# Video-Based Actuvity Search with Siamese Sequence Models

Milind Jagota, Erik Jones, Zhaoyu (Joe) Lou

***Abstract*—We present a video clip search platform which, given a short caption, returns some subset of one of a large number of videos, or "clip", corresponding to that caption. Both the caption and the video frames encode different types of semantic meaning, so all that is needed is some space in which clips and captions may be compared and a mapping from the clip and caption spaces to that space. We train two distinct GRUs in a Siamese network configuration to learn the semantic embedding space mappings for the videos and captions on data obtained from the ActivityNet dataset. We use a triplet-based approach for one shot learning. After training, we obtain a top-20 percent performance of 99.5 percent on a validation set. Our evaluation is based on the search ranking of the ground truth clip for a caption, relative to all other clips. Our search ranking is based on the Euclidean distance between the caption and each clip, in the shared embedding result. Code for the project is located at https://github.com/ejones313/clip_search, along with a demonstration. This result demonstrates the feasibility of successful clip search engines in the future.**

## I. Introduction

Imagine you're giving a talk, and you've just finished preparing the entire presentation. However, the slides are a little bit dull, so to spice them up you plan to tactically add some GIFs. Unfortunately, after a somewhat painful search on the internet you finally succumb to the sad reality that there simply isn't a feasible platform to find relevant GIFs. Video search, more generally, is typically done using user provided tags and viewing numbers. Purely content based search is highly desirable for large collections of videos, such as GIFs, where user provided information might be sparse.

To address this problem, we present clip-search, a deep-learning system for finding relevant short clips just given text. The input to our system is a short caption, comprised of a variable number of individual words. For example, if one were giving a presentation on squash, "playing squash" might be an appropriate search. The system then converts the caption into a series of word embeddings using a pretrained model, FastText [1], then maps the embeddings into a learned "embedding space" using the output of the final hidden layer of a gated recurrent unit (GRU). Next, a clip, or series of frames, is converted into a sequence of frame embeddings using a convolutional neural network (CNN), which is then mapped to the same embedding space using a new GRU. The system then uses a distance metric to compute how similar the caption is to each clip, and sorts the clips based on similarity. The output of our system is technically this entire sorted list of clips, but in practice normally just the closest clip or closest few clips are returned.

## II. Related work

Our architecture is heavily influenced by previous work done on the similar task of person re-identification, where the identity of the person is the analogue of the semantic content of a video or caption. The siamese network architecture was popularized by Schroff et al, who used it as an efficient alternative to previously used metric learning methods. [2] The simplicity of the distance metric used in the siamese network method made it an attractive alternative, and it quickly caught on. The validity of the triplet method was further demonstrated by the work of Parkhi et al, who found that a deep convolutional network trained for person ReID performed considerably better when trained using triplets than it did when using metric learning and pairwise contrastive loss. [3] Hermans et al explored the problem of triplet selection and compared the hardest-triplet and all-hard triplet approaches, and introduced their own modified triplet mining apporach, the "batch-hard" approach, which is similar to the lifted-embedding approach presented by Song et al. [4] [5] Efficient triplet mining is explored by Amos et al in their blog post discussing their OpenFace architecture, and how they halved their execution time by using an online triplet mining method. [6] Our own online triplet mining procedure is motivated by their work.

## III. Dataset and Features

We used the ActivityNet Dataset for our project, as well as the results of a Stanford Vision Lab research project that was conducted on the ActivityNet Dataset. [7] [8]

The ActivityNet Dataset is a collection of time localized visual features for over 10,000 videos of activity and sports. The dataset is generated by passing the frames of each video through AlexNet [9], extracting the activations of the second fully-connected layer of the network, and applying PCA to reduce the dimension to 500. One 500 dimensional vector is generated for every 16 frames of the original video. Importantly, the frame rate of the videos was not constant - implications of this are considered in the discussion section. This dataset thus provided us with a sequence of 500 dimensional vectors for each video describing visual features over time.

The second half of our dataset was collected in a research project conducted by Ranjay Krishna et al of the Stanford Vision Lab. [7] They trained a deep learning system to automatically annotate videos from the ActivityNet dataset using human generated captions collected through Amazon MechanicalTurk. These captions are localized to specific

parts of each video, and a single video had on average about three captions corresponding to unique parts of the video. We preprocessed the human-generated captions by passing each word of each caption into Facebook's FastText model. [1] This turned every caption into a sequence of 300 dimensional vectors. An example of a video from the dataset along with its captions is shown below:

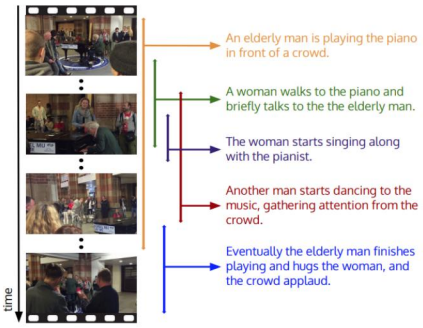


Fig. 1: Video from ActivityNet Dataset with captions [7].

Our processed dataset thus consists of information for 36000 clips. For each clip, we have a sequence of 500 dimensional vectors representing visual features and a sequence of 300 dimensional vectors representing word embeddings of a caption that describes that clip. Because of constraints on computational time, we weren't able to use the entire dataset for training. We instead used a training set of 10000 clips and a dev set of 1000 clips.

## IV. Methods

The entire system, including the preprocessing, is contained within the following figure:

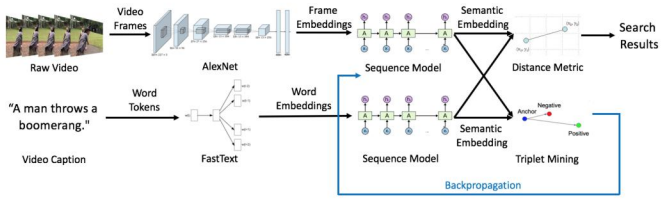


Fig. 2: The modified Siamese architecture

The overall goal of our model is to learn a mapping from videos and captions to semantic embeddings such that we can directly compare the semantic content by comparing embeddings. Our model is a modified version of the Siamese network model used for person ReID. The raw video and captions are first converted to sequences of word and frame embeddings using the preprocessing as described in the previous section. Each sequence is then passed into its corresponding recurrent architecture. Note that the two recurrent models are distinct models which do not share parameters, unlike other Siamese sequence models, owing to the fundamentally different input data to both. We set the hidden dimension for each sequence model to be the same (400), and take only the output of the final hidden state of the recurrent model as a semantic embedding for the video and caption. Given these embeddings, we then use a triplet based training procedure to train the network.

We discuss the details of the different recurrent architectures we tested in our system, some important details of the triplet loss we used to train, as well as the implementation of the final search system post training.

### A. Recurrent architectures

We use a recurrent architecture to map each type of embeddings to the shared embedding space. The use of recurrent architectures for this task is appropriate since our goal is to output a single embedding vector for the whole clip or caption from a sequence of word and frame embeddings. We test out four different recurrent architectures: the vanilla recurrent neural network (RNN), the gated recurrent unit (GRU), the long short-term memory (LSTM), and a bi-directional LSTM.

*1) RNN:* Previous elements in the sequence are critical tool in making a prediction at a given element in a sequence. Unfortunately, standard fully connected nerual networks don't incorporate past outputs, which is why the RNN rose to prominence. The simple RNN's unit is comprised of a single

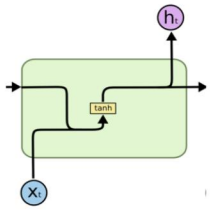


Fig. 3: A single unit of a vanilla RNN [10].

tanh activation, and parameters are shared between units. While RNNs typically preform better than fully connected networks on sequential tasks, they suffer from vanishing and exploding gradients.

*2) LSTM:* There are two major upgrades of the LSTM over the standard RNN. First, LSTMs solve the vanishing and exploding gradient problems that plague vanilla RNNs by introducing a "memory" channel, which is passed as input to each cell along with the previous hidden unit. In this way, the LSTM may hold on to long-term important information, while still being influenced by recently processed cells. The LSTM

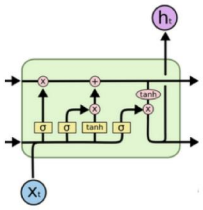


Fig. 4: A single unit of an LSTM [10].

introduces two gates to effectively implement this dual input approach. First, the LSTM has a "forget gate", determining to what extent the past hidden output and memory should be used to make the current prediction. Next, the LSTM has an "input

gate", which is used to regulate how much the current cell may impact the memory channel. The combination of these gates and the two input channels have led to great LSTM performance, especially for longer sequences, relative to the vanilla RNN.

*3) Bi-Directional LSTM:* One significant limitation of an LSTM is that predictions may only be made using past elements of the sequence—future words or frames cannot have an impact. A bi-directional LSTM attempts to solve this problem by having two parallel structures, one going in each direction. Though this is very task dependent, in practice bi-directional LSTMs rarely make significant performance gains compared to regular LSTMs, and impose a fairly high computational cost.

*4) GRU:* Lastly, we present the GRU. The GRU was largely based on the LSTM but, unlike an LSTM, it has just one channel through which information is passed between cells, making it significantly faster to train. To adaptively capture dependencies of different time scales, the GRU has two gates: a reset gate and an update gate. The reset gate is used to determine to what degree past states will influence the current state, while the update gate determines how much the current state impacts the input to the next state. There has been an

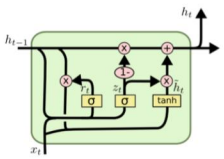


Fig. 5: A single unit of a GRU [10].

increasing trend towards GRUs over the LSTM, due to their comparable performance with reduced training time, allowing for faster iteration.

### B. Triplet Training

Training is done using a modified version of the triplet loss presented by Schroff et al. [2] A single training example consists of a triplet of three examples - an anchor, a positive, and a negative. We modify the original triplet loss to incorporate our two different types of input data. In our case, the anchor is either a caption or a video. If, for the sake of concreteness, the anchor is a video, then the positive example will be the corresponding caption for that video, and the negative example will be another non-matching caption. The case where the anchor is a caption is analagous. The loss function is defined as

$$L = \sum_{i=1}^{N} \Big[ ||f_i^a - f_i^p||_2^2 - ||f_i^a - f_i^n||_2^2 + \alpha \Big]_+$$

where $f_i^a$ is the semantic embedding for the $i$-th anchor, and $\alpha$ is a hyperparameter, the margin. This loss function is minimized when the positive is closer to the anchor than the negative by a margin of $\alpha$ for all triplets. Achieving such an objective would signify that the learned mapping from videos and captions to semantic embeddings is indeed on in which the Euclidean norm captures semantic similarity.

### C. Online Triplet Mining

The triplet based training procedure we used has shown convergence and efficiency issues in previous work, owing largely due to the problem of triplet selection. [4] The number of possible triplets scales quadratically with the number of training examples, generating an abundance of data to train on. However, as training proceeds a majority of these triplets become "easy" triplets, where the network already maps the positive example to be much closer to the anchor than the negative example and thus has zero loss. No learning comes from these triplets, dramatically decreasing the efficiency of the training procedure. Among the remaining "hard" triplets, the research community remains divided on the question of which ones to use as training examples. Some groups have advocated using only the hardest triplets in a given batch, whereas other groups have opted for using all hard triplets or a random subset of the hard triplets. On the recommendation of our mentor, Olivier Moindrot, we decided to use all hard triplets. For additional efficiency, we implemented an online version of triplet mining introduced by Amos et al, in which a large batch is uploaded to the GPU, converted to semantic embeddings, and then mined for all hard triplets in realtime. This saves the time of having to make additional forward passes to get semantic embeddings for constructing triplets.
It is worth noting here that our later results suggested that using all triplets, rather than all hard triplets, and either removing the ReLU function in the loss or setting a very high margin may lead to further improved results.

### D. Efficient Implementation

After training, we were able to decouple the two branches of the siamese network to get a more efficient implementation for evaluation. Given a dataset of videos, we initialize the system by passing all video clips through the video-handling branch of the network, getting the semantic embeddings for all clips. We save this as a matrix. At test time, we pass the query caption through the caption-handling branch of the network to get its semantic embedding. We then use numpy broadcasting to get the Euclidean norm of the differences between this embedding and all the clip embeddings, and return the video with the smallest norm. This efficient implementation allowed us to search a dataset of 36,000 video clips in under a second.

## V. Experiments

Our discussion of experiments will focus on two primary components: the hyperparameters we used and the two different evaluation metrics.

### A. Hyperparameters

We decided to optimize four hyperparameters: the hidden dimension of our recurrent architectures, the learning rate, the amount of regularization, and the triplet loss margin. The hidden dimension and triplet loss margin are particularly important to our specific architecture. The hidden dimension determines the complexity of our common embedding space between the video features and captions, while the triplet loss

margin encapsulates the balance between number of triplets used to optimize and the difficulty of said triplets. Due to constraints on computation time, we were only able to test a handful of values for each of the four parameters - limitations of our hyperparameter search are considered in the discussion section. We ultimately settled on a 400 dimensional embedding space, a learning rate of 0.1, a regularization strength of 0.001, and a margin of 2.5.

Unfortunately, there were several hyperparameters we did not get the chance to optimize at all. One prominent hyperparameter was the batch size, which we set to 128. Our intuition for this value was that the bigger the batch size the more direct convergence is, but each batch takes longer to process. The biggest batch size we could get in memory on the GPU was 512, but 128 was the greatest power of two that was computationally feasible. Additionally, we set the number of epochs to 30, since it appeared that the model consistently stopped making progress on our validation metrics before that point, and more epochs again becomes computationally infeasible. To optimize the parameters of each recurrent architecture, we use the Adadelta optimizer, implemented in Pytorch [11], which has been shown to work well for recurrent neural networks [12]. Our code, along with a demo, may be found at https://github.com/ejones313/clip_search.

### B. Evaluation Metrics

Though we used triplet loss to optimize the network, the triplet loss problem is not the one we're looking to solve—our goal is not to determine which video is the better match out of two videos, but rather which video is the best match out of some significantly larger dataset. To measure our performance on this task, we use the percentile metric, $P(x_{\text{clip}}, y_{\text{clip}})$, which is defined as follows:

$$P(x_{\text{clip}}, y_{\text{clip}}) = 100 * \frac{|x_{\text{clip}}| - \text{rank}(y_{\text{clip}})}{|x_{\text{clip}}|}$$

where $x_{\text{clip}}$ is the entire dataset of clips, $y_{\text{clip}}$ is the true positive clip, and rank outputs the position of a clip in $x_{\text{clip}}$ sorted by likelihood of match (i.e. the closest clip to the inputted caption has rank 1). Based on this evaluation metric, top-20 percent performance refers to the percentage of captions in our datasets with percentile at least 80, top-10 percent performance refers to the percentage of captions in our datasets with percentile at least 90, and median percentile refers to the median percentile taken over the entire validation set.

## VI. Results

Our results are presented in Figure 6. We note that the GRU performs at least as well as the RNN, LSTM, and Bi-LSTM with respect to median percentile, and better than all three of them with respect to top-10 percent and top-20 percent, making it the best method. The histogram for the percentiles obtained from the GRU is as follows: We note that on every example in the validation set, we obtained a percentile of at least 75 and almost always significantly higher. The results will be more thoroughly discussed in the following section.

| Model | Top-20% | Top-10% | Median |
|---|---|---|---|
| RNN | 0% | 0% | 52.3% |
| LSTM | 97.3% | 53.8% | 90.5% |
| Bi-LSTM | 94.6% | 41.8% | 89.2% |
| GRU | 99.5% | 56.6% | 90.5% |

Fig. 6: The of each type of recurrent architecture. The GRU preforms best with respect to each presented evaluation metric.

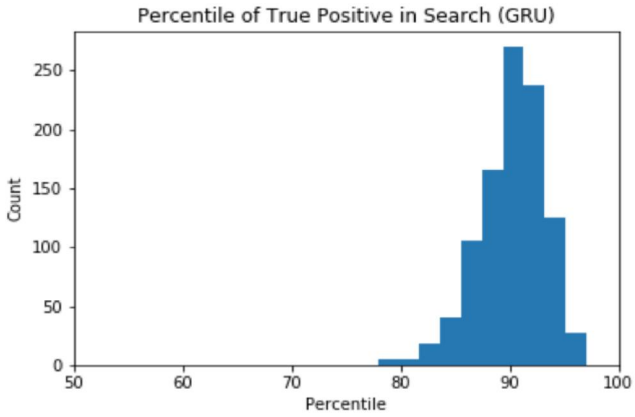


Fig. 7: Histogram of the percentiles of each example in the validation set, using GRUs to map from the word and frame embedding spaces to the shared space.

## VII. Discussion

### A. Evaluating different recurrent architectures

First, it's a good sanity check that the RNN preformed slightly better than random (which would be median percentile of 0.5), but didn't learn much more. This highlights that the task is sufficiently complicated that it requires a network deeper than merely a linear transformation and an arctan.

There are several possible explanations for why the GRU performed better than all of the other models. We will consider in particular the LSTM, which had performance most comparable to the GRU. The primary motivation for an LSTM is that there will be some long-term information worth remembering in addition to short-term information, so two inputs to each hidden unit are necessary. Captions, however, are typically so brief that the long-term input becomes essentially useless, at which point a GRU's speed and optimization to the single hidden input make it a better option. Moreover, for the video clips, while the number of frames is pretty long the amount that actually needs to be remembered between frames is not particularly large—someone playing tennis at frame 1 is probably going to look similar to someone playing tennis at frame 200. In this sense, it's possible that an LSTM would be better suited for general captions as opposed to activity captions, as there's a greater capacity for more long term information.

It was somewhat surprising that the bi-LSTM performed worse than the LSTM, though their performance was comparable. This suggests that the LSTM is definitely the better option going forward, as the bi-LSTM took around 3 times as long

to train. Moreover, the bi-directional LSTM is potentially less suited to video data, where the next frame may be predicted with reasonable accuracy from the previous frame if the separation time is known, using the laws of physics. The laws of physics are not, however, relevant for purely textual data, for which bi-directional LSTMs were designed. One potential caviat here is that the time between included frames is not necessarily constant. Setting it to be constant could be one easy mechanism for improvement.

### B. Problems with Triplet Loss/Evaluation Metric

Based on percentile alone, we were satisfied with the results—it's clear that the model is learning. However, in practice, we want a usable video to appear first—it would be inconvenient to have to process through 100 videos to find one that works (though our demo enables one to do just that). However, observe the model's performance on the query "A man trying to shovel snow":

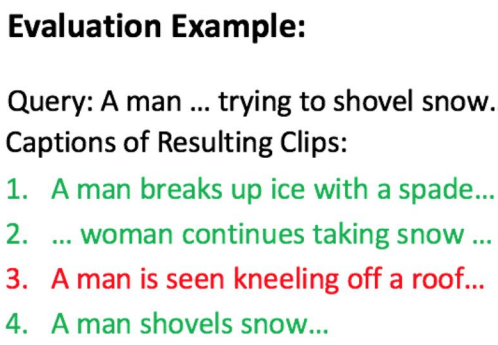


Fig. 8: Example of a query and captions from the clips deemed as best matches. We note that, though the exact caption isn't present, most of these options are good matches in practice.

Here we see that, though none of these 4 correspond to the exact clip we're looking for, it's clear that all but the third contain clips that would correspond to this word. We thus note that our percentile is a strict underestimate for our model's actual performance, and in practice it's still usable since the clips near the top tend ti be good.

On this note, it's not at all clear that triplet loss is best suited to solve this problem. "Good" triplets are those where the positive and negative distance from the anchor is relatively close. Given such a triplet, triplet loss updates the GRU's parameters so that, on the next pass, the anchor is closer to the positive than the negative. However, in the case where our anchor is a clip and our positive and negative examples are captions, it's likely that the original captions were similar, leading to similar word embeddings and thus a similar embedding for the entire caption in the network. In practice, we would actually like the anchor to remain close to the negative example, since the anchor is an appropriate video output for the negative caption. Triplet loss out of the box, thus, seems less than ideal.

Interestingly, increasing the margin monotonically increased performance on our validation set for each of the percentile metrics. It's possible that this was, in part, due to the problems discussed with triplet loss in the preceding paragraph. If we simply train on triplets where the original captions are similar, the model doesn't learn how to adequately distinguish dissimilar captions. Thus, the so-called "bad" triplets, may be best suited for this task. It's also possible that we simply failed to hit the point at which the difference between the gains obtained by training on more data and the cost of training of bad triplets becomes negative.

### C. Overfitting

Our dataset only contains videos of activity. If our goal is to make a search engine for activity clips, our model hasn't overfit - we showed very good performance when testing with activtiy related searches on our validation set. However, we don't know if our model generalizes to non-activity searches and non-activity videos. Inputting non-activity searches into our system doesn't give good output, but this may be just because there are no non-activity videos in the dataset. Establishing whether our model has overfit to activity videos would thus require a different, more diverse dataset.

## VIII. Conclusion/Future Work

We successfully implemented a well-performing video clip search platform which, given a short caption, returns some subset of one of a large number of videos, or "clip", corresponding to that caption. To do so, we trained two recurrent architectures in the siamese configuration, where each model had its own distinct parameters, and used these models to map clips and captions into a learned input space, where they were compared using the Euclidean distance. Our results suggest that the GRU is the best recurrent architecture for this task, and such a system may both be successfully trained and potentially put into practice.

Perhaps the biggest area for improvement on our work would be to train on a more comprehensive dataset, as opposed to just ActivityNet. We predict that this would improve performance on non-activity captions. Before putting our algorithm in practice, it would also be useful to have more clips to select from. Generating an automated clip extraction algorithm—by surfing youtube for videos, randomly taking subsets as clips, and doing the relevant preprocessing, is a good next step. While this could increase the time it takes our system to make a prediction, our system would have more videos to make the prediction from.

One last potentially interesting application is extracting the "most representative clip", or clip that most corresponds to the title of a video from the video. To do so, given a video, one would generate a number of clips, pass these clips through the model, and compare each of the clips to the caption passed through the caption-GRU after proper embedding. This has a number of useful applications. First, using this system one could save time by only looking at the relevant parts of a video. For example, if someone were to watch a video titled "Kid gets hit on head with basketball", they wouldn't have to watch the inevitable minute and a half of noise on either side. Second, this could potentially be used in robotics; a robot could learn specific tasks from youtube videos by first extracting the relevant clip, thus avoiding learning the incorrect thing due to the other content in the video.

## IX. Contributions

### A. Acknowledgements

We received the high level project idea from Ranjay Krishna of the Stanford Vision Lab, whose project created the text part of our dataset. We've also received help from Olivier Moindrot, our project TA, about questions regarding training and triplet loss.

### B. Erik

Erik was the only person with Pytorch experience prior to this project, so he was very involved in identifying how best to implement our desired model in pytorch and actually implementing it. In particular, Erik made many modifications to the starter code to convert the initial rnn code into a script with two optimizers, and helped write the original validation script. Erik also wrote the demo, and used AWS to train models and optimize hyperparameters. Erik, along with Milind, also wrote a custom Pytorch Dataset class for our database.

### C. Milind

Milind wrote scripts to link the caption data and visual feature data and store data in a format we could easily train and test on. He worked with Erik to write a custom Pytorch Dataset class to work with our somewhat unique data format. He spent significant time debugging and rewriting the triplet mining code and wrote the final, vectorized implementation that was used for all our reported results. Milind also ran a portion of the hyperparameter tests. He designed the validation metrics, and analyzed the outputs of both the hyperparameter sweeps and the final trained models. He also raised the discussion questions about frame rate and bad positive-negative pairs being passed through triplet loss.

### D. Zhaoyu (Joe)

Joe wrote the script for preprocessing the captions into sequences of word vectors, and researched avaliable models to determine which best suited our task. He wrote the initial version of the triplet mining procedure and helped debug the final version, as well as cleaning up the code base partway through the project. Like Milind and Erik, he helped in the final debugging and optimization of the model itself. Finally, he found or created many of the graphics used for the poster and this report.

## X. References

## References

[1] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *CoRR*, vol. abs/1607.01759, 2016. [Online]. Available: http://arxiv.org/abs/1607.01759

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[3] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *BMVC*, vol. 1, no. 3, 2015, p. 6.

[4] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[5] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE, 2016, pp. 4004–4012.

[6] B. Amos, "Openface 0.2.0: Higher accuracy and halved execution time," https://bamos.github.io/2016/01/19/openface-0.2.0/, 2016.

[7] R. Krishna, K. Hata, F. Ren, L. Fei-Fei, and J. C. Niebles, "Dense-captioning events in videos," in *International Conference on Computer Vision (ICCV)*, 2017.

[8] B. G. Fabian Caba Heilbron, Victor Escorcia and J. C. Niebles, "Activitynet: A large-scale video benchmark for human activity understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 961–970.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[10] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[12] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701