STN

Spatial Transformer Networks

Introduction

Problem of previous methods

- Need CNN that can disentangle object pose and deformation
- Max pooling alleviates such problem but not enough: max pooling usually use 2x2 kernel sizes which requires a deep network with multiple max pooling layers to have good enough spatial invariance
- Fixed and local max pooling = fixed and local spatial invariance = small effect for spatial invariance
- Even so, the intermediate features maps are not spatially invariant

Solution: Spatial Transformer

- Need CNN that can disentangle object pose and deformation

- Max pooling alleviates such problem but not enough: max pooling usually use 2x2 kernel sizes which requires a deep network with multiple max pooling layers to have good enough spatial invariance
- Can be placed at the beginning of the CNN and make model spatially invariant from the front layers. Also can be applied to multiple layers to overcome any deformations in intermediate feature maps

- Fixed and local max pooling = fixed and local spatial invariance = small effect for spatial invariance
- Dynamic spatial transformer that produce appropriate transformation to image

- Even so, the intermediate features maps are not spatially invariant

Spatial Transformer in Three Steps

Step1: Localization Network

Create Theta

1. Single Spatial Transformer

Conv2d (3,32,5,1,2)

Conv2d (32,32,5,1,2)

Max Pool (2x2)

Linear (32)

Linear (6)

Both have ReLU after every layer

And the last layer is initialized to produce identity matrix (with reference to grid)

Number of theta's can vary on the types of transformation the ST will do, but if there were 6 outputs…

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Set last layer weight to 0 and bias to 1 accordingly

2. Multi Spatial Transformer

Linear (32)

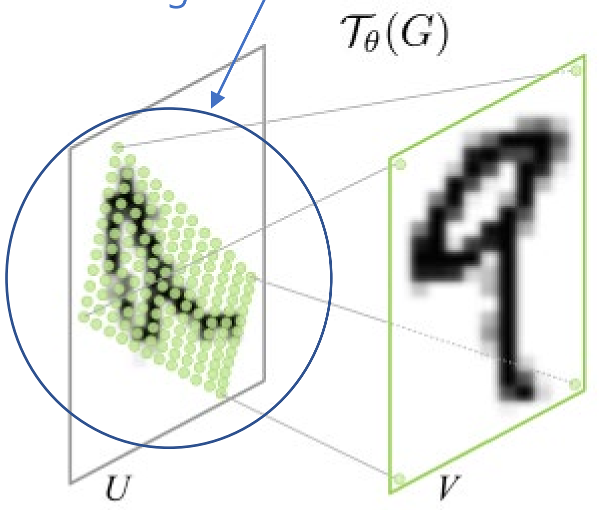
Linear (6)

Step2: Parameterized Sampling Grid

Create Grid (not image)

$$\mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

grid

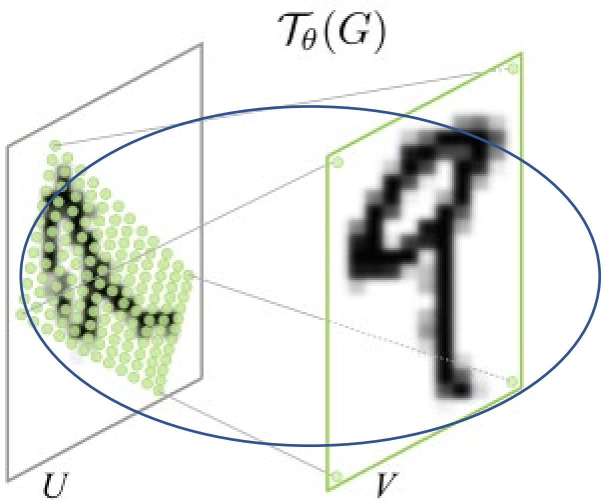


(b)

Step3: Differentiable Image Sampling

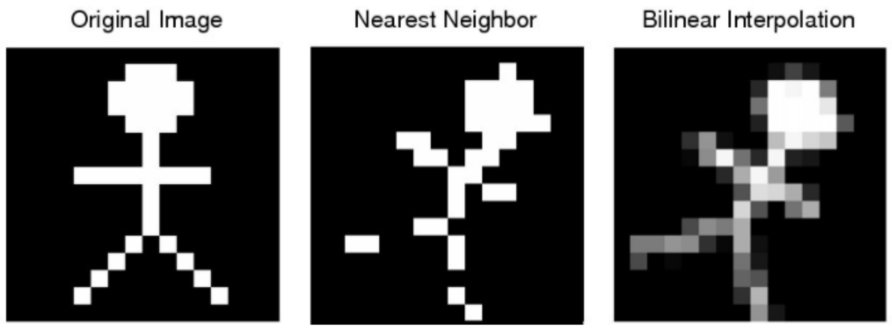
Sample image with grid

Applying grid to image



(b)

https://jamiekang.github.io/2017/05/27/spatial-transformer-networks/



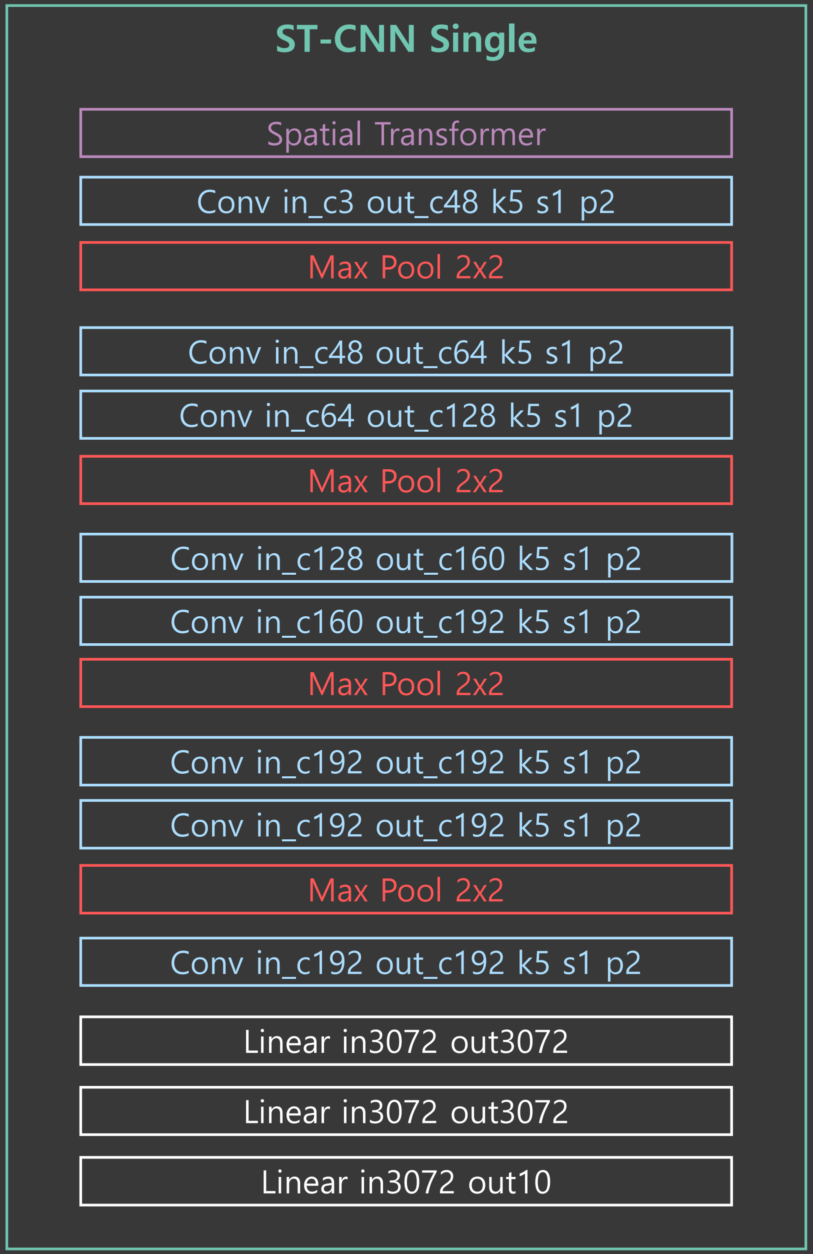
But for in between areas where pixel values are not explicitly defined:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \delta(\lfloor x_i^s + 0.5 \rfloor - m)\delta(\lfloor y_i^s + 0.5 \rfloor - n) \quad (4)$$

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \quad (5)$$

STN for SVHN

Implementation: SVHN STN


ST-CNN Single
Spatial Transformer
Conv in_c3 out_c48 k5 s1 p2
Max Pool 2x2
Conv in_c48 out_c64 k5 s1 p2
Conv in_c64 out_c128 k5 s1 p2
Max Pool 2x2
Conv in_c128 out_c160 k5 s1 p2
Conv in_c160 out_c192 k5 s1 p2
Max Pool 2x2
Conv in_c192 out_c192 k5 s1 p2
Conv in_c192 out_c192 k5 s1 p2
Max Pool 2x2
Conv in_c192 out_c192 k5 s1 p2
Linear in3072 out3072
Linear in3072 out3072
Linear in3072 out10

ST-CNN Single

- Spatial Transformer
- Conv in_c3 out_c48 k5 s1 p2
- Max Pool 2x2
- Conv in_c48 out_c64 k5 s1 p2
- Conv in_c64 out_c128 k5 s1 p2
- Max Pool 2x2
- Conv in_c128 out_c160 k5 s1 p2
- Conv in_c160 out_c192 k5 s1 p2
- Max Pool 2x2
- Conv in_c192 out_c192 k5 s1 p2
- Conv in_c192 out_c192 k5 s1 p2
- Max Pool 2x2
- Conv in_c192 out_c192 k5 s1 p2
- Linear in3072 out3072
- Linear in3072 out3072
- Linear in3072 out10

dropout applied to every layer except first and last layer + spatial transformer

learning rate within Spatial Transformer is 1/10

activation ReLU after every layer except the last FC layer of Spatial Transformer as defined in previous slides (needs to be regression layer)

```
class ST_Single_CNN(nn.Module):
    def __init__(self):
        super(ST_Single_CNN, self).__init__()

        self.conv1 = nn.Conv2d(3, 48, kernel_size=5, stride=1, padding=2)
        self.conv2 = nn.Conv2d(48, 64, kernel_size=5, stride=1, padding=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=2)
        self.conv4 = nn.Conv2d(128, 160, kernel_size=5, stride=1, padding=2)
        self.conv5 = nn.Conv2d(160, 192, kernel_size=5, stride=1, padding=2)
        self.conv6 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv7 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)
        self.conv8 = nn.Conv2d(192, 192, kernel_size=5, stride=1, padding=2)

        self.loc_conv1 = nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2)
        self.loc_conv2 = nn.Conv2d(32, 32, kernel_size=5, stride=1, padding=2)

        self.loc_fc1 = nn.Linear(8192,32)
        self.loc_fc2 = nn.Linear(32, 6)

        # initialize to predict identity transformer
        bias = torch.from_numpy(np.array([1,0,0,0,1,0]))
        nn.init.constant(self.loc_fc2.weight, 0)
        self.loc_fc2.bias.data.copy_(bias)

        self.fc1 = nn.Linear(3072, 3072)
        self.fc2 = nn.Linear(3072, 3072)

        self.fc_digit_length = nn.Linear(3072,7)

        self.fc_digit1 = nn.Linear(3072, 11)
```

```python
    def forward(self, x):
        batch_size = x.size(0)

        theta = self.loc_conv1(x)
        theta = F.relu(theta)
        # didnt omitt  max pooling here
        theta = self.maxpool(theta)
        theta = self.loc_conv2(theta)
        theta = F.relu(theta)

        theta = self.loc_fc1(theta.view(batch_size, -1))
        theta = F.relu(theta)
        theta = self.loc_fc2(theta).view(batch_size, 2, 3)

        grid = F.affine_grid(theta=theta, size=x.shape)
        x = F.grid_sample(x, grid)

        x = self.conv1(x)
        x = F.relu(x)
        # omitted max pooling here

        x = self.dropout(x)
        x = self.conv2(x)
        x = F.relu(x)

        x = self.dropout(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.maxpool(x)

        x = self.dropout(x)
        x = self.conv4(x)
        x = F.relu(x)
```

just a sneek-peek

full code in github link below