3D ConvNets with Optical Flow Based Regularization

Kevin Chavez
Stanford University
Stanford, CA
kjchavez@stanford.edu

Abstract

Video classification using 3D convolutional neural networks still lags behind models with simple classifiers on top of rich, hand-engineered, spatio-temporal features for a number of prominent action recognition datasets. Many of these hand-designed features are built on top of estimates of optical flow. Thus we propose an extension to the 3D convolutional neural network model that incorporates the underlying assumptions of optical flow as a form of regularization and analyze the effects of this extension on early-stage training of a 5-layer architecture on the UCF-101 Human Action Recognition dataset.

1. Introduction

Convolutional neural networks have proven to be extremely effective at single image classification. A natural question is how to extend the model to the video domain. However, the best way to leverage the additional temporal information isn't obvious. Perhaps the most intuitive approach is to extend the convolution along the temporal axis. Some variants of this have been studied by Karpathy *et al*. and Ji *et al*. in recent years [6, 5]. The best published results on many prominent action recognition data sets (excluding Sports 1M), however, are still achieved by simple classifiers on top of rich, hand-engineered spatio-temporal features.

When trained from scratch on datasets like UCF101 or HMDB-51, large 3D convolutional neural networks consistently suffer from overfitting. Newer, larger video datasets such as the Sports 1M will likely help alleviate this problem, yet incur a much greater computational cost. Is there another way? Recently, novel regularization techniques such as dropout and maxout have proven adept at combating overfitting [10]. Further, the best shallow models for action recognition include features which are derived from estimates of dense optical flow [7]. Together, these suggest an alternative and/or complementary approach to improving 3D ConvNets for video classification.

We propose a regularizer which encourages the first layer filter weights to satisfy the assumptions of *brightness constancy* and *smoothness* that are typically used when estimating optical flow. We analyze properties of this regularizer that suggest it might be a useful extension to the 3D ConvNet model. Finally, using a modest 5-layer architecture, we conduct controlled overfitting experiments on a subset of the UCF101 human action recognition dataset.

2. Background

Optical flow estimates are used in numerous models for video classification. For the UCF101 data set, Peng *et al*. achieve a state-of-the-art 87.9% accuracy using "improved dense trajectories" [7] as part of their representation. These trajectories, developed by Wang *et al*., are created by robustly estimating optical flow, then aligning the motion over multiple frames [11]. Further, Simonyan and Zisserman propose a (2D) ConvNet-based approach which includes optical flow. Their model consists of two independent networks [8]. One operates on single static frames from the video. The second takes explicitly calculated optical flow or trajectory estimates as its input. Prediction is then done by treating the two networks as an ensemble and linearly combining their outputs. They also evaluated their model on UCF-101 and achieved competitive results.

There are a few 3D models that attempt to learn from the raw pixel data. One of the results from Baccouche *et al*. shows a small 3D convolutional neural network achieving 91% accuracy on the KTH dataset. They state that "no overtraining is observed," however this is expected since their model only has 3 small convolutional layers. On the a much larger scale, Karpathy *et al*.'s Slow Fusion model is also a 3D convolutional neural network. When trained from scratch on the UCF-101 dataset, it achieves an accuracy of 41.3% and is noted to suffer from overfitting. (They boost this to 65.4% by training on the much larger Sports 1M and then doing transfer learning, but this takes about a month of computation.)

In general, on datasets for which we can compare performance, the models that do best incorporate optical flow in some manner. Further, 3D convolutional models might

do well when they are small and are evaluated on small datasets, but are crippled by overfitting as they scale.

3. OptFlow Regularizer

Let $W \in \mathbf{R}^{T\times H\times W}$ be a spatiotemporal convolution kernel. Typically, for such a parameter, we add an L_2 regularization term $\|W\|_F^2$ to the loss function. This has the effect of keeping weights in the filter small. However, it's not very informative as a surrogate for a prior distribution on useful spatiotemporal filters. In this section we describe in detail the proposed OptFlow regularizer $R(W)$ which aims to address this concern.

3.1. Optical Flow Assumptions

Optical flow estimation attempts to approximate the instantaneous 2D motion field of all visible surfaces in an image from a given sequence [4]. While methods range from simple to quite sophisticated (see for example [3]), all approaches include the assumptions of *brightness constancy* and *smoothness* which we define below. Optical flow estimates are obtained by minimizing the violation of these assumptions.

Brightness constancy. In its most abstract form, this assumption states that corresponding points from different times will have the same apparent brightness,

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{1}$$

By doing a Taylor expansion about (x, y, t), we arrive at a linear approximation of this assumption,

$$I_x u + I_y v + I_t = 0 \tag{2}$$

where subscripts denote partial derivatives, and u and v express the horizontal and vertical velocity of the point (x, y) as it moves to $(x + \Delta x, y + \Delta y)$.

Smoothness. We further assume that the vector field describing the optical flow is "smooth". In our case, we interpret this as the sum of squared partial derivatives of u and v,

$$u_x^2 + u_y^2 + u_t^2 + v_x^2 + v_y^2 + v_t^2,$$

is small. Note, here we are treating u and v as continuous, differentiable functions of x, y, and t for simplicity.

3.2. Optical Flow Loss Function

We define a loss function $L(W, U, V)$ that reflects the violation of the optical flow assumptions of a proposed discretized motion field $U, V \in \mathbf{R}^{T\times H\times W}$ for the convolution kernel W.

To simplify notation, let the superscript $(\cdot)^{(t)}$ refer to the t^{th} "frame" of a spatiotemporal tensor, and let's define the image gradient operators $D_x \in \mathbf{R}^{W\times W}$ and $D_y \in \mathbf{R}^{H\times H}$ of the form

$$D = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & & & & & 1 \\ 0 & 0 & 0 & \cdots & -1 & 0 \end{pmatrix}.$$

With this notation we can express the brightness constancy violation as $B(W, U, V) =$

$$\sum_{t=1}^{T} \left\| W^{(t)} D_x^T \circ U^{(t)} + D_y W^{(t)} \circ V^{(t)} + W^{(t+1)} - W^{(t)} \right\|_F^2,$$

where $\circ$ denotes an elementwise product and we define $W^{(T+1)} = 0$.

Similarly, we define a cost associated with smoothness,

$$S(U, V) = \sum_{t=1}^{T} \left\| \begin{matrix} U^{(t)} D_x^T \\ D_y U^{(t)} \\ U^{(t+1)} - U^{(t)} \end{matrix} \right\|_F^2 + \left\| \begin{matrix} V^{(t)} D_x^T \\ D_y V^{(t)} \\ V^{(t+1)} - V^{(t)} \end{matrix} \right\|_F^2,$$

where we let $V^{(T+1)} = U^{(T+1)} = 0$.

Finally, the optical flow loss function is defined as

$$L(W, U, V) = \frac{1}{2} \left(B(W, U, V) + \alpha S(U, V) \right) \tag{3}$$

where α is a smoothness parameter (in our experiments we set $\alpha = 1$). The minimizers $U^\star$, $V^\star$ of L give an approximation of the motion field for the spatiotemporal filter W.

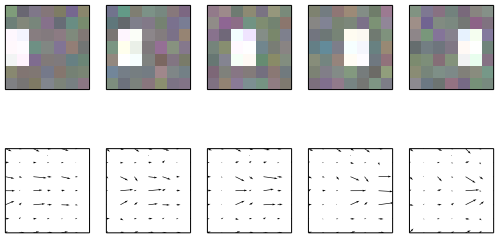

Figure 1. Optical flow estimate using the objective function (3) for a shifting blob with added Gaussian noise with a standard deviation of 10% of the magnitude of the blob.

3.3. Definition

The optflow regularization cost is defined as the minimum over U and V of the optical flow loss function,

$$R(W) = \min_{U,V} L(W, U, V), \tag{4}$$

where W is the spatiotemporal filter. If the filter has multiple input channels, we linearly collapse them to a single channel so that $W \in \mathbf{R}^{T \times H \times W}$. To use this regularizer while training a 3D convolutional neural network, we also need its gradient with respect to the filter,

$$\nabla_W R(W) = \nabla_W L(W, U^\star, V^\star). \tag{5}$$

(Note: the derivation of this is in the supplementary materials.) This cost and gradient will be evaluated at every iteration of training. Therefore it is absolutely critical that the optimization problem in (5) be solved efficiently, which—by design—is possible.

4. Motivation and Intuition

Why might optflow regularization be a useful addition to the standard 3D ConvNet model? First, it biases training towards temporally coherent filters. Second, it can work to complement L_2 regularization. Lastly, from a practical standpoint, it is (relatively) computationally inexpensive. We expand on these three points in this section.

4.1. Temporally Coherent Filters

Optflow regularization, by construction, will push the filter to have weights that produce good optical flow estimates. In other words, the temporal distribution of weights can be well approximated by starting with the first time slice of the filter, and then smoothly moving the weights around over time. This is how we define temporally coherent filters. For example, consider the two $7 \times 7 \times 7$ filters in Figure 2. Both of these filters have the same L_2 norm, yet the top filter incurs an optflow regularization cost nearly three times greater than the bottom one.

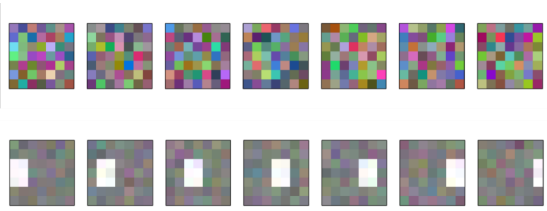

Figure 2. Frames of spatiotemporal filters with regularization costs $R(W_{\text{top}}) = 235$ and $R(W_{\text{bottom}}) = 80$. To visualize, we've shifted all pixel values by 127 so that a filter of all zeros would be gray.

Since the activation of neuron with a filter W looking at a particular space-time patch X is obtained by taking the inner product $W \cdot X$ (and a bias term), a neuron with a temporally coherent filter will be more sensitive to patches which themselves have temporal coherence, *i.e.* moving edges, corners, blobs, *etc.* Intuitively, this seems like a desirable property. Empirically, we note that many of the first layer filters obtained by Karpathy's Slow Fusion model after training on the Sports 1M dataset (`cs.stanford.edu/people/karpathy/deepvideo/slow_motion5x5_filters.gif`) appear to have this property.

4.2. Complementing L2 Regularization

Two important properties of the optflow regularizer suggest that it can work in tandem with standard L_2 regularization but might provide some additional benefit.

First, scaling down all of a filter's weights by a single factor is guaranteed to decrease the optflow regularization cost. Suppose we have a filter W, with the optical flow estimate $U^\star$, $V^\star$ and a scale factor $\gamma < 1$. Then,

$$\begin{aligned} R(\gamma W) &= \min_{U,V} \frac{1}{2}(B(\gamma W, U, V) + S(U, V)) && (6)\\ &\leq \frac{1}{2}(B(\gamma W, U^\star, V^\star) + S(U^\star, V^\star)) && (7)\\ &= \frac{1}{2}(\gamma^2 B(W, U^\star, V^\star) + S(U^\star, V^\star)) && (8)\\ &< \frac{1}{2}(B(W, U^\star, V^\star) + S(U^\star, V^\star)) && (9)\\ &= R(W). && (10) \end{aligned}$$

This scaling operation is precisely the effect that L_2 regularization has on the weights. Thus taking step to decrease the L_2 norm will not increase the optflow regularization cost. Moreover, filters with smaller L_2 norm generally tend to have a lower optflow regularization penalty as well. This correlation is shown in Figure 3 for 600 randomly generated filters with elements drawn from normal, uniform, and Laplace distributions.

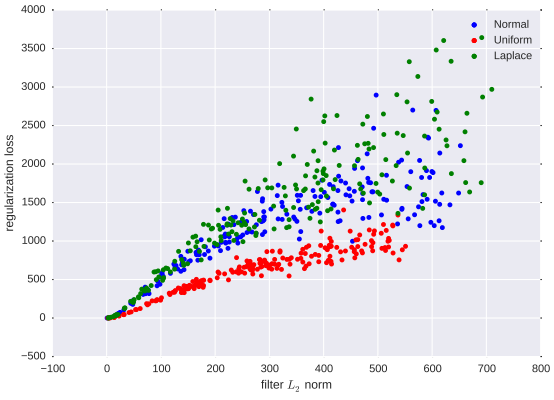

Figure 3. Randomly generated $7 \times 7 \times 7$ filters plotted by their L_2 norm and optflow regularization cost.

Second, and more importantly, optflow regularization does not primarily act as weight decay. For a filter W,

if we take a step along $\nabla_W R(W)$, the optflow regularization cost decreases substantially, while the L_2 norm is only marginally affected, as shown in Figure 4. This suggests that the optflow regularizer is much more sensitive to orientation in the space of possible filters. Therefore we believe its possible for such a regularization technique to improve performance somewhat orthogonally to L_2 regularization (no pun intended).

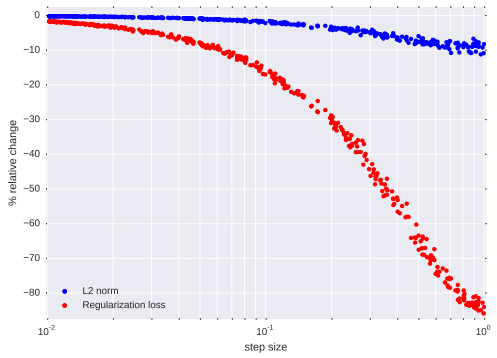

Figure 4. Four hundred randomly initialized filters are shifted by various step sizes along the negative gradient of the optflow regularization cost, showing dramatic changes in the regularization cost, but very modest changes in L_2 norm.

4.3. Computational efficiency

For a fixed W, the function L is a convex quadratic in U and V. Further, since the loss function only includes local spatial and temporal coupling, the Hessian is quite sparse. Together, these two facts imply that we can solve the problem *exactly* using a single Newton step (with step size 1) and the computational complexity is roughly linear in the size of the filter.

Newton's method for a convex quadratic. Suppose we wish to solve an unconstrained convex optimization problem,

$$\text{minimize} \quad f(x).$$

Recall that given some point x_n, a Newton step of size γ produces the update

$$x_{n+1} = x_n - \gamma[Hf(x_n)]^{-1}\nabla_x f(x_n).$$

If f is a convex quadratic, then the Hessian is constant and regardless of the starting point x_0, the solution $x^\star$ is reached in a single step. In particular, let $x_0 = 0$, then

$$x^\star = -H^{-1}\nabla_x f(0),$$

so we can find the solution by solving a system of linear equations,

$$Hx^\star = -\nabla_x f(0). \tag{11}$$

Further, if H is sparse, then we can use sparse factor-solve methods to solve this system in approximately $O(n)$ [2].

Implementation details. The optimization problem necessary for working with the optflow regularizer fits exactly into this framework. In this project, we derive and explicitly construct the Hessian and gradient at $U = V = 0$, then use the `sparse` module included in SciPy (which relies on the UMFPACK) to solve the system in (11).

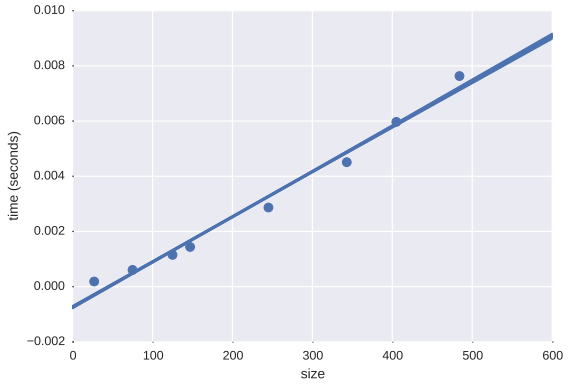

Figure 5. Computational complexity is roughly linear in the size of the filter. The x axis shows the product of height, width and number of frames for a filter. Each point is the mean computation time over 50 random filters.

Figure 5 shows empirical computation times on an Intel Core i5-4200U CPU for a range of different sized random filters. This is just the time for solving the sparse system of linear equations. Constructing the Hessian and gradient require additional time but is also $O(n)$. The range shown is representative of filter sizes typically seen in practice. The 3D ConvNet in Baccouche *et al*. uses $7 \times 7 \times 5$ filters in the first layer [1]. Ji *et al*. use kernels of size $7 \times 7 \times 3$ [5]. Both of these filters operate on single channel input volumes. The largest filter size we see is in the first layer of the Slow Fusion model by Karpathy *et al*., which uses an $11 \times 11 \times 4$ filter.

5. Experiment

To test the effectiveness of the optflow regularizer, we train 3D convolutional neural networks from random initialization both with and without the proposed regularization on a small human action recognition dataset. Since the result of a single trial of training can be very sensitive to initialization and hyperparameter choices, we are careful to make a fair comparison by keeping all other hyperparameters constant and using the same seed for the random number generator. In other words, we conduct multiple pairs of trials where the *only* difference bewteen the two is the exclusion, inclusion or strength of optflow regularization.

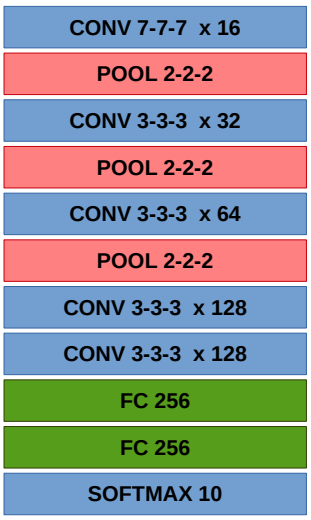

Figure 6. The architecture used in these experiments. Larger filters used in the first layer because stacked $3 \times 3 \times 3$ convolution layers use too much memory.

5.1. Data

We use a subset of the UCF101 dataset. In particular, we only use the first 10 classes: ApplyEyeMakeup, ApplyLipstick, Archery, BabyCrawling, BalanceBeam, BandMarching, BaseballPitch, Basketball, BasketballDunk, BenchPress. This yields a total of 985 video clips for training and 389 videos for testing. We split the 985 videos into 876 for the training set and 109 for the validation set, following the guidelines suggested by CRCV [9] that videos from the same group **not** be included in both training and validation sets.

From each video we extract 4 clips of 16 frames each, equally spaced in time, take a center crop of 224 by 224 pixels and downsample by a factor of 2. Thus, we have 3504 samples for training and 436 for validation. This small dataset makes it tractable for us to study overfitting with a moderately sized convolutional neural network. However, a next step for this work would be to use the full UCF-101 dataset and a larger architecture and repeat these experiments.

5.2. Architecture

Due primarily to memory constraints, our ConvNet architecture is limited in size. We use a network with 5 convolutional layers, 2 fully connected layers and a softmax layer. Figure 6 shows the details.

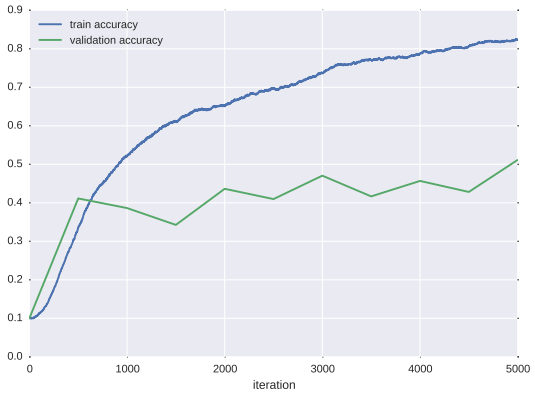

Figure 7. Reference training trial using default hyperparameters.

Parameter	Value
Initial learning rate	10^{-5}
L_2 regularization strength	10^{-3}
Dropout probability	0.20
Learning rate decay (per epoch)	0.95
Momentum (initial)	0.5
Momentum (step per epoch)	0.1
Momentum (max)	0.9
Batch size	8

Table 1. Default hyperparameters for our reference trial.

5.3. Results

For reference, we train a network without optflow regularization and with the hyperparameters shown in Table 1. As expected, this network overfits the training data rather quickly as shown in Figure 7. To avoid significant overhead during training, the training accuracy is approximated obtained by averaging minibatch accuracies over the past epoch. In expectation, this is a lower bound on the true training accuracy at that particular iteration. The true training accuracy at the last iteration was 92.88%. Validation accuracy is evaluated every 500 iterations which is just a little over an epoch.

We then modify our reference settings by incorporating increasing strengths of optflow regularization, ranging from 10^{-2} to 10^{5}. Figure 8 shows the effect on validation and training accuracy. For comparison, we also attempt to alleviate the overfitting problem by increasing L_2 regularization (Figure 9) or increasing the dropout probability (Figure 10). In the trial with the strongest optflow regularization (10^5), the training accuracy fell to 89.7%. With the strongest L_2 regularization (10^1), it fell to 72.9%, and with the highest dropout probability (0.5), to 83.4%.

Finally, we qualitatively compare the first layer filters in the reference model to those in a strongly optflow reg-

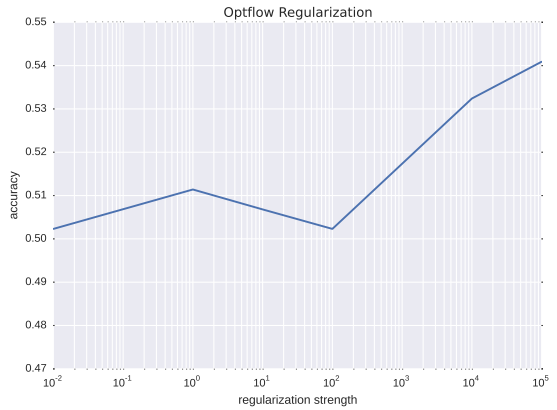

Figure 8. Effect of increasing optflow regularization strength on maximum validation accuracy.

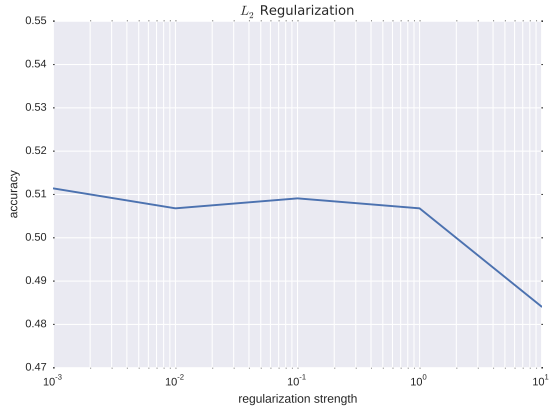

Figure 9. Effect of increasing L_2 regularization strength on maximum validation accuracy.

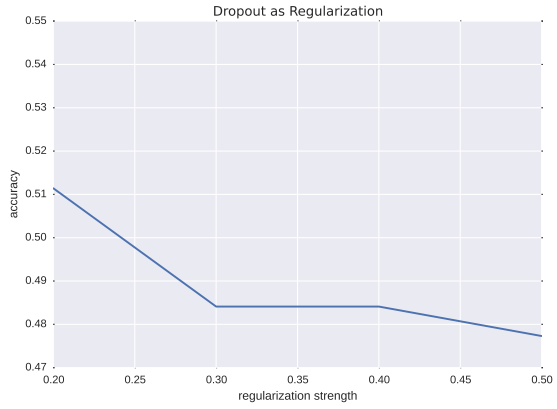

Figure 10. Effect of increasing dropout probability.

ularized model. A sample of first layer activation maps for some training data is available at `www.stanford.edu/~kjchavez/projects/conv3d.html`. While it's difficult to make any definitive statements, in general, it appears that the optflow regularization generally produces more "selective" filters. The activation maps for the strongly regularized model have smaller spatio-temporal regions of high activation.

6. Conclusion

The results presented in this report are preliminary. We have not conducted sufficient trials to make statistically meaningful statements about the efficacy of optflow regularization. However, these initial trials are encouraging.

The default parameters represent a reasonable starting point, found by doing a quick coarse random search in the hyperparameter space. From that point, we notice that we are severely overfitting our dataset after 5000 iterations. We try increasing L_2 regularization, but the validation accuracy decreases. We try increasing the probability of dropout, but again the validation accuracy decreases. At this point, we could run finer-grained random search for more iterations. However, introducing rather strong optflow regularization reduces the amount of overfitting and improve our validation accuracy by about 3%. Further, it had the smallest effect on our training accuracy (which only fell by about 3%). This is the story that unfolded during this project. We're careful **not** to claim that this is meaningful evidence supporting optflow regularization, but at the very least, it does not refute it.

There are a few things that we can do in the immediate future to make these measurements a bit more precise and significant. First, for a few of these trials (including the best optflow regularization trial) the maximum validation accuracy was achieved at the last iteration. Because this was incredibly troubling, we resumed training of both the reference and the best optflow regularized trial for 5000 additional iterations. However, the results do not contradict anything we've discussed so far. In both cases, overfitting overpowered the regularization and performance on the validation set declined. We should try the same with some of the dropout trials since a higher dropout rate will usually take longer to train.

The second thing we can do is repeat the full experiment at many other "reasonable" starting points in the hyperparameter space and for higher optflow regularization strengths. Notice that we achieved the best performance with the greatest optflow regularization that we tried. Since the entire experiment pipeline is built, this simply involves changing a few command line arguments and waiting. Each 5000 iteration trial takes approximately 170 minutes on Terminal.com GPU instances (or roughly forever on the Rye machines).

In the slightly longer term, we'd like to scale up the experiments presented here to the full UCF-101 dataset with deeper architectures. This presents a substantial computational challenge, as even with this modest architecture and downsampling the videos, we could only use a batch size of eight 16-frame clips.

Other interesting future work could be in optimizing the computation of optflow regularization by putting it on the GPU. The cuSPARSE library is likely a good starting point for this. Currently, we have to transfer the filter weights from the GPU to the CPU every iteration to apply optflow regularization which slows down the process considerably.

Finally, in pursuing these experiments, we have developed a code base for 3D convolutional neural networks in Theano that builds on top some existing work on Github from user *lpigou*. We hope that others who might want to explore extensions to 3D convolutional neural networks can benefit from this and possibly contribute. Please see the repository at `https://github.com/kjchavez/cnn-project`.

References

[1] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In A. Salah and B. Lepri, editors, *Human Behavior Understanding*, volume 7065 of *Lecture Notes in Computer Science*, pages 29–39. Springer Berlin Heidelberg, 2011.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[3] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision-ECCV 2004*, pages 25–36. Springer, 2004.

[4] D. J. Fleet and Y. Weiss. Optical flow estimation. In N. Paragios, Y. Chen, and O. Faugeras, editors, *Handbook of Mathematical Models in Computer Vision*, chapter 15, page 239258. 2006.

[5] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):221–231, Jan 2013.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1725–1732, June 2014.

[7] X. Peng, L. Wang, X. Wang, and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *CoRR*, abs/1405.4506, 2014.

[8] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[9] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[11] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action Recognition by Dense Trajectories. In *IEEE Conference on Computer Vision & Pattern Recognition*, pages 3169–3176, Colorado Springs, United States, June 2011.

Supplementary Materials

Proof of OptFlow Regularizer Gradient

For any filter W, the function $f_W(U, V) = L(W, U, V)$ is a convex quadratic. For simplicity of notation, let's flatten all variables into long vectors, but use the same names. Let $U^\star(W)$, $V^\star(W)$ be the minimizers of f_W. With this notation, we have $R(W) = L(W, U^\star(W), V^\star(W))$. Using the chain rule,

$$\begin{aligned}\nabla_W R(W) =& \nabla_W L(W, U^\star(W), V^\star(W)) \\ &+ \left(\frac{\partial U}{\partial W}\bigg|_W\right)^T \nabla_U L(W, U^\star(W), V^\star(W)) \\ &+ \left(\frac{\partial V}{\partial W}\bigg|_W\right)^T \nabla_V L(W, U^\star(W), V^\star(W)).\end{aligned} \tag{12}$$

The expressions $\frac{\partial U}{\partial W}\Big|_W$ and $\frac{\partial V}{\partial W}\Big|_W$ are rather complicated. However, by definition,

$$\nabla_U L(W, U^\star(W), V^\star(W)) = \nabla_U f_W(U^\star, V^\star)$$

and

$$\nabla_V L(W, U^\star(W), V^\star(W)) = \nabla_V f_W(U^\star, V^\star).$$

Further, since $U^\star$ and $V^\star$ are minimizers of the convex function f_W,

$$\nabla_U f_W(U^\star, V^\star) = 0$$

$$\nabla_V f_W(U^\star, V^\star) = 0.$$

Thus, the second two terms in (12) are zero and we are left with

$$\nabla_W R(W) = \nabla_W L(W, U^\star(W), V^\star(W)). \tag{13}$$

Derivation of Gradient and Hessian

We claimed that even though the value of the regularizer is the solution to an optimization problem, that it is efficient to solve. Deriving the expressions for the gradient and Hessian is straightforward, albeit omewhat tedious. A copy of my scratch work is available upon request, as well as a few empirical tests to computationally verify the expressions. Alternatively, the code to compute the gradient and construct the Hessian is in `src/convnet3d/regularization.py`. It is not fully optimized and should be interpretable.

Code Repository

All code is publicly available at `https://github.com/kjchavez/cnn-project`.

Auxilary visuals

It's difficult to display temporal data in a static report. Thus, we share some interesting visualizations online at this link: `www.stanford.edu/~kjchavez/projects/conv3d.html`.