بخش1:

متناسب با دیتا باید تصمیم گرفت در کل سرعت همگرایی متد های adaptive بیتشتر است و زمانی تاثیر بیشتری میگذارند که دیتاها ما نیز کم یا sparse باشند که در بین متدهای adaptive متد adam بیشتر از همه توصیه است باز اگرمدل عمیق پیچیده ای داشته باشیم متد های adaptive پیشنهاد می شود

بخش 2:

بصورت دستی نوشته و ضمیمه شده است اما ناقص است.

بخش 3:

```
transform = transforms.Compose(
    [
        # transforms.Resize((224,224)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomCrop(25),
        transforms.ToTensor(),
        transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
    ]
)
```

فلیپ کردن دیتاها به صورت افقی و عمودی با احتمال 50 درصد و کراپ کرد عکس ها بصورت $25*25$ و درننهایت عملیات نرمال سازی

```python
class Model(nn.Module):
  def __init__(self):
    super(Model, self).__init__()
    self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
    self.norm1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
    self.norm3 = nn.BatchNorm2d(128)
    self.conv4 = nn.Conv2d(128, 128, 3, padding=1)
    self.conv5 = nn.Conv2d(128,256,3, padding=1)
    self.norm5 = nn.BatchNorm2d(256)
    self.conv6 = nn.Conv2d(256,256,5, padding=1)
    self.dropout = nn.Dropout(p=0.05)
    self.fc1 = nn.Linear(256*2*2, 1024)
    self.fc2 = nn.Linear(1024, 512)
    self.fc3 = nn.Linear(512, 10)
  def forward(self, x):
    x = F.relu(self.norm1(self.conv1(x)))#1
    x = self.pool(F.relu(self.conv2(x)))#2
    x = F.relu(self.norm3(self.conv3(x)))#3
    x = self.dropout(self.pool(F.relu(self.conv4(x))))#4
    x = F.relu(self.norm5(self.conv5(x)))
    x = self.pool(F.relu(self.conv6(x)))
    # print(f"6: {x.shape}")
    x = x.view(-1, 256*2*2)
    x = self.dropout(x)
    x = F.relu(self.fc1(x))
    x = self.dropout(F.relu(self.fc2(x)))
    x = self.fc3(x)
    return x
```

مدلی که درتصویر سوال سه داده شده پیاده سازی شده شش لایه کانولشنالی و سه لایه فولی کانکتد به همراه dropout ، batch normalize و max pool هنگام تبدیل لایه کانولشنالی به فولی کانکتد ابعاد در اخرین لایه کانولشنالی محاسبه شد و ابعاد خروجی بصورت یک بعدی محاسبه و به ورودی لایه فولی کانکتد داده شد. طبق سوال از بهینه ساز adam و تابع ضرر CrossEntropyLoss نیز استفاده شد

```
# specify loss function
criterion = nn.CrossEntropyLoss()
# specify optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
```

و برای 2 epoch اموزش مدل انجام شد(برای بخش اموزش مدل از این لینک استفاده شد.)

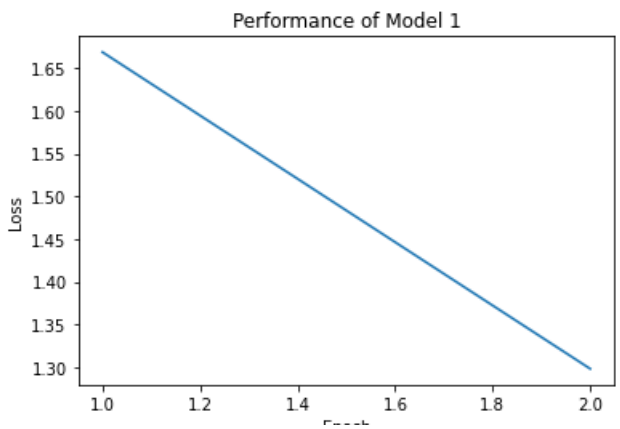
```
for data, target in trainloader:
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    train_loss += loss.item()*data.size(0)

train_loss = train_loss/len(trainloader.dataset)
train_losslist.append(train_loss)
```

در این تکه کد عملیات به روزرسانی وزن ها و محاسبه loss می پردازد

Loss محاسبه شده در دو epoch:

```
Epoch: 1        Training Loss: 1.668888
Epoch: 2        Training Loss: 1.297921
```



خروجی هم به صورت زیر بدست امد

```
Test Loss: 1.255724

Test Accuracy of plane: 34% (15/44)
Test Accuracy of   car: 71% (23/32)
Test Accuracy of  bird: 34% (13/38)
Test Accuracy of   cat: 32% (14/43)
Test Accuracy of  deer: 40% (20/49)
Test Accuracy of   dog: 59% (19/32)
Test Accuracy of  frog: 77% (41/53)
Test Accuracy of horse: 43% (14/32)
Test Accuracy of  ship: 81% (39/48)
Test Accuracy of truck: 62% (18/29)

Test Accuracy (Overall): 54% (216/400)
```