CIFAR problem Using PyTorch

```
# to check if CUDA is available:
train_on_gpu = torch.cuda.is_available()

# Load the data
transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((½,½,½),(½,½,½))
])

train_data = datasets.CIFAR10('data', train=True,
                              download=True,
                              transform=transform)
test_data = datasets.CIFAR10('data', train=False,
                             download=True,
                             transform=transform)
```

torchvision

data to validate

```
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]
```

0.2

to make batches

```
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=batch_size,
                                           sampler=train_sampler,
                                           num_workers=nw)
valid_loader = torch.utils.data.DataLoader(valid_data,
                                           batch_size=batch_size,
                                           sampler=valid_sampler,
                                           num_workers=nw)
test_loader = torch.utils.data.DataLoader(test_data,
                                          batch_size=batch_size,
                                          num_workers=nw)

# it is a classification task
classes = ['airplane', ...., 'truck']
```

model

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)   # sees 32x32x3
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)  # sees 16x16x16
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)  # sees 8x8x32
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(64*4*4, 500)
        self.fc2 = nn.Linear(500, 10)
        self.dropout = nn.Dropout(.25)
```

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))

    x = x.view(-1, 64*4*4)
    x = self.dropout(x)
    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = self.fc2(x)
    return x
```

use the defined model

```
model = Net()
if train_on_gpu: model.cuda()
```

move tensors to Cuda

Training step

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

valid_loss_min = np.Inf
```

```
for epoch in range(epochs):
    train_loss = valid_loss = 0.0
    :
    .
    model.train() <-- train mode
    for data, target in train_loader:
        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)

    model.eval() <-- eval mode
    for data, target in valid_loader:
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()
        output = model(data)
        loss = criterion(output, target)
        valid_loss += loss.item() * data.size(0)
```

```
# Compute the final values for losses
train_loss /= len(train_loader.dataset)
valid_loss /= len(valid_loader.dataset)

# save only if there is an improvement
if valid_loss <= valid_loss_min:
        torch.save(model.state_dict(), 'model.pt')
        valid_loss_min = valid_loss
```

Summary of the Common Strategy:

1. After each conv layer, we do relu and then maxpooling.
2. Then, we flatten the data.
3. Pass through dropouts