Unsupervised Learning of Video Representations using LSTMs

Nitish Srivastava   NITISH@CS.TORONTO.EDU
Elman Mansimov   EMANSIM@CS.TORONTO.EDU
Ruslan Salakhutdinov   RSALAKHU@CS.TORONTO.EDU
University of Toronto, 6 Kings College Road, Toronto, ON M5S 3G4 CANADA

Abstract

We use multilayer Long Short Term Memory (LSTM) networks to learn representations of video sequences. Our model uses an encoder LSTM to map an input sequence into a fixed length representation. This representation is decoded using single or multiple decoder LSTMs to perform different tasks, such as reconstructing the input sequence, or predicting the future sequence. We experiment with two kinds of input sequences – patches of image pixels and high-level representations ("percepts") of video frames extracted using a pretrained convolutional net. We explore different design choices such as whether the decoder LSTMs should condition on the generated output. We analyze the outputs of the model qualitatively to see how well the model can extrapolate the learned video representation into the future and into the past. We try to visualize and interpret the learned features. We stress test the model by running it on longer time scales and on out-of-domain data. We further evaluate the representations by fine-tuning them for a supervised learning problem – human action recognition on the UCF-101 and HMDB-51 datasets. We show that the representations help improve classification accuracy, especially when there are only a few training examples. Even models pretrained on unrelated datasets (300 hours of YouTube videos) can help action recognition performance.

1. Introduction

Understanding temporal sequences is important for solving many problems in the AI-set. Recently, recurrent neural networks using the Long Short Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997) have been used successfully to perform various supervised sequence learning tasks, such as speech recognition (Graves & Jaitly, 2014), machine translation (Sutskever et al., 2014; Cho et al., 2014), and caption generation for images (Vinyals et al., 2014). They have also been applied on videos for recognizing actions and generating natural language descriptions (Donahue et al., 2014). A general sequence to sequence learning framework was described by Sutskever et al. (2014) in which a recurrent network is used to encode a sequence into a fixed length representation, and then another recurrent network is used to decode a sequence out of that representation. In this work, we apply and extend this framework to learn representations of sequences of images. We choose to work in the *unsupervised* setting where we only have access to a dataset of unlabelled videos.

Videos are an abundant and rich source of visual information and can be seen as a window into the physics of the world we live in, showing us examples of what constitutes objects, how objects move against backgrounds, what happens when cameras move and how things get occluded. Being able to learn a representation that disentangles these factors would help in making intelligent machines that can understand and act in their environment. Additionally, learning good video representations is essential for a number of useful tasks, such as recognizing actions and gestures.

1.1. Why Unsupervised Learning?

Supervised learning has been extremely successful in learning good visual representations that not only produce good results at the task they are trained for, but also transfer well to other tasks and datasets. Therefore, it is natural to extend the same approach to learning video representations. This has led to research in 3D convolutional nets (Ji et al., 2013; Tran et al., 2014), different temporal fusion strategies (Karpathy et al., 2014) and exploring different ways of presenting visual information to convolutional nets (Simonyan & Zisserman, 2014a). However, videos are much higher dimensional entities compared to single images. Therefore, it becomes increasingly difficult to do credit assignment and learn long range structure, unless we collect much more labelled data or do a lot of feature engineering (for example computing the right kinds of flow features) to keep the

dimensionality low. The costly work of collecting more labelled data and the tedious work of doing more clever engineering can go a long way in solving particular problems, but this is ultimately unsatisfying as a machine learning solution. This highlights the need for using unsupervised learning to find and represent structure in videos. Moreover, videos have a lot of structure in them (spatial and temporal regularities) which makes them particularly well suited as a domain for building unsupervised learning models.

1.2. Our Approach

When designing any unsupervised learning model, it is crucial to have the right inductive biases and choose the right objective function so that the learning signal points the model towards learning useful features. In this paper, we use the LSTM Encoder-Decoder framework to learn video representations. The key inductive bias here is that the same operation must be applied at each time step to propagate information to the next step. This enforces the fact that the physics of the world remains the same, irrespective of input. The same physics acting on any state, at any time, must produce the next state. Our model works as follows. The Encoder LSTM runs through a sequence of frames to come up with a representation. This representation is then decoded through another LSTM to produce a target sequence. We consider different choices of the target sequence. One choice is to predict the same sequence as the input. The motivation is similar to that of autoencoders – we wish to capture all that is needed to reproduce the input but at the same time go through the inductive biases imposed by the model. Another option is to predict the future frames. Here the motivation is to learn a representation that extracts all that is needed to extrapolate the motion and appearance beyond what has been observed. These two natural choices can also be combined. In this case, there are two decoder LSTMs – one that decodes the representation into the input sequence and another that decodes the same representation to predict the future.

The inputs to the model can, in principle, be any representation of individual video frames. However, for the purposes of this work, we limit our attention to two kinds of inputs. The first is image patches. For this we use natural image patches as well as a dataset of moving MNIST digits. The second is high-level "percepts" extracted by applying a convolutional net trained on ImageNet. These percepts are the states of last (and/or second-to-last) layers of rectified linear hidden states from a convolutional neural net model.

In order to evaluate the learned representations we qualitatively analyze the reconstructions and predictions made by the model. For a more quantitative evaluation, we use these LSTMs as initializations for the supervised task of ac-

tion recognition. If the unsupervised learning model comes up with useful representations then the classifier should be able to perform better, especially when there are only a few labelled examples. We find that this is indeed the case.

1.3. Related Work

The first approaches to learning representations of videos in an unsupervised way were based on ICA (van Hateren & Ruderman, 1998; Hurri & Hyvärinen, 2003). Le et al. (2011) approached this problem using multiple layers of Independent Subspace Analysis modules. Generative models for understanding transformations between pairs of consecutive images are also well studied (Memisevic, 2013; Memisevic & Hinton, 2010; Susskind et al., 2011). This work was extended recently by Michalski et al. (2014) to model longer sequences.

Recently, Ranzato et al. (2014) proposed a generative model for videos. The model uses a recurrent neural network to predict the next frame or interpolate between frames. In this work, the authors highlight the importance of choosing the right loss function. It is argued that squared loss in input space is not the right objective because it does not respond well to small distortions in input space. The proposed solution is to quantize image patches into a large dictionary and train the model to predict the identity of the target patch. This does solve some of the problems of squared loss but it introduces an arbitrary dictionary size into the picture and altogether removes the idea of patches being similar or dissimilar to one other. Designing an appropriate loss function that respects our notion of visual similarity is a very hard problem (in a sense, almost as hard as the modeling problem we want to solve in the first place). Therefore, in this paper, we use the simple squared loss objective function as a starting point and focus on designing an encoder-decoder RNN architecture that can be used with any loss function.

2. Model Description

In this section, we describe several variants of our LSTM Encoder-Decoder model. The basic unit of our network is the LSTM cell block. Our implementation of LSTMs follows closely the one discussed by Graves (2013).

2.1. Long Short Term Memory

In this section we briefly describe the LSTM unit which is the basic building block of our model. The unit is shown in Fig. 1 (reproduced from Graves (2013)).

Each LSTM unit has a cell which has a state c_t at time t. This cell can be thought of as a memory unit. Access to this memory unit for reading or modifying it is controlled through sigmoidal gates – input gate i_t, forget gate f_t and
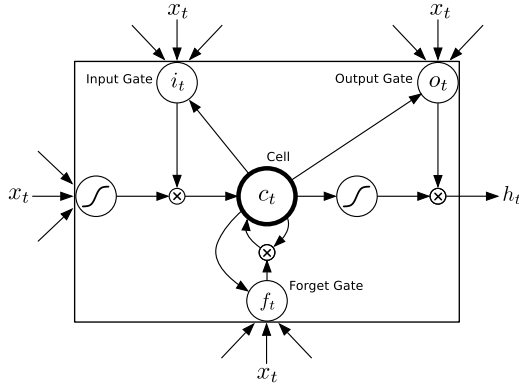
Figure 1. LSTM unit

output gate o_t. The LSTM unit operates as follows. At each time step it receives inputs from two external sources at each of the four terminals (the three gates and the input). The first source is the current frame $\mathbf{x}_t$. The second source is the previous hidden states of all LSTM units in the same layer $\mathbf{h}_{t-1}$. Additionally, each gate has an internal source, the cell state c_{t-1} of its cell block. The links between a cell and its own gates are called *peephole* connections. The inputs coming from different sources get added up, along with a bias. The gates are activated by passing their total input through the logistic function. The total input at the input terminal is passed through the tanh non-linearity. The resulting activation is multiplied by the activation of the input gate. This is then added to the cell state after multiplying the cell state by the forget gate's activation f_t. The final output from the LSTM unit h_t is computed by multiplying the output gate's activation o_t with the updated cell state passed through a tanh non-linearity. These updates are summarized for a layer of LSTM units as follows

$$\mathbf{i}_t = \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right),$$
$$\mathbf{f}_t = \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right),$$
$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t\tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right),$$
$$\mathbf{o}_t = \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{co}\mathbf{c}_t + \mathbf{b}_o\right),$$
$$\mathbf{h}_t = \mathbf{o}_t\tanh(\mathbf{c}_t).$$

Note that all $W_{c\bullet}$ matrices are diagonal, whereas the rest are dense. The key advantage of using an LSTM unit over a traditional neuron in an RNN is that the cell state in an LSTM unit *sums* activities over time. Since derivatives distribute over sums, the error derivatives don't vanish quickly as they get sent back into time. This makes it easy to do credit assignment over long sequences and discover long-range features.

2.2. LSTM Autoencoder Model

In this section, we describe a model that uses Recurrent Neural Nets (RNNs) made of LSTM units to do unsuper-
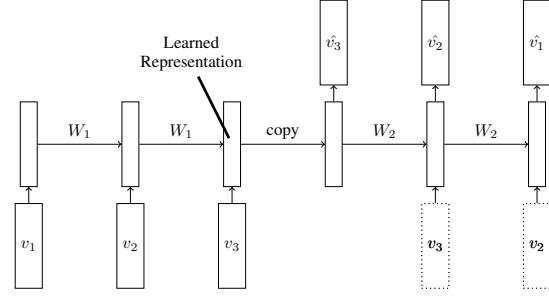

Figure 2. LSTM Autoencoder Model

vised learning. The model consists of two RNNs – the encoder LSTM and the decoder LSTM as shown in Fig. 2. The input to the model is a sequence of vectors (image patches or features). The encoder LSTM reads in this sequence. After the last input has been read, the decoder LSTM takes over and outputs a prediction for the target sequence. The target sequence is same as the input sequence, but in reverse order. Reversing the target sequence makes the optimization easier because the model can get off the ground by looking at low range correlations. This is also inspired by how lists are represented in LISP. The encoder can be seen as creating a list by applying the `cons` function on the previously constructed list and the new input. The decoder essentially unrolls this list, with the hidden to output weights extracting the element at the top of the list (`car` function) and the hidden to hidden weights extracting the rest of the list (`cdr` function). Therefore, the first element out is the last element in.

The decoder can be of two kinds – conditional or unconditioned. A conditional decoder receives the last generated output frame as input, i.e., the dotted input in Fig. 2 is present. An unconditioned decoder does not receive that input. This is discussed in more detail in Sec. 2.4. Fig. 2 shows a single layer LSTM Autoencoder. The architecture can be extend to multiple layers by stacking LSTMs on top of each other.

Why should this learn good features?
The state of the encoder LSTM after the last input has been read is the representation of the input video. The decoder LSTM is being asked to reconstruct back the input sequence from this representation. In order to do so, the representation must retain information about the appearance of the objects and the background as well as the motion contained in the video. However, an important question for any autoencoder-style model is what prevents it from learning an identity mapping and effectively copying the input to the output. In that case all the information about the input would still be present but the representation will be no better than the input. There are two factors that control this behaviour. First, the fact that there are only a fixed number of hidden units makes it unlikely that the model can
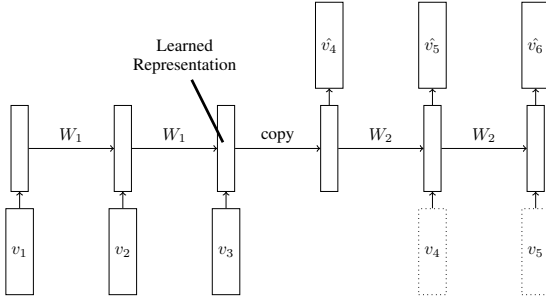
Figure 3. LSTM Future Predictor Model

Figure 4. The Composite Model: The LSTM predicts the future as well as the input sequence.

learn trivial mappings for arbitrary length input sequences. Second, the same LSTM operation is used to decode the representation recursively. This means that the same dynamics must be applied on the representation at any stage of decoding. This further prevents the model from learning an identity mapping.

2.3. LSTM Future Predictor Model

Another natural unsupervised learning task for sequences is predicting the future. This is the approach used in language models for modeling sequences of words. The design of the Future Predictor Model is same as that of the Autoencoder Model, except that the decoder LSTM in this case predicts frames of the video that come after the input sequence (Fig. 3). Ranzato et al. (2014) use a similar model but predict only the next frame at each time step. This model, on the other hand, predicts a long sequence into the future. Here again we can consider two variants of the decoder – conditional and unconditioned.

Why should this learn good features?
In order to predict the next few frames correctly, the model needs information about which objects and background are present and how they are moving so that the motion can be extrapolated. The hidden state coming out from the encoder will try to capture this information. Therefore, this state can be seen as a representation of the input sequence.

2.4. Conditional Decoder

For each of these two models, we can consider two possibilities - one in which the decoder LSTM is conditioned on the last generated frame and the other in which it is not. In the experimental section, we explore these choices quantitatively. Here we briefly discuss arguments for and against a conditional decoder. A strong argument in favour of using a conditional decoder is that it allows the decoder to model multiple modes in the target sequence distribution. Without that, we would end up averaging the multiple modes in the low-level input space. However, this is an issue only if we expect multiple modes in the target sequence distribution. For the LSTM Autoencoder, there is only one correct
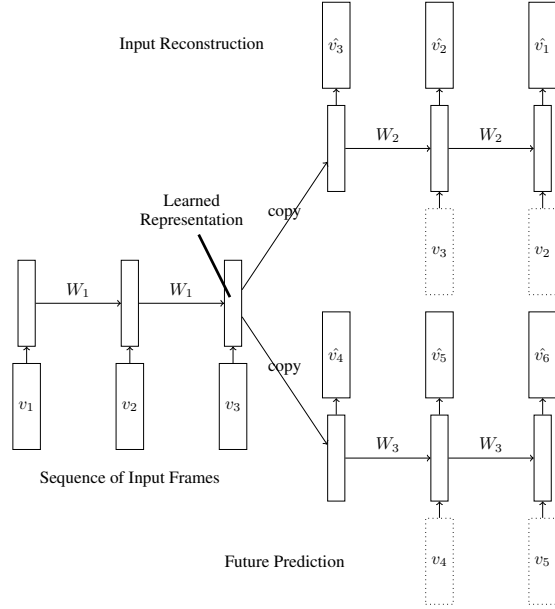
target and hence a unimodal target distribution. But for the LSTM Future Predictor there is a possibility of multiple targets given an input because even if we assume a deterministic universe, everything needed to predict the future will not necessarily be observed in the input.

There is also an argument against using a conditional decoder from the optimization point-of-view. There are strong short-range correlations in video data, for example, most of the content of a frame is same as the previous one. If the decoder was given access to the last few frames while generating a particular frame at training time, it would find it easy to pick up on these correlations. There would only be a very small gradient that tries to fix up the extremely subtle errors that require long term knowledge about the input sequence. In an unconditioned decoder, this input is removed and the model is forced to look for information deep inside the encoder.

2.5. A Composite Model

The two tasks – reconstructing the input and predicting the future can be combined to create a composite model as shown in Fig. 4. Here the encoder LSTM is asked to come up with a state from which we can *both* predict the next few frames as well as reconstruct the input.

This composite model tries to overcome the shortcomings that each model suffers on its own. A high-capacity autoencoder would suffer from the tendency to learn trivial representations that just memorize the inputs. However, this memorization is not useful at all for predicting the future. Therefore, the composite model cannot just memo-

rize information. On the other hand, the future predictor suffers form the tendency to store information only about the last few frames since those are most important for predicting the future, i.e., in order to predict v_t, the frames $\{v_{t-1}, \ldots, v_{t-k}\}$ are much more important than v_0, for some small value of k. Therefore the representation at the end of the encoder will have forgotten about a large part of the input. But if we ask the model to also predict *all* of the input sequence, then it cannot just pay attention to the last few frames.

3. Experiments

We design experiments to accomplish the following objectives:

- Get a qualitative understanding of what the LSTM learns to do.

- Measure the benefit of initializing networks for supervised learning tasks with the weights found by unsupervised learning, especially with very few training examples.

- Compare the different proposed models - Autoencoder, Future Predictor and Composite models and their conditional variants.

- Compare with state-of-the-art action recognition benchmarks.

3.1. Datasets

We use the UCF-101 and HMDB-51 datasets for supervised tasks. The UCF-101 dataset (Soomro et al., 2012) contains 13,320 videos with an average length of 6.2 seconds belonging to 101 different action categories. The dataset has 3 standard train/test splits with the training set containing around 9,500 videos in each split (the rest are test). The HMDB-51 dataset (Kuehne et al., 2011) contains 5100 videos belonging to 51 different action categories. Mean length of the videos is 3.2 seconds. This also has 3 train/test splits with 3570 videos in the training set and rest in test.

To train the unsupervised models, we used a subset of the Sports-1M dataset (Karpathy et al., 2014), that contains 1 million YouTube clips. Even though this dataset is labelled for actions, we did not do any supervised experiments on it because of logistical constraints with working with such a huge dataset. We instead collected 300 hours of video by randomly sampling 10 second clips from the dataset. It is possible to collect better samples if instead of choosing randomly, we extracted videos where a lot of motion is happening and where there are no shot boundaries. However, we did not do so in the spirit of unsupervised

learning, and because we did not want to introduce any unnatural bias in the samples. We also used the supervised datasets (UCF-101 and HMDB-51) for unsupervised training. However, we found that using them did not give any significant advantage over just using the YouTube videos.

We extracted percepts using the convolutional neural net model of Simonyan & Zisserman (2014b). The videos have a resolution of 240×320 and were sampled at almost 30 frames per second. We took the central 224×224 patch from each frame and ran it through the convnet. This gave us the RGB percepts. Additionally, for UCF-101, we computed flow percepts by extracting flows using the Brox method and training the temporal stream convolutional network as described by Simonyan & Zisserman (2014a). We found that the fc6 features worked better than fc7 for single frame classification using both RGB and flow percepts. Therefore, we used the 4096-dimensional fc6 layer as the input representation of our data. Besides these percepts, we also trained the proposed models on 32×32 patches of pixels.

All models were trained using backprop on a single NVIDIA Titan GPU. A two layer 2048 unit Composite model that predicts 13 frames and reconstructs 16 frames took 18-20 hours to converge on 300 hours of percepts. We initialized weights by sampling from a uniform distribution whose scale was set to 1/sqrt(fan-in). Biases at all the gates were initialized to zero. Peep-hole connections were initialized to zero. The supervised classifiers trained on 16 frames took 5-15 minutes to converge.

3.2. Visualization and Qualitative Analysis

The aim of this set of experiments to visualize the properties of the proposed models.

Experiments on MNIST
We first trained our models on a dataset of moving MNIST digits. In this dataset, each video was 20 frames long and consisted of two digits moving inside a 64×64 patch. The digits were chosen randomly from the training set and placed initially at random locations inside the patch. Each digit was assigned a velocity whose direction was chosen uniformly randomly on a unit circle and whose magnitude was also chosen uniformly at random over a fixed range. The digits bounced-off the edges of the 64×64 frame and overlapped if they were at the same location. The reason for working with this dataset is that it is infinite in size and can be generated quickly on the fly. This makes it possible to explore the model without expensive disk accesses or overfitting issues. It also has interesting behaviours due to occlusions and the dynamics of bouncing off the walls.

We first trained a single layer Composite Model. Each LSTM had 2048 units. The encoder took 10 frames as in-
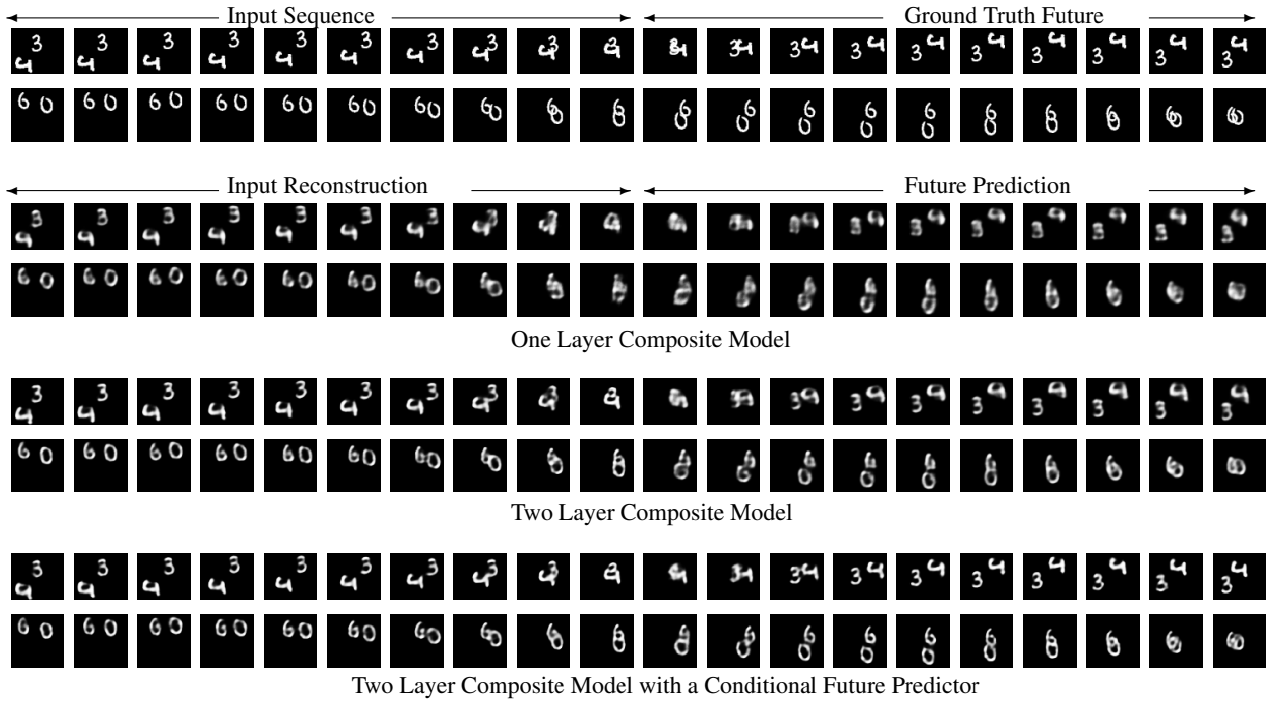
Figure 5. Reconstruction and future prediction obtained from the Composite Model on a dataset of moving MNIST digits.

put. The decoder tried to reconstruct these 10 frames and the future predictor attempted to predict the next 10 frames. We used logistic output units with a cross entropy loss function. Fig. 5 shows two examples of running this model. The true sequences are shown in the first two rows. The next two rows show the reconstruction and future prediction from the one layer Composite Model. It is interesting to note that the model figures out how to separate superimposed digits and can model them even as they pass through each other. This shows some evidence of *disentangling* the two independent factors of variation in this sequence. The model can also correctly predict the motion after bouncing off the walls. In order to see if adding depth helps, we trained a two layer Composite Model, with each layer having 2048 units. We can see that adding depth helps the model make better predictions. Next, we changed the future predictor by making it conditional. We can see that this model makes sharper predictions.

Experiments on Natural Image Patches

Next, we tried to see if our models can also work with natural image patches. For this, we trained the models on sequences of 32×32 natural image patches extracted from the UCF-101 dataset. In this case, we used linear output units and the squared error loss function. The input was 16 frames and the model was asked to reconstruct the 16 frames and predict the future 13 frames. Fig. 6 shows the results obtained from a two layer Composite model with

2048 units. We found that the reconstructions and the predictions are both very blurry. We then trained a bigger model with 4096 units. The outputs from this model are also shown in Fig. 6. We can see that the reconstructions get much sharper.

Generalization over time scales

In the next experiment, we test if the model can work at time scales that are different than what it was trained on. We take a one hidden layer unconditioned Composite Model trained on moving MNIST digits. The model has 2048 LSTM units and looks at a 64×64 input. It was trained on input sequences of 10 frames to reconstruct those 10 frames as well as predict 10 frames into the future. In order to test if the future predictor is able to generalize beyond 10 frames, we let the model run for 100 steps into the future. Fig. 7(a) shows the pattern of activity in the LSTM units of the future predictor pathway for a randomly chosen test input. It shows the activity at each of the three sigmoidal gates (input, forget, output), the input (after the tanh non-linearity, before being multiplied by the input gate), the cell state and the final output (after being multiplied by the output gate). Even though the units are ordered randomly along the vertical axis, we can see that the dynamics has a periodic quality to it. The model is able to generate persistent motion for long periods of time. In terms of reconstruction, the model only outputs blobs after the first 15 frames, but the motion is relatively well preserved. More results, including long range future predictions over hundreds of time steps can see
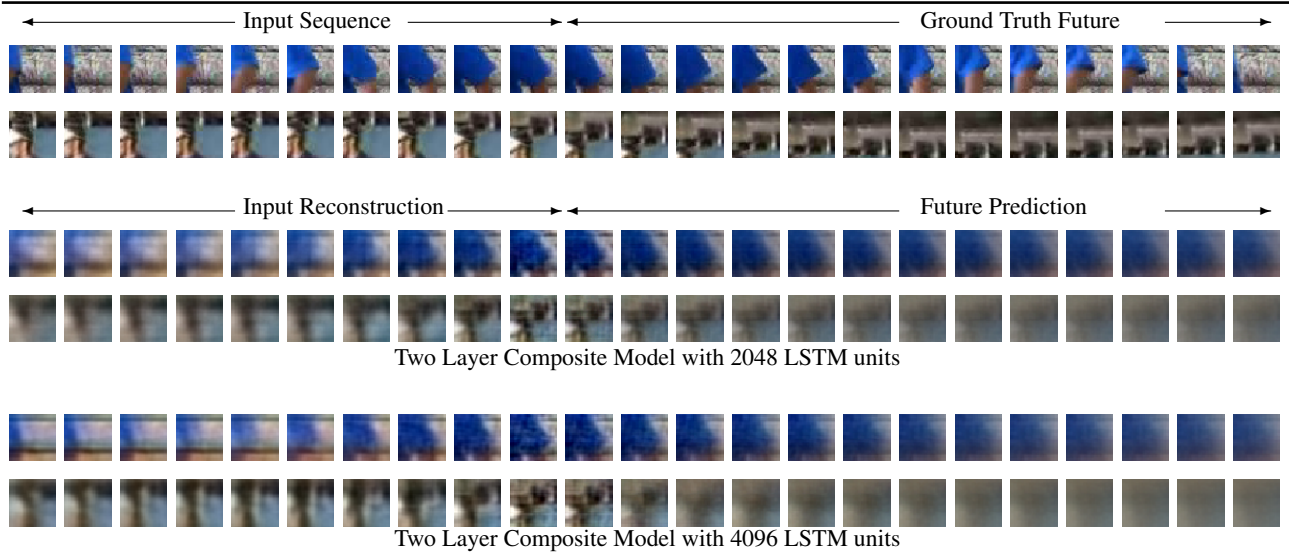
Figure 6. Reconstruction and future prediction obtained from the Composite Model on a dataset of natural image patches. The first two rows show ground truth sequences. The model takes 16 frames as inputs. Only the last 10 frames of the input sequence are shown here. The next 13 frames are the ground truth future. In the rows that follow, we show the reconstructed and predicted frames for two instances of the model.

been at http://www.cs.toronto.edu/˜nitish/unsupervised_video. To show that setting up a periodic behaviour is not trivial, Fig. 7(b) shows the activity from a randomly initialized future predictor. Here, the LSTM state quickly converges and the outputs blur completely.

Out-of-domain Inputs
Next, we test this model's ability to deal with out-of-domain inputs. For this, we test the model on sequences of one and three moving digits. The model was trained on sequences of two moving digits, so it has never seen inputs with just one digit or three digits. Fig. 8 shows the reconstruction and future prediction results. For one moving digit, we can see that the model can do a good job but it really tries to hallucinate a second digit overlapping with the first one. The second digit shows up towards the end of the future reconstruction. For three digits, the model merges digits into blobs. However, it does well at getting the overall motion right. This highlights a key drawback of modeling entire frames of input in a single pass. In order to model videos with variable number of objects, we perhaps need models that not only have an attention mechanism in place, but can also learn to execute themselves a variable number of times and do variable amounts of computation.

Visualizing Features
Next, we visualize the features learned by this model. Fig. 9 shows the weights that connect each input frame to the encoder LSTM. There are four sets of weights. One set of weights connects the frame to the input units. There are three other sets, one corresponding to each of the three gates (input, forget and output). Each weight has a size of 64×64. A lot of features look like thin strips. Others

look like higher frequency strips. It is conceivable that the high frequency features help in encoding the direction and velocity of motion.

Fig. 10 shows the output features from the two LSTM decoders of a Composite Model. These correspond to the weights connecting the LSTM output units to the output layer. They appear to be somewhat qualitatively different from the input features shown in Fig. 9. There are many more output features that are local blobs, whereas those are rare in the input features. In the output features, the ones that do look like strips are much shorter than those in the input features. One way to interpret this is the following. The model needs to know about motion (which direction and how fast things are moving) from the input. This requires *precise* information about location (thin strips) and velocity (high frequency strips). But when it is generating the output, the model wants to hedge its bets so that it does not suffer a huge loss for predicting things sharply at the wrong place. This could explain why the output features have somewhat bigger blobs. The relative shortness of the strips in the output features can be explained by the fact that in the inputs, it does not hurt to have a longer feature than what is needed to detect a location because information is coarse-coded through multiple features. But in the output, the model may not want to put down a feature that is bigger than any digit because other units will have to conspire to correct for it.

It is much harder to visualize the recurrent weights going from the outputs of the LSTM units into the gates at the next time step. We are currently working on good ways to get those visualizations.
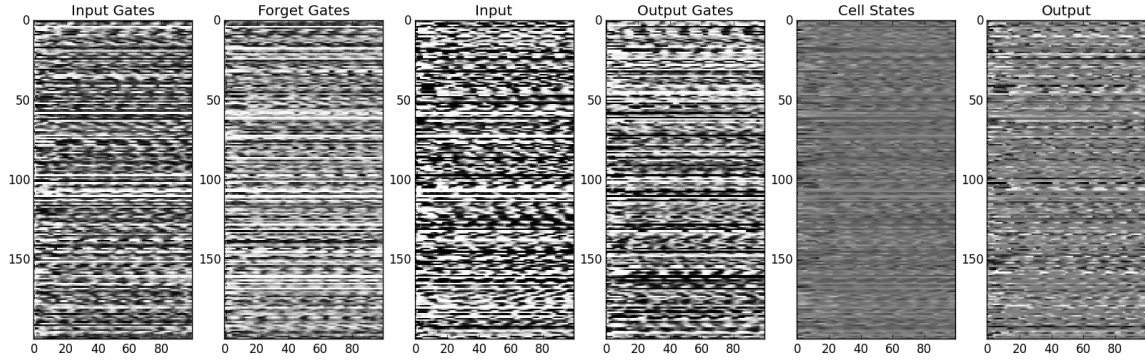
(a) Trained Future Predictor
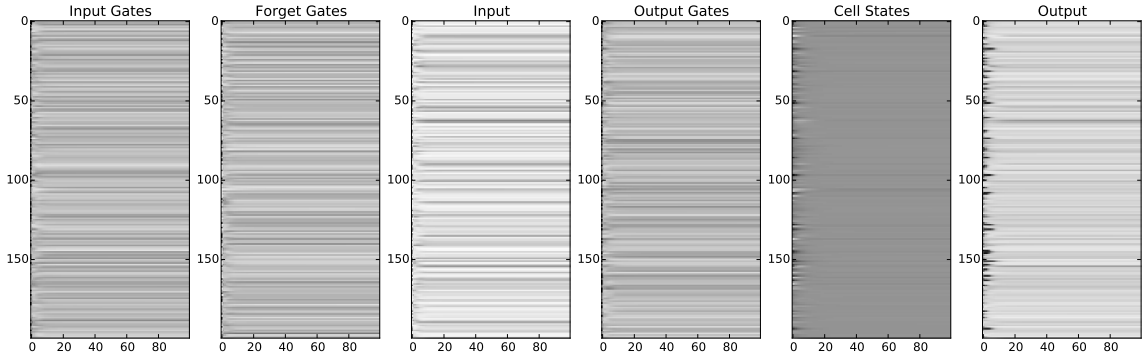
(b) Randomly Initialized Future Predictor

Figure 7. Pattern of activity in 200 randomly chosen LSTM units in the Future Predictor of a 1 layer (unconditioned) Composite Model trained on moving MNIST digits. The vertical axis corresponds to different LSTM units. The horizontal axis is time. The model was only trained to predict the next 10 frames, but here we let it run to predict the next 100 frames. **Top**: The dynamics has a periodic quality which does not die out. **Bottom** : The pattern of activity, if the trained weights in the future predictor are replaced by random weights. The dynamics quickly dies out.

3.3. Action Recognition on UCF-101/HMDB-51

The aim of this set of experiments is to see if the features learned by unsupervised learning can help improve performance on supervised tasks.

We trained a two layer Composite Model with 2048 hidden units with no conditioning on either decoders. The model was trained on percepts extracted from 300 hours of YouTube data. The model was trained to autoencode 16 frames and predict the next 13 frames. We initialize an LSTM classifier with the weights learned by the encoder LSTM from this model. The classifier is shown in Fig. 11. The output from each LSTM in the second layer goes into a softmax classifier that makes a prediction about the action being performed at each time step. Since only one action is being performed in each video in the datasets we consider, the target is the same at each time step. At test time, the predictions made at each time step are averaged. To get a prediction for the entire video, we average the predictions from all 16 frame blocks in the video with a stride of 8 frames. Using a smaller stride did not improve results.

The baseline for comparing these models is an identical LSTM classifier but with randomly initialized weights. All classifiers used dropout regularization, where we dropped activations as they were communicated across layers but not through time within the same LSTM as proposed in Zaremba et al. (2014). We emphasize that this is a very strong baseline and does significantly better than just using single frames. Using dropout was crucial in order to train good baseline models especially with very few training examples.

Fig. 12 compares three models - single frame classifier (logistic regression), baseline LSTM classifier and the LSTM classifier initialized with weights from the Composite Model as the number of labelled videos per class is varied. Note that having one labelled video means having many labelled 16 frame blocks. We can see that for the case of very few training examples, unsupervised learning gives a substantial improvement. For example, for UCF-101, the performance improves from 29.6% to 34.3% when training on only one labelled video. As the size of the labelled dataset grows, the improvement becomes smaller. Even for
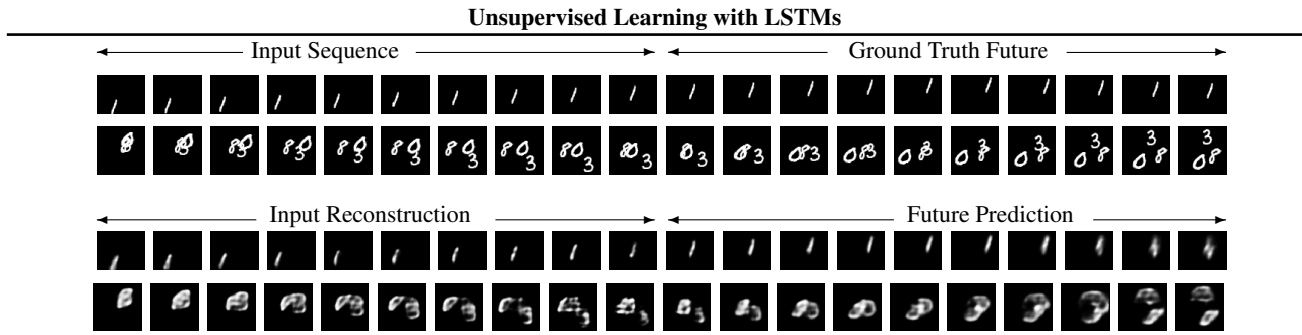
Figure 8. Out-of-domain runs. Reconstruction and Future prediction for test sequences of one and three moving digits. The model was trained on sequences of two moving digits.
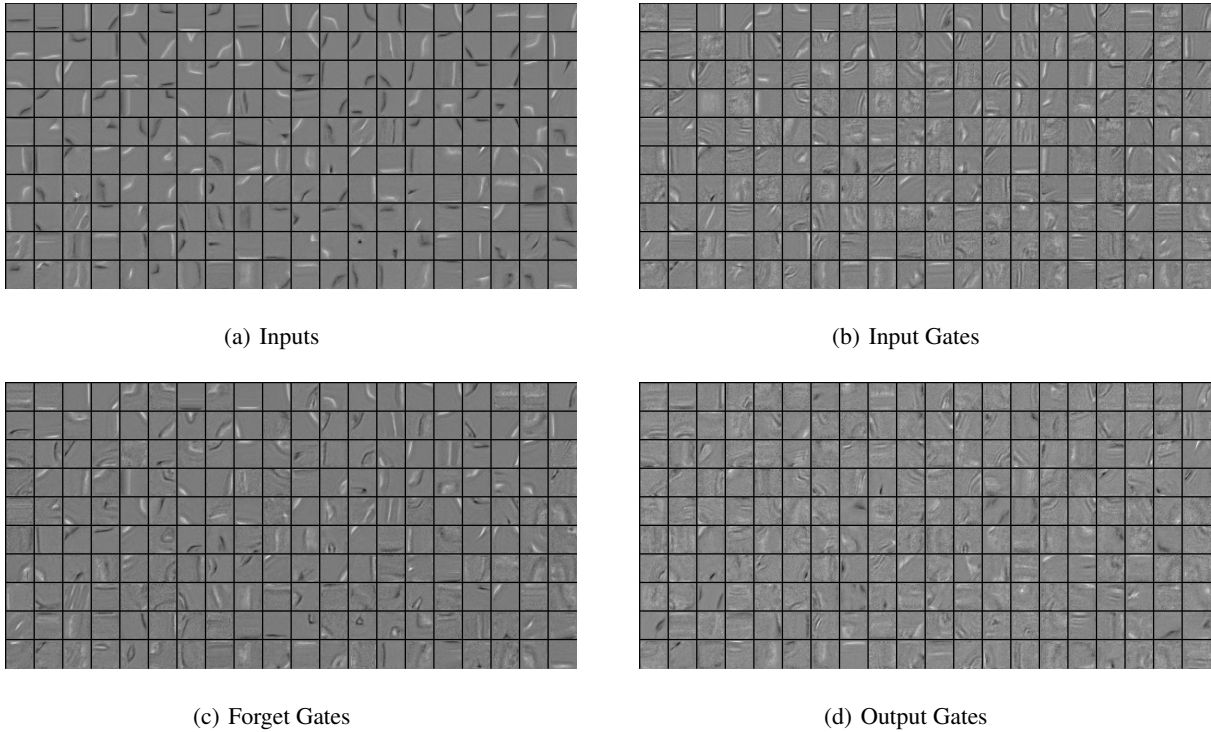

(a) Inputs

(b) Input Gates

(c) Forget Gates

(d) Output Gates

Figure 9. Input features from a Composite Model trained on moving MNIST digits. In an LSTM, each input frame is connected to four sets of units - the input, the input gate, forget gate and output gate. These figures show the top-200 features ordered by L_2 norm of the input features. The features in corresponding locations belong to the same LSTM unit.

(a) Input Reconstruction
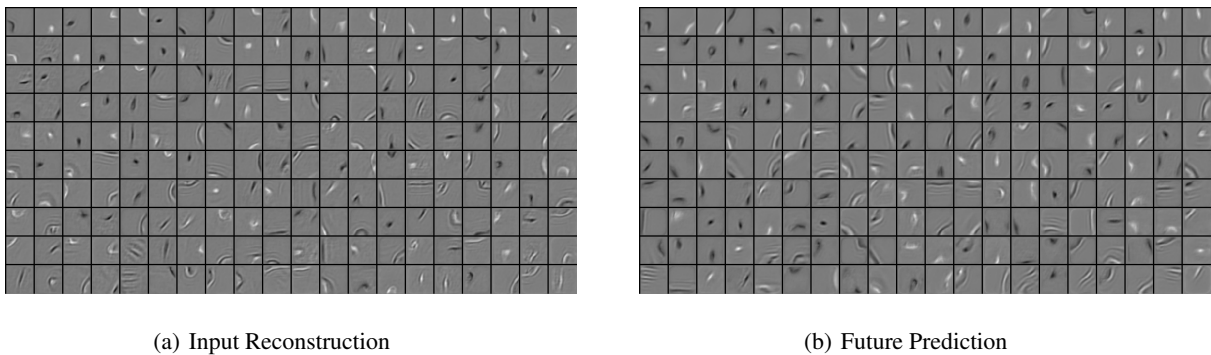
(b) Future Prediction

Figure 10. Output features from the two decoder LSTMs of a Composite Model trained on moving MNIST digits. These figures show the top-200 features ordered by L_2 norm.
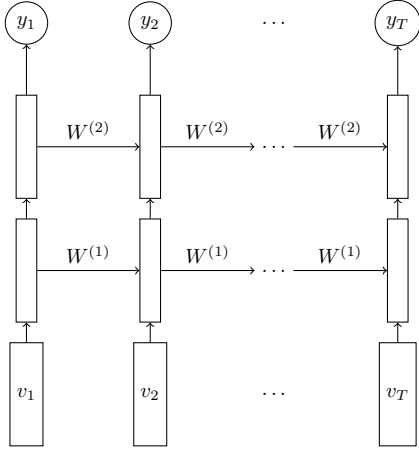
Figure 11. LSTM Classifier.

Model	UCF-101 RGB	UCF-101 1- frame flow	HMDB-51 RGB
Single Frame	72.2	72.2	40.1
LSTM classifier	74.5	74.3	42.8
Composite LSTM Model + Finetuning	**75.8**	**74.6**	**44.1**

Table 1. Summary of Results on Action Recognition.

the full UCF-101 dataset we still get a considerable improvement from 74.5% to 75.8%. On HMDB-51, the improvement is from 42.8% to 44.0% for the full dataset (70 videos per class) and 14.4% to 19.1% for one video per class. Although, the improvement in classification by using unsupervised learning was not as big as we expected, we still managed to yield an additional improvement over a strong baseline. We discuss some avenues for improvements later.

We further ran similar experiments on the optical flow data extracted from the UCF-101 dataset. A Temporal stream convolutional net, similar to the one proposed by Simonyan & Zisserman (2014b), was trained on single frame optical flows and stacks of 10 optical flows. This gave an accuracy of 72.2% and 77.5% respectively. LSTMs with 128 hidden units trained on sequences of 4 frames improved the accuracy by 2.1% to 74.3% for the single frame case. Bigger LSTMs with more frames did not improve results. By pretraining the LSTM, we were able to further improve the classification to 74.6%. Similarly for stacks of 10 frames we improved slightly to 77.7%. These results are summarized in Table 1.

3.4. Comparison of Different Model Variants

The aim of this set of experiments is to compare the different variants of the model proposed in this paper. Since it is always possible to get lower reconstruction error by copying the inputs, we cannot use input reconstruction error as a measure of how good a model is doing. However,

Model	Cross Entropy on MNIST	Squared loss on image patches
Future Predictor	350.2	225.2
Composite Model	344.9	210.7
Conditional Future Predictor	343.5	221.3
Composite Model with Conditional Future Predictor	341.2	208.1

Table 2. Future prediction results on MNIST and image patches. All models use 2 layers of LSTMs.

we can use the error in predicting the future as a reasonable measure of how good the model is doing. Besides, we can use the performance on supervised tasks as a proxy for how good the unsupervised model is doing. In this section, we present results from these two analyses.

Future prediction results are summarized in Table 2. For MNIST we compute the cross entropy of the predictions with respect to the ground truth, both of which are 64 × 64 patches. For natural image patches, we compute the squared loss. We see that the Composite Model always does a better job of predicting the future compared to the Future Predictor. This indicates that having the autoencoder along with the future predictor to force the model to remember more about the inputs actually helps predict the future better. Next, we can compare each model with its conditional variant. Here, we find that the conditional models perform better, as was also noted in Fig. 5.

Next, we compare the models using performance on a supervised task. Table 3 shows the performance on action recognition achieved by finetuning different unsupervised learning models. Besides running the experiments on the full UCF-101 and HMDB-51 datasets, we also ran the experiments on small subsets of these to better highlight the case where we have very few training examples. We find that all unsupervised models improve over the baseline LSTM which is itself well-regularized by using dropout. The Autoencoder model seems to perform consistently better than the Future Predictor. The Composite model which combines the two does better than either one alone. Conditioning on the generated inputs does not seem to give a clear advantage over not doing so. The Composite Model with a conditional future predictor works the best, although its performance is almost same as that of the Composite Model.

3.5. Comparison with Other Action Recognition Benchmarks

Finally, we compare our models to the state-of-the-art action recognition results. The performance is summarized in Table 4. The table is divided into three sets. The first set compares models that use only RGB data (single or multiple frames). The second set compares models that use
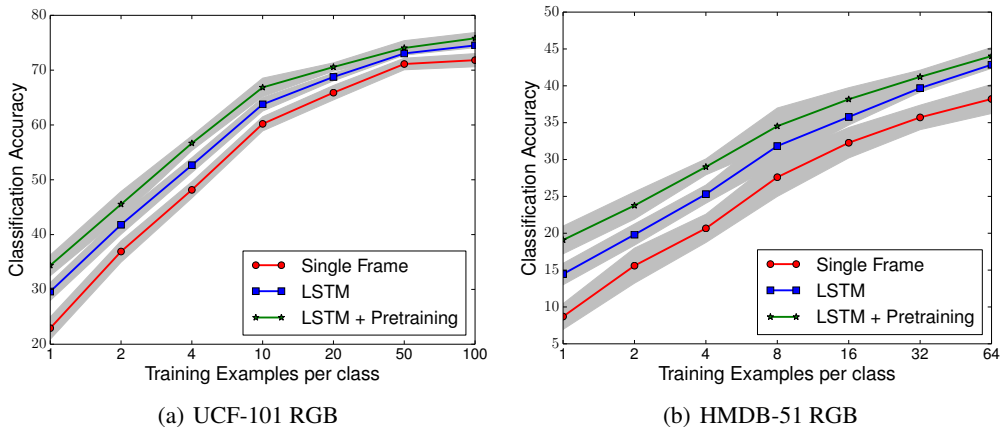
(a) UCF-101 RGB

(b) HMDB-51 RGB

Figure 12. Effect of pretraining on action recognition with change in the size of the labelled training set. The error bars are over 10 different samples of training sets.

Method	UCF-101 small	UCF-101	HMDB-51 small	HMDB-51
Baseline LSTM	63.7	74.5	25.3	42.8
Autoencoder	66.2	75.1	28.6	44.0
Future Predictor	64.9	74.9	27.3	43.1
Conditional Autoencoder	65.8	74.8	27.9	43.1
Conditional Future Predictor	65.1	74.9	27.4	43.4
Composite Model	67.0	**75.8**	29.1	**44.1**
Composite Model with Conditional Future Predictor	**67.1**	**75.8**	**29.2**	44.0

Table 3. Comparison of different unsupervised pretraining methods. UCF-101 small is a subset containing 10 videos per class. HMDB-51 small contains 4 videos per class.

explicitly computed flow features only. Models in the third set use both.

On RGB data, our model performs at par with the best deep models. It performs 3% better than the LRCN model that also used LSTMs on top of convnet features[1]. Our model performs better than C3D features that use a 3D convolutional net. However, when the C3D features are concatenated with fc6 percepts, they do slightly better than our model.

The improvement for flow features over using a randomly initialized LSTM network is quite small. We believe this is atleast partly due to the fact that the flow percepts already capture a lot of the motion information that the LSTM would otherwise discover.

When we combine predictions from the RGB and flow models, we obtain 84.3 accuracy on UCF-101. We believe further improvements can be made by running the model over different patch locations and mirroring the patches. Also, our model can be applied deeper inside the convnet instead of just at the top-level. That can potentially lead to further improvements. In this paper, we focus on showing that unsupervised training helps consistently across both datasets and across different sized training sets.

---

[1]However, the improvement is only partially from unsupervised learning, since we used a better convnet model.

Method	UCF-101	HMDB-51
Spatial Convolutional Net (Simonyan & Zisserman, 2014a)	73.0	40.5
C3D (Tran et al., 2014)	72.3	-
C3D + fc6 (Tran et al., 2014)	**76.4**	-
LRCN (Donahue et al., 2014)	71.1	-
Composite LSTM Model	75.8	44.0
Temporal Convolutional Net (Simonyan & Zisserman, 2014a)	**83.7**	54.6
LRCN (Donahue et al., 2014)	77.0	-
Composite LSTM Model	77.7	-
LRCN (Donahue et al., 2014)	82.9	-
Two-stream Convolutional Net (Simonyan & Zisserman, 2014a)	88.0	59.4
Multi-skip feature stacking (Lan et al., 2014)	**89.1**	**65.1**
Composite LSTM Model	84.3	-

Table 4. Comparison with state-of-the-art action recognition models.

4. Conclusions

We proposed models based on LSTMs that can learn good video representations. We compared them and analyzed their properties through visualizations. Moreover, we managed to get an improvement on supervised tasks. The best performing model was the Composite Model that combined an autoencoder and a future predictor. Conditioning on generated outputs did not have a significant impact on the

performance for supervised tasks, however it made the future predictions look slightly better. The model was able to persistently generate motion well beyond the time scales it was trained for. However, it lost the precise object features rapidly after the training time scale. The features at the input and output layers were found to have some interesting properties.

To further get improvements for supervised tasks, we believe that the model can be extended by applying it convolutionally across patches of the video and stacking multiple layers of such models. Applying this model in the lower layers of a convolutional net could help extract motion information that would otherwise be lost across max-pooling layers. In our future work, we plan to build models based on these autoencoders from the bottom up instead of applying them only to percepts.

Acknowledgments

References

Cho, Kyunghyun, van Merrienboer, Bart, Gülçehre, Çaglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pp. 1724–1734, 2014.

Donahue, Jeff, Hendricks, Lisa Anne, Guadarrama, Sergio, Rohrbach, Marcus, Venugopalan, Subhashini, Saenko, Kate, and Darrell, Trevor. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

Graves, Alex. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

Graves, Alex and Jaitly, Navdeep. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1764–1772, 2014.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Hurri, Jarmo and Hyvärinen, Aapo. Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691, 2003.

Ji, Shuiwang, Xu, Wei, Yang, Ming, and Yu, Kai. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35 (1):221–231, Jan 2013.

Karpathy, Andrej, Toderici, George, Shetty, Sanketh, Leung, Thomas, Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

Lan, Zhen-Zhong, Lin, Ming, Li, Xuanchong, Hauptmann, Alexander G., and Raj, Bhiksha. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. *CoRR*, abs/1411.6660, 2014.

Le, Q. V., Zou, W., Yeung, S. Y., and Ng, A. Y. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

Memisevic, Roland. Learning to relate images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8): 1829–1846, 2013.

Memisevic, Roland and Hinton, Geoffrey E. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, June 2010.

Michalski, Vincent, Memisevic, Roland, and Konda, Kishore. Modeling deep temporal dependencies with recurrent grammar cells. In *Advances in Neural Information Processing Systems 27*, pp. 1925–1933. Curran Associates, Inc., 2014.

Ranzato, Marc'Aurelio, Szlam, Arthur, Bruna, Joan, Mathieu, Michaël, Collobert, Ronan, and Chopra, Sumit. Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, abs/1412.6604, 2014.

Simonyan, K. and Zisserman, A. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014a.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014b.

Soomro, k., Roshan Zamir, A., and Shah, M. UCF101: A dataset of 101 human actions classes from videos in the wild. In *CRCV-TR-12-01*, 2012.

Susskind, J., Memisevic, R., Hinton, G., and Pollefeys, M. Modeling the joint density of two images under a variety of transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. 2014.

Tran, Du, Bourdev, Lubomir D., Fergus, Rob, Torresani, Lorenzo, and Paluri, Manohar. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

van Hateren, J. H. and Ruderman, D. L. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings. Biological sciences / The Royal Society*, 265(1412):2315–2320, 1998.

Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.