.cuda.is_available():
20     device = torch.device('cuda:0')
21 else:
22     device = torch.device('cpu')
23
24 # Data augmentation
25 transform = transforms.Compose([
26     transforms.RandomHorizontalFlip(),
27     transforms.RandomRotation(10),
28     transforms.ToTensor(),
29 ])
30
31 # Load data
32 train_loader = DataLoader(dataset=train_data_loader, batch_size=batch_size, shuffle=True)
33 test_loader = DataLoader(dataset=test_data_loader, batch_size=batch_size, shuffle=True)
34
35 # Model
36 model = nn.Sequential(
37     nn.Conv2d(1, 10, kernel_size=5),
38     nn.MaxPool2d(2),
39     nn.Conv2d(10, 20, kernel_size=5),
40     nn.MaxPool2d(2),
41     nn.Flatten(),
42     nn.Linear(1000, 10)
43 )
44
45 # Optimizer
46 optimizer = optim.Adam(model.parameters())
47
48 # Training
49 for epoch in range(1, 30):
50     for batch_idx, (data, target) in enumerate(train_loader):
51         data, target = data.to(device), target.to(device)
52         optimizer.zero_grad()
53         output = model(data)
54         loss = nn.CrossEntropyLoss()(output, target)
55         loss.backward()
56         optimizer.step()
57
58     # Test
59     test_loss, test_acc = test(model, test_loader, device)
60     print('Epoch: %d, batch: %d, loss: %f, accuracy: %f' % (epoch, batch_idx, test_loss, test_acc))
61
62 # Save model
63 torch.save(model.state_dict(), 'fashion_mnist.pth')
64
65 # Draw figure
66 if drawfigure == True:
67     plt.figure()
68     plt.plot(epochs, test_loss, 'b', label='Test Loss')
69     plt.plot(epochs, test_acc, 'r', label='Test Accuracy')
70     plt.legend()
71     plt.show()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

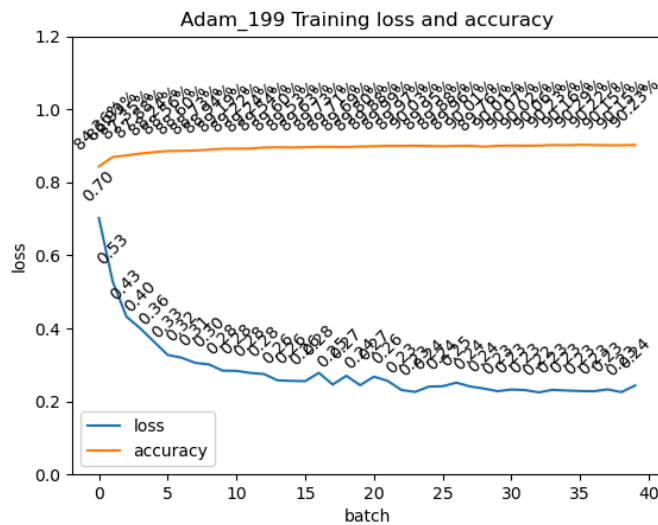
```
bash
Python
Python...
```

[epoch 21, batch 190] loss: 0.132  
[epoch 21, batch 200] loss: 0.132  
Accuracy on the test set: 93.250000 %  
[epoch 21, batch 210] loss: 0.128  
[epoch 21, batch 220] loss: 0.135  
[epoch 21, batch 230] loss: 0.142  
[epoch 22, batch 10] loss: 0.123  
[epoch 22, batch 20] loss: 0.123  
[epoch 22, batch 30] loss: 0.143  
[epoch 22, batch 40] loss: 0.125  
[epoch 22, batch 50] loss: 0.118  
Accuracy on the test set: 93.340000 %  
[epoch 22, batch 60] loss: 0.110

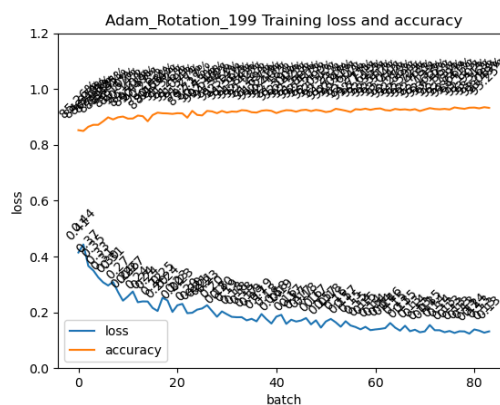
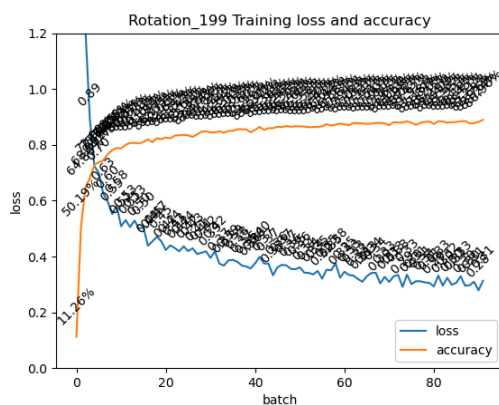
## Bonus 2.2-2

Experiment with data augmentation techniques such as `torchvision.transforms.RandomHorizontalFlip` and `torchvision.transforms.RandomRotation`, **discuss** the impact of these data augmentation parameters on the final

classification accuracy. (0.5 point)



without rotation



with rotation train phase1 and train phase 2

from the image we can find that, without random rotation and random mirror, the model can reach 0.90 in only 40 batches (256 image for each batch) however, with the rotation and mirror, we need 90(in phase1, above left)+20(in phase2, above right)=110 batches to reach similar accuracy. **So, the rotation and mirror will slow the training compared to similar accuracy.**

**But with the rotation, the model have potential to be trained a better performance,** With 60 more batches, the accuracy will reach 93%, and still have potential to reach higher if the following train would be applied.

**generally, it can provide a more general model despite it requiring more training consumption.**