Massively Parallel Video Networks
Supplemental Material

João Carreira[†,1], Viorica Pătrăucean[†,1], Laurent Mazare[1], Andrew Zisserman[1,2], Simon Osindero[1]

[1]DeepMind
[2]Department of Engineering Science, University of Oxford
{joaoluis, viorica, mazare, zisserman, osindero}@google.com
[†]shared first author

1 Timing details with execution timelines

In this section we give more details on the timing measurements. Table 1 and table 2 report the average throughput of Par-DenseNet and Par-Inception models. GPU measurements have been done using Nvidia K-40 GPUs. We include the throughput in frames per second in these tables. These numbers are only indicative as they depend on the implementation (which is why they were not included in the original paper).

Model	# Par. Subnets	48 cores	2 GPUs	4 GPUs	8 GPUs
without multi-rate clocks					
sequential	1	4.9 (1.0×)	14.1 (1.0×)	14.1 (1.0×)	14.1 (1.0×)
semi-parallel	2	6.4 (1.3×)	22.3 (1.6×)	23.9 (1.7×)	23.9 (1.7×)
semi-parallel	4	8.8 (1.8×)	23.5 (1.7×)	35.7 (2.5×)	40.2 (2.9×)
semi-parallel	7	11.0 (2.2×)	22.3 (1.6×)	36.4 (2.6×)	51.6 (3.7×)
parallel	14	12.5 (2.6×)	23.5 (1.7×)	37.7 (2.7×)	53.8 (3.8×)
with multi-rate clocks					
sequential	1	12.8 (2.6×)	48.1 (3.4×)	47.6 (3.4×)	47.8 (3.4×)
semi-parallel	2	14.8 (3.0×)	55.1 (3.9×)	55.8 (4.0×)	56.2 (4.0×)
semi-parallel	4	17.9 (3.6×)	63.3 (4.5×)	72.4 (5.1×)	72.8 (5.2×)
semi-parallel	7	22.5 (4.6×)	64.0 (4.5×)	78.3 (5.6×)	85.4 (6.1×)
parallel	14	25.2 (5.1×)	69.9 (5.0×)	87.8 (6.2×)	103.7 (7.4×)

Table 1. Throughput measurements in frames per second for Par-DenseNet models.

Figure 1 represents the usage of each of the GPUs when running a sequential, a semi-parallel, and a fully-parallel Par-Inception model. The semi-parallel and fully-parallel models run 2.6 times faster than the sequential one (the time axis has been rescaled accordingly). Each inception block uses a different color.

We forced the sequential model to use all 8 GPUs but as expected each inception block only gets executed after the previous one in this case. It is also worth noting that the 4 branches of a given inception block are not executed in parallel, although they could be – we tried this and did not see any noticeable

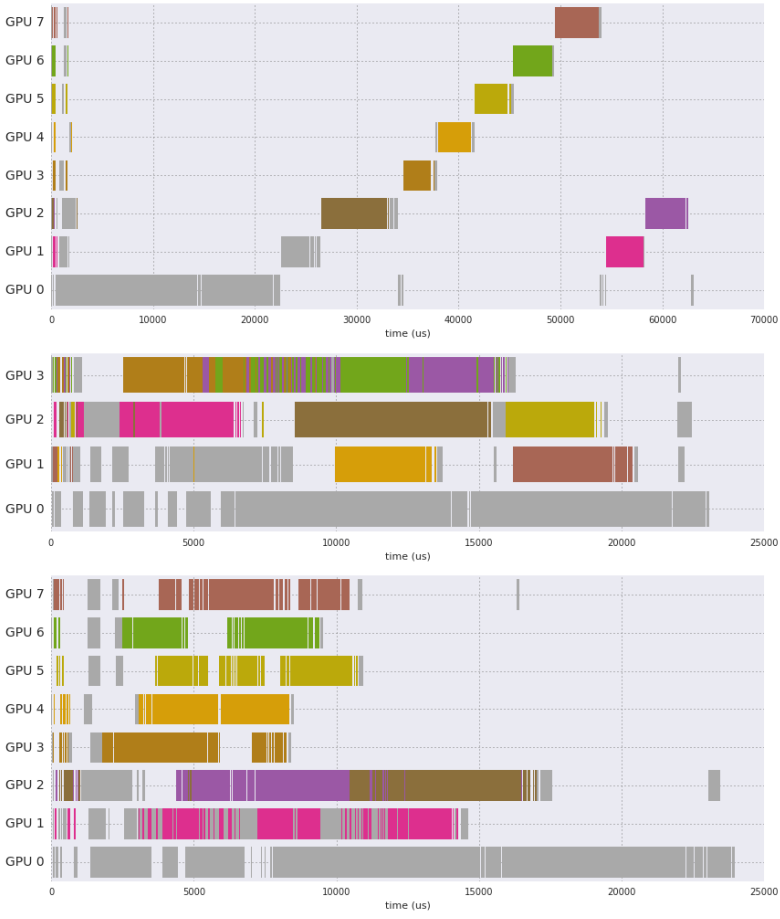

Fig. 1. Timeline for GPU usage for a sequential Par-Inception model on 8 GPUs at the top, a semi-parallel Par-Inception model using 4 GPUs in the middle, and a fully parallel model on 8 GPUs at the bottom. Each inception block is represented with a different color. Operations outside of inception blocks are colored in grey. Note that the timescale is different in the first picture compared to the two following ones.

speedup as one of the branches is far slower than the other three. The inter-GPU communication overhead caused by using all 8 GPUs appears to be negligible: the frame rate we measured did not depend on the number of GPUs being used.

However when using the parallel model, all the inception blocks are able to run at the same time. The bottleneck when running with 8 GPUs is that the first three convolution layers represent roughly a third of the computation and in our model they are executed sequentially. The same bottleneck applies to the semi-parallel model, but the GPU usage is much more balanced in this case as each of the other GPUs have at least two inception blocks to compute. A simple

Model	# Par. Subnets	48 cores	2 GPUs	4 GPUs	8 GPUs
without multi-rate clocks					
sequential	1	6.0 (1.0×)	18.6 (1.0×)	18.0 (1.0×)	18.1 (1.0×)
semi-parallel	5	7.9 (1.3×)	33.8 (1.8×)	48.7 (2.7×)	49.2 (2.7×)
parallel	10	7.8 (1.3×)	33.2 (1.8×)	46.4 (2.6×)	48.1 (2.6×)
with multi-rate clocks					
sequential	1	14.3 (2.4×)	48.2 (2.6×)	47.1 (2.6×)	47.1 (2.6×)
semi-parallel	5	18.1 (3.0×)	63.9 (3.4×)	90.9 (5.0×)	90.3 (5.0×)
parallel	10	18.1 (3.0×)	63.7 (3.4×)	88.6 (4.9×)	90.7 (5.0×)

Table 2. Throughput measurements in frames per second for Par-Inception models.

workaround would be to run these three convolutional layers in parallel branches – we plan on doing so in future work.

In order to visualise the trade-off between efficiency improvements and performance degradation, figure 2 plots model performance with respect to efficiency for the Par-DenseNet and Par-Inception models with clocks. Round markers represent the accuracy on the action task, normalized by the accuracy obtained by the sequential model. Square markers represent the inverse of the loss on the pose estimation task, again normalized by the loss obtained by the sequential model.

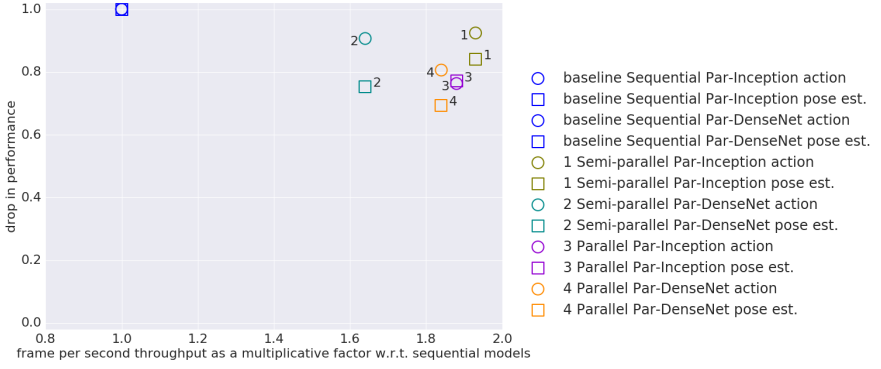

Fig. 2. Performance/efficiency trade-off introduced by depth-parallelising models with multi-rate clocks on 4 GPUs. Note that the baseline models use multi-rate clocks hence the smaller speedups compared to tables 1 and 2.

2 Pseudocode for predictive depth-parallelism

Using the toy example in the figure below, we illustrate the construction of the TensorFlow graph for the proposed predictive depth-parallel models with

multi-rate clocks in Algorithm 20. The model here has $n = 6$ layers and a final classifier, and it is unrolled over 5 time steps. The model outputs predictions y_i at the same rate as the rate at which frames I_i arrive. The layers are distributed into 2 parallel subnetworks, with $k = 3$ sequential layers in each subnetwork, and uses a clock rate of 1 for the first subnetwork and clock rate of 2 for the second one.

This has three implications: (1) when we break the sequence path between subnetworks, the output layer of subnetwork 1 should cache its activations for one time step, when they can be processed by the second subnetwork; (2) but because the second subnetwork ticks only every two time steps, the last layer of subnetwork 1 must actually cache its activations for two time steps, and (3) in every other time steps, the output classifier makes predictions based on stale inputs. The model is unrolled over time (line 5), similar to an RNN, and maintains its state, more precisely maintains two steps of computation as mentioned in observation (2) above.

At the first time step, the state is initialised to 0 (line 3). In every unroll step, the outputs of the network are first initialised from the state (line 6), and the current frame is appended to state (line 7) to be processed by the first layer. Then the computation traverses the network in depth (line 8). If the clock of a layer did not reach its tick time (line 9), then the layer is simply not connected in the graph for this time step (line 10), and its output will carry over a copy of the state to the next time step. If the clock does tick, we then need to check if the layer is to be connected in sequence (line 12) – inputs are taken from the last output of the previous layer as in standard models (line 15) – or in parallel – inputs are taken only from the state (line 13), using the two last entries in state, since the second subnetwork ticks slower. Eventually, the state is updated with the current outputs (line 18) and the loop is repeated for the remaining frames. The output predictions of the network are extracted from the last layer, by applying (in sequence) a classifier (line 19).

3 Additional details on training setups

We used randomly extracted subsequences of 32 frames for pose and 64 frames for action in training; the evaluation was done on the full sequences, that have up to 250 frames – 10 seconds of video. The spatial resolution of the input frames at both training and evaluation time is 224×224, obtained by random cropping at training time and central cropping for evaluation. We also randomly flipped the videos horizontally during training.

For the task of keypoint localisation, we generate dense per-frame labels by convolving the binary joint maps with a gaussian filter and obtain 17D heatmaps. Note that there can be multiple people in a single image and although this is a state-of-the-art system the labels are slightly noisy since they were automatically extracted and not verified by human experts. Also, for simplicity, we do not consider the problem of person detection, just direct keypoint localization. As a

```
input  : video frames {I}
input  : number of sequential layers k
output: predictions {y}
n ← len{layers}
clock_rates ← [1, 1, 1, 2, 2, 2]
state ← [0]
y ← [ ]
for t ← 0 to len{I} do
    outputs ← state.copy()
    state.append(I[t])
    for d ← 0 to n do
        if t  mod clock_rates[d]! = 0 then
            continue
        end
        if d  mod k == 0  then
            outputs[d].append(layer[d](state[d − 1][-2:]))
        else
            outputs[d].append(layer[d](outputs[d − 1][-1]))
        end
    end
    state ← outputs
    y.append(classifier(outputs[-1]))
end
```

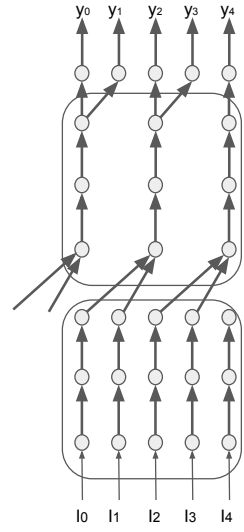

Fig. 3. Graph construction for predictive depth-parallel models, with multi-rate clocks.

consequence of these aspects of our setup, one predicted heatmap can contain several joints of the same type belonging to different individuals.

All our models were trained using SGD with momentum 0.9. For both tasks, the Par-Inception models were trained with initial learning rate of 0.1, and batch size of 4. For keypoint localisation, the learning rate was decreased by a factor of 10 after 35k and 55k iterations of training, whereas for action classification, it was decreased after 50k and 60k iterations. For both tasks, we ran 70k iterations of training.

The Par-DenseNet models were more memory intensive so we used a smaller batch size, 1 for keypoint localization, and 2 for classification. We trained the models with learning rate 1.0 for keypoints and 0.1 for actions, for a total of 150k iterations, lowering the learning rate by a factor of 10 at 100k iterations.

4 Architecture details

In this section, we explain how the proposed principles were applied on two popular image classification models: DenseNet [1] and Inception [2].

DenseNet model. DenseNet [1] is a state-of-the-art model for image classification. It consists of densely-connected blocks, each composed of b miniblocks. These blocks are densely connected such that every miniblock sends its activa-

tions to all the subsequent miniblocks in the same block. The model has growth-rate as a hyperparameter to control how many new features are added by each layer. Pooling layers are interleaved between the blocks. We used 4 blocks with average pooling operators in between, and growth-rate of 64. The blocks have 4, 8, 8, and 6 miniblocks (each with a 1x1 convolution followed by a 3x3 convolution). The model starts with a 7x7 convolutional layer and ends with a 3x3 heatmap prediction head for dense predictions tasks. The input to this head is a stack of skip-connections (upsampled feature maps, 56x56, from the end of each block plus the first convolutional layer). For classification tasks the head is replaced by a fully connected layer. We experimented with this model both with and without variable clock rates and temporal kernels (kernel dimension 2 along time for all layers but the first convolutional layer, where the model inputs a single image at a time). We also experimented with versions with and without feedback. When using feedback, the heatmaps predictions from the previous frame are stacked with the output of the first convolutional layer for the current frame. We trained the resulting models in all cases recurrently, restricting inputs to past frames so it behaves causally.

Inception model. The Inception architecture [2] is a directed graph that begins with 3 convolutional layers, followed by 9 inception blocks. Each inception block is composed of 4 parallel branches. For this model we experimented only with a version with temporal filters and variable clock rates, similar to the 3D ConvNet for action classification from the literature, I3D [3], but transformed into a causal recurrent-style network. Masked 3D convolutions can be used for this (or, equivalently, shifting along the time dimension, as mentioned in [4], sec. 2.1), but we prefer the unrolling since we are interested in frame-by-frame operation. The parameters between time steps are shared, the unrolling in this case being equivalent to shifting the convolutional kernel over the time dimension when applying a 3D convolution. The variable temporal strides of I3D are incorporated by removing blocks from the unrolling graph at time steps when there is a stride gap. Similar to DenseNet model, for per-frame prediction tasks, we introduce skip-connections (upsampling the activations of each inception block and passing them through a prediction head to produce spatial heatmaps).

Discussion.

In terms of model capacity, the two models are comparable, the temporal version of Inception has 12M parameters, and the temporal version of DenseNet has 10M parameters. The length of the longest sequential path for the Inception-based model is only 22 (counted as convolutional layers), whereas for DenseNet it is 54. Hence there are more possible options for breaking down this path into parallel subnetworks for DenseNet than for Inception. The information latency is however shorter for DenseNet because of its dense connectivity. The next section gives speedups for the two architectures, for different levels of parallelism.

We specify here the architectures trained for keypoint localisation or action, giving the layer structure (kernel shapes, number of channels, strides) and number of weights. ReLU and batch normalization layers are not shown, to reduce clutter, but are used as in the original image architectures.

4.1 DenseNet: table 3

4.2 Inception: tables 4 - 5

References

1. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, IEEE Computer Society (2017) 2261–2269
2. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, IEEE Computer Society (2015) 1–9
3. Carreira, J., Zisserman, A.: Quo vadis, action recognition? A new model and the kinetics dataset. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, IEEE Computer Society (2017) 4724–4733
4. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. CoRR **abs/1609.03499** (2016)

Layer name	Type	# channels	Kernel shape	Strides
Conv3d_1a_7x7	conv3d	64	1x7x7	1, 2, 2
MaxPool3d_sp	maxpool3d	-	1x3x3	1, 2, 2
MaxPool3d_t	maxpool3d	-	2x1x1	2, 1, 1
Block_1				
bottleneck_1_0	conv3d	256	1x1x1	1, 1, 1
conv_1_0	conv3d	64	1x3x3	1, 1, 1
bottleneck_1_1	conv3d	256	1x1x1	1, 1, 1
conv_1_1	conv3d	64	1x3x3	1, 1, 1
bottleneck_1_2	conv3d	256	1x1x1	1, 1, 1
conv_1_2	conv3d	64	1x3x3	1, 1, 1
bottleneck_1_3	conv3d	256	1x1x1	1, 1, 1
conv_1_3	conv3d	64	1x3x3	1, 1, 1
skip_1	conv3d	16	1x1x1	1, 1, 1
bottleneck_1_4	conv3d	168	1x1x1	1, 1, 1
AvgPool_1	conv3d	-	1x2x2	1, 2, 2
output = skip_1, AvgPool_1				
Block_2				
bottleneck_2_0	conv3d	256	1x1x1	1, 1, 1
conv_2_0	conv3d	64	1x3x3	1, 1, 1
...	...	...	...	...
conv_2_7	conv3d	64	1x3x3	1, 1, 1
bottleneck_2_7	conv3d	256	1x1x1	1, 1, 1
skip_2	conv3d	16	1x1x1	1, 1, 1
bottleneck_2_8	conv3d	340	1x1x1	1, 1, 1
AvgPool_2	conv3d	-	2x2x2	2, 2, 2
output = skip_2, AvgPool_2				
Block_3				
bottleneck_3_0	conv3d	256	1x1x1	1, 1, 1
conv_3_0	conv3d	64	1x3x3	1, 1, 1
...	...	...	...	...
conv_3_7	conv3d	64	1x3x3	1, 1, 1
bottleneck_2_7	conv3d	256	1x1x1	1, 1, 1
skip_3	conv3d	16	1x1x1	1, 1, 1
bottleneck_3_8	conv3d	426	1x1x1	1, 1, 1
AvgPool_3	conv3d	-	2x2x2	2, 2, 2
output = skip_3, AvgPool_3				
Block_4				
bottleneck_4_0	conv3d	256	1x1x1	1, 1, 1
conv_4_0	conv3d	64	1x3x3	1, 1, 1
...	...	...	...	...
conv_4_5	conv3d	64	1x3x3	1, 1, 1
skip_4	conv3d	16	1x1x1	1, 1, 1
output = skip_4				
Upsample and concat(MaxPool3d_sp, skip_1...4)				
Logits	conv3d	n_keypoints	1x3x3	1, 1, 1
Total number of weights: 10,843,464				

Table 3. Parameters of Par-DenseNet models for human keypoint localization. This version has multi-rate clocks and temporal filters but no feedback – in the version with feedback the input to "Logits" is fed back and concatenated with "MaxPool3d_sp". The classification version does not use the "skip" layers and instead has a classification head with inputs from just block 4.

Layer name	Type	# channels	Kernel shape	Strides
Conv3d_1a_7x7	conv3d	64	7x7x7	2, 2, 2
MaxPool3d_2a_3x3	maxpool3d	-	1x3x3	1, 2, 2
Conv3d_2b_1x1	conv3d	64	1x1x1	1, 1, 1
Conv3d_2c_3x3	conv3d	192	3x3x3	1, 1, 1
MaxPool3d_3a_3x3	maxpool3d	-	1x3x3	1, 2, 2
Mixed_3b				
branch0	conv3d	64	1x1x1	1, 1, 1
branch1_0	conv3d	96	1x1x1	1, 1, 1
branch1_1	conv3d	128	3x3x3	1, 1, 1
branch2_0	conv3d	16	1x1x1	1, 1, 1
branch2_1	conv3d	32	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	32	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
Mixed_3c				
branch0	conv3d	128	1x1x1	1, 1, 1
branch1_0	conv3d	128	1x1x1	1, 1, 1
branch1_1	conv3d	192	3x3x3	1, 1, 1
branch2_0	conv3d	32	1x1x1	1, 1, 1
branch2_1	conv3d	96	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	64	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
MaxPool3d_4a_3x3	maxpool3d	-	3x3x3	2, 2, 2
Mixed_4b				
branch0	conv3d	192	1x1x1	1, 1, 1
branch1_0	conv3d	96	1x1x1	1, 1, 1
branch1_1	conv3d	208	3x3x3	1, 1, 1
branch2_0	conv3d	16	1x1x1	1, 1, 1
branch2_1	conv3d	48	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	64	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
Mixed_4c				
branch0	conv3d	160	1x1x1	1, 1, 1
branch1_0	conv3d	112	1x1x1	1, 1, 1
branch1_1	conv3d	224	3x3x3	1, 1, 1
branch2_0	conv3d	24	1x1x1	1, 1, 1
branch2_1	conv3d	64	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	64	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
Mixed_4d				
branch0	conv3d	128	1x1x1	1, 1, 1
branch1_0	conv3d	128	1x1x1	1, 1, 1
branch1_1	conv3d	256	3x3x3	1, 1, 1
branch2_0	conv3d	24	1x1x1	1, 1, 1
branch2_1	conv3d	64	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	64	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
cont. on next page				

Table 4. Parameters of Par-Inception models.

Layer name	Type	# channels	Kernel shape	Strides
Mixed_4e				
branch0	conv3d	128	1x1x1	1, 1, 1
branch1_0	conv3d	144	1x1x1	1, 1, 1
branch1_1	conv3d	288	3x3x3	1, 1, 1
branch2_0	conv3d	32	1x1x1	1, 1, 1
branch2_1	conv3d	64	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	64	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
Mixed_4f				
branch0	conv3d	256	1x1x1	1, 1, 1
branch1_0	conv3d	160	1x1x1	1, 1, 1
branch1_1	conv3d	320	3x3x3	1, 1, 1
branch2_0	conv3d	32	1x1x1	1, 1, 1
branch2_1	conv3d	128	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	128	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
MaxPool3d_5a_3x3	maxpool3d	-	2x2x2	2, 2, 2
Mixed_5b				
branch0	conv3d	256	1x1x1	1, 1, 1
branch1_0	conv3d	160	1x1x1	1, 1, 1
branch1_1	conv3d	320	3x3x3	1, 1, 1
branch2_0	conv3d	32	1x1x1	1, 1, 1
branch2_1	conv3d	128	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	128	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
Mixed_5c				
branch0	conv3d	384	1x1x1	1, 1, 1
branch1_0	conv3d	192	1x1x1	1, 1, 1
branch1_1	conv3d	384	3x3x3	1, 1, 1
branch2_0	conv3d	48	1x1x1	1, 1, 1
branch2_1	conv3d	128	3x3x3	1, 1, 1
branch3_0	maxpool3d	-	3x3x3	1, 1, 1
branch3_1	conv3d	128	1x1x1	1, 1, 1
output = concat(branch0, branch1, branch2, branch3)				
AvgPool3d	avgpool3d	-	2x7x7	1, 1, 1
Logits	conv3d	num_classes	1x1x1	1, 1, 1
Total number of weights: 12,501,056				

Table 5. (cont.) Parameters of Par-Inception models.