

```

10
11 class STN3d(nn.Module):
12     def __init__(self):
13         super(STN3d, self).__init__()
14         self.conv1 = torch.nn.Conv1d(3, 64, 1)
15         self.conv2 = torch.nn.Conv1d(64, 128, 1)
16         self.conv3 = torch.nn.Conv1d(128, 1024, 1)
17         self.fc1 = nn.Linear(1024, 512)
18         self.fc2 = nn.Linear(512, 256)
19         self.fc3 = nn.Linear(256, 9)
20         self.relu = nn.ReLU()
21
22         self.bn1 = nn.BatchNorm1d(64)
23         self.bn2 = nn.BatchNorm1d(128)
24         self.bn3 = nn.BatchNorm1d(1024)
25         self.bn4 = nn.BatchNorm1d(512)
26         self.bn5 = nn.BatchNorm1d(256)
27
28
29     def forward(self, x):
30         batchsize = x.size()[0]
31         x = F.relu(self.bn1(self.conv1(x)))
32         x = F.relu(self.bn2(self.conv2(x)))
33         x = F.relu(self.bn3(self.conv3(x)))
34         x = torch.max(x, 2, keepdim=True)[0]
35         x = x.view(-1, 1024)
36
37         x = F.relu(self.bn4(self.fc1(x)))
38         x = F.relu(self.bn5(self.fc2(x)))
39         x = self.fc3(x)
40
41         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)
42         if x.is_cuda:
43             iden = iden.cuda()
44         x = x + iden
45         x = x.view(-1, 3, 3)
46         return x
47

```

shared MLP

$(B \times 3 \times n)$

$(B \times 64 \times n)$

$(B \times 128 \times n)$

$(B \times 1024 \times n)$

$(B \times 512)$

$(B \times 256)$

$(B \times 9)$

identity matrix
(residual connection)

$(B \times 3 \times 3)$

STN3d:

 3×3 transformation matrixinput: $B \times 3 \times n$

```

48
49 class STNkd(nn.Module):
50     def __init__(self, k=64):
51         super(STNkd, self).__init__()
52         self.conv1 = torch.nn.Conv1d(k, 64, 1)
53         self.conv2 = torch.nn.Conv1d(64, 128, 1)
54         self.conv3 = torch.nn.Conv1d(128, 1024, 1)
55         self.fc1 = nn.Linear(1024, 512)
56         self.fc2 = nn.Linear(512, 256)
57         self.fc3 = nn.Linear(256, k*k)
58         self.relu = nn.ReLU()
59
60         self.bn1 = nn.BatchNorm1d(64)
61         self.bn2 = nn.BatchNorm1d(128)
62         self.bn3 = nn.BatchNorm1d(1024)
63         self.bn4 = nn.BatchNorm1d(512)
64         self.bn5 = nn.BatchNorm1d(256)
65
66         self.k = k
67
68     def forward(self, x):
69         batchsize = x.size()[0]
70         x = F.relu(self.bn1(self.conv1(x)))
71         x = F.relu(self.bn2(self.conv2(x)))
72         x = F.relu(self.bn3(self.conv3(x)))
73         x = torch.max(x, 2, keepdim=True)[0]
74         x = x.view(-1, 1024)
75
76         x = F.relu(self.bn4(self.fc1(x)))
77         x = F.relu(self.bn5(self.fc2(x)))
78         x = self.fc3(x)
79
80         iden = Variable(torch.from_numpy(np.eye(self.k).flatten().astype(np.float32))).view(1,self.k*self.k).repeat(batchsize,1)
81         if x.is_cuda:
82             iden = iden.cuda()
83         x = x + iden
84         x = x.view(-1, self.k, self.k)
85         return x
86

```

shared MLP

$(B \times 64 \times n)$

$(B \times 128 \times n)$

$(B \times 1024 \times n)$

$(B \times 512)$

$(B \times 256)$

$(B \times (64 \times 64))$

identity matrix
(residual connection)

$(B \times 64 \times 64)$

STNkd:

 64×64 transformation matrixinput: $B \times 64 \times n$ high-dimensional
transformation

```

87 class PointNetFeat(nn.Module):
88     def __init__(self, global_feat = True, feature_transform = False):
89         super(PointNetFeat, self).__init__()
90         self.stn = STN3d()
91         self.conv1 = torch.nn.Conv1d(3, 64, 1)
92         self.conv2 = torch.nn.Conv1d(64, 128, 1)
93         self.conv3 = torch.nn.Conv1d(128, 1024, 1)
94         self.bn1 = nn.BatchNorm1d(64)
95         self.bn2 = nn.BatchNorm1d(128)
96         self.bn3 = nn.BatchNorm1d(1024)
97         self.global_feat = global_feat
98         self.feature_transform = feature_transform
99         if self.feature_transform:
100             self.fstn = STNkd(k=64)
101
102     def forward(self, x):
103         n_pts = x.size()[2]
104         trans = self.stn(x) # (B x 3 x 3)
105         x = x.transpose(2, 1) # (B x n x 3)
106         x = torch.bmm(x, trans) # (B x n x 3)
107         x = x.transpose(2, 1) # (B x 3 x n)
108         x = F.relu(self.bn1(self.conv1(x))) # (B x 64 x n)
109
110         if self.feature_transform:
111             trans_feat = self.fstn(x) # (B x 64 x 64)
112             x = x.transpose(2, 1) # (B x n x 64)
113             x = torch.bmm(x, trans_feat)
114             x = x.transpose(2, 1) # (B x 64 x n)
115         else:
116             trans_feat = None
117
118         pointfeat = x # B x 64 x n
119         x = F.relu(self.bn2(self.conv2(x))) # (B x 128 x n)
120         x = self.bn3(self.conv3(x)) # (B x 1024 x n)
121         x = torch.max(x, 2, keepdim=True)[0] # (B x 1024 x 1)
122         x = x.view(-1, 1024)
123         if self.global_feat:
124             return x, trans, trans_feat # (B x 1024)
125         else:
126             x = x.view(-1, 1024, 1).repeat(1, 1, n_pts) # (B x 1024 x n)
127             return torch.cat([x, pointfeat, 1], 1), trans, trans_feat # (B x 1088 x n)
128

```

shared MLP

Feature Extraction

Output:

if global-feature:

 $B \times 1024$

else

 $B \times 1088 \times n$

```

129 class PointNetCls(nn.Module):
130     def __init__(self, k=2, feature_transform=False):
131         super(PointNetCls, self).__init__()
132         self.feature_transform = feature_transform
133         self.feat = PointNetFeat(global_feat=True, feature_transform=feature_transform)
134         self.fc1 = nn.Linear(1024, 512)
135         self.fc2 = nn.Linear(512, 256)
136         self.fc3 = nn.Linear(256, k)
137         self.dropout = nn.Dropout(p=0.3)
138         self.bn1 = nn.BatchNorm1d(512)
139         self.bn2 = nn.BatchNorm1d(256)
140         self.relu = nn.ReLU()
141
142     def forward(self, x):
143         x, trans, trans_feat = self.feat(x) # (B x 1024)
144         x = F.relu(self.bn1(self.fc1(x))) # (B x 512)
145         x = F.relu(self.bn2(self.dropout(self.fc2(x)))) # (B x 256)
146         x = self.fc3(x) # (B x k)
147         return F.log_softmax(x, dim=1), trans, trans_feat

```

Classification Network

Output:

 $B \times k$

```

150 class PointNetDenseCls(nn.Module):
151     def __init__(self, k = 2, feature_transform=False):
152         super(PointNetDenseCls, self).__init__()
153         self.k = k
154         self.feature_transform = feature_transform
155         self.feat = PointNetFeat(global_feat=False, feature_transform=feature_transform)
156         self.conv1 = torch.nn.Conv1d(1088, 512, 1)
157         self.conv2 = torch.nn.Conv1d(512, 256, 1)
158         self.conv3 = torch.nn.Conv1d(256, 128, 1)
159         self.conv4 = torch.nn.Conv1d(128, self.k, 1)
160         self.bn1 = nn.BatchNorm1d(512)
161         self.bn2 = nn.BatchNorm1d(256)
162         self.bn3 = nn.BatchNorm1d(128)
163
164     def forward(self, x):
165         batchsize = x.size()[0]
166         n_pts = x.size()[2]
167         x, trans, trans_feat = self.feat(x) # (B x 1088 x n)
168         x = F.relu(self.bn1(self.conv1(x))) # (B x 512 x n)
169         x = F.relu(self.bn2(self.conv2(x)))
170         x = F.relu(self.bn3(self.conv3(x))) # (B x 128 x n)
171         x = self.conv4(x) # (B x k x n)
172         x = x.transpose(2, 1).contiguous()
173         x = F.log_softmax(x.view(-1, self.k), dim=-1) # (B x n x k)
174         x = x.view(batchsize, n_pts, self.k)
175         return x, trans, trans_feat

```

$B \times 1536 \times n$

(Global feature + point-wise feature)
Segmentation Network

Output:

 $B \times n \times k$