Spatial Transformer Networks

Max Jaderberg Karen Simonyan Andrew Zisserman Koray Kavukcuoglu

Google DeepMind, London, UK

15210240008 贺珂珂

这篇论文发表在 nips2015 上，作者来自于 deepmind 团队。

这个报告分为 3 个部分。1.论文的解读。2.方法实践。3.代码的理解。

一、论文解读

在这里，先对论文进行解读，包括

1.当前存在的问题

2.作者基于问题，和现有的方法的思考

3.作者提出的方法思想

4.方法的实现

5.实验效果这 5 个方面进行具体的展开。

1. 首先，作者分析了当前存在的问题

深度学习中的卷积神经网络模型（Convolutional Neural Networks)在图像检测，分割，识别任务中取得了突出的结果。但它依旧存在问题，缺乏对输入数据的空间不变性的判断。

比如，在物体分类中，同样是一件衣服，但由于衣服有旋转变化，衣服的尺度变化，网络对 2 者的判别会存在差异。为了解决这个问题，之前的网络训练中，我们通常会对训练数据进行增强，即同样是衣服的图片，我们人为地对图片进行旋转，缩放，平移等等的操作，但标签依旧为衣服，不发生变化。正是通过这样的方法，让网络学习到，即使衣服大小变化了，方向变化了，还依旧是衣服。

2. 作者的动机

基于现状,作者想到能不能有一种更为优雅的方法呢?即使一件衣服，大小有差异，有不同的旋转，网络能在内部识别出这种变化。并且自动地还原到一个标准的空间上去。这样能够大大减轻后续进行分类任务的工作量，从而更好地提升网络的性能。正是基于这种考虑，作者提出了一个新的网络层，spatial transformer layer。即这个网络层能够对输入图像的空间信息进行感知，同时对网络的输入数据进行相应的空间变换。比如，一件衣服是歪着放的，这层网络能够识别出它放歪了，把它摆到正的位置上。同时，这个空间变换层，本身是独立的，能够嵌入到现有的各种深度网络模型中去。加了 spatial transformer layer 的 cnn 网络，能够自动地，在没有额外标签的情况下，学习到平移不变性，旋转不变性，尺度不变性和更多通用的变化，使网络有更好的性能。

3. Spatial Transformer 思想的引入

cnn 网络通过局部的 max-pooling 实现空间不变性。但一般 max-pooling 的大小为 2*2，是很小的，所以，一般在网络的高层（这是 feature map 也比较小了），能够达到一定的空间不不变性，但在网络的中间层的 feature map ,几乎是没有对输入数据的空间不变形的，这是 cnn 网络的局限性。因为作者提出了 spatial transformer 模块。这个模块可以插入到现有的神经网络结构中去。在网络的训练阶段（没有其他的监督信息），网络学习到了合适的变换策略，测试的时候，新来一张图片，spatial transformer 模块就能够对这个图片单独进行相应的变换。这种变换不像 cnn 中的 pooling ,有固定的相应区域和大小。这种变换是直接作用于整个图片（或者下一个阶段的 feature map)，对整体进行相应的尺度变换，裁剪，旋转等等的变换操作。这种变换，不仅能够使网络选择相应的感兴趣的区域，同时能够对输入的图片进行一个类似矫正的操作。spatial transformer 的模块的参数能够通过标准的反向传播算法计算获得。

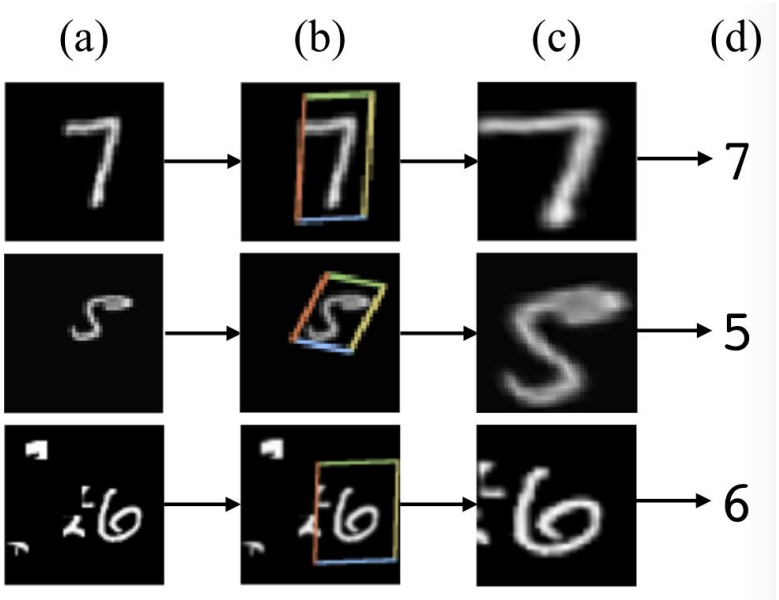


图 1.

spatial transformers 可以加入到现有的 cnn 中，有非常多的作用。

3.1 图像分类

图 1 显示了，在训练扭曲的 mnist 数字分类中，spatial transformer 起到的效果。

a 列为输入图片，这些图片有不同的旋转，缩放，和其他杂乱的信息。

b 列为 spatial transformer 中对输入数据进行相应的定位的结果。

c 为为应用 spatial transformer 对数据进行变换，摆正后的结果。

包含 spatial transformer 模块的 cnn 只用分类标签端到端训练的，并不需要给数据变换的 ground truth.

d 后续的 cnn 网络对摆正的图片进行类别的预测。

3.2 定位

给定一系列同类，但是类别未知的图像，spatial transformer 可以对每张图片中的物体进行定位。

4. Spatial Transformers 的方法详解

前面部分主要将了 spatial transformer 模块的思想，下面主要对 spatial transformer 的算法进行详细的展开。

这里再次强调，每一个特定的 input，都会有自己相对应的变换方式。如果输入是多通道的，每个通道上都做同样的变换。（这也是好理解的，如果输入是 3 通道的彩色图像，每个通道上做的变换都是一样的）

spatial transformer 模块，主要可以分为 3 大部分,见图 2。

按照计算顺序上，第一个为 localization network（即大概定位区域的网络），

它接受输入的 feature map，经过几个 hidden layer，输出这个 feature map 的变换参数。

第二个为 grid generator,这个部分的作用就是根据第一个产生的变换参数，确定在原输入的 feature map 和变换后 feature map 的映射关系。

第三部结合输入的 feature map 和映射关系，获得变换后的结果。

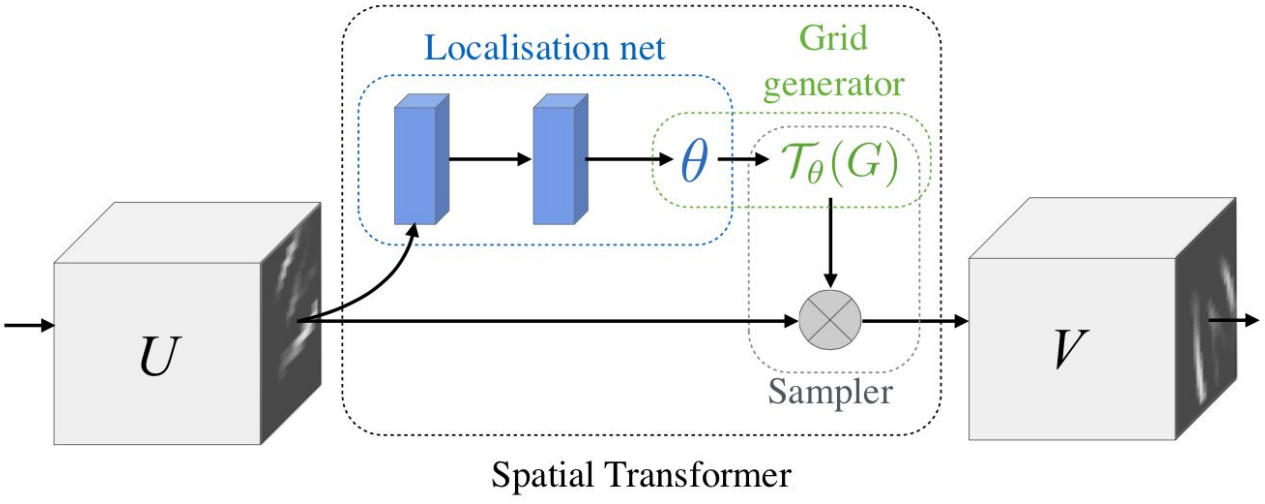


图 2.

下面将对这 3 部分进行详细的展开。

4.1 定位网络（Localization Network）

接受的输入：feature map U（H 为高度，W 为宽度，C 为通道）

经过变换 $U \in \mathbb{R}^{H \times W \times C}$ ：

输出这个 feature map 的需要进行的变换的参数。

一般来说，这个网络会有几个隐层，这些隐层可以是全连接层，也可以是卷积层，

但后一层，需要做回归，因为要输出变换参数的值。

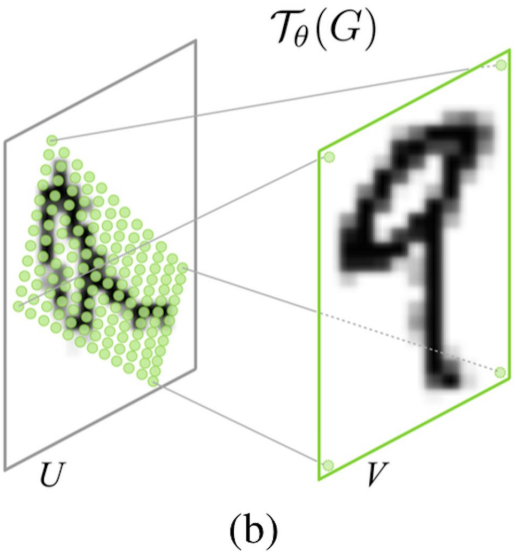
4.2 生成采样网格（Parameterized Sampling Grid ）

为了实现输入的 feature map 的变换，每个输出的值都是在以输入 input feature map

为中心的位置上，进行采样得到的。

第二个为 grid generator,这个部分的作用就是根据第一个产生的变换参数，确定在原输入的 feature map 和变换后 feature map 的映射关系。

比如图上的例子，输出的 V，通过变换参数映射（参数对应着一种特定的仿射变换），在 U 中找到对应的值。这样 V 就相当于摆正了。



(b)

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

这个式子是仿射变换的例子。θ_{13} 和 θ_{23} 表示平移。

θ_{11} 和 θ_{22} 表示 $\theta = f_{\mathrm{loc}}(U)$. 缩放。$\theta_{11}$ ， θ_{12}， θ_{21}， θ_{22} 表示旋转。

x^s ，y^s 为 source ，即对应的输入的坐标

x^t ，y^t 为输出目标的坐标。这个公式相当于，从找前面该是什么值。

4.3 保证可微的图像采样（Differentiable Image Sampling ）

图像采样要保证可微。因为参数都需要反向传播算法，求导优化。

结合输入的 feature map 和映射关系，获得变换后的结果。

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \ \forall i \in [1 \dots H'W'] \ \forall c \in [1 \dots C]$$

kernel k 代表了一种插值的方法，（比如线性插值）

这样就可以通过第二步得到的坐标，在原图上得到具体的像素值。

4.4 空间转换网络（ Spatial Transformer Networks ）

将上面的定位网络，网格生成，采样，3 个部件组合起来，就形成了 spatial transformer 模块。这个模块可以无缝地添加到现有的 cnn 网络的任何位置，称新的网络为 spatial transformer network。加入的这个模块对现有的 cnn 网络的计算影响是很小的。

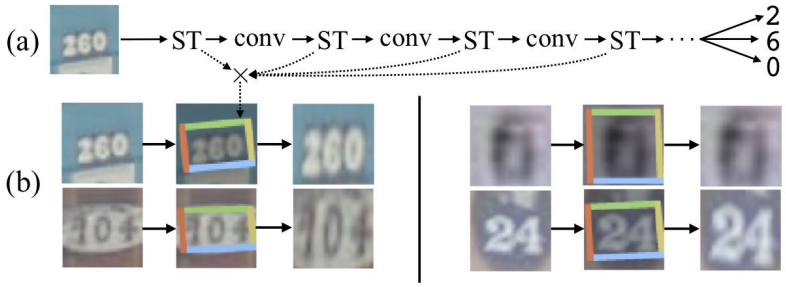
将 spatial transformers 模块集成到 cnn 网络中，允许网络自动地学习如何进行 feature map 的转变，从而有助于降低网络训练中整体的代价。定位网络中输出的值，指明了如何对每个训练数据进行转化。

5 实验结果

1.在街景字符数据集上的识别结果

通过对数据进行变换，取得了 state-of-art 的结果

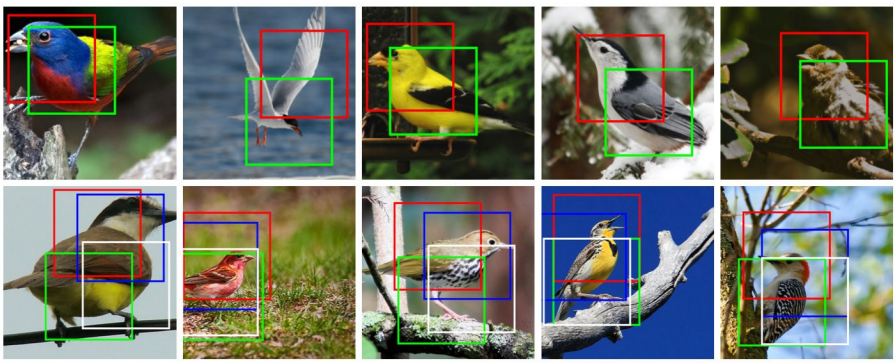
Model		Size 64px	128px
Maxout CNN [13]		4.0	-
CNN (ours)		4.0	5.6
DRAM* [1]		3.9	4.5
ST-CNN	Single	3.7	**3.9**
	Multi	**3.6**	**3.9**



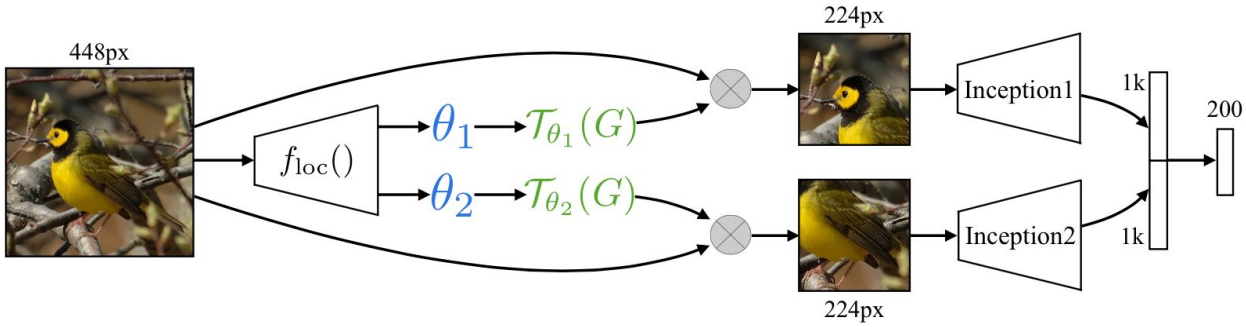
2.在鸟的细分类结果

通过 2 路的 spatial transformer networks
发现网络定位到了鸟的头部，和鸟的身体，获得了更好的检测结果。

Model	
Cimpoi '15 [5]	66.7
Zhang '14 [39]	74.9
Branson '14 [3]	75.7
Lin '15 [23]	80.9
Simon '15 [29]	81.0
CNN (ours) 224px	82.3
2×ST-CNN 224px	83.1
2×ST-CNN 448px	83.9
4×ST-CNN 448px	**84.1**



这是 2 路 spatial network 的定位结果。



二、方法实践

在网络上找到了一个基于 theano 的 lasagne 工具，这个工具已经先实现了 spatial transformer 模块，于是，我进行了实践。

1. 环境准备

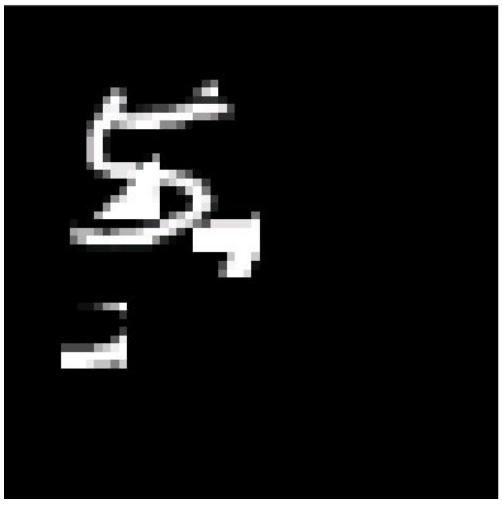
以 mnist 字符识别为基础。

首先，下载安装好 theano ，然后安装好 lasagne。

2. 数据准备

下载好相应的数据。数据都有部分的偏移和扰动信息。

下图为数据的样例。



3.模型的定义

```
def build_model(input_width, input_height, output_dim,
                batch_size=BATCH_SIZE):
    ini = lasagne.init.HeUniform()
    l_in = lasagne.layers.InputLayer(shape=(None, 1, input_width, input_height),)

    # Localization network
    b = np.zeros((2, 3), dtype='float32')
    b[0, 0] = 1
    b[1, 1] = 1
    b = b.flatten()
    loc_l1 = pool(l_in, pool_size=(2, 2))
    loc_l2 = conv(
        loc_l1, num_filters=20, filter_size=(5, 5), W=ini)
    loc_l3 = pool(loc_l2, pool_size=(2, 2))
    loc_l4 = conv(loc_l3, num_filters=20, filter_size=(5, 5), W=ini)
    loc_l5 = lasagne.layers.DenseLayer(
        loc_l4, num_units=50, W=lasagne.init.HeUniform('relu'))
    loc_out = lasagne.layers.DenseLayer(
        loc_l5, num_units=6, b=b, W=lasagne.init.Constant(0.0),
        nonlinearity=lasagne.nonlinearities.identity)

    # Transformer network
    l_trans1 = lasagne.layers.TransformerLayer(l_in, loc_out,
downsample_factor=3.0)
    print "Transformer network output shape: ", l_trans1.output_shape

    # Classification network
    class_l1 = conv(
        l_trans1,
        num_filters=32,
        filter_size=(3, 3),
        nonlinearity=lasagne.nonlinearities.rectify,
        W=ini,
    )
    class_l2 = pool(class_l1, pool_size=(2, 2))
    class_l3 = conv(
        class_l2,
        num_filters=32,
        filter_size=(3, 3),
        nonlinearity=lasagne.nonlinearities.rectify,
        W=ini,
    )
    class_l4 = pool(class_l3, pool_size=(2, 2))
    class_l5 = lasagne.layers.DenseLayer(
```

```
        class_l4,
        num_units=256,
        nonlinearity=lasagne.nonlinearities.rectify,
        W=ini,
    )

    l_out = lasagne.layers.DenseLayer(
        class_l5,
        num_units=output_dim,
        nonlinearity=lasagne.nonlinearities.softmax,
        W=ini,
    )

    return l_out, l_trans1

model, l_transform = build_model(DIM, DIM, NUM_CLASSES)
model_params = lasagne.layers.get_all_params(model, trainable=True)
```

上述代码是模型的定义过程。
首先是定位网络。接受图像输入，经过 1 层卷积，全连接等操作后输出 6 维数据。
这 6 维数据对应着仿射变换的参数，指明了图像的变换过程。
之后，

```
l_trans1 = lasagne.layers.TransformerLayer(l_in, loc_out, downsample_factor=3.0)
```

相当于利用得到的仿射变换参数，对原始的输入进行变换。
之后还是同样的分类网络，接着卷积网络，采样网络。最后 softmax 进行分类。

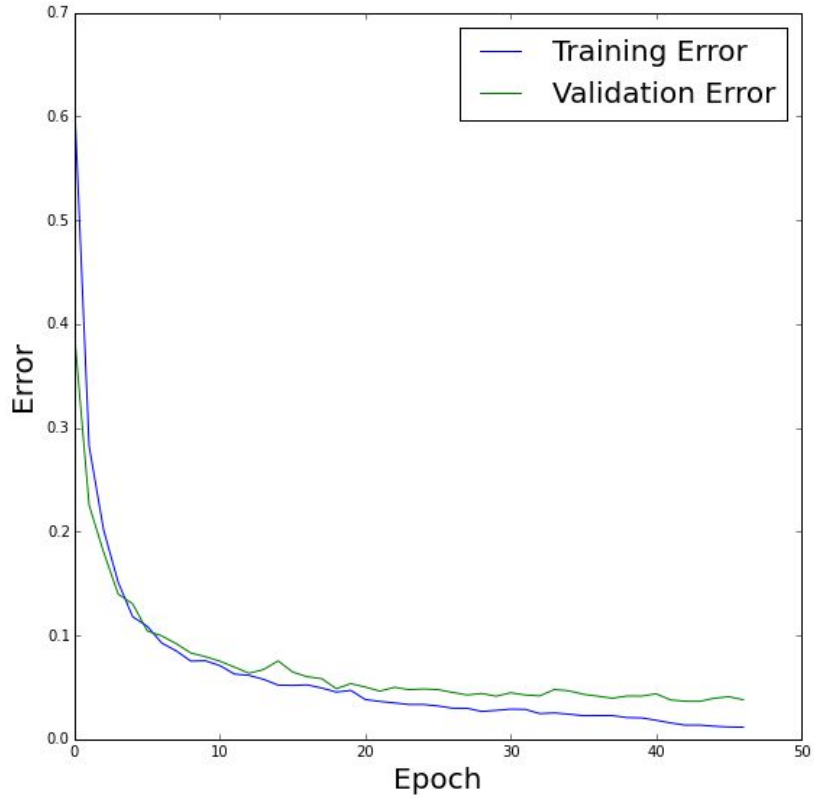
4.网络训练

进行训练，可以看到测试误差在持续下降，分类的准确率在持续提升中。

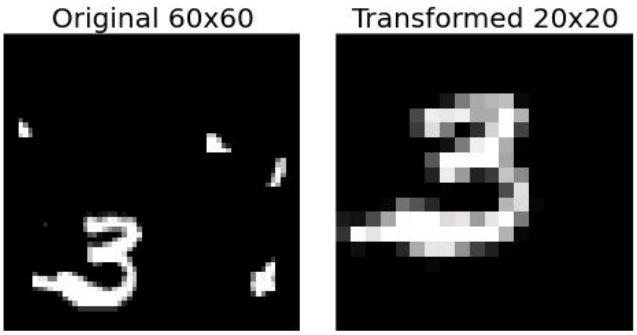
```
Epoch 0: Train cost 1.72300577164, Train acc 0.38824, val acc 0.6114, test acc 0.6087
Epoch 1: Train cost 0.867130100727, Train acc 0.71758, val acc 0.7745, test acc 0.7759
Epoch 2: Train cost 0.618825733662, Train acc 0.79848, val acc 0.8199, test acc 0.827
Epoch 3: Train cost 0.475057393312, Train acc 0.8489, val acc 0.8602, test acc 0.8613
Epoch 4: Train cost 0.369837403297, Train acc 0.88208, val acc 0.8697, test acc 0.8723
Epoch 5: Train cost 0.336995840073, Train acc 0.89126, val acc 0.8957, test acc 0.8974
Epoch 6: Train cost 0.288021206856, Train acc 0.90742, val acc 0.9005, test acc 0.8993
Epoch 7: Train cost 0.260697960854, Train acc 0.915, val acc 0.9081, test acc 0.9091
Epoch 8: Train cost 0.235620766878, Train acc 0.92484, val acc 0.917, test acc 0.9214
Epoch 9: Train cost 0.232491567731, Train acc 0.9245, val acc 0.9205, test acc 0.921
Epoch 10: Train cost 0.214803680778, Train acc 0.92916, val acc 0.9249, test acc 0.926
Epoch 11: Train cost 0.191879570484, Train acc 0.93728, val acc 0.9306, test acc 0.9317
Epoch 12: Train cost 0.187945634127, Train acc 0.93854, val acc 0.9365, test acc 0.937
Epoch 13: Train cost 0.177504748106, Train acc 0.94238, val acc 0.9329, test acc 0.933
Epoch 14: Train cost 0.161393344402, Train acc 0.9479, val acc 0.9246, test acc 0.9269
```

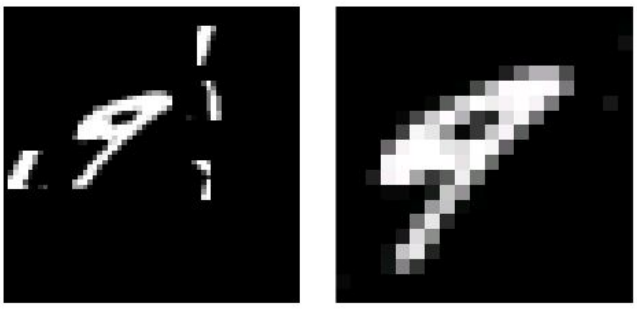
5. 结果显示

首先是误差下降情况显示。蓝色为训练误差曲线，绿色为验证误差曲线。



经过 spatial transformer 后的结果。可以 spatial transformer layer 的确学到了变换。





3.代码的理解

找出了 lasagne 工具的 theano 实现代码，进行代码的学习。

主体为 Transformer function

```
def _transform(theta, input, downsample_factor):
    num_batch, num_channels, height, width = input.shape
    theta = T.reshape(theta, (-1, 2, 3))

    # grid of (x_t, y_t, 1), eq (1) in ref [1]
    out_height = T.cast(height / downsample_factor[0], 'int64')
    out_width = T.cast(width / downsample_factor[1], 'int64')
    grid = _meshgrid(out_height, out_width)

    # Transform A x (x_t, y_t, 1)^T -> (x_s, y_s)
    T_g = T.dot(theta, grid)
    x_s = T_g[:, 0]
    y_s = T_g[:, 1]
    x_s_flat = x_s.flatten()
    y_s_flat = y_s.flatten()

    # dimshuffle input to  (bs, height, width, channels)
    input_dim = input.dimshuffle(0, 2, 3, 1)
    input_transformed = _interpolate(
        input_dim, x_s_flat, y_s_flat,
        out_height, out_width)

    output = T.reshape(
        input_transformed, (num_batch, out_height, out_width, num_channels))
    output = output.dimshuffle(0, 3, 1, 2)  # dimshuffle to conv format
    return output
```

这个函数主要做了 3 个工作，

1._meshgrid(out_height, out_width)

主要是根据 output size ,建立对应的 output 的网格。

2.T_g = T.dot(theta, grid)

这个主要是计算输出的 map 和 输入的 map 的对应关系。

3._interpolate(
        input_dim, x_s_flat, y_s_flat,
        out_height, out_width)

这个是进行插值。

1.Meshgrid 的实现。
根据输出的 height 和 width
返回网格。网格的形状为 (3 , height*width) 对应着所有点的坐标(x,y,1)
1 是为了将来计算平移方便。

```
def _meshgrid(height, width):
    x_t = T.dot(T.ones((height, 1)),
                _linspace(-1.0, 1.0, width).dimshuffle('x', 0))
    y_t = T.dot(_linspace(-1.0, 1.0, height).dimshuffle(0, 'x'),
                T.ones((1, width)))

    x_t_flat = x_t.reshape((1, -1))
    y_t_flat = y_t.reshape((1, -1))
    ones = T.ones_like(x_t_flat)
    grid = T.concatenate([x_t_flat, y_t_flat, ones], axis=0)
    return grid
```

2.T_g = T.dot(theta, grid)

在前面的学习中，得到 theta 为 2*3 的矩阵
Grid 为 3 *（height*width)的矩阵，这样相乘后，可以得到对应的 input 的坐标。

3.插值的实现
插值的实现时候需要 input 和 output 对应到的 input 的坐标。
然后需要对计算得到的坐标四周进行采样，得到最后的输出值。

```
def _interpolate(im, x, y, out_height, out_width):
    # *_f are floats
    num_batch, height, width, channels = im.shape
    height_f = T.cast(height, theano.config.floatX)
    width_f = T.cast(width, theano.config.floatX)

    # clip coordinates to [-1, 1]
    x = T.clip(x, -1, 1)
    y = T.clip(y, -1, 1)

    # scale coordinates from [-1, 1] to [0, width/height - 1]
    x = (x + 1) / 2 * (width_f - 1)
    y = (y + 1) / 2 * (height_f - 1)

    # obtain indices of the 2x2 pixel neighborhood surrounding the coordinates;
    # we need those in floatX for interpolation and in int64 for indexing. for
    # indexing, we need to take care they do not extend past the image.
    x0_f = T.floor(x)
    y0_f = T.floor(y)
    x1_f = x0_f + 1
    y1_f = y0_f + 1
```

```
    x0 = T.cast(x0_f, 'int64')
    y0 = T.cast(y0_f, 'int64')
    x1 = T.cast(T.minimum(x1_f, width_f - 1), 'int64')
    y1 = T.cast(T.minimum(y1_f, height_f - 1), 'int64')

    # The input is [num_batch, height, width, channels]. We do the lookup in
    # the flattened input, i.e [num_batch*height*width, channels]. We need
    # to offset all indices to match the flat version
    dim2 = width
    dim1 = width*height
    base = T.repeat(
        T.arange(num_batch, dtype='int64')*dim1, out_height*out_width)
    base_y0 = base + y0*dim2
    base_y1 = base + y1*dim2
    idx_a = base_y0 + x0
    idx_b = base_y1 + x0
    idx_c = base_y0 + x1
    idx_d = base_y1 + x1

    # use indices to lookup pixels for all samples
    im_flat = im.reshape((-1, channels))
    Ia = im_flat[idx_a]
    Ib = im_flat[idx_b]
    Ic = im_flat[idx_c]
    Id = im_flat[idx_d]

    # calculate interpolated values
    wa = ((x1_f-x) * (y1_f-y)).dimshuffle(0, 'x')
    wb = ((x1_f-x) * (y-y0_f)).dimshuffle(0, 'x')
    wc = ((x-x0_f) * (y1_f-y)).dimshuffle(0, 'x')
    wd = ((x-x0_f) * (y-y0_f)).dimshuffle(0, 'x')
    output = T.sum([wa*Ia, wb*Ib, wc*C, wd*Id], axis=0)
    return output
```

其他的参考资料：

1. https://github.com/Lasagne/Lasagne
2. http://www.zhihu.com/question/20666664