A Hierarchical Quasi-Recurrent approach to Video Captioning

Federico Bolelli, Lorenzo Baraldi, Federico Pollastri, Costantino Grana
Università degli Studi di Modena e Reggio Emilia
Email: {name.surname}@unimore.it

***Abstract*—Video captioning has picked up a considerable attention thanks to the ability of Recurrent Neural Networks to extrapolate an encoded representation of the input video, and then use it to generate a description. We propose a recurrent encoding approach able to find and exploit the layered design of the video. Differently from the established encoder-decoder procedure, in which a video is repeatedly encoded by a recurrent layer, we employ revised Quasi-Recurrent Neural Networks. We further extend their basic cell with a boundary detector in order to recognize discontinuous segments boundaries and likewise correct the temporal connections of the encoding layer accordingly. Experiments, on the Montreal Video Annotation dataset, demonstrate that our approach can find suitable levels of representation of the input information, while reducing the computational requirements.**

***Index Terms*—Video captioning, Recurrent Neural Networks, Content description**

I. Introduction

Video captioning aims to provide an automatic description of a video in natural language, also understanding the whole visual content and respecting the coherence between different scenes. This is a crucial, challenging task, towards the intelligence of machines and can become a fundamental element of many applications. In fact, by combining vision and language, video descriptions can be exploited for video recovery, to improve content-based search on video streaming services, and to provide automatic subtitles.

The first approaches to video captioning have essentially extended the existing image captioning techniques, achieving good results on brief videos with a single simple subject [1]–[3]. Video captioning is now moving from highly constrained or unedited, user-created videos [4], [5], towards more complicated and elaborated video types, due to the escalation of well-developed, ad-hoc datasets [6], [7].

To this date, Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM) [8] have been the most successful technologies for video captioning, thanks to their ability to manage both sequences of frames and long-range temporal patterns.

However, the performance of LSTMs drops when scenes are longer than 60-80 frames [9] and, on the other hand, the RNN is not compatible with the layered structure of videos. For example, a video showing an event, such as a basketball match, may be composed of different actions (running, passing the ball, shooting the ball); in this case, an efficient encoder should examine both the temporal dependencies inside each action

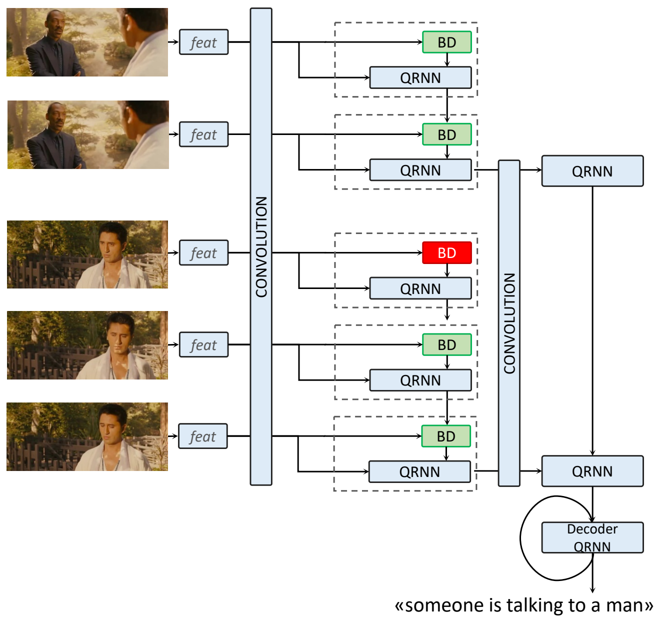


Fig. 1. We propose a novel video encoding network which can adaptively modify its structure to improve video captioning. Our Time Boundary-aware QRNN cell (depicted with dashed rectangles) extends the standard QRNN unit by adding a trainable boundary detector (BD), which can alter the temporal connections of the network depending on the input video.

and the temporal dependencies between different actions. This becomes even more evident in edited videos, where events are happening consecutively in different shots.

Another significant limitation of LSTM-based solutions is that they require a significantly long training time, as the computation of features and of the hidden states at different timesteps cannot occur in parallel. In [10] a new model for neural sequence modeling, called Quasi-Recurrent Neural Networks (QRNNs), is proposed. QRNNs address both drawbacks of standard models: like CNNs, QRNNs allow for parallel computation across both time and minibatch dimensions, enabling high throughput and good scaling to long sequences. Like RNNs, QRNNs allow the output to depend on the overall order of elements in the sequence. These models outperform strong LSTM baselines on many tasks while dramatically reducing computation time.

In this paper, we introduce a novel video encoding architecture for video captioning, which combines the effective QRNN strategy in a hierarchical structure, capable of identifying tem-

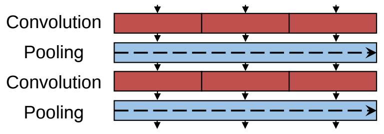


Fig. 2. Block diagram showing the computation structure of the QRNN. Red represents convolutions which can proceed in parallel, blue represents the pooling layer of the QRNN model.

poral boundaries and producing a better video representation.

The hierarchical structure of our sequence-to-sequence model, shown in Fig. 1, handles frames features, extracted by a CNN, which are fed to our time boundary-aware QRNN. The connectivity over time of the QRNN layer is changed when an appearance or an action discontinuity is detected. This leads to a variable length encoding with both granularity and length that depend on the input video. Moreover, a recurrent layer encodes the output of the first boundary-aware layer in a fixed-length vector, which is fed to another QRNN layer in order to generate the final caption.

To sum up, the contributions of the paper are twofold:

- We introduce a boundary-aware QRNN cell which enables the encoding layer to modify its temporal connectivity scheme, while preserving end-to-end differentiability.
- The time boundary-aware QRNN is used to build a hierarchical encoder for video captioning. While not requiring a previous temporal segmentation step, this enables us to implicitly leverage the video structure imposed by the editor, if present.

We assess our proposal on M-VAD [6], a public, large-scale movie description and video captioning dataset. The results are comparable with the state-of-the-art on movie description, with a fraction of the required training time.

II. Related Works

Sentence templates were an important tool employed in early captioning methods [11]: visual classifiers would be used to identify subject, verb and object from a scene, a language model would then generate captions fitting the predicted triplets to said templates. This architecture definitely harms the richness of the used language and lessens the framework's ability to generalize and analyze new data. Recurrent networks have been the go-to model to cope with these limitations, thanks to their natural ability to deal with series of words, given a vectored description of a visual content [12].

Venugopalan *et al.* [13] employed CNNs to extract features from individual frames, portrayed the entire video as their mean pooling, and made use of a LSTM layer [8] to generate the caption. This method, of course, did not manage to recognize the sequential nature of videos, not really filling the gap between simple image captioning and the task of video captioning. This is why, in following research, a lot of effort was put into the realization of more fitting video encoding techniques. Donahue *et al.* [14], employed both a LSTM network and CRFs to respectively encode the input video (without ignoring the sequential nature of videos) and extract semantic data regarding location, tool, activity, and object. The architecture output was a sentence obtained from the semantic tuple, exploiting one last LSTM layer.

In 2015, Venugopalan *et al.* [15] developed an end-to-end framework, extending the *sequence to sequence* approach, previously applied to machine translation [2], to video captioning. Two distinct LSTMs are used to deal with both video encoding and sentence decoding through neural networks, and the second LSTM, in charge of generating the final caption, is conditioned on the last hidden state of the first one, which encodes the input sequence. The authors were able to optimize the architecture by exploiting two LSTMs with shared parameters. Further works tried to embody attentive mechanisms [1] in the sentence decoder, exploiting independent language models [16], visual classifiers [17], or shared visual-semantic embeddings [18].

In the most recent years, many research efforts have been focused on improving this video captioning framework [19], following multiple approaches. Some of them explored more sophisticated sentence decoders. Yu *et al.* [3] proposed, for example, a hierarchical model containing both a sentence and a paragraph generator. On the other hand, Pan *et al.* [20] designed a hierarchical recurrent video encoder, enhancing a different part of the architecture. In their work, features are extracted from the video and then analyzed at various time scales and granularities, realizing a model where time can be seen as a dimension on which convolutional operations are applied. Finally, small overlapped video patches are fed to a LSTM, creating a series of feature vectors, which then become the input of a second recurrent layer. In [21], another hierarchical video encoder model has been proposed: in this case, a first layer estimates a temporal segmentation of the video, which is used to build a variable-length representation of the video.

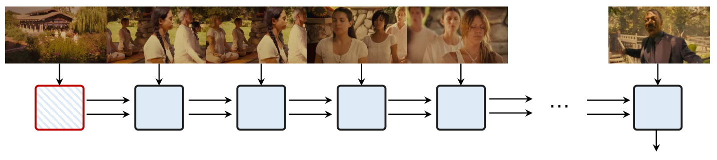
Also in this paper, we focus on the video encoding stage. Inspired by [21], instead of building an hand-crafted variation of the plain LSTM layer as in [20], we propose a quasi-recurrent network which can learn to adapt its temporal structure to input data. Our strategy ensures that the cell memory encoding each chunk always contains homogeneous information, and significantly reduces the computational requirements of the network.

III. Method

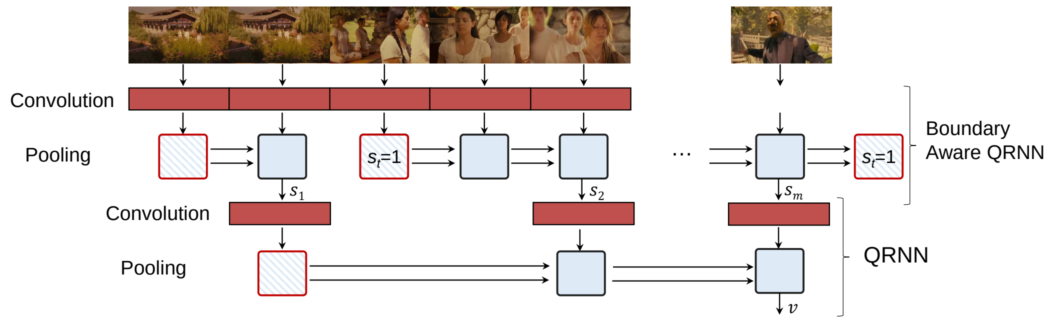
The basic idea behind QRNN is that of avoiding the costly matrix products that recurrent architectures perform between the input and the internal states, and replace them with convolutions over the input sequence. Formally, given an input sequence $\mathbf{X} \in \mathbb{R}^{T \times n}$, where T is its length, and n is the dimensionality of the input feature vectors, the operations performed by recurrent architectures can be usually expressed as a variant of the following equation:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta) \tag{1}$$

where f is, generally, a non-linear function, and θ represents a set of learnable weights. The value of the hidden states,



(a) Traditional LSTM network



(b) Time Boundary-aware QRNN network

Fig. 3. Comparison between a standard LSTM encoder and the Time Boundary-aware QRNN network. The proposed video encoder can *learn* to modify its temporal connections according to boundaries which are found inside the video: when a boundary is detected, the state of the QRNN is reinitialized and a representation of the ended segment is given to the output. Boxes surrounded by red represent LSTM/QRNN units with reset state, boxes surrounded by black stand for LSTM/QRNN unit with modified states.

as it can be observed, needs to be iteratively computed step by step as the value of $\mathbf{h}_t$ always depends on the application of $f(\cdot)$ over the previous state $\mathbf{h}_{t-1}$, preventing any possible parallelism on the time axis.

As an example, one variant of LSTM, known as dynamic average pooling [22], relies on the following formulation:

$$\begin{aligned}\mathbf{z}_t &= \tanh(\mathbf{W}_{xz}\mathbf{x}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z) \quad (2)\\ \mathbf{f}_t &= \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)\\ \mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)\end{aligned}$$

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t. \quad (3)$$

More complex versions have also been proposed [23]–[25].

QRNNs, rather, perform masked convolutions over the input sequence to obtain candidate feature vectors, instead of using matrix products. Since convolutions can be applied beforehand, and their computation can proceed in parallel along the time dimension, QRNNs can reduce the computational requirements of recurrent architectures. When convolutions are masked [26], moreover, the filters do not allow the computation for any given timestep to access information from future timesteps. This is usually implemented by padding the input to the left by the convolution's filter size minus one.

In such an architecture, the sequence of values for gates $\mathbf{z}_t, \mathbf{f}_t$ and $\mathbf{o}_t$ are computed as follows,

$$\begin{aligned}\mathbf{Z} &= \tanh(\mathbf{W}_z * \mathbf{X}) \quad (4)\\ \mathbf{F} &= \sigma(\mathbf{W}_f * \mathbf{X})\\ \mathbf{O} &= \sigma(\mathbf{W}_o * \mathbf{X})\end{aligned}$$

where $*$ represents the masked convolution operator. Then, the value of $\mathbf{h}_t$ is obtained by applying Eq. 3 as before. Given an input video, our video encoder takes as input the sequence of visual features $\mathbf{X}$ and processes it in a hierarchical fashion. At the first level, it outputs a sequence of vectors $(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_m)$ as the representation for the whole video, using a connectivity schema that varies with respect to both the current input and the hidden state. A second layer, instead, computes a fixed-length feature vectors by taking into account the variable length output of the first layer. Both layers are implemented using QRNNs.

To represent the input video in the first layer, we define a quasi-recurrent boundary-aware layer. In this layer, the connectivity through time can be modified during the processing of the input video: when a new boundary is detected, the hidden state of the layer is reinitialized, and at the end of a detected segment the same hidden state is given to the output, as a summary of the segment. As such, the boundary-aware layer creates a hierarchical representation of the video with variable length, in which each segment is composed by homogeneous frames. In Fig. 3a and 3b we show the connections through time determined in the layer when processing a sample case, compared to those of a plain LSTM encoder.

At each timestep, the layer internally selects whether to transfer its hidden state to the next timestep, as in a plain recurrent layer, or to reinitialize it, interrupting the update and processing of the input video. This choice depends on a boundary detection module, which allows the layer to process each detected chunk independently. The boundaries of each segment are detected by a learnable function which depends on the input.

The boundary detection unit $s_t \in \{0, 1\}$ is defined as a linear combination of the input and of the hidden state, followed by an activation function τ, given by the composition

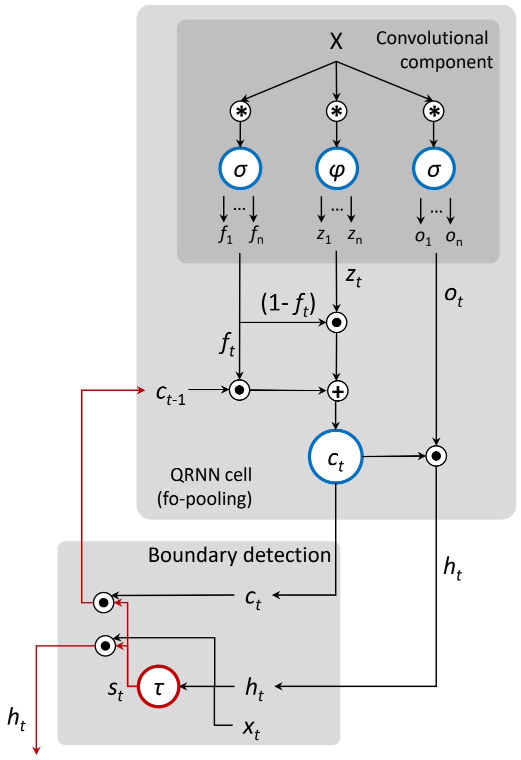


Fig. 4. Schema of the Boundary-aware QRNN cell.

of a sigmoid and a step function:

$$s_t = \tau(\mathbf{v}_s^T \cdot (W_{si}\mathbf{x}_t + W_{sh}\mathbf{h}_{t-1} + \mathbf{b}_s)) \tag{5}$$

$$\tau(x) = \begin{cases} 1, & \text{if } \sigma(x) > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{v}_s^T$, W_{si}, W_{sh} and $\mathbf{b}_s$ are all learnable weights and biases.

Once the current boundary unit s_t has been computed, before applying the update equation of the recurrent layer (Eq. 3), the following substitution is performed, to reinitialize the hidden state in case s_t has been activated:

$$\mathbf{h}_{t-1} \leftarrow \mathbf{h}_{t-1} \cdot (1 - s_t) \tag{7}$$

The resulting state is then used to compute the next hidden state value. This layer produces an output only at the end of a segment: if $s_t = 1$ the hidden state of the previous timestep $t-1$ is passed to the next layer, which is implemented, again using a QRNN. Fig. 4 shows a schema of the internals of the time boundary-aware cell, for the ease of the reader.

Beyond using the dynamic average pooling presented in Eq. 3, we also experiment with other LSTM-like variants, on which the QRNN architecture has been proved to work well. In particular, we also include an output gate in this formulation:

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t \\ \mathbf{h}_t &= \mathbf{o}_t \odot \mathbf{c}_t; \end{aligned} \tag{8}$$

and an independent input and forget gate in this formulation:

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t \\ \mathbf{h}_t &= \mathbf{o}_t \odot \mathbf{c}_t. \end{aligned} \tag{9}$$

where the time series of $\mathbf{I}$ is obtained similarly to the other gates by performing a masked convolution over $\mathbf{X}$.

The boundary-aware layer will produce a variable length sequence of outputs $(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_m)$, where m amounts to the number of detected segments. Each $\mathbf{s}_i$ encodes the content of a detected chunk, and the overall sequence of outputs is then passed to a second recurrent layer, thus building a hierarchical encoder. To this end, the output of the first layer is fed to a second QRNN layer without boundary unit: the last hidden state of this layer is then used as the representation of the entire video.

A. Training

Since computing the boundary detector involves adding binary variables in the computational graph, we need to take special training expedients.

In particular, the boundary detector s_t is treated as a stochastic neuron [27] during the forward pass of the training. To do so, we exploit a stochastic version of function $\tau(x)$ (Eq. 6), by sampling from a uniform probability distribution conditioned on $\sigma(x)$. Formally, the stochastic version of τ is computed as

$$\tau(x) = \mathbf{1}_{\sigma(x)>z}, \text{ with } z \sim U\,[0,1]\,, \text{forward pass} \tag{10}$$

where $U\,[0,1]$ is the uniform probability distribution over $[0,1]$ and $\mathbf{1}$ is the indicator function. As it can be noticed, the version of s_t defined above is stochastic, and with a probability of being 0 or 1 which is proportional to the value of a sigmoid applied to the input of τ.

During the backward pass, instead, we employ a differentiable estimator [28]. Therefore, while we use a discrete operation in the forward pass, in the backward pass we use a differentiable approximation: specifically, the step function is approximated with the identity function, as suggested in [28]. Being τ the composition of a sigmoid and a step function, the derivative of τ used in backward is simply the derivative of the sigmoid function.

$$\frac{\partial \tau}{\partial x}(x) = \sigma(x)(1 - \sigma(x)), \text{backward pass} \tag{11}$$

At test time, the deterministic version of the step function (Eq. 6) is used. In this way the number of detected segments is stochastic during training and deterministic during test.

B. Sentence generation

To generate the output caption, given the feature vector of the video extracted by the Boundary-aware network, we employ a QRNN decoder network, following the encoder-decoder scheme [1].

Given the video vector $\mathbf{v}$ produced by the video encoder, and a sentence $(\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_T)$, encoded with one-hot vectors, we condition the decoder on the first t words of the caption and on the corresponding video descriptor, and train it to produce the next word of the caption. The objective function which is optimized is the log-likelihood of correct words over the sequence

$$\max_{\mathbf{w}} \sum_{t=1}^{T} \log \Pr(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, ..., \mathbf{y}_0, \mathbf{v}) \qquad (12)$$

where $\mathbf{w}$ are the parameters of the overall model. The probability of a word is modeled via a softmax layer applied on the output of the decoder. To reduce the dimensionality of the decoder, a linear embedding transformation is used to project one-hot word vectors into the input space of the decoder and, viceversa, to project the output of the decoder to the dictionary space:

$$\Pr(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, ..., \mathbf{y}_0, \mathbf{v}) \propto \exp(\mathbf{y}_t^T W_p \mathbf{h}_t) \qquad (13)$$

where W_p is a matrix for transforming the decoder output space to the word space and $\mathbf{h}_t$ is the output of the QRNN decoder.

IV. Experimental evaluation

Evaluation is carried out on a large-scale dataset for video captioning specifically built for movie description: the Montreal Video Annotation dataset (M-VAD) [6]. M-VAD is based on Descriptive Video Service (DVS), which are audio tracks describing the visual elements of a movie, produced to help visually impaired people. The dataset is composed of 46,523 video clips covering a wide variety of different scenes. We follow the standard split of the dataset, which consists of a training set of 36,921 clips, a validation set of 4,651 clips and a test set of 4,951 clips.

Compared to other video captioning collections such as the Microsoft Video Description Corpus [5], this dataset is more challenging, due to the high semantic variety of captions, and due to the presence of a single sentence per video, which limits both the effectiveness of the training and that of evaluation.

We employ METEOR [29], a popular metrics for evaluation. This measure evaluates captions by comparing them with ground truths. Matches between words and phrases can be exact, stem, synonym, and paraphrase. METEOR is very semantically suitable, more than metrics such as BLEU and ROUGE$_L$.

We ensure a fair evaluation computing all results with the Microsoft CoCo evaluation toolkit[1], as done by others previous video captioning researches [3], [20].

Static appearance features are extracted from input videos of all datasets: ResNet50 model [30] pretrained on the Imagenet dataset [31] is employed as video encoder and a descriptor is estimated every five frames. We use the activations from the second-last layer of the network, which provides a 2,048 dimensional feature vector. Moreover, a linear embedding is learned as the input of the model, substituting the plain visual features.

[1]https://github.com/tylin/coco-caption

The ground truth data are stripped from special characters, converted to lower case, and finally tokenized. We remove every word with less than 5 appearances in the datasets, producing a vocabulary counting 6,090 words. During training, we insert two types of tag: begin-of-sentence <BOS> (at the beginning of the caption), and end-of-sentence <EOS> (at its end). This way, our model can deal with variable caption lengths. During test, a <BOS> tag serves as the first input at the first timestep for the decoder, then the framework outputs the best word according to the estimated distribution: this word will be the input of the model for the next timestep. The whole caption is produced this way, until the network predicts a <EOS>.

The RMSprop optimizer is eployed to minimize the log-likelihood loss, with an initial learning rate of 0.001. We also use a learning rate scheduling policy in which we lower the learning rate by a factor of 10 after 3 epochs in which we do not observe an improvement over the validation set. The mini-batch size is set to 100.

The size of all recurrent hidden states is empirically set to 512, as are the embeddings for video features and words. The number of rows in matrices W_{si}, W_{sh} and in vector $\mathbf{b}_s$ (Eq. 6) is set to 128. Regarding initialization, the Gaussian initialization suggested by Glorot *et al.* [32] is used for both weight matrices applied to inputs and embedding matrices. All biases were initialized to zero. Orthogonal initialization is put into use for weight matrices applied to internal states. We train the model as long as we notice a loss improvement over the validation set.

We compare our method with 4 different proposals: Temporal attention (SA) [1], S2VT [15], the approach from Venugopalan *et al.* [16], and HRNE [20]. SA employed GoogleNet [33] and a 3D spatio-temporal CNN as feature extractors, it then exploited this features through a LSTM decoder with a temporal attention mechanism. S2VT, instead, extracted frame-level features using the VGG model and used stacked LSTMs both for the encoder and the decoder stage. The approach from Venugopalan *et al.* [16] extends the S2VT architecture by adding knowledge from text corpora to the language model. Finally, HRNE runs a LSTM on short video segments exploiting a sliding window algorithm, the decoder selectively attends to the resulting set of vectors, optionally through a soft attention mechanism.

Results are shown in Table I. Firstly, to investigate the potential of substituting LSTMs with QRNNs, we compare its performance against that of a single QRNN layer, trained using the same features as well as the same hyperparameters. In this case, a single QRNN layer encodes the video sequence, and the last hidden state serves as the video vector for the decoder. This baseline achieves a 4.5%, while using the QRNN significantly boosts performance, yielding a 5.0%. Introducing the boundary-aware encoder with LSTMs further improves the performance to 5.6%, but the QRNN based version pushes the performance to 6.5%. The use of image features only is limiting the capabilities of our tests, which are not able to reach the best performing proposals, but the improvement of

TABLE I
EXPERIMENTAL RESULTS ON THE M-VAD DATASET. K REPRESENTS THE KERNELS SIZES OF QRNNS LAYERS.

Model	METEOR
SA-GoogleNet+3D-CNN [1]	4.1
S2VT-RGB(VGG) [15]	5.6
HRNE [20]	5.8
HRNE with attention [20]	6.8
Venugopalan *et al.* [16]	6.8
One layer LSTM encoder, LSTM decoder	4.5
One layer QRNN encoder, QRNN decoder - k=3,7	5.0
Boundary-aware LSTM encoder, LSTM decoder	5.6
Boundary-aware QRNN encoder, QRNN decoder - k=7,7,11	6.5

the introduction of both QRNNs and BA version of it is still significant and could be further exploited.

The training time required by a naive implementation in PyTorch was 135s for a LSTM epoch, while it was 189s for a QRNN one. However, QRNN converged faster (*e.g.* 57 epochs versus 137 epochs for the BA version of the model), saving a considerable amount of training time, without loss of accuracy.

V. CONCLUSION

With this work, we proposed a novel boundary-aware video encoder for the task of video captioning, which obtains competitive results on a popular benchmark. Our method can find the hierarchical structure of the video, and accordingly modify the temporal connections of a recurrent layer. The use of QRNNs allows to significantly reduce the computational time while retaining competitive performance. Further experiments are needed, since the introduction of specific video features has shown significant benefits in preliminary tests and should be incorporated in the system.

REFERENCES

[1] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville, “Describing videos by exploiting temporal structure,” in *ICCV*, 2015, pp. 4507–4515.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[3] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu, “Video paragraph captioning using hierarchical recurrent neural networks,” *CVPR*, 2016.

[4] A. Rohrbach, M. Rohrbach, W. Qiu, A. Friedrich, M. Pinkal, and B. Schiele, “Coherent multi-sentence video description with variable level of detail,” in *German Conference on Pattern Recognition*. Springer, 2014, pp. 184–195.

[5] D. L. Chen and W. B. Dolan, “Collecting highly parallel data for paraphrase evaluation,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-2011)*, 2011.

[6] A. Torabi, C. Pal, H. Larochelle, and A. Courville, “Using descriptive video services to create a large data source for video annotation research,” *arXiv preprint arXiv:1503.01070*, 2015.

[7] A. Rohrbach, M. Rohrbach, N. Tandon, and B. Schiele, “A dataset for movie description,” in *CVPR*, 2015.

[8] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *ICASSP*. IEEE, 2013, pp. 6645–6649.

[9] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, “Beyond short snippets: Deep networks for video classification,” in *CVPR*, 2015, pp. 4694–4702.

[10] J. Bradbury, S. Merity, C. Xiong, and R. Socher, “Quasi-recurrent neural networks,” in *ICLR*. Toulon, France: OpenReview.net, 2017.

[11] S. Guadarrama, N. Krishnamoorthy, G. Malkarnenkar, S. Venugopalan, R. Mooney, T. Darrell, and K. Saenko, “Youtube2text: Recognizing and describing arbitrary activities using semantic hierarchies and zero-shot recognition,” in *ICCV*, 2013, pp. 2712–2719.

[12] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *CVPR*, 2015, pp. 3156–3164.

[13] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, “Translating videos to natural language using deep recurrent neural networks,” *North American Chapter of the Association for Computational Linguistics*, 2014.

[14] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” in *CVPR*, 2015, pp. 2625–2634.

[15] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, “Sequence to sequence-video to text,” in *ICCV*, 2015, pp. 4534–4542.

[16] S. Venugopalan, L. A. Hendricks, R. Mooney, and K. Saenko, “Improving lstm-based video description with linguistic knowledge mined from text,” in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[17] A. Rohrbach, M. Rohrbach, and B. Schiele, “The long-short story of movie description,” in *German Conference on Pattern Recognition*. Springer, 2015, pp. 209–221.

[18] Y. Pan, T. Mei, T. Yao, H. Li, and Y. Rui, “Jointly modeling embedding and translation to bridge video and language,” *CVPR*, 2016.

[19] S. Pini, M. Cornia, F. Bolelli, L. Baraldi, and R. Cucchiara, “M-vad names: a dataset for video captioning with naming,” *Multimedia Tools and Applications*, 2018.

[20] P. Pan, Z. Xu, Y. Yang, F. Wu, and Y. Zhuang, “Hierarchical recurrent neural encoder for video representation with application to captioning,” *CVPR*, 2016.

[21] L. Baraldi, C. Grana, and R. Cucchiara, “Hierarchical boundary-aware neural encoder for video captioning,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] D. Balduzzi and M. Ghifary, “Strongly-typed recurrent neural networks,” *ICML*, 2016.

[23] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with lstm,” *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[25] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, “Training recurrent networks by evolino,” *Neural computation*, vol. 19, no. 3, pp. 757–779, 2007.

[26] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.

[27] T. Raiko, M. Berglund, G. Alain, and L. Dinh, “Techniques for learning binary stochastic feedforward neural networks,” in *ICLR*, 2015.

[28] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.

[29] S. Banerjee and A. Lavie, “Meteor: An automatic metric for mt evaluation with improved correlation with human judgments,” in *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, vol. 29, 2005, pp. 65–72.

[30] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[32] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, May 2010, pp. 249–256.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *CVPR*, 2015, pp. 1–9.