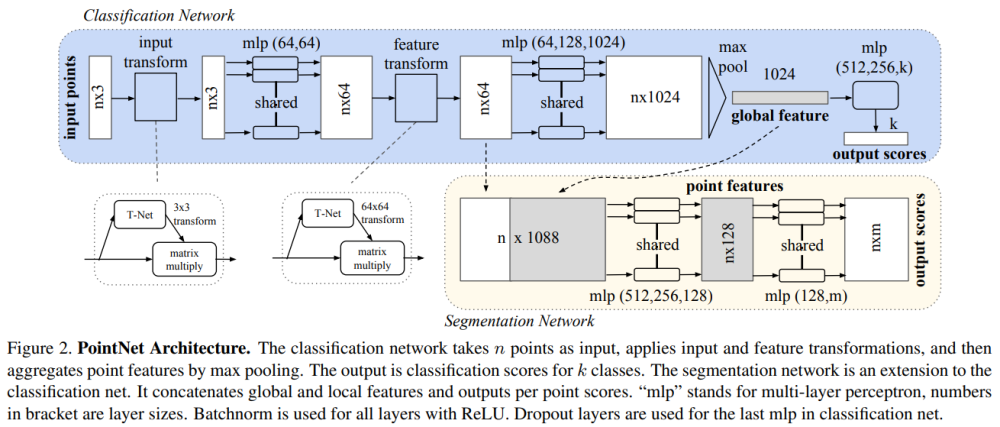
# PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

## 论文地址

<https://arxiv.org/abs/1612.00593>

## 模型架构



### 点云的两个重要特性

点云的旋转不变性

点云的置换不变性

具体来说，对于每一个N×3的点云输入，网络先通过一个**T-Net**将其在空间上对齐（旋转到正面），再通过MLP将其映射到64维的空间上，再进行对齐，最后映射到1024维的空间上。这时对于每一个点，都有一个1024维的向量表征，而这样的向量表征对于一个3维的点云明显是冗余的，因此这个时候引入最大池化操作，将1024维所有通道上都只保留最大的那一个，这样得到的1×1024的向量就是N个点云的全局特征

如果做的是分类的问题，直接将这个全局特征再进过MLP去输出每一类的概率即可；但如果是分割问题，由于需要输出的是逐点的类别，因此其将全局特征拼接在了点云64维的逐点特征上，最后通过MLP，输出逐点的分类概率

**代码解析**

**PonitNet的代码主要由两部分组成，就是T-Net和Encoder-Decoder结构**

T-Net代码:

class T\_Net(nn.Module):  
 def \_\_init\_\_(self):  
 super(T\_Net, self).\_\_init\_\_()  
 # 这里需要注意的是上文提到的MLP均由卷积结构完成  
 # 比如说将3维映射到64维，其利用64个1x3的卷积核  
 self.conv1 = torch.nn.Conv1d(3, 64, 1)  
 self.conv2 = torch.nn.Conv1d(64, 128, 1)  
 self.conv3 = torch.nn.Conv1d(128, 1024, 1)  
 self.fc1 = nn.Linear(1024, 512)  
 self.fc2 = nn.Linear(512, 256)  
 self.fc3 = nn.Linear(256, 9)  
 self.relu = nn.ReLU()  
  
 self.bn1 = nn.BatchNorm1d(64)  
 self.bn2 = nn.BatchNorm1d(128)  
 self.bn3 = nn.BatchNorm1d(1024)  
 self.bn4 = nn.BatchNorm1d(512)  
 self.bn5 = nn.BatchNorm1d(256)  
  
  
 def forward(self, x):  
 batchsize = x.size()[0]  
 x = F.relu(self.bn1(self.conv1(x)))  
 x = F.relu(self.bn2(self.conv2(x)))  
 x = F.relu(self.bn3(self.conv3(x)))  
 x = torch.max(x, 2, keepdim=True)[0]  
 x = x.view(-1, 1024)  
  
 x = F.relu(self.bn4(self.fc1(x)))  
 x = F.relu(self.bn5(self.fc2(x)))  
 x = self.fc3(x)  
  
 iden = Variable(torch.from\_numpy(np.array([1,0,0,0,1,0,0,0,1]).astype(np.float32))).view(1,9).repeat(batchsize,1)  
 if x.is\_cuda:  
 iden = iden.cuda()  
 x = x + iden  
 x = x.view(-1, 3, 3) # 输出为Batch\*3\*3的张量  
 return x

PointNet代码架构:

class PointNetEncoder(nn.Module):  
 def \_\_init\_\_(self, global\_feat = True):  
 super(PointNetEncoder, self).\_\_init\_\_()  
 self.stn = T\_Net()  
 self.conv1 = torch.nn.Conv1d(3, 64, 1)  
 self.conv2 = torch.nn.Conv1d(64, 128, 1)  
 self.conv3 = torch.nn.Conv1d(128, 1024, 1)  
 self.bn1 = nn.BatchNorm1d(64)  
 self.bn2 = nn.BatchNorm1d(128)  
 self.bn3 = nn.BatchNorm1d(1024)  
 self.global\_feat = global\_feat  
 def forward(self, x):  
 '''生成全局特征'''  
 n\_pts = x.size()[2]  
 trans = self.stn(x)  
 x = x.transpose(2,1)  
 x = torch.bmm(x, trans) # batch matrix multiply 即乘以T-Net的结果  
 x = x.transpose(2,1)  
 x = self.conv1(x)  
  
 x = F.relu(self.bn1(x))  
 pointfeat = x  
 x\_skip = self.conv2(x)  
  
 x = F.relu(self.bn2(x\_skip))  
 x = self.bn3(self.conv3(x))  
 x = torch.max(x, 2, keepdim=True)[0]  
 x = x.view(-1, 1024)  
 if self.global\_feat:  
 return x, trans  
 else:  
 x = x.view(-1, 1024, 1).repeat(1, 1, n\_pts)  
 return torch.cat([x, pointfeat], 1), trans  
  
class PointNetCls(nn.Module):  
 def \_\_init\_\_(self, k = 2):  
 super(PointNetCls, self).\_\_init\_\_()  
 self.k = k  
 self.feat = PointNetEncoder(global\_feat=False)  
 self.conv1 = torch.nn.Conv1d(1088, 512, 1)  
 self.conv2 = torch.nn.Conv1d(512, 256, 1)  
 self.conv3 = torch.nn.Conv1d(256, 128, 1)  
 self.conv4 = torch.nn.Conv1d(128, self.k, 1)  
 self.bn1 = nn.BatchNorm1d(512)  
 self.bn2 = nn.BatchNorm1d(256)  
 self.bn3 = nn.BatchNorm1d(128)  
  
 def forward(self, x):  
 '''分类网络'''  
 batchsize = x.size()[0]  
 n\_pts = x.size()[2]  
 x, trans = self.feat(x)  
 x = F.relu(self.bn1(self.conv1(x)))  
 x = F.relu(self.bn2(self.conv2(x)))  
 x = F.relu(self.bn3(self.conv3(x)))  
 x = self.conv4(x)  
 x = x.transpose(2,1).contiguous()  
 x = F.log\_softmax(x.view(-1,self.k), dim=-1)  
 x = x.view(batchsize, n\_pts, self.k)  
 return x  
  
  
class PointNetPartSeg(nn.Module):  
 def \_\_init\_\_(self,num\_class):  
 super(PointNetPartSeg, self).\_\_init\_\_()  
 self.k = num\_class  
 self.feat = PointNetEncoder(global\_feat=False)  
 self.conv1 = torch.nn.Conv1d(1088, 512, 1)  
 self.conv2 = torch.nn.Conv1d(512, 256, 1)  
 self.conv3 = torch.nn.Conv1d(256, 128, 1)  
 self.conv4 = torch.nn.Conv1d(128, self.k, 1)  
 self.bn1 = nn.BatchNorm1d(512)  
 self.bn1\_1 = nn.BatchNorm1d(1024)  
 self.bn2 = nn.BatchNorm1d(256)  
 self.bn3 = nn.BatchNorm1d(128)  
  
 def forward(self, x):  
 '''分割网络'''  
 batchsize = x.size()[0]  
 n\_pts = x.size()[2]  
 x, trans = self.feat(x)  
 x = F.relu(self.bn1(self.conv1(x)))  
 x = F.relu(self.bn2(self.conv2(x)))  
 x = F.relu(self.bn3(self.conv3(x)))  
 x = self.conv4(x)  
 x = x.transpose(2,1).contiguous()  
 x = F.log\_softmax(x.view(-1,self.k), dim=-1)  
 x = x.view(batchsize, n\_pts, self.k)  
 return x, trans

<https://blog.csdn.net/weixin_39373480/article/details/88878629>