Gradient Forward-Propagation for Large-Scale Temporal Video Modelling Supplementary Material

Mateusz Malinowski* DeepMind
Dimitrios Vytiniotis DeepMind
Grzegorz Świrszcz DeepMind
Viorica Pătrăucean DeepMind
João Carreira DeepMind

In this supplementary material we provide additional details about the experimental setup used in the main paper, results about the stability of the proposed *Skip-Sideways* training, and additional qualitative results obtained in the reconstruction task.

1. Notation

We denote input temporal sequences by $\boldsymbol{x} = (\boldsymbol{x}^{[t]})_{t=1}^{K}, \boldsymbol{x}^{[t]} \in \mathbb{R}^d$ and outputs by $\boldsymbol{y} = (\boldsymbol{y}^{[t]})_{t=1}^{K}, \boldsymbol{y}^{[t]} \in \mathbb{R}^{d_y}$. We mainly focus on video clips, where each frame has dimension $d = h \times w \times 3$. In action recognition tasks, we have $\boldsymbol{y}^{[t]} = \boldsymbol{y}^{[1]}$. We use single-frame 2D CNNs that map inputs to logits $\mathcal{M}_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^{d_y}$. These networks are compositions of layers, written as

$$\mathcal{M}_\theta(\boldsymbol{x}^{[t]}) = H_D(\cdot, \theta_D) \circ H_{D-1}(\cdot, \theta_{D-1}) \circ \ldots \circ H_1(\boldsymbol{x}^{[t]}, \theta_1)$$

where each layer $H_l(\cdot, \cdot)$ is a function $H_l : \mathbb{R}^{d_{l-1}} \times \mathbb{R}^{p_l} \rightarrow \mathbb{R}^{d_l}$, $\circ$ is a composition, *i.e.*, $G \circ F(\boldsymbol{x}^{[t]}) = G(F(\boldsymbol{x}^{[t]}))$, and $\theta = (\theta_l)_l, \theta_l \in \mathbb{R}^{p_l}$ are trainable parameters. We also use $\boldsymbol{h}_l^{[t]} = H_l(\cdot, \theta_l) \circ \ldots \circ H_1(\boldsymbol{x}^{[t]}, \theta_1)$ to denote activations at layer l.

We use $\mathcal{L} : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \rightarrow \mathbb{R}$ defined as $\mathcal{L}(\boldsymbol{h}_D, \boldsymbol{y}) = \sum_{t=1}^{K} l(\boldsymbol{h}_D^{[t]}, \boldsymbol{y}^{[t]}) = \sum_t l^{[t]}$, where $\boldsymbol{h}_D^{[t]}$ are logits, *i.e.*, $\boldsymbol{h}_D^{[t]} = \mathcal{M}_\theta(\boldsymbol{x}^{[t]})$, and l is a loss at time t, *e.g.*, a cross-entropy loss $l(\boldsymbol{h}, \boldsymbol{y}) = -\sum_i p(\boldsymbol{y}_i) \log q(\boldsymbol{h}_i)$. We update model parameters with the temporally averaged gradients, *i.e.*, $\theta := \theta - \alpha \sum_t \nabla_\theta l^{[t]}$, where α is a learning rate.

We use the following notation for Jacobian tensors. Let $\frac{\partial H(h,\theta)}{\partial h}(\boldsymbol{h}, \boldsymbol{\theta}) = \left.\frac{\partial H(h,\boldsymbol{\theta})}{\partial h}\right|_{h=\boldsymbol{h}}$ be the Jacobian tensor of $H(h, \theta)$ with respect to the variable h evaluated at $h = \boldsymbol{h}$, $\theta = \boldsymbol{\theta}$. Note that, we use bold fonts to distinguish between formal and actual values of the variables. We also use a short-hand notation $\frac{\partial h_l}{\partial h_{l-1}}(\boldsymbol{h}_{l-1}, \boldsymbol{\theta}_l) = \frac{\partial H_l(h_{l-1},\theta_l)}{\partial h_{l-1}}(\boldsymbol{h}_{l-1}, \boldsymbol{\theta}_l)$.

*The corresponding author: mateuszm@google.com

2. Experimental setup

Datasets. We use three datasets:

- HMDB51 has 6,770 video clips representing 51 actions [7], sampled at 30fps. We use the same train and test splits as [5, 9, 11].
- UCF101 [13] has 13,320 videos clips and 101 human actions, sampled at 25fps, with clips having 7.21sec on average.
- Kinetics600 (K600) is a large-scale dataset for action recognition [2] with around 490,000 videos. It has 600 human actions, and each action has at least 600 video clips per action. The clip duration is 10s, with frame-rate 25fps.

Tasks. Here we give additional details for the reconstruction task used in the main paper. At each time-step t, the network predicts the frame at time-step $t + 8$, *i.e.*, we want to learn a mapping $\mathcal{M}_\theta(\boldsymbol{x}^{[t]}; \theta) \rightarrow \boldsymbol{y}^{[t]} = \boldsymbol{x}^{[t+8]}$ for all t. We choose a displacement of 8 frames into the future as this corresponds to the number of *Skip-Sideways* or *Sideways* units that we use. That is, the output layer has to make a prediction for the frame that the first layer is about to see. Hence the models have no access to the observation they must predict. To successfully solve this task, it may be helpful for the models to learn to capture the underlying motion in the scene so they can re-distribute pixels accordingly. We minimise the following objective: $\frac{1}{T} \sum_{t=1}^{T} ||\mathcal{M}_\theta(\boldsymbol{x}^{[t]}) - \boldsymbol{x}^{[t+8]}||_{l_2}^2$. We compare *Skip-Sideways* against *Sideways*. The results of *Skip-Sideways* are significantly better than the equivalent model trained with *Sideways* (see additional qualitative results included in Figure 6) and it shows the benefit of integrating temporal information through direct and shortcut connections.

Distributed setting. We experiment on K600 using two multi-host setups, where we use either 8 or 16 hosts. Each host has 8 TPUs (Tensor Processing Units) amounting to 64

TPUs in the first setup, and 128 in the second setup. We use the second setup only for large batch-size experiments shown in Table 6, in the main paper. Unless we write otherwise, we use batch-size 16 per host. Hence, the total batch-size is 128 in the first setup and 256 in the second setup.

For VGG8, we put each layer of the network into a single *Skip-Sideways* unit and stick all units to different devices. For VGG16, we group two consecutive layers into a single *Skip-Sideways* unit and also stick all units to different devices. Note that VGG's design is convenient for the experiments as it has no shortcut connections and has roughly the same number of layers as the maximal number of devices (one per device for VGG8 and two for VGG16) that we use to distribute *Skip-Sideways* units. In the distributed setup, we use JAX [1] and Haiku [3].

GPipe. We have adapted GPipe [4], another realisation of a distributed training by pipelining. Here, we use the same breakdown of our models to different pipeline stages, *e.g.*, two consecutive VGG16 layers are allocated to a single pipeline stage and hence to a single device. Since the original GPipe has been designed to work with non-temporal data, we have extended it to work with videos by creating batches of frames as micro-batches, using the GPipe terminology. As GPipe implements correct BP, it also has two phases, forward and backward, with a parameters update that follows that on each device separately. Figure 1 illustrates comparison between GPipe and *Skip-Sideways* training in terms of device allocation.

In summary, in GPipe, (i) every micro-batch induces a forward and a backward computation on every device, (ii) activations on every device have to be kept alive until the backwards pass, hence increasing memory pressure, (iii) unless the number of micro-batches is sufficiently large there exists a non-negligible pipeline 'bubble' effect, and (iv) latency, and hence the overall performance, varies as the result of the moving 'bubble'. By contrast, (i) we only pay the cost of the backward computation (to simultaneously compute activations and gradients), (ii) each device does not have to hold previous activations as we exploit temporal correlation, (iii) we induce no bubble, (iv) latency is the same for each frame.

3. Models

3D VGG and Causal 3D VGG. We 'inflate' [2] 2D VGG to 3D VGG and train it from scratch, following the I3D model. That is, we extend spatial VGG16 [12] kernels to include temporal direction. We use $3\times3\times3$ kernel shapes for convolutions and max-pooling. Since *Skip-Sideways* training is real-time and hence causal, for a fair comparison, we have also implemented Causal 3D VGG. We have observed a slightly better performance with kernels $3 \times 3 \times 3$ over $2 \times 3 \times 3$ (from 63.7 to 64.2). For simiplicity, we illustrate the later variant in Figure 2a). We train all the models with ADAM [6] and a cosine learning rate schedule.

Full-Res network. As a proof-of-concept, we have implemented a network that maintains the input resolution through all the layers. We have designed the architecture to be (i) conceptually simple, (ii) not reduce spatial dimensions, and (iii) be better at utilising hardware resources in a distributed setting. Even though it better utilises the distributed hardware resources, its training is expensive with BP and using a single device. The network consists of a stack of convolutional kernels with 3×3 filters, and pooling layers with the same kernel shape (3×3). In all cases, we use striding one, so that there is no reduction in spatial dimensions. We use 64 channels in the first two consecutive blocks, and next 32 channels for rest of the network except from the last layer that maps back the latent space into 3 channels, which we interpret as RGB channels. We use Rectified Linear Units (ReLU) after each convolution, and hard hyperbolic tangent [10] in the last layer. Each two consecutive convolutional layers are followed by a single pooling layer. We use eight such blocks that we assign to different distributed devices within the *Skip-Sideways* units. We do not use a pooling layer after the last convolutional layer. We use batch-normalization just before ReLU. Figure 2b) illustrates the architecture showing all the blocks.

4. Speed and Memory

We show results on speed improvements and memory savings.

Space-to-depth. In all our experiments, we use space-to-depth transformation, which makes computations more TPU-friendly. More precisely, we first divide each frame into 2-by-2 non-overlapping patches and then we align them along the channel dimension. This transformation decreases each spatial dimension by two, and increases the channel dimension by four. Equivalently, it also increases the spatial receptive field of convolutional neural networks two times per spatial dimension and results in fewer but more costly convolutions (see Figure 4 in [14])[1]. Overall, however, space-to-depth transformation often speeds up training without affecting the accuracy of the model, but this is hardware and model specific. We also experiment with spacetime-to-depth transformation where we repeat the same procedure also along the time dimension. It further speeds up the computations as shown in Table 1. For our experiments with *Sideways* and *Skip-Sideways*, we keep a more pure setup with only space-to-depth transformation. This is because we want to explicitly model the temporal transitions through the *Sideways* or *Skip-Sideways* temporal connections.

Rematerialization. Rematerialization [8] is a method that saves memory at the cost of extra computation. That is,

[1] https://www.tensorflow.org/api_docs/python/tf/nn/space_to_depth

	$\frac{\text{steps}}{\text{sec}}$	Devices
Skip-Sideways + VGG16	2.3	1
Skip-Sideways + VGG16	3.6	2
Skip-Sideways + VGG16	**7.2**	8
GPipe [4] + VGG16	4.2	8
BP + 3D VGG16	1.9	1
BP + 3D VGG16 (Remat)	1.5	1
BP + Causal 3D VGG16	2.4	1
BP + I3D [2]	2.9	1
BP + 3D VGG16 (Batch)	5.5	8
BP + I3D (Batch)	6.5	8
BP + I3D (Batch)*	7.9	8

Table 1: Speed comparisons of various training strategies and architectures in number of steps per second. Column *devices* indicates the number of parallel devices used for distributed training. All models use K600, 32 frames, resolution 224 × 224, and batch-size 8. (Remat) denotes rematerialization [8]. The last two rows are the exceptions and use batch-size 1 per device. We use space-to-depth transformation. * denotes space-time-to-depth transformation.

the technique recomputes some intermediate activations instead of storing them in memory. Specifically, we use 'haiku.remat' where we rematerialize each vgg-block consisting of two or three 3D CNNs.

5. Theoretical savings

Sideways and *Skip-Sideways* do not propagate gradients back in time, only forward in time. This means that, in contrast to the traditional backpropagation, both mechanisms do not need to store activations over a long sequence. Therefore the memory cost is the same as the cost of doing backpropagation of per-frame models and is *independent of* the number of frames. Hence, as long as the memory savings are concerned, both mechanisms should scale up indefinitely with respect to the duration of the video. Moreover, in a distributed setting, if we do not count the costs of inter-device communication, the overall speed should be bounded only by the speed of the slowest device. That is, if the network is a composition of D modules $\mathcal{M}_\theta(\boldsymbol{x}^{[t]}) = H_D(\cdot, \theta_D) \circ H_{D-1}(\cdot, \theta_{D-1}) \circ \ldots \circ H_1(\boldsymbol{x}^{[t]}, \theta_1)$ and each module H_j sits on the j-th device, the run-time $\mathcal{T}$ of $\mathcal{M}_\theta$ is $\mathcal{T}(\mathcal{M}_\theta) = \max_j \mathcal{T}(H_j)$. Notice that with regular backprop the same runtime is the sum of runtimes of individual modules.

6. Stability

Our method assumes smoothness of the input signal. In this section, we investigate two settings that break that assumption and see how those impact our training.

Frame-rate. We run experiments on HMDB51 to investigate the effect of hyperparameters (learning rate and input frame rate) on training stability for the proposed *Skip-Sideways* compared to the original *Sideways* [9]. Intuitively, the higher the learning rate or the lower the frame-rate, the more we depart from our initial assumptions about the smoothness of the input or intermediate activations across time steps, hence the less stable the training should be. For frame-rate, we consider the values 30fps, 15fps, 7fps, 4fps. For learning rates, we use the values $1.0, 0.1, 0.01$.

Figure 3 shows the behaviour when using different initial learning rates and Figure 4 shows the behaviour when varying the input frame-rate.

In both figures, it can be observed that the proposed *Skip-Sideways* (on the right) has a more stable training compared to *Sideways*. In particular, our *Skip-Sideways* is much more robust w.r.t. frame-rate. We hypothesise that through shortcut and direct connections, the network can build smoother representation, for instance, by interpolating between frames in the hidden space.

Montage shots. We run experiments on HMDB51, where we create training sequences by assembling a new video from two or more different clips, which we call *montage shots*. Here, we investigate two scenarios. In the first scenario, we assemble 32 frames videos by concatenating 16 frames from one video and 16 frames from another video sampled from the same batch. Labels follow the same procedure. That is, the first 16 labels come from the first video, and the next 16 labels come from the second video. We also experiment with the concatenation of 3 video clips, each with 16 frames. Figure 5 shows the results. We can see a certain degree of robustness to cuts in a video clip. The more cuts, the worse the performance of the method.

In the second scenario, we interleave frames from all videos. That is, we use the 1st frame from the video 1, then the 1st frame from the video 2, then the 1st frame from the video 3, then the 2nd frame from the video 1, then the 2nd frame from the video 2, then the 2nd frame from the video 3, and repeat that for a total of 48 frames, with 16 frames per clip. Note that the method, by design, is robust for interleaving 2 video clips as there is no interference between neurons operating on consecutive frames (e.g., $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$ in Figure 1, in the main paper). We did not manage to train the network with *Skip-Sideways* in that setting.

In both experiments, we use per-frame losses with per-frame labels. All the results above indicate that our *Skip-Sideways* training is robust if the input is reasonably smooth.

7. Qualitative results

We provide further qualitative results for the future frame prediction task. Here, we use higher frame resolution 224 × 224. Note that the same resolution is maintained over all the layers of our neural networks, from the input input towards the output. We compare *Sideways* with *Skip-Sideways* to show the difference between both when the information is aggregated temporally. As noted in the main paper, using Full-Res models is expensive for BP. This shows that we can possibly use computationally or memory-wise more expensive models in video modelling. We show the results in Figure 6.

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[2] João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[4] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[5] Longlong Jing and Yingli Tian. Self-supervised spatiotemporal feature learning by video geometric transformations. *arXiv preprint arXiv:1811.11387*, 2018.

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 12 2014.

[7] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011.

[8] Ravi Kumar, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua Wang. Efficient rematerialization for deep networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[9] Mateusz Malinowski, Grzegorz Swirszcz, Joao Carreira, and Viorica Patraucean. Sideways: Depth-parallel training of video models. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[10] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[11] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[13] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[14] Longguang Wang, Yulan Guo, Zaiping Lin, Xinpu Deng, and Wei An. Learning for video super-resolution through hr optical flow estimation. In *Asian Conference on Computer Vision (ACCV)*, 2018.

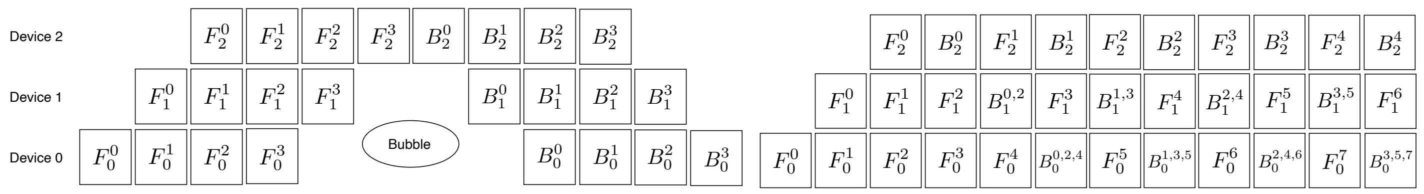
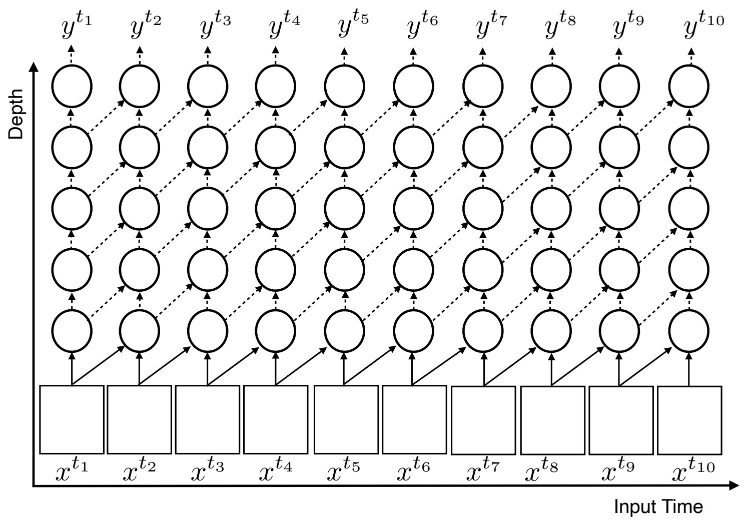
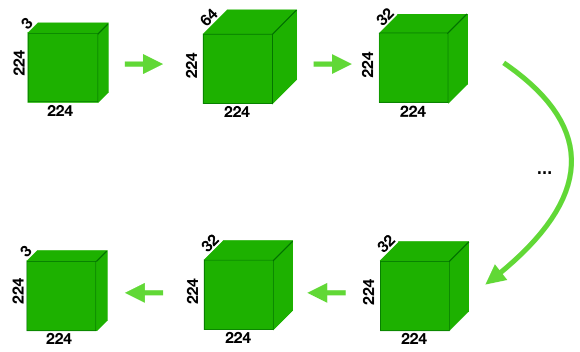


Figure 1: We show our adaptation of GPipe [4] to video modelling (left) and compare it with *Sideways* [9] and our *Skip-Sideways* (right). We only show the device placement during the forward pass (F) and backward pass (B). Because of that, we ignore shortcut connections in *Skip-Sideways* in this illustration. Here, we consider three distributed devices. F_k^t denotes that activations computed in the forward pass from the t-th input frame are placed on the k-th device. We use similar notation for gradients during the backward pass, *i.e.*, B_k^l. For *Sideways* and *Skip-Sideways*, we use $B_k^{l_1,l_2}$ to denote that the gradient was obtained from the incoming gradient computed based on the l_1-th input frame and Jacobian based on the l_2-th input frame. We extend this notation to $B_k^{l_1,l_2,l_3}$ that denotes the gradient $B_{k+1}^{l_1,l_2}$ is combined with Jacobian computed based on the l_3-th input frame on the k-th device. For the sake of clarity, we only show four input frames for GPipe. However, we can see that in the same number of twelve computation steps, *Skip-Sideways* can process more input frames and does not induce the 'bubble'. The vertical axis denote different devices and horizontal axis computation time.



a) Causal 3D VGG.



b) Full-Res.

Figure 2: Our two baseline architectures. Causal 3D VGG and Full-Res. Note that, contrary to the Figure 1 in the main paper, here we only show the connectivity of the architecture, and not the flow of pseudo-gradients nor activations. Moreover, the time in the figure refers to input data frames – not the computation time.

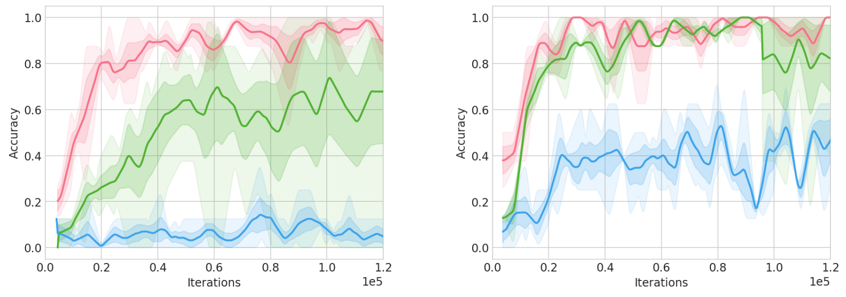


Figure 3: Robustness on the training fold with respect to the choice of the initial learning rate for *Sideways* (left) and our *Skip-Sideways* (right). Red, green and blue denote the initial learning rate as $0.01, 0.1, 1.0$. Solid curves depict the mean and shaded areas show variance across various input frame-rates.

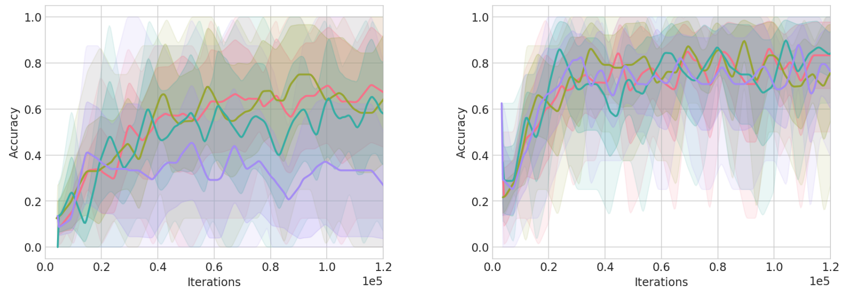


Figure 4: Robustness on the training fold with respect to the choice of the input frame-rate for *Sideways* (left) and our *Skip-Sideways* (right). Red, olive, green and violet denote the input frame-rate as $30, 15, 7, 4$ respectively. Solid curves depict the mean and shaded areas show variance across various input initial learning rates.

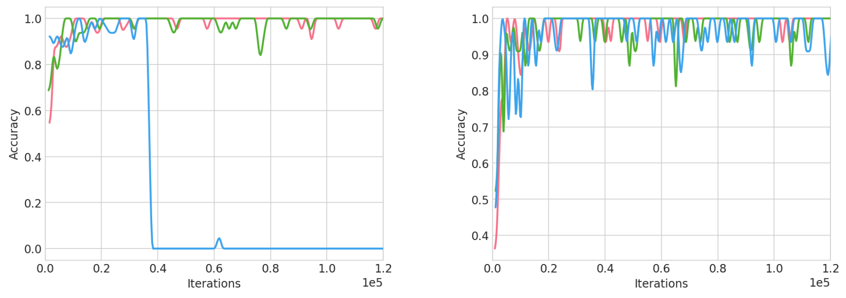


Figure 5: Robustness of *Skip-Sideways* on the montage experiments where we concatenate two (left) or three (right) video clips into a larger video. Red, green and blue denote the initial learning rate as $0.0001, 0.001, 0.01$. We can see that in general more video cuts hurt the performance. Sometimes the learning collapses for larger learning rates.

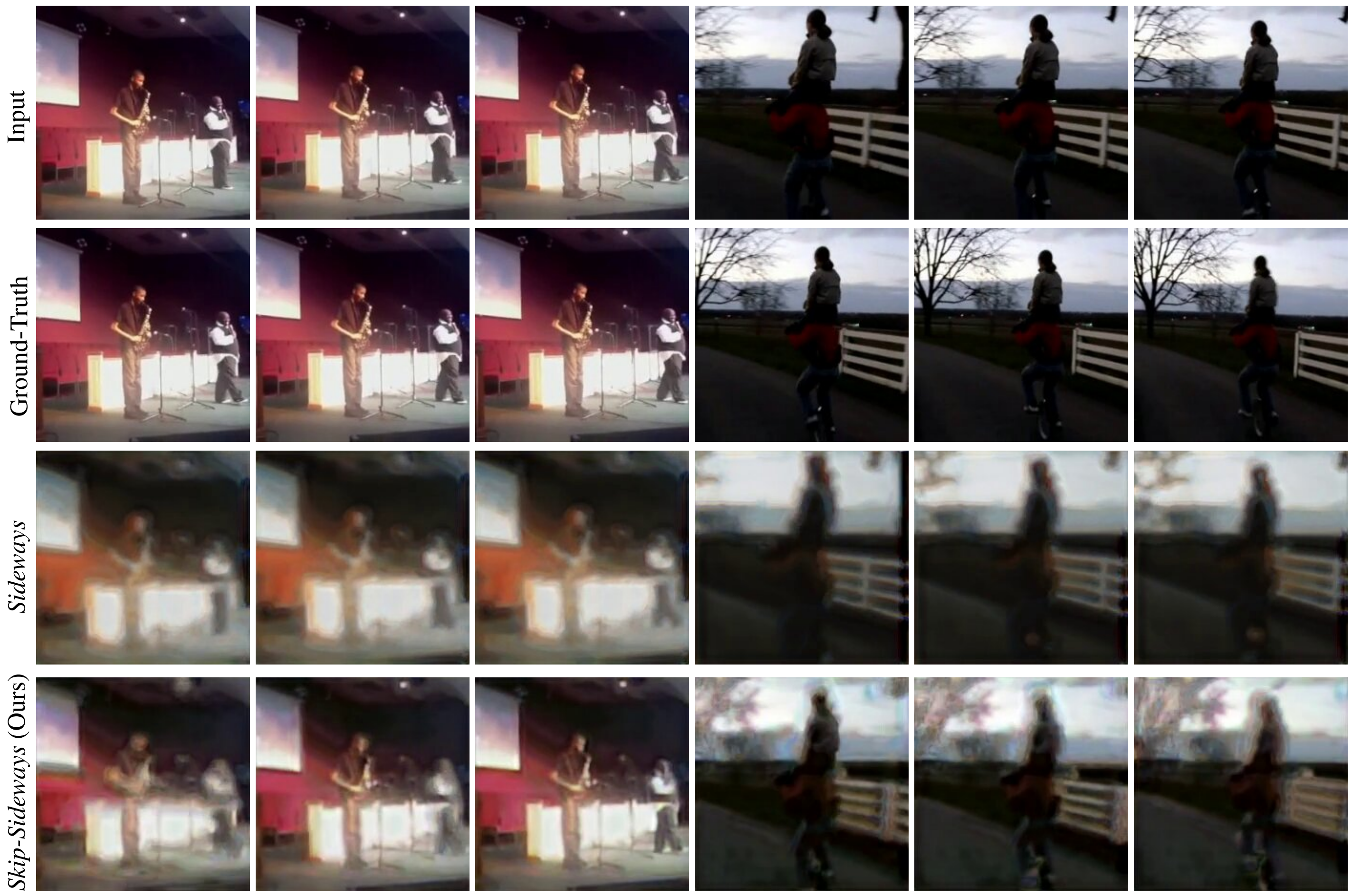


Figure 6: Qualitative results for two video sequences from the validation fold of Kinetics600, where the models need to generate the frame at step $t + 8$ into the future given frames up to t. From top to bottom: input frames, ground-truth output frames, predictions of Full-Res trained with *Sideways*, predictions of Full-Res trained with *Skip-Sideways*. We can observe that the predictions of the model trained with *Sideways* are blurry and semantically similar to inputs, indicating that the model lags behind. On the contrary, the predictions of the model trained with *Skip-Sideways* are sharper and semantically closer to the ground-truth output frames. We use resolution 224×224.