Sideways: Depth-Parallel Training of Video Models

Mateusz Malinowski mateuszm@google.com
Grzegorz Świrszcz swirszcz@google.com
João Carreira joaoluis@google.com
Viorica Pătrăucean viorica@google.com

DeepMind
London, U.K.

Abstract

We propose Sideways, an approximate backpropagation scheme for training video models. In standard backpropagation, the gradients and activations at every computation step through the model are temporally synchronized. The forward activations need to be stored until the backward pass is executed, preventing inter-layer (depth) parallelization. However, can we leverage smooth, redundant input streams such as videos to develop a more efficient training scheme? Here, we explore an alternative to backpropagation; we overwrite network activations whenever new ones, i.e.*, from new frames, become available. Such a more gradual accumulation of information from both passes breaks the precise correspondence between gradients and activations, leading to theoretically more noisy weight updates. Counter-intuitively, we show that Sideways training of deep convolutional video networks not only still converges, but can also potentially exhibit better generalization compared to standard synchronized backpropagation.*

1. Introduction

The key ingredient of deep learning is stochastic gradient descent (SGD) [7, 42, 53], which has many variants, including SGD with Momentum [47], Adam [26], and Adagrad [14]. E.g., SGD approximates gradients using mini-batches sampled from full datasets. Efficiency considerations primarily motivated the development of SGD as many datasets do not fit in memory. Moreover, computing full gradients over them would take a long time, compared to mini-batches, *i.e*., performing SGD steps is often more preferred [7, 16, 53]. However, SGD is not only more efficient but also produces better models. E.g, giant-sized models trained using SGD are naturally regularized and may generalize better [18, 43], and local minima do not seem to be a problem [11]. Explaining these phenomena is still an open theoretical problem, but it is clear that SGD is doing more than merely optimizing a given loss function [52].

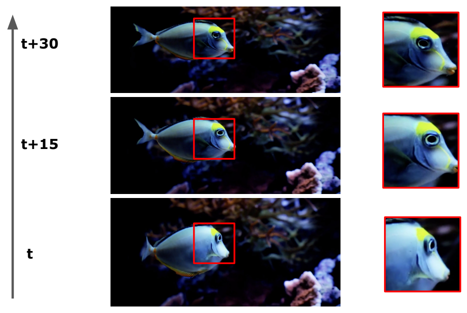

Figure 1: Three frames of a fish swimming, sampled 15 frames apart, or about every half a second. Note how little variation there is in the patch within the red square. Can we leverage such redundancies and the smoothness in local neighborhoods of such type of data for more efficient training? Our results suggest we can and there could be generalization benefits in doing that.

In this paper, we propose a further departure from the gradient descent, also motivated by efficiency considerations, which trains models that operate on sequences of video frames. Gradients of neural networks are computed using the backpropagation (BP) algorithm. However, BP operates in a *synchronized* blocking fashion: first, activations for a mini-batch are computed and stored during the forward pass, and next, these activations are re-used to compute Jacobian matrices in the backward pass. Such blocking means that the two passes must be done sequentially, which leads to high latency, low throughput. This is particularly sub-optimal if there are parallel processing resources available, and is particularly prominent if we cannot parallelize across batch or temporal dimensions, e.g., in online learning or with causal models.

The central hypothesis studied in this paper is whether we can backpropagate gradients based on activations from different timesteps, hence removing the locking between the layers. Intuitively, one reason this may work is that high frame rate videos are temporally smooth, leading to similar representations of neighboring frames, which is illustrated in Figure 1.

We experiment with two types of tasks that have different requirements in terms of latency: a per-sequence action recognition, and a per-frame autoencoding. In both cases, our models do not use any per-frame blocking during the forward or backward passes. We call the resulting gradient update procedure *Sideways*, owing to the shape of the data flow, shown in Figure 2.

In experiments on action recognition, UCF101 [46] and HMDB51 [29], we have found that training with *Sideways* not only does not diverge but often has led to improved performance over *BP* models, providing a surprising regularization effect. Such training dynamics create a new line of inquiry into the true nature of the success of SGD, as it shows that it is also not critical to have precise alignment between activations and gradients. Additionally, we show that *Sideways* provides a nearly linear speedup in training with depth parallelism on multiple GPUs compared to a *BP* model using the same resources. We believe that this result also opens up possibilities for training models at higher frame rates in online settings, e.g., where parallelization across mini-batches is not an option.

We use per-frame autoencoding task to investigate the effect of the blocking mechanism of *BP* models in tasks where the input stream cannot be buffered or where we require immediate responses. This is particularly problematic for *BP* if the input stream is quickly evolving, *i.e.*, the input change rate is higher than the time required to process the per-step input. In this case, the blocking mechanism of *BP* will result in discarding the new inputs received while the model is being blocked processing the previous input. However, this is considerably less problematic in *Sideways* due to its lock-free mechanism. We run experiments on synthetically generated videos from the CATER dataset [15], where we observe that *Sideways* outperforms the *BP* baseline.

2. Related Work

Our work connects with different strands of research around backpropagation, parallelization and video modelling. We list here a few of the most relevant examples.

Alternatives to backpropagation. Prior work has shown that various modifications of the 'mathematically correct' backpropagation can actually lead to satisfactory training. For instance, some relaxations of backpropagation implemented with a fixed random matrix yield a surprisingly good performance on MNIST [31]. There is also a recent growing interest in building more biologically-plausible or model-parallel approaches to train networks. This includes Feedback Alignment [31], Direct Feedback Alignment [37], Target Propagation [5], Kickback [2], Online AM [10], Features Replay [21], Decoupled Features Replay [3], and Synthetic Gradients [23], where various decouplings between forward or backward pass are proposed. A good comparative overview of those frameworks is presented in [12]. Another recent innovative idea is to meta-learn local rules for gradient updates [34], or to use either self-supervised techniques [39] or local losses to perform gradient-isolated updates locally [32, 38]. Asynchronous distributed SGD approaches like Hogwild [41] also do not strictly fit into clean backprop as they allow multiple workers to partially overwrite each others weight updates, but provide some theoretical guarantees as long as these overwrites are sparse. However, most of these prior works are applied to visually simpler domains, some require buffering activations over many training steps, or investigate local communication only. In contrast, here, we take advantage of the smoothness of temporal data. Moreover, we investigate a global, top-down, and yet asynchronous communication between the layers of a neural network during its training without buffering activations over longer period and without auxiliary networks or losses. This view is consistent with some mathematical models of cortex [6, 28, 30, 48]. We also address forward and backward locking for temporal models. Finally, most of the works above can also potentially be used together with our *Sideways* training, which we leave as a possible future direction.

Large models. Parallelism has grown in importance due to the success of gigantic neural networks with billions of parameters [49], potentially having high-resolution inputs [40], that cannot fit into individual GPUs. Approaches such as GPipe [20] or DDG [22] show that efficient pipelining strategies can be used to decouple the forward and backward passes by buffering activations at different layers, which then enables the parallel execution of different layers of the network. Similarly, multiple modules of the network can be processed simultaneously on activations belonging to different mini-batches [22]. Such pipelining reduces the training time for image models but at the cost of increased memory footprint.

Efficient video processing. Conditional computation [4] or hard-attention approaches can increase efficiency [33, 35] when dealing with large data streams. These are, however, generic approaches that do not exploit the temporal smoothness of sequential data such as video clips [50]. For video, sampling key frames is shown to be a quite powerful mechanism when performing classification [27, 51], but may not be appropriate if a more detailed temporal representation of the input sequence is needed [15]. Recently, a deep decoupled video model [8] has been proposed that achieves high throughput and speed at inference time, while preserving the accuracy of sequential models. However, [8] uses regular backprop, and hence does not benefit from parallelization fully, *i.e.*, backprop still blocks the computations, and requires buffering activations during the forward pass.

In this paper, we build upon [8] that uses parallel inference, but go further and make both inference and learning depth-parallel. Note that, if we only consider inference, *Sideways* reduces to [8].

3. Sideways

In this section, we define the formulation of our problem and formalize both algorithms: *BP* and *Sideways*.

3.1. Notation and Definitions

We consider the following general setting:

- a finite input time-series $\boldsymbol{x} = (\boldsymbol{x}^t)_{t=1}^K, \boldsymbol{x}^t \in \mathbb{R}^d$, *e.g.*, a video clip with $d = \text{height} \times \text{width} \times 3$,
- a finite output time-series $\boldsymbol{y} = (\boldsymbol{y}^t)_{t=1}^K, \boldsymbol{y}^t \in \mathbb{R}^{d_y}$, e.g., an action label; in the action recognition task, in our work, we use the same label over the whole video clip, *i.e.*, $\boldsymbol{y}^t = \boldsymbol{y}^{t+1}$ for all t,
- a frame-based neural network $\mathcal{M}_\theta : \mathbb{R}^d \to \mathbb{R}^{d_y}$ that transforms the input signal $\boldsymbol{x}^t$ into logits $\boldsymbol{h}_D^t = \mathcal{M}_\theta(\boldsymbol{x}^t)$, and is defined by a composition of modules

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

  where:

  - each module, or layer, $H_i(\cdot, \cdot)$ is a function $H_i : \mathbb{R}^{d_{i-1}} \times \mathbb{R}^{p_i} \to \mathbb{R}^{d_i}, i = 1, \ldots D$,
  - $\theta_i \in \mathbb{R}^{p_i}, i = 1, \ldots, D$ are the (trainable) parameters, and we use θ for all the parameters,
  - $\circ$ is a composition, *i.e.*, $G \circ F(\boldsymbol{x}) = G(F(\boldsymbol{x}))$

  and
- a loss function $\mathcal{L} : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$, *e.g.*, $\mathcal{L}(\boldsymbol{h}, \boldsymbol{y}) = ||\boldsymbol{h} - \boldsymbol{y}||^2$, or $\mathcal{L}(\boldsymbol{h}, \boldsymbol{y}) = -\sum_i p((\boldsymbol{h})_i) \log q(\boldsymbol{y}_i)$.

We extend the notation above to $\boldsymbol{h}_i^t = H_i(\cdot, \theta_i) \circ H_{i-1}(\cdot, \theta_{i-1}) \circ \ldots \circ H_1(\boldsymbol{x}^t, \theta_1)$.

To avoid the common confusion coming from using the same letters to denote both the function formal arguments and actual values of the variables, we will use bold font for the latter, *e.g.*, x to denote a formal argument and $\boldsymbol{x}$ for its actual value. We also use the following notation for the derivatives of the functions H_i. Let $\mathcal{J}_h H(\boldsymbol{h}, \boldsymbol{\theta}) = \left.\frac{\partial H(h, \boldsymbol{\theta})}{\partial h}\right|_{h=\boldsymbol{h}}$ be the Jacobian matrix of $H(h, \theta)$ with respect to the variable h evaluated at $h = \boldsymbol{h}, \theta = \boldsymbol{\theta}$. Similarly, $\mathcal{J}_\theta H(\boldsymbol{h}, \boldsymbol{\theta}) = \left.\frac{\partial H(\boldsymbol{h}, \theta)}{\partial \theta}\right|_{\theta=\boldsymbol{\theta}}$ denote the Jacobian matrix of $H(h, \theta)$ with respect to the variable θ evaluated at $h = \boldsymbol{h}$, $\theta = \boldsymbol{\theta}$. We will use the same notation for the gradient ∇.

Finally, to train neural networks, we base our computations on the empirical risk minimization framework, *i.e.* $\mathcal{R}(\mathcal{M}_\theta) = E_{x,y}[\mathcal{L}(\mathcal{M}_\theta(x), y)] \approx \sum_{\boldsymbol{x},\boldsymbol{y} \sim \mathcal{D}} \frac{1}{K} \sum_{t=1}^K \mathcal{L}(\boldsymbol{h}_D^t, \boldsymbol{y}^t)$, where $\mathcal{D}$ is a training set.

3.2. Update Cycle

For simplicity, we assume in our modelling a constant time for a layer (or some set of layers organized into a module) to fully process its inputs, both in the forward or backward pass and call this a *computation step*. We define the *computation cycle* as the sequence of computation steps that a given data frame is used to update all the layers, and the *cycle length* as the number of computation steps in the computation cycle. Hence, the cycle length depends only on the depth of the network D and is equal to $2D - 1$ computation steps. Figure 2 illustrates a single computation cycle with nine computation steps for both models.

3.3. The *BP* algorithm ('regular' backpropagation)

The *BP* algorithm refers to regular training of neural networks. Here, due to the synchronization between the passes, computations are blocked each time a data frame is processed. This is illustrated in Figure 2 (left). Whenever the first frame is processed, here indicated by the blue square, the computations are blocked in both forward and backward passes over the whole computation cycle.

With our notation, the standard backpropagation formula becomes

$$\begin{aligned}
\nabla_{\theta_i}^t \mathcal{L} \quad = \quad & \nabla_{\theta_i} \mathcal{L}(\mathcal{M}_\theta(\boldsymbol{x}^t), \boldsymbol{y}^t)|_{\theta=\boldsymbol{\theta}} = \\
& \nabla_{h_D} \mathcal{L}(\boldsymbol{h}_D^t, \boldsymbol{y}^t) \cdot \mathcal{J}_{h_{D-1}} H_D(\boldsymbol{h}_{D-1}^t, \boldsymbol{\theta}_D) \cdot \\
& \mathcal{J}_{h_{D-2}} H_{D-1}(\boldsymbol{h}_{D-2}^t, \boldsymbol{\theta}_{D-1}) \cdot \\
& \vdots \\
& \mathcal{J}_{h_i} H_{i+1}(\boldsymbol{h}_i^t, \boldsymbol{\theta}_{i+1}) \cdot \\
& \mathcal{J}_{\theta_i} H_i(\boldsymbol{h}_{i-1}^t, \boldsymbol{\theta}_i)
\end{aligned}$$

with the update rule $\theta_i := \theta_i - \alpha \frac{1}{K} \sum_{t=1}^K \nabla_{\theta_i}^t \mathcal{L}$, where α is the learning rate, and K is the length of the input sequence.

We can compactly describe the algorithm above with the following recursive rules

$$\nabla_{\theta_i}^t \mathcal{L} = \nabla_{h_i}^t \mathcal{L} \cdot \mathcal{J}_{\theta_i} H_i(\boldsymbol{h}_{i-1}^t, \boldsymbol{\theta}_i) \quad (1)$$

$$\nabla_{h_{i-1}}^t \mathcal{L} = \nabla_{h_i}^t \mathcal{L} \cdot \mathcal{J}_{h_{i-1}} H_i(\boldsymbol{h}_{i-1}^t, \boldsymbol{\theta}_i) \quad (2)$$

where $\boldsymbol{h}_0^t = \boldsymbol{x}^t$. However, note that in standard implementations, Jacobian matrices are not computed explicitly; instead efficient vector matrix multiplications are used to backpropagate errors from the loss layer towards the input [1].

3.4. *Sideways* algorithm

We aim at pipelining computations for the whole computation cycle during training and inference. *Sideways* removes synchronization by continuously processing information, either in the forward or backward pass. This is illustrated in Figure 2 (right). Once a data frame is available, it is immediately processed and sent to the next layer, 'freeing' the current layer so it can process the next data frame. Hence, in the first computation step of the computation cycle, a data frame $\boldsymbol{x}^t$ is processed by the first *Sideways* module, freeing resources and 'sending' $\boldsymbol{h}_1^t$ to the second *Sideways* module at computation step $t+1$. At computation step $t+1$, the first module can now take the next data frame $\boldsymbol{x}^{t+1}$ for processing, and, simultaneously, the second module processes $\boldsymbol{h}_1^t$; this step results in two representations $\boldsymbol{h}_2^t$ and $\boldsymbol{h}_1^{t+1}$. Please note that our notation $\boldsymbol{h}_2^t$ does not indicate the current computation step but instead that the representation has originated at $\boldsymbol{x}^t$. We continue the same process further during the training. This is illustrated in Figure 2, where we use color-encoding to track where the information being processed has originated from. Dotted arrows represents the forward pass.

For simplicity, we assume that the computation of the loss takes no time and does not require an extra computation cycle. In such setting the activation arriving at the loss function computing module at timestep t is $\boldsymbol{h}_D^{t-D+1}$, an activation spawned by the frame $\boldsymbol{x}^{t-D+1}$. Once this final representation $\boldsymbol{h}_D^{t-D+1}$ is computed at computation step t, we calculate its 'correct' gradient $\nabla_{h_D}^t \mathcal{L}(\boldsymbol{h}_D^{t-D+1}, \boldsymbol{y}^t)$, and we backpropagate this information down towards the lower layers of the neural network. This computational process is illustrated in Figure 2 (right) by the solid arrows.

Let us formalize this algorithm in a similar manner to the 'regular' backpropagation. In the *Sideways* algorithm the gradient $\nabla_{\theta_i}\mathcal{L}|_{(\boldsymbol{x}^t,\boldsymbol{\theta}_i)}$ is replaced with a *pseudo-gradient* $\widetilde{\nabla}_{\theta_i}\mathcal{L}|_{(\boldsymbol{x}^t,\boldsymbol{\theta}_i)}$, defined as follows

$$\begin{aligned}\widetilde{\nabla}_{\theta_i}^t\mathcal{L} \;\; = \;\; & \nabla_{h_D}\mathcal{L}(\boldsymbol{h}_D^{t_i-D+1}, \boldsymbol{y}^{t_i}) \cdot \mathcal{J}_{h_{D-1}} H_D(\boldsymbol{h}_{D-1}^{t_i-D+1}, \boldsymbol{\theta}_D) \cdot \\ & \mathcal{J}_{h_{D-2}} H_{D-1}(\boldsymbol{h}_{D-2}^{t_i-D+3}, \boldsymbol{\theta}_{D-1}) \cdot \\ & \vdots \\ & \mathcal{J}_{h_i} H_{i+1}(\boldsymbol{h}_i^{t-i-1}, \boldsymbol{\theta}_{i+1}) \cdot \\ & \mathcal{J}_{\theta_i} H_i(\boldsymbol{h}_{i-1}^{t-i+1}, \boldsymbol{\theta}_i)\end{aligned}$$

where $t_i = t + i - D$.

The equations above can next be written succinctly and recursively as the *Sideways* backpropagation rules

$$\widetilde{\nabla}_{\theta_i}^t \mathcal{L} \;\; = \;\; \underline{\widetilde{\nabla}_{h_i}^{t-1}\mathcal{L}} \cdot \underline{\mathcal{J}_{\theta_i} H_i(\boldsymbol{h}_{i-1}^{t-i+1}, \boldsymbol{\theta}_i)} \qquad (3)$$

$$\widetilde{\nabla}_{h_{i-1}}^t \mathcal{L} \;\; = \;\; \underline{\widetilde{\nabla}_{h_i}^{t-1}\mathcal{L}} \cdot \underline{\mathcal{J}_{h_{i-1}} H_i(\boldsymbol{h}_{i-1}^{t-i+1}, \boldsymbol{\theta}_i)} \qquad (4)$$

where $\widetilde{\nabla}_{h_D}^{t-1}\mathcal{L} = \nabla_{h_D}\mathcal{L}(\boldsymbol{h}_D^{t-D+1}, \boldsymbol{y}^t)$, and $\boldsymbol{h}_0^t = \boldsymbol{x}^t$.

In the equations above, we use a color-encoding similar to Figure 2 (right) to indicate that we combine information originated from different time steps. For instance, information originated in 'blue' and 'yellow' input frames is combined (6-th computation step and second-last unit) as indicated by the red circle in Figure 2 (right)). By following the arrows we can track the origins of the combined information.

Due to the nature of these computations, we do not compute proper gradients as the *BP* algorithm does, but instead we compute their more noisy versions, $\widetilde{\nabla}_{\boldsymbol{h}_i}\mathcal{L} = \nabla_{\boldsymbol{h}_i}\mathcal{L} + \epsilon_i(\boldsymbol{x})$, which we call pseudo-gradients. The amount of noise varies with respect to the smoothness of the input $\boldsymbol{x}$, and the number of the layer i. That is, deeper layers have less noisy pseudo-gradients, and *e.g.*, the pseudo-gradient of the final layer is exact.

We organize training as a sequence of episodes. Each episode consists of one or more computation cycles, runs over the whole sampled video clip $\boldsymbol{x}$ or its subsequence, and ends with the weights update. We assume the input $\boldsymbol{x}$ is smooth within the episode, *e.g.*, $\boldsymbol{x}$ is a video of an action being performed with a reasonable frame-rate. We 'restart' *Sideways* by setting up all the activations and pseudo-gradients to zero whenever we sample a new video to avoid aliasing with a pseudo-gradient originated from a data frame from another video clip, and thus breaking our assumptions about the smoothness of the input sequence. Mini-batching can optionally be applied in the usual way.

We average gradients computed at each layer over all computation steps within the episode, *i.e.*,

$$\widetilde{\nabla}_{\theta_i}\mathcal{L} = \frac{1}{L}\sum_{t=1}^{L} \widetilde{\nabla}_{\theta_i}^t \mathcal{L} \qquad (5)$$

where L is the length of the episode. In our experiments we consider two cases. In the *classification* task, the episode is the same as the sampled sequence, *i.e.*, $L = K$. In the *auto-encoding* task, the episode is a single data frame, *i.e.*, $L = 1$. We use pseudo-gradients $\widetilde{\nabla}_{\theta_i}\mathcal{L}$ for the weight updates, *i.e.*, $\theta_i := \theta_i - \alpha\widetilde{\nabla}_{\theta_i}\mathcal{L}$.

Figure 3 (right) illustrates the situation when the pipeline is full and suggests, the information flow is tilted sideways. Therefore, there is no information available in the upper layers at the beginning of the sequence (empty circles in the figure). For that reason, we modify Equation 5 by including a binary mask, *i.e.*, $\widetilde{\nabla}_{\theta_i}\mathcal{L} = \frac{1}{\gamma_i}\sum_{t=1}^{L}\gamma_i^t\widetilde{\nabla}_{\theta_i}^t\mathcal{L}$, where $\gamma_i = \sum_t \gamma_i^t$. The mask is zero for unavailable gradients. For similar reasons, to avoid gradient computations whenever suitable information is unavailable, we modify Equation 4 with $\widetilde{\nabla}_{\boldsymbol{h}_i}^t\mathcal{L} = \gamma_i^t\widetilde{\nabla}_{h_i}^{t-1}\mathcal{L}\cdot\mathcal{J}_{h_{i-1}}H_i(\boldsymbol{h}_{i-1}^{t-i+1}, \boldsymbol{\theta}_i)$. Without masking, we have observed more unstable training in practice.

Intuitions. As we make the input sequence increasingly more smooth, in the limits, each data frame has identical content. In such a case, since $\epsilon_i(\boldsymbol{x}) = 0$, pseudo-gradients equal gradients, and our algorithm is the same as the 'regular' backpropagation. In practice, if the input sequence has different data frames, we assume that two consecutive frames are similar, and especially essential features are slowly evolving, sharing their semantics within the neighborhood [50].

4. Experiments

We investigate both algorithms – *BP* and *Sideways* – on several benchmarks. Since, to the best of our knowledge, this is possibly the first work on depth-parallel training on challenging video tasks, we focus on simple convolutional networks, and aim to explore the training dynamics instead of seeking state-of-the-art results. We leave data augmentation, additional features such as optical flow, or pre-training on large datasets [9, 13, 24, 25, 44] for future work. We compare frame-based video models [24, 25, 44] that are *trained from scratch* and using standard setups.

4.1. Tasks and Datasets

We benchmark our algorithms on two different tasks and three datasets.

Classification. We start with the classical classification task, here, in the form of action recognition. Since the classification is at the core of regular supervised learning, we believe, any alternative, sequential or parallel, to SGD should be evaluated on this common task. Figure 2 illustrates both algorithms under the classification scenario. Differently to the next, auto-encoding task, here, we test the networks under the regular circumstances, where each frame is always guaranteed to be processed by the neural network.

Auto-encoding. While majority of our key results are on the classification task, it is also informative to validate *Sideways* on tasks where the target output is continuously changing with the input. As a proof of concept, we experiment here with the simpler task of auto-encoding. To clearly illustrate advantages of *Sideways* training, and for the sake of simplicity, we assume that the input frame rate and the processing time for each individual neural network layer are equal. This is shown in Figure 3. If the stream is a sequence $(x^{t_1}, x^{t_2}, x^{t_3}, \ldots)$, D is the number of modules, then *BP* blocks the processing of the input for $2(D-1)$ computation steps, hence ignoring data frames between t_1 and t_{10} during training for $D=5$. This is illustrated in Figure 3 (left). In contrast, *Sideways* pipelines computations and uses all the data frames in both training and inference modes. This often results in superior performance of *Sideways* under the circumstances mentioned above. Finally, by comparing Figures 2 and 3, we can clearly see the *Sideways* algorithm behaves identically, even if we artificially introduce the blocking mechanism described above.

Datasets. We choose to benchmark our models of computations on the following video datasets. We include experiments on two popular action recognition datasets: HMDB51 [29] and UCF-101 [46]. Both datasets consist of short video clips. On one hand they have enough complexity and realism. On the other hand, we can easily train frame-based models on all these datasets. In addition, we also experiment on a synthetic CATER dataset [15] of moving simple 3D objects. Here, we use only the video frames and we set up an unsupervised auto-encoding task. These videos have two desired properties – i) they are visually simple, and ii) they have diverse motion patterns of various objects – making it an excellent benchmark for *Sideways*. We provide more details in the supplementary material.

4.2. Architectures

In our study, we experiment with two standard convolutional network architectures. The first one is organized into 6 *Sideways* modules, another one with 8 *Sideways* modules. Note, however, that we can use more than one trainable layers inside a single *Sideways* module.

Simple-CNN is a simple and fast baseline consisting of 5 convolutional layers with kernel size $3x3$ followed by global average pooling and a softmax on the linear layer that projects the internal representation into classes. The convolutional layers have the following number of channels: $(32, 64, 64, 128, 256)$. To reduce resolution progressively in the network, we use striding 2 in every second layer starting from the first one.

For the auto-encoding experiments, we train a simple encoder-decoder architecture having the same five convolutional blocks followed by symmetrical five deconvolutional blocks. We use *Sideways* blocks only for the convolutional encoder; the decoder layers are connected all in a single sequential block, and hence the decoder-block is trained with a regular *BP* with 'correct' gradients. For simplicity, we also assume the whole decoder takes just a single computation step. We use this setting to better investigate the quality of the features extracted by the *Sideways* encoder.

VGG-net refers to VGG-8, which is a direct re-implementation of the RGB network in the original two-stream model [45] with the addition of batchnorm in every VGG-block (in between the convolution and the ReLU).

4.3. Results (Classification)

We evaluate networks trained with *Sideways* and *BP* with the regular accuracy metric. For better understanding, and to show how general the *Sideways* training is, we also conduct several experiments measuring not only accuracy but also training dynamics and robustness of the method.

Quantitative results. Table 1 directly compares both algorithms, backpropagation with the pipelined *Sideways* training. For the sake of comparison, we also report referenced models that are trained using 'regular' training. As we can see, we have reproduced similar results with the *BP* model, and in several cases, we have achieved higher ac-

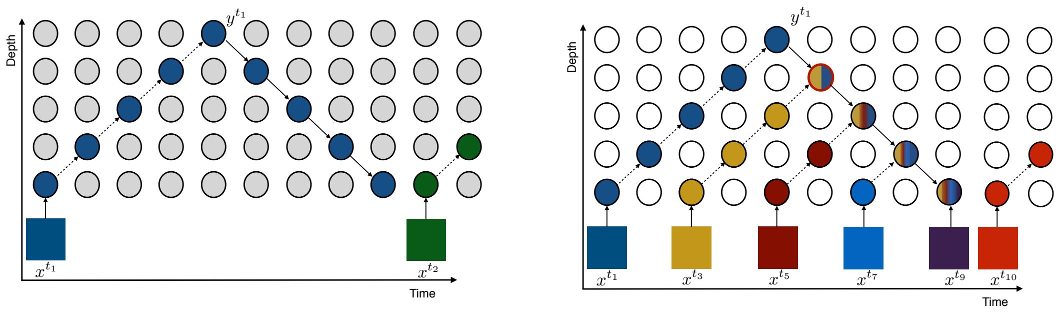

Figure 2: From left to right. Standard (*BP*) and fully pipelined (*Sideways*) approaches to temporal training and inference. We show a single computation cycle, and the beginning of the next cycle. Both architectures are unrolled in time. Colorful squares indicate data frames. Circles indicate 'regular' or *Sideways* modules. Dotted arrows show how information is passed between layers and time steps in forward pass. Solid arrows show the same in backward pass. In *Sideways* (right), we only exemplify a single update path with the arrows, and use empty circles for all other units. Gray circles denote blocked modules, *i.e.*, units waiting for forward or backward pass. Note that for *BP*, we use the same color for all the units on the data path, in both the forward and the backward passes, to highlight that all the layers work on information originated in a single data frame, the blue one. Differently, the back-pass in *Sideways* shows circles with many colors to illustrate that information from different data frames is combined in one update cycle. For instance, combining 'blue gradient' with 'yellow activations' yields 'blue-yellow gradient' (6th computation step and second-last unit). Best viewed in color.

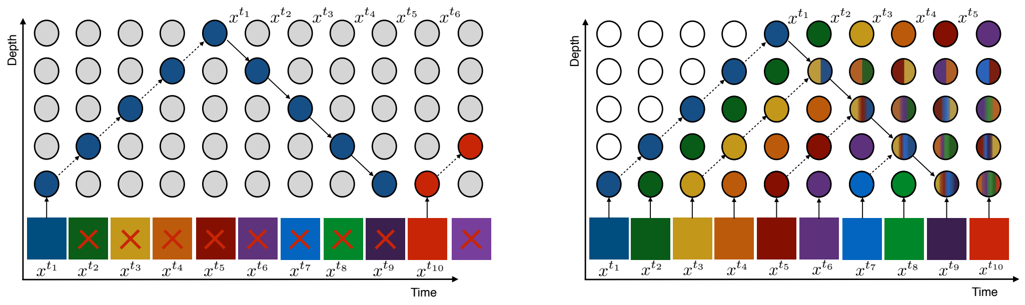

Figure 3: From left to right. *BP* and *Sideways* approaches to temporal training and inference. In the figure, we illustrate the auto-encoding task, where the network needs to synthesize input frames. Crossed frames denote input data ignored because the system cannot operate in real-time to process all the inputs (left). In contrast, *Sideways* works simultaneously at full capacity once the pipeline is full; and since we show the beginning of the episode some units are unused (empty circles) due to the shape of the data flow (right). All the colors and arrows have the same meaning as in Figure 2. Best viewed in color.

curacy (e.g., VGG-8 + Dropout (0.9)). Even though higher accuracy numbers have been previously reported on both datasets, these are achieved using larger models pre-trained on larger datasets. Our focus is, however, different.

Results presented in Table 1 suggest that the *Sideways* training achieves competitive accuracy to *BP*, and the introduced noise due to i) the sampling error, the same as SGD updates, and ii) the pseudo-gradients computations, does not seem to harm the overall performance. Quite the opposite, under certain conditions, we observe *Sideways* generalizes better than *BP*. Since such behavior occurs during training larger models on relatively small video datasets, *e.g.*, training VGG-8 on UCF-101, which often results in overfitting [44], we hypothesize *Sideways* acts as an implicit regularizer for video processing [36, 52].

Training dynamics. Since we compare *Sideways* algorithm to *BP*, it is instructive to investigate their training behavior. Intuitively, similar training behavior should result in a similar final performance. Therefore, we have conducted experiments where we measure various statistics throughout training, and we report them in Figure 4. There are a few interesting observations. First, the training dynamics of the VGG-8 architecture with *Sideways* training closely follows 'regular' training (first two columns). How-

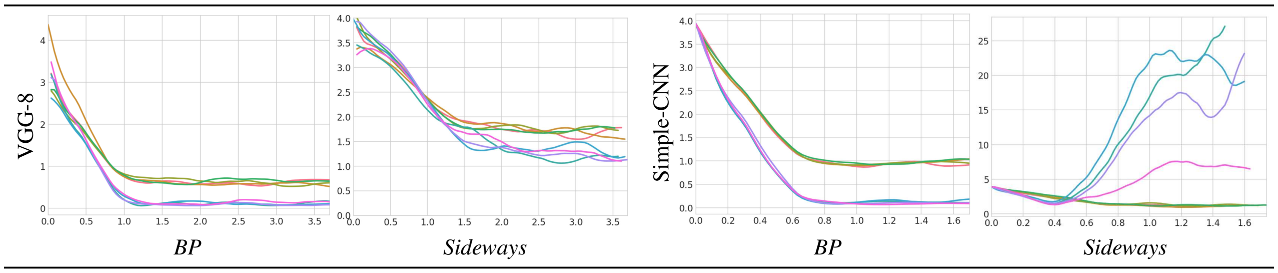

Figure 4: Training dynamics of Simple-CNN and VGG-8 with different models of computations. Experiments are conducted on the HMDB51 dataset. Different colors denote different hyper-parameters (red, green, olive, orange refer to the initial learning rate 10^{-5} and teal, pink, violet, blue to 10^{-4}, all with various weight decay). On the x-axis, we report number of iteration steps, in 10^5 scale. On the y-axis, we report loss values. Note that the figures have different y-limits to allow a detailed visualization of the training dynamics as training progresses.

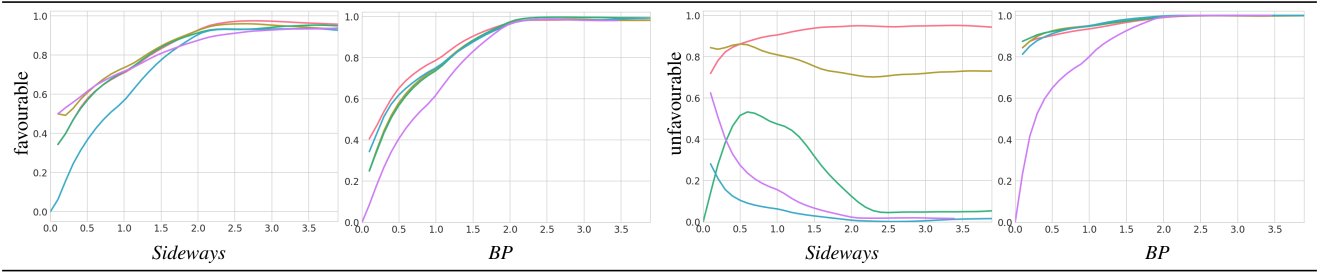

Figure 5: We experiment with different temporal striding settings ($\{2, 3, 4, 5, 6\}$ encoded as red, olive, green, blue, violet, respectively) for the input videos, on UCF101. First two columns show favourable hyper-parameters (initial learning rate equals to 10^{-5}). Last two columns show unfavourable hyper-parameters (initial learning rate equals to 10^{-4}). Each for *Sideways* and *BP*. On the x-axis, we report number of iteration steps. On the y-axis, we report accuracy numbers. In the setting with unfavourable hyper-parameters, training of networks collapses with higher striding numbers.

HMDB51	*BP*	*Sideways*
Simple-CNN	17.2	16.5
VGG-8	24.6	25.8
3DResNet (scratch) [17, 24]	17.0	-
UCF101	*BP*	*Sideways*
Simple-CNN	40.7	42.16
VGG-8	49.1	53.8
VGG-8 + Dropout (0.9)	56.0	58.2
VGG-8 (scratch)+ Dropout (0.9) [44]	52.3	-
3DResNet (scratch) [17, 24]	42.5	-

Table 1: Comparison of our implementation of two architectures using *Sideways* and *BP* training on different datasets. For reference, we also report similar models from prior work [17, 24, 44]. We report accuracy in %.

ever, for the Simple-CNN architecture, training dynamics between both algorithms differ under some choice of the hyper-parameters. For instance, we can notice in Figure 4 (last two columns) the loss function become quite unstable. This happens consistently with a larger learning rate, *e.g.*, above 10^{-4}. Even though this seemingly should also transfer into unstable training accuracy, we have found training does not collapse. Quite the opposite, we report a relatively high training accuracy (above 85%). After a more careful inspection, we observe that Simple-CNN trained with *Sideways* and larger learning rates tends to give confident predictions that result in high loss whenever they miss the class. Results on UCF-101 are similar, but slightly less pronounced. We provide more results on the training dynamics in the supplementary material.

Sensitivity to frame rate. The smoothness of the input space is the key underlying assumption behind the *Sideways* algorithm. When the input space is the space of video clips, this assumption translates, *e.g.*, into a high frame-rate. To further stretch this assumption, we have artificially decreased the frame-rate by skipping data frames in

	BP	*Sideways*	speedup
Simple CNN	1.7	8.4	4.9x
VGG-8	0.1	0.6	6.0x

Table 2: Number of training steps per second for two architectures, using batch size of 8 clips, each having 64 frames and resolution 112x112. The results were obtained using one GPU per network module (6 for Simple CNN and 8 for VGG-8).

the input video clip. This can easily be implemented with the striding operation, *i.e.*, we skip k frames with striding $k+1$. To keep the length of video clips unchanged between the experiments, we sample $k+1$ times longer input sequences, with padding, before we apply striding. We have experimented with striding in $\{2,3,4,5,6\}$. In our experiments, we have found *Sideways* to be surprisingly robust to the changes in striding. Only some choice of the hyper-parameters, *e.g.*, relatively high learning rate, have resulted in the performance collapse, where the network has transitioned from high into low training accuracies. Nonetheless, *BP* and *Sideways* never collapses with the same set of the carefully chosen hyper-parameters. Distortions introduced by padding could be another explanation for the collapse of models trained with 'unfavorable' hyper-parameters and higher striding numbers. We report these results in Figure 5.

Training speed-up using multiple GPUs. We evaluate speedups of training the VGG-8 and Simple-CNN models using a single V100 GPU per module – 8 for VGG and 6 for Simple-CNN. To isolate training speed from the data loading aspect, in this study, we artificially construct videos consisting of random numbers. We train each model for 100 steps, repeat this 3 times and return the highest average number of training steps per second. The results are shown in Table 2, which validate that there is a large speed-up for *Sideways* when parallel resources are assigned along the network depth. In particular, VGG has a more balanced decomposition in terms of FLOPs per module. The *BP* model benefits little from the multiple GPUs since they are locked most of the time waiting for the other GPUs to complete their processing. Note also, that placing different *Sideways* modules in different GPUs will also significantly reduce memory requirements for training large neural networks.

4.4. Results (Auto-Encoding)

We evaluate both algorithms using mean squared error, between pixels, between the predicted and the ground truth sequences, under the same conditions, in particular, under the same frame rate. Here, we present quantitative results, and we point a curious reader to the supplementary material regarding the qualitative results.

	BP	*Sideways*
Auto-encoding	0.014	0.002

Table 3: Mean squared error between predictions and ground truth data; the lower, the better.

Table 3 shows mean squared error (the lower, the better) between predicted frames and ground truth (input frames). We compare the same architecture trained with *Sideways* and the regular *BP*. Because of the synchronization, the method trained with *BP* cannot output at a fast enough pace to keep up with the input frame rate, which yields a significant error.

These results show that the proposed training scheme can be successfully applied also to tasks where both input and output are continuously evolving, reducing considerably the latency of the system during training. Together with the results on the classification task, we conclude that this training scheme is general enough to be applied for a wide range of video tasks.

5. Conclusion

We propose *Sideways* – a backpropagation variant to train networks, where activations from different computation steps are used in the weight updates. We motivate our training algorithm by the smoothness of video signals, and especially we assume that important features vary slowly, at least in the latent space [19, 50].

We have found that *Sideways* is not only a valid learning mechanism but can also potentially provide an implicit regularization during the training of neural networks. Decoupling provided by the *Sideways* algorithm makes it especially attractive for training large models in parallel.

We hope that our work will spark further interest in decoupled training of more advanced temporal models or in a better understanding of the role of slow features, temporal redundancies, and stochasticity in the learning process of such models. Although biological plausibility is not our primary motivation, we believe our architecture has some desired properties. For instance, top-down and global communication implemented in *Sideways* does not necessarily require neither depth-synchronization nor instantaneous propagation; it also does not require local caching of the activations during the weights updates for the backward pass. Finally, its unrolling in time could be viewed as more biologically correct [6, 28].

Acknowledgements

We thank Jean-Baptiste Alayrac, Carl Doersch, Tamara Norman, Simon Osindero, Andrew Zisserman for their help.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. In *AAAI Conference on Artificial Intelligence*, 2015.

[3] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019.

[4] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. In *2nd Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2015.

[5] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[6] Alessandro Betti and Marco Gori. Backprop diffusion is biologically plausible. *arXiv preprint arXiv:1912.04635*, 2019.

[7] Léon Bottou and Yann L Cun. Large scale online learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2004.

[8] João Carreira, Viorica Patraucean, Laurent Mazare, Andrew Zisserman, and Simon Osindero. Massively parallel video networks. In *European Conference on Computer Vision (ECCV)*, 2018.

[9] 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.

[10] Anna Choromanska, Benjamin Cowen, Sadhana Kumaravel, Ronny Luss, Mattia Rigotti, Irina Rish, Brian Kingsbury, Paolo DiAchille, Viatcheslav Gurev, Ravi Tejwani, et al. Beyond backprop: Online alternating minimization with auxiliary variables. In *International Conference on Machine Learning (ICML)*, 2019.

[11] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, 2015.

[12] Wojciech Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *International Conference on Machine Learning (ICML)*, 2017.

[13] Ali Diba, Vivek Sharma, Luc Van Gool, and Rainer Stiefelhagen. Dynamonet: Dynamic action and motion network. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 2011.

[15] Rohit Girdhar and Deva Ramanan. Cater: A diagnostic dataset for compositional actions and temporal reasoning. *International Conference on Learning Representations (ICLR)*, 2020.

[16] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[17] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *International Conference on Computer Vision (ICCV)*, 2019.

[19] Geoffrey E Hinton. Connectionist learning procedures. In *Machine learning*, pages 555–610. Elsevier, 1990.

[20] 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.

[21] Zhouyuan Huo, Bin Gu, and Heng Huang. Training neural networks using features replay. In *Advances in Neural Information Processing Systems*, pages 6659–6668, 2018.

[22] Zhouyuan Huo, Bin Gu, Qian Yang, and Heng Huang. Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning (ICML)*, volume 80, 2018.

[23] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning (ICML)*, 2017.

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

[25] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2014.

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

[27] Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *2011 International Conference on Computer Vision (ICCV)*, 2019.

[28] Jonas Kubilius, Martin Schrimpf, Aran Nayebi, Daniel Bear, Daniel LK Yamins, and James J DiCarlo. Cornet: modeling the neural mechanisms of core object recognition. *BioRxiv*, 2018.

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

[30] Matthew Larkum. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends in neurosciences*, 36(3):141–151, 2013.

[31] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 2016.

[32] Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[33] Mateusz Malinowski, Carl Doersch, Adam Santoro, and Peter Battaglia. Learning visual question answering by bootstrapping hard attention. In *European Conference on Computer Vision (ECCV)*, 2018.

[34] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning unsupervised learning rules. In *International Conference on Learning Representations (ICLR)*, 2019.

[35] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[36] Behnam Neyshabur. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017.

[37] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[38] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

[39] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[40] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[41] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, 2011.

[42] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, 1951.

[43] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Object detection from scratch with deep supervision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019.

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

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

[46] 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.

[47] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on International Conference on Machine Learning (ICML)*, 2013.

[48] Hyoe Tomita, Machiko Ohbayashi, Kiyoshi Nakahara, Isao Hasegawa, and Yasushi Miyashita. Top-down signal from prefrontal cortex in executive control of memory retrieval. *Nature*, 401(6754), 1999.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[50] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[51] Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, and Shilei Wen. Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition. *CoRR*, abs/1907.13369, 2019.

[52] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR)*, 2017.

[53] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *International Conference on Machine learning (ICML)*. ACM, 2004.