MC4竞赛Introduction

Momenta Challenge4 的赛题为“人脸视线检测”。人脸视线预测是人脸识别领域的一个重要课题，在辅助驾驶，视频监控等领域有重要的应用。 人脸视线的方向定义为从两眼正中点到人眼注视的目标连线的方向。这里，我们以水平方向为x轴，竖直方向为y轴，照片平面向里的方向为z轴建立坐标系。 我们将给出拍摄到的人脸注视目标的照片，以及人脸关键点等信息。选手需要根据这些信息预测人脸视线的角度，用经度和纬度表示。

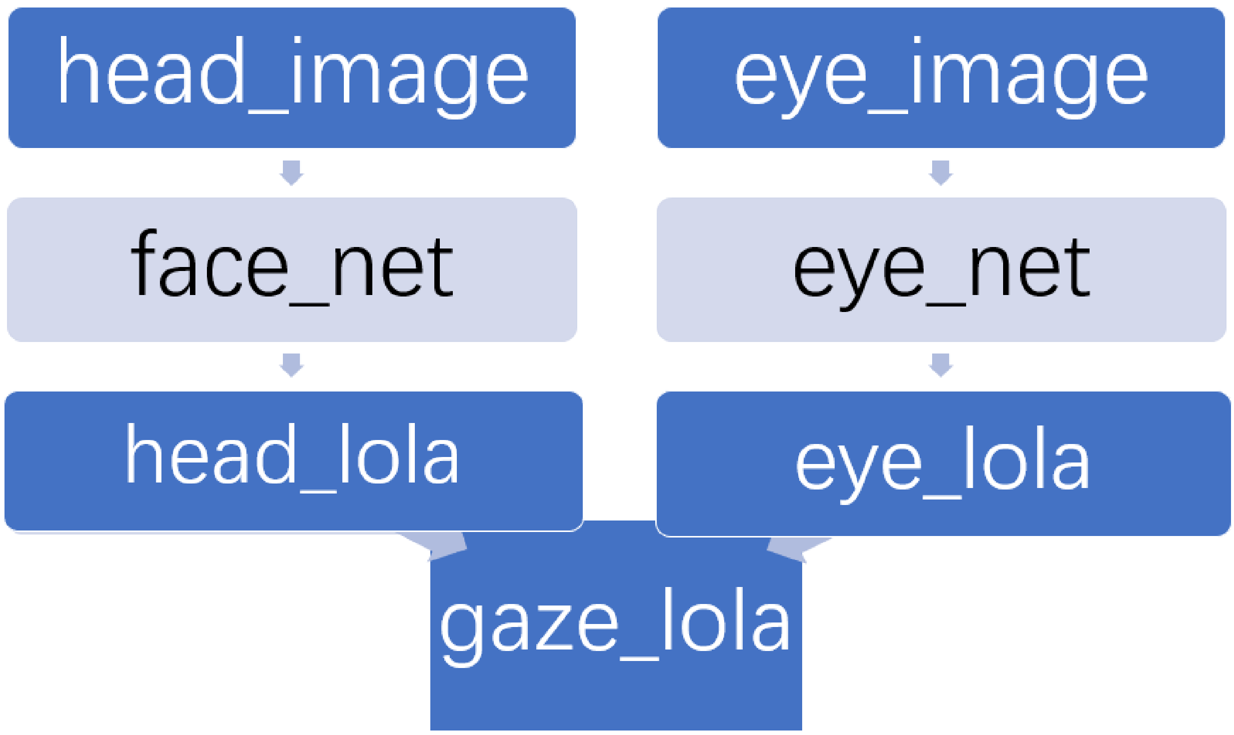
下面，我们将按步骤给出一个预测视线角度的baseline模型，以供选手参考。

首先，我们导入必要的库。本代码使用pytorch作为深度学习框架。

In [ ]:
```
import os
import numpy as np
from numpy.linalg import norm,inv
import cv2
from collections import OrderedDict
import torch
import torch.nn as nn
from torch.nn import SmoothL1Loss
from torch.nn import init
import torch.nn.functional as F
from torch.autograd import Variable
from torch.utils.data import Dataset,DataLoader
from imgaug import augmenters as iaa
import gc
import time
from torchvision import transforms
from torch.utils.data.sampler import SubsetRandomSampler
```

人脸视线的角度可以由头部姿态的角度和眼睛转向的角度组合得到。这里，我们将根据人脸的关键点裁剪出人脸和眼睛部分的图片，然后分别送入face_net和eye_net两个子网络。这两个子网络分别输出头部和眼睛的朝向角度，用经度lo和纬度la表示。然后将这两个角度组合计算得到最终视线(gaze)朝向角度lo,la

这里，我们采用ResNetX50作为face_net和eye_net的网络结构。输入的图片大小是224*224*1的灰度图，输出两个范围在(-90°,90°)内的角度。整个网络结构如下图所示：



In [ ]:

```python
from __future__ import division
"""
Creates a ResNeXt Model as defined in:
Xie, S., Girshick, R., Dollar, P., Tu, Z., & He, K. (2016).
Aggregated residual transformations for deep neural networks.
arXiv preprint arXiv:1611.05431.
import from https://github.com/facebookresearch/ResNeXt/blob/master/models/resnext.lua
"""
import math
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import init
import torch

class Bottleneck(nn.Module):
    """
    RexNeXt bottleneck type C
    """
    expansion = 4

    def __init__(self, inplanes, planes, baseWidth, cardinality, stride=1, downsample=None):
        """ Constructor
        Args:
            inplanes: input channel dimensionality
            planes: output channel dimensionality
            baseWidth: base width.
            cardinality: num of convolution groups.
            stride: conv stride. Replaces pooling layer.
        """
        super(Bottleneck, self).__init__()

        D = int(math.floor(planes * (baseWidth / 64)))
        C = cardinality

        self.conv1 = nn.Conv2d(inplanes, D*C, kernel_size=1, stride=1, padding=0, bias=False)
        self.bn1 = nn.BatchNorm2d(D*C)
        self.conv2 = nn.Conv2d(D*C, D*C, kernel_size=3, stride=stride, padding=1, groups=C, bias=False)
        self.bn2 = nn.BatchNorm2d(D*C)
        self.conv3 = nn.Conv2d(D*C, planes * 4, kernel_size=1, stride=1, padding=0, bias=False)
        self.bn3 = nn.BatchNorm2d(planes * 4)
        self.relu = nn.ReLU(inplace=True)

        self.downsample = downsample

    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)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)

        return out


class ResNeXt(nn.Module):
    """
    ResNext optimized for the ImageNet dataset, as specified in
    https://arxiv.org/pdf/1611.05431.pdf
    """
    def __init__(self, baseWidth, cardinality, layers, num_classes):
        """ Constructor
        Args:
            baseWidth: baseWidth for ResNeXt.
            cardinality: number of convolution groups.
            layers: config of layers, e.g., [3, 4, 6, 3]
            num_classes: number of classes
        """
        super(ResNeXt, self).__init__()
```

```python
        block = Bottleneck

        self.cardinality = cardinality
        self.baseWidth = baseWidth
        self.num_classes = num_classes
        self.inplanes = 64
        self.output_size = 64

        self.conv1 = nn.Conv2d(1, 64, 7, 2, 3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], 2)
        self.layer3 = self._make_layer(block, 256, layers[2], 2)
        self.layer4 = self._make_layer(block, 512, layers[3], 2)
        self.avgpool = nn.AvgPool2d(7)
        self.fc = nn.Linear(512 * block.expansion, 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):
        """ Stack n bottleneck modules where n is inferred from the depth of the network.
        Args:
            block: block type used to construct ResNext
            planes: number of output channels (need to multiply by block.expansion)
            blocks: number of blocks to be built
            stride: factor to reduce the spatial dimensionality in the first bottleneck of the block.
        Returns: a Module consisting of n sequential bottlenecks.
        """
        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, self.baseWidth, self.cardinality, stride, downsample))
        self.inplanes = planes * block.expansion
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes, self.baseWidth, self.cardinality))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool1(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 resnext50(baseWidth, cardinality):
    """
    Construct ResNeXt-50.
    """
    model = ResNeXt(baseWidth, cardinality, [3, 4, 6, 3], 2)
    return model
```

下面的GazeNet就是我们baseline使用的模型，我们从人脸和眼睛部分图片获得头部朝向和眼睛朝向的经纬度。视线角度可以由头部朝向和眼睛朝向组合计算得到（请参考文献http://openaccess.thecvf.com/content_ICCV_2017/papers/Zhu_Monocular_Free-Head_3D_ICCV_2017_paper.pdf (http://openaccess.thecvf.com/content_ICCV_2017/papers/Zhu_Monocular_Free-Head_3D_ICCV_2017_paper.pdf)）。这里我们在函数calc_gaze_lola中经过复杂的计算获得最终的视线朝向。为了方便损失函数的优化，我们对角度的输出均归一化到[0,1]范围内。

In [ ]:
```
class GazeNet(nn.Module):
    """
    The end_to_end model of Gaze Prediction Training
    """
    def __init__(self):
        super(GazeNet, self).__init__()
        self.face_net = resnext50(4,32)
        self.eye_net = resnext50(4,32)

    def calc_gaze_lola(self,head,eye):
        head_lo = head[:,0]
        head_la = head[:,1]
        eye_lo = eye[:,0]
        eye_la = eye[:,1]
        cA = torch.cos(head_lo/180*np.pi)
        sA = torch.sin(head_lo/180*np.pi)
        cB = torch.cos(head_la/180*np.pi)
        sB = torch.sin(head_la/180*np.pi)
        cC = torch.cos(eye_lo/180*np.pi)
        sC = torch.sin(eye_lo/180*np.pi)
        cD = torch.cos(eye_la/180*np.pi)
        sD = torch.sin(eye_la/180*np.pi)

        g_x = - cA * sC * cD + sA * sB * sD - sA * cB * cC * cD
        g_y =   cB * sD + sB * cC * cD
        g_z =   sA * sC * cD + cA * sB * sD - cA * cB * cC * cD
        gaze_lo = torch.atan2(-g_x,-g_z)*180.0/np.pi
        gaze_la = torch.asin(g_y)*180.0/np.pi
        gaze_lo = gaze_lo.unsqueeze(1)
        gaze_la = gaze_la.unsqueeze(1)
        gaze_lola = torch.cat((gaze_lo,gaze_la),1)
        return gaze_lola

    def forward(self,img_face,img_eye):

        return_dict = {}
        head = self.face_net(img_face)
        eye = self.eye_net(img_eye)
        #print("head",head.shape)
        #print("eye",eye.shape)
        gaze_lola = self.calc_gaze_lola(head,eye)
        #对头部，眼睛和视线朝向的角度做归一化到[0,1]范围内
        head = (head + 90)/180
        eye = (eye + 90)/180
        gaze_lola = (gaze_lola + 90)/180
        return_dict['head'] = head
        return_dict['eye']  = eye
        return_dict['gaze'] = gaze_lola
        return return_dict
```

接下来，我们准备做数据的读取和预处理工作。训练用数据集分为两部分，训练集(train)和验证集(val)。训练数据包含头部图片，眼睛图片，以及头部，眼睛和视线朝向的真实label，均使用经度la和纬度lo表示。

在将图片送入网络训练前，我们需要使用image_normalize函数将图片需要做resize到固定的输入大小(1,224,224)。在读入图片数据时，我们还可以做图像增强操作，来获得更丰富的训练样本并增强模型的鲁棒性。

In [ ]:
```
def image_normalize(im_data,transform=None):
    im_data = cv2.resize(im_data,(224,224))
    if transform:
        im_data = transform.augment_image(im_data)
    im_data = im_data[np.newaxis,:].astype(np.float64)
    return im_data
```

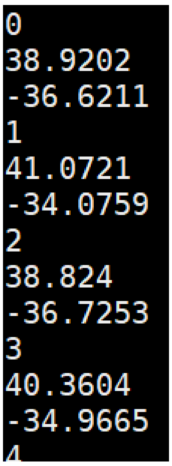
In [ ]:
```python
transform =  iaa.Sequential([
    iaa.CropAndPad(percent=(-0.1, 0.1)), # random crops
    # Small gaussian blur with random sigma between 0 and 0.5.
    # But we only blur about 50% of all images.
    iaa.Sometimes(0.5,
        iaa.GaussianBlur(sigma=(0, 0.5))
    ),
    # Strengthen or weaken the contrast in each image.
    iaa.ContrastNormalization((0.75, 1.5)),
    # Add gaussian noise.

    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255)),
], random_order=True) # apply augmenters in random order
```

下面是读取数据的过程。总的数据路径在data_dir下。训练数据共约10万组，头部及眼睛图片的子路径为{data_dir}/head/和{data_dir}/l_eye, {data_dir}/r_eye，其中l_eye和r_eye代表左眼和右眼。对于同一张人脸图片，其左眼图片(l_eye)和右眼图片(r_eye)对应的眼睛角度数值相同。在训练网络时，对每一张训练图像，选手只需要选择其对应的一只眼睛的图片参与训练即可。图片样例如下：

头部图片：

眼睛图片：

头部，眼睛和视线朝向的角度保存在txt文件中。 第一行图片名 第二行和第三行包含两个(-90,90)内的角度，分别为经度lo和纬度la。 具体样例如下图：


0
38.9202
-36.6211
1
41.0721
-34.0759
2
38.824
-36.7253
3
40.3604
-34.9665
4

我们使用pytorch的Dataset和DataLoader来打包和加载我们的训练数据。

In [ ]:

```
class GazeDataset(Dataset):
    def __init__(self,data_dir,mode,transform = None):
        """
        Args:
            data_dir (string): Directory with all the images.
            mode (string): train/val/test subdirs.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """

        self.data_dir = data_dir
        self.transform = transform
        self.mode = mode
        self.img_list = os.listdir(os.path.join(data_dir,"head"))
        if self.mode == "train":
            self.head_label = self.load_gt(os.path.join(data_dir,"head_label.txt"))
            self.gaze_label = self.load_gt(os.path.join(data_dir,"gaze_label.txt"))
            self.eye_label = self.load_gt(os.path.join(data_dir,"eye_label.txt"))


    def load_kp(self,filename):
        ret = {}
        with open(filename,'r') as kpfile:
            while True:
                line = kpfile.readline()
                if not line:
                    break
                img_filename = line.strip("\n")
                #im_data = cv2.imread(os.path.join(p,str(i),img_filename))
                src_point = []
                line = kpfile.readline()
                p_count = int(line.strip("\n"))
                for j in range(p_count):
                    x = float(kpfile.readline().strip("\n"))
                    y = float(kpfile.readline().strip("\n"))
                    src_point.append((x,y))
                ret[img_filename] = src_point
        return ret

    def load_gt(self,filename):
        ret = {}
        with open(filename,"r") as f:
            while True:
                line = f.readline()
                if not line:
                    break
                line = line.strip("\n")+".png"

                lo = float(f.readline().strip("\n"))
                la = float(f.readline().strip("\n"))
                ret[line] = np.array([lo,la],dtype=np.float32)
        return ret

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

    def __getitem__(self, idx):

        head_image = cv2.imread(os.path.join(self.data_dir,"head",self.img_list[idx]),cv2.IMREAD_GRAYSCALE)
        #头部图像还包含了大量背景区域，需要做居中裁剪
        mid_x,mid_y = head_image.shape[0]//2,head_image.shape[1]//2
        head_image = head_image[mid_x-112:mid_x+112,mid_y-112:mid_y+112]
        leye_image = cv2.imread(os.path.join(self.data_dir,"l_eye",self.img_list[idx]),cv2.IMREAD_GRAYSCALE)
        reye_image = cv2.imread(os.path.join(self.data_dir,"r_eye",self.img_list[idx]),cv2.IMREAD_GRAYSCALE)
        eye_image = leye_image if np.random.rand()<0.5 else reye_image

        head_image = image_normalize(head_image)
        eye_image = image_normalize(eye_image)
        if self.mode == "train":
            head_lola = self.head_label[self.img_list[idx]]
            eye_lola = self.eye_label[self.img_list[idx]]
            gaze_lola = self.gaze_label[self.img_list[idx]]

            sample = {'img_name':self.img_list[idx], 'head_image': head_image,'eye_image': eye_image,'head_lola': head
_lola,'eye_lola': eye_lola, 'gaze_lola': gaze_lola}
        else:
            sample = {'img_name':self.img_list[idx], 'head_image': head_image,'eye_image': eye_image}

        return sample
```

为了更好的训练模型，我们将训练样本分为训练集(train_set)和验证集(valid_set)。一个训练epoch为跑完所有train_set的数据所用的迭代次数。在完成一个训练epoch后，我们计算模型在验证集上的loss。最后我们根据验证集上的loss选择最优的模型。

In [ ]:
```
def get_train_valid_loader(dataset,
                           batch_size,
                           seed = 0,
                           valid_size=0.1,
                           shuffle=True,
                           show_sample=False,
                           num_workers=8,
                           pin_memory=False):

    num_train = len(dataset)
    indices = list(range(num_train))
    split = int(np.floor(valid_size * num_train))

    if shuffle:
        np.random.seed(seed)
        np.random.shuffle(indices)

    train_idx, valid_idx = indices[split:], indices[:split]
    train_sampler = SubsetRandomSampler(train_idx)
    valid_sampler = SubsetRandomSampler(valid_idx)

    train_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, sampler=train_sampler,
        num_workers=num_workers)
    valid_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, sampler=valid_sampler,
        num_workers=num_workers)

    return (train_loader, valid_loader)
```

经过预处理后的人脸和眼睛图片如下:

我们在多块GPU上进行模型的并行训练，并监控模型的损失函数loss。模型的损失函数采用SmoothL1Loss。我们将头部朝向的loss_head，眼睛朝向的loss_eye,视线朝向loss_gaze三个loss相加得到total_loss。在每个迭代得到训练的误差后，我们反向传播误差的梯度。模型优化方法使用随机梯度下降(SGD)。

我们设置总的迭代次数为200000。初始的学习率设为0.1，我们在10000,20000,25000个epoch后将学习率变为0.01,0.001,0.0001.

In [ ]:
```
def main(dataset):
    os.environ['CUDA_VISIBLE_DEVICES'] = "0,1,2,3,4,5,6,7"#8 gpus per node
    use_cuda = torch.cuda.is_available()

    model = GazeNet()
    if use_cuda:
        model = nn.DataParallel(model).cuda()
    GPU_COUNT = torch.cuda.device_count()
    epoch = 0#one epoch is to iterate over the entire training set
    steps = 200000
    print("Training Starts!")
    phase = "train"
    train_loader,valid_loader = get_train_valid_loader(dataset,16*GPU_COUNT)
    dataiterator = iter(train_loader)
    start_time = time.time()
    for step in range(steps):
        if phase == "train":
            #train mode
            #define optimizer
            if epoch == 0:
                learning_rate = 0.1
                optimizer = torch.optim.SGD(model.parameters(),lr=learning_rate,momentum=0.9,weight_decay=5e-4)
            elif epoch == 10000:
                learning_rate = 0.1
                optimizer = torch.optim.SGD(model.parameters(),lr=learning_rate,momentum=0.9,weight_decay=5e-4)
            elif epoch == 20000:
                learning_rate = 0.1
                optimizer = torch.optim.SGD(model.parameters(),lr=learning_rate,momentum=0.9,weight_decay=5e-4)
            elif epoch == 25000:
                learning_rate = 0.1
                optimizer = torch.optim.SGD(model.parameters(),lr=learning_rate,momentum=0.9,weight_decay=5e-4)
            optimizer.zero_grad()

            try:
                input_data = next(dataiterator)
            except StopIteration:
                phase = "val"
                continue
            for key in input_data:
                if key!="img_name":
                    if use_cuda:
                        input_data[key] = Variable(input_data[key]).type(torch.cuda.FloatTensor)
                    else:
                        input_data[key] = Variable(input_data[key]).type(torch.FloatTensor)
            img_head = input_data['head_image']
            img_eye = input_data['eye_image']

            head_gt = input_data['head_lola']
            gaze_gt = input_data['gaze_lola']
            eye_gt = input_data['eye_lola']
            #将真值也做归一化，和模型输出量纲相同
            head_gt = (head_gt + 90)/180
            gaze_gt = (gaze_gt + 90)/180
            eye_gt = (eye_gt + 90)/180
            output = model(img_head,img_eye)
            loss_fn = SmoothL1Loss()
            loss_head = loss_fn(output['head'],head_gt)
            loss_eye = loss_fn(output['eye'],eye_gt)
            loss_gaze = loss_fn(output['gaze'],gaze_gt)
            total_loss = loss_head + loss_eye + loss_gaze
            if (step+1)%100==0:
                print("Step: {} Elapsed Time: {}s".format(step+1,time.time()-start_time))
                print("head: {:.5f} eye: {:.5f} gaze: {:.5f} total: {:.5f}"\
                      .format(loss_head.item(),loss_eye.item(),loss_gaze.item(),total_loss.item()))

            if (step+1)%1000==0:
                if not os.path.exists("ckpt/"+str(learning_rate)):
                    os.makedirs("ckpt/"+str(learning_rate))
                torch.save({"model":model.module.state_dict(),"optim":optimizer.state_dict()},\
                           "./ckpt/{}/train_{}_step.pth".format(learning_rate,1+step))
                gc.collect()
            total_loss.backward()
            optimizer.step()
        else:
            #val mode
            print("###one training epoch ends. Now validation###")
            epoch += 1
            valid_gaze = []
            valid_total = []
            valid_head = []
            valid_eye = []
```

```
            dataiterator = iter(valid_loader)
            while True:
                try:
                    input_data = next(dataiterator)
                except StopIteration:
                    break
                for key in input_data:
                    if key!="img_name":
                        if use_cuda:
                            input_data[key] = Variable(input_data[key]).type(torch.cuda.FloatTensor)
                        else:
                            input_data[key] = Variable(input_data[key]).type(torch.FloatTensor)
                head_gt = (head_gt + 90)/180
                gaze_gt = (gaze_gt + 90)/180
                eye_gt = (eye_gt + 90)/180
                output = model(img_head,img_eye)
                loss_fn = SmoothL1Loss()
                loss_head = loss_fn(output['head'],head_gt).item()
                loss_eye = loss_fn(output['eye'],eye_gt).item()
                loss_gaze = loss_fn(output['gaze'],gaze_gt).item()
                valid_gaze.append(loss_gaze)
                valid_head.append(loss_head)
                valid_eye.append(loss_eye)
                valid_total.append(loss_head + loss_eye + loss_gaze)
            print("head: {:.5f} eye: {:.5f} gaze: {:.5f} total: {:.5f}"\
                  .format(np.mean(valid_head),np.mean(valid_eye),np.mean(valid_gaze),np.mean(valid_total)))


            print("#############################################")
            phase = "train"
            train_loader,valid_loader = get_train_valid_loader(gaze_set,16*GPU_COUNT)
            dataiterator = iter(train_loader)
```

下面，我们在测试集上做预测。我们读取测试数据，它们只包含头部和眼睛图片，在{test_data_dir}下。然后我们加载{model_path}路径下的模型，将图片输入到模型，得到预测结果gaze_lola。最后我们将预测结果按照输出文件格式输出到路径为{output_path}的文件中。在ssh登录窗口(如MobaXterm)我们可以把文件从客户端下载到本地，并提交到竞赛网站上。

In [ ]:
```
def get_test_loader(data_dir,
                    batch_size=1,
                    shuffle=False,
                    num_workers=1,
                    pin_memory=False):


    dataset = GazeDataset(data_dir,"test")

    data_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, shuffle=shuffle,
        num_workers=num_workers, pin_memory=pin_memory,
    )
    return data_loader
```

In [ ]:
```
def output_predict(dataloader,output_path,pretrained_model = None):
    #test mode
    model = GazeNet()
    if pretrained_model:
        pt = torch.load(pretrained_model)
        model.load_state_dict(pt["model"])
    os.environ['CUDA_VISIBLE_DEVICES'] = "0"#use one gpu for testing?
    use_cuda = torch.cuda.is_available()
    if use_cuda:
        model = model.cuda()
    dataiterator = iter(dataloader)
    pred = {}
    while True:
        try:
            input_data = next(dataiterator)
        except StopIteration:
            break
        for key in input_data:
            if key!= "img_name":
                if use_cuda:
                    input_data[key] = Variable(input_data[key]).type(torch.cuda.FloatTensor)
                else:
                    input_data[key] = Variable(input_data[key]).type(torch.FloatTensor)
        img_head = input_data['head_image']
        img_eye = input_data['eye_image']

        output = model(img_head,img_eye)
        gaze_lola = output["gaze"].data.cpu().numpy()
        gaze_lola = gaze_lola*180 - 90
        img_name_batch = input_data['img_name']
        for idx,img_name in enumerate(img_name_batch):
            pred[img_name] = gaze_lola[idx]
    with open(output_path,"w") as f:
        for k,v in pred.items():
            f.write(k.split(".")[0]+"\n")
            f.write("%0.3f\n" % v[0])
            f.write("%0.3f\n" % v[1])
```

In [ ]:
```
if __name__ == "__main__":
    gaze_set = GazeDataset('/data/mc_data/MC4/train',"train",transform)
    main(gaze_set)
    test_data_dir = "/data/mc_data/MC4/test"
    output_path = "./pred.txt"
    model_path = "./ckpt/0.1/train_1000_step.pth"
    test_loader = get_test_loader(test_data_dir)
    output_predict(test_loader,output_path,model_path)
```