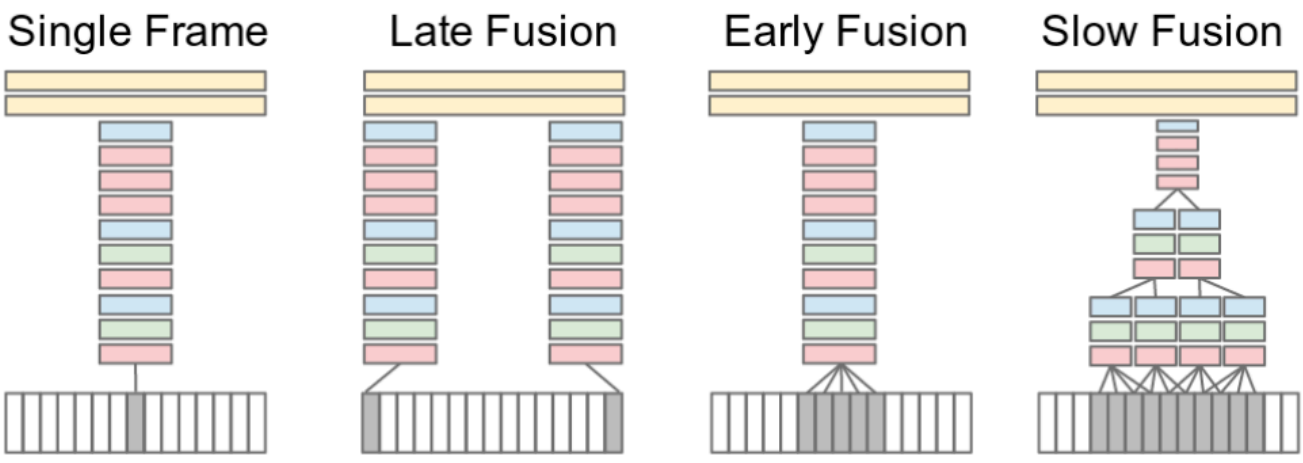
Video Understanding Skewers

> 笔者最近发现了视频理解这个很好玩的领域，也似乎非常promosing的领域。首先，视频的图像有着天然的数据加强；其次，视频中含有时序信息，这是单张图像无法媲美的；最后，在如今数据集larger and larger的时代，收集大量图片成本非常高昂，不如视频数据丰富。

DeepVideo

> Large-scale Video Classification with Convolutional Neural Networks(CVPR 2014)

本文主要贡献是首次将卷积神经网络从图片识别应用到视频识别里。而图片和视频的主要区别就是多了个时间轴。为此，作者尝试了以下四种结构。



第一种是单帧图片形式，它随便选了一帧，通过CNN进行分类，完全没有做到时间轴的上下文交互。

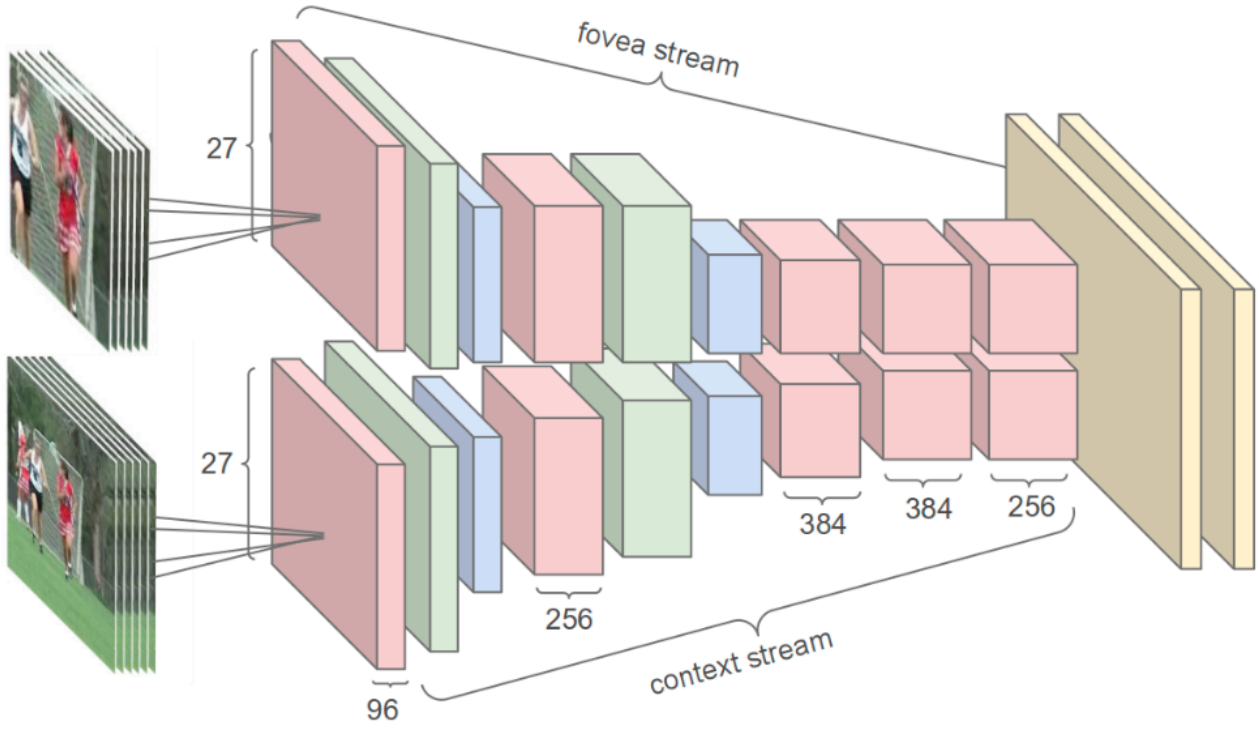
第二种随便抽取两帧，分别通过CNN进行特征抽取，后面通过一种称为Late Fusion的加权求和得到分类结果，这种稍微有点时序信息。

第三种则是在输入时直接把几个帧在channel上合并，称为Early Fusion，能在输入时就感受时序信息。

第四种是Early Fusion和Late Fusion的结合，称为Slow Fusion。如图所示，非常直观。

然而干不过手工特征～

于是作者又尝试了另外的思路。



这种架构类似于双流网络，但是两个网络权重共享，同时上面那个是在中间裁剪，所以可以看作是一种注意力机制，强行聚焦于中心。

但是还是干不过手工特征～

So people started to search for a new method.

双流网络

> Two-Stream Convolutional Networks for Action Recognition in Videos(CVPR 2014)

传送门

Beyond Short Snippets

> Beyond Short Snippets: Deep Networks for Video Classification

本文是在探索特征抽取后如何融合的问题。

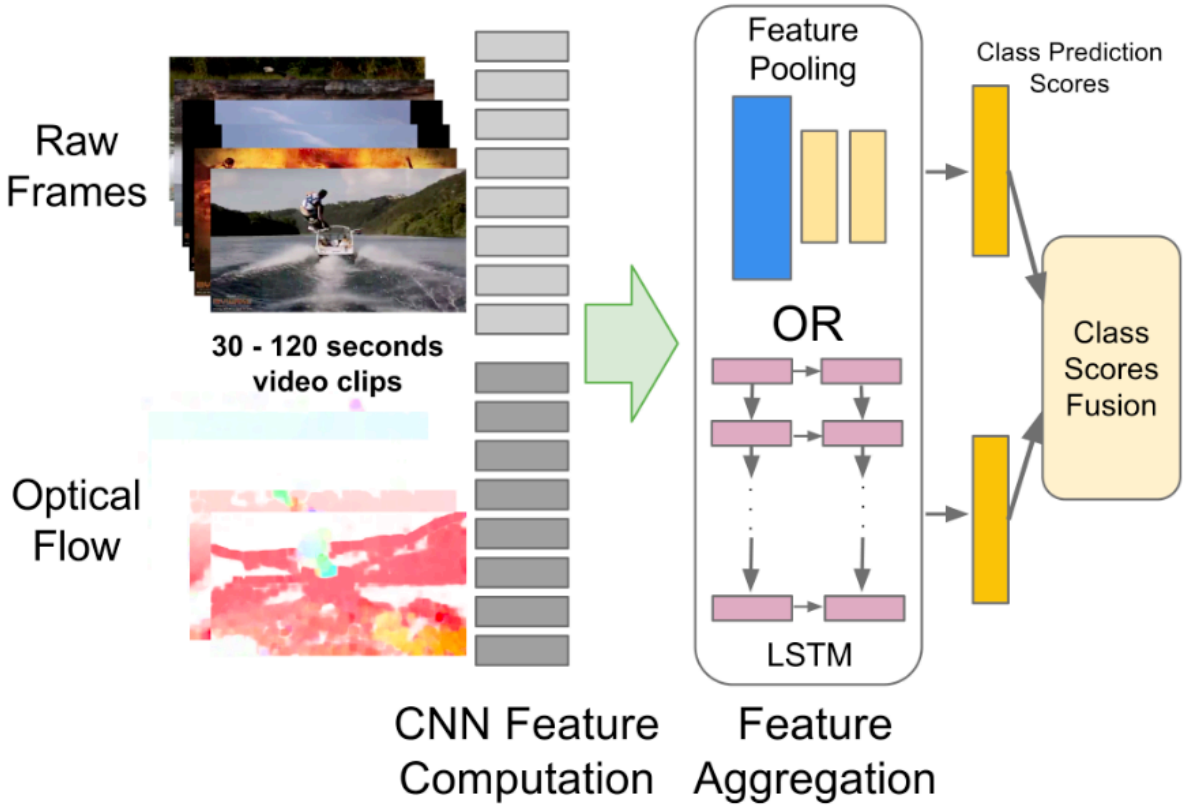
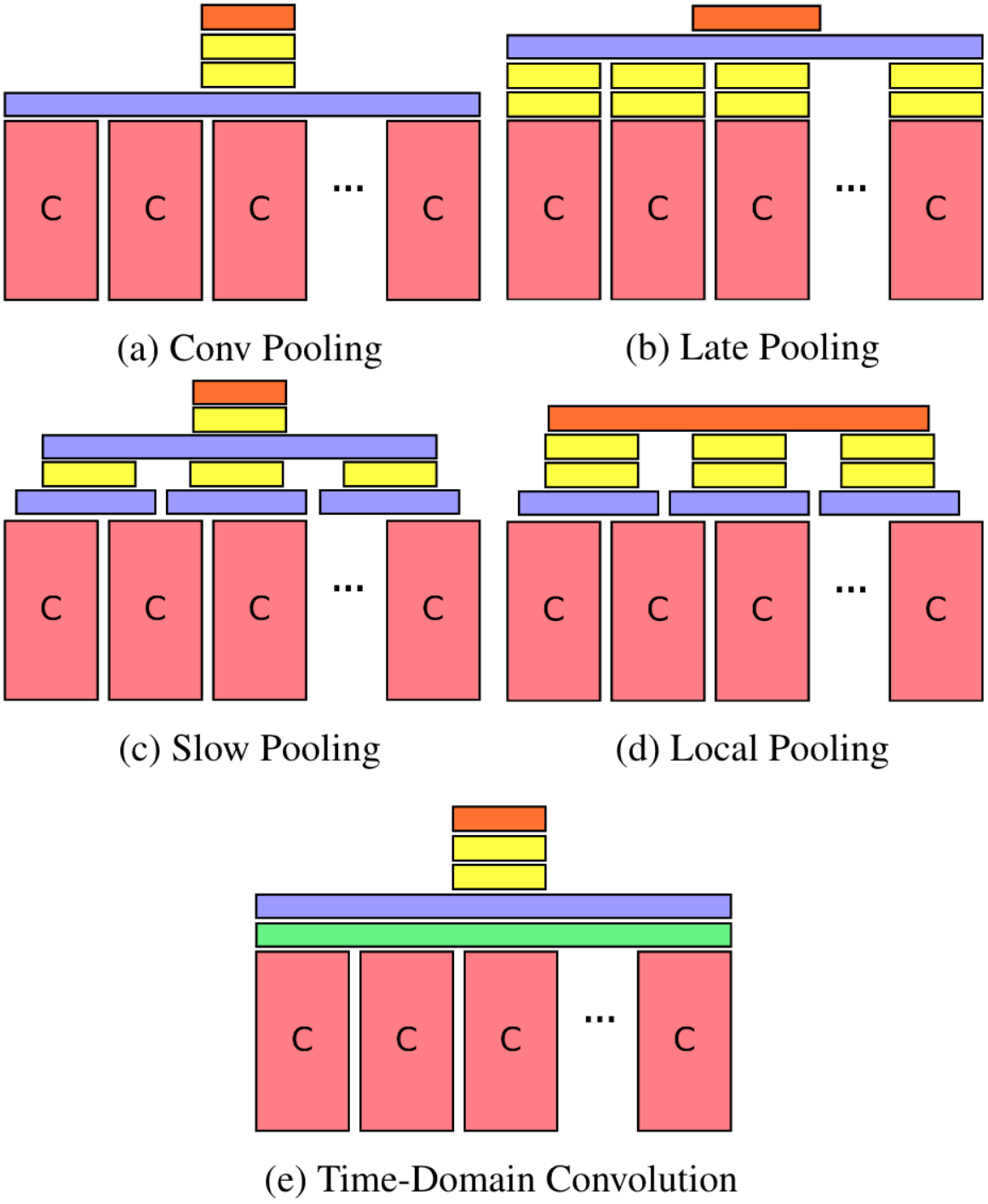


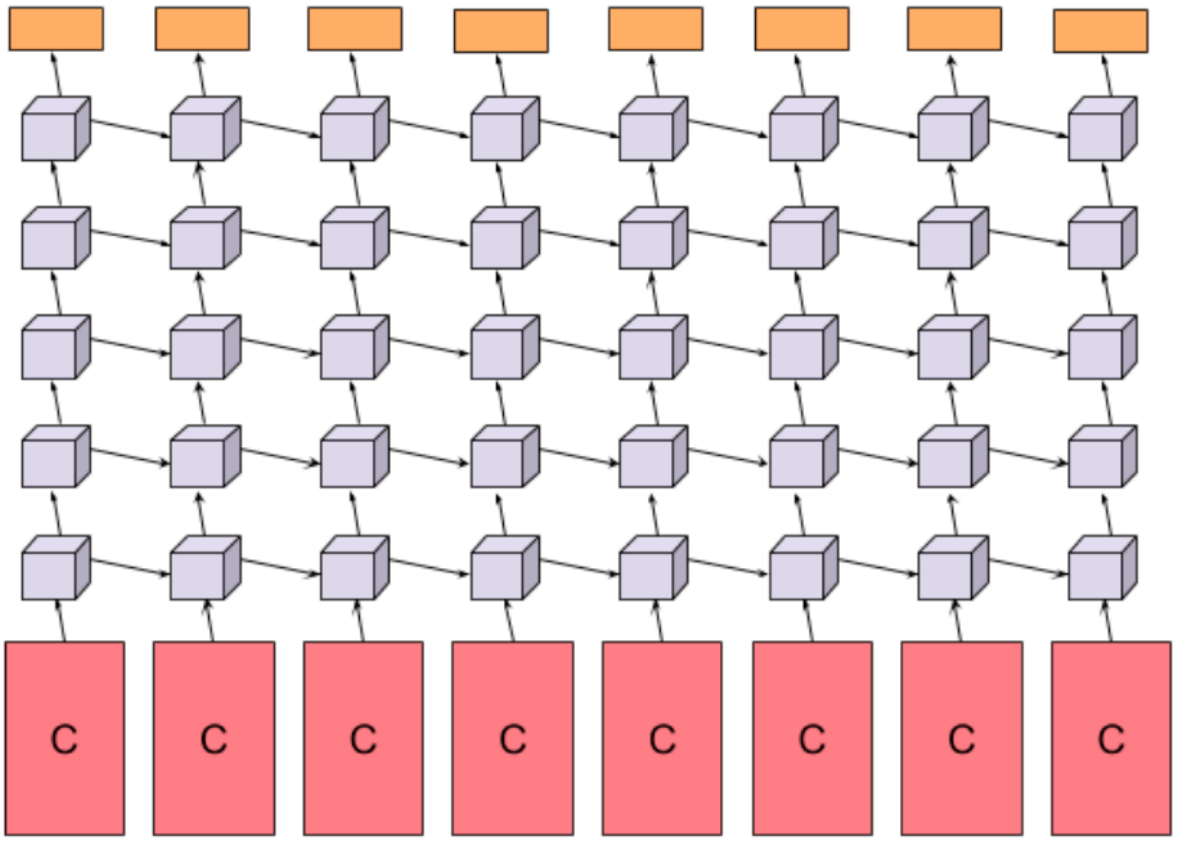
Figure 1: Overview of our approach.

主要架构还是双流网络，作者分别尝试了使用Pooling和LSTM进行特征融合。

直接上结论，Pooling是conv pooling最有效；



而对于LSTM做时序信息融合，红色的框框是抽取的特征，中间的灰色block是LSTM网络，最后黄色的是softmax。



最后对比发现，LSTM并没有非常大优势，原因可能是输入的视频太短（只有几十帧），LSTM只有在处理long-term信息才能发挥对时间感受的优势。

Convolutional Fusion

> Convolutional Two-Stream Network Fusion for Video Action Recognition

看起来标题和双流网络很像，实际上内容也确实很像（）

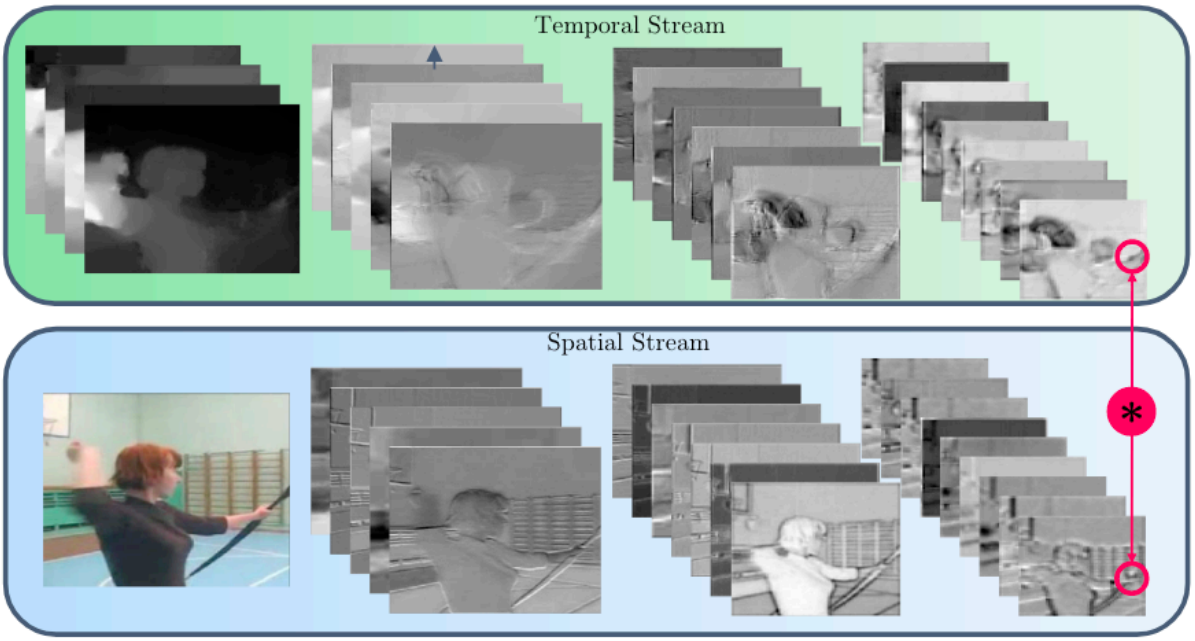


Figure 1. Example outputs of the first three convolutional layers from a two-stream ConvNet model [22]. The two networks separately capture spatial (appearance) and temporal information at a fine temporal scale. In this work we investigate several approaches to fuse the two networks over space and time.

本文主要也探索了特征融合的方式。

Spatial fusion

最简单粗暴的是Sum fusion直接加起来。

Sum fusion. $\mathbf{y}^{\text{sum}} = f^{\text{sum}}(\mathbf{x}^a, \mathbf{x}^b)$ computes the sum of the two feature maps at the same spatial locations i, j and

feature channels d:

$$y^{\text{sum}}_{i,j,d} = x^a_{i,j,d} + x^b_{i,j,d}, \tag{1}$$

where $1 \leq i \leq H, 1 \leq j \leq W, 1 \leq d \leq D$ and $\mathbf{x}^a, \mathbf{x}^b, \mathbf{y} \in \mathbb{R}^{H \times W \times D}$

Since the channel numbering is arbitrary, sum fusion simply defines an arbitrary correspondence between the networks. Of course, subsequent learning can employ this arbitrary correspondence to its best effect, optimizing over the filters of each network to make this correspondence useful.

第二种是Max fuison，简单来说不同通道同一位置，哪个数值大取哪个。

Max fusion. $\mathbf{y}^{\max} = f^{\max}(\mathbf{x}^a, \mathbf{x}^b)$ similarly takes the maximum of the two feature map:

$$y^{\max}_{i,j,d} = \max\{x^a_{i,j,d}, x^b_{i,j,d}\}, \tag{2}$$

where all other variables are defined as above (1).

Similarly to sum fusion, the correspondence between network channels is again arbitrary.

第三种是Concatenation fusion，在某个维度上直接把两个特征图拼接起来。

Concatenation fusion. $\mathbf{y}^{\text{cat}} = f^{\text{cat}}(\mathbf{x}^a, \mathbf{x}^b)$ stacks the two feature maps at the same spatial locations i, j across the feature channels d:

$$y^{\text{cat}}_{i,j,2d} = x^a_{i,j,d} \qquad y^{\text{cat}}_{i,j,2d-1} = x^b_{i,j,d}, \tag{3}$$

where $\mathbf{y} \in \mathbb{R}^{H \times W \times 2D}$.

Concatenation does not define a correspondence, but leaves this to subsequent layers to define (by learning suitable filters that weight the layers), as we illustrate next.

第四种是Conv fusion，简单来说就是先拼起来，再对特征图再做个卷积。

Conv fusion. $\mathbf{y}^{\text{conv}} = f^{\text{conv}}(\mathbf{x}^a, \mathbf{x}^b)$ first stacks the two feature maps at the same spatial locations i, j across the feature channels d as above (3) and subsequently convolves the stacked data with a bank of filters $\mathbf{f} \in \mathbb{R}^{1\times1\times2D\times D}$ and biases $b \in \mathbb{R}^D$

$$\mathbf{y}^{\text{conv}} = \mathbf{y}^{\text{cat}} * \mathbf{f} + b, \tag{4}$$

where the number of output channels is D, and the filter has dimensions $1 \times 1 \times 2D$. Here, the filter $\mathbf{f}$ is used to reduce the dimensionality by a factor of two and is able to model weighted combinations of the two feature maps $\mathbf{x}^a, \mathbf{x}^b$ at the same spatial (pixel) location. When used as a trainable filter kernel in the network, $\mathbf{f}$ is able to *learn* correspondences of the two feature maps that minimize a joint loss function. For example, if $\mathbf{f}$ is learnt to be the concatenation of two permuted identity matrices $\mathbf{1}' \in \mathbb{R}^{1\times1\times D\times D}$, then the ith channel of the one network is only combined with the ith channel of the other (via summation).

Note that if there is no dimensionality reducing conv-layer injected after concatenation, the number of input channels of the upcoming layer is $2D$.

第五种是Bilinear fusion，就是对两个特征图做个outer product（叉乘）。

Bilinear fusion. $\mathbf{y}^{\text{bil}} = f^{\text{bil}}(\mathbf{x}^a, \mathbf{x}^b)$ computes a matrix outer product of the two features at each pixel location, followed by a summation over the locations:

$$\mathbf{y}^{\text{bil}} = \sum_{i=1}^{H}\sum_{j=1}^{W} \mathbf{x}_{i,j}^{a\top}\mathbf{x}_{i,j}^{b}. \tag{5}$$

同时，作者也探索了在哪里进行fusion比较好，提出了两种方式。

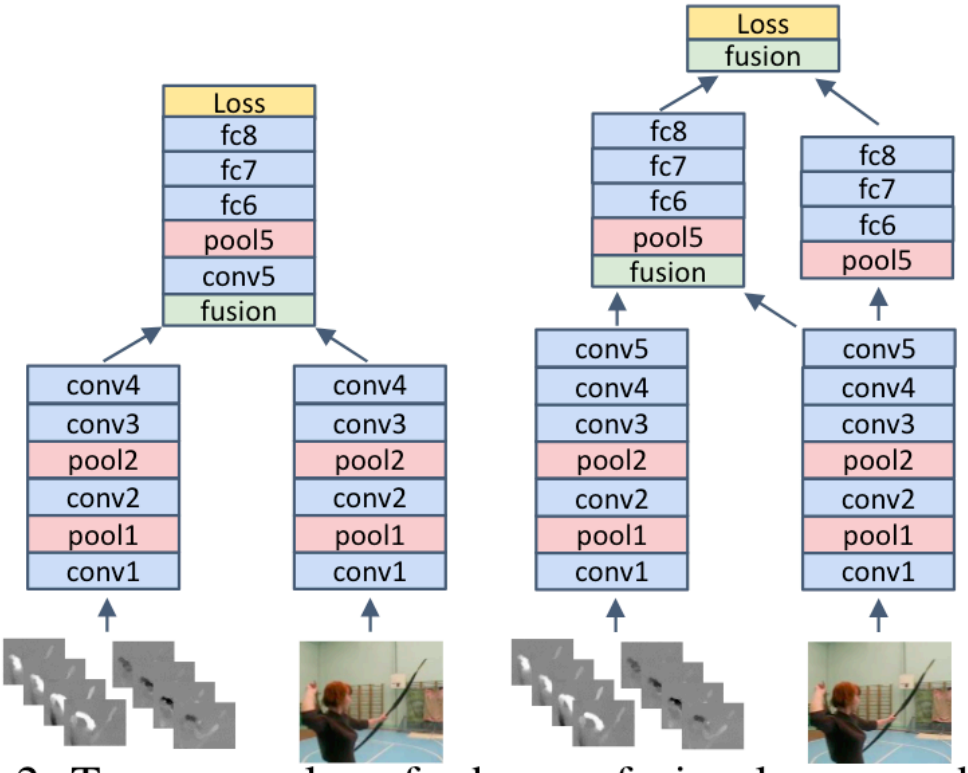


Figure 2. Two examples of where a fusion layer can be placed. The left example shows fusion after the fourth conv-layer. Only a single network tower is used from the point of fusion. The right figure shows fusion at two layers (after conv5 and after fc8) where both network towers are kept, one as a hybrid spatiotemporal net and one as a purely spatial network.

左边这种是在时间流和空间流分别做完了特征提取后，直接进行fusion，后面就没有fusion了；右边这种更加高校，是先进行一次fusion帮助时间流进行学习，后面在进行了fc后语义已经比较高级，再把空间流结果和之前融合的结果再进行一次fusion，这样就可以保持空间信息完整性。

Temporal fusion

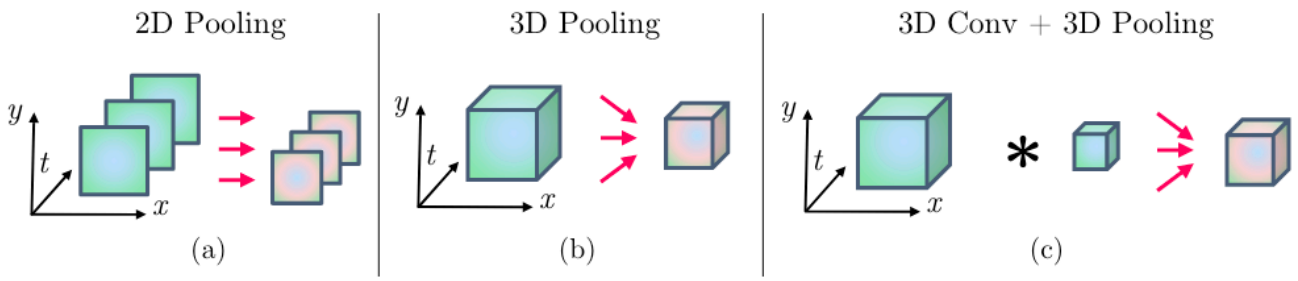


Figure 3. Different ways of fusing temporal information. (a) 2D pooling ignores time and simply pools over spatial neighbourhoods to individually shrink the size of the feature maps for each temporal sample. (b) 3D pooling pools from local spatiotemporal neighbourhoods by first stacking the feature maps across time and then shrinking this spatiotemporal cube. (c) 3D conv + 3D pooling additionally performs a convolution with a fusion kernel that spans the feature channels, space and time before 3D pooling.

作者选择了后面两种fusion方式。

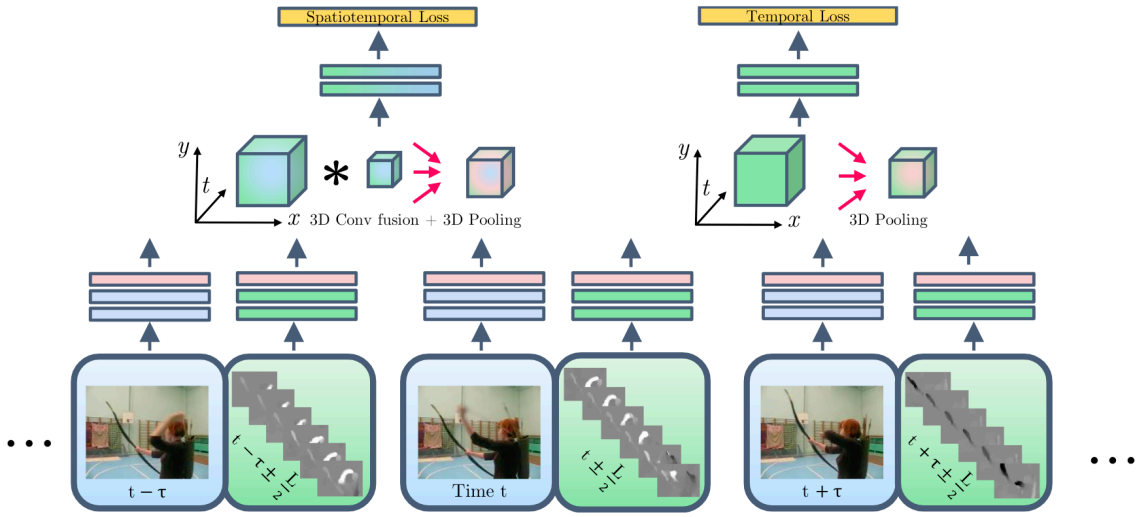
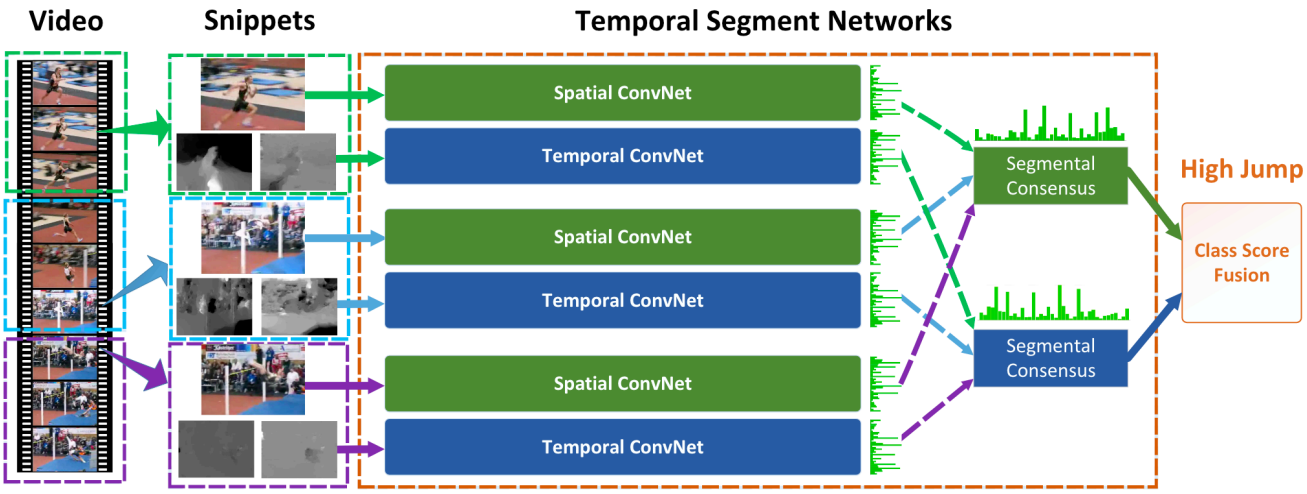


Figure 4. Our spatiotemporal fusion ConvNet applies two-stream ConvNets, that capture short-term information at a fine temporal scale ($t \pm \frac{L}{2}$), to temporally adjacent inputs at a coarse temporal scale ($t + T\tau$). The two streams are fused by a 3D filter that is able to learn correspondences between highly abstract features of the spatial stream (blue) and temporal stream (green), as well as local weighted combinations in x, y, t. The resulting features from the fusion stream and the temporal stream are 3D-pooled in space and time to learn spatiotemporal (top left) and purely temporal (top right) features for recognising the input video.

这是最终的框架。值得注意的是时空损失和时间损失分别计算（因为时间流非常重要，需要单独算）。

TSN

> Temporal Segment Networks: Towards Good Practices for Deep Action Recognition（ECCV2016）



本文的主要想法是把一个视频分成三段（不论长短），每一段随机抽出一帧，分别做空间流和时间流的特征提取；然后三个空间流经过Segmental Consensus进行融合，三个时间流也经过Segmental Consensus进行融合；最后时间流和空间流进行融合。

关于ImageNet预训练迁移学习，输入是个RGB 3通道，但是光流有十张图片，分成x，y两个方向就有20个channel了，你不可能凭空再变出9张图片吧。所以作者直接复制了20遍.....没错，我也不知道为什么这样work。（光流静态初始）

除此以外，因为初期数据集很小，所以BN很容易产生过拟合。你可能想直接冻住BN，但是冻住全部的话迁移学习效果就不太好了。所以作者提出了Partial BN，只留下输入层的BN来学习输入特征，后面的BN全部冻住。

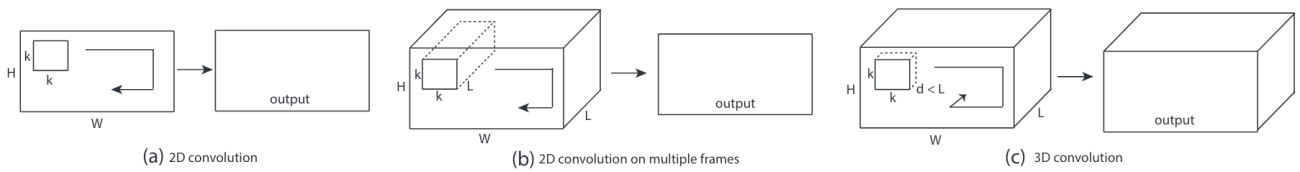
最后一个点是数据增强。（1）corner cropping，强制裁剪到边角位置，防止老是裁到中间位置。（2）scale-jittering，随机按以下几个值裁剪尺寸

$$\{256, 224, 192, 168\}$$

C3D

> Learning Spatiotemporal Features with 3D Convolutional Networks

因为光流法有个致命缺点，抽取光流非常非常耗费时间，几乎无法做到实时效果。所以人们转向了3D神经网络。



什么是3D神经网络呢？其实就是在2D网络上加了个时间维度变成了3D。卷积核也从1x3x3变成了3x3x3，可以同时对多个帧的图像做处理。

网络架构如下：

Figure 3. **C3D architecture**. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling layers are denoted from `pool1` to `pool5`. All pooling kernels are $2 \times 2 \times 2$, except for `pool1` is $1 \times 2 \times 2$. Each fully connected layer has 4096 output units.

这篇文章效果并不突出，计算量非常大，但是他提供了抽特征的接口，你可以啥都不用管就可以去做下游特征。

I3D

> Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset

重头戏来了，这文章主要贡献有两个，一个是提出了inflate方法，一个是提出了一个新数据集Kinetics。

首先是这令人疑惑的名字。

Figure 1. **A still from 'Quo Vadis' (1951).** Where is this going? Are these actors about to kiss each other, or have they just done so? More importantly, where is action recognition going? Actions can be ambiguous in individual frames, but the limitations of existing action recognition datasets has meant that the best-performing video architectures do not depart significantly from single-image analysis, where they rely on powerful image classifiers trained on ImageNet. In this paper we demonstrate that video models are best pre-trained on videos and report significant improvements by using spatio-temporal classifiers pre-trained on Kinetics, a freshly collected, large, challenging human action video dataset.

文章名字来源于一个电影，估计这作者是个推子。他狡辩道这：“你无法单单通过这图片知道人物下一个动作。”他们可能在靠近对方，也可能刚刚瑟瑟玩在远离对方，也可能下一秒美式居合干掉对方。

首先是inflate技术，其实就是你可以把一个设计好的2D网络架构改改（比如Res50），直接可以扩充为3D，用来分析视频。还可以通过一种名为bootstrapping的方法直接使用2D网络预训练出来的参数。

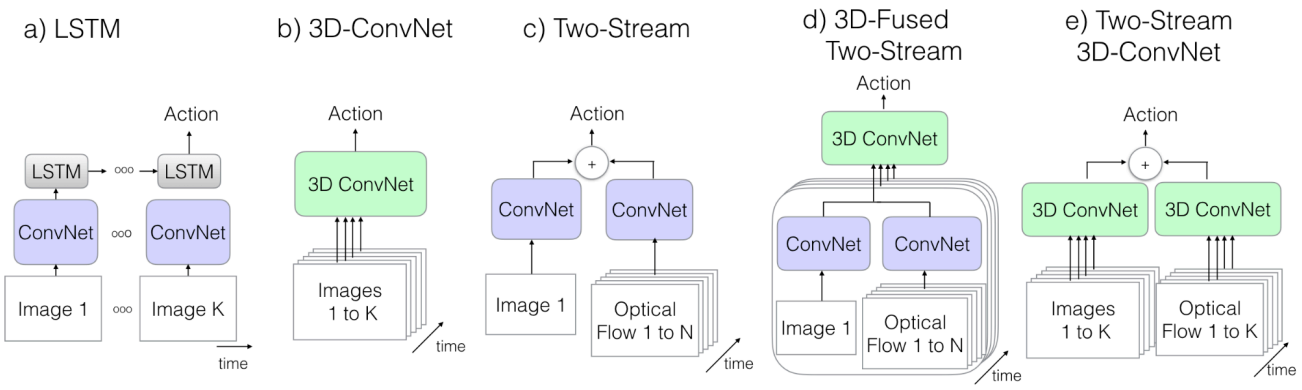
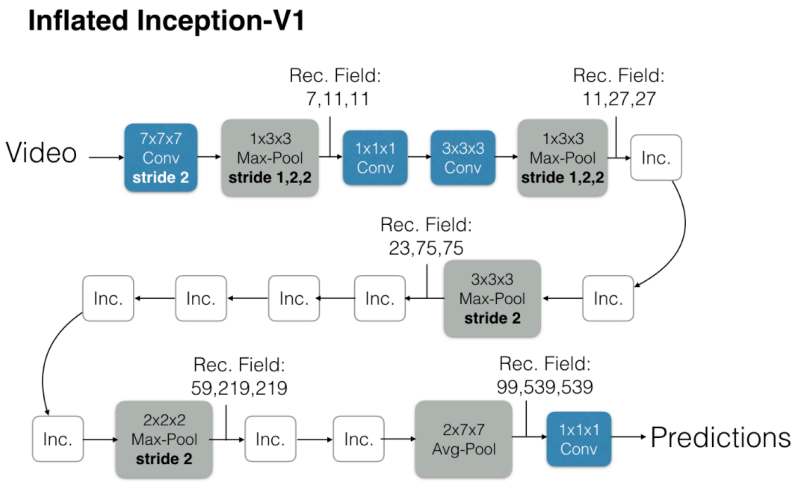
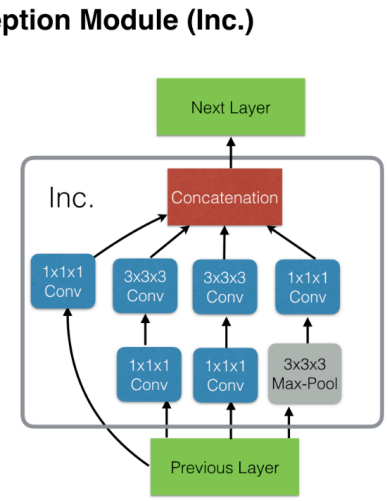


Figure 2. Video architectures considered in this paper. K stands for the total number of frames in a video, whereas N stands for a subset of neighboring frames of the video.

这是先前网络架构和作者架构（最右边那个）的比较。

关于如何证明可以直接拿2D的预训练参数用在初始化3D网络上呢？作者想了个妙法子，他将2D的输入图片复制了20遍成为一个视频（boring video），输入3D网络中，只要两个网络输出一致，就可以证明啦！





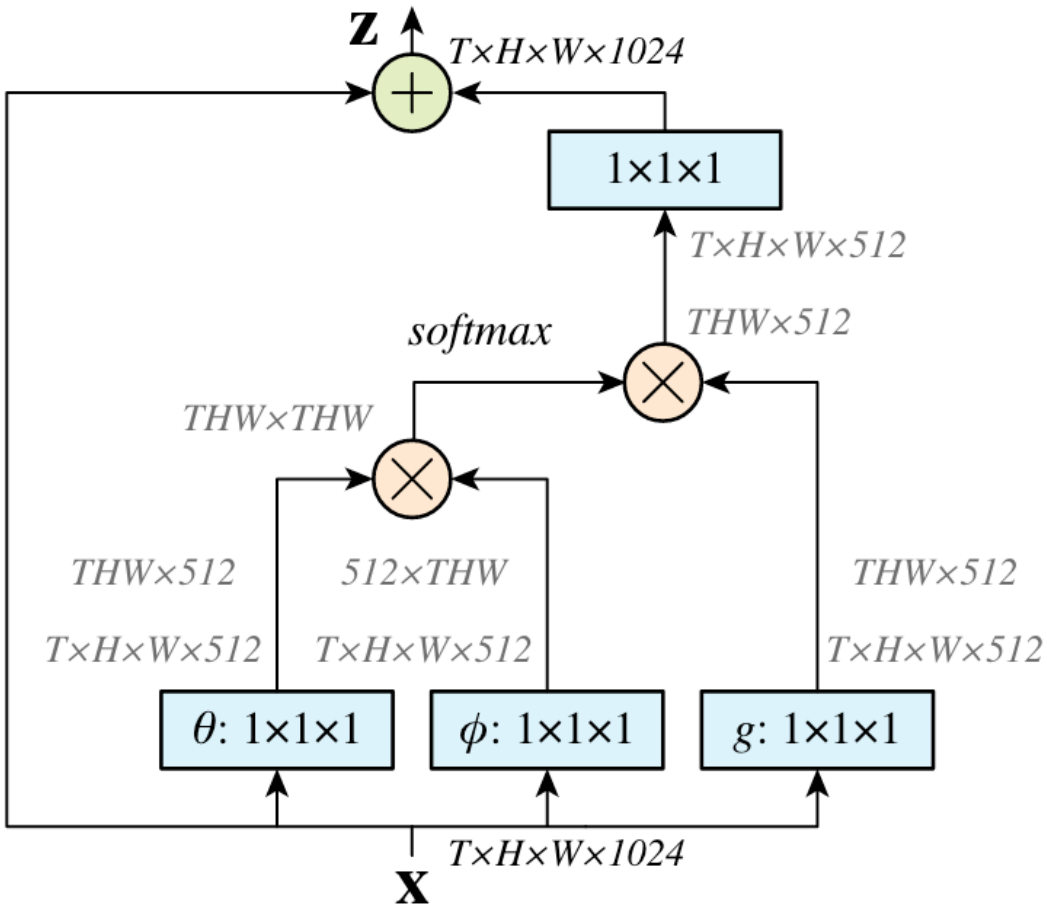
inflate就直接把卷积核从1X3X3变成3X3X3；同时为了保留时间信息（你视频就那么短），时间维度上就没有做下采样了，下采样全部用在另外两个维度上。

另外就是Kinetics，这个数据集更大，更难，而且之前的数据集老登已经被I3D刷到98%了，所以大家只能用Kinetics了。

Non-local

> Non-local Neural Networks

这篇主要贡献是使用了自注意力。



从左到右分别是K、Q、V，只不过是3D。

R（2+1）D

> A Closer Look at Spatiotemporal Convolutions for Action Recognition

本文作者想着2D CNN比3D CNN便宜那么多，有没有可能使用部分的2D来代替3D?

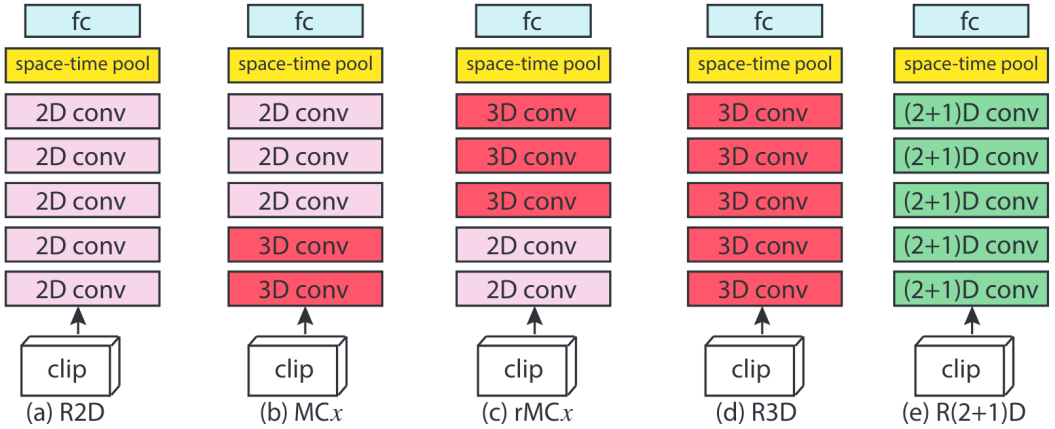


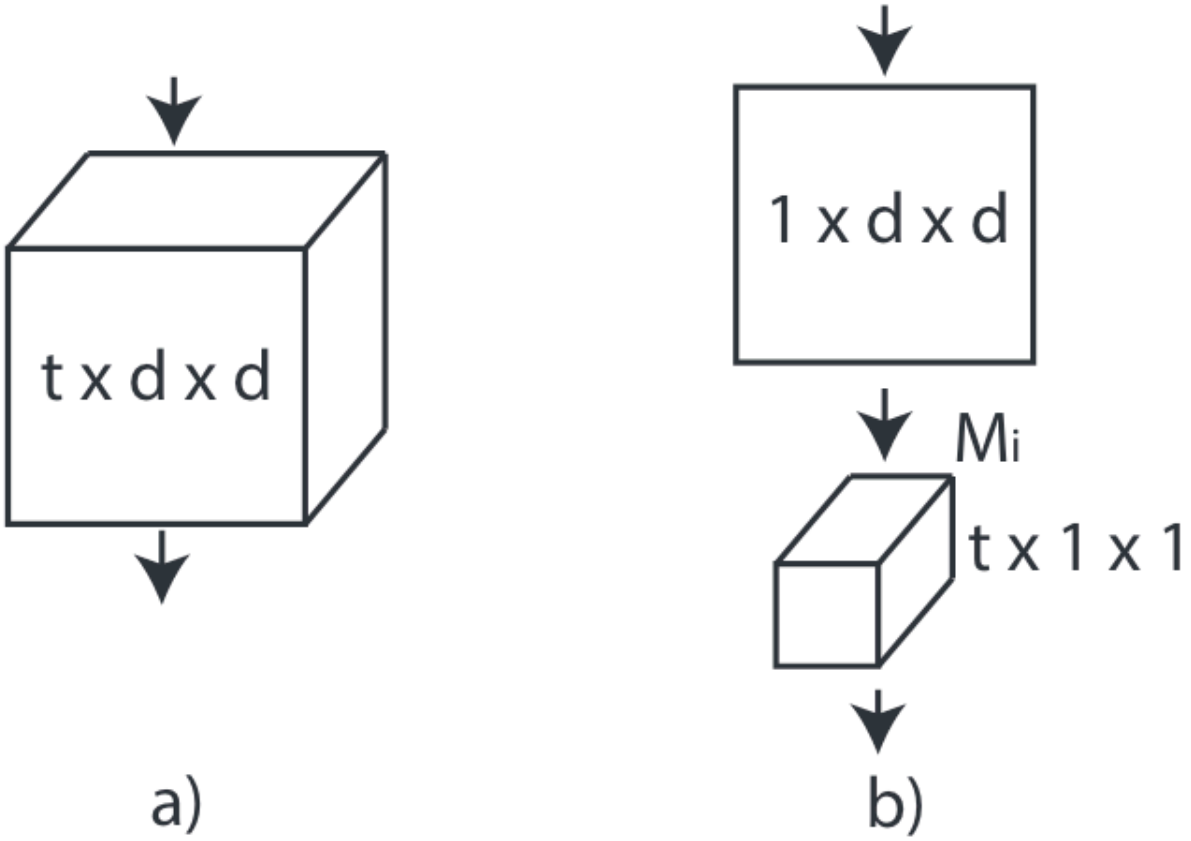
Figure 1. **Residual network architectures for video classification considered in this work**. (a) R2D are 2D ResNets; (b) MCx are ResNets with mixed convolutions (MC3 is presented in this figure); (c) rMCx use reversed mixed convolutions (rMC3 is shown here); (d) R3D are 3D ResNets; and (e) R(2+1)D are ResNets with (2+1)D convolutions. For interpretability, residual connections are omitted.

这是几种方法的对比，都被最后一个（作者的）爆杀了。

Net	**# params**	**Clip@1**	**Video@1**	**Clip@1**	**Video@1**
Input		8×112×112		16×112×112	
R2D	11.4M	46.7	59.5	47.0	58.9
f-R2D	11.4M	48.1	59.4	50.3	60.5
R3D	33.4M	49.4	61.8	52.5	64.2
MC2	11.4M	50.2	62.5	53.1	64.2
MC3	11.7M	50.7	62.9	53.7	64.7
MC4	12.7M	50.5	62.5	53.7	65.1
MC5	16.9M	50.3	62.5	53.7	65.1
rMC2	33.3M	49.8	62.1	53.1	64.9
rMC3	33.0M	49.8	62.3	53.2	65.0
rMC4	32.0M	49.9	62.3	53.4	65.1
rMC5	27.9M	49.4	61.2	52.1	63.1
R(2+1)D	33.3M	**52.8**	**64.8**	**56.8**	**68.0**

Table 2. **Action recognition accuracy for different forms of convolution on the Kinetics validation set**. All models are based on a ResNet of 18 layers, and trained from scratch on either 8-frame or 16-frame clip input. R(2+1)D outperforms all the other models.

那么究竟什么是R（2+1）D?



其实就是拆分成了一个2D和一个tx1x1的卷积核。这样可以在两层之间加一个非线性层，这样可以提高学习效果；同时也可以降低计算成本，方便优化。

SlowFast

> SlowFast Networks for Video Recognition

作者发现人体视觉细胞中p细胞处理场景信息，占80%；而m细胞处理运动信息，只占20%。所以作者想这不就是双流网络吗，可以改改啊。

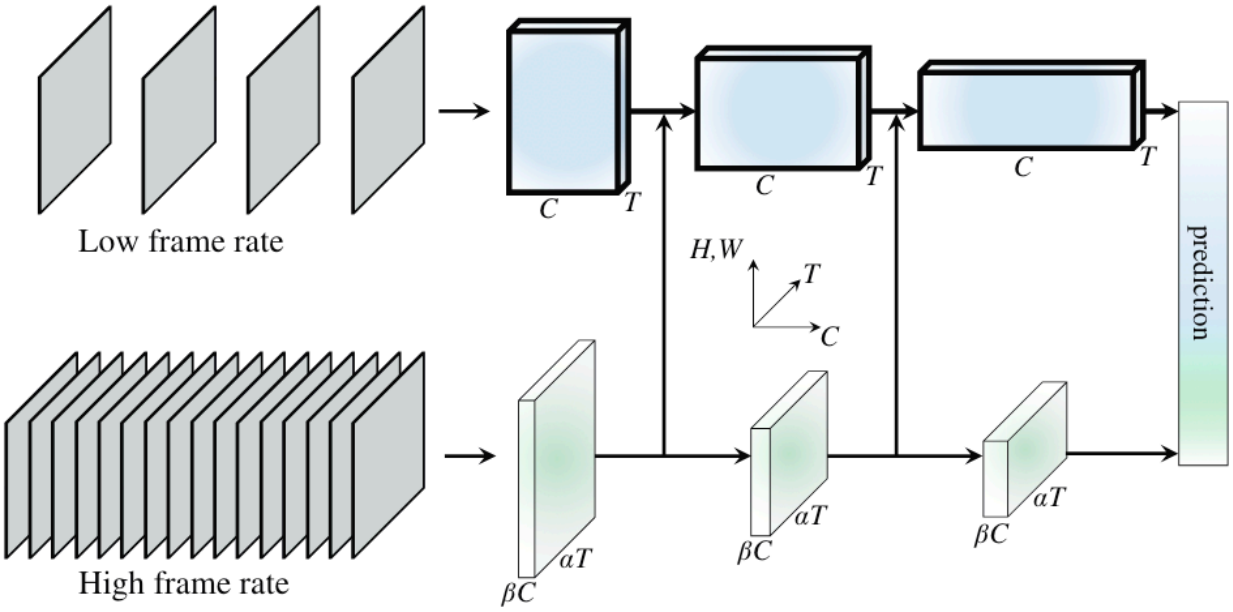


Figure 1. **A SlowFast network** has a low frame rate, low temporal resolution *Slow* pathway and a high frame rate, $\alpha\times$ higher temporal resolution *Fast* pathway. The Fast pathway is lightweight by using a fraction (β, *e.g.*, 1/8) of channels. Lateral connections fuse them.

上面的slow帧间比较长，负责学习静态信息，同时这一支网络结构比较大（小输入大网络）；下面的fast分支帧间隔比较短，负责处理运动信息，网络结构比较小（大输入小网络）。

stage	*Slow* pathway	*Fast* pathway	output sizes $T\times S^2$
raw clip	-	-	64×224^2
data layer	stride 16, 1^2	stride **2**, 1^2	*Slow* : 4×224^2 *Fast* : **32**$\times 224^2$
conv_1	1×7^2, 64 stride 1, 2^2	$\underline{5\times 7^2}$, 8 stride 1, 2^2	*Slow* : 4×112^2 *Fast* : **32**$\times 112^2$
pool_1	1×3^2 max stride 1, 2^2	1×3^2 max stride 1, 2^2	*Slow* : 4×56^2 *Fast* : **32**$\times 56^2$
res_2	$\begin{bmatrix} 1\times 1^2, 64 \\ 1\times 3^2, 64 \\ 1\times 1^2, 256 \end{bmatrix}\times 3$	$\begin{bmatrix} \underline{3\times 1^2}, 8 \\ 1\times 3^2, 8 \\ 1\times 1^2, 32 \end{bmatrix}\times 3$	*Slow* : 4×56^2 *Fast* : **32**$\times 56^2$
res_3	$\begin{bmatrix} 1\times 1^2, 128 \\ 1\times 3^2, 128 \\ 1\times 1^2, 512 \end{bmatrix}\times 4$	$\begin{bmatrix} \underline{3\times 1^2}, 16 \\ 1\times 3^2, 16 \\ 1\times 1^2, 64 \end{bmatrix}\times 4$	*Slow* : 4×28^2 *Fast* : **32**$\times 28^2$
res_4	$\begin{bmatrix} \underline{3\times 1^2}, 256 \\ 1\times 3^2, 256 \\ 1\times 1^2, 1024 \end{bmatrix}\times 6$	$\begin{bmatrix} \underline{3\times 1^2}, 32 \\ 1\times 3^2, 32 \\ 1\times 1^2, 128 \end{bmatrix}\times 6$	*Slow* : 4×14^2 *Fast* : **32**$\times 14^2$
res_5	$\begin{bmatrix} \underline{3\times 1^2}, 512 \\ 1\times 3^2, 512 \\ 1\times 1^2, 2048 \end{bmatrix}\times 3$	$\begin{bmatrix} \underline{3\times 1^2}, 64 \\ 1\times 3^2, 64 \\ 1\times 1^2, 256 \end{bmatrix}\times 3$	*Slow* : 4×7^2 *Fast* : **32**$\times 7^2$
global average pool, concate, fc			# classes

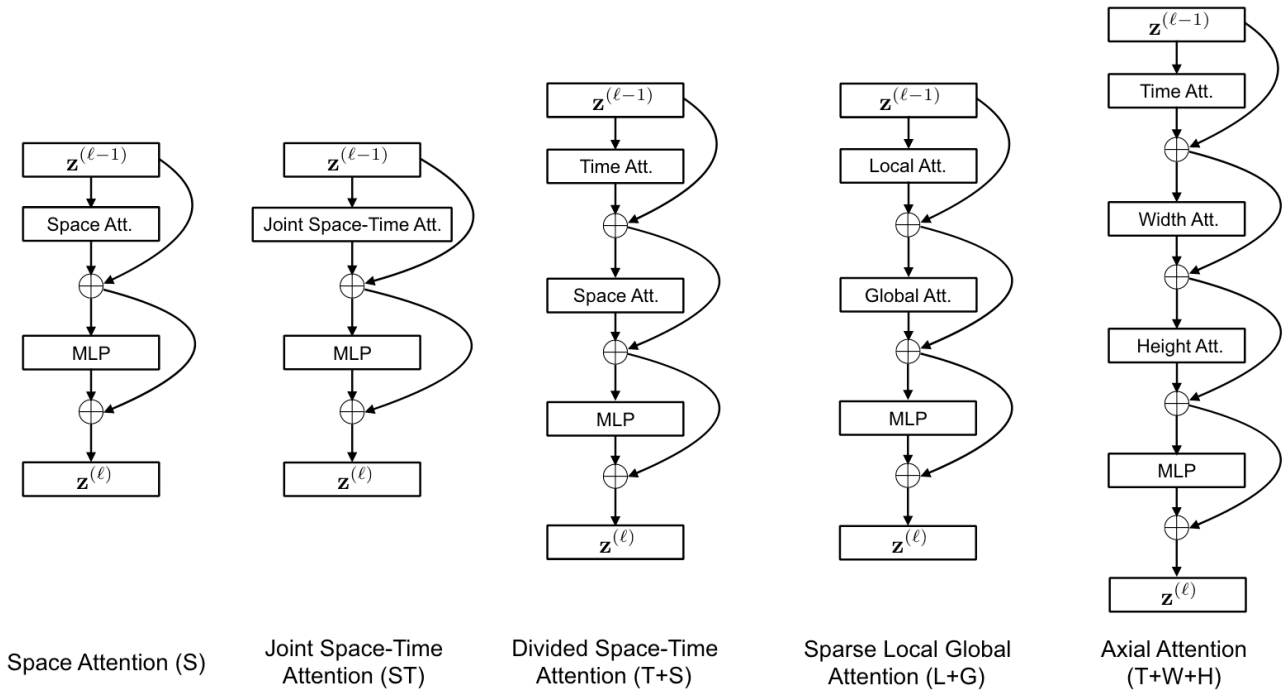
Table 1. **An example instantiation of the SlowFast network**. The dimensions of kernels are denoted by $\{T\times S^2, C\}$ for temporal, spatial, and channel sizes. Strides are denoted as {temporal stride, spatial stride^2}. Here the speed ratio is $\alpha = 8$ and the channel ratio is $\beta = 1/8$. τ is 16. The **green** colors mark *higher* temporal resolution, and orange colors mark *fewer* channels, for the Fast pathway. Non-degenerate temporal filters are underlined. Residual blocks are shown by brackets. The backbone is ResNet-50.

TimesFormer

> Is Space-Time Attention All You Need for Video Understanding?

本文使用了Transformer框架，

以下是不同的拆分方式，最右边那个是作者的最终框架。



第一种是只有空间，每一帧都输入，没有时序信息。

第二种是暴力塞，GPU表示阿米诺斯。

第三种，作者想到可以分成时间和空间两个维度上分别算注意力啊。

第四种，先在局部算，后面再算全局，类似SwinTransformer。

最后一种，先沿时间轴算，再沿横轴，最后沿竖轴。

作者给了可视化方式如下。