Large-Scale Content-Only Video Recommendation

Joonseok Lee
Google Research
Mountain View, CA, USA
joonseok@google.com

Sami Abu-El-Haija
Google Research
Mountain View, CA, USA
haija@google.com

Abstract

Traditional recommendation systems using collaborative filtering (CF) approaches work relatively well when the candidate videos are sufficiently popular. With the increase of user-created videos, however, recommending fresh videos gets more and more important, but pure CF-based systems may not perform well in such cold-start situation. In this paper, we model recommendation as a video content-based similarity learning problem, and learn deep video embeddings trained to predict video relationships identified by a co-watch-based system but using only visual and audial content. The system does not depend on availability on video meta-data, and can generalize to both popular and tail content, including new video uploads. We demonstrate performance of the proposed method in large-scale datasets, both quantitatively and qualitatively.

1. Introduction

Recommendation systems play more and more crucial role in information filtering, as the amount of data we collect gets explosively larger, in a variety of areas including movies, music, videos, books, news, or scholarly articles. In many cases, recommendation problem comes with a given context, and proper items to retrieve are usually highly dependent on the context. The target user is often such an important context, especially when relevance is dependent highly on taste. Another popular context is given by another item(s). In online shopping mall, for example, we would like to recommend products given a set of other items that the user recently purchased.

Video recommendation problem is also usually defined with those two context variables. The target user is an important context in many video recommendation systems (*user-to-video* recommendation), as different persons usually have different taste on videos. Some user may prefer to watch lots of soccer videos, while some other user may want to watch more music videos. Other video(s) may be given as a context as well. Continuous play, for example, recommends a good video to play next, once the current video the user is watching is done (*video-to-video* recommendation). If a user is watching an episode of TV series, for instance, it is natural to infer that the user may want to watch the next episode of the same series next.

A concrete example of video recommendation systems above is online video sharing service. On YouTube Homepage, for example, you may observe a section "Recommended", which is personalized. This is a good example of user-to-video recommendation, as this section is filled based on the user's profile, watch history, and site-wide behaviors. While you are watching a video, YouTube shows a set of related videos on the right side. This recommendation may depend not only on the inferred user taste, but also on the video currently being watched. These examples are shown in Figure 1.

One way to solve personalized video recommendation problems is using collaborative filtering (CF) approaches, which suggest related items for a user based on other users with similar taste. They may work well for video recommendation problems if enough user preference data are available. However, CF-based models also have limitations. Most importantly, CF methods seriously suffer from the *cold-start* problem. When a new user joins and gets recommendations, the system does not have ratings or watch history to utilize for her, so is unable to infer her taste. It is same for a new item (video); when a new video is uploaded, no one has watched or rated it, so it is impossible to find users who may like it. This cold-start problem gets alleviated until the system collects enough feedback from users. In the mean time, recommendations generated for the new user or for the new item can be still poor as it has little information to utilize. This problem can be even more severer in a system where new items are produced and consumed in fast pace. In YouTube, for example, 300 hours of new videos are uploaded every minute.[1] As it has more and more new videos, it becomes more crucial to search and recommend from the fresh videos, but pure CF methods are unable to perform this.

[1] As of Mar 23, 2017. https://fortunelords.com/youtube-statistics/

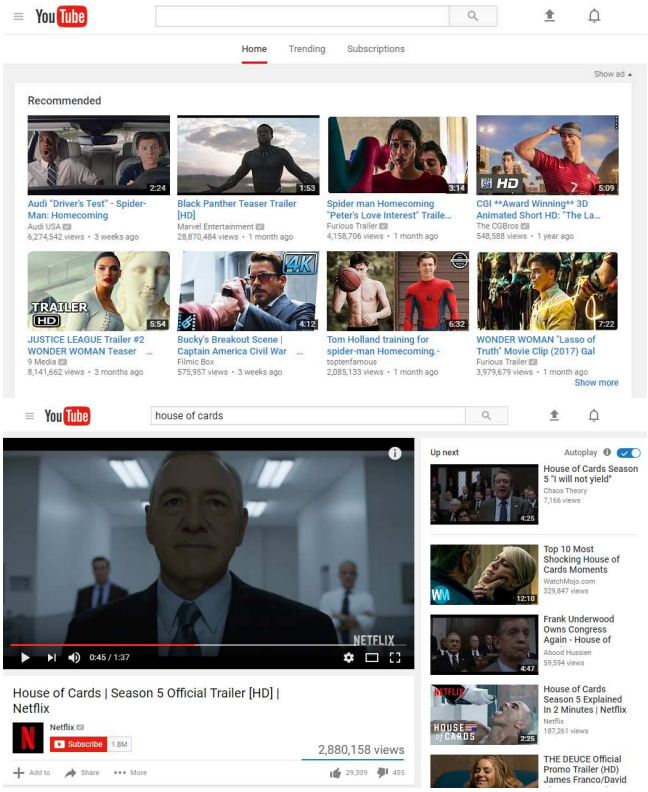

Figure 1. Examples of YouTube video recommendations. (*Top*): Homepage recommendations (user-to-video) for a user who recently has watched movie trailers of "Spider-Man: Homecoming (2017)" and "Wonder Woman (2017)". (*Bottom*): Autoplay recommendations (video-to-video) for a user who is currently watching a trailer for "House of Cards Season 5 (2017)".

Another limitation of pure CF approaches is that the system is completely ignorant of content. As CF does not utilize who is the user or what is the item, it makes hard for us to see the reason for recommendation. In other words, it is hard to explain in a human-interpretable way.

To overcome these limitations of CF-based recommendations, content-based approaches have been proposed. In general, they recommend items with similar content to what the user liked or purchased before. For instance, movies with the same genres, main actor/actress, or director may be considered as similar. Given this kind of content information about the items (and some background about the users) available from the beginning, content-based methods can compute similarity and recommend items even without any user feedback or interaction with those. In some domains like movies or music, items are usually released with rich meta-data, which are proved to be useful for recommendations. [3]

Nevertheless, meta-data-driven content-based recommendation is not always a feasible solution, as such meta-data may not be available or not precise for some applications. In video recommendation problems, most user-created videos do not come with well-structured meta-data. We might be able to take advantage of the title and description of the video, but in general there is no guarantee that they are accurate or in high quality. Relying on those meta-data may lead to inaccurate or poor recommendations.

In this paper, we propose a content-based video recommendation relying on **raw video and audio content**, to resolve these issues. Specifically, we model video recommendation problem as a video similarity learning problem, and learn a compact representation of a video preserving its semantics from its visual and audial signals using deep neural networks. We embed all videos into an embedding space, where similar (recommendable) videos are located close to each other. We show that the learned video embeddings generalize beyond simple visual and audial similarity and are able to capture complex semantic relationships. We summarize main advantages of our proposed methods:

- As visual and audial information is available right after the video has created and uploaded, our model is able to extract features and embed it right away. This solves the *cold-start problem*, where we had to wait days to months with CF-based methods to collect user feedback for new videos to be recommended to the right person in a right context.
- Our proposed method is more *robust to spamming*, as it relies on raw video and audio data which are not easily alterable, contrast to meta-data like title or description. Recommendations based on video content itself fundamentally block spamming videos with an irrelevant, but sometimes provocative title.
- We build a highly scalable video recommendation framework based on the compact video representation. We demonstrate performance of our system with YouTube 8M [1], the largest public video dataset as of this writing.

We start by reviewing related work in literature in Section 2. Then, we introduce our proposed system with details in Section 3. In Section 4, we demonstrate performance of our system both with public data and with real system. We conclude with our contributions and propose future work in Section 5.

2. Related Work

Recent development of deep learning has led notable progress on video understanding. Content-based video classification (or video annotation) takes advantage of convolutional neural networks (CNN) for image (frame) understanding [12] as well as recurrent neural networks (RNN) for temporal aspect of videos. [30, 27, 21, 24, 28] Deep

learning also has been applied to action recognition [14, 16, 2, 10, 4], as well as video search [29, 11].

Content-based recommendations also have previously appeared in literature. Han et al. [7] proposed a dance-style recommendation method based on action representation. Van den Oord et al. [26] explored deep content-based music recommendations, learning CNNs fitting mel-frequency cepstral coefficients from songs to the ratings. Rothe et al. [19] used computer vision features to "regularize" matrix factorization (MF). They included an additive term to the MF model, which penalizes if the dot-product of two latent vectors from the MF model is far from the cosine similarity of their visual features.

While the previously mentioned works extract features from the **actual** content, there is a larger wealth of works that employ **editorial** content-based information, such as movie genres, description and user-generated tags, traditionally known as content-based filtering (CBF). [17, 31, 18, 15, 23, 22, 5] We do not use editorial information, with the exception of movie titles, which we only use to collect YouTube trailers for movies.

3. Method

In this section, we detail our video feature learning model and scalable recommendation system based on it. We start by describing how we extract video and audio features, followed by our neural network model fine-tuning for recommendation task. Then, we introduce techniques we applied to make the system more scalable.

3.1. Video and Audio Features

It is impractical to process videos as their raw form, as the size of dataset may be terabytes to petabytes. To start with more compact representation of videos, we pre-process them and extract frame-level features using a state-of-the-art deep model: the Inception-v3 network [25] trained on ImageNet [6]. Concretely, we decode each video at 1 frame-per-second up to the first 3,600 seconds, feed the decoded frames into the Inception network, and fetch the ReLu activation of the last hidden layer, before the classification layer. Afterwards, we apply PCA (and whitening) to reduce feature dimensions to 1,500 for storage and computational reasons. These frame-level features are aggregated into a video-level feature by average pooling. There may be more sophisticated methods to combine frame-level features, but we leave it as a future work.

We extract audio features using a VGG-inspired acoustic model with a modified version of ResNet-50 [9]. Specifically, the audio is divided into non-overlapping 960 ms frames, and then decomposed with a short-time Fourier transform with 25 ms windows for every 10 ms, producing 64 mel-spaced frequency spectrogram. We feed 100 feature frames (corresponding to 1 second) into the ResNet [8], followed by average pooling to aggregate them into the video level.

3.2. Fine-Tuning for Recommendation

Although the visual and audial features we extract above somewhat represent the video content, it may not be optimal for recommendation task, as they are not trained for this purpose. Thus, we train a feedforward network on top of the input features to fine-tune for recommendation.

Suppose a user is continuously watching videos $\{v_1, v_2, v_3...\}$ on YouTube. If a video v_{k+1} has been watched after v_k, we call v_k and v_{k+1} are *co-watched*. This notion can be extended to aggregate statistics with many users. That is, we may call video v_a and v_b are co-watched in general, not just by a user. Especially for video-to-video recommendations, this aggregated co-watch relationship may be more robust to deliver content similarity.

We may want to locate the embeddings of two co-watched videos close to each other. To achieve this, we directly optimize a ranking loss, called the triplet loss [20]. A training data point is defined as a triplet of three videos: *anchor*, *positive*, and *negative*. While training, we update the feature vectors of these three videos so that the *anchor* video is closer to *positive* than to *negative*. We put two co-watched videos as the *anchor* and *positive* respectively, and randomly assign a *negative* video to create a triplet. The objective function is given as

$$\min \sum_{i=1}^{n} L(f(\mathbf{x}_i^a), f(\mathbf{x}_i^p), f(\mathbf{x}_i^n)) \tag{1}$$

where $\mathbf{x}_i^a$, $\mathbf{x}_i^p$, and $\mathbf{x}_i^n$ correspond to the vector of *anchor*, *positive*, and *negative* video of i-th training data point, respectively, f is the feed-forward network, and L is a loss function penalizing if $f(x_i^a)$ is closer to $f(\mathbf{x}_i^n)$ than to $f(\mathbf{x}_i^p)$. Some widely-used loss functions include hinge loss $L_{\texttt{hinge}}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = [\|\mathbf{x}-\mathbf{y}\|_2^2 - \|\mathbf{x}-\mathbf{z}\|_2^2 + \alpha]_+$, log loss $L_{\texttt{log}}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \log(1 + \exp\{\|\mathbf{x}-\mathbf{y}\|_2^2 - \|\mathbf{x}-\mathbf{z}\|_2^2 + \alpha\})$, and exponential loss $L_{\texttt{exp}}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \exp\{\|\mathbf{x}-\mathbf{y}\|_2^2 - \|\mathbf{x}-\mathbf{z}\|_2^2 + \alpha\}$, where α is a margin parameter.

As we normalize the final embedding, we can compute distance (or similarity) between two video vectors more efficiently; taking dot-product gives relative similarity, not necessarily computing exact cosine similarity. Henceforth, we take dot-product and cosine similarity inter-changeably.

With this network, features are trained to locate in a similar place with other videos watched together. As we initialize with visual and audial features extracted from content, the output features represent not only visual and audial content, but also watching behavior patterns of users. Also, as the output feature is usually in much smaller dimension than combined input features, the produced feature represents semantics of the video in a more compact way.

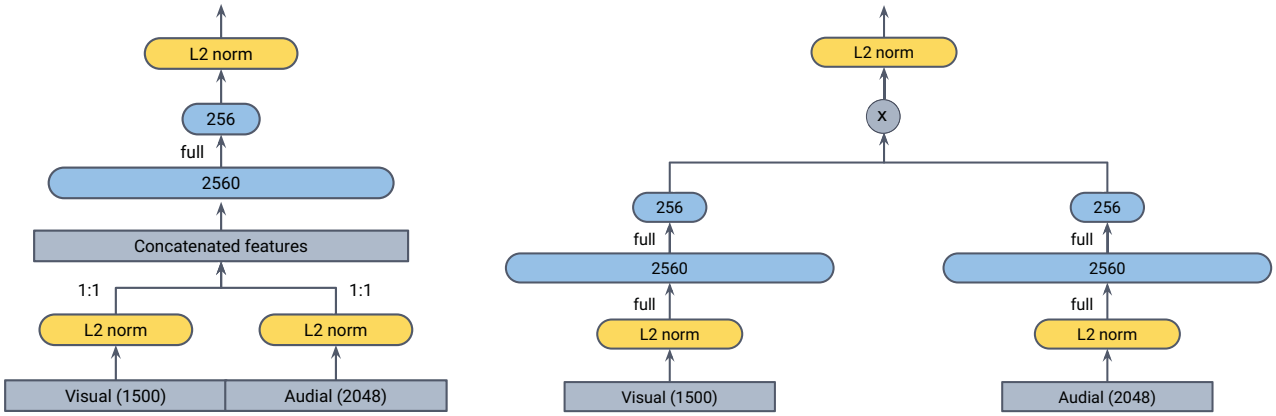

Figure 2. Neural network models for fine-tuning video features optimized for recommendation. (*Left*): Early fusion, combining the visual and audial features from the beginning. (*Right*): Late fusion, learning two separate towers for visual and audial signals, and combining them by element-wise multiplication in the end.

Network Architecture

We propose two different types of networks as shown in Figure 2. The left one concatenates two input features from the beginning (after L_2 normalization individually), and we stack a feed-forward network on top of it. We call this *Early Fusion* network. On the other hand, the right one in Figure 2 trains each input signal in separate towers, and the output is combined by element-wise multiplications. We call this *Late Fusion* model. (We tried addition instead of element-wise multiplication as well, but it performed less well.) Other than these two extremes, there can be more options such as training separate towers at the beginning but concatenated at some level and train additional layers before output.

3.3. Scalable Recommendation System

With the feature learning described previously, each video is now represented as a compact vector representing content semantics. With this representation, **video-to-video** recommendation problem can be formulated as a similar video search problem in the embedding space. Formally, we solve the following optimization problem:

$$\min_{v \in V - \{q\}} dist(\mathbf{x}_q, \mathbf{x}_v) \tag{2}$$

where q is the query (seed) video, V is the set of all videos, and $dist(\cdot, \cdot)$ is a distance metric between two vectors. In practice, we may output top $k > 1$ items to recommend.

User-to-video recommendation can be formulated similarly, if we represent each user as a sequence of videos that she has watched recently. Unlike the video-to-video problems above, however, we may have more than one query video. Computing similarity with a candidate video now gives a sequence of scores, so we need to aggregate those into one for easy comparison. An intuitive way is *average aggregation*, where we take average of similarity scores between the candidate video and previously watched videos. Intuitively, a candidate video will be high-scored if it fits with most of previous videos overall. Formally,

$$\min_{v \in V - Q} \frac{1}{|Q|} \sum_{q \in Q} dist(\mathbf{x}_q, \mathbf{x}_v) \tag{3}$$

where Q is the set of videos the user has watched. Another intuitive approach is *max aggregation*, where we take the maximum score among all videos in watch history. For instance, suppose the user has watched many animation videos and one "House of Cards" episode. Another episode of "House of Cards" may get high score, since it is very similar to one video in watch history. Formally,

$$\min_{v \in V - Q} \max_{q \in Q} dist(\mathbf{x}_q, \mathbf{x}_v) \tag{4}$$

In real services like YouTube, $|V|$ can be millions or billions, but these recommendations need to be computed at the level of micro-seconds. They may cache features for quick retrieval, so it is important to reduce the feature size to save storage. We quantize feature values to have only a limited number of representative values. Specifically, we choose 2^k representative values to minimize the expected squared distance between original and quantized features, representing each dimension with k bits. For faster computation of similarity, we may pre-compute and cache all possible combinations of multiplications. Instead of actually taking float multiplication in serving time, we may lookup the cached values to quickly dot-product.

Another possible bottle neck of this system is solving Eq. (3) - (4). A naive implementation of this may iterate all possible pairs of q and v, leading to quadratic time complexity $O(|Q||V|)$. For the case of dot-product similarity ($dist(\mathbf{x}_q, \mathbf{x}_v) = \mathbf{x}_q^\top \mathbf{x}_v$), average aggregation in Eq. (3) can

be done in linear time by taking advantage of distributive property of inner-product:

$$\min_{v \in V-Q} \frac{1}{|Q|} \sum_{q \in Q} \mathbf{x}_q^\top \mathbf{x}_v = \min_{v \in V-Q} \left(\frac{1}{|Q|} \sum_{q \in Q} \mathbf{x}_q \right)^\top \mathbf{x}_v$$

As the averaging part of vectors in Q does not rely on v, this can be pre-computed once and reused. This way of implementation achieves time complexity of $O(|V|+|Q|)$.

4. Experiment

We extracted video and audio features from 278M YouTube videos. We randomly split the videos into training and eval partition with 7:3 ratio, and created the training triplets based on in-house related video graph, with videos in the training partition only. Assuming video-to-video recommendation scenario, we evaluate with two different cold-start cases: 1) where we recommend from eval (fresh) videos for a query from train partition (*train-to-eval, T2E*), and 2) where we recommend established videos for a query with fresh video (*eval-to-train, E2T*). We evalute end-to-end recommendation performance in two widely-used ranking metrics:

1) Normalized Discounted Cumulative Gain (**NDCG**) considers the order of recommended items in the list. Formally, DCG@k is defined as

$$DCG@k = \frac{1}{|U|} \sum_{u \in U} \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \quad (5)$$

where i is the position in the recommendation list and $rel_i \in \{0, 1\}$ indicates whether the i-th item is relevant to the user or not. NDCG is the ratio of DCG to the maximum possible DCG for that user. This maximum occurs when the recommended items are presented in decreasing order of user preference. We used $k = 10$ for our experiment.

2) Mean Average Precision **(MAP)** is the area under precision-recall curve, given by

$$MAP = \frac{1}{|U|} \sum_{u \in U} \int_0^1 P(r) dr, \quad (6)$$

where r ranges over all possible recall levels and $P(r)$ is the precision at recall level r. In practice, the integral is replaced with a finite sum over every position in the ranked sequence of recommendations.

4.1. Model Parameters

We first show empirical results with several training options and parameters for our video-to-video model.

At first, we compared performance of the video-only model and video + audio model. Video-only model is equivalent to the left tower of late fusion model in Figure 2. Table 1 shows NDCG and MAP scores for both models. In both types of cold-start scenarios, we see that the audio signal clearly helps to improve end-to-end recommendation performance over the video-only model.

Input Features	NDCG		MAP	
	T2E	E2T	T2E	E2T
Video Only	7.22%	10.38%	2.27%	2.68%
Video + Audio	**8.48%**	**12.04%**	**2.70%**	**3.20%**

Table 1. Recommendation performance in NDCG and MAP with video features only vs. video + audio features. This clearly shows having both visual and audial signals helps.

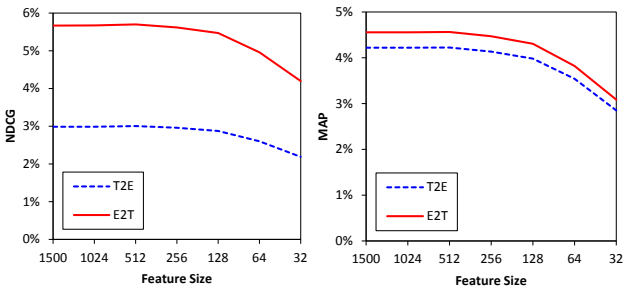

Figure 3. Recommendation performance in NDCG and MAP with various output feature size.

Another interesting parameter is the size of output feature vector. There is general trade-off between capacity and cost; the larger the size, the more expressive the feature can be, potentially leading to better performance. For this, however, we pay the cost like longer training time and computational resources. If we are about to serve the features online, the storage size may be also an important concern, preventing large feature size despite its better performance. In Figure 3, we see that increasing the feature size above 512 gives little gain. On the other hand, the performance dramatically drops when the size is smaller than 128. We chose 256 dimensions as a good compromise between performance and computational cost.

We also tried several different neural network architectures, with shallower vs. deeper models, different number of nodes in hidden layers, and early vs. late fusion. Table 2 summarizes end-to-end performance with different architectures we tried. Interestingly, we see little advantage with deeper models. It is probably because the input features have already gone through deep models (Inception and ResNet), and this fine-tuning may not be that complex to take advantage of deeper models. Instead, deeper models take longer time to train. Another observation is that increased capacity of the first hidden layer actually helped to improve the performance. Lastly, late fusion tends to always perform better than early fusion. This might be because one signal is almost ignored with early fusion if it is not strong enough to survive against the other. From this empirical evidence, we chose 4000-256 architecture with

Architecture (Fusion)	NDCG		MAP	
	T2E	E2T	T2E	E2T
2560-256 (Early)	8.48%	12.04%	2.70%	3.20%
4000-256 (Early)	9.51%	13.13%	3.01%	3.49%
4000-256 (Late)	**9.91%**	**13.46%**	**3.13%**	**3.58%**
4000-512-256 (Early)	9.29%	12.64%	2.90%	3.33%
4000-1024-256 (Early)	9.46%	12.80%	2.95%	3.38%
4000-1024-256 (Late)	9.82%	13.05%	3.07%	3.44%

Table 2. Recommendation performance in NDCG and MAP with several different neural network architectures.

Output Dim	Quantization	Size (byte)	T2E	E2T
512	original	2048	2.99%	5.68%
256	original	1024	2.96%	5.61%
32	original	128	2.19%	4.20%
128	8 bit/dim	128	2.87%	5.47%
256	4 bit/dim	128	**2.94%**	**5.58%**
512	2 bit/dim	128	2.87%	5.47%
128	4 bit/dim	64	**2.84%**	**5.40%**
256	2 bit/dim	64	2.75%	5.23%
128	2 bit/dim	32	2.48%	4.74%

Table 3. Comparison in NDCG for various feature size with quantization. With quantization, we can have larger dimensionality with same size, achieving better end-to-end performance. Bold-faced figures mean the best performance within the same size.

late fusion.

Lastly, we compare how much performance we can retain with different levels of quantization. We quantize each dimension into 2, 4, and 8 bits (from 4 bytes float), by having 4, 16, and 256 representative values. Table 3 compares end-to-end performance with various quantization options. The top two rows show the best performance we achieved with largest output size. When we reduce the output size to 128 bytes (16x from the best), we compare 4 options: {32 floats, 128 * 8 bits, 256 * 4 bits, 512 * 2 bits}. We observe that 4 bits per dimension are enough to preserve the original performance, as NDCGs dropped just 0.02% and 0.03% for each scenario, respectively. 2 bits per dimension seem not enough to preserve information, when we compare it against performance with original 512 dimensions. When we reduce further to 64 bytes, we still achieve reasonable performance with 4 bits/dim quantization with 128 dimensions. From this comparison, we choose 256 dimensions with 4 bits each as our model for experiment.

4.2. MovieLens Trailers

We compare against previous models on MovieLens, one of the most widely-used public dataset for recommendations. As it does not come with video and audio data of movies, we collected movie trailers available on YouTube as a proxy. To obtain the trailer YouTube video IDs for the MovieLens movies, we queried Google with the canonical

Figure 4. Search results for query "Toy Story (1995)". Usually, the YouTube result is the third result, after IMDB and Wikipedia.

"Title (year)" for all movies in MovieLens 20M[2], and took the first YouTube result with 'Trailer' in its title, limiting to the top 20 results. Figures 4 and 5, respectively, show the search results of a title query and random frames from a few trailers. In this way, we collected 1,489 trailers (out of 1,682 movies, 88.5%) in MovieLens 100K dataset, and 22,798 trailers (out of 27,279 movies, 83.6%) in MovieLens 20M dataset. Movies without trailers were excluded from experiments.

For each user, we split ratings into training and test partitions by 5:5 ratio. Users with less than 10 test ratings were excluded from the experiment. For each user, we define the set of preferred movies as movies in training partition with a rating higher than some threshold: 1) fixed threshold with 5 stars only, 4 stars or higher, and 3 stars or higher; 2) the user's own mean rating; and 3) no threshold (simulating applications with watch history given without explicit ratings). We then rank candidate videos in the test partition by similarity to the preferred movie set, computed as dot-product in our embedding space. We aggregated the scores by max and average aggregation proposed in Eq. (3) - (4). For this experiment, we used $rel_i \in \{1, 2, 3, 4, 5\}$ for DCG (Eq. (5)) to make the setting comparable to [13].

Table 4 summarizes performance on MovieLens dataset in NDCG@10 and MAP. Below is our observations and discussions:

1. The higher threshold we have, the performance gets better in general. This makes sense, since with higher threshold we only take videos that we are sure the user liked, as long as we have enough ratings from the user.

[2] https://grouplens.org/datasets/movielens/

Figure 5. Frames from four MovieLens movies. Each row shows frames from one YouTube trailer. Top-to-bottom: "Toy Story (1995)", "Race the Sun (1996)", "S.W.A.T.: Firefight (2011)" and "The Journey of August King (1995)".

We see some performance drop when we do not filter out videos with low ratings, but in applications only with implicit feedback (e.g, clicks, views) this filtering may not be possible.

2. Max aggregation performs generally better than average aggregations. This means recommending the most similar item to one of the user's favorite performs better, probably because many users have more than one preferred types of movies. We also tried some variations of these aggregations, such as average of top $k = 2, 3, 5$, and variance, but these did not outperform max and average aggregations.

3. Another purpose of this comparison is to see how much video-content-only recommendation models can perform compared to CF-based models. We compared against Local Collaborative Filtering [13], using the same experimental setting. Their models achieve NDCG@10 of 0.71 - 0.72, and MAP of 0.76 - 0.77 (See Figure 1 - 3 in [13].) As it is known that CF-based models are powerful than content-based with sufficient ratings, it is not surprising that the proposed method does not outperform it. This result shows that we still have some headroom to improve, especially if we learn on the target dataset directly.

4.3. YouTube8M Demonstration

Lastly, we demonstrate qualitative performance of our proposed method on YouTube8M dataset [1], which is the largest multi-label video classification dataset as of this writing. It is composed of about 8 million videos of 500K hours of videoannotated with a vocabulary of 4,800 visual entities.

Aggregation	Watch History Threshold	NDCG	MAP
Max	Higher than user mean	0.6242	0.7186
Max	5 stars only	**0.6282**	0.7155
Max	4 stars or higher	0.6251	**0.7196**
Max	3 stars or higher	0.6149	0.7141
Max	All rated movies	0.6071	0.7078
Average	Higher than user mean	0.6022	0.7073
Average	5 stars only	0.6133	0.7120
Average	4 stars or higher	0.6018	0.7065
Average	3 stars or higher	0.5954	0.7013
Average	All rated movies	0.5900	0.6966

Table 4. Recommendation performance in NDCG and MAP with several different neural network architectures.

We aim to the video-to-video recommendation problem since this dataset does not contain any individual user data. We are given a query video, and our task is retrieving the most relevant top-k videos to the seed. To illustrate cold-start situation, we choose all query videos from the eval partition (approximately 30%), thus no seed videos in this example has been shown to the model during training. All videos in the dataset other than the chosen query are considered as candidates. As described in the previous section, we did not use any meta-data other than video and audio signals.

Figure 6 illustrates some examples of the result. From the top, 1) the query video is about Sergio Busquets, a Spanish soccer player. All recommended videos are obviously about soccer, and in the last two thumbnails we see the same uniform from F.C. Barcelona. This shows our proposed method effectively finds out similarity even in fine details. 2) The second example is about a video game with fixed screen. As expected, all recommended videos are other videos playing the same game. 3) Next query is about a hamster. All recommended videos show hamsters in them, but each hamster is in different color and shape. Nevertheless, our proposed method still outputs these as relevant videos. 4) In the next seed video, a girl demonstrates how to make her hair braided ('tresse' in Spanish, seen in the title). The recommended videos look semantically very relevant, as they are also showing how to braid hair. Interestingly, the title of the first and third recommended videos are in Russian and Korean, respectively. This example shows that our proposed method can retrieve visually relevant videos across languages, which usually are not supported in most systems. (Recommending videos in different languages may not maximize user satisfaction in real system though. For practical use, some heuristics may be applied to filter videos in other languages out.) 5) The last example is similar; the seed and recommended videos share the same theme (live music) and similar stage setting. However, these might not be the best recommendations, since singers and even languages are different, so some of these

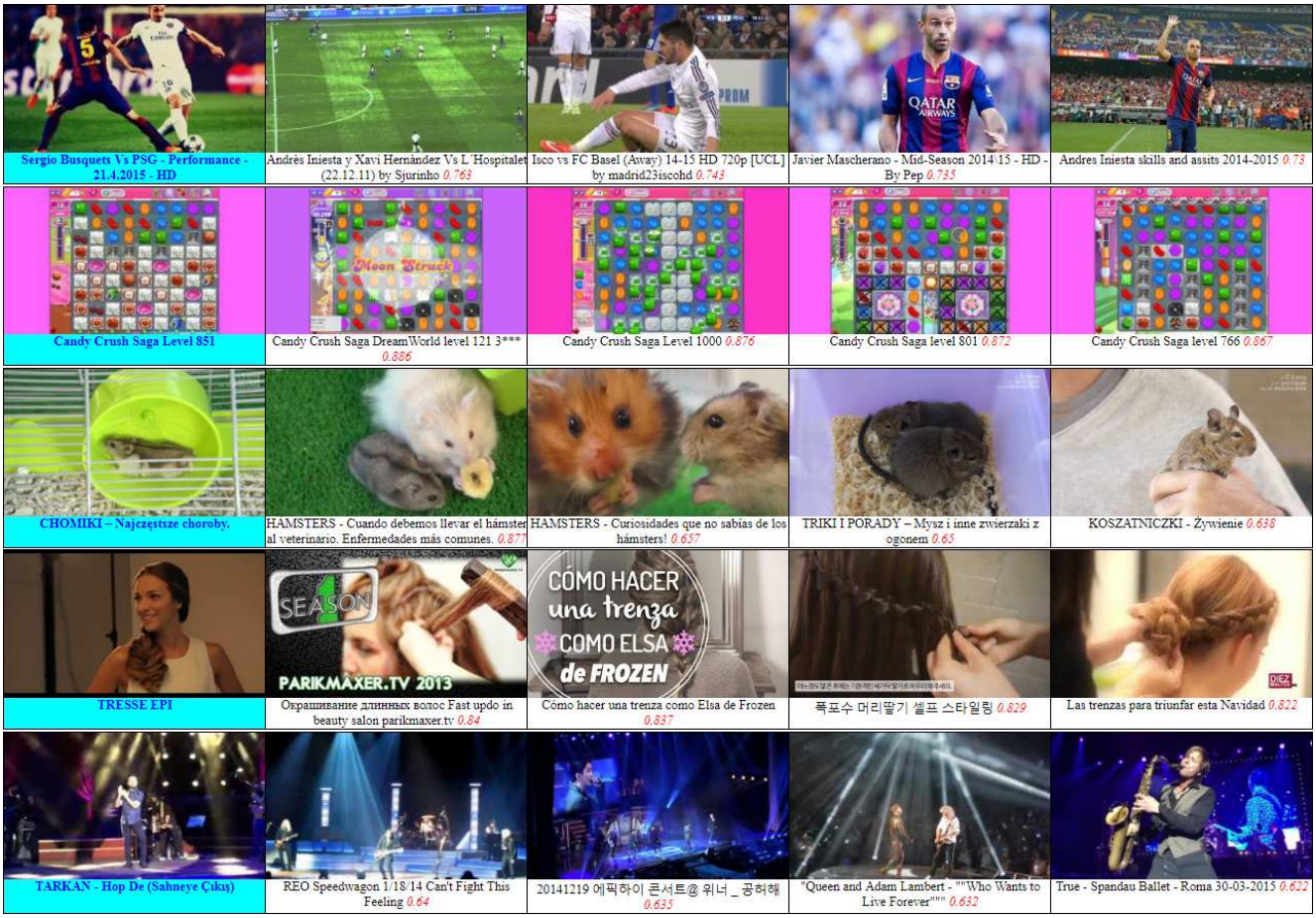

Figure 6. Demonstration of video-to-video recommendation with YouTube8M Dataset. The left-most column is the query video, and other videos in the same row are top 4 recommended videos by our proposed system. We show YouTube thumbnail, title, and relevance score we computed (in red italic).

may not be the ones users are looking for. Overall, visual and audial features are powerful to retrieve relevant videos from the corpus, but need to be used in conjunction with other sources like meta-data or collaborative filtering signals to compensate its blind spots.

5. Summary and Future Work

In this paper, we proposed a video recommendation system based on raw visual and audial content. The proposed model learns a compact representation of videos optimized not only for video and audio content, but also for semantics extracted from watch patterns. We also develop a scalable system based on the content features, evaluated quantitatively as well as qualitatively on two large-scale benchmark datasets.

There are some areas we may be able to improve this work further. First of all, it is obvious that averaging frame features to get the video-level feature may not be the optimal way to do. We may apply advanced deep learning models such as LSTM for this part to improve the performance. Second, we may also apply more advanced quantization methods such as product quantization. With more compact and effective quantization, we may learn even larger content features with smaller size, potentially achieving higher precision. Lastly, we may also try different loss functions to learn video embeddings. We leave them as potential future work.

References

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016. 2, 7

[2] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In *International Workshop on Human Behavior Understanding*, 2011. 3

[3] C. Basu, H. Hirsh, W. Cohen, et al. Recommendation as classification: Using social and content-based information in

recommendation. In *Proc. of the AAAI Conference on Artificial Intelligence*, 1998. 2

[4] C. Beaudry, R. Péteri, and L. Mascarilla. An efficient and sparse approach for large scale human action recognition in videos. *Machine Vision and Applications*, 27(4):529–543, 2016. 3

[5] L. M. de Campos, J. M. Fernandez-Luna, J. F. Huete, and M. A. Rueda-Morales. International journal of approximate reasoning. *International Journal of Approximate Reasoning*, 2010. 3

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 3

[7] T. Han, H. Yao, C. Xu, X. Sun, Y. Zhang, and J. J. Corso. Dancelets mining for video recommendation based on dance styles. *IEEE Transactions on Multimedia*, 19(4):712–724, 2017. 3

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[9] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. Cnn architectures for large-scale audio classification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017. 3

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

[11] L. Jiang, Y. Kalantidis, L. Cao, S. Farfade, J. Tang, and A. G. Hauptmann. Delving deep into personal photo and video search. In *Proc. of the ACM International Conference on Web Search and Data Mining (WSDM)*, 2017. 3

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 2

[13] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer. Local collaborative ranking. In *Proc. of the International Conference on World Wide Web (WWW)*, pages 85–96, 2014. 6, 7

[14] J. Liu, J. Luo, and M. Shah. Recognizing realistic actions from videos "in the wild". In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 3

[15] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering. In *Proc. of the SIGIR Workshop on Recommender Systems*, 2001. 3

[16] J. C. Niebles, C.-W. Chen, and L. Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2010. 3

[17] D. Y. Pavlov and D. M. Pennock. A maximum entropy approach to collaborative filtering in dynamic, sparse, high-dimensional domains. In *Proc. of Neural Information Processing Systems*, 2002. 3

[18] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000. 3

[19] R. Rothe, R. Timofte, and L. V. Gool. Some like it hot - visual guidance for preference prediction. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[20] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[21] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *Proc. of the International Conference on Machine Learning (ICML)*, 2015. 2

[22] F. Strub, J. Mary, and R. Gaudel. Hybrid collaborative filtering with neural networks. *CoRR*, abs/1603.00806, 2016. 3

[23] X. Su, R. Greiner, T. M. Khoshgoftaar, and X. Zhu. Hybrid collaborative filtering algorithms using a mixture of experts. In *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence*, 2007. 3

[24] Y. Sun, Z. Wu, X. Wang, H. Arai, T. Kinebuchi, and Y.-G. Jiang. Exploiting objects with lstms for video categorization. In *Proc. of the ACM Internacional Conference on Multimedia*, 2016. 2

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[26] A. van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26*, 2013. 3

[27] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proc. of the ACM International Conference on Multimedia*, 2015. 2

[28] X. Yang, P. Molchanov, and J. Kautz. Multilayer and multimodal fusion of deep neural networks for video classification. In *Proc. of the ACM International Conference on Multimedia*, 2016. 2

[29] S.-I. Yu, L. Jiang, Z. Xu, Y. Yang, and A. G. Hauptmann. Content-based video search over 1 million videos with 1 core in 1 second. In *Proc. of the ACM on International Conference on Multimedia Retrieval*, 2015. 3

[30] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[31] C.-N. Ziegler, G. Lausen, and L. Schmidt-Thie. Taxonomy-driven computation of product recommendations. In *Proc. of the thirteenth ACM international conference on Information and knowledge management*, 2004. 3