NATURAL VIDEO UNDERSTANDING

Andrew Floren

University of Texas at Austin
Electrical and Computer Engineering
Austin, Texas

ABSTRACT

Abstract

***Index Terms*—** One, two, three, four, five

1. INTRODUCTION

Natural video – that is, any video of a real scene that has not been synthesized – is both ubiquitious and complex. Movies, news, television shows, and even funny videos on YouTube are all examples of natural videos that we are exposed to every day. Despite the complexity of the data, humans are able to immediately understand the content of such videos and boil them down to highly condensed textual descriptors such as "text description of simple video sequence" (Figure #). An automated and efficient algorithm that could perform this task would have many applications such as allowing users to search video databases for specific content or searching through individual videos for particular scenes.

1.1. Figure

3 or 4 frames from a video that tell a simple story along with a text description

Despite the ease with which humans can solve this task, it is still incredibly difficult for a computer. To begin to solve this problem, we first need to parse the 'words' and 'sentences' that make up a video. The scenes in a video correspond to these sentences while the objects in that scene make up the words. Image recognition has come a long way in the last few years and differentiating between thousands of natural objects in an image with high accuracy is now possible [refs]. By applying these techniques to video, we can detect the objects in a frame, and by clustering these objects, we can begin to construct scenes.

We have developed an efficient method to detect, store, and search through scenes in a video based on image content. The algorithm is specifically designed to be used in conjunction with H.264 encoded video, however it is applicable to any video compression technique that utilizes I-frames. A deep convolutional network is used to classify image content of each I-frame in a video. The output of the neural network is a 1,000-dimensional classification-vector that represents how likely each of the 1,000 content classes is present in the image. Agglomerative clustering is used to create a hierarchical scene representation from the classification-vectors. A strong adjacency constraint is enforced during clustering both for algorithmic efficiency and to produce temporally consistent scenes. Finally, we introduce a method for constructing search-vectors based on textual labels and searching this hierarchical scene representation using these search-vectors.

2. IMAGE RECOGNITION AND OBJECT DETECTION

Image recognition is the task of identifying the content of an image. For example, determining whether an image is of a dog or a fox. However, images are generally made up of multiple objects of varying categories. Object detection is the further task of identifying these distinct objects – what they are and where in the image they are located. Object detection is usually accomplished by applying image recognition to candidate windows of varying sizes and locations within an image [ref]. One of the best performing image recognition techniques (based on the results of ILSVRC2014 [ref]) is deep convolutional neural networks.

Neural networks are a type of model inspired by the networks of neurons found in the brain. They are composed of nodes with directed weighted edges in a graph with each node having a particular activation function. Each node calculates the weighted sum of all the outputs of the nodes connected to it and then applies it's activation function to the result. The output of the activation function then becomes that node's output. For practical reasons, networks are generally structured in layers with no connections between nodes in a layer and no connections to non-adjacent layers allowed (figure #). This allows for a simple and efficient vector implementation. Networks are configuration for classification by creating an input layer with a node for each dimension of the data points to be trained on, one ore more 'hidden' layers of varying sizes, and finally an output layer with a node for each output class (figure #). The intermediate layers are known as 'hidden' layers because they presumably help model some hidden process in the data.

2.1. Figure

simple neural network model

The weights between each layer are learned using a gradient descent optimization technique. Training data points are placed in the input layer and the output values for each subsequent layer are calculated until the values for the output layer are determined. The difference between the output values and the true labels are used to calculate a descent vector on the weights. Calculating the descent vector requires differentiating the output nodes of the network with respect to the input nodes. The Jacobian matrix representing this differential must be recalculated for each training example because the model is non-linear. However, the matrix can be efficiently calculated by applying the chain rule to each layer. This process is called back-propogation training because the weight updates are first calculated for the output layer and then backwards for each layer until the input layer is reached.

A convolutional layer in a neural network refers to a specific layer structure where each node is only connected to a small number of nodes in the next layer based on some neighborhood. Additionally, the weights of each neighborhood are tied, that is, they are forced to be the same for all neighborhoods. If the input layer is an image, the output of a convolutional layer can be visualized as the convolution of the input image with a kernel where the kernel weights are the model parameters to be learned. These convolutional layers are then stacked on top of each other to form a deep convolutional network.

In the GoogLeNet model [ref] we used, there are nine inception layers [ref] (figure #) each of which contain three convolutional layers at varying scales and one max pooling layer. Between the second and seventh inception layers, there are max pooling layers to reduce the spatial resolution of the resulting maps and allow for more kernels to be learned. Finally there is an average pooling and drop-out layer before a fully rectified layer connecting to the output softmax layer (figure #).

2.2. Figure

inception layer

3. HIERARCHICAL CLUSTERING

Hierarchical clustering is a technique that builds a hierarchy or tree of clusters rather than a single set. This tree is also known as a dendrogram (Figure #). The root node is the cluster containing all of the data points, while the children of a node form a partition of that cluster. An advantage of hierarchical clustering is that the computation does not depend on knowing the number of clusters a priori. The two primary approaches to constructing the dendrogram are called agglomerative (bottom-up) and divisive (top-down). In divisive clustering, all data points start in a single cluster which is recursively split. In agglomerative clustering, each data point is placed in its own cluster and these clusters are recursively merged. The order in which clusters are merged in agglomerative clustering is generally based on some similarity measure, though in practice any objective function can be used [ref].

3.1. Figure

Dendrogram

For the general case, agglomerative clustering requires N^2 similarity measure calculations at each recursive step, where N is the number of clusters. This makes the calculation intractable on large datasets. This problem can be somewhat aleviated by the use of neighborhood or connectivity constraints [ref]. We can represent these constraints as the weighted connectivity graph $G(V, E)$ where V is the set of vertices and E is the set of weighted edges. The vertices of this graph are the clusters while the weights on the edges are the similarity measures between those clusters. The problem at each recursive step then becomes updating G and calculating the similarity measure along each edge of the graph. Ideally, the number of edges $|E|$ is much less than $|V|^2$ which corresponds to a fully connected graph, i.e., no connectivity constraints. For the specific case of nearest neighbor connectivity, each vertice has a single edge ($|V| = |E|$) which greatly improves the scalability of this algorithm to large datasets. The nearest neighbor calculation need not use the same similarity measure as the clustering algorithm. For example, we use an adjacent frame constraint; edges in G are only formed between temporally adjacent frames. This not only considerably speeds up the calculation as $|V| = |E|$ but also results in temporally consistent scenes which is a desirable trait.

4. VIDEO RECOGNITION AND SCENE DETECTION

Performing image recognition on a video is a difficult task due to processing constraints. Without GPU acceleration, classifying a single image takes approximately four seconds. However, video content tends to change slowly over-time only with sharp discontinuities at scene changes. For this reason, we propose only classifying the I-frames of encoded video. I-frames allow for parallel access patterns and are regularly spaced throughout the video. They also tend to be aligned along scene changes because scene changes create large motion vector reconstruction errors. This is helpful because it is then reasonable to assume that the content of the video does not change drastically between I-frames

Image recognition of I-frames was performed using a pretrained GoogLeNet model network [ref]. The network was trained to classify between 1,000 different image types based on hand labaled data from the image-net database [ref]. The video recognition technique was implemented in a parallel cluster environment to speed up processing of large video

files. First, the video file is broken into pieces along I-frame boundaries and distributed to the nodes of the cluster. Then, each node in the cluster begins processing the individual pieces and produces a single 1,000 dimension classification vector for each I-frame. These classification vectors are collected across all the nodes and returned as a single $Nx1,000$ array where N is the number of I-frames in the video. Finally, this array is stored in a database along with the full movie.

After image recognition, the classification vectors are hierarchically clustered to detect scene boundaries and speed up search operations. We performed agglomerative clustering on the I-frame classification vectors in the same parallel cluster environment. Frame adjacency constraints were used to increase clustering efficiency and restrict scenes to be temporally adjacent. Each frame had a single neigher – the next frame in the sequence. We built a custom agglomerative clustering algorithm to maximize computational and memory efficiency for this constraint along with allowing for distributed calculation. Edges in the weighted connectivity graph G were distributed to the nodes in the cluster along with the centroids. Each node than calculated the similarity between all of the edges it was assigned based on simple euclidean distance between cluster centroids. Then, each node returned the minimum distance edge from the group it was assigned. These minimum edges were then collected across all nodes to find the true minimum edge. This minimum edge was then used to updated the weighted connectivity graph G. This procedure was iterated until all clusters were merged into a single cluster. The resulting cluster hierarchy was then saved in the database along with the full movie and the original classification vectors.

5. REAL-TIME VIDEO SEARCH

We developed and implemented a simple video search application to demonstrate the power of the classification vector cluster hierarchy. While viewing a video that has already been processed, a user can type a search term into the search box below the video. This search is then processed by the server using the precomputed cluster hierarchy and video information to return the segments of the video where that search term is likely to appear. The application then updates the progress bar of the video to indicate these segments to the user.

When the text query is submitted to the server, it first makes a text-based query against all of the labels associated with the 1,000 image classes. Each image class is actually associated with many different text labels. For example the 'polar bear' image class has the following image labels associated with it: one, two, three. Using elasticsearch [ref], labels that match the text query are returned along with a score indicating how well they matched.

A search vector is constructed by creating a zero vector of length 1,000 and then setting the value of the vector associated with each returned label equal to the returned score. Finally, the length of this vector is normed to one. For a particular number of clusters in the hierarchy (configurable in the application), the inner product of the search vector is taken with the centroid of the cluster. Every scene whose inner product with the search vector exceeds some threshold (also configurable in the application), that segment of the video is returned as a probable match.

6. RESULTS

As an example, we can examine the resulting classification vectors from the images in figure #. To better visualize these classification vectors, we have constructed word-clouds. The words come from the first text label associated with each image class and the size of the word corresponds to the value of that class. In other words, the larger the label the more likely it is the correct image class. Although the word-clouds are not perfect, we can see that the network is correctly identifying both a fox and a dog in the images. It does appear to be having trouble separating the two objects, and there are many image classes which are clearly wrong yet are still considered highly likely to be the correct class by the network.

6.1. Example word-clouds

We can also explore an example cluster hierarchy or dendrogram of these six classifcation vectors. Looking at the dendrogram makes it appear that there may be two distinct scenes – the first three frames and the last three frames. This seems reasonable when looking at the classification vectors as the first three seem to pick up on the fox more while the last three seem to pick up on the dog more. However, when viewing the images themselves it seems clear that all six images are from the same scene and both animals are present in all of the pictures.

6.2. Example dendrogram

We can also visualize the cluster centroids of the first three and last three frames. It appears that the fox has become more pronounced in the first cluster and the dog has become more pronounced in the second cluster, while the more spurious responses have been reduced.

6.3. Example clustered word-clouds

The interface for the real-time video search application is shown. The first episode of the BBC series Planet Earth is playing and the search bar, as well as the cluster count and threshold value, are also visible.

6.4. Show website

Here we show the resulting search vector for the text search 'polar bear'. The search vector has been represented as a

word-cloud for convenience. The search vector correctly includes the polar bear label, but it also includes several other bear labels to a lesser degree.

6.5. Show example search-vector

Now we show the resulting highlighted progress bar for both 'polar bear' with two different cluster settings. For the small cluster setting, the grouping of polar bear scenes is much better and there are fewer false positives.

6.6. Polar bear scenes for two cluster settings

We are also showing the resulting highlighted progress bar for 'shark'. Although we can see that the the largest cluster associated with the 'shark' label does correctly contain a shark, there are many more false positives throughout the video even for small cluster settings.

6.7. Shark scenes

The search query takes on average 0.25s to process by the server, and the highlighted progress bar is visible on average 0.5s after submitting the query. There is very little dependence on the cluster and threshold settings as the clusters are all pre-computed.

7. CONCLUSION

Scenes based on similar content rather than similar images (e.g., jump cut between two horses at different times of day could still be considered a single scene).

Where it fails

The output of the image recognition algorithm also tells us what types of objects are likely in the scene. Eventually, we want to detect and track these objects within and across scenes to gain a broader understanding of the video such as ... [refs]. However, the state of the art object detection algorithms are not yet as robust as those in image recognition.

How to fix

Future work and applications

Search vectors could be constructed from template images. Allows to search for similiar images in a video rather than text.

Instead of searching within a video, could be used to search across databases of videos based on content, e.g., for YouTube or Vine. Probably just want to use a single or few clusters per video in this case.

Could be used in conjunction with quality assessment algorithms to better account for content-based opinion score biases.

8. REFERENCES