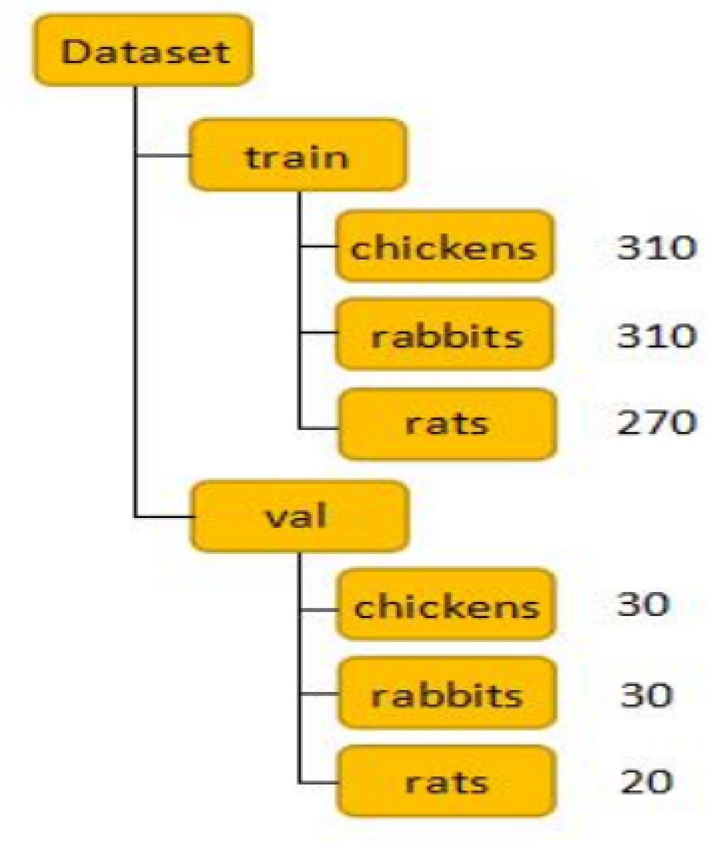
作业

任务：选兔子和鸡作为数据集，预测属于哺乳类(Mammals)还是鸟类(Birds)，数据集格式如下



- **编写数据集处理的代码**
  - 熟悉数据读取机制Dataloader与Dataset；数据预处理transforms模块机制(网上搜索)，整理提交笔记（**很重要**）
  - 二十二种transforms数据预处理方法；学会自定义transforms方法（大概浏览）
- **搭建resnet18网络 (搭建一个简单网络，resnet, googlenet, shufflenet, squeezenet)**
  - 学习池化层，全连接层和激活函数层，在池化层中有正常的最大值池化，均值池化，还有图像分割任务中常用的反池化——MaxUnpool，在激活函数中会学习Sigmoid,Tanh和Relu，以及Relu的各种变体，如LeakyReLU，PReLU， RReLU，通过搭建网络，查阅资料。(大概浏览)
  - 网络搭建常用的容器，如Sequential，ModuleList, ModuleDict，通过搭建网络模型结构加深对模型容器的认识,整理提交笔记（**很重要**）
- **编写训练的代码**
  - 十种优化器优化器，学习率调整策略，基类_LRScheduler基本属性与方法，分别Step、MultiStep、Exponential、CosineAnnealing、ReduceLROnPleateau和Lambda，一共六种学习率调整策略，14种损失函数（大概浏览文档）
- **学习建议**
  - 对例子自己注释一遍，不明白先记录下来，课堂上交流
  - 这部分代码和上节课讲的基本一样，3个模块：数据处理代码，搭建网络代码， 训练的代码基本相互独立，按步骤一步步完成
  - 查阅官方文档

```
from PIL import Image
import torch
import torch.utils.data
```

```python
from torch.autograd import Variable
from torch.utils.data import  DataLoader
import torch.optim as optim
import torch.nn as nn
import torchvision
from torchvision.transforms import transforms
import numpy as np
import matplotlib
import os
import pandas as pd
import math
import copy
import torch.nn.functional as F
```

```python
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, filepath, transform=None):
        super(MyDataset,self).__init__()
        self.img_files, self.labels = self.load_data(filepath)
        self.transform = transform

    def __getitem__(self,index):

        img_file, label = self.img_files[index], self.labels[index]
        img_file = img_file.replace('\\', '/')
        img = Image.open(img_file).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label

    def __len__(self):
        return len(self.img_files)

    def load_data(self, filepath):
        data = pd.read_csv(filepath, index_col=0)
        img_files = data.iloc[:,0].values
        labels = data.iloc[:,1].values
        return img_files, labels
```

```python
train_transforms = transforms.Compose([transforms.Resize((500, 500)),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       ])
val_transforms = transforms.Compose([transforms.Resize((500, 500)),
                                     transforms.ToTensor()
                                     ])

TRAIN_ANNO = 'class_classification/Classes_train_annotation.csv'
VAL_ANNO = 'class_classification/Classes_val_annotation.csv'
CLASSES = ['Mammals', 'Birds']
train_dataset = MyDataset(TRAIN_ANNO, transform = train_transforms)
test_dataset = MyDataset(VAL_ANNO, transform = val_transforms)

train_loader = DataLoader(dataset=train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=1, shuffle=False)
data_loaders = {'train': train_loader, 'val': test_loader}
```

```
## 搭建model: resnet18  ----------官方源码-----------
# def conv3x3(in_planes, out_planes, stride=1):
#     return nn.Conv2d(in_planes, out_planes,
#                      kernel_size = 3,stride = stride,
#                      padding = 1, bias = False)

# class BasicBlock(nn.Module):
#     expansion = 1

#     def __init__(self, inplanes, planes, stride = 1, downsample = None):
#         super(BasicBlock,self).__init__()
#         self.conv1 = conv3x3(inplanes, planes, stride)
#         self.bn1 = nn.BatchNorm2d(planes)
#         self.relu = nn.ReLU(inplace = True)
#         self.conv2 = conv3x3(planes, planes)
#         self.bn2 = nn.BatchNorm2d(planes)
#         self.downsample = downsample
#         self.stride = stride

#     def forward(self, x):
#         residual = x
#         out = self.conv1(x)
#         out = self.bn1(out)
#         out = self.relu(out)

#         out = self.conv2(out)
#         out = self.bn2(out)

#         if self.downsample is not None:
#             residual = self.downsample(x)
#         out += residual
#         out = self.relu(out)
#         return out

# class ResNet(nn.Module):
#     def __init__(self, block, layers, num_classes = 2):
#         self.inplanes = 64
#         super(ResNet, self).__init__()
#         self.conv1 = nn.Conv2d(3, 64, kernel_size = 7, stride = 2, padding = 3, bias = False)
#         self.bn1 = nn.BatchNorm2d(64)
#         self.relu = nn.ReLU(inplace = True)
#         self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 0, ceil_mode = True)
#         self.layer1 = self._make_layer(block, 64, layers[0])
#         self.layer2 = self._make_layer(block, 128, layers[1], stride = 2)
#         self.layer3 = self._make_layer(block, 256, layers[2], stride = 2)
#         self.layer4 = self._make_layer(block, 512, layers[3], stride = 2)
#         self.avgpool = nn.AvgPool2d(7)
#         self.fc = nn.Linear(512 * block.expansion * 4, num_classes)

#         for m in self.modules():
#             if isinstance(m,nn.Conv2d):
#                 n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
#                 m.weight.data.normal_(0,math.sqrt(2./n))
#             elif isinstance(m,nn.BatchNorm2d):
#                 m.weight.data.fill_(1)
#        m.bias.data.zero_()

#  def _make_layer(self, block, planes, blocks, stride = 1):
#    downsample = None
#    if stride != 1 or self.inplanes != planes * block.expansion:
#      downsample = nn.Sequential(
#       nn.Conv2d(self.inplanes, planes * block.expansion, kernel_size =
1, stride = stride, bias = False),
#       nn. BatchNorm2d(planes * block.expansion))
#    layers = []
#    layers.append(block(self.inplanes, planes, stride, downsample))
#    self.inplanes = planes * block.expansion
#    for i in range(1, blocks):
#      layers.append(block(self.inplanes, planes))

#    return nn.Sequential(*layers)

#  def forward(self, x):
#    x = self.conv1(x)
#    x = self.bn1(x)
#    x = self.relu(x)
#    x = self.maxpool(x)

#    x = self.layer1(x)
#    x = self.layer2(x)
#    x = self.layer3(x)
#    x = self.layer4(x)

#    x = self.avgpool(x)
#    x = x.view(x.size(0), -1)
#    x = self.fc(x)

#    return x

# def resnet18(pretrained = False):
#  model = ResNet(BasicBlock,[2,2,2,2])
#  if pretrained:
#    model.load_state_dict(model_zoo.load_url(model_urls['resnet18']))
#  return model

class Residual(nn.Module): # 残差单元

  def __init__(self, in_channels, out_channels, stride=1, downsample=False):
    super(Residual, self).__init__()
    self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1) # 指定stride
    self.bn1 = nn.BatchNorm2d(out_channels)
    self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)
    self.bn2 = nn.BatchNorm2d(out_channels)
    self.downsamples = downsample
    if self.downsamples:
      self.downsample = nn.Sequential(
                  nn.Conv2d(in_channels, out_channels,
kernel_size=1, stride=stride),
                  nn.BatchNorm2d(out_channels)
                  )
    else:
            self.downsample = None

    def forward(self, x):
        identity = x
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        if self.downsamples:
            identity = self.downsample(x)
        return F.relu(identity + out)


def Residual_block(in_channels, out_channels, num_Residual, first_block=False):
# 残差块

    if first_block:
        assert in_channels==out_channels

    BasicBlock = []
    for i in range(num_Residual):
        if i==0 and not first_block:
            BasicBlock.append(Residual(in_channels, out_channels,
downsample=True, stride=2))
        else:
            BasicBlock.append(Residual(out_channels, out_channels))
    return nn.Sequential(*BasicBlock)

class FlattenLayer(nn.Module):

    def __init__(self):
        super(FlattenLayer, self).__init__()

    def forward(self, x):
        return x.view(x.size(0), -1)


resnet18 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
            )
resnet18.add_module('layer1', Residual_block(64, 64, 2, first_block=True)) #2*2
resnet18.add_module('layer2', Residual_block(64, 128, 2)) #
resnet18.add_module('layer3', Residual_block(128, 256, 2))
resnet18.add_module('layer4', Residual_block(256, 512, 2))
resnet18.add_module('avgpool', nn.AdaptiveAvgPool2d((1,1)))

fc = nn.Sequential(
           FlattenLayer(),
           nn.Linear(512, 2)
        )
resnet18.add_module('fc', fc)
```

```
## 编写训练代码
def train_model(model, criterion, optimizer, scheduler,device, num_epochs=50):
    Loss_list = {'train': [], 'val': []}
    Accuracy_list_classes = {'train': [], 'val': []}
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    model = model.to(device)
    for epoch in range(num_epochs):
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
            running_loss = 0.0
            corrects_classes = 0
            for idx, data in enumerate(data_loaders[phase]):
                inputs, labels_classes = Variable(data[0].to(device)),
Variable(data[1].to(device))
                optimizer.zero_grad()#梯度清零
                with torch.set_grad_enabled(phase == 'train'):
                    x_classes = model(inputs) # 前向传播
                    _, preds_classes = torch.max(x_classes, 1) # 结果解析
                    loss = criterion(x_classes, labels_classes)
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                corrects_classes += torch.sum(preds_classes == labels_classes)

            epoch_loss = running_loss / len(data_loaders[phase].dataset)
            Loss_list[phase].append(epoch_loss)

            epoch_acc_classes = corrects_classes.double() /
len(data_loaders[phase].dataset)
            epoch_acc = epoch_acc_classes

            Accuracy_list_classes[phase].append(100 * epoch_acc_classes)
            print('epoch {}/{}, {} Loss: {:.4f}  Acc_classes:
{:.2%}'.format(epoch,num_epochs - 1, phase, epoch_loss,epoch_acc_classes))

            if phase == 'val' and epoch_acc > best_acc:

                best_acc = epoch_acc_classes
                best_model_wts = copy.deepcopy(model.state_dict())
                print('Best val classes Acc: {:.2%}'.format(best_acc))

    model.load_state_dict(best_model_wts)
    torch.save(model.state_dict(), 'best_model.pt')
    print('Best val classes Acc: {:.2%}'.format(best_acc))
    return model, Loss_list, Accuracy_list_classes
```

```
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')
network = resnet18
optimizer = torch.optim.SGD(network.parameters(), lr=0.01, momentum=0.9)
criterion = nn.CrossEntropyLoss()
exp_lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1,
gamma=0.1) # Decay LR by a factor of 0.1 every 1 epochs
model, Loss_list, Accuracy_list_classes = train_model(network, criterion,
optimizer, exp_lr_scheduler, device, num_epochs=100)
```

```
epoch 0/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 1.0265  Acc_classes: 50.97%
```

```
/home/yd/anaconda3/envs/pytracking/lib/python3.7/site-packages/PIL/Image.py:952:
UserWarning: Palette images with Transparency expressed in bytes should be
converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
```

```
val Loss: 18.1666  Acc_classes: 50.00%
Best val classes Acc: 50.00%
epoch 1/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.8216  Acc_classes: 56.45%
val Loss: 0.6850  Acc_classes: 71.67%
Best val classes Acc: 71.67%
epoch 2/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.7068  Acc_classes: 60.97%
val Loss: 0.5923  Acc_classes: 75.00%
Best val classes Acc: 75.00%
epoch 3/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.7228  Acc_classes: 60.48%
val Loss: 1.3541  Acc_classes: 50.00%
epoch 4/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.7134  Acc_classes: 61.29%
val Loss: 3.0153  Acc_classes: 66.67%
epoch 5/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.6157  Acc_classes: 69.52%
val Loss: 1.9337  Acc_classes: 65.00%
epoch 6/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.5945  Acc_classes: 69.03%
val Loss: 1.5431  Acc_classes: 58.33%
epoch 7/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.6244  Acc_classes: 70.16%
val Loss: 3.8288  Acc_classes: 58.33%
epoch 8/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.6060  Acc_classes: 69.03%
val Loss: 2.0399  Acc_classes: 61.67%
epoch 9/99
```

```
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.6225  Acc_classes: 70.65%
val Loss: 0.6861  Acc_classes: 71.67%
epoch 10/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5262  Acc_classes: 74.19%
val Loss: 0.8099  Acc_classes: 70.00%
epoch 11/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5431  Acc_classes: 73.39%
val Loss: 1.2553  Acc_classes: 70.00%
epoch 12/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5724  Acc_classes: 71.94%
val Loss: 0.9320  Acc_classes: 68.33%
epoch 13/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5756  Acc_classes: 73.06%
val Loss: 0.6773  Acc_classes: 73.33%
epoch 14/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.6330  Acc_classes: 71.61%
val Loss: 0.7556  Acc_classes: 71.67%
epoch 15/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5542  Acc_classes: 74.84%
val Loss: 0.5883  Acc_classes: 75.00%
epoch 16/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5205  Acc_classes: 76.94%
val Loss: 1.0803  Acc_classes: 68.33%
epoch 17/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4787  Acc_classes: 78.55%
val Loss: 0.7964  Acc_classes: 75.00%
epoch 18/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5026  Acc_classes: 76.77%
val Loss: 0.9064  Acc_classes: 63.33%
epoch 19/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4676  Acc_classes: 77.26%
val Loss: 0.5712  Acc_classes: 83.33%
Best val classes Acc: 83.33%
epoch 20/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5261  Acc_classes: 76.29%
val Loss: 1.0338  Acc_classes: 75.00%
epoch 21/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5634  Acc_classes: 73.55%
val Loss: 0.6351  Acc_classes: 73.33%
epoch 22/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.5099  Acc_classes: 78.06%
val Loss: 0.9252  Acc_classes: 76.67%
epoch 23/99
-*-*-*-*-*-*-*-*-*-*
```

```
train Loss: 0.4742  Acc_classes: 78.39%
val Loss: 0.5099  Acc_classes: 76.67%
epoch 24/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4666  Acc_classes: 78.71%
val Loss: 0.5323  Acc_classes: 73.33%
epoch 25/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4242  Acc_classes: 81.61%
val Loss: 0.8291  Acc_classes: 68.33%
epoch 26/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4331  Acc_classes: 79.68%
val Loss: 1.1119  Acc_classes: 75.00%
epoch 27/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4633  Acc_classes: 78.39%
val Loss: 0.5625  Acc_classes: 70.00%
epoch 28/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4390  Acc_classes: 81.94%
val Loss: 0.6421  Acc_classes: 76.67%
epoch 29/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.3850  Acc_classes: 83.06%
val Loss: 0.6786  Acc_classes: 78.33%
epoch 30/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4454  Acc_classes: 79.84%
val Loss: 0.9551  Acc_classes: 76.67%
epoch 31/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4398  Acc_classes: 82.10%
val Loss: 0.7402  Acc_classes: 75.00%
epoch 32/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4383  Acc_classes: 78.71%
val Loss: 2.1010  Acc_classes: 63.33%
epoch 33/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4108  Acc_classes: 80.32%
val Loss: 0.9089  Acc_classes: 71.67%
epoch 34/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.4025  Acc_classes: 82.74%
val Loss: 0.9434  Acc_classes: 70.00%
epoch 35/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.3988  Acc_classes: 82.42%
val Loss: 0.8302  Acc_classes: 70.00%
epoch 36/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.3555  Acc_classes: 84.52%
val Loss: 0.5448  Acc_classes: 78.33%
epoch 37/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.3699  Acc_classes: 84.03%
val Loss: 0.5071  Acc_classes: 78.33%
```

```
epoch 38/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3741  Acc_classes: 84.68%
val Loss: 0.4985  Acc_classes: 80.00%
epoch 39/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3843  Acc_classes: 84.19%
val Loss: 0.6999  Acc_classes: 75.00%
epoch 40/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3468  Acc_classes: 85.00%
val Loss: 0.8494  Acc_classes: 73.33%
epoch 41/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3412  Acc_classes: 84.52%
val Loss: 0.6326  Acc_classes: 81.67%
epoch 42/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3395  Acc_classes: 85.81%
val Loss: 0.5203  Acc_classes: 76.67%
epoch 43/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3616  Acc_classes: 84.84%
val Loss: 0.5586  Acc_classes: 78.33%
epoch 44/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3261  Acc_classes: 87.74%
val Loss: 0.5476  Acc_classes: 71.67%
epoch 45/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3495  Acc_classes: 86.77%
val Loss: 0.5165  Acc_classes: 81.67%
epoch 46/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3099  Acc_classes: 86.77%
val Loss: 0.6329  Acc_classes: 68.33%
epoch 47/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3121  Acc_classes: 87.58%
val Loss: 0.6421  Acc_classes: 76.67%
epoch 48/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.2817  Acc_classes: 89.19%
val Loss: 0.9286  Acc_classes: 68.33%
epoch 49/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3364  Acc_classes: 85.32%
val Loss: 0.7432  Acc_classes: 71.67%
epoch 50/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.2932  Acc_classes: 88.39%
val Loss: 0.6654  Acc_classes: 71.67%
epoch 51/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.3455  Acc_classes: 84.84%
val Loss: 0.5234  Acc_classes: 78.33%
epoch 52/99
_*_*_*_*_*_*_*_*_*_*
```

```
train Loss: 0.2484  Acc_classes: 90.32%
val Loss: 0.5326  Acc_classes: 75.00%
epoch 53/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2334  Acc_classes: 90.81%
val Loss: 0.7444  Acc_classes: 78.33%
epoch 54/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.3243  Acc_classes: 86.94%
val Loss: 0.8792  Acc_classes: 83.33%
epoch 55/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2881  Acc_classes: 86.94%
val Loss: 0.6170  Acc_classes: 75.00%
epoch 56/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2661  Acc_classes: 89.52%
val Loss: 0.7238  Acc_classes: 78.33%
epoch 57/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2595  Acc_classes: 89.35%
val Loss: 0.5977  Acc_classes: 73.33%
epoch 58/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2212  Acc_classes: 90.81%
val Loss: 0.6270  Acc_classes: 78.33%
epoch 59/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1613  Acc_classes: 94.68%
val Loss: 0.7853  Acc_classes: 76.67%
epoch 60/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2045  Acc_classes: 92.58%
val Loss: 1.3615  Acc_classes: 68.33%
epoch 61/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2797  Acc_classes: 88.55%
val Loss: 1.2628  Acc_classes: 66.67%
epoch 62/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2976  Acc_classes: 88.39%
val Loss: 0.6556  Acc_classes: 81.67%
epoch 63/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2399  Acc_classes: 89.35%
val Loss: 0.8430  Acc_classes: 66.67%
epoch 64/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2187  Acc_classes: 90.97%
val Loss: 0.5550  Acc_classes: 73.33%
epoch 65/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1949  Acc_classes: 91.94%
val Loss: 0.8476  Acc_classes: 75.00%
epoch 66/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2095  Acc_classes: 91.77%
val Loss: 0.6220  Acc_classes: 80.00%
```

```
epoch 67/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2134  Acc_classes: 91.94%
val Loss: 1.6876  Acc_classes: 70.00%
epoch 68/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2943  Acc_classes: 89.03%
val Loss: 0.5940  Acc_classes: 83.33%
epoch 69/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1650  Acc_classes: 93.23%
val Loss: 0.6879  Acc_classes: 76.67%
epoch 70/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1672  Acc_classes: 93.87%
val Loss: 0.5210  Acc_classes: 76.67%
epoch 71/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1776  Acc_classes: 92.74%
val Loss: 0.8504  Acc_classes: 73.33%
epoch 72/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1402  Acc_classes: 95.32%
val Loss: 1.1302  Acc_classes: 73.33%
epoch 73/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1288  Acc_classes: 93.71%
val Loss: 0.7896  Acc_classes: 76.67%
epoch 74/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1719  Acc_classes: 93.23%
val Loss: 1.4102  Acc_classes: 58.33%
epoch 75/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1676  Acc_classes: 94.19%
val Loss: 0.7539  Acc_classes: 83.33%
epoch 76/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1971  Acc_classes: 92.10%
val Loss: 0.4171  Acc_classes: 83.33%
epoch 77/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1716  Acc_classes: 93.39%
val Loss: 0.6339  Acc_classes: 76.67%
epoch 78/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2240  Acc_classes: 91.45%
val Loss: 1.0535  Acc_classes: 66.67%
epoch 79/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1472  Acc_classes: 94.03%
val Loss: 0.7463  Acc_classes: 81.67%
epoch 80/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2451  Acc_classes: 91.29%
val Loss: 0.9813  Acc_classes: 70.00%
epoch 81/99
-*-*-*-*-*-*-*-*-*-*
```

```
train Loss: 0.1391  Acc_classes: 95.81%
val Loss: 0.5934  Acc_classes: 83.33%
epoch 82/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1364  Acc_classes: 95.16%
val Loss: 0.9249  Acc_classes: 75.00%
epoch 83/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1097  Acc_classes: 96.13%
val Loss: 0.9114  Acc_classes: 78.33%
epoch 84/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0900  Acc_classes: 96.77%
val Loss: 0.7707  Acc_classes: 75.00%
epoch 85/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1099  Acc_classes: 95.48%
val Loss: 0.5384  Acc_classes: 76.67%
epoch 86/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1065  Acc_classes: 95.16%
val Loss: 1.0482  Acc_classes: 73.33%
epoch 87/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0648  Acc_classes: 97.58%
val Loss: 0.8492  Acc_classes: 78.33%
epoch 88/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0904  Acc_classes: 97.10%
val Loss: 0.9073  Acc_classes: 76.67%
epoch 89/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1319  Acc_classes: 95.32%
val Loss: 1.0234  Acc_classes: 71.67%
epoch 90/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1336  Acc_classes: 95.48%
val Loss: 0.8986  Acc_classes: 83.33%
epoch 91/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.2009  Acc_classes: 94.03%
val Loss: 0.6896  Acc_classes: 81.67%
epoch 92/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.1054  Acc_classes: 96.29%
val Loss: 0.6327  Acc_classes: 80.00%
epoch 93/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0596  Acc_classes: 97.58%
val Loss: 0.7826  Acc_classes: 66.67%
epoch 94/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0604  Acc_classes: 97.74%
val Loss: 1.0299  Acc_classes: 78.33%
epoch 95/99
-*-*-*-*-*-*-*-*-*-*
train Loss: 0.0810  Acc_classes: 97.42%
val Loss: 0.9415  Acc_classes: 68.33%
```

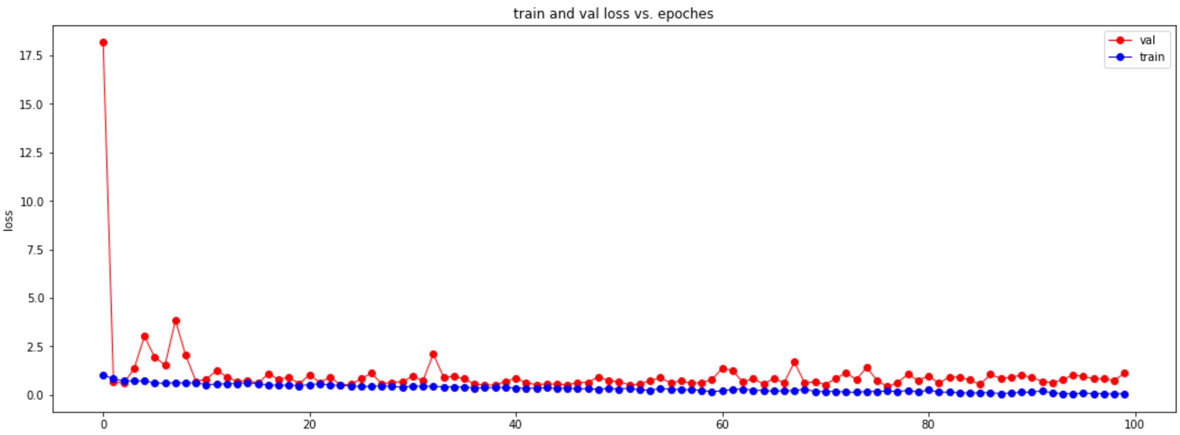
```
epoch 96/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.0678  Acc_classes: 96.77%
val Loss: 0.8388  Acc_classes: 76.67%
epoch 97/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.0585  Acc_classes: 98.06%
val Loss: 0.8308  Acc_classes: 75.00%
epoch 98/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.0416  Acc_classes: 99.03%
val Loss: 0.7508  Acc_classes: 78.33%
epoch 99/99
_*_*_*_*_*_*_*_*_*_*
train Loss: 0.0640  Acc_classes: 97.74%
val Loss: 1.1091  Acc_classes: 71.67%
Best val classes Acc: 83.33%
```

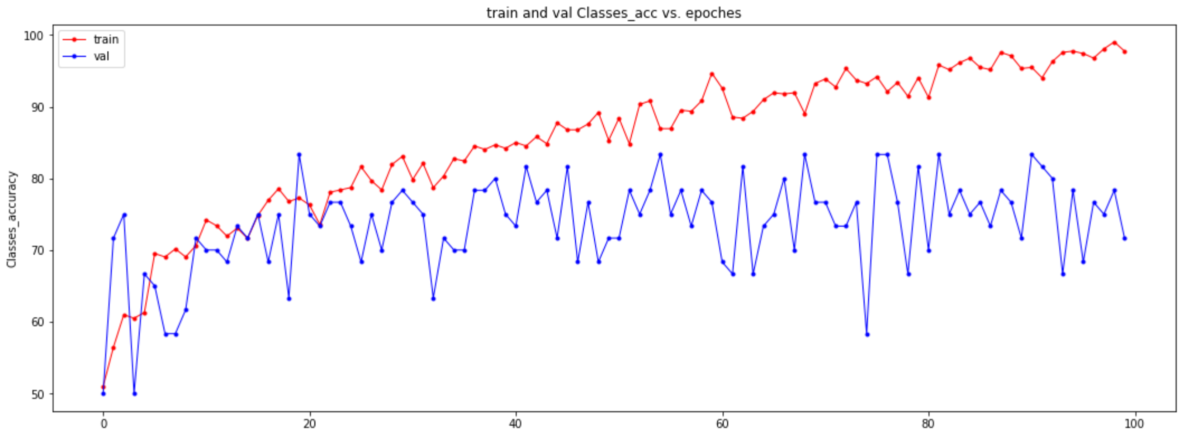
```python
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
x = range(0, 100)
y1 = Loss_list["val"]
y2 = Loss_list["train"]
plt.figure(figsize=(18,14))
plt.subplot(211)
plt.plot(x, y1, color="r", linestyle="-", marker="o", linewidth=1, label="val")
plt.plot(x, y2, color="b", linestyle="-", marker="o", linewidth=1,
label="train")
plt.legend()
plt.title('train and val loss vs. epoches')
plt.ylabel('loss')
#plt.savefig("train and val loss vs epoches.jpg")

plt.subplot(212)
y5 = Accuracy_list_classes["train"]
y6 = Accuracy_list_classes["val"]
plt.plot(x, y5, color="r", linestyle="-", marker=".", linewidth=1,
label="train")
plt.plot(x, y6, color="b", linestyle="-", marker=".", linewidth=1, label="val")
plt.legend()
plt.title('train and val Classes_acc vs. epoches')
plt.ylabel('Classes_accuracy')
```

```
Text(0, 0.5, 'Classes_accuracy')
```





```
def visualize_model(model):
    model.eval()

    with torch.no_grad():
        plt.rcParams['figure.figsize'] = (8, 6)
        plt.rcParams['savefig.dpi'] = 300  # 图片像素
        plt.rcParams['figure.dpi'] = 300  # 分辨率

        fig, ax = plt.subplots(4,4)
        axes = ax.flatten()
        for i, data in enumerate(data_loaders['val']):
            inputs = data[0]
            labels_classes = Variable(data[1].to(device))

            x_classes = model(Variable(inputs.to(device))) #
            x_classes=x_classes.view(-1,2)
            _, preds_classes = torch.max(x_classes, 1)
            img_input = transforms.ToPILImage()(inputs.squeeze(0))
            axes[i].imshow(img_input)#显示原图
            axes[i].set_title('predicted : {}\n GT: {}'.format(CLASSES[preds_classes],CLASSES[labels_classes]))

            if i == 15: #
                break
        plt.suptitle('Batch1')
        plt.tight_layout()
        plt.show()
```

```
visualize_model(model)
```