YouTube-8M Video Classification

Alexandre Gauthier and Haiyu Lu
Stanford University
450 Serra Mall
Stanford, CA 94305
agau@stanford.edu hylu@stanford.edu

Abstract

Convolutional Neural Networks (CNNs) have been widely applied for image classification. Encouraged by these results, we apply a CNN model to large scale video classification using the recently published YouTube-8M dataset, which contains more than 8 million videos labeled with 4800 classes. We study models with different architectures, and find best performance on a 3-layer spatial-temporal CNN ([CNN-BatchNormzliation-LeakyReLU]^3-FullyConnected). We propose that this is because this model takes both feature and frame information into account. By optimizing hyperparameters including learning rate, we achieve a Hit@1 of 79.1%, a significant improvement over the 64.5% for the best model Google trained in their initial analysis. Further improvements may be achievable by incorporating audio information into our models.

1. Introduction

Online video sharing websites are a common way for people to share their experiences with others, as well as a popular source of both entertainment and education. YouTube, the most popular online video sharing website, hosts over one billion videos. With such a large database, it can be difficult for users to find exactly the video they're looking for. Therefore, a highly efficient search mechanism is essential in order to serve the most appropriate videos to users.

It can be difficult to implement video search, as videos have little inherent text-based data that can be used for indexing. Most videos have only uploader-provided metadata (title, category and labels). Labels can be extremely useful for search indexing, but contributors cannot be counted upon to provide every label appropriate for their videos.

In order to improve search quality on YouTube, it is desirable to devise a way to automatically assign relevant labels to each video, even ones with limited user-supplied metadata. To this end, Google has released the YouTube-8M dataset [1]. This set of over 8 million labeled videos allows the public to train algorithms to learn how to associate labels with YouTube videos, helping to solve the problem of video classification.

We use multilayer convolutional neural networks to classify YouTube-8M videos. The input to our algorithm is pre-processed video data taken from individual frames of the videos. We then use a CNN to output a score for each label in the vocabulary. We briefly tried LSTM and fully-connected models, but found that convolutional models perform better.

2. Related Work

Large-scale datasets such as ImageNet [2] have played a crucial role for the rapid progress of image recognition [3, 4, 5] to near human level accuracy [6]. In a similar vein, the amount and size of video classification is growing [7]. Training data available ranges from small, well-labeled datasets like KTH [8], Hollywood2 [9] and Weizmann [10], with a few thousand video clips, to medium-scale datasets such as UCF101 [11], HMDB51 [12] and Thumos14 [13], which allow for classification into more than 50 action categories. The largest available dataset prior to the release of YouTube-8M was Sport-1M, 1 million videos labelled with 487 sports related activities [14]. These datasets have been extensively analyzed in the literature [15]

The main improvement provided by the YouTube-8M dataset is its sheer size. It includes 4,800 classes, compared to under 500 for other datasets; and it has more than 8 times the number of videos than the Sport-1M dataset. Furthermore, the YouTube-8M classes aim to describe the topic of a video (e.g. "gardening"), while older datasets merely identify objects or activities found in videos (e.g. "flower"). This was allowed by Google's development of an efficient video annotation system [16].

Making predictions from a larger number of potential classes is desirable because, in the real world, there are far more than 500 types of videos users could want to watch.

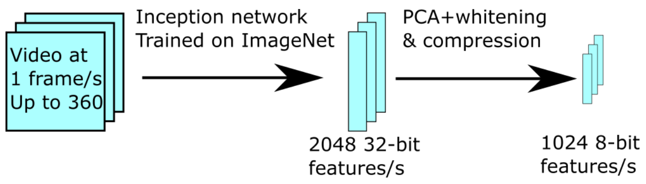

Figure 1. Schematic of preprocessing algorithm for YouTube-8M dataset. Videos are compressed down to 1024 8-bit features per second using a network pre-trained on ImageNet, followed by PCA.

Larger datasets are necessary when expanding the number of classes in order to provide a sufficient variety of training examples so that the classifier will perform well on a larger fraction of real-world videos.

3. YouTube-8M Dataset

The entire YouTube-8M dataset consists of 4,800 classes and a total of 8,264,650 videos, representing over 500,000 hours. Each video is labeled with one or more class. On average, each video has 1.8 labels. The entire dataset is split into Training, Validation and Testing partitions, with ratios of 70%, 20% and 10%.

3.1. Vocabulary Construction

Google generated labels in millions of classes with the algorithmic YouTube video annotation system [16] based on the contents and associated metadata of a selection of well-annotated videos. Then, the labels were filtered down to 4,800 classes through a combination of human raters and automated curation strategies. There were two main tenets when selecting the labels to include in the vocabulary for the label. First, every label is identifiable using visual information alone; second, each label has a sufficient number of videos in the dataset for adequate training.

3.2. Frame-level and Video-level Features

The raw video dataset is hundreds of terabytes, which is impractical to deal with. Therefore, the dataset has been pre-processed using a publically available Inception network trained on ImageNet. Entire video frame images, one per second, up to 360 per video, are sent through this network, reducing their dimensionality to 2048 features per frame. PCA with whitening is then used to further reduce the feature dimensions to 1024, which is finally compressed from 32-bit to 8-bit data types. See Figure 1.

Full audio data is also provided in one second intervals. Video-level data is computed by taking the average of the frame-level visual and auditory data over the entire video.

Training on video-level features is computationally simpler, because there are far fewer of these features per video. Although this can speed up training, a lot of information is lost in the frame-by-frame averaging process used to compute video-level data. This is especially true if individual frames in a video are dissimilar, and the label we need to predict is contained in only a small fraction of the frames. Therefore, we focus on training models which use frame-based features.

3.3. Evaluation Metrics

Our goal is to predict the labels associated with each video using image and audio data. However, because many videos are associated with more than one label, there is no single obvious way to quantify model quality. The primary evaluation metrics we used with this dataset are “Hit@k” and “PERR” (precision at equal recall rate).

To compute PERR, you first look at the ground truth labels for each video. Then you compute scores for all 4,800 possible labels using your model. If there are n ground truth labels for a particular video, then you will predict that the n highest-scoring labels are associated with that video. PERR is then the fraction of these predicted labels that are correct, averaged over a large number of videos.

To compute Hit@k, you also compute scores for each possible label for each video. You then say that the video has been successfully classified if at least one of the k highest-scoring labels is one of its ground truth labels. The Hit@k metric is then the fraction of videos in your dataset that have been successfully classified.

Two additional metrics are mean average precision (mAP) and global average precision (gAP). These metrics are a measure of the area under the precision-recall curve for a given model.

4. Approach

Google already applied some simple models to the YouTube-8M dataset in the paper they released along with the data [1]: a fully-connected model, a deep bag of frames (DBoF) model, and a multilayer LSTM. Google also trained hinge loss, logistic, and mixture-of-2-experts models on the video-level data. The performance of these models as reported by Google is detailed in Table 2.

4.1. Working on frame level features

We created our models to work with frame-level features. This allows our models to take into account how the video changes throughout its length. Google’s frame-level models exhibited slightly better performance than video-level models, suggesting that the video-level data is missing some important information.

The data for a video consists of a $1024 \times f$ matrix, where f is the number of frames. Because f can vary between videos, the data is padded with zeros up to the maximum of 360 frames. The other dimension contains the 1024 spatial features extracted from each frame using the pre-processing algorithm described earlier. For privacy purposes, we do not know which YouTube videos we are classifying; nor do we have access to the un-processed video data.

We tried training two variants of CNN models on our data: a spatial-temporal model which convolves over features and frames, and an early-fusion model which only convolves over features.

4.2. Spatial-Temporal CNN model

We feed the frame-level data into a pair of deep convolutional neural networks (CNNs). A 3-layer spatial CNN performs 1D convolution over the 1024 single-frame features, with filters equal in depth to the number of frames. A second, 3-layer temporal CNN performs 1D convolution over the frames of each video, with filters of depth 1024.

Each convolutional layer is followed by batch normalization and a leaky ReLU activation function. Batch normalization rescales our activations after each convolution layer so that they have a mean of zero and a standard deviation of one. This prevents the activations from getting too big after passing through several layers. The leaky ReLU nonlinearity outputs $f(x_{in}) = \max(x_{in}, \alpha x_{in})$, where we set α to be 0.01. Compare this to a traditional ReLU layer, which sets α to zero.

We chose to use CNNs because convolutional layers allow our model to learn relationships both between spatial features and between adjacent frames. CNNs are also useful because their filters, which slide along one dimension of our data, are much smaller than fully-connected filters. This lets us use a larger batch size without running out of memory on our GPU.

Following the CNNs, we reshape the data into a 1D matrix, and concatenate the output of the two networks. A final fully-connected layer with sigmoid activation function gives us scores for all 4,800 labels. The fully connected layer works by providing a connection between each output of the final convolutional layer, and each of the possible scores. This allows for every possible connection between inputs and outputs to be considered in a single layer.

See Figure 2 for a schematic of our model.

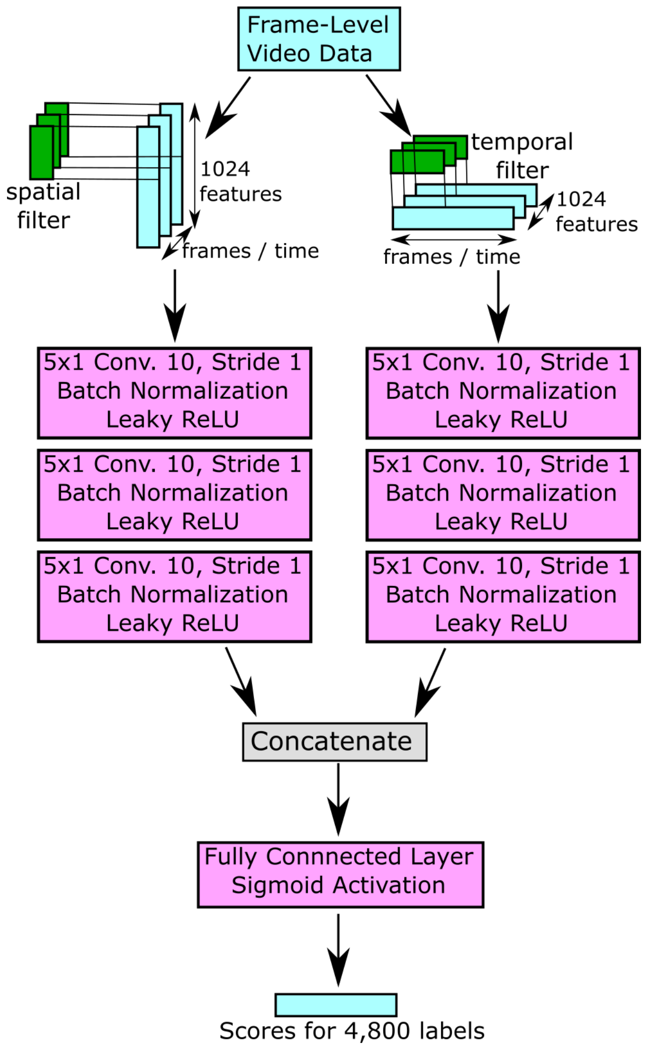

Figure 2. Schematic of our best-performing model, with two 3-layer CNNs in parallel followed by a single fully-connected layer. After tuning hyperparameters, this model achieved a hit@1 of 0.795 and PERR of 0.658.

4.3. Early-Fusion CNN model

Unlike the Spatial-Temporal CNN model, the Early-Fusion CNN model only contains a spatial CNN. In other words, the Early-Fusion CNN model only convolves over the features; it integrates over the frames of the videos. We calculate the scores for all 4,800 labels with a fully-connected layer in the same fashion as the the Spatial-Temporal CNN model.

We found that the early-fusion CNN performed about 2 percentage points worse than the spatial-temporal model. This suggests that looking at the time evolution of the video with temporal convolution results in the extraction of important information.

4.4. Training Process

We created our video classification model using the Tensorflow [17] library. Training is carried out using the Adam optimizer and a cross entropy loss function. We used mini-batches of 128 videos; this is large enough to train at a reasonable speed, yet small enough so that our GPUs don't run out of memory.

Google has provided a framework for accessing the dataset through Google Cloud ML Engine [18]. This allows us to submit classification models we have designed to be trained and evaluated on Google's servers without having to download the entire dataset (over 1 TB) to our personal

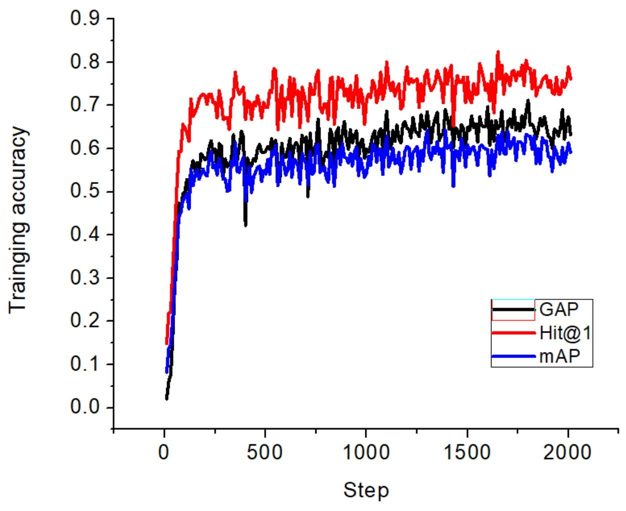

Figure 3. Training process of the 3-layer spatial-temporal CNN model we trained which achieved the highest validation accuracies.

Google Cloud instances. Having access to the framework also freed us from having to figure out how to use the ML Engine, allowing us to concentrate on developing models.

We used Tensorboard to evaluate the performance of our models during the training process. While we were developing our initial techniques, we didn't even take the time to train for a full epoch, because the models achieved most of their performance gains during the first 10% of an epoch. For our final models, we trained for one full epoch over all the training data, which took approximately 5.5 hours. By this point the models had essentially converged - training accuracy had stabilized.

Because we were able to converge after a single epoch, overfitting was not much of a problem. Each training example is only seen once, so the model was not overly exposed to any one video. Nevertheless, we still implemented L2 regularization. Because we only looked at each training example once, we were able to attain validation Hit@1 of 79.1%. Compare this to the training accuracy, plotted in Figure 3. The Hit@1 obtained on the training set is essentially the same as that obtained from the validation set.

5. Experiment

We evaluated the performance of CNN models with different architectures and hyperparameters by training several models and comparing validation performance.

5.1. Selecting Model Architecture

Selecting the proper CNN architecture is critical for creating high-performance models. Beyond convolutional layers, proper usage of batch normalization and activation layers can be critical for optimizing performance.

We improved our model performance by adding batch normalization layers after every convolutional layer, making our loss less sensitive to small changes in input weights. Following every batch normalization layer, we used a leaky ReLU layer. We found that leaky ReLU activation layers outperformed traditional ReLU (see Table 1). This is likely due to their reduced susceptibility to the vanishing gradient problem.

We also tried inserting dropout layers into our model, which randomly set some activations to zero. We expected dropout to help improve accuracy by adding additional regularization to our model, but performance decreased in practice. This could be because we only trained our models over a single epoch, so overfitting to the training data was not too much of a concern.

We also evaluated the influence of adding additional convolutional layers to our model. We expected that the increased complexity of 5-layer models could result in improved performance over 3-layer models. However, we actually saw a small drop in performance on 5-layer models after training for one full epoch.

Finally, we tried different filter widths and number of filters for our convolutional layers. We achieved the highest performance with 10 filters of size 5. Using 5 or 20 filters of size 3 or 7 resulted in inferior performance.

5.2. Optimizing Hyperparameters

We used a learning rate that decayed in steps as training progressed. This meant we had three key hyperparameters to optimize related to the learning rate: the initial learning rate (LR), the decay factor (LRD), and the number of examples after which you decrease the learning rate (LRDE).

We trained models with different LR, LRD, and LRDE in order to optimize performance of our model. We got the best validation performance with a learning rate of 0.001, and learning rate decay of 0.95 every 100,000 training examples.

3-Layer Spatial-Temporal (CNN-BN-ReLU)-Dense

LR	LRD	LRDE	Hit@1	PERR	mAP
0.1	0.9	100,000	17.7	11.8	0
0.01	0.9	50,000	75.6	61.0	26.7
0.01	0.9	100,000	75.3	60.6	27.6
0.01	0.95	50,000	62.0	45.5	5.5
0.001	0.9	100,000	77.4	63.1	29.7

3-Layer Spatial-Temporal (CNN-BN-LeakyReLU)-Dense

LR	LRD	LRDE	Hit@1	PERR	mAP
0.01	0.9	100,000	78.9	64.9	34.4
0.01	0.95	100,000	79.1	65.2	36.1
0.005	0.9	100,000	79.1	65.3	34.5
0.001	0.9	100,000	77.4	63.1	29.7
0.001	0.95	100,000	79.5	65.8	35.0

5-Layer Spatial-Temporal (CNN-BN-leakyReLU)-Dense

LR	LRD	LRDE	Hit@1	PERR	mAP
0.01	0.9	100,000	77.4	63.1	31.9
0.001	0.9	100,000	77.2	63.0	30.0
0.001	0.95	100,000	78.9	65.0	34.5

TABLE 1. Optimizing Structure and Hyperparameters

5.3. Comparing our best models with Google's

Model	Hit@1	PERR	mAP
Google's Frame-Level Models			
Fully-Connected Logistic	50.8	42.2	11.0
Deep Bag of Frames	62.7	55.1	26.9
LSTM	64.5	57.3	26.6
Google's Video-Level Models			
Hinge Loss	56.3	47.9	17.0
Logistic Regression	60.5	53.0	28.1
Mixture of 2 Experts	62.3	54.9	29.6
Our Frame-Level Models			
3-Layer Early-Fusion CNN	73.7	59.0	24.6
3-Layer Spatial-Temporal CNN	79.1	65.2	36.1
5-Layer Spatial-Temporal CNN	78.9	65.0	34.5

TABLE 2. Model Comparison

Table 2 demonstrates that our CNN models performed significantly better than Google's models in their initial YouTube-8M paper. Our best model, the 3 layer spatial-temporal CNN, achieved Hit@1 of 79.1% and PERR of 65.2%.

The increased complexity of the 5-layer model did not result in higher accuracy. The spatial-temporal CNN outperformed the early-fusion CNN. We attribute this to the face that the spatial-temporal model's temporal convolution allows it to take into account how the frames in the video change over time. For example, perhaps a video tagged "mediation" shows less change from frame to frame, but a video tagged "monster truck rally" shows a lot of change between frames.

6. Conclusion

We were able to improve upon the performance of Google's initially published video classification models by taking into consideration the differences between individual frames in a video, and by switching from fully-connected to convolutional layers. We achieved a Hit@1 of 79.1% and PERR of 65.2% on our best model, compared to 64.5% and 57.3% on Google's LSTM model, the best of their initial models.

Google created a public Kaggle contest [19] for classifying YouTube-8M data. The winners of this contest achieved 84.9% on the "Google Global Average Precision" metric. In comparison, we had a global average precision of 72.3% for our best model. Google did not publish a global average precision for their models, so we cannot make a direct comparison there. The top scorers in the contest eventually have to publish their results, so it will be interesting to look at how their models compare to ours.

We could potentially improve our performance by adding the provided audio data to our model. For instance, we could create two additional 1D CNNs which convolve the audio data in the time and frequency domains. The output of these CNNs could then be concatenated with the output of the visual CNNs before the final fully-connected layer.

There are also a lot of less-radical changes that we didn't explore, but could potentially improve our models. For example, we could add additional fully-connected layers at the end of our dataflow. Or, we could train over multiple epochs to see if there is a slow increase in performance for additional training.

When it comes to video classification, $\sim$80% is a reasonably good accuracy. If 80% of the videos you see in response to a YouTube search are relevant to your query, you will probably be able to find a video that satisfies you. However, if you're searching for one specific video, there's still a $\sim$20% change that it will be miscategorized, and you'll never be able to find it. For these situations, it is important to keep pushing the frontiers of video and image recognition algorithms to further improve classification accuracy.

References

[1] S. Abu-El-Haija et al. YouTube-8M: A Large-Scale Video Classification Benchmark. *arXiv:1609.08675be*, 2016

[2] J. Deng, W. Dong, R. Socher, L. Jia Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015

[5] He, Kaiming, et al. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[6] Russakovsky, Olga, et al. Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision 115.3: 211-252*, 2015

[7] Karpathy, Andrej, et al. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.

[8] I. Laptev and T. Lindeberg. Space-time interest points. In *Proceedings of the International Conference on Computer Vision (ICCV), 2003*

[9] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008*

[10] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *Proceedings of the International Conference on Computer Vision (ICCV), 2005*

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

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

[13] Y. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes.*http://crcv.ucf.edu/THUMOS14, 2014*

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 17251732, Columbus, Ohio, USA, 2014*

[15] Z. Wu et al. Deep Learning for Video Classification and Captioning. *arXiv:1609.06782v*, 2016

[16] Google I/O 2013 - Semantic Video Annotations in the Youtube Topics API: Theory and applications. youtube.com/watch?v=wf_77z1H-vQ.

[17] Tensorflow: Image recognition. tensorflow.org/tutorials/image_recognition

[18] Google GitHub YouTube-8M Library, github.com/google/youtube-8m

[19] Kaggle “Google Cloud & YouTube-8M Video Understanding Challenge”, kaggle.com/c/youtube8m