Visual-Tex: Video Tagging using Frame Captions

CS229 Final Project Report, December 16, 2016

Kat Gregory, Dylan Moore, and Nick Troccoli

***Abstract*— In this paper, we developed a simple yet effective method for video classification. Our specific task was labeling video clips from Hollywood movies with one of twelve action tags. We generated captions for a sampling of still frames from each clip and then compared the performance of 58 combinations of feature extractors on these captions and classifiers on these features. We achieve over 42% accuracy.**

Possible applications of our system include indexing uncaptioned video on sites like YouTube as well as video retrieval, video surveillance, and making video content more accessible to people who are visually impaired.

I. Task Definition and Project Goals

Every minute, three hundred hours of content are uploaded to YouTube (Brouwer). As video technology becomes ubiquitous and every cell phone represents a portable way to capture yet another snippet of the rich audiovisual world, we gain access to a mind boggling amount of video content. This translates to an unprecedented amount of information at our fingertips, yet unlike text content, which can be automatically searched and analyzed, most information about videos comes from manual categorization and summarization, or from non-descriptive closed captions. We must develop ways of automatically processing video information for it to be more useful and accessible.

For our project, we built a machine learning algorithm that takes a video clip as input and, using existing image captioning software, captions freeze frames within that video and uses these captions to output a predicted video tag. In the end, this auto-categorization of content rather than relying on pre-provided captions or other manually-inputted textual information will be immensely useful for indexing uncaptioned video on sites like YouTube as well a for video retrieval, video surveillance, and making video content more accessible to people who are visually impaired.

II. Related Work

Darin Brezeale and Diane Cook's "Automatic Video Classification: A Survey of the Literature" overviews a broad array of different papers and suggests that existing approaches to video classification fall into four categories: text-based, audio-based, visual-based, or a mix of these. With regard to classification itself, standard classifiers like Bayesian, SVMs, and neural networks have performed well, as have the less common Gaussian Mixture Models and HMMs. We focused our research on text based approaches, as our own approach falls into this line.

Text-based approaches are least common of all four approaches. Although there is a great deal of research on text classification, text-based approaches to video classification have only focused on text that is viewable in the video or provided via video transcripts, featurized using a "bag of words" model or TF-IDF, which weights each word by its relation to the task at hand. This text is of limited use: viewable text suffers from the high error rate of OCR, and transcript text concerns primarily dialog and thus misses much of the available information. In addition, feature vectors from text-based approaches have high dimensionality and are thus harder to work with.

As Rohrbach et al. noted in their paper "A Dataset for Movie Description," movie scripts are not always aligned temporally to the film and sometimes differ from what is actually said, and they fail to capture all of the unspoken information present in a scene. Audio descriptions of movies for the visually impaired do note visual information about the scene but are rare because they require a significant amount of time to create. In both cases, using features from these textual inputs would restrict us to considering film that had been captioned or that had an available script.

Despite the above described limitations of automatic speech transcription, Taskiran, Cüneyt M., et al. note in their paper "Automated video program summarization using speech transcripts" that automatically transcribed human speech from video alone can be used produce reasonably accurate video summaries. Although many video captioning models focus only on the visuals, most likely due to the difficulties of incorporating visuals and audio, Taskiran, Cneyt M., et al.'s findings suggest that combining these two data streams, perhaps via many levels of feature extractors (e.g. in the audio case, a pipeline of speech to text, followed by the NLP text analysis feature extractors that Taskiran uses) will (perhaps always, under known boundary conditions) ultimately lead to a more accurate model.

Yu, Haonan, et al., in their paper "Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks," explore another approach to this same problem of video captioning, which seems to bear incredible promise. Yus team developed an RNN that can caption a short segment of video with a sentence description. Then, they split a video into segments (similar to how we looked at freeze frames) and caption specific intervals. Finally, they use a hierarchical model to combine these sentences into a coherent description. In many ways, their approach is similar to how we would envision modifying this project if we were to tackle the problem of generating a synopsis of a video.

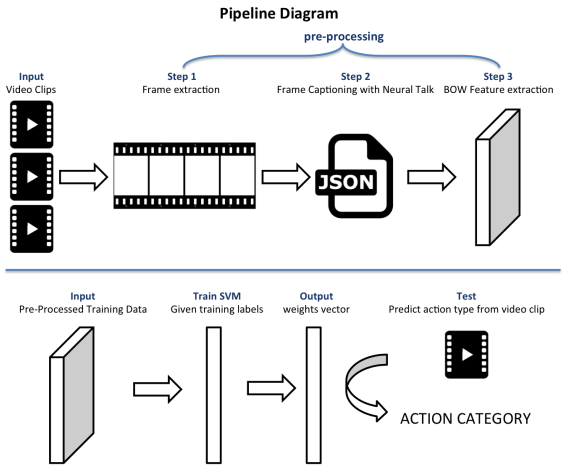

Fig. 1: System pipeline for a single feature+classifier combination.

A. Approach

We focused on Brezeale and Cook's observation that one of the main flaws of text-based approaches is that they are restricted to dialogue and thus cannot use any of the visual data present in a scene. We asked ourselves if there was a way to utilize visual information in text format by leveraging the power of existing image captioning programs - notably NeuralTalk, a recurrent neural net developed by Andrej Karpathy at Stanford, which creates captions for still images. Our approach is detailed in Figure 1. Specifically, given a video clip, we extracted a sampling of frames and generated a NeuralTalk caption for each. Then, we used the captions from all the sampled frames to select an action tag for the clip from twelve available action tags.

III. Infrastructure

A. Data

We used the Hollywood2 Human Action and Scenes Dataset developed by Ivan Laptev, Marcin Marszaek, Cordelia Schmid, and Benjamin Rozenfeld, which contains 2,517 short clips from 69 Hollywood movies. Each video clip is labeled with one or more of the following action tags: "AnswerPhone", "DriveCar", "Eat", "FightPerson", "GetOutCar", "HandShake", "HugPerson", "Kiss", "Run", "SitDown", "SitUp", or "StandUp". A total of 229 clips were tagged with multiple actions: for these clips, we randomly selected a single action tag.

B. Preprocessing

We preprocessed these clips by generating captions of freeze frames for each. To do this, we relied on NeuralTalk, an existing image-captioning program, and our own script written in Processing to, for each video, extract 24 evenly-spaced frames and feed each frame into NeuralTalk to generate a caption. We used a constant number of frames so that each clip, regardless of length, would have the same number of captions generated. These captions are sentences or phrases, such as "a woman is holding a cell phone in her hand" or "a city street at night with a traffic light." To configure NeuralTalk, we used pre-trained weights because image processing is not the primary focus of our project. The end result was a JSON file for each video containing an array of its 24 NeuralTalk captions. These were the preprocessed inputs we provided to our algorithm.

IV. Methods

A. Harness

Once we had a set of captions for each of the movie clips, we constructed a harness that allowed us to easily run and compare many different feature-classifier combination experiments. We adapted this harness from one written by Max Wang for a CS221 project with Kenny Xu and Kat Gregory. This harness was critical to the success of this project because it made our program completely modular and allowed us to efficiently run tests, tweak parameters, and find bugs. The harness includes options to change the number of clips to use (defaults to all), the percentage of these clips to use for training (defaults to 80%), the number of cross fold validations (defaults to 5), and the number of actions tags to consider (defaults to all).

B. Feature Extractors

We concatenated the captions from each movie and then compared the performance of six different NLP feature extractor: "bag of words" (regular and binary), N-grams (for N=2, 3 and 4), and TF-IDF. We give a brief description of each feature extractors below.

1) Bag of Words: A "Bag of Words" represents the captions as a frequency list with the number of times each word appears in a video's captions. Binarized "bag of words" simply stores whether or not each word appears in a video's caption data without taking into account its frequency.

2) N-Grams: N-grams store groups of up to N consecutive words that appear in a video's captions which, unlike "bag of words", preserves information about word ordering within captions. We used 2-grams, 3-grams, and 4-grams for our project and did not go higher than 4 due to concerns about overfitting our data for specific captions.

3) TF-IDF: Finally, TF-IDF normalizes the raw count frequencies by multiplying TF (term frequency) times IDF (inverse document frequency). This increases the prominence of terms whose presence or absence reveals a great deal of information about which action the clip shows, while decreasing the prominence of terms that are common across all clip captions.

C. Classifiers

We tested these feature extractors against eleven different multinomial classification algorithms. For each of these algorithms, we used a standard implementation from the sklearn toolkit for python. Because the focus of this project was to identify in general the most promising avenues for further

exploration, we used standard hyperparameters for each classifier and left experimentation with these hyperparameters to future research. We give a brief description of how each classifier works below.

1) Naïve Bayes: A Naïve Bayes classifier applies Bayes' theorem with the naïve assumption of independence between every pair of input features.

2) Adaptive Boosting: Boosting algorithms work by fitting many simple classifiers to the dataset. These classifiers are designed to have complementary weaknesses (i.e. each new classifier prioritizes correctly classifying data missed by its predecessors) so that they can be weighted and combined to minimize the overall number of misclassifications. For our specific AdaBoost implementation, we used an algorithm known as AdaBoost-SAMME.

3) Gradient Descent: The gradient descent algorithm learns a good classifier (i.e. one that minimizes loss) by making iterative updates to a weights vector using the gradient of a chosen loss function on the dataset. For our loss function, we chose to use hinge loss because it is a standard loss function for gradient descent. Because it is more efficient, we chose to use stochastic gradient descent, which means that we updated our weights after each training sample (as opposed to one update for each pass over the entire set of training data).

4) K-Nearest Neighbors (KNN): K-nearest neighbor works by finding a predefined number of training samples, in our case k=12 (one for each action), that are closest in distance to the new point, and predicting the label from an average value of these.

5) Logistic Regression: Logistic regression predicts the probability of particular outcomes using coordinate descent. Since our project required more than binary classifications, we used an implementation of multinomial logistic regression that decomposes the multinomial problem in a "one-vs-rest" fashion so that separate binary classifiers are trained for all classes.

6) Multiclass One Vs. Rest using Linear Regression: As noted in the implementation for multinomial logistic regression, multiclass one vs. rest works by fitting one classifier to each particular classification label. For each classifier, the chosen class is fitted against all the other classes. In effect, this classification model just extends linear regression (i.e. fitting a binary linear classifier to a dataset) to the multinomial case.

7) Decision Tree: A decision tree works by creating a model that predicts the classification label of a data point by learning simple decision rules inferred from the training data. A good example of a simple decision rule, which can be generalized from a 1-dimensional case to fit our feature space, would be an if-else statement.

8) Random forest: The random forest classifier fits a number of decision tree classifiers on various sub-samples of the dataset.

9) Extra Trees: Our extra trees classifier creates many randomized decision trees on various sub-samples of the dataset and averages over these to provide a single classification. Using many decision trees helps prevent overfitting and, we found, seems to capture more nuances of the problem from the training data. Specifically, the main differences between extra trees and random forest is that for our extra trees algorithm when choosing variables at a split, samples are drawn from the entire training set instead of a bootstrap sample of the training set. Additionally, splits are chosen completely at random from the range of values in the sample at each split.

10) SVM: SVMs operate by finding a separating hyperplane that maximizes the margin between classes.

11) Voting: The ensemble "voting" classifier is made up of a combination of the three independently best performing classifiers: Random Forest, SVM, and Naïve Bayes.

V. Experiments

A. Cross Validation

We evaluated using 5-fold cross validation, which gives a more robust measure of predictive power than does a single trial. For each fold, we trained our classifier on 80% of the clips in our dataset and then evaluated its ability to classify the remaining 20%. Each fold used a different 20% of the data as the test set.

B. Evaluation Metrics

We evaluated the performance of each feature extractor+classifier pair on two metrics: precision and efficiency. Precision is the fraction of predicted tags that are accurate. We considered both the average precision over 5 folds and the best accuracy over all observed folds (results shown in Figure 2). In addition, we measured efficiency as the average computation time each experiment required. However, because video tagging need not happen in real time, we decided that accuracy was more important to our task than efficiency.

C. Actions Considered

Although we could achieve much higher Precision rates by limiting the number of action tags considered (classifying over only three of the most common actions, "Run," "Kiss," and "DriveCar," resulted in an precision of close to 80%), we felt that it was more interesting and relevant to classify over all twelve action tags.

VI. Results and Discussion

A. Performance of Different Models

The average and best precision for each feature extractor + classifier model are given in Figure 2. We consider these results by first comparing the performance of different feature extractors and then comparing that of different classifiers. Note that, given that there are twelve action tags, random chance gives 8.33% precision.

There is not a significant degree of variation between different feature extractors. Quad-grams perform best, achieving an average of 34.5% precision over all classifiers. Given that "Bag of Words" is equivalent to 1-grams, that precision increases smoothly over N-grams from N=1 to N=4 suggests

average (best)	bow	binary_bow	2grams	3grams	4grams
adaboost	28.8% (31.61%)	-	29.7% (32.41%)	28.4% (32.41%)	28.5% (31.61%)
bayes	32.2% (35.59%)	-	33.9% (35.98%)	36.6% (40.95%)	37.2% (34.59%)
dtree	27.5% (30.02%)	26.6% (26.64%)	27.6% (30.22%)	28.0% (30.22%)	27.9% (30.02%)
extratrees	34.6% (35.39%)	30.0% (30.02%)	33.8% (37.57%)	34.6% (36.98%)	34.9% (36.18%)
gradient_desc	27.5% (32.21%)	37.0% (36.98%)	28.7% (33.80%)	29.5% (34.39%)	32.6% (35.39%)
knn	32.2% (36.58%)	-	31.6% (36.98%)	32.8% (35.59%)	32.8% (36.98%)
logistic_rgss	34.6% (37.97%)	-	32.8% (34.79%)	34.6% (39.36%)	36.2% (41.95%)
multiclass	30.5% (32.60%)	-	28.5% (30.82%)	29.7% (37.57%)	32.5% (38.37%)
rforest	38.8% (40.95%)	-	39.2% (41.75%)	38.9% (41.95%)	41.2% (42.15%)
svm	38.0% (41.95%)	-	40.0% (41.75%)	35.6% (35.19%)	32.6% (22.66%)
voting	33.0% (36.98%)	-	34.0% (36.18%)	36.6% (41.15%)	37.3% (39.76%)
Overall	32.9% (36.02%)	31.2% (31.21%)	33.0% (35.98%)	33.7% (37.34%)	34.5% (35.81%)

Fig. 2: Precision for each feature+classifier experiment. Each cell represents "average precision over 5 folds (best precision over 5 folds)". Random performance would be 8.33%.

that the phrase structure of captions adds context that is important for action prediction. However, using the least helpful feature extractor, binary "bag of words", results in only a 3.3% decrease in precision.

The 10.7% range in precision between different classifiers is far more dramatic. The Random Forest classifier achieved an average of 39.4% precision and a max of 41.91% precision. SVMs also perform well at 36.3% accuracy, closely followed by Voting and Naïve Bayes. While our literature review suggested the effectiveness of SVMs for video classification tasks, we were surprised by the standout performance of the Random Forest classifier. We believe that this may reflect Random Forests' ability to cope with some of the weaknesses of our dataset. Random Forests are known to do well with small datasets, and although we used over 2,500 clips in our analysis, this averages out to only just over 200 clips per action tag. Further, Random Forests are robust to datasets that are not balanced over the difference classes, and the number of clips per action in our dataset ranges significantly from from 84 (for "SitUp") to 463 (for "Run").

In terms of efficiency, we noticed a significant range in processing time over different features and classifiers. While simple classifiers ran in a matter of seconds (Naïve Bayes, Gradient Descent, KNN, etc.), others took hours (in particular, Voting, Adaptive Boosting, and Multiclass). Voting was the most computationally expensive classifier and 4-grams the most expensive feature extractor: in combination, this model required around 34,650 seconds to run 5 folds. In contrast, Random Forest, our most successful classifier, ran 5 folds in the respectable range of 3 seconds (for "Bag of Words" features) to 90 seconds (for 4-gram features).

B. Performance on Different Actions

Looking at the heatmaps, which display per-action accuracy, we get some additionally interesting results. First, as the cumulative heatmap in Figure 5 shows, overall our project did a fairly good job correctly predicting for each category. Ideal results would have strong colors only on the diagonal of the heatmap, indicating that each category was correctly

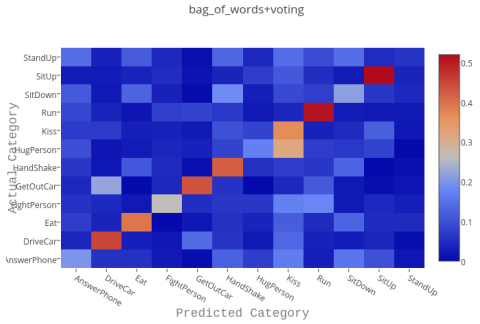

Fig. 3: Heatmap of action specific accuracy for "Bag of words" and our voting classifier.

predicted 100% of the time. A few standout performers, both good and bad, was the "Run" action, the "Kiss", and "HugPerson" actions, and the "GetOutCar" and "DriveCar" actions.

The "Run" action was by far the most accurately-predicted action of the 12 actions in our dataset, which we believe is both because it is the most common action in our dataset and because "Run" is an action that has identifying entities that overlap very little with the other actions. For instance, running is commonly associated with outdoor landscapes and blurred motion. The outdoors aspect in particular is one that may help dramatically improve detection of "Run" vs. other actions, since no other actions in our dataset are as tightly associated with outdoors.

The "Kiss" and "HugPerson" actions were two of the more easily conflated categories, which we predict is due to the ambiguous entities defining affectionate actions. For instance, it could be that an embrace is also a part of a kiss, or a kiss is also a part of an embrace. We believe that fewer clearly-identifying entities made it more difficult for our algorithm to distinguish between the two.

The "GetOutCar" and "DriveCar" actions faced a similar conflation problem. We predict this is because of the presence of a car as a distinguishing factor for both, but the lack of

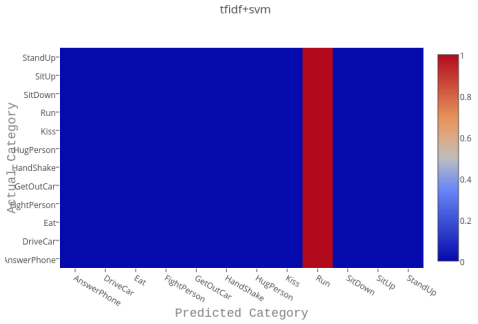

Fig. 4: Heatmap of action specific accuracy for one (particularly disappointing) trial of SVM+TF-IDF.

temporal information (i.e. is the person getting out of the car or not) makes it difficult to further distinguish between the two. This was also a problem we noticed in other poorly-performing categories such as "SitDown" and "SitUp", both of which are likely to be difficult to predict using only freeze frames. This is because in both cases, it is the progression between frames that identifies what action is taking place.

Overall, these results indicate that actions with more associated entities or scenes, such as cars, the outdoors, or embraces, are easy to detect. However, actions with only a slight nuance distinguishing between them, such as "HugPerson" and "Kiss" or "SitDown" and "SitUp", pose challenges to our algorithm because of the lack of temporal information.

We also investigated heatmaps for individual feature extractor-classifier combinations. One particularly strong combination was the "Bag of Words" feature extractor paired with the voting classifier, the results of which you can see in Figure 3. Notice high probabilities along the diagonal, indicating accurate action predictions.

C. Other Performance Notes

It is interesting to note that the feature extractor+classifier pairs that achieve the highest precision are not always the same as those that have the strongest diagonals in the actions heatmaps. This is a result of the unequal distribution of actions in our dataset. While the heatmaps normalize for the number of clips that correspond to each action, the precision metric weights performance on each clip equally. As a result, a feature extractor+classifier pair can achieve significant results by only doing well on common actions. For example, one trial of the svm+tfidf model achieved 17% accuracy by nearly always guessing the most common action tag, "Run" (see Figure 4).

We noted that the cumulative performance of all of our models improved with increased dataset size. Figure 5 demonstrates the increase in cumulative precision over all feature extractor+classifier combinations when we increased our dataset size from 1,000 clips to the full 2,517 clips.

VII. Conclusions and Future Work

Over 58 different feature extractor+classifier models, we saw the highest precision from high N-gram feature extractors and from the Random Forest classifier. There are two significant implications of our results. The first is that features that capture more of the structure of a caption seem most useful: as we increase N from one to four, the precision of the resulting model increases too. We saw that our model struggles most with actions that involve a time-based sequence of steps, like "StandUp" or "SitDown". Therefore, we believe that increasing the structure of our features even more to take into account not just the order of words within each caption but also the sequential order of the captions themselves is a promising avenue of exploration. For example, we would like to try using an N-gram equivalent with words that appear in order across captions rather than just sequentially. The second result is that, given the standout performance of the Random Forest classifier and the fact that that classifier in particular does well on small datasets, we would expect a larger dataset to improve the other classifiers. Overall, with features that account for caption sequence, a larger training set, and possibly other techniques such as entity extraction and training NeuralTalk on our own task-specific data instead of using pre-trained weights, we believe that our frame captioning approach would provide a promising way to categorize videos.

The original motivation for our project was to generate text summaries of videos based on captions of frames within those videos. Looking forward, while the NLP involved was beyond the scope of this project, now that we have explored a simpler classification task we would like to return to this objective and create a NeuralTalk equivalent for video.

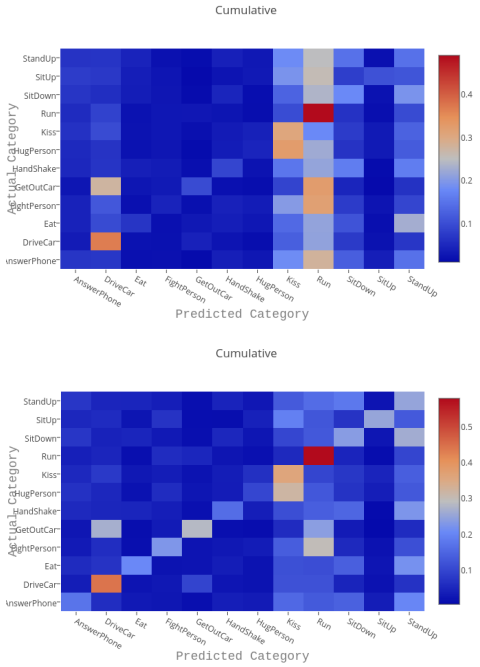

Fig. 5: Heatmap of cumulative action specific accuracy with a dataset size of 1000 (top) and 2517 (bottom).

REFERENCES

[1] Brezeale, D., and D.j. Cook. "Automatic Video Classification: A Survey of the Literature." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38.3 (2008): 416-30. Web. fea

[2] Brouwer, Bree. "YouTube Now Sees 300 Hours Of Video Uploaded Every Minute." Tubefilter. N.p., 01 Dec. 2014. Web. 20 Nov. 2016.

[3] "Learning Action from Movie Datasets." Visage Technologies Blog. Visage Technologies, 11 Sept. 2015. Web. 20 Nov. 2016.

[4] Marszaek, Marcin, Ivan Laptev and Cordelia Schmid. "Actions in Context." IEEE Conference on Computer Vision & Pattern Recognition. 2009.

[5] Rohrbach, Anna, Marcus Rohrbach, Niket Tandon, and Bernt Schiele. "A Dataset for Movie Description." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015). Web. 20 Nov. 2016.

[6] Yu, Haonan, et al. "Video paragraph captioning using hierarchical recurrent neural networks." arXiv preprint arXiv:1510.07712 (2015).

[7] Taskiran, Cneyt M., et al. "Automated video program summarization using speech transcripts." IEEE Transactions on Multimedia 8.4 (2006): 775-791.

[8] Chapdelaine, C., et al. "Improving video captioning for deaf and hearing-impaired people based on eye movement and attention overload." Electronic Imaging 2007. International Society for Optics and Photonics, 2007.

[9] Shetty, Rakshith, and Jorma Laaksonen. "Video captioning with recurrent networks based on frame-and video-level features and visual content classification." arXiv preprint arXiv:1512.02949 (2015).

[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.