resnet18_changeplot

June 13, 2020

[2]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.models as models

import os
import torch
import librosa

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from datasets import AudioDataset
```

0.0.1 package path

[3]:

```
# path to urban sound 8k
data_root = "data/UrbanSound8K/"
# path to label
label_path = "data/UrbanSound8K/metadata/UrbanSound8K.csv"
```

0.0.2 learning rate

[4]:

```
def lr_schedule(epoch, init_lr):
    if epoch <20:
        return init_lr
    elif epoch>=20 and epoch<40:
        return init_lr/10
    elif epoch>=40 and epoch <80:
        return init_lr/100
    else:
        return init_lr/1000
```

0.0.3 model construction

[5]:

```
class ResidualBlock(nn.Module):
    def __init__(self,inchannel,outchannel,stride,shortcut=None):
        super(ResidualBlock,self).__init__()
        self.basic = nn.Sequential(
            nn.Conv2d(inchannel,outchannel,3,stride,1,bias=False),#     stride
            nn.BatchNorm2d(outchannel),#
            nn.ReLU(inplace=True),#
            nn.Conv2d(outchannel,outchannel,3,1,1,bias=False),#    feature␣
 ↪map
            nn.BatchNorm2d(outchannel),
        )
        self.shortcut = shortcut

    def forward(self,x):
        out = self.basic(x)
        residual = x if self.shortcut is None else self.shortcut(x)#
        out += residual
        return nn.ReLU(inplace=True)(out)#
#ResNet
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet,self).__init__()
        self.pre = nn.Sequential(
            nn.Conv2d(3,64,7,2,3,bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3,2,1),
        )#
        self.body = self.makelayers([3,4,6,3])#
        self.classifier = nn.Linear(512,10)#

    def makelayers(self,blocklist):#
        self.layers = []
        for index,blocknum in enumerate(blocklist):
            if index != 0:
                shortcut = nn.Sequential(
                    nn.Conv2d(64*2**(index-1),64*2**index,1,2,bias=False),
                    nn.BatchNorm2d(64*2**index)
                )#
                self.layers.
 ↪append(ResidualBlock(64*2**(index-1),64*2**index,2,shortcut))#
            for i in range(0 if index==0 else 1,blocknum):
                self.layers.append(ResidualBlock(64*2**index,64*2**index,1))
        return nn.Sequential(*self.layers)
    def forward(self,x):
        x = self.pre(x)
        x = self.body(x)
        x = nn.AvgPool2d(7)(x)#kernel_size 7      feature␣
↪map   7*7 224->112->56->28->14->7
        x = x.view(x.size(0),-1)
        x = self.classifier(x)
        return x
```

[6]:

```
net = ResNet()
print(net)
```

```
ResNet(
  (pre): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  )
  (body): Sequential(
    (0): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (2): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (3): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (4): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (9): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (10): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (11): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (12): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (13): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (14): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (15): ResidualBlock(
      (basic): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (2): ReLU(inplace=True)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (classifier): Linear(in_features=512, out_features=10, bias=True)
)
```

[7]:
```
print(models.resnet34(num_classes=10))
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
   (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
 )
 (layer3): Sequential(
  (0): BasicBlock(
   (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
   (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (downsample): Sequential(
    (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   )
  )
  (1): BasicBlock(
   (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (2): BasicBlock(
   (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
   (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

0.0.4 training fuction

[14]:

```
def train(MAX_EPOCH = 100):

#     eval_interval = 5

    # initialize dataset (feature can be "mfcc", "spec", "mel_raw")
    audio_dataset = AudioDataset(5, DataRoot=data_root, LabelPath=label_path,␣
 ↪feature="mfcc", mode="train")

    # define resnet model
    cnn_model = models.resnet18(num_classes=10)
    #cnn_model =ResNet()
    # to gpu
    cnn_model = cnn_model.cuda()
    #print(cnn_model)
    # initialize dataloader
    data_loader = torch.utils.data.DataLoader(audio_dataset, batch_size=32, shuffle=True, num_workers=1)

    # loss function
    loss_fn = nn.CrossEntropyLoss().cuda()

    # lr
    learning_rate = 1e-3

    # initialize optimizer
    optimizer = torch.optim.Adam(cnn_model.parameters(), lr=learning_rate)

    # initialize logger
    train_acc = []
    test_acc = []

    train_loss = []
    test_loss = []

    for epoch in range(MAX_EPOCH):
        # iterate through dataset
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr_schedule(epoch, learning_rate)

        # initialize epoch stat
        correct_num = 0
        total_num = 0
        loss_sum = 0

        for idx, data in enumerate(data_loader):
            #print(idx)
            train_data, labels = data

            #train_data = train_data.type(torch.float32)/255

            # data to gpu
            train_data = train_data.cuda()
            labels = labels.cuda()

            prob = cnn_model(train_data)
            loss = loss_fn(prob, labels)

            output = prob.argmax(1)
            loss_sum += loss.item()*float(labels.shape[0])
            correct_num += (output==labels).sum().double()
            total_num += float(labels.shape[0])

            optimizer.zero_grad()

            loss.backward()

            optimizer.step()

        train_acc.append(correct_num/total_num)
        train_loss.append(loss_sum/total_num)

        print("epoch: {} acc: {:.4} avg loss: {:.4f}".format(epoch, correct_num/
↪total_num, loss_sum/total_num))

#         if epoch%5 == 4:

        a,b = test(cnn_model)
        test_acc.append(a)
        test_loss.append(b)


    plt.figure()
    plt.plot(np.arange(MAX_EPOCH), train_acc)
#     plt.plot(np.arange(eval_interval-1, MAX_EPOCH, eval_interval), test_acc)
    plt.plot(np.arange(MAX_EPOCH), test_acc)
    plt.title("accuracy")
    plt.legend(["train","val"])

    plt.figure()
    plt.plot(np.arange(MAX_EPOCH), train_loss)
#     plt.plot(np.arange(eval_interval-1, MAX_EPOCH, eval_interval), test_loss)
    plt.plot(np.arange(MAX_EPOCH), test_loss)
    plt.title("loss")
    plt.legend(["train","val"])
```

0.0.5 Test function

[15]:
```
def test(model, ):
    test_dataset = AudioDataset(5, DataRoot=data_root, LabelPath=label_path,␣
↪feature="mfcc", mode="test")
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32,␣
↪shuffle=True, num_workers=1)

    model.eval()
    loss_fn = nn.CrossEntropyLoss().cuda()

    correct_num = 0
    total_num = 0
    loss_sum = 0

    for idx, data in enumerate(test_loader):
        test_data, labels = data
        test_data = test_data.cuda()
        labels = labels.cuda()

        test_data = test_data.cuda()

        prob = model(test_data)
        loss = loss_fn(prob, labels)

        output = prob.argmax(1)

        correct_num += (output==labels).sum().double()
        loss_sum += loss.item()*float(labels.shape[0])
        total_num += float(labels.shape[0])

    model.train()

    print("##Testing## epoch acc: {:.4}".format(correct_num/total_num))
    return correct_num/total_num, loss_sum/total_num
```

[16]:

```
train(70)
```

```
verify mfcc feature success
epoch: 0 acc: 0.2199 avg loss: 2.1515
verify mfcc feature success
##Testing## epoch acc: 0.2201
epoch: 1 acc: 0.2851 avg loss: 1.9610
verify mfcc feature success
##Testing## epoch acc: 0.1763
epoch: 2 acc: 0.3638 avg loss: 1.7972
verify mfcc feature success
##Testing## epoch acc: 0.3387
epoch: 3 acc: 0.4433 avg loss: 1.5925
verify mfcc feature success
##Testing## epoch acc: 0.3782
epoch: 4 acc: 0.5249 avg loss: 1.3813
verify mfcc feature success
##Testing## epoch acc: 0.3985
epoch: 5 acc: 0.6026 avg loss: 1.1548
```

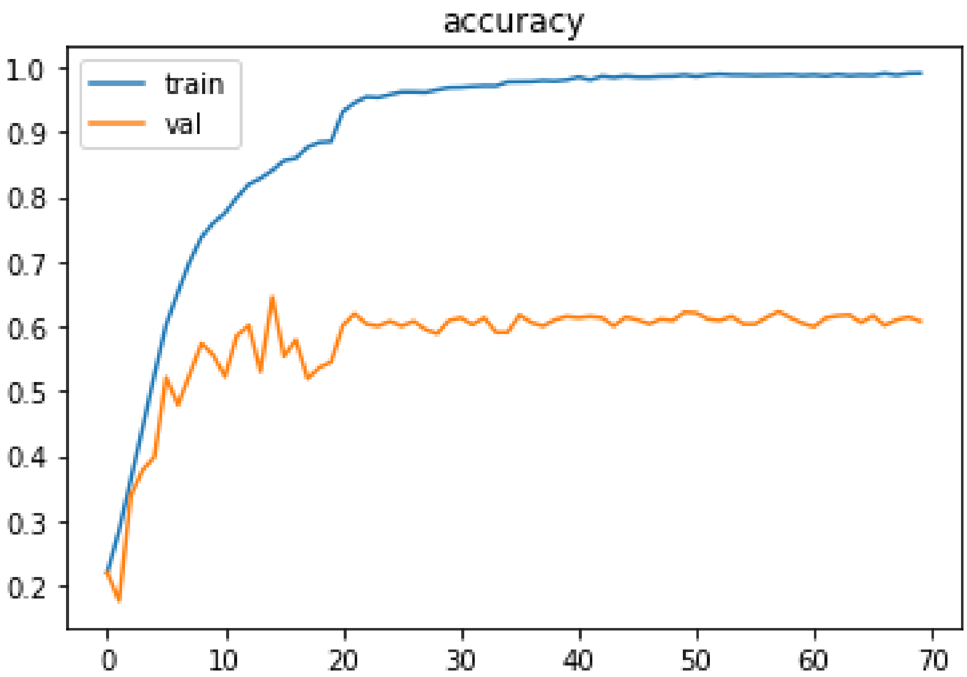
```
verify mfcc feature success
##Testing## epoch acc: 0.5214
epoch: 6 acc: 0.6528 avg loss: 1.0095
verify mfcc feature success
##Testing## epoch acc: 0.4786
epoch: 7 acc: 0.6997 avg loss: 0.8901
verify mfcc feature success
##Testing## epoch acc: 0.5256
epoch: 8 acc: 0.7376 avg loss: 0.7779
verify mfcc feature success
##Testing## epoch acc: 0.5737
epoch: 9 acc: 0.7594 avg loss: 0.6997
verify mfcc feature success
##Testing## epoch acc: 0.5556
epoch: 10 acc: 0.7748 avg loss: 0.6646
verify mfcc feature success
##Testing## epoch acc: 0.5224
epoch: 11 acc: 0.7986 avg loss: 0.5944
verify mfcc feature success
##Testing## epoch acc: 0.5855
epoch: 12 acc: 0.819 avg loss: 0.5373
verify mfcc feature success
##Testing## epoch acc: 0.6015
epoch: 13 acc: 0.8284 avg loss: 0.5023
verify mfcc feature success
##Testing## epoch acc: 0.5299
epoch: 14 acc: 0.8406 avg loss: 0.4581
verify mfcc feature success
##Testing## epoch acc: 0.6453
epoch: 15 acc: 0.8556 avg loss: 0.4220
verify mfcc feature success
##Testing## epoch acc: 0.5534
epoch: 16 acc: 0.8593 avg loss: 0.4047
verify mfcc feature success
##Testing## epoch acc: 0.5791
epoch: 17 acc: 0.8763 avg loss: 0.3589
verify mfcc feature success
##Testing## epoch acc: 0.5192
epoch: 18 acc: 0.8837 avg loss: 0.3414
verify mfcc feature success
##Testing## epoch acc: 0.5363
epoch: 19 acc: 0.8849 avg loss: 0.3244
verify mfcc feature success
##Testing## epoch acc: 0.5449
epoch: 20 acc: 0.9312 avg loss: 0.2049
verify mfcc feature success
##Testing## epoch acc: 0.6015
epoch: 21 acc: 0.945 avg loss: 0.1666
```

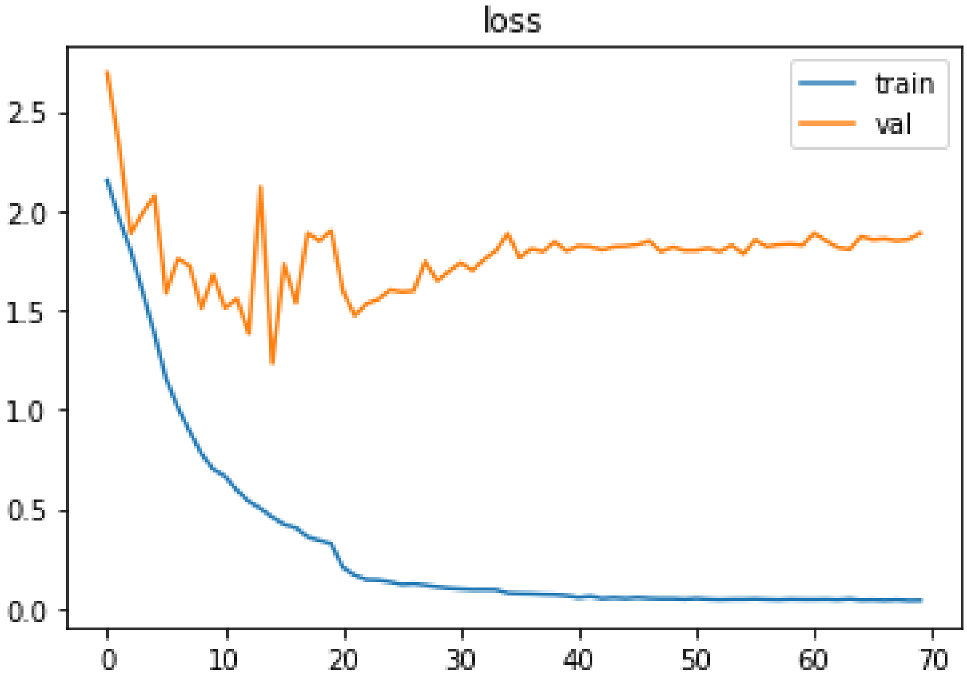
```
verify mfcc feature success
##Testing## epoch acc: 0.6197
epoch: 22 acc: 0.9541 avg loss: 0.1447
verify mfcc feature success
##Testing## epoch acc: 0.6036
epoch: 23 acc: 0.9531 avg loss: 0.1425
verify mfcc feature success
##Testing## epoch acc: 0.6004
epoch: 24 acc: 0.9574 avg loss: 0.1335
verify mfcc feature success
##Testing## epoch acc: 0.6079
epoch: 25 acc: 0.9616 avg loss: 0.1199
verify mfcc feature success
##Testing## epoch acc: 0.6004
epoch: 26 acc: 0.9616 avg loss: 0.1234
verify mfcc feature success
##Testing## epoch acc: 0.6079
epoch: 27 acc: 0.961 avg loss: 0.1166
verify mfcc feature success
##Testing## epoch acc: 0.5951
epoch: 28 acc: 0.9649 avg loss: 0.1081
verify mfcc feature success
##Testing## epoch acc: 0.5887
epoch: 29 acc: 0.9683 avg loss: 0.1015
verify mfcc feature success
##Testing## epoch acc: 0.609
epoch: 30 acc: 0.9687 avg loss: 0.0975
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 31 acc: 0.9701 avg loss: 0.0944
verify mfcc feature success
##Testing## epoch acc: 0.6026
epoch: 32 acc: 0.971 avg loss: 0.0947
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 33 acc: 0.9709 avg loss: 0.0939
verify mfcc feature success
##Testing## epoch acc: 0.5908
epoch: 34 acc: 0.9772 avg loss: 0.0764
verify mfcc feature success
##Testing## epoch acc: 0.5908
epoch: 35 acc: 0.9774 avg loss: 0.0743
verify mfcc feature success
##Testing## epoch acc: 0.6175
epoch: 36 acc: 0.9777 avg loss: 0.0733
verify mfcc feature success
##Testing## epoch acc: 0.6058
epoch: 37 acc: 0.9792 avg loss: 0.0706
```

```
verify mfcc feature success
##Testing## epoch acc: 0.6004
epoch: 38 acc: 0.9785 avg loss: 0.0691
verify mfcc feature success
##Testing## epoch acc: 0.61
epoch: 39 acc: 0.9801 avg loss: 0.0642
verify mfcc feature success
##Testing## epoch acc: 0.6154
epoch: 40 acc: 0.9846 avg loss: 0.0543
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 41 acc: 0.9801 avg loss: 0.0612
verify mfcc feature success
##Testing## epoch acc: 0.6154
epoch: 42 acc: 0.986 avg loss: 0.0500
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 43 acc: 0.9838 avg loss: 0.0539
verify mfcc feature success
##Testing## epoch acc: 0.6004
epoch: 44 acc: 0.9863 avg loss: 0.0502
verify mfcc feature success
##Testing## epoch acc: 0.6143
epoch: 45 acc: 0.9845 avg loss: 0.0546
verify mfcc feature success
##Testing## epoch acc: 0.61
epoch: 46 acc: 0.9845 avg loss: 0.0508
verify mfcc feature success
##Testing## epoch acc: 0.6036
epoch: 47 acc: 0.9858 avg loss: 0.0491
verify mfcc feature success
##Testing## epoch acc: 0.6111
epoch: 48 acc: 0.986 avg loss: 0.0492
verify mfcc feature success
##Testing## epoch acc: 0.609
epoch: 49 acc: 0.9878 avg loss: 0.0461
verify mfcc feature success
##Testing## epoch acc: 0.6218
epoch: 50 acc: 0.9856 avg loss: 0.0503
verify mfcc feature success
##Testing## epoch acc: 0.6207
epoch: 51 acc: 0.9879 avg loss: 0.0461
verify mfcc feature success
##Testing## epoch acc: 0.6111
epoch: 52 acc: 0.9891 avg loss: 0.0438
verify mfcc feature success
##Testing## epoch acc: 0.609
epoch: 53 acc: 0.9879 avg loss: 0.0457
```

```
verify mfcc feature success
##Testing## epoch acc: 0.6154
epoch: 54 acc: 0.9878 avg loss: 0.0459
verify mfcc feature success
##Testing## epoch acc: 0.6036
epoch: 55 acc: 0.9872 avg loss: 0.0476
verify mfcc feature success
##Testing## epoch acc: 0.6036
epoch: 56 acc: 0.9876 avg loss: 0.0450
verify mfcc feature success
##Testing## epoch acc: 0.6143
epoch: 57 acc: 0.9877 avg loss: 0.0434
verify mfcc feature success
##Testing## epoch acc: 0.6229
epoch: 58 acc: 0.9883 avg loss: 0.0462
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 59 acc: 0.987 avg loss: 0.0449
verify mfcc feature success
##Testing## epoch acc: 0.6047
epoch: 60 acc: 0.9879 avg loss: 0.0447
verify mfcc feature success
##Testing## epoch acc: 0.5994
epoch: 61 acc: 0.9865 avg loss: 0.0458
verify mfcc feature success
##Testing## epoch acc: 0.6132
epoch: 62 acc: 0.9886 avg loss: 0.0429
verify mfcc feature success
##Testing## epoch acc: 0.6165
epoch: 63 acc: 0.9869 avg loss: 0.0476
verify mfcc feature success
##Testing## epoch acc: 0.6175
epoch: 64 acc: 0.9878 avg loss: 0.0414
verify mfcc feature success
##Testing## epoch acc: 0.6058
epoch: 65 acc: 0.9873 avg loss: 0.0427
verify mfcc feature success
##Testing## epoch acc: 0.6165
epoch: 66 acc: 0.9903 avg loss: 0.0401
verify mfcc feature success
##Testing## epoch acc: 0.6015
epoch: 67 acc: 0.9876 avg loss: 0.0431
verify mfcc feature success
##Testing## epoch acc: 0.61
epoch: 68 acc: 0.9903 avg loss: 0.0390
verify mfcc feature success
##Testing## epoch acc: 0.6143
epoch: 69 acc: 0.9906 avg loss: 0.0397
```

```
verify mfcc feature success
##Testing## epoch acc: 0.6079
```


accuracy
train
val


loss
train
val