A Torch Library for Action Recognition and Detection Using CNNs and LSTMs

Gary Thung and Helen Jiang
Stanford University
{gthung, helennn}@stanford.edu

Abstract

It is very common now to see deep neural networks being applied to all sorts of image classification tasks. Video classification tasks are also now becoming more popular, as it is the logical next step to try following image classification.

We write a Torch library for a recurrent convolutional neural network architecture for large-scale video action recognition and detection; the short for Long-term Recurrent Convolutional Neural Network (LRCN) features a long short-term memory (LSTM) on top of a convolutional neural network (CNN) and can be trained with limited training data and handles complex target concepts well.

1. Introduction

The problem of action recognition can be explained as taking some amount of sequential inputs and outputting a single classification. We will be investigating action recognition and detection using the LRCN, a combination of a CNN and LSTM.

The LRCN is a recently researched architecture for visual recognition and description which combines convolutional layers and long-range temporal recursion and is trainable from start to finish [2]. This LRCN model is advantageous for many reasons. For example, they can directly map variable-length inputs like video frames to variable length outputs[2], something that simple RNN's and CNN's cannot do but is extremely useful for tasks like video description. Simple recurrent neural network (RNN) models also have a significant limitation, which is the vanishing gradient effect. This is fixed by using an LSTM[2].

The general problem for our project will be applying these artificial neural architectures to videos for action recognition. Working with videos is a difficult problem because of the additional time component that images do not have. This means that preprocessing is more complicated, and computation is much more expensive, but we can take advantage of the temporal features that come with videos.

The motivation for this project is to apply classification to a medium beyond images. Action detection has many use cases in real time: analyzing what a robot with a front-facing camera is doing, using a video camera at a sink to determine if doctors are washing their hands, and so on. Successful, accurate action detection systems that run in real time will advance the automation of many tasks.

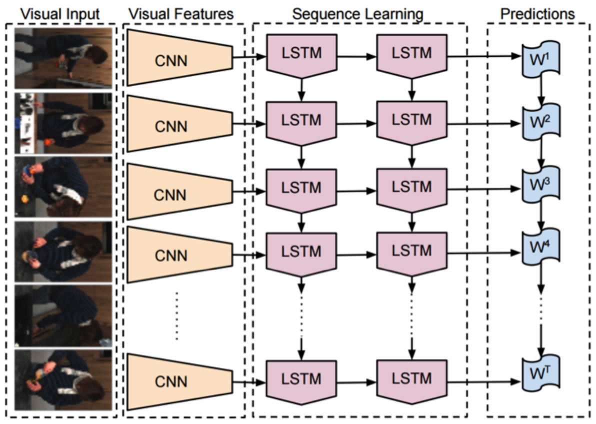

Figure 1: Long-term Recurrent Convolutional Network researched by Donahue et. al.[2]. The input is videos which have their frames extracted and fed into a CNN. The output of the CNN is then fed into the LSTM, which finally outputs the classification for each frame.

To achieve our task we will be first encoding sequential data with CNNs and using LSTMs to attain the classifications. The input to the model is videos. We encode the videos as vectors that are fed into the LSTM and a classification of each frame of the video is the output.

Action recognition is the accumulation of action detection by taking into consideration a sequence of frames and determining the overall action. Action detection can also be used to describe the main events in a video.

The main contribution of this project is to write a Torch library to perform action recognition over videos. There is an existing library written by Donahue et. al. in Caffe, but we would like to contribute to the Torch ecosystem and provide a useful tool for research and practical applications.

We are working on this project under the direction of Serena Yeung, who is part of the Stanford Vision Lab.

2. Related Work

There has been a lot of previous work with large-scale video classification[2][3][7]. Many people have explored activity recognition tasks using deep learning as well, specifically with RNN's [9][10]. LSTM's have also been shown to be very successful, especially for tasks like object recognition[1][11].

As mentioned earlier, we will be creating our model based on Donahue et. al.'s LRCN model, which builds upon these previous works. Donahue et. al.'s architecture works for action recognition, detection, and description tasks. However, we decided to only focus on action recognition and detection. Due to time and resource constraints, our model and testing will be focused on action recognition. Their library is written in Caffe, whereas ours is in Torch.

Donahue et. al. proposed the LRCN model in their paper, stating that the LRCN has many advantages for video-related tasks over the basic RNN or CNN model. Their model passes each visual input through a feature transformation to produce a fixed-length vector representation. They then allow the sequence model to use that fixed-length vector representation. For activity recognition, a task we will be focusing on, their input is a sequence of frames from a UCF-101 video, and the output is a classification label, e.g. "Apply Eye Makeup."

In their paper, they created two variants of the LRCN architecture. One of them is the LRCN-fc$_6$, where the LSTM is placed after the first fully connected layer of the CNN. The LRCN-fc$_7$ is the variant where the LSTM is placed after the second fully connected layer of the CNN. They train their LRCN networks with video clips of 16 frames and during testing, the model predicts the action label at each frame, and the final classification is the average of these predictions. We will be doing the same average technique during our test time to get the final classification of the input video. They pretrained their neural network on a subset of the ImageNet dataset[4]. Pretraining this net prevents overfitting and allows for faster training. Additionally, they trained their LRCN models from end to end.

They evaluated their LRCN architecture on the UCF-101 dataset[12] by comparing their two LRCN models against the baseline architecture for both RGB and flow inputs (separately).

Donahue et. al. showed that LRCN models have distinct advantages over state-of-the-art models for video recognition. For tasks like action recognition, the LRCN model is far superior. Their average accuracy over all three splits of the UCF-101 dataset using their LRCN-fc$_6$ model was 87.6%. This best model of theirs utilized two models: one taking in optical flow features and another taking in RGB frames. The output score was computed by taking a weighted average of the RGB and flow scores. This result is markedly improved from the 65.4% average split accuracy from a sole convolutional network architecture [7]. For this project, we opt not to use flow to keep our pipeline simpler.

An encoder approach for our model was modeled similar to [13]. We use [5][6] as a models for our Torch library.

3. Methods

3.1. Algorithm

Our task first requires the use of CNNs. The CNN consists of multiple layers that operate on some input to get output activations. Some of these layers are convolutional, ReLU, and pooling. Each of these layers will affect the data to extract or tune the features that the network learns. For example, the convolutional layer runs small convolutional filters on the input to learn important features. Since the network is sequential, the output of a convolutional layer may be put into a batch normalization layer which will normalize the input. Eventually, we will reach the desired output of class scores with a fully connected layer, which is usually a linear transformation. Based on these scores and a loss function, we can calculate the gradients with respect to the loss function and backpropagate these gradients throughout the network. Thus, we fine tune the parameters of the network in accordance to the training data. The detail of our architecture is shown in Table 1.

The other contributing model is the LSTM. The LSTM is an optimized variant of the basic RNN. The basic RNN takes in some sequential input and for each element in the sequence, it outputs some activations to an output and a hidden state, which is then used to inform the next element in the sequence. Therefore, the RNN performs classification for an element in a sequence by using the temporal information from the previous elements in the sequence. The LSTM is different from the RNN because the calculation of hidden states involves adding previous hidden state information, rather than multiplying it. Multiplication causes the values stored to blow up and the gradients to vanish. The LSTM also gates its operations so that it does not cram all the the information that it receives into its memory/states because not all information may be useful.

The algorithm for the LRCN requires the use of a CNN followed by an LSTM. A CNN is used to extract spatial features from the video and then encode this as a sequence. The sequential encoding is then given to an LSTM to attain the class scores for each part of the video. At this point in the algorithm, we have completed action detection: fine grained classification of a video. To perform action recognition, we would use the scores for a video's frames and average the scores and take the max score, which would be the classification for the entire video.

3.2. Machines/Framework

When building toy networks to familiarize ourselves with Lua and Torch we trained them on our own machines. We are now training our networks on Amazon Web Services EC2 for access to CUDA-enabled GPUs for faster training and results. Using EC2 in our testing gives us a realistic environment for our development and testing of the Torch library.

We are using Torch as the framework for our entire project. Torch is quite nice because it is relatively flexible and has a growing number of libraries for us to use and draw inspiration from. Implementing our framework in Torch will be very helpful for research development and will be quite useful for comparisons on various other lab projects.

As stated previously, our main goal for this project is to create an elegant Torch library for action recognition via LRCNs. Ideally, this library would allow users to tweak LRCNs for action recognition on video datasets.

Additionally, we used instructor Justin Johnson's `torch-rnn` module [5] for the implementation of the LSTM. His module provides efficient, easy-to-use RNN and LSTM modules that subclass the Torch `nn` library, therefore allowing us to easily integrate them into our overall network architecture.

3.3. Network Architecture

We experimented with multiple network architectures, but we will be focusing on the LCRN model introduced by Donahue et. al.[2]. The LRCN as a whole is relatively straightforward and consists of a CNN followed by an LSTM. The CNN we are using is fairly deep with many filters so as to detect the various features of videos and account for such complications. However, we expect the CNN will not be much different than a CNN for image classification because it is essentially the same task. Where the two differ is using the temporal information of a video in the LSTM.

The CNN architecture we use (see: Table 1) very closely resembles AlexNet [8], as does the CNN used by Donahue et. al. However, we had to scale down our CNN to meet the memory constraints of the NVIDIA GRID K520 GPU on the EC2 g2 instance. In our CNN, the convolutional layers, on average, have about half of the amount of hidden layers as AlexNet, and the first fully connected layer has less than 10% of the number of hidden layers in AlexNet. Also, we chose not to use any pretrained models so that we could realistically debug and test the functionality of the library.

Input	$320 \times 240 \rightarrow 216 \times 216$, 3 channels
Conv	64 hidden, 7×7 kernel, preserve shape
Batchnorm	$\downarrow$
ReLU	$\downarrow$
Max Pool	2×2 kernel, 2 stride, 0 pad
Conv	96 hidden, 5×5 kernel, preserve shape
Batchnorm	$\downarrow$
ReLU	$\downarrow$
Max Pool	2×2 kernel, 2 stride, 0 pad
Conv	128 hidden, 3×3 kernel, preserve shape
ReLU	$\downarrow$
Conv	128 hidden, 3×3 kernel, preserve shape
ReLU	$\downarrow$
Conv	196 hidden, 3×3 kernel, preserve shape
ReLU	$\downarrow$
Max Pool	2×2 kernel, 2 stride, 0 pad
FC	320 hidden
ReLU	$\downarrow$
Dropout	$p = 0.5$
LSTM	256 hidden, 8 sequence length
Dropout	$p = 0.5$
FC	101 classes
Softmax	Negative Log Likelihood

Table 1: The architecture we used for the LRCN model.

4. Datasets

UCF-101 For the task of action recognition, we used the University of Central Florida's Center for Research in Computer Vision's vast UCF-101 action recognition dataset [12]. This dataset consists of short, succinct clips from YouTube depicting actions such as bowling, pullups, applying makeup, and more (examples shown in Figure 2). It contains about 13,000 videos categorized into 101 action classes. This dataset is commonly used in video related tasks, and thus a lot of work has been published on it. One thing to note is that UCF-101 does not support fine grained action detection because the labels provided are only for the entire video.

ECM^2 egocentric For the task of action detection, we would have liked to evaluate our model using the ECM^2 egocentric dataset from the Stanford Computer Vision Lab, which consists of first person video of daily tasks like walking, cooking, talking, bike riding, and more (examples shown in Figure 3). ECM^2 contains about 100,000 frames with 20 classes of physical activities. Note that there is no mention of this dataset online because it is unpublished and we had been granted access to it by the lab.

However, due to time constraints we did not get around to processing and testing this dataset.

Figure 2: These are a few examples of some frames in the UCF-01 dataset. There are 101 class labels in total, like ”Yo-Yo” or ”Baby Crawling”.

Figure 3: These are a few examples of some frames in ECM^2. There are 20 class labels in total, like ”Biking”.

Evaluating on the ECM^2 would be a good exploration because it would be insightful to test and analyze the performance differences between the two datasets on our model, and whether this is related to the differing perspectives of first and third person point of view.

4.1. Pipeline

We used one of the given UCF-101 training and testing splits, which has around 9500 train videos and 3700 test videos. The only feature we use is the raw RGB values.

The UCF-101 videos come in resolution 320 $\times$ 240 and run at 25 frames per second (FPS). The videos vary from 1 second to about 70 seconds, with the average clip being about 7 seconds.

To allow maximum generalization in our Torch library, the pipeline first uses `ffmpeg` to convert all of the input videos to 4-dimensional Torch tensors at the FPS desired by the user. This gives us some amount of frames for each video, and depending on the user's desired sequence length we skip videos that were not long enough to generate the required number of frames. We then segment the number of frames for each video into semi-equal blocks of size (# of frames / sequence length). We select and save a random frame from each block to capture the data in that block of the video.

When it comes to training and testing time, we load the videos in batches and scale them down to the desired dimensions. We also calculate the mean image of the scaled training set to perform mean subtraction as preprocessing for the training and testing sets. The videos represented as sequences of frames are then batch fed into the CNN to be encoded as vectors. These encoded vectors will be input into the LSTM and class scores will be generated. To perform action recognition, we take the maximum of the average scores.

We chose this order of operations to allow easy tuning of the scaled resolution of the videos. There is no best choice in the order of operations because there are tradeoffs in memory and time. For example, the video frames must be dumped again if the user wants to downsample the videos to a different FPS or change the sequence length.

In our own training and testing, we downsampled the data drastically to accommodate for the massive computational resources required. This testing with downsampling and finding usable parameters (batch size, sequence length) took a decent amount of time because working with videos added another dimension of processing resulting in many confusing errors and bugs. Ultimately, we downsampled by extracting the videos at 5 FPS, used a sequence length of 8, and reduced the video size from 320 $\times$ 240 to 216 $\times$ 216.

5. Results

5.1. Experiments

We used a single split given from UCF-101 to train and test the LRCN model. Since we are trying to perform experiments similar to Donahue et. al., we do not use a validation set.

We implemented the LRCN model to perform action recognition on videos in the Torch framework. We trained and tested our model on the UCF-101 dataset, and compare our train and test accuracy with those of Donahue et. al. However, we cannot directly compare, because our architecture is much smaller than theirs. Donahue et. al.'s LRCN model was able to achieve an average accuracy of 87.6%, compared to the baseline of 65.4% of a sole CNN [7].

Initially, we started training using stochastic gradient descent (SGD) and an aggressive learning rate of `1e-3`. We found that this had trouble learning because the learning rate was too high and the loss fluctuated heavily. After testing learning rates, we found that around `1e-6` worked decently well. Because SGD was showing relatively slow progress in the training, we implemented Adam, the generally recommended adaptive learning rate method, into our training to speed up the learning. With Adam implemented, we found

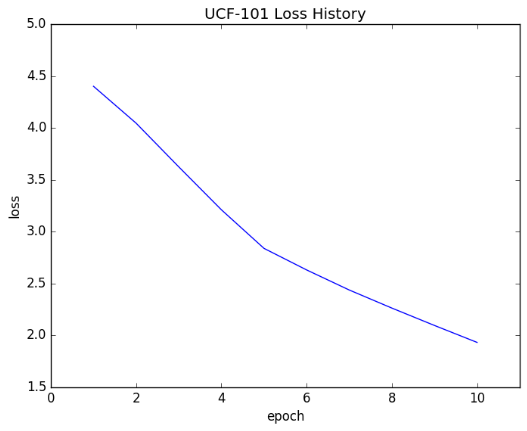

Figure 4: minibatch average Softmax loss using negative log likelihood

that the loss decreased much more than SGD after about half an epoch.

After many attempts, we settled with a learning rate of `5e-5`. Higher learning rates gave us dismal results, with much more overfitting and test accuracy rates of less than 5%. `5e-5` was fairly aggressive to begin with, and with step decay after every 5 epochs, this worked well. We chose to decay the learning rate in half. Given the amount of data and progress we saw, it may have been beneficial to decay less frequently to allow the system to reach a better position. With more time, we would have liked to tune the learning rate decay as well. We chose a batch size of 2 videos and a sequence length of 8 frames. Due to the LRCN taking up nearly all of the video memory of the GPU, we were limited in choosing those two hyperparameters, as well as the scaled resolution of the videos. Sometimes even increasing the sequence length of 1 caused the program to crash due to running out of GPU memory.

Hyperparameter tuning was not rigorously performed due to time constraints. Working with video presented unanticipated hurdles in regards to data processing and training. Training a single epoch required slightly over an hour, computing the training accuracy required slightly over half an hour, and computing the testing accuracy required about 10 minutes. Along with that, dumping the video frames if the output FPS or sequence length changed was not a trivial amount of time, and the parameters must be chosen to fit on the GPU used for training.

5.2. Accuracy

Currently, with a learning rate of `5e-5` and 10 epochs, we were able to achieve a loss of 1.93 (calculated per frame), a training accuracy of 43.9%, and a test accuracy of 23.6% with our LRCN model on the UCF-101 dataset (see: Table 2). Although the UCF-101 dataset was not designed for fine-grained action detection, we report an action detection accuracy of 20.2%. The detection accuracy is the frame level accuracy, which can be lower than the recognition accuracy because the recognition accuracy for a video averages the detection scores.

While we are not at the level of accuracies that Donahue et. al.'s LRCN model achieves, we have been consistently raising our accuracy levels every time we run. Due to time and model constraints as mentioned before, we were not able to achieve the accuracy values as Donahue et. al. Given that our network is a size considerably smaller than Donahue et. al.'s and was trained for a relatively short time, we feel our results show good promise of the model, and that our Torch library works.

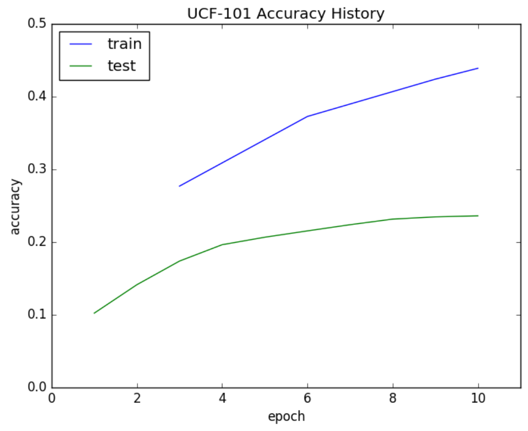

Figure 5: Action recognition accuracy. The training curve only has 3 points because the training accuracy was computed every 3 epochs to save time.

Model	Accuracy on UCF-101
Donahue et. al. LRCN-fc$_6$	87.6%
Our LRCN	**23.6%**

Table 2: Comparison of accuracy values on the UCF-101 dataset.

5.3. Analysis

Seeing our training accuracy of 43.9% and test accuracy of 23.6%, it is clear to see that our model is overfitting. As Figure 5 shows, as each epoch continues, our model overfits even more. This overfitting is something we have noticed multiple times while testing our model, and we have been overfitting less every time, though it is not enough to completely get rid of the problem. We attempted to mitigate the overfitting problem by tuning the dropout rate. Donahue et.

al. pretrains their network on a dataset that is not UCF-101, which helps prevent overfitting. We are considering doing this too.

As shown in Figure 5, our training accuracy did not plateau, even after running our model for 10 epochs but the testing accuracy did plateau slightly. The training accuracy line was still steadily increasing. The model could definitely benefit from training for more epochs to increase the accuracy. More hyperparameter tuning can be performed as well, like finding a good mix of sequence length, scaled resolution, and learning rate.

By examining the individual videos where the model succeeded and failed, we can analyze the model's pitfalls. Let's take a look at the YoYo video class, which consists of people performing tricks with a yo-yo in front of some background.

In some cases, especially in videos of yo-yo competitions, the yo-yo(s) can be moving very quickly. In other cases, the subject is not as good and performs their tricks relatively slow. The model succeeded in a few such cases, where the yo-yo moved fairly slow.

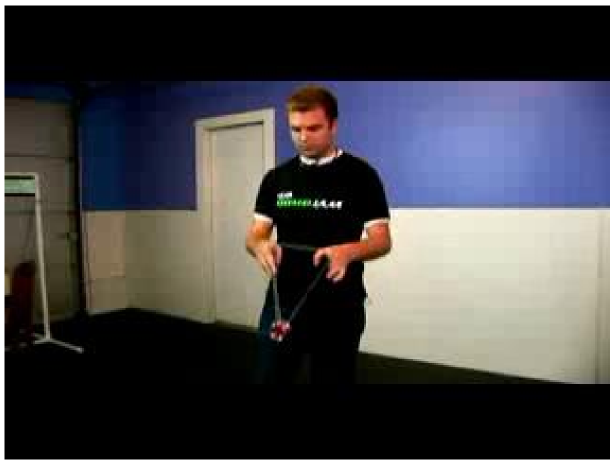

Figure 6: frame part of a successful YoYo video classification

In these frames, we can see that the yo-yo has been captured clearly because the subject was not performing tricks at a quick speed (see: Figures 6 & 7). The yo-yo is an important feature so capturing it clearly is of utmost importance for extracting features.

In failed classifications, (see: Figure 8) we can see that the yo-yo moves very quickly and looks more like a blur than a circular object attached to a string. This could be remedied by extracting frames at a higher FPS or using a larger network to learn more features as well as training for longer.

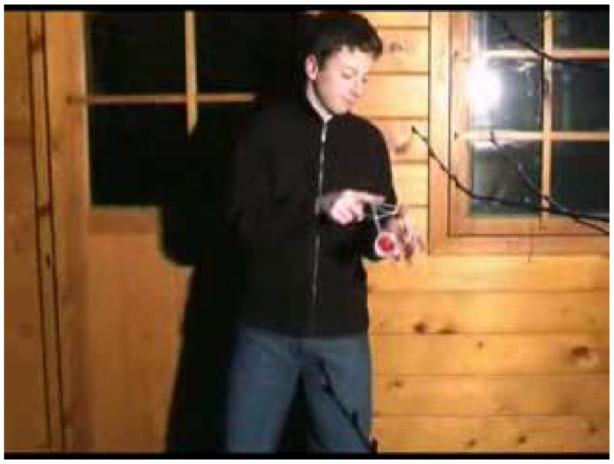

Figure 7: frame part of a successful Yo-Yo video classification

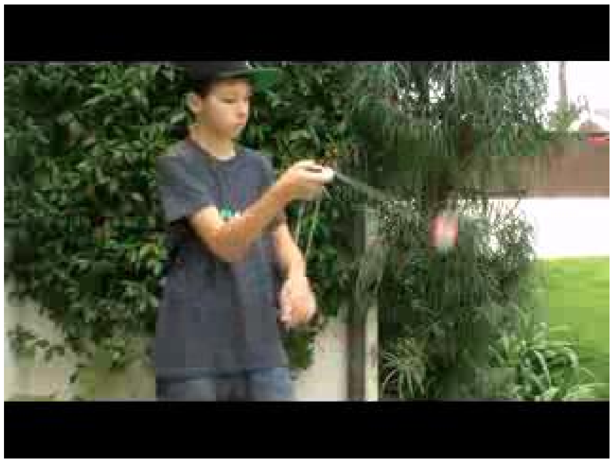

Figure 8: frame part of an unsuccessful YoYo video classification

6. Conclusion

We were able to train and test the LRCN model on the complete UCF-101 dataset, and produce relatively good results with the comparatively shallow network that we have. Given more time, we could have partitioned some of the training set for a validation set and fine-tuned our network to achieve better results since we know the model is capable of achieving good results.

A lot of focus and effort was directed towards writing the Torch library, which is functional and relatively easy to use. At the absolute minimum, the user needs three text files consisting of video paths and their labels for training, validation, and testing splits, as well as the native resolution of the videos and the desired scaled resolution. Since action recognition is not a trivial task in training time and successful computation, the user will still need to perform hyperparameter tuning to find values that work for their needs.

6.1. Future Work

As for future work, there are many things we want to try. First, we did not have time to use the ECM^2 dataset. That would be our next step because it would be very interesting to see how our network applies to a set of videos from a first-person perspective. Exploring first-person video has many applications, like body cameras on police officers or analyzing actions of robots with front-facing cameras.

If we had more computational resources, we would more rigorously fine-tune our hyperparameters with a random search and use a larger architecture to match the architecture that Donahue et. al. used when evaluating the LRCN model on UCF-101.

Our next step would be to make our Torch library completely fleshed out, allowing for easy configuration of the CNN. As it stands now, the library is easy to use with little user impediment, but we want to expand on this because the user can only easily tune the parameters of the AlexNet-like CNN. Nonetheless, it requires a little bit of user effort to change the CNN architecture.

We would like to try different data extraction methods. It would be optimal to extract the largest sequence possible from each video to capture maximum information. One method we would like to try is capturing multiple sequences of frames from each video with some stride between the start of each sequence and then using these multiple sequences to generate classifications.

7. Acknowledgments

We would like to thank Serena Yeung for giving us advice throughout the project. We would also like to acknowledge Katsuyuki Nakamura for providing us access to the ECM^2 egocentric dataset. Lastly, we would like to thank the CS 231N staff for granting funds to use Amazon Web Services for the computation and for a great class.

Find the public repository for the Torch library at `https://github.com/garythung/torch-lrcn`.

References

[1] I. S. A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. Technical report, 2012.

[2] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[3] C.-W. C. J. C. Niebles and L. Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. Technical report, 2010.

[4] R. S. L.-J. L. K. L. J. Deng, W. Dong and L. FeiFei. Imagenet: A large-scale hierarchical image database., 2009.

[5] J. Johnson. torch-rnn. `https://github.com/jcjohnson/torch-rnn`, 2016.

[6] A. Karpathy. Multi-layer recurrent neural networks (lstm, gru, rnn) for character-level language models in torch. `https://github.com/karpathy/char-rnn`.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[9] C. W. C. G. M. Baccouche, F. Mamalet and A. Baskurt. Action classification in soccer videos with long short-term memory recurrent neural networks. Technical report, 2010.

[10] C. W. C. G. M. Baccouche, F. Mamalet and A. Baskurt. Sequential deep learning for human action recognition. Technical report, 2011.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. Technical report, 2014.

[12] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human action classes from videos in the wild. Technical report, University of Central Florida, November 2012.

[13] I. S. W. Zaremba and O. Vinyals. Recurrent neural network regularization. Technical report, 2014.