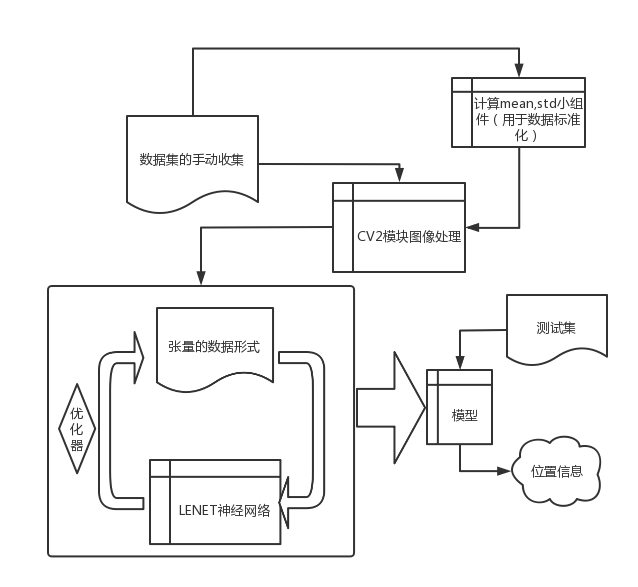
# 办公室位置信息图片识别系统

## 流程图架构图



## 数据集的收集

1. 分五类地点：
   1. 正门口前台前廊道。
   2. 男厕所门前。
   3. 资料室门前廊道。
   4. 会议室门前廊道。
   5. 办公室B区门前廊道。
2. 收集数量计划：
   1. 站立姿势竖直拍摄方向
   2. 一点365度，50张照片
   3. 5点共250张照片
   4. 一个地点一个数据集文件夹

## 数据处理及拆分

1. 创建图像处理类：

class RMB\_dataset(Dataset):  
 def \_\_init\_\_(self, path="", transform=None):  
 # 输入：图像的存储位置  
 # 负责：根据指定的路径 将训练集 or 测试集 or 验证集 组织成如下形式  
 # train\_data = [[图像名称1, 类别1],[图像名称2, 类别2],....[图像名称n, 类别n]]  
 self.img\_info = []  
 for root, dir, files in os.walk(path):  
 for file in files:  
 file\_name = os.path.join(root, file)  
 label = int(file\_name.split('\\')[-2])  
 # print(file\_name)  
 # 0: '办公室B区门前廊道', 1: '会议室门前廊道', 2: '男厕所门前', 3: '正门口前台前廊道', 4: '资料室门前廊道'  
 self.img\_info.append([file\_name, label])  
 self.transfrom = transform  
  
 # def \_\_getitem\_\_(self, item):  
  
 def \_\_getitem\_\_(self, index):  
 # 逐张读取图像  
 # 之后进行图像处理：旋转、缩放、模糊，翻转  
 # index 是个下标。范围【0,训练集的长度）  
 img\_name, label = self.img\_info[index] # 读图像  
  
 img = Image.open(img\_name) # image mode=RGB size=3468x4624  
  
 if self.transfrom is not None:  
 img = self.transfrom(img)  
  
 return img, label  
  
 def \_\_len\_\_(self):  
 # 返回值是整个数据集的长度  
 return len(self.img\_info)

该类及方法得作用是将图像像素点得RGB值，加载为网络可以输入得数据格式。[[图像1, 类别1],[图像2, 类别2],....[图像n, 类别n]]。

1. 计算图像集合中图像RGB值得均差，标准差方法：

# 计算数据集的均值和标准差  
def getStat(train\_data):  
 *'''  
 Compute mean and variance for training data* ***:param*** *train\_data: 自定义类Dataset(或ImageFolder即可)* ***:return****: (mean, std)  
 '''* print('Compute mean and variance for training data.')  
 print(len(train\_data))  
 train\_loader = torch.utils.data.DataLoader(  
 train\_data, batch\_size=1, shuffle=False, num\_workers=0,  
 pin\_memory=True)  
 mean = torch.zeros(3)  
 std = torch.zeros(3)  
 for X, \_ in train\_loader:  
 for d in range(3):  
 mean[d] += X[:, d, :, :].mean()  
 std[d] += X[:, d, :, :].std()  
 mean.div\_(len(train\_data))  
 std.div\_(len(train\_data))  
 return list(mean.numpy()), list(std.numpy())

将数据集得目录输入进方法，得到数据集得mean及std。norm\_std = [0.211, 0.223, 0.237]，norm\_mean = [0.472, 0.456, 0.425]。

1. transform模块将图像4维numpy矩阵进行缩放，灰度，归一化处理。

# 仅在训练集中增加大量的图像处理，灰度处理  
trans\_train = transforms.Compose([  
 transforms.Resize((Image\_size, Image\_size,)), # 缩放图像  
 transforms.RandomGrayscale(p=0.9), # 90%的数据灰度化  
 transforms.ToTensor(), # 将图像转换为tensor, 除255  
 transforms.Normalize(norm\_mean, norm\_std) # 归一化  
])

1. 最终得转化为网络可用得数据格式得图像数据，打乱分批（batch）神经网路模型一个批次一个批次得训练。

# 针对训练集  
train\_path = "dataset/train"  
train\_data = RMB\_dataset(path=train\_path, transform=trans\_train)  
  
train\_loader = DataLoader(  
 dataset=train\_data, # 数据类的对象  
 batch\_size=Batch\_size,  
 shuffle=True,  
 # num\_workers=1  
)

Batch\_size = 20 每二十个图像数据为一个批次。

## 模型的建立(LeNet-5)

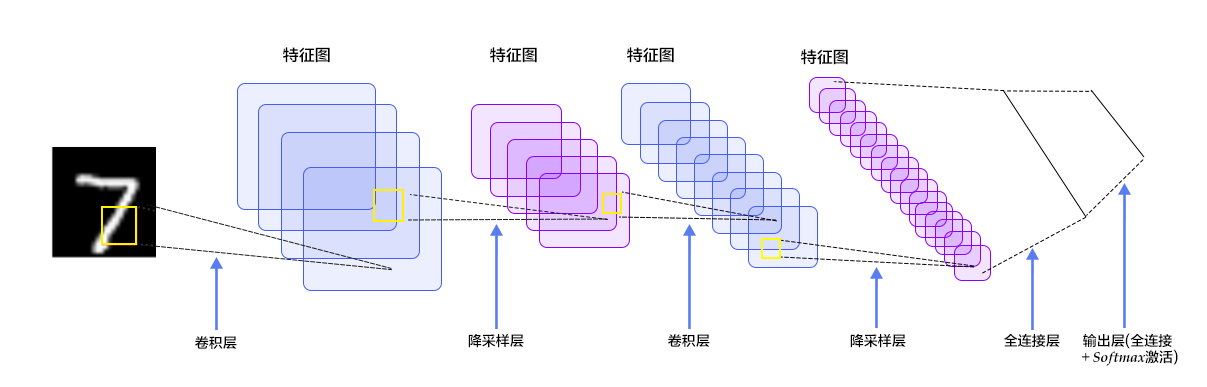
参考https://blog.csdn.net/qq\_42570457/article/details/81460807

1. 使用pytorch模块建立网络结构：

# 网络结构  
# 基本定义\_\_init\_\_, 前向传播forward  
class LeNet(nn.Module):  
 def \_\_init\_\_(self, classes):  
 # 初始化函数中，定义每层  
 super(LeNet, self).\_\_init\_\_()  
 # 输入的通道数3，输出的通道数6，卷积核的宽和高都是5  
 # 卷积核：6\*3\*5\*5  
 self.conv1 = nn.Conv2d(3, 6, 5)  
 self.conv2 = nn.Conv2d(6, 16, 5) # 卷积核 16\*6\*5\*5  
 self.fc1 = nn.Linear(16\*13\*13, 120) # 全连接  
 self.fc2 = nn.Linear(120, 84)  
 self.fc3 = nn.Linear(84, classes)  
 self.relu = nn.ReLU(inplace=True)  
  
 def forward(self, x):  
 # X: B\*C\*H\*W  
 # X: 1\*3\*64\*64  
 out = F.relu(self.conv1(x)) # 1\*6\*64\*64  
 out = F.max\_pool2d(out, 2) # 核的大小2\*2； 1\*6\*60\*60  
 out = F.relu(self.conv2(out)) # 1\* 16\*30\*30  
 out = F.max\_pool2d(out, 2) # 核的大小2\*2；1\*16\*26\*26  
 out = out.view(out.size(0), -1) # 展平(1, 16\*13\*13)==(1, 2704)  
 out = F.relu(self.fc1(out)) # full connect (1, 120)  
 out = F.relu(self.fc2(out)) # (1, 84)  
 out = self.fc3(out) # (1, 类别数)  
 return out

1. 网络识别图片过程：

LeNet-5是一个较简单的卷积神经网络。下图显示了其结构：输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用softmax分类作为输出层。



1. 各层参数详解：

1）INPUT层-输入层，首先是数据 INPUT 层，输入图像的尺寸统一缩放并归一化为64\*64。

2）conv1卷积层：

输入图片：64\*64

卷积核大小：5\*5

卷积核种类：6

输出featuremap大小：60\*60 （64-5+1）=60

神经元数量：60\*60\*6

可训练参数：（5\*5+1) \* 6（每个滤波器5\*5=25个unit参数和一个bias参数，一共6个滤波器）

连接数：（5\*5+1）\*6\*60\*60=129600

1. max\_pool2d池化层：

输入：60\*60

采样区域：2\*2

采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。

采样种类：6

输出featureMap大小：30\*30（60/2）

神经元数量：30\*30\*6

可训练参数：2\*6（和的权+偏置）

连接数：（2\*2+1）\*6\*30\*30

S2中每个特征图的大小是C1中特征图大小的1/4。

1. conv2卷积层：

输入：30\*30

卷积核大小：5\*5

卷积核种类：16

输出featureMap大小：26\*26 (30-5+1)=26

神经元数量：26\*26\*16

可训练参数：（5\*5+1) \* 6（每个滤波器5\*5=25个unit参数和一个bias参数，一共6个滤波器）

连接数：（5\*5+1）\*6\*26\*26

1. max\_pool2d池化层：

输入：26\*26

采样区域：2\*2

采样方式：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置。

采样种类：16

输出featureMap大小：13\*13（26/2）

神经元数量：13\*13\*6

可训练参数：2\*16（和的权+偏置）

连接数：（2\*2+1）\*6\*13\*13

S2中每个特征图的大小是C1中特征图大小的1/4。

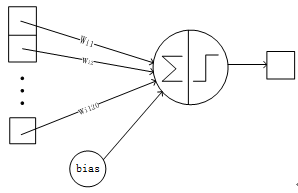
1. fc1全连接层：

输入：16\*13\*13 = (1, 2704)

计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置。

可训练参数:120\*( 2704+1)

输出：(1,120)



1. Fc2全连接层：

输入：(1,120)

计算方式：计算输入向量和权重向量之间的点积，再加上一个偏置。

可训练参数:84\*( 120+1)

输出：84

1. Fc2全连接层：

84==>5

## 模型的训练过程

1. 设置超参数：

max\_epochs = 30 表示样集训练30轮。

LR = 0.01 初始学习率0.01。

Batch\_size = 20 一批20个样本。

Image\_size = 64 图片缩放尺寸64。

1. loss\_fun = nn.CrossEntropyLoss() 使用交叉熵损失函数。
2. optimizer = Adam(model.parameters(), lr=LR) 梯度下降方法ADAM，牛顿法的一种升级方法，可以尽快的达到收敛效果。
3. 保存模型:

if len(acc\_rate\_list) > 2 and acc\_rate > acc\_rate\_list[-2]:  
 # 保存模型  
 torch.save(model.state\_dict(), 'model/location.pth')

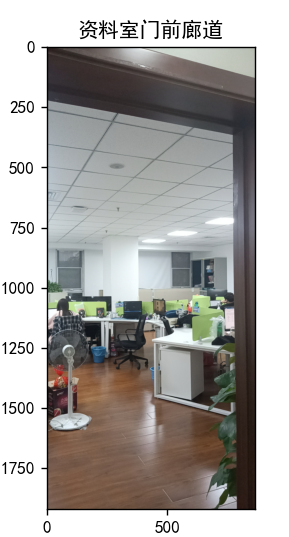
保证了模型保存准确率最高的那一个。

## 结果展示

1. 预测方法：

def predicted(img\_name):  
 img = Image.open(img\_name).convert('RGB')  
 norm\_mean = [0.472, 0.456, 0.425]  
 norm\_std = [0.211, 0.223, 0.237]  
  
 img = F.resize(img, size=(64, 64)) # 缩放  
 img = F.to\_tensor(img).to(device)  
 img = F.normalize(img, norm\_mean, norm\_std)  
 # print(img)  
 # print(img.shape)  
 # 将图片扩展为四维  
  
 img = img.expand(1, 3, 64, 64)  
 # print(img)  
 # print(img.shape)  
  
 output = model(img)  
 \_, y\_pred = torch.max(output, dim=1)  
 y\_pred = y\_pred.data.cpu().numpy()[0]  
 transition\_dict = {0: '办公室B区门前廊道', 1: '会议室门前廊道', 2: '男厕所门前', 3: '正门口前台前廊道', 4: '资料室门前廊道'}  
 pred\_location = transition\_dict[y\_pred] # 转化为位置信息  
  
 org\_img = Image.open(img\_name)  
 plt.imshow(org\_img)  
 plt.title(pred\_location)  
 plt.show()

将想要预测的图像路径传如预测方法。得到预测结果。



2，目前模型训练30轮，在测试集中现实的准确率达到70%左右。

## 其他识别方式的调研

进行中。。。

https://github.com/Boris-2021/Location\_awareness-.git