DL Project Report

Sanchit Agrawal, Ishu Dharmendra Garg

April 2017

1 Introduction

Our task is to come up with an effective (and practically feasible) method for video classification, specific to the context of the YouTube video classification challenge.

2 Literature Survey

Our initial literature survey consisted of the following papers:

- 3D Convolutional Neural Networks for Human Action Recognition
- Large scale Video Classification with Convolutional Neural Networks
- Two-Stream Convolutional Networks for Action Recognition in Videos
- Beyond Short Snippets: Deep Networks for Video Classification
- Delving Deeper into Convolutional Networks for Learning Video Representations
- Modeling Spatial-Temporal Clues in a Hybrid Deep Learning Framework for Video Classification
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description
- Action Recognition with Trajectory-Pooled Deep-Convolutional Descriptors
- Actions $\sim$ Transformations
- Learning End-to-end Video Classification with Rank-Pooling
- Exploiting Image-trained CNN Architectures for Unconstrained Video Classification
- Unsupervised Learning of Video Representations using LSTMs)

- Deep Learning for Video Classification and Captioning

Modifications of some of the ideas presented in these papers have been incorporated in our work. However, we would like to point out that most of these papers assume that the raw videos are available, hence they are not directly applicable to our task.

3 Dataset

We use the YouTube-8M dataset consisting of around 8 million video features with multi-labels from a set of 4800 visual entities.

To generate the features, frames from the video are sampled at a rate of one frame per second. The features of last layer (name: pool_3/_reshape) of inception net are extracted for each frame. These are reduced using PCA and normalized.

4 Challenges

Our project poses some unique challenges listed below:

- The features given to us are very abstract. Most of the current methods of video classification used today can not be used they use optical flow or other features which require raw videos.
- Data size is enormous - frame level features take around 1.7 TB, leading to data management challenges.
- Large (and possibly expensive) computation resource is needed.
- Even with decent computing power, it takes very long for for a model to train for a single epoch.
- The dataset is noisy. Labels can be missing or there can be wrong labels associated to a video.
- The number of frames per video can be up to 300, which is hard for naive sequence models to capture.

5 Models Implemented

We implemented several models after considering the salient features of the problem. Some of them are described in this section.

5.1 Baseline Models

The baseline models were not implemented by us. They were present in the starter code.

5.1.1 Logistic Regression

The frame features are averaged and fed to a logistic regression model, which predicts the class probabilities.

5.1.2 Mixture of Experts

Several logsitic models are trained on the data, and their ensemble is used to decide class probabilities.

5.1.3 LSTM Model

The individual frame features are fed to an LSTM, and the final state is fed to logistic regression.

5.2 Our Models

The following models were implemented by us. Needless to say, they have several further design choices (like the non-linearities, methods of normalization etc.). While we have experimented with these, we do not list them here for the benefit of the reader's sanity.

5.2.1 Hierarchical LSTM

Motivation: To cope with the task of managing long sequences, we let one LSTM summarize local temporal information, while another LSTM summarizes global video information.
Implementation: We partition the frames into buckets of equal size. An LSTM is run on each bucket, and the final state is used as the summary of the bucket. The state obtained from each bucket is further given as input to a second level LSTM, which aggregates them and feeds the final output to logistic regression.

5.2.2 Poor-None Hierarchical Attention LSTM

Motivation: Since some frames are likely to be more informative than others, we add attention to the model.
Implementation: An LSTM predicts a scalar $c_i \in [0, 1]$ value denoting the importance of frame i. The frame features are multiplied with the importance scalars and fed to the hierarchical LSTM model.

5.2.3 Poor-Poor Hierarchical Attention LSTM

Motivation: The attention can also be extended to decide which video sections are more important than others.
Implementation: This model further augments the Poor-None Hierarchical Attention LSTM by also predicting importance weights at the second (bucket) level.

5.2.4 Poor-Rich Hierarchical Attention LSTM

Motivation: The attention scores for video sections need not be "fixed". At different time steps the importance of video sections may change depending on what we are looking for.
Implementation: This model further augments the Poor-Poor Hierarchical Attention LSTM by making the attention weights at the second (bucket) level a function of the LSTM state. The bucket attention weights need to be recomputed at each time step, leading to the name "rich attention".

5.2.5 Rich-Rich Hierarchical Attention LSTM

Motivation: The attention scores for frames need not be "fixed". At different time steps the importance of frames may change depending on what we are looking for.
Implementation: This model further augments the Poor-Rich Hierarchical Attention LSTM by using rich attention at the first (frame) level too.

5.2.6 Poor-Poor Hierarchical Feature Attention LSTM

Motivation: A frame may be important only due to the presence or absence of certain features, and may not be important overall. Hence we should focus attention at the level of features instead of frames.
Implementation: This model uses two levels of poor attention, but instead of a single scalar for a frame, attention weights are computed for each frame feature.

5.2.7 Poor-Poor Hierarchical Feature Summation

Motivation: We already know that the mean of the frame features works well in practice. We augment the simplicity of this idea by selectively deciding which features in which frames are important, and add them up.
Implementation: This model computes feature attention weights for each frame in a bucket, and the after multiplying them with the frame vectors, simply adds them up to produce the bucket vector. Then feature attention weights are computed for each bucket vector, and after multiplying with the bucket vectors, they are added to produce the final video summary vector.

5.2.8 Window Detection Model

Motivation: Short durations may be sufficient to detect an activity. If an activity is present in any section of the video, we should detect it and assign that label to the video.
Implementation: We ask an LSTM to predict the labels for each bucket. These probabilities are taken and max-pooled over all the buckets.

5.2.9 Hierarchical Window Detection Model

Motivation: An activity may be decomposed into several micro-activities. At each time-step within a bucket, we try to predict if a local micro-activity has occurred. Using these, we try to predict the presence of activities across buckets.
Implementation: Within a bucket, at each time-step, an LSTM outputs a vector with elements lying in $[0, 1]$. We interpret this as a micro-activity probability vector. These are max-pooled across time steps, and the aggregate acts as the summary of the bucket. This is fed to the second level LSTM, which makes activity level predictions at each step. These are max-pooled across all buckets to obtain the class probabilities.

5.2.10 Hierarchical Residual LSTM

Motivation: Both Hierarchical LSTMs and simple averaging of features works well. We should somehow combine their simplicity and power. Moreover, empirically we know that an LSTM is unable to perform simple averaging for this particular task.
Implementation: We add a residual connection that adds the averaged frame vectors to the output of the Hierarchical LSTM.

5.2.11 Double Hierarchical Residual LSTM

Motivation: Same as above.
Implementation: We extend Hierarchical Residual LSTMs by adding residual connections that adds the averaged bucket frame vectors to the output of the first level (bucket) LSTM.

5.2.12 Convolution Model

Motivation: This idea is not original. Several papers have performed convolutions over time.

5.2.13 Dilated Convolution Model

Motivation: Inspired from WaveNet, we might be able to perform convolutions cheaply, while still retaining much of the benefit.

6 Experiments & Results

We have the following observations to make regarding the experiments & results obtained so far:

- The model that has worked best so far is the Hierarchical Residual LSTM, giving 0.8 GAP.

- Surprisingly, more complex attention mechanisms perform poorly. Models with no attention perform better. This might be because it takes a very long time to tune so many parameters.
- Convolutional models are extremely slow and consume a lot of memory, hence impractical in the context of the contest.
- For some models, the GAP score is often very low for a few thousand steps, and suddenly shoots up. Thus we need to maintain a patience parameter while trying a new model, before we discard it.
- Adding audio features provides a significant boost to the models.

7 Future Work

The following are the ideas we plan to work on

- Explore practical CNN based models.
- Make ensemble models using models we have already trained.
- Use the class hierarchy present in the dataset to train models for each class cluster.
- Tune hyper-parameters of the current models.
- Utilize both frame level and video level features in the same model. This can be done along the lines of the successful residual model.
- We can use the difference in consecutive frames as a proxy for optical flow features.
- Retry the complex attention models with a large patience parameter.