Weekly Work Report

Liangjie Cao

VISION@OUC

August 26, 2018

1 This week's work

This week I and my partner Guan Cheng start to help our brother's work of video classification. Our first tqask is to make two lists. One of it is the train list, the other is val list. Our work is to get the results of UCF101 dataset and hmdb51 dataset with using non-local network. Actually we do have some problems at first, we finally make it. The accuracy of the UCF101 dataset is 96% in top_1 and 99% in top_5. Here's the non-local block network in Figure 1.

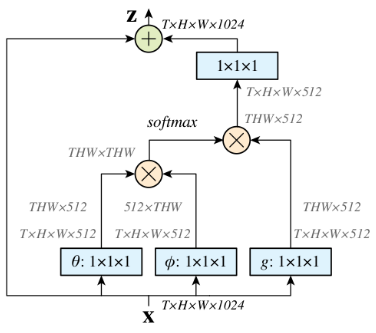


Figure 1: Non-local block network

2 ResNet

It uses a connection method called "shortcut connection". As the name implies, shortcut means "cutting the road". We can roughly understand the following figure 2:

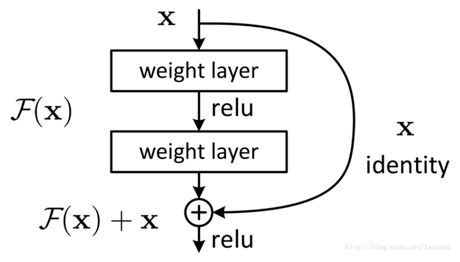
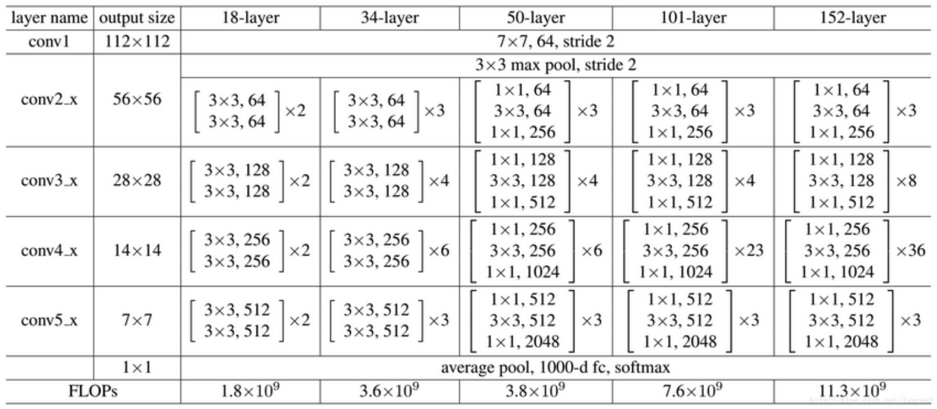

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: Non-local block network

Here, ResNet50 and ResNet101 are specially proposed, mainly because of their high appearance rate, so special instructions are needed. Given their specific structure in the following figure 3:

First, let's take a look at the picture. The above five depths of ResNet are proposed, which are 18, 34, 50, 101 and 152. First, look at the left side of the picture. We found



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3: Non-local block network

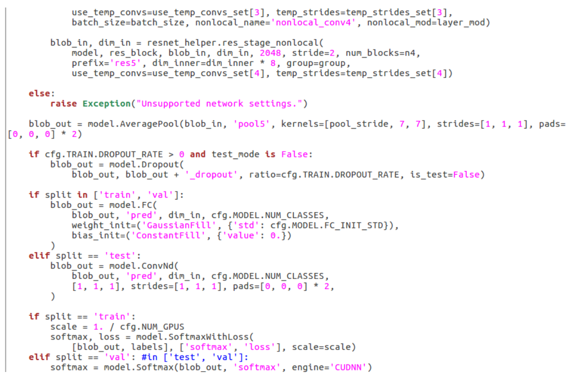
that all the networks are divided into five parts, namely: conv1. Conv2_x, conv3_x, conv4_x, conv5_x, and other papers later use this term to refer to each part of ResNet50 or 101.Take the 101-layer column, let's see if 101-layer is really a 101-layer network. First, we have a 7x7x64 convolution, then 3 + 4 + 23 + 3 = 33 building blocks, each block. It is 3 layers, so there are 33 x 3 = 99 layers, and finally there is an fc layer (for classification), so 1 + 99 + 1 = 101 layers, there is indeed a 101 layer network.(Actually I only know CNN now)

3 The basic codes of non-local network

I start to see the origin code about how the network is established with online help. To be honest, I can only know how it work but I cannot understand why it can work by using these several lines of code. Here's the create_model function about the network in figure 4, it's construction is similar to the ResNet101. ResNet used in network structure and image classification has some differences in dimensions, but the overall network structure still contains conv1 to conv5_x and some pooled and fully connected layers like ResNet. For ResNet-50, conv2_x, conv3_x, conv4_x, conv5_x The number of blocks is 3, 4, 6, and 3, respectively, for ResNet-101, 3, 4, 23, and 3, respectively.

Actually I feel difficult at first, so I listened to my partner's advice. I try to type the basic code of CNN learned from Coursera for a full afternoon. It's really useful, thought I don't know how it works, I know the process of building a network is not a familiar work for me. I need these experience to enrich myself. CNN is a typical model for the beginner to learn. The connection is from my github `https://github.com/Caoliangjie/CNN/blob/master/CNN.py`.

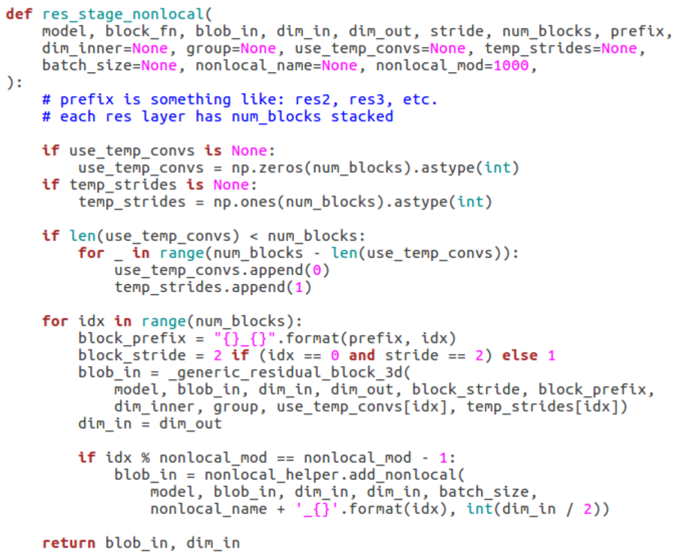
I have known how to build the basic model step by step. The process is obvious. Convolutional layer to pooling layer are all the basic unit. Non-local network's speciality is using the block called the non-local block. The code is also can be seen in figure 5. For ResNet-50's conv3_x and conv4_x, nonlocal_mod is 2, So when num_blocks==4



```
            use_temp_convs=use_temp_convs_set[3], temp_strides=temp_strides_set[3],
            batch_size=batch_size, nonlocal_name='nonlocal_conv4', nonlocal_mod=layer_mod)

        blob_in, dim_in = resnet_helper.res_stage_nonlocal(
            model, res_block, blob_in, dim_in, 2048, stride=2, num_blocks=n4,
            prefix='res5', dim_inner=dim_inner * 8, group=group,
            use_temp_convs=use_temp_convs_set[4], temp_strides=temp_strides_set[4])

    else:
        raise Exception("Unsupported network settings.")

    blob_out = model.AveragePool(blob_in, 'pool5', kernels=[pool_stride, 7, 7], strides=[1, 1, 1], pads=
[0, 0, 0] * 2)

    if cfg.TRAIN.DROPOUT_RATE > 0 and test_mode is False:
        blob_out = model.Dropout(
            blob_out, blob_out + '_dropout', ratio=cfg.TRAIN.DROPOUT_RATE, is_test=False)

    if split in ['train', 'val']:
        blob_out = model.FC(
            blob_out, 'pred', dim_in, cfg.MODEL.NUM_CLASSES,
            weight_init=('GaussianFill', {'std': cfg.MODEL.FC_INIT_STD}),
            bias_init=('ConstantFill', {'value': 0.})
        )
    elif split == 'test':
        blob_out = model.ConvNd(
            blob_out, 'pred', dim_in, cfg.MODEL.NUM_CLASSES,
            [1, 1, 1], strides=[1, 1, 1], pads=[0, 0, 0] * 2,
        )

    if split == 'train':
        scale = 1. / cfg.NUM_GPUS
        softmax, loss = model.SoftmaxWithLoss(
            [blob_out, labels], ['softmax', 'loss'], scale=scale)
    elif split == 'val': #in ['test', 'val']:
        softmax = model.Softmax(blob_out, 'softmax', engine='CUDNN')
```

Figure 4: Code 4

(*e.g.* conv3_x of ResNet-50), Then idx==1 or 3 will execute the non local operation, and the output blob_in obtained by the non local operation will overwrite the blob_in obtained by the regular block construct. Therefore, the effect is that every other block will be constructed in a non local manner. The last parameter that emphasizes the add_nonlocal function is int(dim_in / 2),That is, half of the number of input channels, and the number of channels is equal.



```
def res_stage_nonlocal(
    model, block_fn, blob_in, dim_in, dim_out, stride, num_blocks, prefix,
    dim_inner=None, group=None, use_temp_convs=None, temp_strides=None,
    batch_size=None, nonlocal_name=None, nonlocal_mod=1000,
):
    # prefix is something like: res2, res3, etc.
    # each res layer has num_blocks stacked

    if use_temp_convs is None:
        use_temp_convs = np.zeros(num_blocks).astype(int)
    if temp_strides is None:
        temp_strides = np.ones(num_blocks).astype(int)

    if len(use_temp_convs) < num_blocks:
        for _ in range(num_blocks - len(use_temp_convs)):
            use_temp_convs.append(0)
            temp_strides.append(1)

    for idx in range(num_blocks):
        block_prefix = "{}_{}".format(prefix, idx)
        block_stride = 2 if (idx == 0 and stride == 2) else 1
        blob_in = _generic_residual_block_3d(
            model, blob_in, dim_in, dim_out, block_stride, block_prefix,
            dim_inner, group, use_temp_convs[idx], temp_strides[idx])
        dim_in = dim_out

        if idx % nonlocal_mod == nonlocal_mod - 1:
            blob_in = nonlocal_helper.add_nonlocal(
                model, blob_in, dim_in, dim_in, batch_size,
                nonlocal_name + '_{}'.format(idx), int(dim_in / 2))

    return blob_in, dim_in
```

Figure 5: nlnet function

4 Basic Work II

At Friday brother told us we should make sure how the network is built, and maybe we could build the network by ourselves. So I wonder how to do this first. The previous model I've test in my computer is C3D [1]. I try to test on the server. We have two free GPUs to use, so I download the floders from their github and construct an virtual environment. During the traininig process, I wonder if I can transfer the i3D model to fit in the C3D training process. This is not a easy thing for us. Up to now, the training process hasn't be finished. To be honest, this method is a fundamental way for video recognition, very similiar to the two-stream. Anyway, I have to wait for the training result and test tomorrow. Then trying to transplant i3D in this framework.

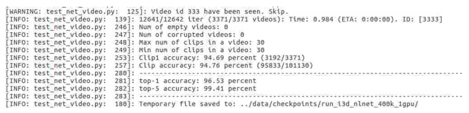
5 Some work in this week

Figure 7 shows the training process of C3D, and figure 6 is the work log from the non-local network for video recognition.

6 Conclusion

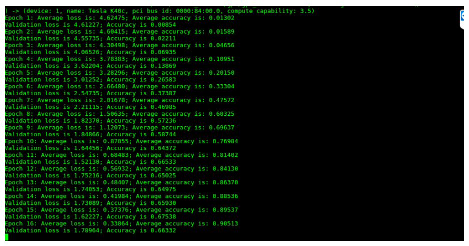
Through this week's work, I learned that I still have a long way to go to video behavior recognition. This is not a simple matter, in order to ensure the smooth progress of future work. My plan is as follows:

1. communicate with my partner and the brother more, solve the question at the same day.
2. Understand the specific usage of the model in the paper.
3. Practice makes perfect, wheras for work or life.



```
[WARNING: test_net_video.py:  125]: Video id 333 have been seen. Skip.
[INFO: test_net_video.py:  139]: 12641/12642 iter (3371/3371 videos): Time: 0.984 (ETA: 0:00:00). ID: [3333]
[INFO: test_net_video.py:  246]: Num of empty videos: 0
[INFO: test_net_video.py:  247]: Num of corrupted videos: 0
[INFO: test_net_video.py:  248]: Max num of clips in a video: 30
[INFO: test_net_video.py:  249]: Min num of clips in a video: 30
[INFO: test_net_video.py:  253]: Clip1 accuracy: 94.69 percent (3192/3371)
[INFO: test_net_video.py:  257]: Clip accuracy: 94.76 percent (95833/101130)
[INFO: test_net_video.py:  280]: ---------------------------------------------------------------------------------
[INFO: test_net_video.py:  281]: top-1 accuracy: 96.53 percent
[INFO: test_net_video.py:  282]: top-5 accuracy: 99.41 percent
[INFO: test_net_video.py:  283]: ---------------------------------------------------------------------------------
[INFO: test_net_video.py:  180]: Temporary file saved to: ../data/checkpoints/run_i3d_nlnet_400k_1gpu/
```

Figure 6: Non-local test process



```
) -> (device: 1, name: Tesla K40c, pci bus id: 0000:84:00.0, compute capability: 3.5)
Epoch 1: Average loss is: 4.62475; Average accuracy is: 0.01302
Validation loss is 4.61227; Accuracy is 0.00854
Epoch 2: Average loss is: 4.60415; Average accuracy is: 0.01589
Validation loss is 4.55735; Accuracy is 0.02211
Epoch 3: Average loss is: 4.30498; Average accuracy is: 0.04656
Validation loss is 4.06526; Accuracy is 0.06935
Epoch 4: Average loss is: 3.78383; Average accuracy is: 0.10951
Validation loss is 3.62204; Accuracy is 0.13869
Epoch 5: Average loss is: 3.28296; Average accuracy is: 0.20150
Validation loss is 3.01252; Accuracy is 0.26583
Epoch 6: Average loss is: 2.66480; Average accuracy is: 0.33304
Validation loss is 2.54735; Accuracy is 0.37387
Epoch 7: Average loss is: 2.01678; Average accuracy is: 0.47572
Validation loss is 2.21115; Accuracy is 0.46985
Epoch 8: Average loss is: 1.50635; Average accuracy is: 0.60325
Validation loss is 1.82370; Accuracy is 0.57236
Epoch 9: Average loss is: 1.12073; Average accuracy is: 0.69637
Validation loss is 1.84866; Accuracy is 0.58744
Epoch 10: Average loss is: 0.87055; Average accuracy is: 0.76984
Validation loss is 1.64456; Accuracy is 0.64372
Epoch 11: Average loss is: 0.68483; Average accuracy is: 0.81402
Validation loss is 1.52130; Accuracy is 0.66533
Epoch 12: Average loss is: 0.56932; Average accuracy is: 0.84130
Validation loss is 1.75216; Accuracy is 0.65025
Epoch 13: Average loss is: 0.48407; Average accuracy is: 0.86370
Validation loss is 1.74053; Accuracy is 0.64975
Epoch 14: Average loss is: 0.41984; Average accuracy is: 0.88536
Validation loss is 1.73089; Accuracy is 0.65930
Epoch 15: Average loss is: 0.37376; Average accuracy is: 0.89537
Validation loss is 1.62227; Accuracy is 0.67538
Epoch 16: Average loss is: 0.33864; Average accuracy is: 0.90513
Validation loss is 1.78964; Accuracy is 0.66332
```

Figure 7: C3D training process

References

[1] Chestnut. C3d completes video motion recognition. https://zhuanlan.zhihu.com/p/32934943. 5