ENGN8636 Lab2 Report

Student Name: Jieli Zheng

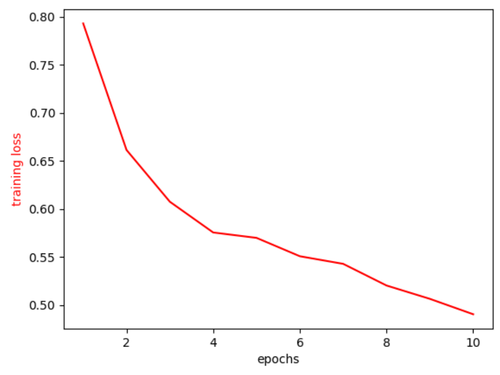
Student ID: u6579712

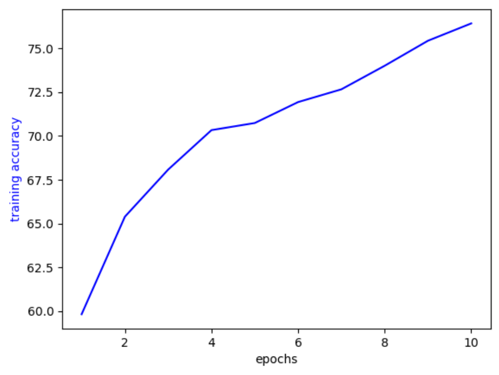
1 Custom Data Loader

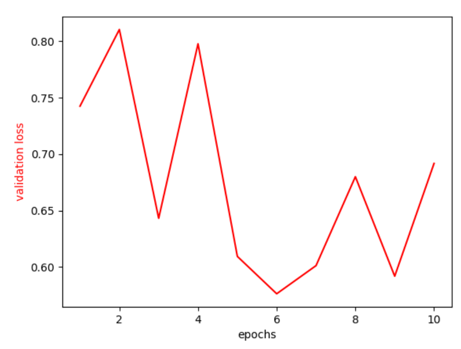
As shown in Appendix 1.1 and 1.2, I use the standard PyTorch Dataloader as well as a customized dataset class for the dog-cat dataset. PyTorch has standard random split function, and I make use of it to split training and validation data in 1.1. The dog or cat one-hot label is generated in the customized dataset class as shown in 1.2. The training and validation data apply both Q4(resize and normalize) and Q5(randomly flip, zero-padding and randomly crop) transformations, and the test data only apply Q4(resize and normalize) transformation as shown in 1.1. I make use of compose() in standard transformation library of PyTorch to compose all transformations.

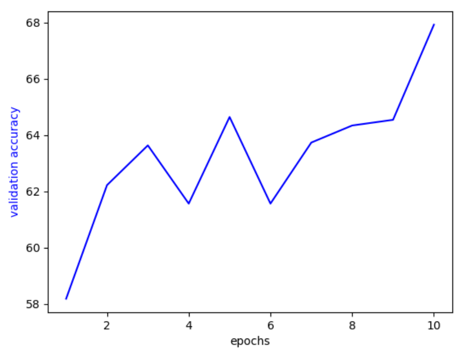
2 Implement a CNN

The implemented CNN architecture is shown in Appendix 2.1. From 2.2, the loss function I use is the cross-entropy loss via standard PyTorch library. The optimizer is Adam optimizer with a constant learning rate 0.005. The number of epochs is set to 10 empirically. Batch size is set to 50 because larger batch size will easily cause CUDA out of memory error and I run this network on an RTX2080 which has only 8GB capacity. Other training details can be found in 2.3,2.4. The model result plots are as follows.



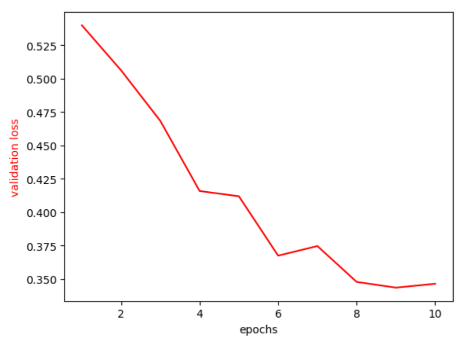


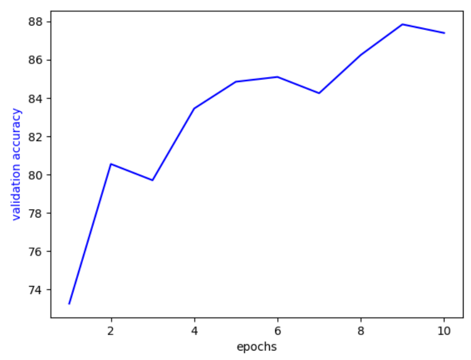


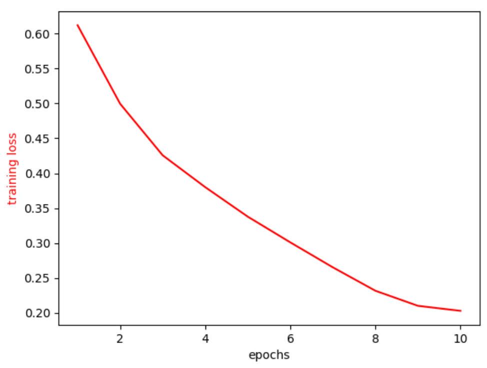


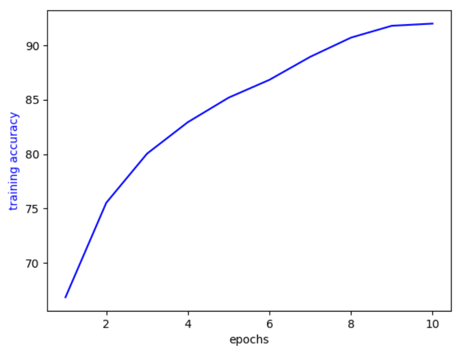
The final test accuracy is 68.622%, test loss is 0.57877. From the plot above, I find that although the training loss is decreasing among epochs, the validation loss is increasing. This result indicates that the model has the overfitting problem. The performance of validation accuracy is very unstable during training because the batch size is too small, and the learning rate is too large. Therefore, I will try to improve this in the next question.

For question 2.5, firstly since the network suffers from the overfitting problem, I try to firstly reduce the last fully connected layer to 512. To find a better learning rate, I use cosine decay learning function to dynamically adjust learning rate, its implementation is shown in 2.5.2. To make the validation loss curve more stable, I set the batch size to 100 to have stable validation curve. The network architecture is slightly modified as some batchnorm layers are replaced by the dropout layer to reduce the overfitting issue. The new architecture can be found in Appendix 2.5.1. The result of modified network is as follows.





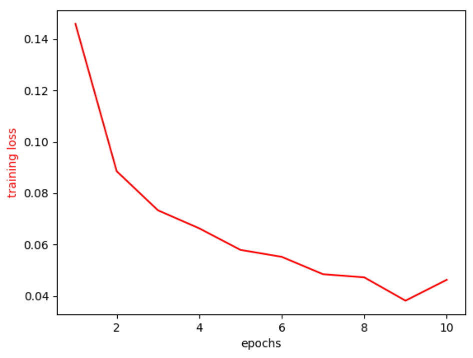


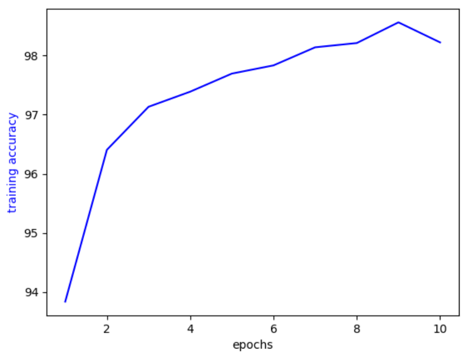


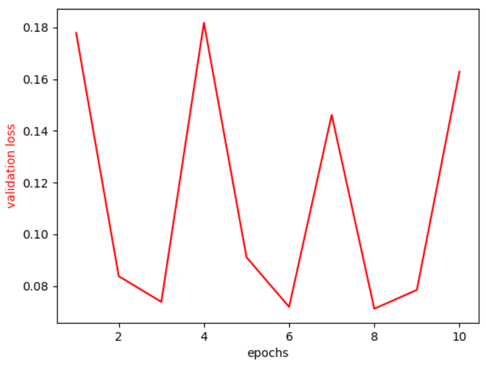
The final test accuracy and loss of modified model are 85.275% and 0.3491. From the plots above, I find that the validation loss curve is stable, and the performance of the modified model is much better, and the model gets rid of overfitting problem.

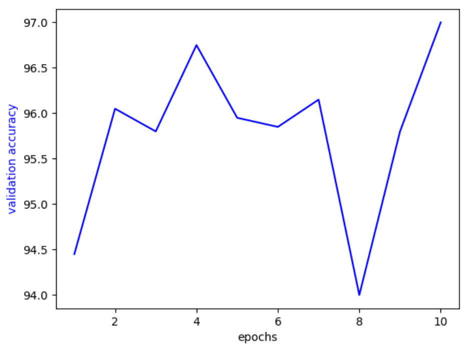
3 Transfer Learning

The backbone network I choose is ResNet-18. Since ResNet-18 is preloaded in TorchVision online library, I can import and set pretrained to True to make use of it directly. In order to make it work for our task here, I modified and connect an extra fully connected layer to the end of ResNet-18 as is shown in 3.1. After this modification, the pretrained model uses nearly the same script as Q2 does as is shown in 3.2,3.3 and 3.4. The result plots are as below.





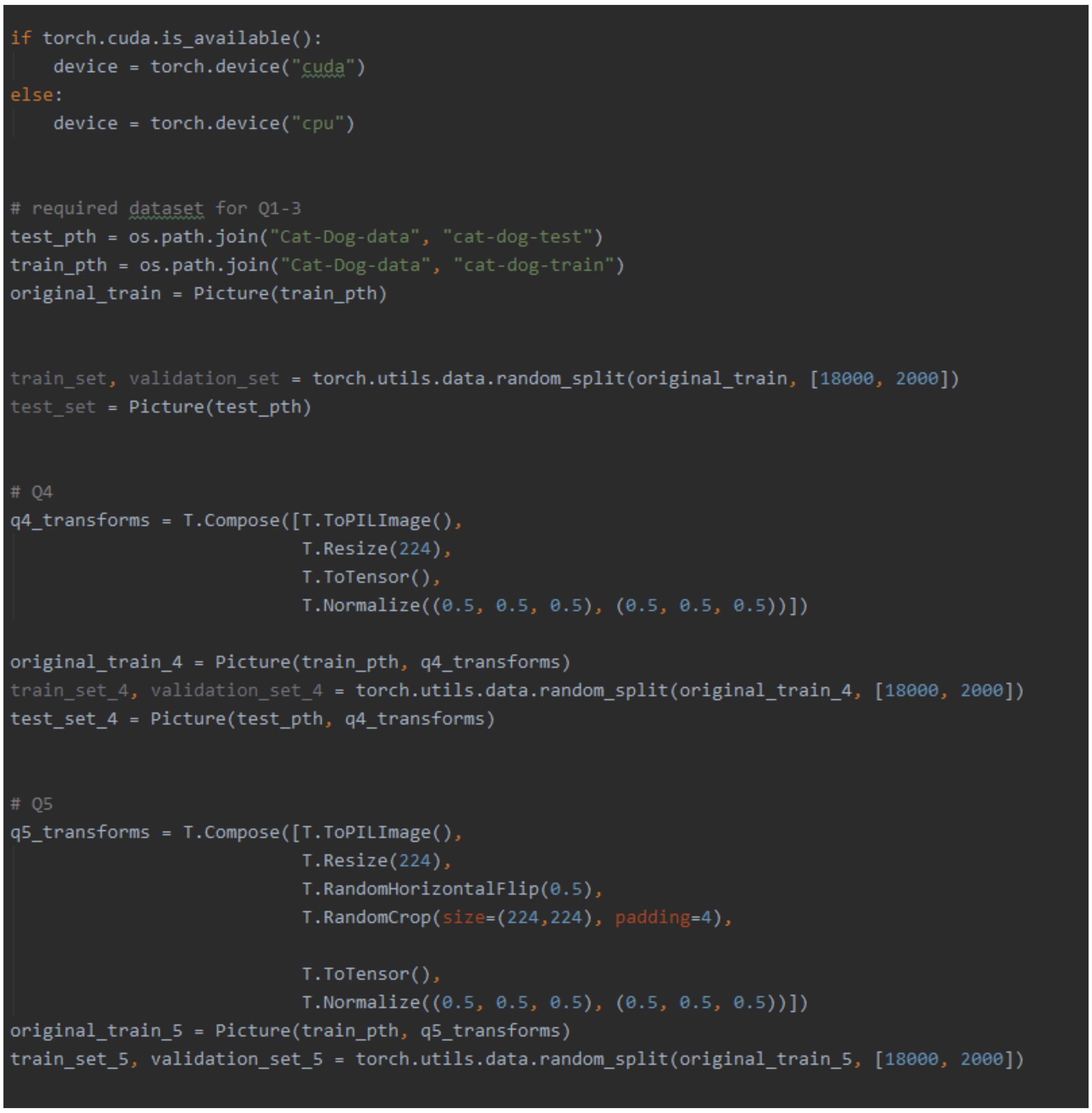




The final test accuracy and loss are 97.1% and 0.1628. As shown above, the training loss is converged in epoch 9 and a bit overfitting in epoch 10. The validation loss curve is stable around 0.12 although it looks unstable due to the scaling of the plot. Overall, the pretrained model has dominant performance compared to the model in task 2.4. The reason is that Res-block can prevent overfitting in deeper CNN training and learn more powerful representation from the data and the pretrained network is trained under larger dataset. Also, the pretrained ResNet-18 is more efficient than the original CNN network. Therefore, larger batch size can be set to get more stable results.

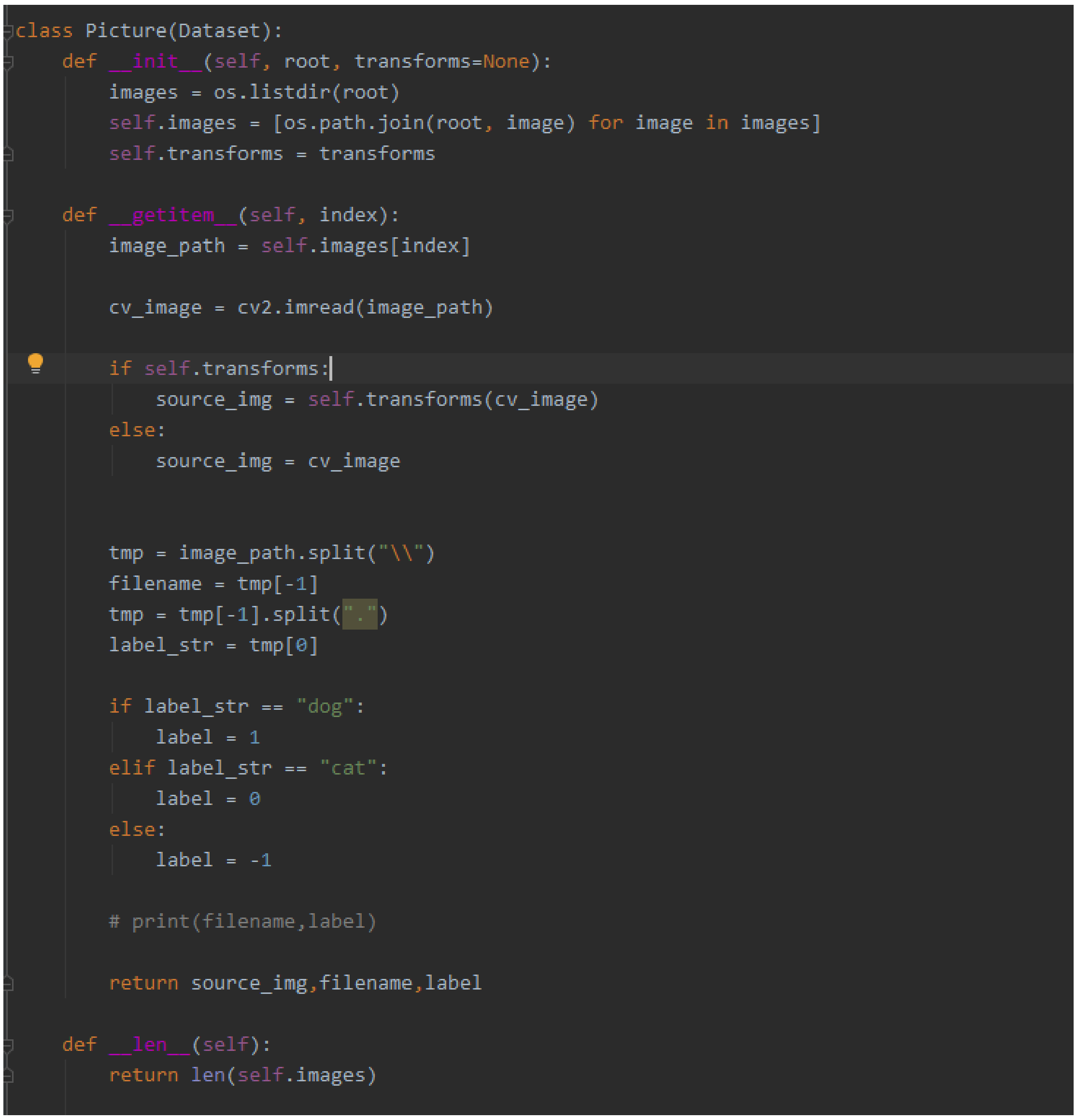
Appendix

Q1



```
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")


# required dataset for Q1-3
test_pth = os.path.join("Cat-Dog-data", "cat-dog-test")
train_pth = os.path.join("Cat-Dog-data", "cat-dog-train")
original_train = Picture(train_pth)


train_set, validation_set = torch.utils.data.random_split(original_train, [18000, 2000])
test_set = Picture(test_pth)


# Q4
q4_transforms = T.Compose([T.ToPILImage(),
                           T.Resize(224),
                           T.ToTensor(),
                           T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

original_train_4 = Picture(train_pth, q4_transforms)
train_set_4, validation_set_4 = torch.utils.data.random_split(original_train_4, [18000, 2000])
test_set_4 = Picture(test_pth, q4_transforms)


# Q5
q5_transforms = T.Compose([T.ToPILImage(),
                           T.Resize(224),
                           T.RandomHorizontalFlip(0.5),
                           T.RandomCrop(size=(224,224), padding=4),

                           T.ToTensor(),
                           T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
original_train_5 = Picture(train_pth, q5_transforms)
train_set_5, validation_set_5 = torch.utils.data.random_split(original_train_5, [18000, 2000])
```

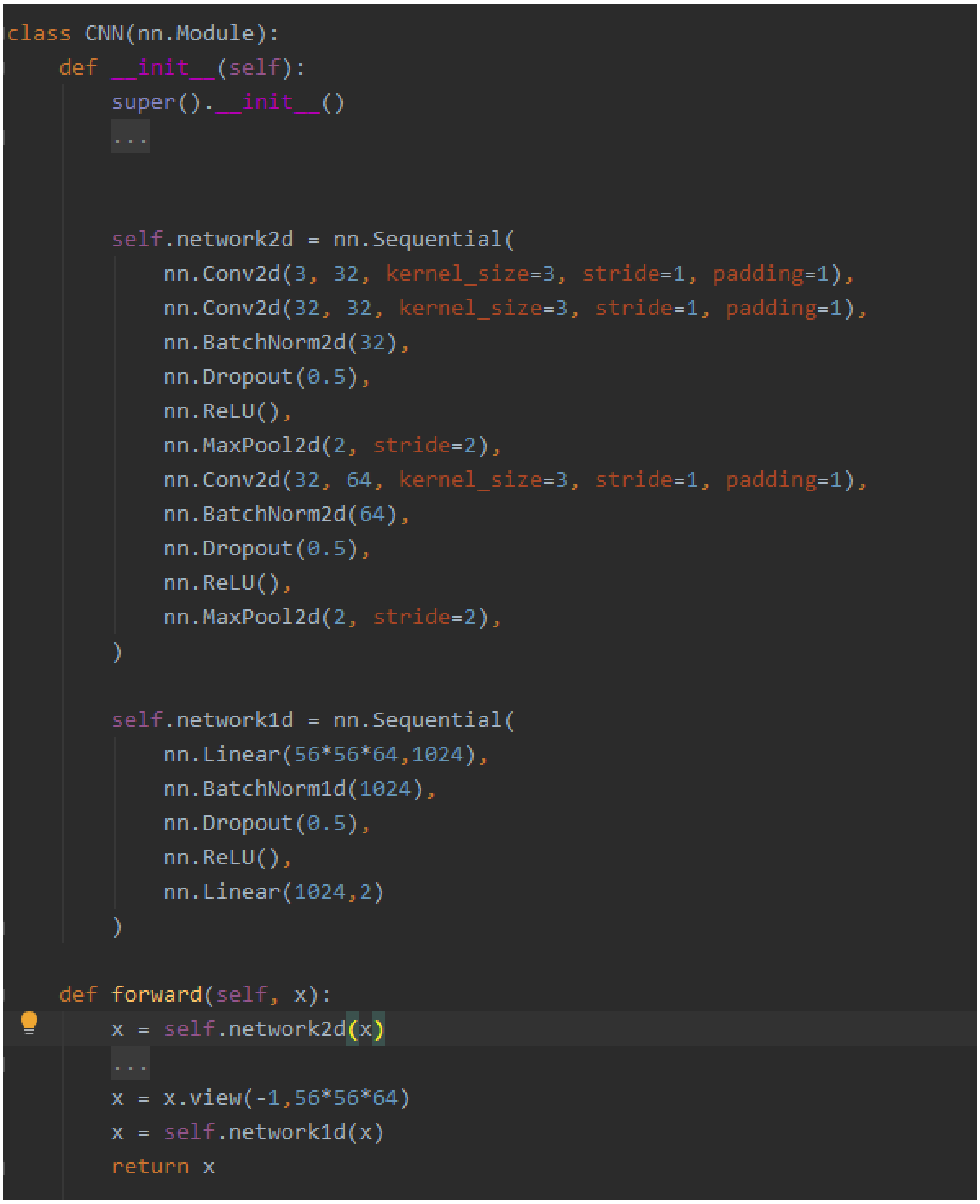
1.1 Dataloader as well as transformations.



```python
class Picture(Dataset):
    def __init__(self, root, transforms=None):
        images = os.listdir(root)
        self.images = [os.path.join(root, image) for image in images]
        self.transforms = transforms

    def __getitem__(self, index):
        image_path = self.images[index]

        cv_image = cv2.imread(image_path)

        if self.transforms:
            source_img = self.transforms(cv_image)
        else:
            source_img = cv_image


        tmp = image_path.split("\\")
        filename = tmp[-1]
        tmp = tmp[-1].split(".")
        label_str = tmp[0]

        if label_str == "dog":
            label = 1
        elif label_str == "cat":
            label = 0
        else:
            label = -1

        # print(filename,label)

        return source_img,filename,label

    def __len__(self):
        return len(self.images)
```

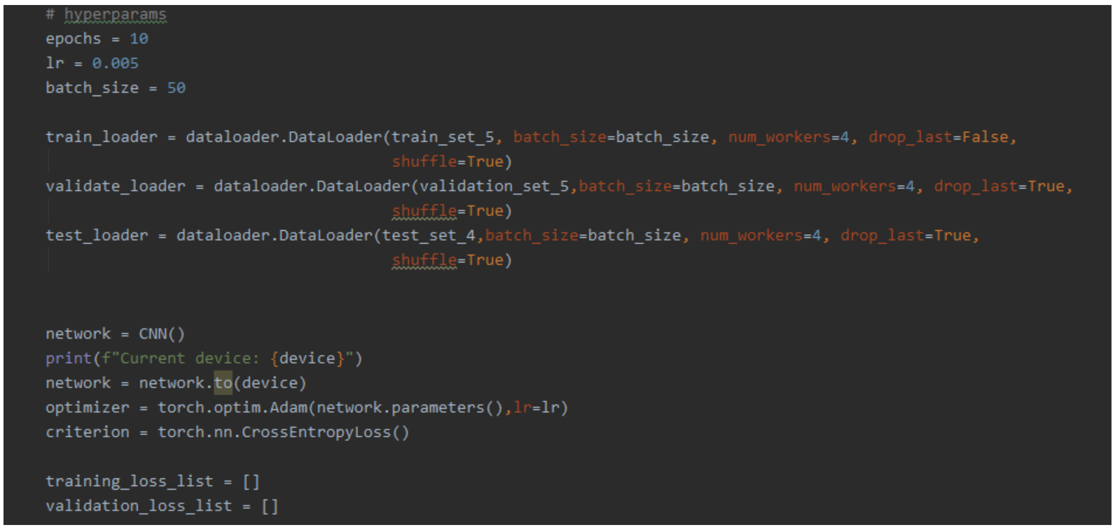
1.2 customized class of picture dataset

Q2



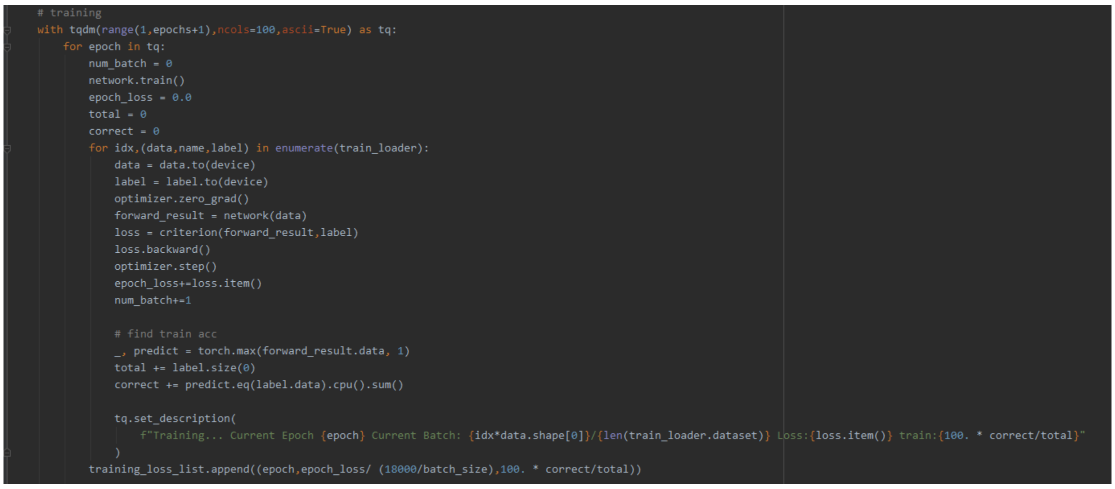
```python
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        ...


        self.network2d = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
        )

        self.network1d = nn.Sequential(
            nn.Linear(56*56*64,1024),
            nn.BatchNorm1d(1024),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(1024,2)
        )

    def forward(self, x):
        x = self.network2d(x)
        ...
        x = x.view(-1,56*56*64)
        x = self.network1d(x)
        return x
```

2.1 Network Architecture



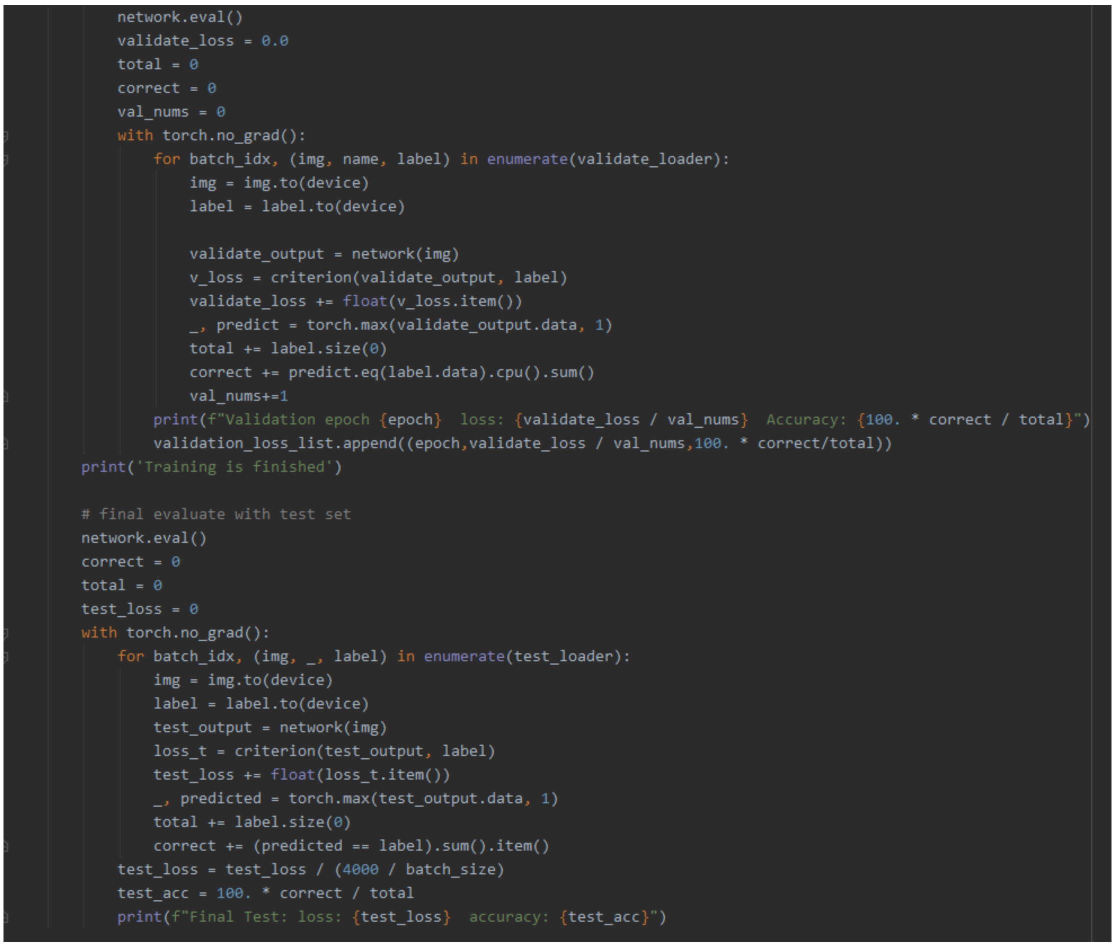
```python
# hyperparams
epochs = 10
lr = 0.005
batch_size = 50

train_loader = dataloader.DataLoader(train_set_5, batch_size=batch_size, num_workers=4, drop_last=False,
                                     shuffle=True)
validate_loader = dataloader.DataLoader(validation_set_5,batch_size=batch_size, num_workers=4, drop_last=True,
                                     shuffle=True)
test_loader = dataloader.DataLoader(test_set_4,batch_size=batch_size, num_workers=4, drop_last=True,
                                     shuffle=True)


network = CNN()
print(f"Current device: {device}")
network = network.to(device)
optimizer = torch.optim.Adam(network.parameters(),lr=lr)
criterion = torch.nn.CrossEntropyLoss()

training_loss_list = []
validation_loss_list = []
```

2.2 training script part1: preparation



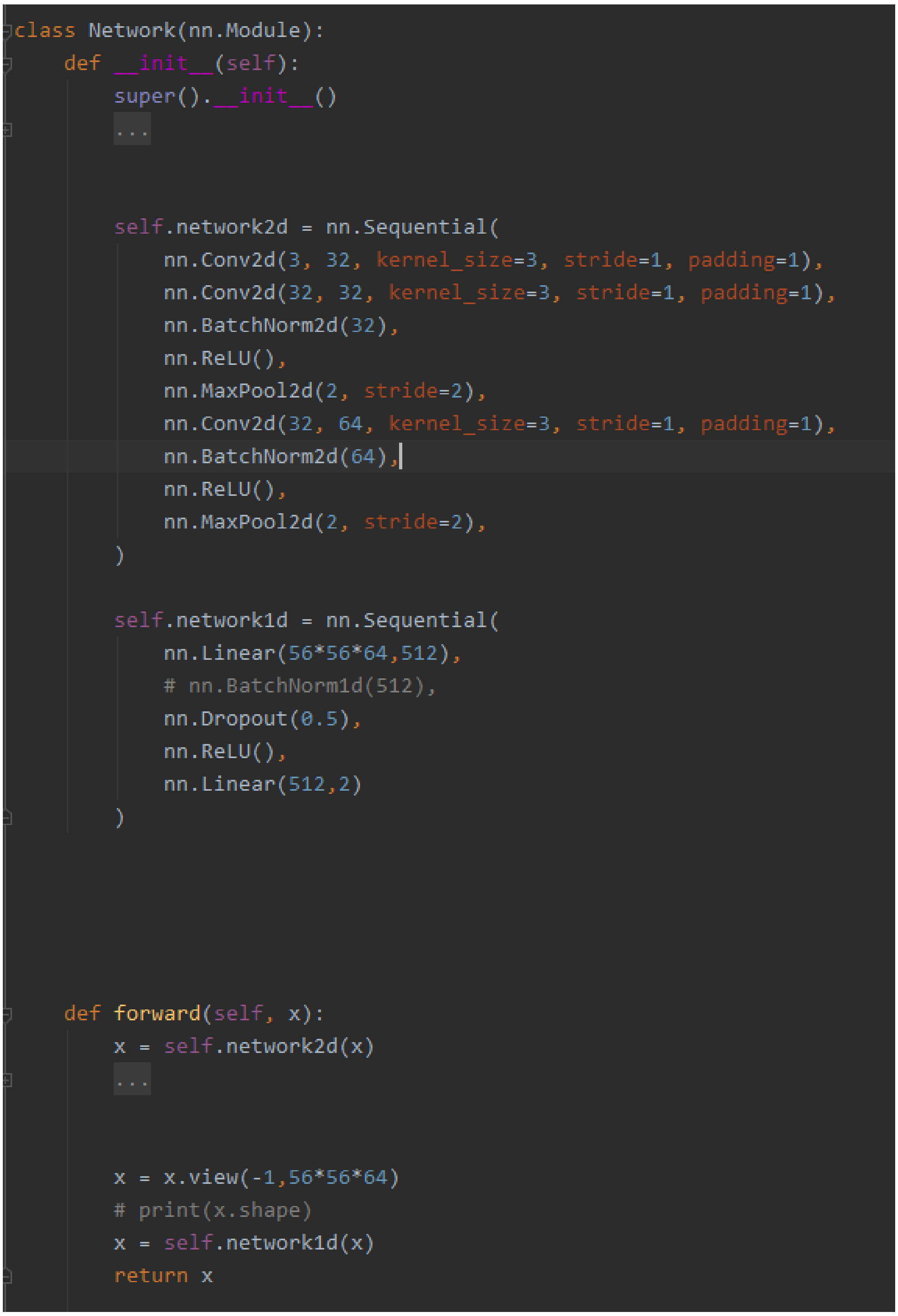
```python
# training
with tqdm(range(1,epochs+1),ncols=100,ascii=True) as tq:
    for epoch in tq:
        num_batch = 0
        network.train()
        epoch_loss = 0.0
        total = 0
        correct = 0
        for idx,(data,name,label) in enumerate(train_loader):
            data = data.to(device)
            label = label.to(device)
            optimizer.zero_grad()
            forward_result = network(data)
            loss = criterion(forward_result,label)
            loss.backward()
            optimizer.step()
            epoch_loss+=loss.item()
            num_batch+=1

            # find train acc
            _, predict = torch.max(forward_result.data, 1)
            total += label.size(0)
            correct += predict.eq(label.data).cpu().sum()

            tq.set_description(
                f"Training... Current Epoch {epoch} Current Batch: {idx*data.shape[0]}/{len(train_loader.dataset)} Loss:{loss.item()} train:{100. * correct/total}"
            )
        training_loss_list.append((epoch,epoch_loss/ (18000/batch_size),100. * correct/total))
```

2.3 training script part2: training by epochs



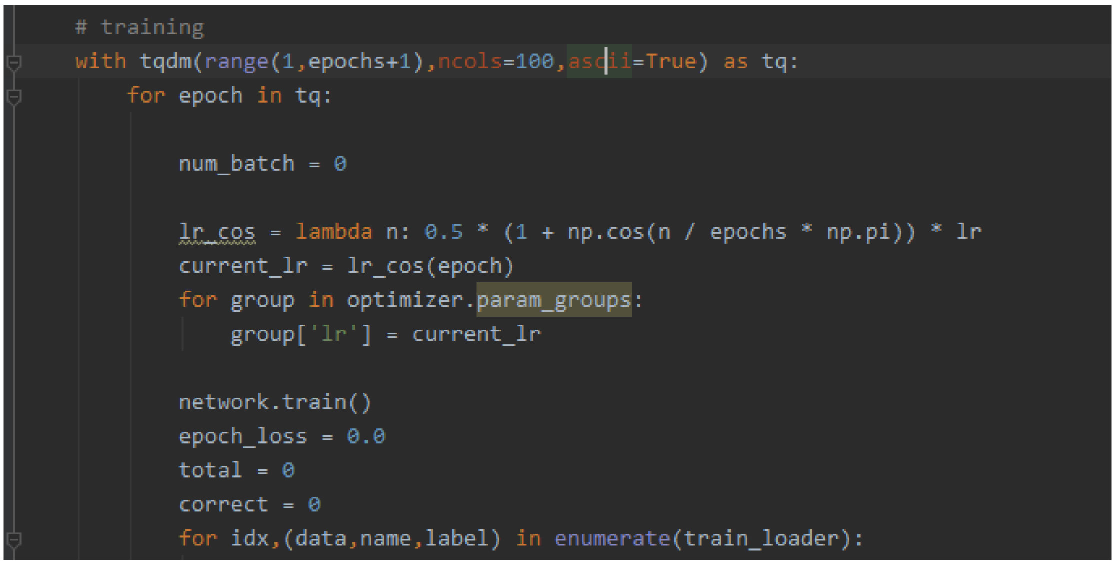
```python
        network.eval()
        validate_loss = 0.0
        total = 0
        correct = 0
        val_nums = 0
        with torch.no_grad():
            for batch_idx, (img, name, label) in enumerate(validate_loader):
                img = img.to(device)
                label = label.to(device)

                validate_output = network(img)
                v_loss = criterion(validate_output, label)
                validate_loss += float(v_loss.item())
                _, predict = torch.max(validate_output.data, 1)
                total += label.size(0)
                correct += predict.eq(label.data).cpu().sum()
                val_nums+=1
            print(f"Validation epoch {epoch}  loss: {validate_loss / val_nums}  Accuracy: {100. * correct / total}")
            validation_loss_list.append((epoch,validate_loss / val_nums,100. * correct/total))
    print('Training is finished')

    # final evaluate with test set
    network.eval()
    correct = 0
    total = 0
    test_loss = 0
    with torch.no_grad():
        for batch_idx, (img, _, label) in enumerate(test_loader):
            img = img.to(device)
            label = label.to(device)
            test_output = network(img)
            loss_t = criterion(test_output, label)
            test_loss += float(loss_t.item())
            _, predicted = torch.max(test_output.data, 1)
            total += label.size(0)
            correct += (predicted == label).sum().item()
        test_loss = test_loss / (4000 / batch_size)
        test_acc = 100. * correct / total
        print(f"Final Test: loss: {test_loss}  accuracy: {test_acc}")
```

2.4 training script part3: evaluation via validation during training, and final test



```python
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        ...


        self.network2d = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
        )

        self.network1d = nn.Sequential(
            nn.Linear(56*56*64,512),
            # nn.BatchNorm1d(512),
            nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(512,2)
        )




    def forward(self, x):
        x = self.network2d(x)
        ...


        x = x.view(-1,56*56*64)
        # print(x.shape)
        x = self.network1d(x)
        return x
```

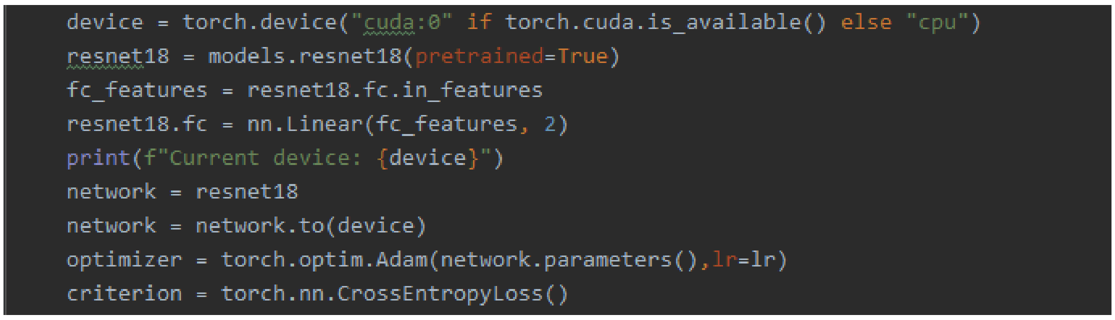
2.5.1 improved CNN architecture



```python
# training
with tqdm(range(1,epochs+1),ncols=100,ascii=True) as tq:
    for epoch in tq:

        num_batch = 0

        lr_cos = lambda n: 0.5 * (1 + np.cos(n / epochs * np.pi)) * lr
        current_lr = lr_cos(epoch)
        for group in optimizer.param_groups:
            group['lr'] = current_lr

        network.train()
        epoch_loss = 0.0
        total = 0
        correct = 0
        for idx,(data,name,label) in enumerate(train_loader):
```

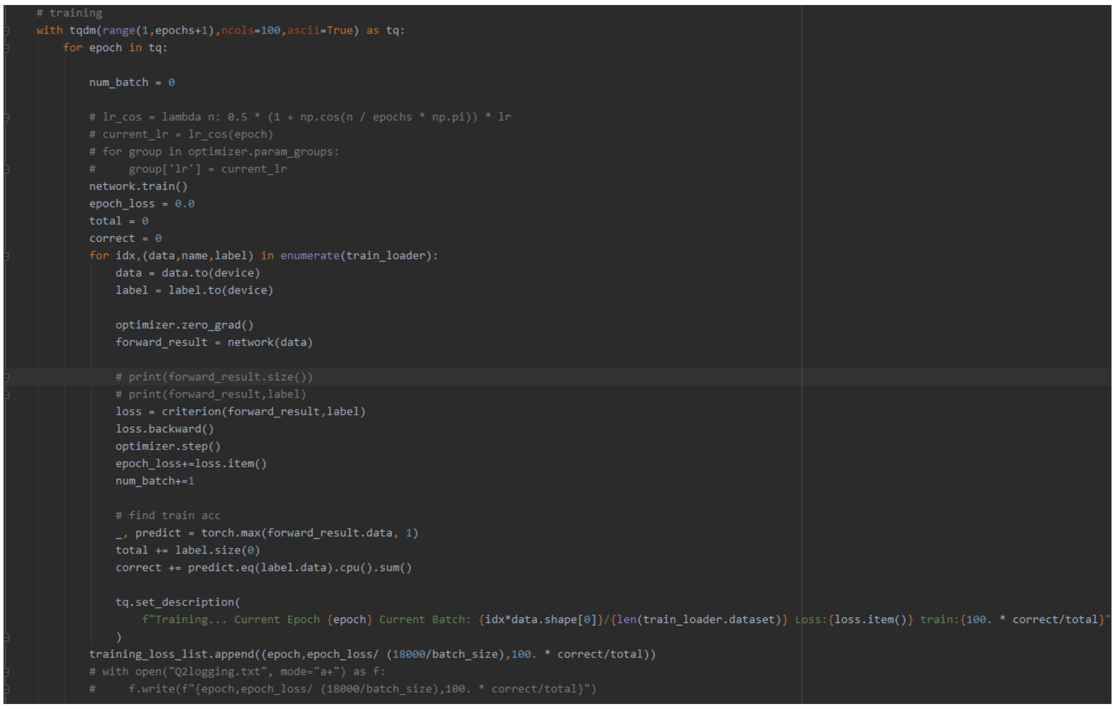
2.5.2 cosine learning rate approach

Q3



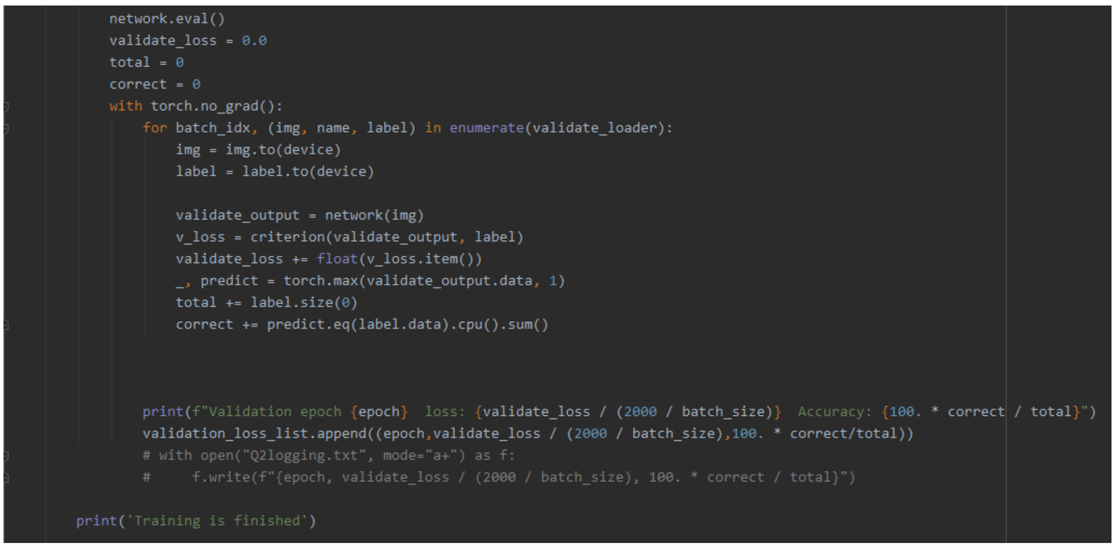
```python
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
resnet18 = models.resnet18(pretrained=True)
fc_features = resnet18.fc.in_features
resnet18.fc = nn.Linear(fc_features, 2)
print(f"Current device: {device}")
network = resnet18
network = network.to(device)
optimizer = torch.optim.Adam(network.parameters(),lr=lr)
criterion = torch.nn.CrossEntropyLoss()
```

3.1 preloaded ResNet-18



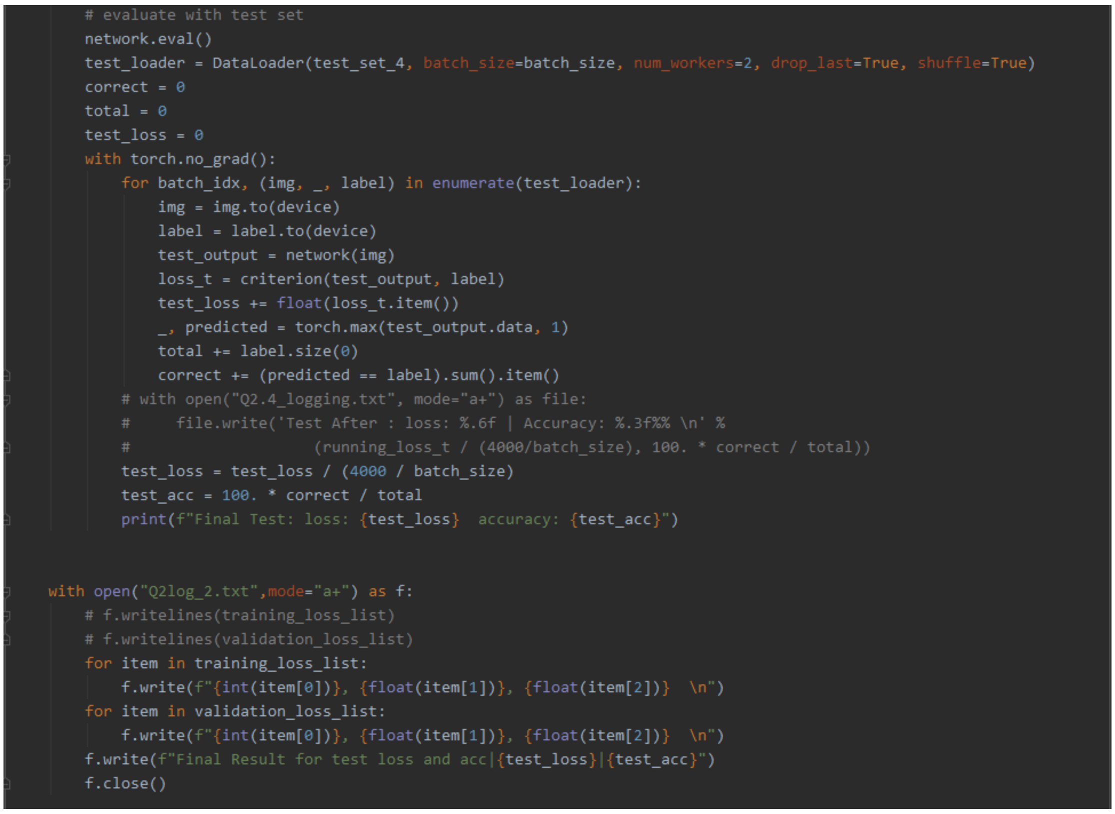
```python
# training
with tqdm(range(1,epochs+1),ncols=100,ascii=True) as tq:
    for epoch in tq:

        num_batch = 0

        # lr_cos = lambda n: 0.5 * (1 + np.cos(n / epochs * np.pi)) * lr
        # current_lr = lr_cos(epoch)
        # for group in optimizer.param_groups:
        #     group['lr'] = current_lr
        network.train()
        epoch_loss = 0.0
        total = 0
        correct = 0
        for idx,(data,name,label) in enumerate(train_loader):
            data = data.to(device)
            label = label.to(device)

            optimizer.zero_grad()
            forward_result = network(data)

            # print(forward_result.size())
            # print(forward_result,label)
            loss = criterion(forward_result,label)
            loss.backward()
            optimizer.step()
            epoch_loss+=loss.item()
            num_batch+=1

            # find train acc
            _, predict = torch.max(forward_result.data, 1)
            total += label.size(0)
            correct += predict.eq(label.data).cpu().sum()

            tq.set_description(
                f"Training... Current Epoch {epoch} Current Batch: {idx*data.shape[0]}/{len(train_loader.dataset)} Loss:{loss.item()} train:{100. * correct/total}"
            )
        training_loss_list.append((epoch,epoch_loss/ (18000/batch_size),100. * correct/total))
        # with open("Q2logging.txt", mode="a+") as f:
        #     f.write(f"{epoch,epoch_loss/ (18000/batch_size),100. * correct/total}")
```

3.2 pretrained approach part1



```
    network.eval()
    validate_loss = 0.0
    total = 0
    correct = 0
    with torch.no_grad():
        for batch_idx, (img, name, label) in enumerate(validate_loader):
            img = img.to(device)
            label = label.to(device)

            validate_output = network(img)
            v_loss = criterion(validate_output, label)
            validate_loss += float(v_loss.item())
            _, predict = torch.max(validate_output.data, 1)
            total += label.size(0)
            correct += predict.eq(label.data).cpu().sum()


        print(f"Validation epoch {epoch}  loss: {validate_loss / (2000 / batch_size)}  Accuracy: {100. * correct / total}")
        validation_loss_list.append((epoch,validate_loss / (2000 / batch_size),100. * correct/total))
        # with open("Q2logging.txt", mode="a+") as f:
        #     f.write(f"{epoch, validate_loss / (2000 / batch_size), 100. * correct / total}")

print('Training is finished')
```

3.3 pretrained approach part2



```
    # evaluate with test set
    network.eval()
    test_loader = DataLoader(test_set_4, batch_size=batch_size, num_workers=2, drop_last=True, shuffle=True)
    correct = 0
    total = 0
    test_loss = 0
    with torch.no_grad():
        for batch_idx, (img, _, label) in enumerate(test_loader):
            img = img.to(device)
            label = label.to(device)
            test_output = network(img)
            loss_t = criterion(test_output, label)
            test_loss += float(loss_t.item())
            _, predicted = torch.max(test_output.data, 1)
            total += label.size(0)
            correct += (predicted == label).sum().item()
        # with open("Q2.4_logging.txt", mode="a+") as file:
        #     file.write('Test After : loss: %.6f | Accuracy: %.3f%% \n' %
        #                    (running_loss_t / (4000/batch_size), 100. * correct / total))
        test_loss = test_loss / (4000 / batch_size)
        test_acc = 100. * correct / total
        print(f"Final Test: loss: {test_loss}  accuracy: {test_acc}")


with open("Q2log_2.txt",mode="a+") as f:
    # f.writelines(training_loss_list)
    # f.writelines(validation_loss_list)
    for item in training_loss_list:
        f.write(f"{int(item[0])}, {float(item[1])}, {float(item[2])}  \n")
    for item in validation_loss_list:
        f.write(f"{int(item[0])}, {float(item[1])}, {float(item[2])}  \n")
    f.write(f"Final Result for test loss and acc|{test_loss}|{test_acc}")
    f.close()
```

3.4 pretrained approach part3